Dt - 12th Jan 18

Algorithm

It is a sequence of well-defined steps solved in a finite time.

Characteristics of Algorithm

→ It requires finite inputs.

→ Input to the algorithm is also called Instance of a problem.

→ Number of input is called Input size or Instance size.

Dt - 13th Jan 18

→ It should produce correct output for every possible input instance.

→ The algorithm should not have any ambiguity.

→ Each step must have number [Definiteness]

→ The algorithm should stop in finite number of steps. [Finiteness]

→ Program is an implementation of algorithm.

→ Programs are written in any programming language such as C, C++, Java.

Standard Notations used to write Algorithm →

* Assignment → $x \leftarrow 5$ or $x := 5$
* Each step should be provided with a number.
* All the variables should be expressed in capital letters.
* Comments: //, /* */

LectureNotes.in

IF (condition)

Statement 1;

ELSE

Statement 2;

[END IF 1]

LOOP

FOR I ← 1 to N Step 1

—

—

[END FOR 1]

One dimentional array

A[1.....N] - It's 1st index is 1 and last is N.

Two dimensional array

A[1.....N, 1.....M] - It's 2 dimensional array of order N x M.

Accessing an element of 2D array

A[i, j] - refers the element raw in i and j in column.

LectureNotes.in

Analysis of Algorithm

* It refers to the task of determining computing time and storage space required to execute an algorithm.
* Time complexicity of the algorithm is that computational time required to execute an algorithm.
* The computational time is directly propartional to number of steps.

Q. Algorithm : Find x^N

```
POWER (X, N)
/* To find X^N */
1.  P ← 1 ------ 1
2.  FOR I ← 1 to N ---- N+1
3.      P ← P * X ---- N
    [END FOR 2]
4.  RETURN P ---- 1
```

I	P
1	1
2	5
3	25
4	125
5	625

$2N+3$

* Time complexcity is a function of N. where n is input size or instant size. $f(n)$

Q. Algorithm : Find x^N

```
POWER (X, N)
/* To find X^N where n is power of 2 */
1.  FOR  I ← 1 to 2 // log2 N
2.       X ← X * X ---- log2 N
    [END FOR 1]
3.  RETURN X ---- 1
```

I	X
1	5
2	25
3	625

$2\log_2 N + 2$

$T_2(N) = 2\log_2 N + 2$

$T_2(N) < T_1(N)$

↓ Logarithmic function

↓ Linear function

Q. Find out the number of times the function fcn will execute.

→ 1. FOR I ← 1 to N

2. fcn

[END FOR 1]

$T(n) = \sum_{I=1}^{i=n} 1 = n$ (Linear)

→ 1. FOR I ← 1 to N

2. fcn

[END FOR 1]

3. FOR J ← 1 to M

4. fcn

[END FOR 2]

$\sum_{i=1}^{i=N} 1 + \sum_{j=1}^{i=M} 1$

$= N + M$

If $M = N$

$T(n) = 2N$

$T_1(n) = n$

$T_2(50) = 50n$

→ 1. FOR J ← 1 to N

2. FOR J ← 1 to N

3. fcn

[END FOR 2]

[END FOR 1]

LectureNotes.in

$$T(N) = \sum_{i=1}^{I=N} \sum_{j=1}^{J=N} 1$$

$$= \sum_{i=1}^{I=N} N$$

$$= N + N + N \text{ - - - - } N \text{ times}$$

$$= N^2 \text{ (Quadratic)}$$

→ 1. FOR I ← 1 to N

2. FOR J ← 1 to I

3. f(n)

[END FOR 2]

[END FOR 1]

$$T(N) = \sum_{i=1}^{I=N} \sum_{j=1}^{J=I} 1$$

$$= \sum_{i=1}^{I=N} I$$

$$= 1 + 2 + 3 + \text{- - - - } + N$$

$$= \frac{N(N+1)}{2}$$

$$= \frac{1}{2}(N^2 + N)$$

(Quadratic)

LectureNotes.in

Conclusion

While comparing the algorithms based on Growth Rate -

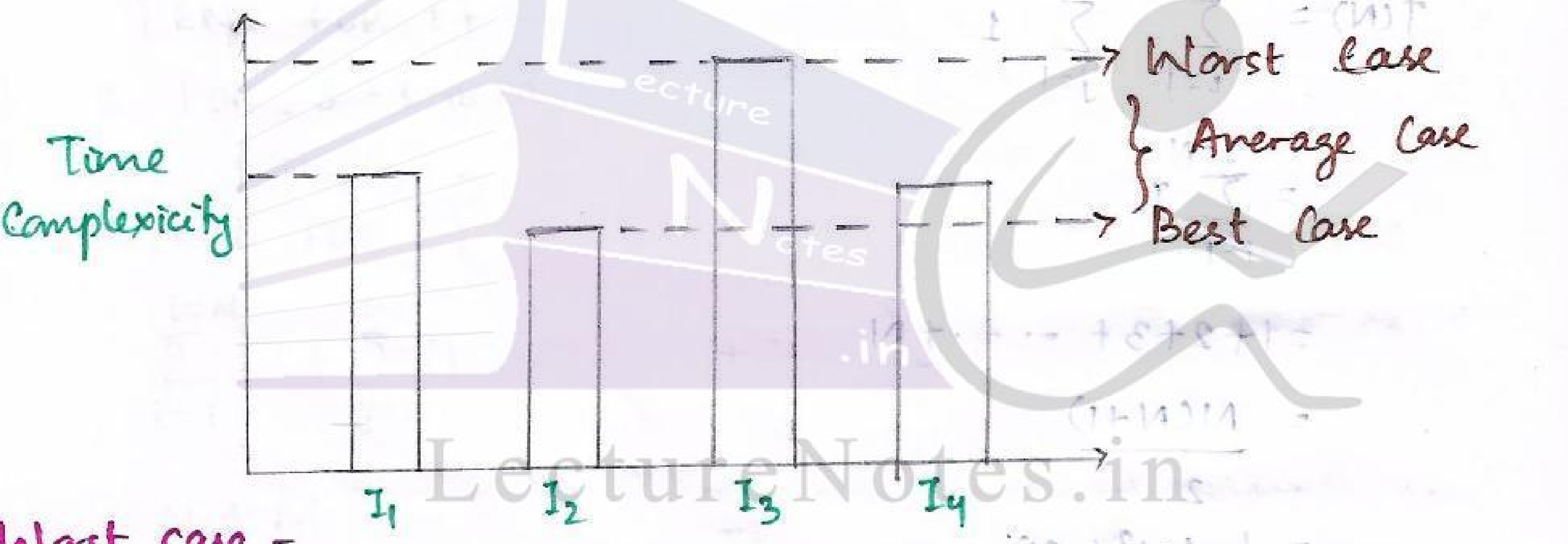
→ The co-efficients are ignored.

→ The lower ordered terms are ignored.

→ Higher order term is considered.

* Time Complexicity can be measured in the following 3 cases -

- Best
- Worst
- Average



Worst case -

* Worst case time complexicity of an algoritem is the function defined by maximum number of steps taken on input size n.

* The worst case running time of an algorithm is an upper baund running time for any input.

Best case time complexicity -

* Best case time complexicity of an algoritem is the function defined by minimum

* The best case running time of an algorithm is a lower running time for any input.

Average Case Time Complexicity -

* Average case time complexicity of an algorithm is the function defined by the number of steps between maximum and minimum number of steps taken on input size n.

Example - In Linear Search, the time complexicity in

Best case is $T(n) = \theta(1)$ (constant time)

Worst case is $T(n) = \theta(n)$

Average case is $T(n) = \frac{\theta(n+1)}{2}$

NOTE -

The constant time is faster than any function which is dependent to n.

Growth Rate

The Growth Rate of an algorithm is the rate at which the running time of an algorithm grows when input size grows.

Order of Growth

The leading term of the time $T(n)$ is the Order of Growth.

Ex - $T(n) = 5n^2 + 3n + 5$ (Quadratic)

Here, the higher order term is n^2.

$$f(n) = \sum_{k=0}^{k=n1} a_k n^k$$

$$= a_0 + a_1 n + a_2 n^2 + \cdots + a_{n1} n^{n1}$$

Order of growth may be described as

⇒ T(n) = O(1) [Constant time]

⇒ T(n) = f(log n) [Logarithmic]

⇒ T(n) = f(n) [Linear]

ex: f(n) = 3n ⇒ T(n) = f(n²) [Quadratic]

ex: f(n) = 3n² ⇒ T(n) = (a)ⁿ where a>0 [Exponential time]

⇒ T(n) = (n)ⁿ

Asymptotic Analysis

* It specifies the time complexicity in terms of order of growth.
* Following assumptions are considered during Asymptotic Analysis -
  - Input size is considered as large value [tends to infinite]
  - The time complexicity is a function of N where f(N) > 0.
  - Input size are the natural neembers.

Asymptotic Notations

* The notations used in Asymptotic Analysis is called Asymptotic Notations.
* It describes the time complexicity in terms of order of growth.
* Following 5 asymptotic notations are used is asymptotic analysis -
  - → Big-Oh Notation (O)
  - → Big-omega Notation (Ω)

→ Theta Notation (θ)

→ Little - Oh Notation (o)

→ Little - Omega Notation (ω)

Dt - 19th Jan 18
Friday

Big Oh (O)	Big Omega (Ω)	Theta θ
$f(n) \leq c * g(n)$	$f(n) \geq c * g(n)$	$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

→ f(n) and g(n) are two non-negative functions.

→ c, c_1, c_2 are the constants and is always greater than zero (0).

→ n is a constant which is greater than n_0.

$c, c_1, c_2 > 0$

$n > n_0$

n_0 → any interger value

f(n)
g(n) } non-ve function

Little Oh Notation (o)

$f(n) = o(g(n))$

$0 \leq f(n) < c * g(n)$

Little Oh is also called Tight Upper Baund.

Little Omega Notation (ω)

$f(n) > c * g(n) \geq 0$

It is also called Tight lower Baund

Big Oh (O)	$f(n) \leq c * g(n)$	Upper Bound
Big Omega (Ω)	$f(n) \geq c * g(n)$	Lower Bound
Little Oh (o)	$0 \leq f(n) < c * g(n)$	Tight Upper Bound
Little Omega (ω)	$f(n) > c * g(n) \geq 0$	Tight Lower Bound
Theta (Θ)	$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$	—

We Know

$$f(n) \leq c * g(n)$$

Ex: $f(n) = n^2 + n \quad n > n_0 \geq 0$

$$n^2 + n \leq c * g(n)$$

$$n^2 + n \leq c * n^2 \quad n \geq 0,$$

$1 + 1 \leq 2 * 1 \quad n = 1$

$1 + 1 \leq 2 * 1 \quad n = 2$

$2^2 + 2 \leq 2 * 4 \quad n = 3$

$6 \leq 8$

$3^2 + 3 \leq 2 * 9$

$12 \leq 18$

$c = 2, \ n_0 = 1$

for $c = 2, n_0 = 1$, such that $n \geq 0$

$$f(n) \leq c * g(n)$$

$$n^2 + n \leq 2 * n^2$$

$$f(n) = O(n^2)$$

LectureNotes.in

Ex - $f(n) \leq c * g(n)$

$f(n) = 6n^2 + 5$

Sol. $6n^2 + 5 \leq c * n^2 \quad , n=1$

$6 + 5 \leq c * 1 \qquad c = 11$

$11 \leq 11 * 1$

$c = 11, \; n = 1$

$\boxed{f(n) = O(n^2)}$

Ex - $f(n) = 7n^2 + 8n + 39$

Sol. $7n^2 + 8n + 39 \leq c * g(n)$

$7n^2 + 8n + 39 \leq c * n^2 \qquad n=1$

$7 + 8 + 39 \leq 54 * 1$

$54 \leq 54$

$n = 1, \quad c = 54$

$f(n) \leq c * g(n)$

$7n^2 + 8n + 39 \leq 54 * n^2$

$\boxed{f(n) = O(n^2)}$

Ex - $f(n) = n^3 + 35n^2 + 1$

$n^3 + 35n^2 + 1 \leq c * g(n)$

$n^3 + 35n^2 + 1 \leq c * n^3$

$1 + 35 + 1 \leq 37 * 1 \qquad n=1$

$37 \leq 37$

$c = 37, \quad n = 1$

$\boxed{f(n) = O(n^3)}$

LectureNotes.in

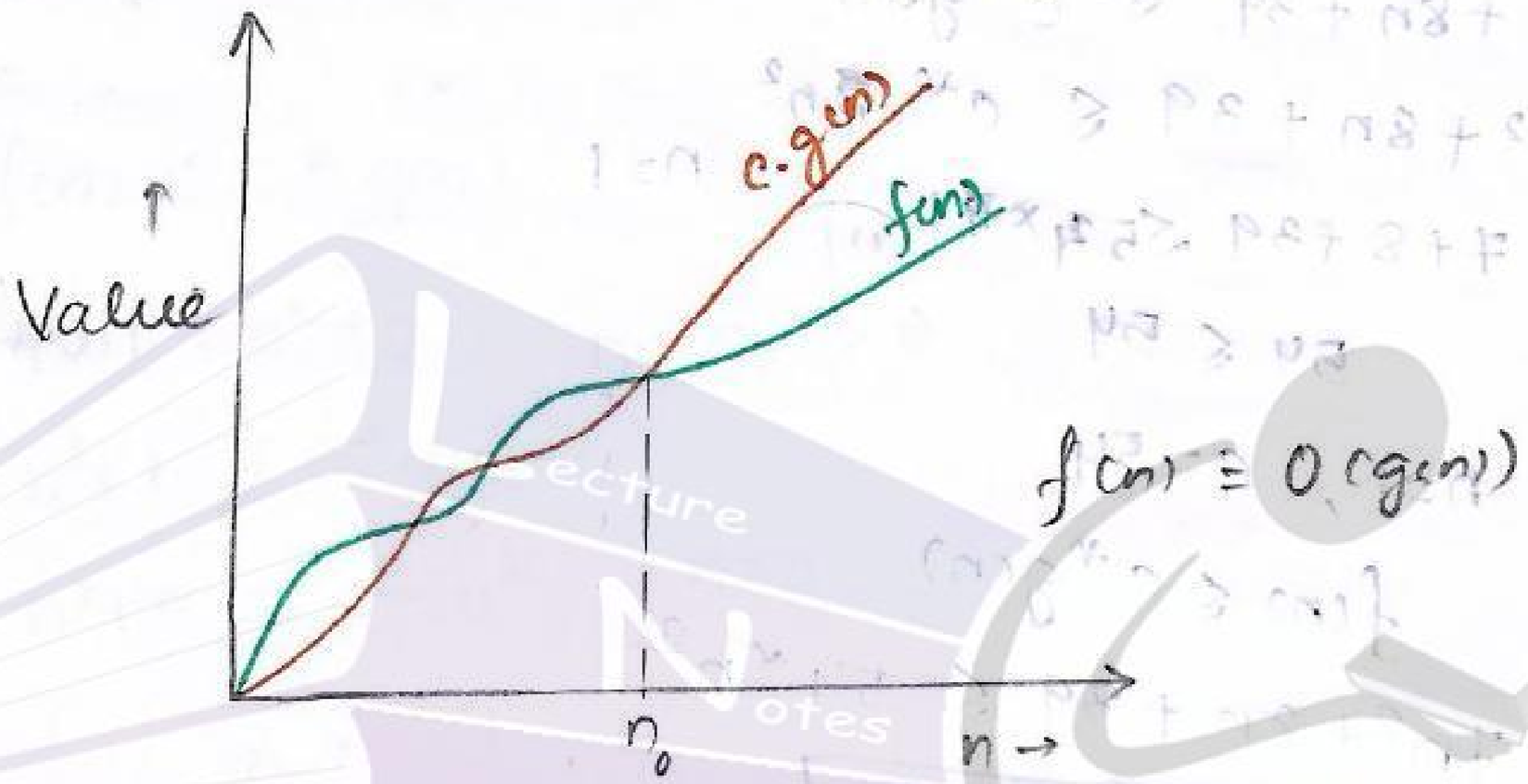
Dt → 20th Jan 18
Saturday

1. Big Oh Notation (O)

→ Let f(n) and g(n) are two non-negative functions.

→ The function, f(n) = O(g(n)) iff there exists a positive constant c and no such that

$$f(n) \leq c \cdot g(n), \quad \forall\ n \geq n_0 \text{ and } \exists\ c > 0$$



Ex →

n	f(n) = 2n+4	$3n$ c.g(n)	Remarks
1.	6	3	f(n) > c.g(n)
2	8	6	f(n) > c.g(n)
3	10	9	f(n) > c.g(n)
4	12	12	f(n) = c.g(n)
5	14	15	f(n) < c.g(n)

$f(n) \leq c \cdot g(n), \quad \forall\ n \geq n_0 = 4$ and $c = 3$

$\therefore$ **$f(n) = O(g(n))$**

Ex →

n	$f(n) = 2n+4$	$c \cdot g(n) = 1 \cdot n^2$	Remarks
1	6	1	$f(n) > c \cdot g(n)$
2	8	4	$f(n) > c \cdot g(n)$
3	10	9	$f(n) > c \cdot g(n)$
4	12	16	$f(n) < c \cdot g(n)$
5	14	15	$f(n) < c \cdot g(n)$

$f(n) \leq c \cdot g(n)$,

$\forall\ n \geq n_0 = 4$ and $c = 1$

$\therefore \boxed{f(n) = O(n^2)}$

→ If the degree of $g(n)$ is greater or equal to the degree of $f(n)$, then we can say that

$$f(n) = O(g(n))$$

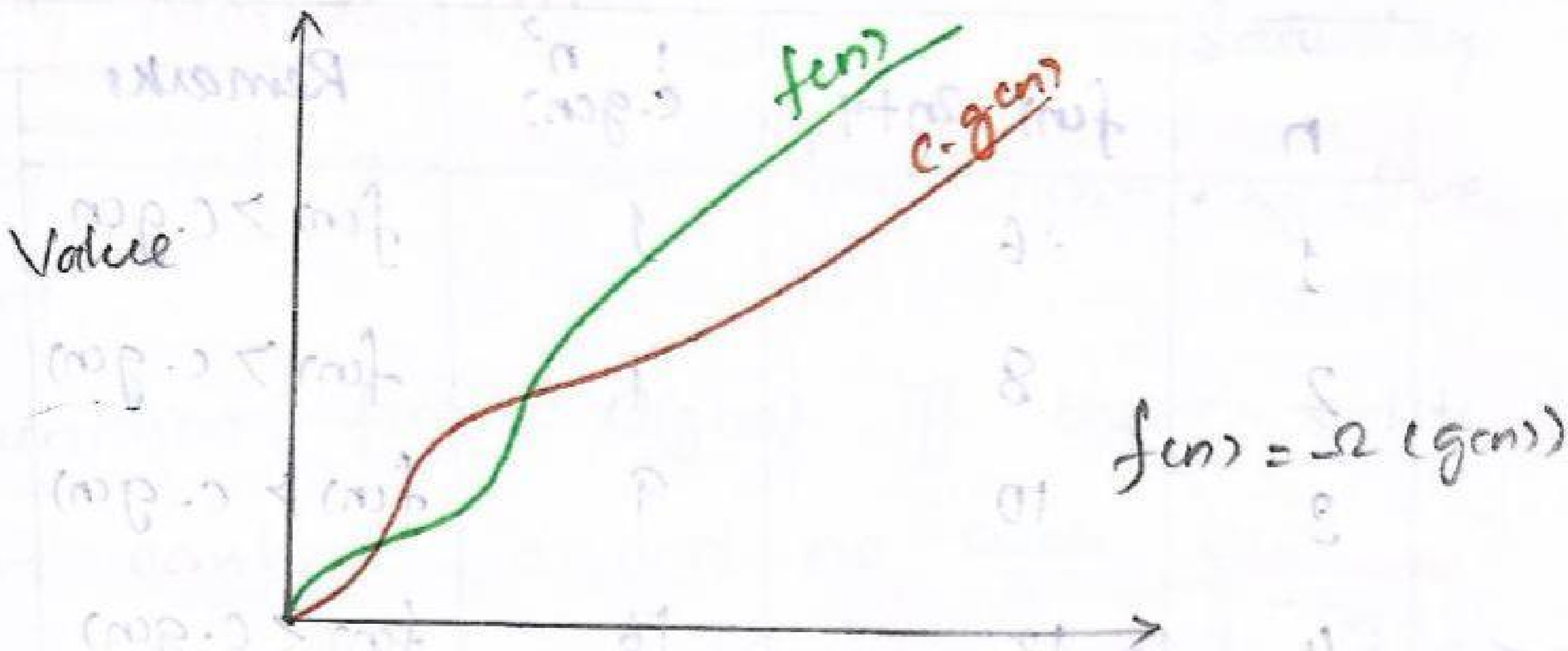
→ The Big-Oh Notation describes the time complexity in Upper Baund.

2. Big Omega Notation (Ω)

→ Let $f(n)$ and $g(n)$ are two non-negative functions.

→ The function, $f(n) = \Omega(g(n))$ iff there exists a positive constant c and n_0 such that:

$f(n) \geq c \cdot g(n)$, $\forall\ n \geq n_0$ and $\exists\ c > 0$



$f(n) = \Omega(g(n))$

The Big-Omega Notation is used to specify the time complexity in Lower bound.

Dt - 27th Jan 18
Saturday

Ex- $f(n) = 2n^2 + 2$

$f(n) = \Omega(n)$

n	$f(n) = 2n^2+2$	$c \cdot g(n) = 1 \cdot n^2$	Remarks
1	4	1	$f(n) \geq c \cdot g(n)$ $n \geq n_0, c > 0$

$f(n) = \Omega(n)$

$g(n) = n^2$

$f(n) = \Omega(n^2)$ ~

Ex - $f(n) = 5n^4 + 3$

$f(n) = 5n^4 + 3 \geq n^5$

∴ The more appropiate is

$f(n) = \Omega(n^2)$

LectureNotes.in

If the degree of g(n) ≤ degree of f(n), then f(n) can be expressed as

$$f(n) = \Omega(g(n))$$

3. Theta (θ) Notation

→ Let f(n) and g(n) are two non-negative functions.

→ then the function, $f(n) = \theta g(n)$ iff there exists two positive constants c1, c2 and n_0 such that

$$c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n), \quad \forall\ n \geq n_0, \quad \exists\ c_1 > 0, \exists\ c_2 > 0$$

Q. Let $f(n) = 2n^2 + 2$, then show that $f(n) = \theta(n^2)$

Sol. $f(n) \leq 4n^2, \quad \forall\ n \geq n_0 = 1, \quad c = 4$ — (1)

and $f(n) \geq 1 \cdot n^2, \quad \forall\ n \geq n_0 = 1, \quad c = 1$ — (2)

$$1 \cdot n^2 \leq f(n) \leq 4n^2$$

$\therefore f(n) = \theta(n^2), \quad \forall\ n \geq n_0 = 1, \quad c_1 = 4, c_2 = 1$

Q. Let $f(n) = 2n^2 + 3$, then show that $f(n) = \theta(n^2)$.

Sol. $f(n)$

$$1 \cdot n^2 \leq f(n) \leq 5n^2, \quad \forall\ n \geq n_0 = 1,$$

$c_1 = 5, \quad c_2 = 1$

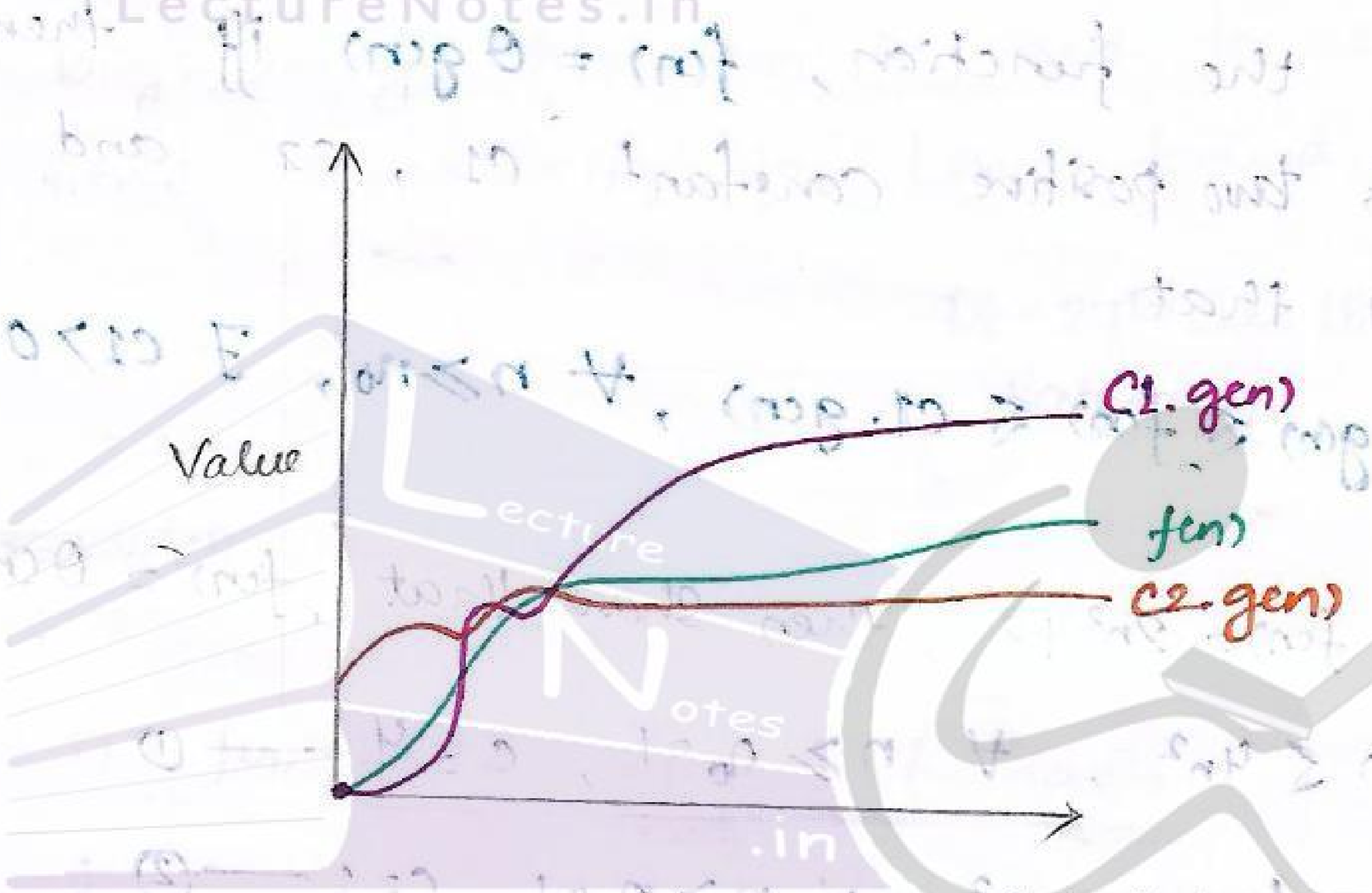
Q. Let $f(n) = 4n^3 + 2n + 3$, then show that $f(n) = \theta(n^3$

Sol. $1 \cdot n^3 \leq f(n) \leq 9n^3 \quad \forall\ n \geq n_0 = 1, \quad c_1 = 9,$

NOTE- If the degree of $c(n)$ is same as the degree of $g(n)$, then $f(n)$ can be expressed as

$$f(n) = \theta(g(n))$$

It specifies the time complexity in Tight bound.



4. Little Oh (o) Notation →

Let $f(n)$ and $g(n)$ are two non-negative functions then, the function, $f(n) = o(g(n))$ iff there are 2 positive constants c and n_0 such that.

$$f(n) < c.g(n), \quad \forall\ n \geq n_0 \text{ and } \underline{\forall}\ c > 0$$

Q. If $f(n) = 2n^2+2$, justify that $f(n) \neq o(n^2)$

Sol. $f(n) = 2n^2+2$

$f(n) \not< 1 \cdot n^2$

Hence, $f(n)$ is not little oh of n^2.

Q. If $f(n) = 2n^2+2$, then show that $f(n) = o(n^3)$

Sol. $f(n) < 1 \cdot n^3$, $\forall n \geq n_0 = 3$, $c > 0$

If the degree of $g(n) \geq$ degree of $f(n)$, then $f(n)$ can be expressed as

$$f(n) = o\,g(n).$$

It gives the fine complexity in strictly Upper bound.

5. Little Omega (ω) Notation →

Let $f(n)$ and $g(n)$ are two non-negative functions Then the function, $f(n) = \omega(g(n))$ iff there are 2 positive constants c and n_0 such that

$$f(n) > c \cdot g(n), \quad \forall n \geq n_0 \text{ and } \forall c > 0$$

LectureNotes.in

Q. Write the difference between Big Oh and Little Oh.

Sol.

Big Oh	Little Oh
→ $f(n) = O(g(n))$	→ $f(n) = o(g(n))$
$f(n) \leq c \cdot g(n)$	$f(n) < c \cdot g(n)$
→ It satisfies for some value of 'c'	→ It satisfies for all values of 'c'.
→ Upper Baund	→ Strictly Upper Baund

Q. Write the difference between Big Omega and Little Omega.

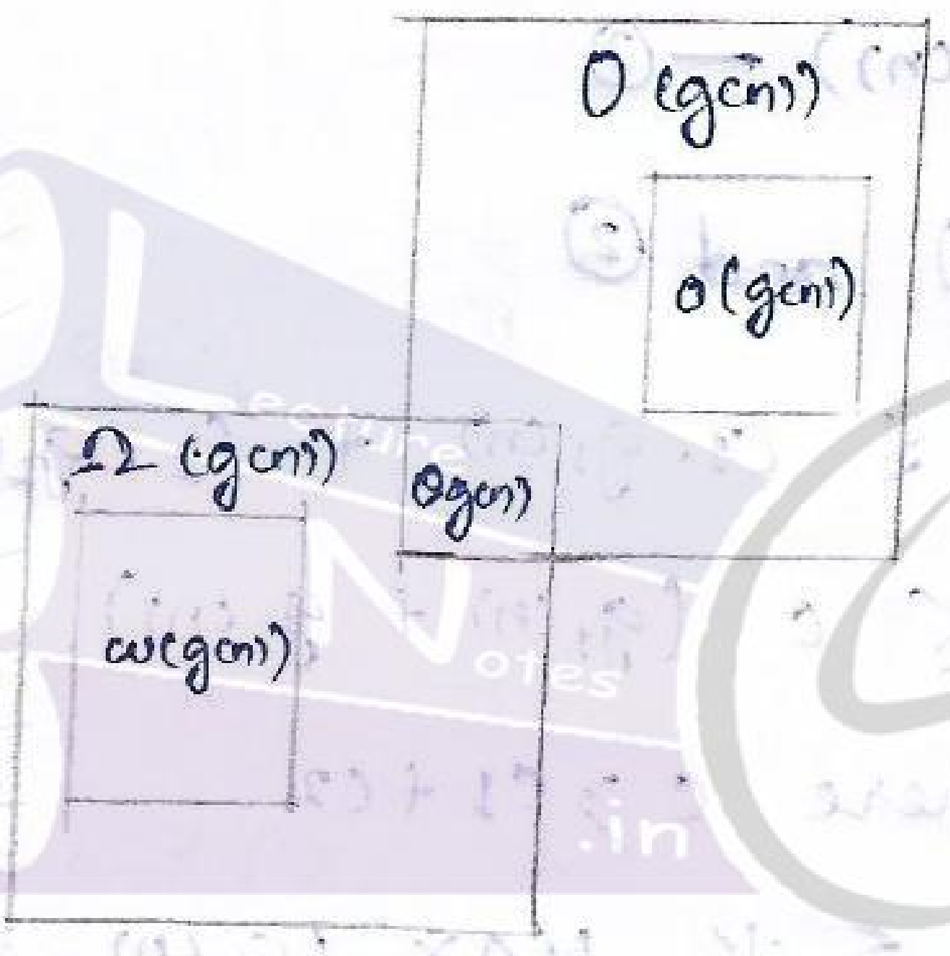
Sol.

Big Omega	Little Omega
→ $f(n) \geq c \cdot g(n)$	→ $f(n) > c \cdot g(n)$
→ It satisfies for all values of 'c'	→ It satisfies for some values of 'c'.
→ Lower Baund	→ Strictly lower Baund

LectureNotes.in

Dt → 29th Jan 18
Monday

1. $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$, $\exists c > 0 \Rightarrow f(n) = O(g(n))$
2. $f(n) \geq c \cdot g(n)$, $\forall n \geq n_0$, $\exists c > 0 \Rightarrow f(n) = \Omega(g(n))$
3. $c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)$, $\forall n \geq n_0$, $\exists c > 0$
   $\Rightarrow f(n) = \Theta(g(n))$
4. $f(n) < c \cdot g(n)$, $\forall n > n_0$, $\forall c > 0 \Rightarrow f(n) = o(g(n))$
5. $f(n) > c \cdot g(n)$, $\forall n > n_0$, $\forall c > 0 \Rightarrow f(n) = \omega(g(n))$



Properties of Asymptotic Notations →

1. If $f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$
   but the reverse is not true
2. If $f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n))$
   but the reverse is not true
3. If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ then
   $f(n) = \Theta(g(n))$
4. If $f_1(n) + f_2(n) = MAX(g_1(n), g_2(n))$

5. Let $f_1(n) + f_2(n) = MIN(g_1(n), g_2(n))$

where $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$

Q. Prove that $f_1(n) + f_2(n) = MAX(g_1(n), g_2(n))$

where $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$.

Sol. $f_1(n) = O(g_1(n))$.

$\Rightarrow f_1(n) \leq c_1 \cdot (g_1(n)) \quad \text{---} ①$

$f_2(n) = O(g_2(n))$

$\Rightarrow f_2(n) \leq c_2 \cdot (g_2(n)) \quad \text{---} ②$

Adding eqn ① and ②

$\Rightarrow f_1(n) + f_2(n) \leq c_1 \cdot g_1(n) + c_2 \cdot g_2(n)$

$\Rightarrow f_1(n) + f_2(n) \leq c\,(g_1(n) + g_2(n))$

where $c \geq c_1 + c_2$

$\Rightarrow f_1(n) + f_2(n) \leq K\ MAX(g_1(n), g_2(n))$

where $K = 2c$

$\Rightarrow$ $f_1(n) + f_2(n) = O(MAX(g_1(n), g_2(n)))$

Q. Prove that $f_1(n) + f_2(n) = MIN(g_1(n), g_2(n))$

where $f_1(n) = \Omega(g_1(n))$ and $f_2(n) = \Omega(g_2(n))$

Sol. $f_1(n) = \Omega(g_1(n))$

$\Rightarrow f_1(n) \geq c_1 \cdot (g_1(n)) \quad \text{---} ①$

$f_2(n) = \Omega(g_2(n))$

$\Rightarrow f_2(n) \geq c_2 \cdot (g_2(n)) \quad \text{---} ②$

Adding eq^n ① and ②,

$\Rightarrow f_1(n) + f_2(n) \geq c_1 \cdot g_1(n) + c_2 \cdot g_2(n)$

$\Rightarrow f_1(n) + f_2(n) \geq c\,(g_1(n) + g_2(n))$.

where $c < c_1$ and $c < c_2$.

$\Rightarrow f_1(n) + f_2(n) \geq \text{MIN}\,(g_1(n), g_2(n))$

$\Rightarrow$ $f_1(n) + f_2(n) = \Omega(\text{MIN}\,(g_1(n), g_2(n)))$.

Q. Prove that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. then $f(n) = \Theta(g(n))$

Sol. $f_1(n) = O(g_1(n))$

$\Rightarrow f_1(n) \leq c_1 \cdot g_1(n)$ —— ①

$f_1(n) = \Omega(g_1(n))$

$\Rightarrow f_1(n) \geq c_2 \cdot g(n)$ —— ②

from eq^n ① and ②,

$c_2 \cdot g(n) \leq f_1(n) \leq c_1 \cdot g(n)$

$\Rightarrow f(n) = \Theta(g(n))$

Q. Prove that $\log(n!) = O(n \log n)$.

Sol. $\log(n!) = \log(n * (n-1) * (n-2) * \ldots * 1)$

$\Rightarrow \log(n!) \leq \log(n * n * n * \ldots * n)$

$\Rightarrow \log(n!) \leq \log n^n$

$\Rightarrow \log(n!) \leq n \log n$, $\forall n \geq n_0$ and $c = 1$

$\Rightarrow \boxed{\log(n!) = O(n \log n)}$

Expressing Asymptotic Notations using Limits Theorem →

Let f(n) and g(n) are two non-negative functions, then

1. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = O(g(n))$

2. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then $f(n) = \omega(g(n))$

3. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, then $f(n) = \Theta(g(n))$

where $c > 0$ (positive constant)

Q. Show that $2n^2+5 = O(n^3)$ using limits theorem.

Sol. $2n^2+5 = O(n^3)$

$$\lim_{n\to\infty} \frac{2n^2+5}{n^3} = \lim_{n\to\infty} \left(\frac{2n^2}{n^3} + \frac{5}{n^3}\right)$$

$$= \lim_{n\to\infty} \left(\frac{2}{n} + \frac{5}{n^3}\right)$$

$$= \lim_{n\to\infty} \left(\frac{2}{n}\right) + \lim_{n\to\infty} \left(\frac{5}{n^3}\right)$$

$$= 0 + 0$$

$$= 0$$

Therefore, $2n^2+5$ is $= O(g(n))$

As it is $2n^2+5 = O(g(n))$, it is also

$$\boxed{2n^2+5 = O(g(n))}$$

Dt - 30th Jan 18
Tuesday

Q. Using limits theorem, Prove that

$$n^2+5 = \Theta(n^2)$$

Sol. $n^2+5 = \Theta(n^2)$

$$\lim_{n\to\infty} f\left(\frac{n^2+5}{n^2}\right) = \lim_{n\to\infty}(1) + \lim_{n\to\infty}\left(\frac{5}{n^2}\right)$$

$$= 1 + 0$$

$$= 1$$

Therefore,

$$n^2+5 = \Theta(n^2)$$

$$n^2+5 = \Theta(g(n))$$

Q. Show that $2^{n+1} = O(2^n)$ using limit theorem.

Sol. $2^{n+1} = O(2^n)$

$$\lim_{n\to\infty} \frac{2\cdot 2^n}{2^n} = \lim_{n\to\infty} 2 = 2$$

$$2^{n+1} = \Theta(2^n)$$

Therefore,

$$2^{n+1} = O(2^n)$$

Q. Justify that $3n+2 = O(n^2)$

Sol. $3n+2 = O(n^2)$

$$\lim_{n\to\infty} \frac{3n+2}{n^2} = \lim_{n\to\infty}\left(\frac{3}{n}\right) + \lim_{n\to\infty}\left(\frac{2}{n^2}\right) = 0$$

$$3n+2 = O(n^2) = O(g(n))$$

LectureNotes.in

8. Arrange the following elements in the increasing order of Asymptotic Growth Rate.

n, $\log n$, n^3, $n \log n$, $n!$, 2^n, n^n, 25.

Sol. The increasing order of Asymptotic Growth Rate is -

25, $\log n$, n, $n \log n$, n^3, 2^n, $n!$, n^n

Q. Show that $n!$ is $n! = O(n^n)$. using limits theorem.

Sol. $n! = O(n^n)$

$\lim_{n \to \infty} \frac{n!}{n^n}$ = It can't be solved by limits theorem.

Stirling's Formula -

Stirling gave an approximate value for $n!$, i.e.,

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \frac{c}{n}\right), \text{ where } c > 0$$

Then,

$$\lim_{n \to \infty} \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \frac{c}{n}\right)}{n^n}$$

$$= \lim_{n \to \infty} \frac{\sqrt{2\pi n} \cdot n^n \left(1 + \frac{c}{n}\right)}{e^n \cdot n^n}$$

$$= \lim_{n \to \infty} \frac{\sqrt{2\pi n} \times \left(1 + \frac{c}{n}\right)}{e^n}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi n} + \sqrt{2\pi n} \times \frac{c}{n}}{e^n}$$

$$= \lim_{n\to\infty} \frac{n\sqrt{2\pi n} + c\cdot\sqrt{2\pi}}{e^n \sqrt{n}}$$

$$= \lim_{n\to\infty} \frac{n\sqrt{2\pi} + c\sqrt{2\pi}}{e^n \sqrt{n}}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi}\cdot n}{e^n \sqrt{n}} + \lim_{n\to\infty} \frac{c\sqrt{2\pi}}{e^n \sqrt{n}}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi n}}{e^n} + \lim_{n\to\infty} \frac{c\sqrt{2\pi}}{e^n \sqrt{n}}$$

$$= \lim_{n\to\infty} \frac{\sqrt{n}\cdot\sqrt{2\pi}}{e^n} + \lim_{n\to\infty} \frac{c\sqrt{2\pi}}{e^n \sqrt{n}}$$

$$= 0 + 0$$

$$= 0$$

Therefore,

$$n! = O(n^n)$$

LectureNotes.in

Recurrance

Dt- 2nd Feb 18
Friday

* A Recurrance is an equation or inequality that describes the function in terms of it's smaller value.

* Recurrance is, usually, used in divided and conquer approach.

* Recurrance is found when the algorithm is recursive in nature.

* Ex - → Recurrance of Binary Search →

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

→ Recurrance of n! →

$$T(n) = T(n-1) + 1$$

→ Recurrance of Quick Sort →

$$T(n) \leq 2T\left(\frac{n}{2}\right) + n$$

Solving the Recurrance →

The Recurrance can be solved in any of the following approaches -

1. Master Method
   - Variations of Master Method
     (a) Changing Variable Method
     (b) Extension Master Method.

LectureNotes.in

2. Iteration Method / Recursive Tree Method
3. Substitution Method

1. Master Method -

* The Master Method is used to solve the recurrance if the recurrance is in the following forms -

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$, $b \geq 1$ and $f(n) \geq 0$

* Master Method has the following three theorems -

Theorem 1 -

If $$f(n) = O(n^{\log_b^a - \varepsilon}), \exists\ \varepsilon > 0$$

then $$T(n) = \theta(n^{\log_b^a})$$

Theorem 2 -

If $$f(n) = \theta(n^{\log_b^a})$$

then $$T(n) = \theta(n^{\log_b^a} \log n)$$

LectureNotes.in

Theorem 3 -

If $f(n) = \Omega(n^{\log_b^a + \varepsilon})$, $\exists\ \varepsilon > 0$

and

it should satisfy the Regularity Condition (Neccessary Condition) -

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n), \quad \exists\ 0 < c < 1$$

then

$$T(n) = \Theta(f(n))$$

Q. Solve the Recurrance -

1. $T(n) = 4T\left(\frac{n}{2}\right) + n$

Sol. Here,

$a = 4, \ b = 2, \ f(n) = n$

$$n^{\log_b^a} = n^{\log_2^4} = n^2$$

$$f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$$

It satisfies Master Theorem 1.

According to Master Theorem 1,

$$T(n) = \Theta\left(n^{\log_b^a}\right)$$

$$T(n) = \Theta(n^2)$$

LectureNotes.in

Q. Solve the Recurrance

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

Sol. Here,

$$a = 9, \quad b = 3, \quad f(n) = n$$

$$n^{\log_b^a} = n^{\log_3^9} = n^2$$

$$f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$$

It satisfies Master Theorem 1.

According to Master Theorem 1,

$$T(n) = \theta\left(n^{\log_b^a}\right)$$

$$T(n) = \theta(n^2)$$

Q. Solve the Recurrance

$$T(n) = 8T\left(\frac{n}{2}\right) + n\log n$$

Sol. Here,

$$a = 8, \quad b = 2, \quad f(n) = n\log n$$

$$n^{\log_b^a} = n^{\log_2^8} = n^3$$

As $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$

It satisfies Master Theorem 1.

According to Master Theorem 1,

$$T(n) = \theta\left(n^{\log_b^a}\right)$$

$$T(n) = \theta(n^3)$$

Q. Solve the Recurrance

$$T(n) = 7\,T\left(\frac{n}{2}\right) + n^2$$

Sol. Here,

$$a = 7 \;,\; b = 2 \;,\; f(n) = n^2$$

$$n^{\log_b^a} = n^{\log_2^7} = n^{2+\varepsilon}$$

As, $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$ where $0 < \varepsilon < \varepsilon$

It satisfies Master Theorem 1.

So, according to Master Theorem 1,

$$T(n) = \theta\left(n^{\log_b^a}\right)$$

$$T(n) = \theta\left(n^{2-\varepsilon}\right)$$

Q. Solve the Recurrance of Binary Search Algorithm.

Sol. The Recurrence of Binary Search is

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Here, $a = 1, b = 2, f(n) = 1$

$$n^{\log_b^a} = n^{\log_2^1} = n^0 = 1$$

As, $f(n) = \theta\left(n^{\log_b^a}\right)$

It satisfies Master Theorem 2,

So, according to Master Theorem 2,

$$T(n) = \theta\left(n^{\log_b^a} \log n\right)$$

$$T(n) = \theta(\log n)$$

LectureNotes.in

The Binary Search is faster than Linear Search

Q. Solve the Recurrance of Quick Sort in best case.

Sol. The recurrance of Quick Sort in best case is

$$T(n) \leq 2T\left(\frac{n}{2}\right) + n$$

Here, $a = 2$, $b = 2$, $f(n) = n$

$$n^{\log_2^2} = n$$

As, $f(n) = \theta(n^{\log_b^a})$

According to Master Theorem 2,

$$T(n) = \theta(n^{\log_b^a} \log n)$$

$$T(n) = \theta(n \log n)$$

But according to the question,

$$T(n) = O(n \log n)$$

Q. Solve the Recurrance,

1. $T(n) = 2T\left(\frac{n}{2}\right) + n$

2. $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

3. $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

Sol. 1 $T(n) = 2T\left(\frac{n}{2}\right) + n$

Here, $a = 2$, $b = 2$, $f(n) = n$

$n^{\log_2^2} = n$

As, $f(n) = \Theta(n^{\log_b^a})$

According to Master theorem 2,

$$T(n) = \Theta(n^{\log_b^a} \log n)$$

$$T(n) = \Theta(n \log n)$$

Sol 2 $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

Here, $a = 4$, $b = 2$, $f(n) = n^2$

$n^{\log_2^4} = n^2$

As, $f(n) = \Theta(n^{\log_b^a})$

According to Master Theorem 2,

$$T(n) = \Theta(n^{\log_b^a} \log n)$$

$$T(n) = \Theta(n^2 \log n)$$

Sol 3 $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

Here, $a = 8$, $b = 2$, $f(n) = n^2$

$n^{\log_2^8} = n^3$

As $f(n) = O(n^{\log_b^a - \epsilon})$

According to Master Theorem 3,

$$T(n) = \Theta(n^{\log_b^a})$$

$$T(n) = \Theta(n^2)$$

LectureNotes.in

Dt → 3rd Feb 18
Saturday

Q. Solve the Recurrance

$$T(n) = 2T\left(\frac{n}{2}\right) + n^4$$

Sol. $a = 2, \; b = 2, \; f(n) = n^4$

$$n^{\log_2^2} = n$$

$$f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$$

According to Master Theorem 3,

Checking the Necessary Conditions for Master Theorem 3,

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n) \;, \; \exists \; 0 < c < 1$$

$$\Rightarrow 2 \cdot \frac{n^4}{16} \leq c \cdot n^4$$

$$\Rightarrow \frac{n^4}{8} \leq c \cdot n^4 \;, \; \frac{1}{8} \leq c < 1$$

Hence, it satisfies the Regularity conditions of Master Theorem 3.

According to Master Theorem 3, the solution is

$$T(n) = \Theta(f(n))$$

$$\boxed{T(n) = \Theta(n^4)}$$

Q. Solve the Recurrance

$$T(n) = 2T\left(\frac{n}{2}\right) + n^3$$

Sol. Here, $a = 2, \quad b = 2, \quad f(n) = n^3$

$$n^{\log_2^2} = n$$

$$f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$$

Checking Necessary Conditions for Master Theorem 3,

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n), \quad \exists\ 0 < c < 1$$

$$\Rightarrow 2 \cdot \frac{n^3}{8} \leq c \cdot n^3$$

$$\Rightarrow \frac{n^3}{4} \leq c \cdot n^3, \quad \frac{1}{4} \leq c < 1$$

Hence, it satisfies Regularity conditions of Master Theorem 3.

According to Master Theorem 3,

$$T(n) = \Theta(f(n))$$

$$\boxed{T(n) = \Theta(n^3)}$$

Q. Solve the Recurrance

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$$

Sol. Here,

$$a = 3, \quad b = 4, \quad f(n) = n\log n$$

$$n^{\log_b^a} = n^{\log_4^3} = n^{0.7} = n^{\varepsilon}, \quad \varepsilon \rightarrow 0 < \varepsilon < 1$$

$$f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$$

$$n\log n = \Omega\left(n^{\varepsilon + \varepsilon}\right)$$

Let's check the necessary conditions of Master Theorem 3,

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

$$\Rightarrow 3 \cdot f\left(\frac{n}{4}\right) \leq c \cdot f(n)$$

LectureNotes.in

$\Rightarrow 3.\frac{n}{4}\log\frac{n}{4} \leq c.n\log n$

$\Rightarrow \frac{3n}{4}(\log n - \log 4) \leq c.n\log n$

$\Rightarrow \frac{3n}{4}(\log n - 2) \leq c.n\log n$

$\Rightarrow \frac{3n}{4}\log n - \frac{3n}{2} \leq c.n\log n$

$\Rightarrow \frac{3n}{4}\log n \leq c.n\log n + \frac{3n}{2}$, $\frac{3}{4} \leq c < 1$

Hence, it satisfies Regularity conditions of Master Theorm 3,

$$T(n) = \Theta(f(n))$$

$$\boxed{T(n) = \Theta(n\log n)}$$

Q. Solve the Recurrance

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

Sol. Here,..

$a=4$, $b=2$, $f(n) = n^3$

$n^{\log_b^a} = n^{\log_2^4} = n^2$

Then,

$f(n) = n^3$

$f(n) = n^{\log_b^a + \varepsilon}$

$f(n) = \Omega(n^{\log_b^a + \varepsilon})$

Checking the necessary conditions of Master Theorem 3,

LectureNotes.in

$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

$4 \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

$4 f\left(\frac{n^3}{8}\right) \leq c \cdot f(n)$

$\frac{n^3}{2} \leq c \cdot f(n)$

$\frac{1}{2} \cdot n^3 \leq c \cdot f(n)$

$c = \frac{1}{2}$

$\frac{1}{2} \leq c \leq 1$

Q. Solve the Recurrance

$$T(n) = T\left(\frac{9n}{10}\right) + n$$

Sol. Here,

$$a = 1, \quad b = \frac{10}{9}, \quad f(n) = n$$

$$n^{\log_b^a} = n^{\log_{10/9}^{(10/9)^0}} = n^{\log_{10/9}^{1}} = n^0 = 1$$

$$f(n) = \Omega\left(n^{\log_b^a + \epsilon}\right)$$

$$f(n) = \Omega\left(n^{0+\epsilon}\right)$$

Let's check the necessary conditions of Master theorem 3,

$\Rightarrow a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

$\Rightarrow 1 f\left(\frac{n}{10/9}\right) \leq c \cdot n$

$\Rightarrow \frac{9}{10} n \leq c \cdot n \quad , \frac{9}{10} \leq c < 1$

Hence, it satisfies Regularity conditions of Master Theorem 3,

$$T(n) = \theta(f(n))$$

$$\Rightarrow \boxed{T(n) = \theta(n)}$$

Master Theorem by changing the Variables

LectureNotes.in

If the recurrance is not in the form of

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

by changing the variables, the recurrance can be converted to General form of Master Theorem.

Q. Solve the Recurrance

$$T(n) = 2T(\sqrt{n}) + \log n \quad \text{---} (1)$$

Sol. Let $n = 2^m$

Now, $T(2^m) = 2T(\sqrt{2^m}) + \log 2^m$

$$\Rightarrow T(2^m) = 2T(2^{m/2}) + m$$

Let $T(2^m) = S(m)$,

then $S\left(\frac{m}{2}\right) = T(2^{m/2})$

$$S(m) = 2S\left(\frac{m}{2}\right) + m \quad \text{---} (2)$$

Now,

The eqn ② is in the form of Master Theorem.

Here,

$$a = 2, \quad b = 2, \quad f(m) = m$$

$$m^{\log_b^a} = m^{\log_2^2} = m$$

$$f(m) = \theta(m^{\log_b^a})$$

$$f(m) = \theta(m)$$

According to Master Theorem 2,

$$f(m) = \theta(m^{\log_b^a} \log m)$$

$$f(m) = \theta(m \log m)$$

Now,

$$n = 2^m$$

Taking log both sides,

$$\log n = m$$

Then,

$$T(2^m) = \theta(\log n \log \log n)$$

Therefore,

$$\boxed{T(n) = \theta(\log n \log \log n)}$$

LectureNotes.in

Q. Solve the Recurrance

$$T(n) = T(\sqrt{n}) + 1$$

Sol. $T(n) = T(\sqrt{n}) + 1$ ——①

$$T(n) = T(2^m) + T(\sqrt{2^m}) + 1$$

$$= T(2^{m/2}) + 1$$

$$T(2^m) = S(m)$$

Here,

$$S\left(\frac{m}{2}\right) = T(2^{m/2})$$

$$S(m) = S\left(\frac{m}{2}\right) + 1 \quad \text{——②}$$

Now eqⁿ ② is in the form of general form of Master Theorem,

Here $a=1$, $b=2$, $f(m)=1$

$$m^{\log_b^a} = m^{\log_2^1} = m^0 = 1$$

$$f(m) = \theta(m^{\log_b^a})$$

Hence, it satisfy master Theorem 2

$$g(m) = \theta(m^{\log_b^a} \log m)$$

$$= \theta(m^0 \log m)$$

$$= \theta(\log m)$$

LectureNotes.in

Q. Solve the Recurrance

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

Sol. Here,

$$a = 4, \quad b = 2, \quad f(n) = n^3$$

$$n^{\log_b^a} = n^{\log_2^4} = n^2$$

$$f(n) = n^3$$

$$f(n) = n^{\log_b^a + \varepsilon}$$

$$f(n) = \Omega\left(n^{\log_b^a + \varepsilon}\right)$$

let the necessary condition of recurrance relation.

$$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

$$4 f\left(\frac{n}{b}\right) \leq c \cdot f(n)$$

$$4\left(\frac{n^3}{8}\right) \leq c \cdot f(n)$$

$$\frac{n^3}{2} \leq c \cdot f(n)$$

$$\frac{1}{2} n^3 \leq c \cdot f(n)$$

$$c = \frac{1}{2}$$

$$\frac{1}{2} \leq c \leq 1$$

LectureNotes.in

Dt - 5th Feb 2018
Monday

Extension Master Method -

If the recurrance is in the form of

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \text{ where } a > 0 \text{ and } b > 0$$

and if

$$f(n) = \Theta(n^{\log_b^a} (\log)^k n) \text{ where } k > 0$$

then

$$T(n) = \Theta(n^{\log_b^a} (\log)^{k+1} n)$$

Q. Justify that the following Recurrance does not satisfy Master Theorem 1, 2 and 3. Find the time bound (Θ) of the recurrance.

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Sol. Here,

$$a = 2, \quad b = 2, \quad f(n) = n \log n$$

$$n^{\log_b^a} = n^{\log_2^2} = n$$

Now,

$$f(n) \neq \Omega(n^{\log_b^a + \varepsilon})$$

$$\downarrow \qquad \downarrow$$

$$n \log n \quad \geq \quad n$$

As $f(n)$ is not polynomially larger then $n^{\log_b^a}$.

Hence, we cannot apply Master Theorem 3.

Applying Extension Master Theorem,

the solution is

$$T(n) = \theta\left(n^{\log_b^a} \cdot (\log)^{K+1} n\right)$$

$$\boxed{T(n) = \theta\left(n \log^2 n\right)}$$

Q. Solve the Recurrance

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log n$$

Sol. Here,

$$a = 4, \quad b = 2, \quad f(n) = n^2 \log n$$

$$n^{\log_2^4} = n^2$$

Now,

$$f(n) \neq \Omega\left(n^{\log_b^a + \varepsilon}\right)$$

$\downarrow$ $\qquad$ $\downarrow$

$n^2 \log n$ $\qquad$ n^2

As $f(n)$ is not polynomially larger than $n^{\log_b^a}$, we cannot apply Master Theorem 3,

Applying Extension Master Theorem,

the solution is

$$T(n) = \theta\left(n^{\log_b^a} (\log)^{K+1} n\right)$$

$$\boxed{T(n) = \theta\left(n^2 \log^2 n\right)}$$

Some Mathematical Formulae →

(1) $1+2+3+\ldots\ldots+n = \frac{n(n+1)}{2}$

(2) $1^2+2^2+3^2+\ldots\ldots+n^2 = \frac{n(n+1)(2n+1)}{6}$

(3) $\sum_{i=0}^{\infty}(x)^i = \frac{1}{1-x}$ when $x<1$

(4) $\sum_{i=0}^{n}(x)^i = \frac{x^{(n+1}-1}{x+1}$, where $x>1$

Iteration Method

Q. Solve the following Recurrance,

$$T(n) = T(n-1)+1$$

Sol. $T(n) = T(n-1)+1$

$= [T(n-2)+1]+1$

$= T(n-2)+1+1$

$= T(n-3)+1+1+1$

$\vdots$

$= 1+1+1+\ldots+T((n)-(n-1))$

$= 1+1+1+\ldots+T(1)$

$= 1+1+1+\ldots+1$

Therefore,

LectureNotes.in

The above problem is solved by using Iteration Method.

Recursive Tree Method

* In Recursive Tree Method,

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

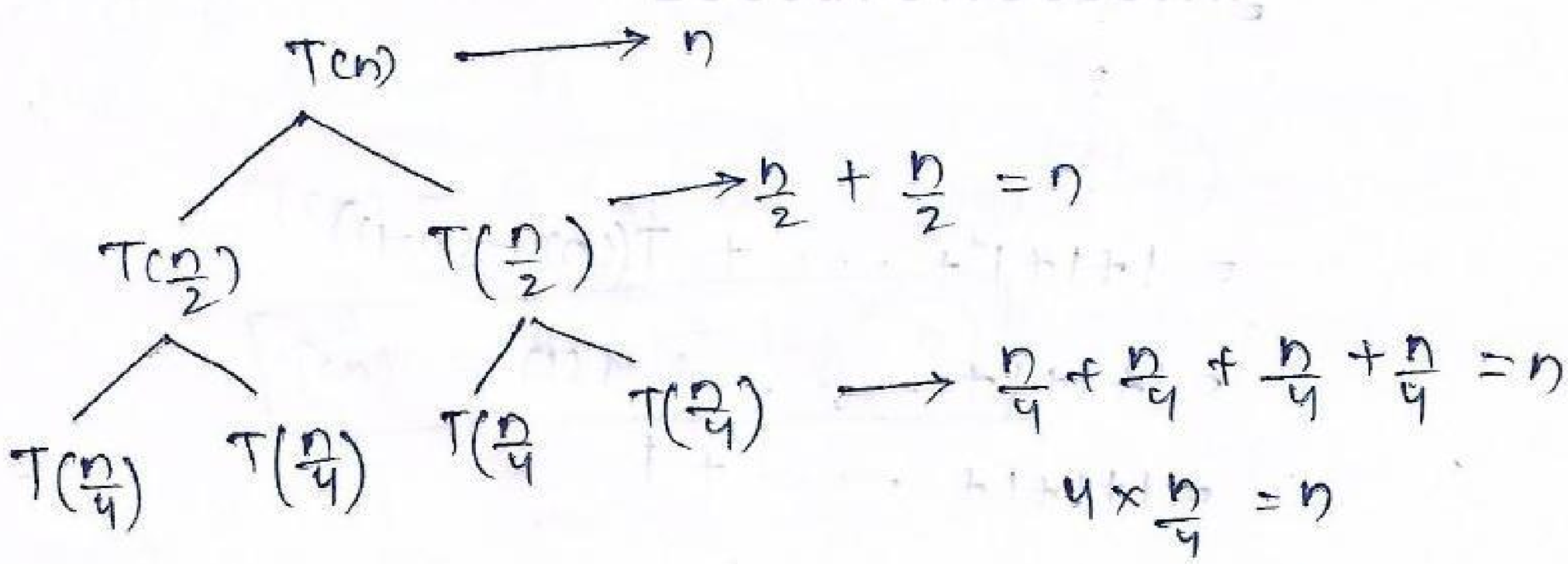
means that the input size n is divided into each of size $\frac{n}{b}$ and the cost of division is $f(n)$.

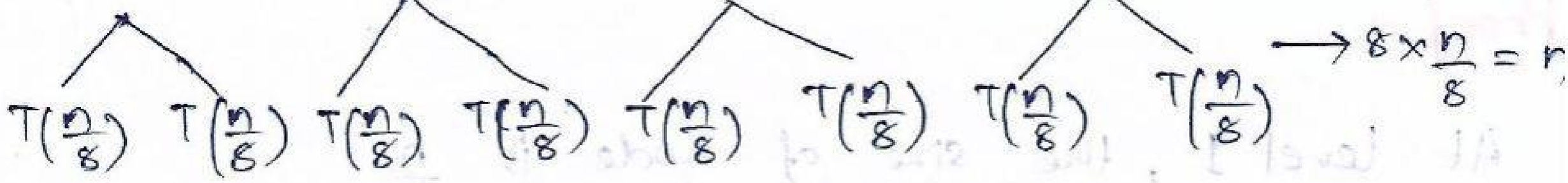
* It needs to sum the cost of each nodes of the tree to find overall cost.

Example -

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Sol. $T(n) = 2T\left(\frac{n}{2}\right) + n$



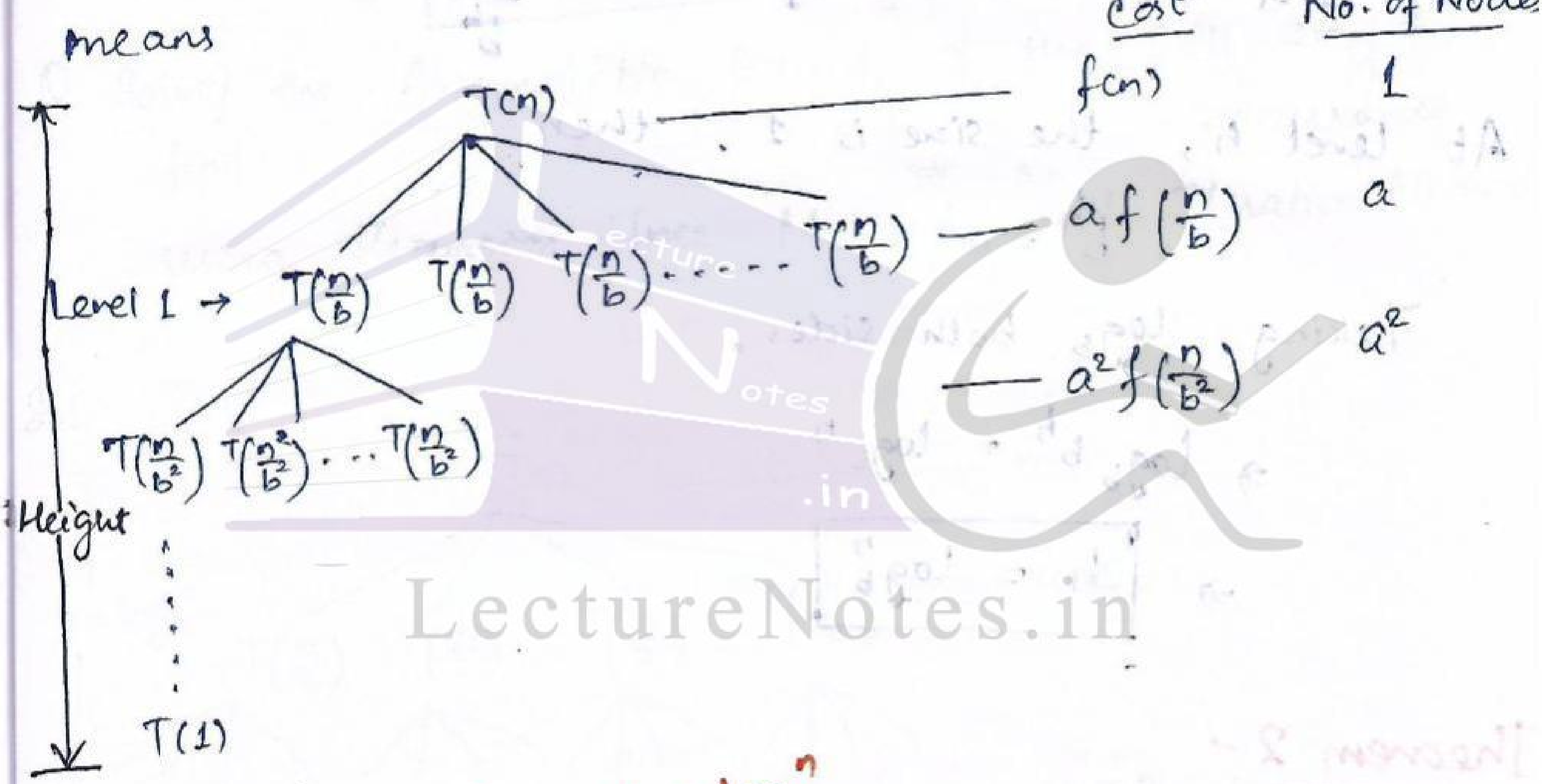


Dt → 6th Feb 18

Tuesday

LectureNotes.in

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$



LectureNotes.in

Theorem 1 :- Height $= \log_b^n$

If the recurrance is $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, then the height of the Tree is

$$h = \log_b^n$$

P

Proof →

At level 1, the size of node is $\frac{n}{b}$

At level 2, the size of node is $\frac{n}{b^2}$

At level 3, the size of node is $\frac{n}{b^3}$

⋮

At level h, the size of node is $\frac{n}{b^h}$

At level h, the size is 1, then

$$b^h = n$$

Taking $\log_b$ both sides,

$$\Rightarrow \log_b b^h = \log_b n$$

$$\Rightarrow \boxed{h = \log_b n}$$

Theorem 2 →

If the recurrance is $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ then the number of nodes at last level is $n^{\log_b^a}$.

$a^h = n^{\log_b^a}$

Proof →

At level 1, the no. of node is a.

At level 2, the no. of node is a^2.

At level h, the no. of nodes is a^h.

Therefore,

The no. of nodes at last level is a^h.

$$\Rightarrow a^h = a^{\log_b^n}$$

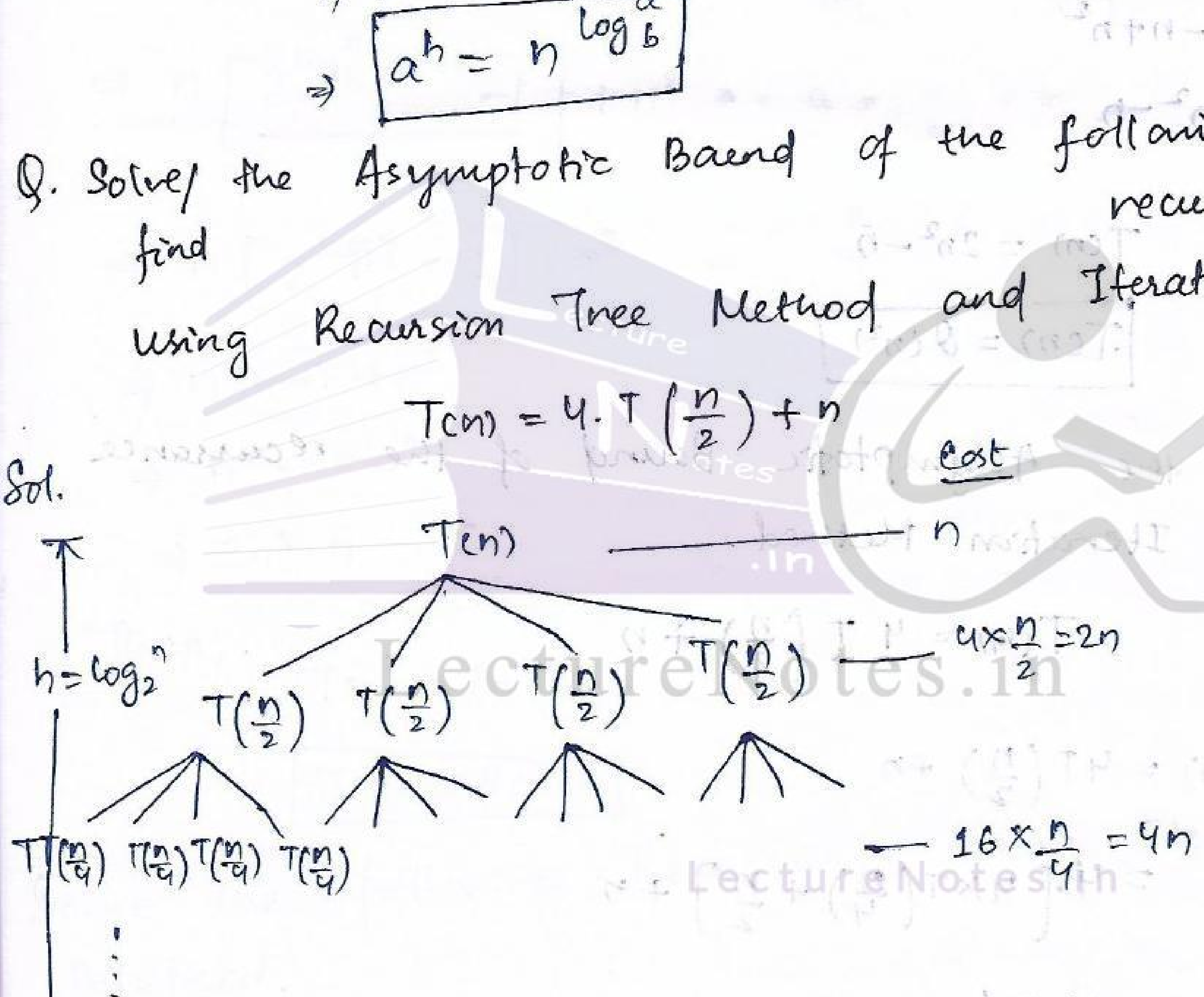
$$\Rightarrow a^h = a^{\log_b^n}$$

$$\Rightarrow \boxed{a^h = n^{\log_b^a}}$$

Q. Solve/ find the Asymptotic Baend of the following recurrance using Recursion Tree Method and Iteration Method.

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n$$

Sol.



Add the cost of each level to get overall cost,

$$n + 2n + 4n + 8n + \cdots + T(1) \cdot n^{\log_2^4}$$

$$\Rightarrow n(1 + 2 + 4 + 8 + \cdots) + n^2$$

$$\Rightarrow n \sum_{i=0}^{\log_2 n - 1} (2)^i + n^2$$

$$\because \sum_{i=0}^{n} (x)^i = \frac{x^{n+1} - 1}{x - 1}, \quad x > 1$$

$$\Rightarrow n \left[\frac{2^{\log_2 n - 1 + 1} - 1}{2 - 1} \right] + n^2$$

$$\Rightarrow n \left[2^{\log_2 n} - 1 \right] + n^2$$

$$\Rightarrow n (n - 1) + n^2$$

$$\Rightarrow n^2 - n + n^2$$

$$\Rightarrow 2n^2 - n$$

then,

$$T(n) = 2n^2 - n$$

$$\boxed{T(n) = \theta(n^2)}$$

Q. Find the Asymptotic Bound of the recurrance using Iteration Method.

$$T(n) = 4\,T\left(\frac{n}{2}\right) + n$$

Sol.

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$= 4\left[4 \times T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$= 16\,T\left(\frac{n}{4}\right) + 2n + n$$

$$= 16\left[4\,T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n + n$$

$$= 64.T\left(\frac{n}{8}\right) + 4n + 2n + n$$

$$= 64\left[4\,T\left(\frac{n}{16}\right) + \frac{n}{8}\right] + 4n + 2n + n$$

LectureNotes.in

$$\Rightarrow n \sum_{i=0}^{\log_2 n - 1} (2)^i + n^2$$

$$\because \sum_{i=0}^{n} (x)^i = \frac{x^{n+1} - 1}{x - 1}, \quad x > 1$$

$$\Rightarrow n \left[\frac{2^{\log_2 n - 1 + 1} - 1}{2 - 1} \right] + n^2$$

$$\Rightarrow n \left[2^{\log_2 n} - 1 \right] + n^2$$

$$\Rightarrow n (n - 1) + n^2$$

$$\Rightarrow n^2 - n + n^2$$

$$\Rightarrow 2n^2 - n$$

Then,

$$T(n) = 2n^2 - n$$

$$\boxed{T(n) = \theta(n^2)}$$

Q. Find the Asymptotic Bound of the recurrence using Iteration Method.

$$T(n) = 4\,T\left(\frac{n}{2}\right) + n$$

Sol. $$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$= 4\left[4 \times T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$

$$= 16\,T\left(\frac{n}{4}\right) + 2n + n$$

$$= 16\left[4\,T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n + n$$

$$= 64.T\left(\frac{n}{8}\right) + 4n + 2n + n$$

$$= 64\left[4\,T\left(\frac{n}{16}\right) + \frac{n}{8}\right] + 4n + 2n + n$$

LectureNotes.in

$= 256\,T\left(\frac{n}{16}\right) + 8n + 4n + 2n + n$

.
.
.

$\Rightarrow n + 2n + 4n + 8n + \cdots + T(1)\, n^{\log_2^4}$

$\Rightarrow n + 2n + 4n + 8n + \cdots + n^2$

$\Rightarrow n \sum_{i=0}^{\log_2^n - 1} (2)^i + n^2$ $\quad \because \sum_{i=0}^{n} (x)^i = \frac{x^{n+1} - 1}{x - 1}, \; x > 1$

$\Rightarrow n\left[\frac{2^{\log_2^n - 1 + 1} - 1}{2 - 1}\right] + n^2$

$\Rightarrow n\left[2^{\log_2^n} - 1\right] + n^2$

$\Rightarrow n(n-1) + n^2$

$\Rightarrow n^2 - n + n^2$

$\Rightarrow 2n^2 - n$

Then, $T(n) = 2n^2 - n$

$$\boxed{T(n) = \theta(n^2)}$$

Solve the following Recurrance using Iteration Method.

$$T(n) = 2T\left(\frac{n}{4}\right) + n$$

Sol.

$$T(n) = 2T\left(\frac{n}{4}\right) + n$$

$$= 2\left[2T\left(\frac{n}{16}\right) + \frac{n}{4}\right] + n$$

LectureNotes.in

$= 2\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + n$

$= 4T\left(\frac{n}{8}\right) + \frac{n}{2} + n$

$= 4\Big[$

$T(n) = 2T\left(\frac{n}{4}\right) + n$

$= 2\left[2T\left(\frac{n}{16}\right) + \frac{n}{4}\right] + n$

$= 4T\left(\frac{n}{16}\right) + \frac{n}{2} + n$

$= 4\left[2T\left(\frac{n}{64}\right) + \frac{n}{16}\right] + \frac{n}{2} + n$

$= 8T\left(\frac{n}{64}\right) + \frac{n}{4} + \frac{n}{2} + n$

⋮

$= n + \frac{n}{2} + \frac{n}{4} + \cdots + T(1)\ n^{\log_4 2}$

$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \sqrt{n}\right)$

LectureNotes.in

$$= 256\,T\left(\frac{n}{16}\right) + 8n + 4n + 2n + n$$

$$\vdots$$

$$\Rightarrow n + 2n + 4n + 8n + \cdots + T(1)\, n^{\log_2^4}$$

$$\Rightarrow n + 2n + 4n + 8n + \cdots + n^2$$

$$\Rightarrow n \sum_{i=0}^{\log_2^n - 1} (2)^i + n^2$$

$$\Rightarrow n\left[\frac{2^{\log_2^n - 1 + 1} - 1}{2 - 1}\right] + n^2$$

$$\Rightarrow n\left[2^{\log_2^n} - 1\right] + n^2$$

$$\Rightarrow n(n-1) + n^2$$

$$\Rightarrow n^2 - n^2 + n^2$$

$$\Rightarrow 2n^2 * n$$

Then,

$$T(n) = 2n^2 - n$$

$$\boxed{T(n) = \Theta(n^2)}$$

Q. Solve the following Recurrance using Iteration Method

$$T(n) = 2T\left(\frac{n}{4}\right) + n$$

Sol. $$T(n) = 2T\left(\frac{n}{4}\right) + n$$

$$= 2\left[2T\left(\frac{n}{16}\right) + \frac{n}{4}\right] + n$$

$$= 4T\left(\frac{n}{16}\right) + \frac{n}{2} + n$$

LectureNotes.in

Dt → 9th Feb 18
Friday

Substitution Method →

The Substitution Method is used to solve the recurrance in following two steps -

1. Guess the solution.
2. Use Mathematical Induction Method to find the constant and so the solution works.

NOTE →

Substitution Method gives the time complexity either in Upper baund or Lower baund

Q. Find the solution of the recurrance using Substitution Method.

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \quad \text{—— (1)}$$

Sol. Let the solution of the recurrance is

$$T(n) = O(n\log n)$$

$$\Rightarrow T(n) \leq c \cdot n\log n \quad \text{where } c > 0 \quad \text{—— (2)}$$

let, it is true for $\frac{n}{2}$

$$\Rightarrow T\left(\frac{n}{2}\right) \leq c \cdot \frac{n}{2} \log \frac{n}{2} \quad \text{—— (3)}$$

the eqn (1) becomes

$$\Rightarrow T(n) \leq 2 \cdot c \frac{n}{2} \log \frac{n}{2} + n$$

$$\Rightarrow T(n) \leq cn \log \frac{n}{2} + n$$

$$\Rightarrow T(n) \leq c \cdot n(\log n - \log 2) + n$$

$\Rightarrow T(n) \leq cn\log n - cn + n$

$\Rightarrow T(n) \leq cn\log n$ where $-cn+n \leq 0$

$n(-c+1) \leq 0$

$-c+1 \leq 0$

$c \geq 1$

Therefore,

$T(n) \leq c \cdot n\log n$ where $c \geq 1$.

So,

$$\boxed{T(n) = O(n\log n)}$$

Q. Solve the recurrance using Substitution Method.

1. $T(n) = T\left(\frac{n}{2}\right) + 1$

2. $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Hint: $T(n) \leq 2T\left(\frac{n}{2}\right) + cn$ where $c > 0$

Q. Solve the recurrance using Recursive Tree Method

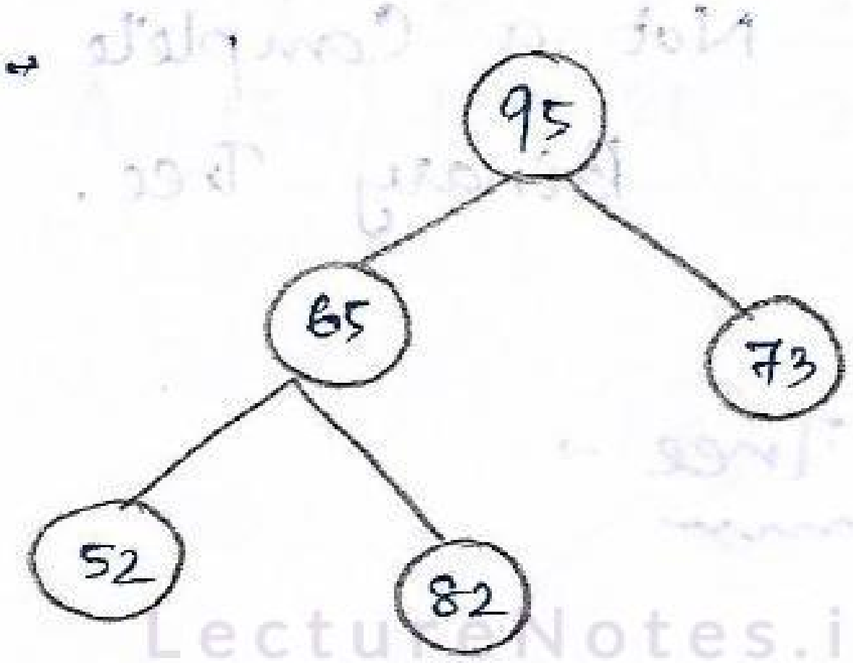
3. $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$

LectureNotes.in

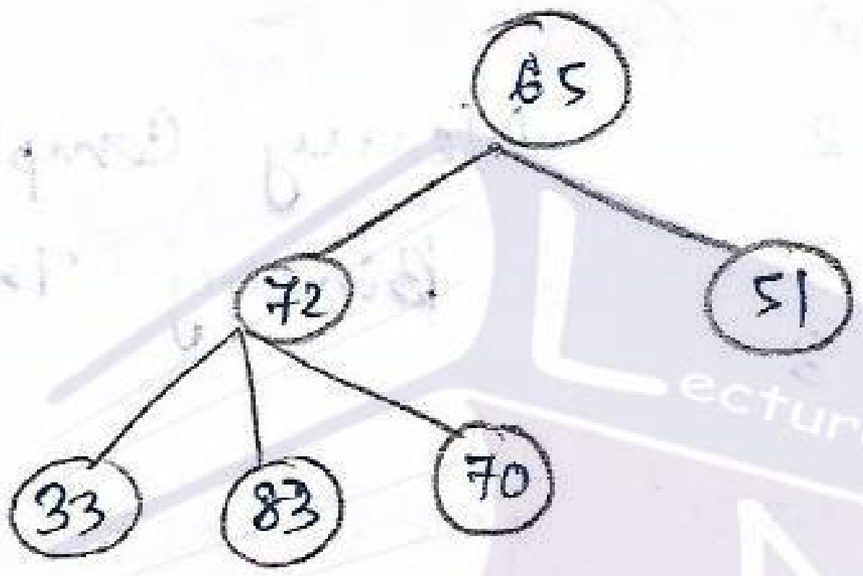
Binary Tree →

It is a tree having at most two child nodes.

Example →



(Binary Tree)

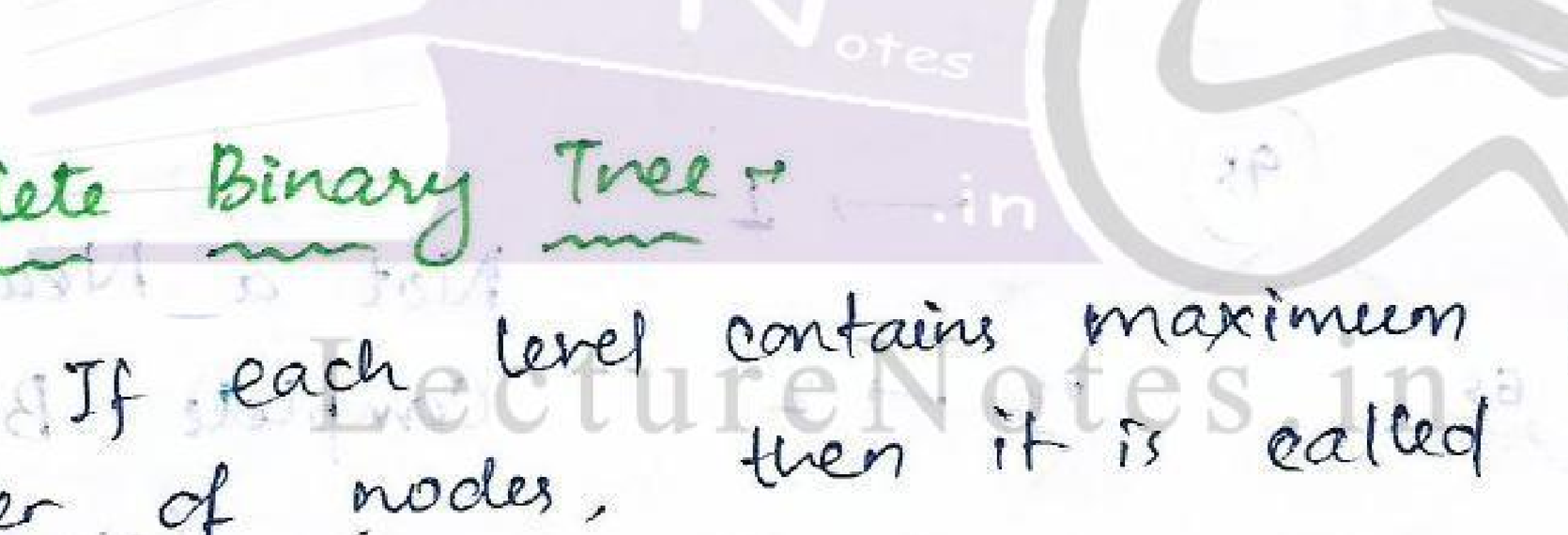


(Not a Binary Tree)

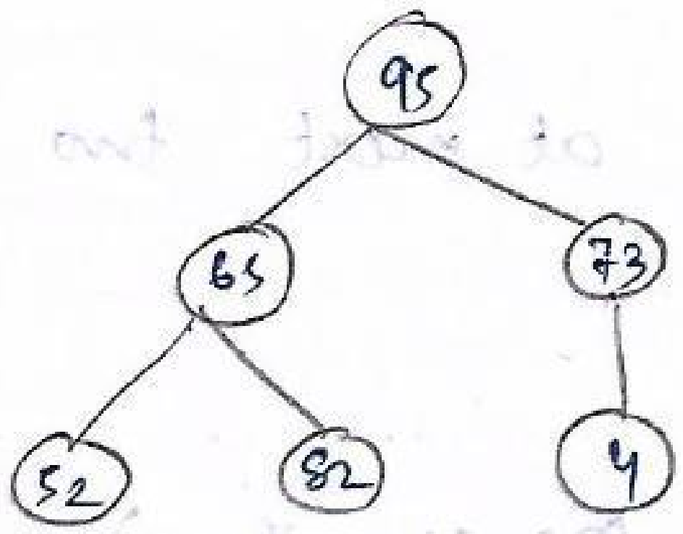
Complete Binary Tree →

If each level contains maximum possible number of nodes, then it is called Complete Binary Tree.

Example →



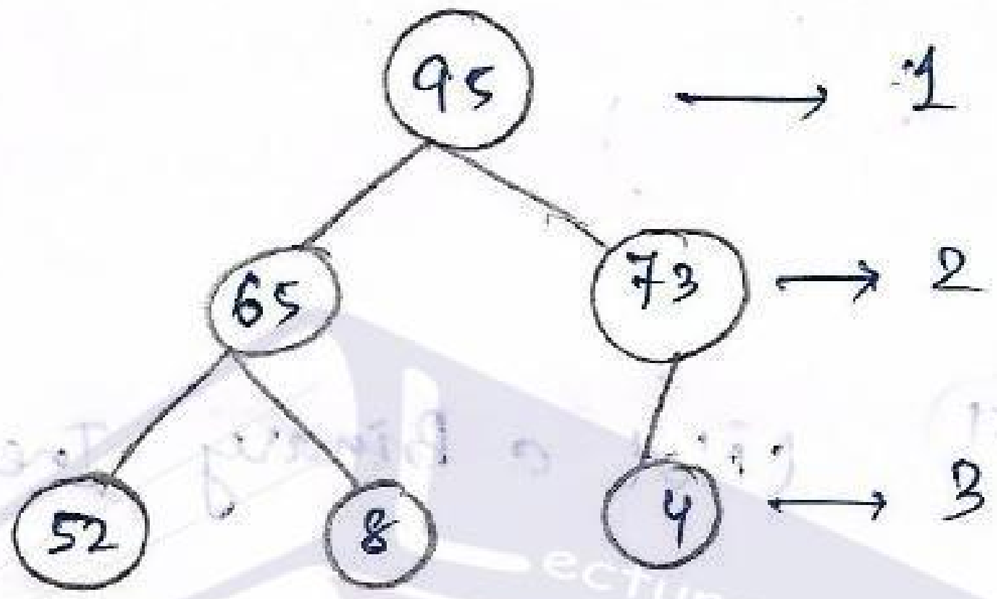
(Complete Binary Tree)



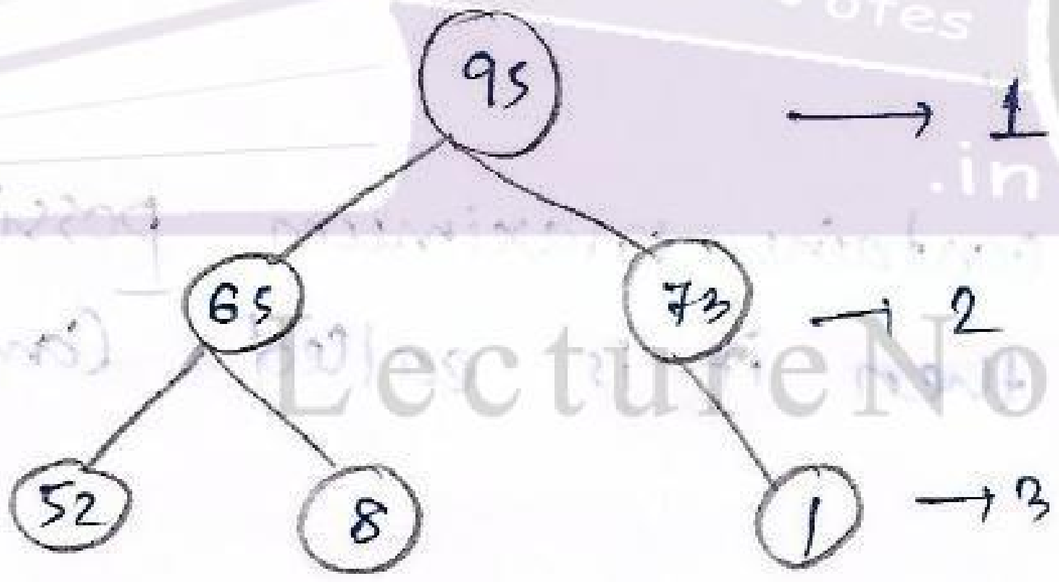
Not a Complete Binary Tree.

Nearly Complete Binary Tree →

Example →



Nearly Complete Binary Tree

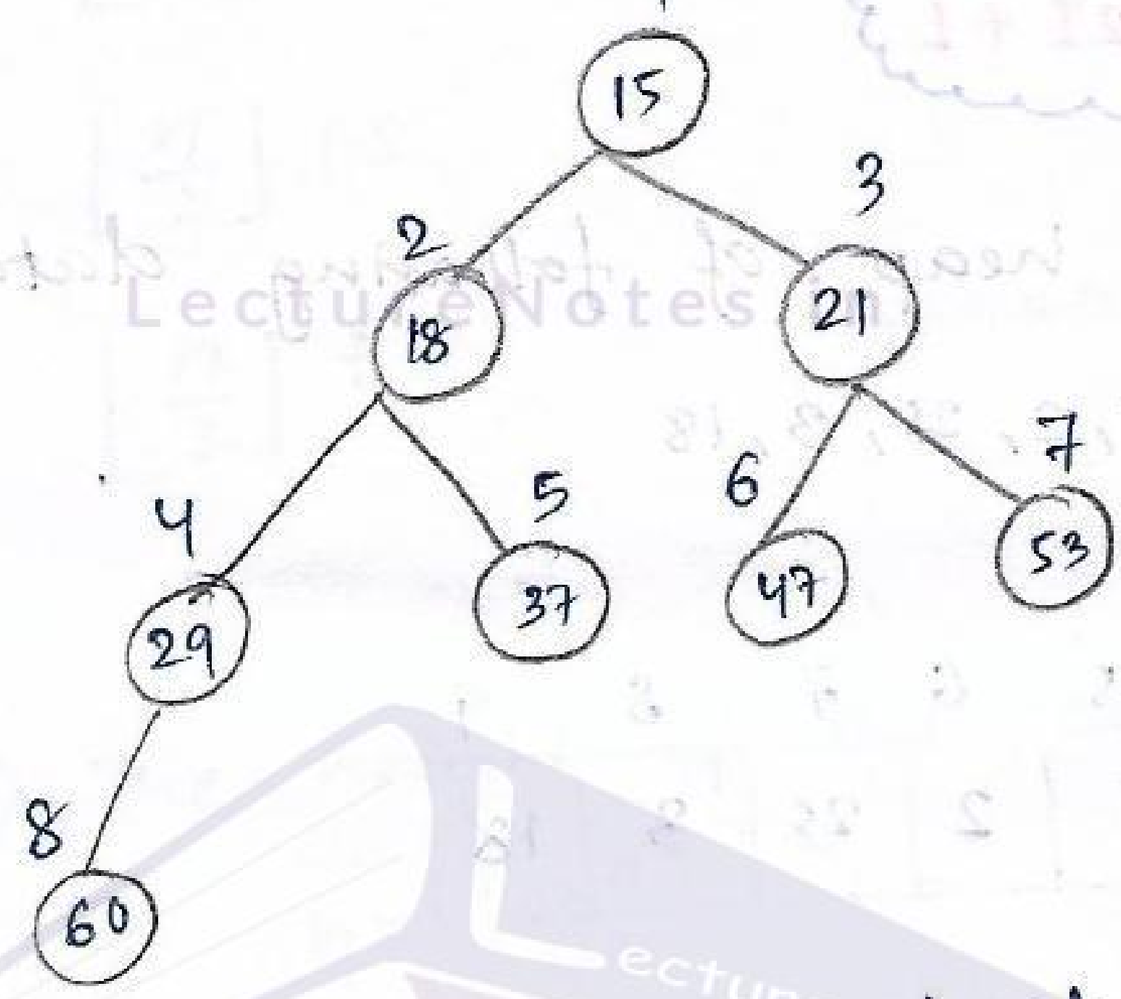


Not a Nearly Complete Binary Tree.

Each level filled with maximum possible number of nodes except last level and the last level should fill from left to right.

Binary Heap →

	1	2	3	4	5	6	7	8
A	15	18	21	29	37	47	53	60



Binary Heap Data structure is an array which can be represented as Nearly Complete Binary Tree

LENGTH(A) →

No. of elements in the array.

HEAP_SIZE(A) →

No. of elements in the binary tree.

$$HEAP_SIZE(A) \leq LENGTH(A)$$

LEFT(I) →

Index of left child of I.

Return 2×I

LectureNotes.in

Return 2×I

RIGHT (I) →

Index of Right Child I

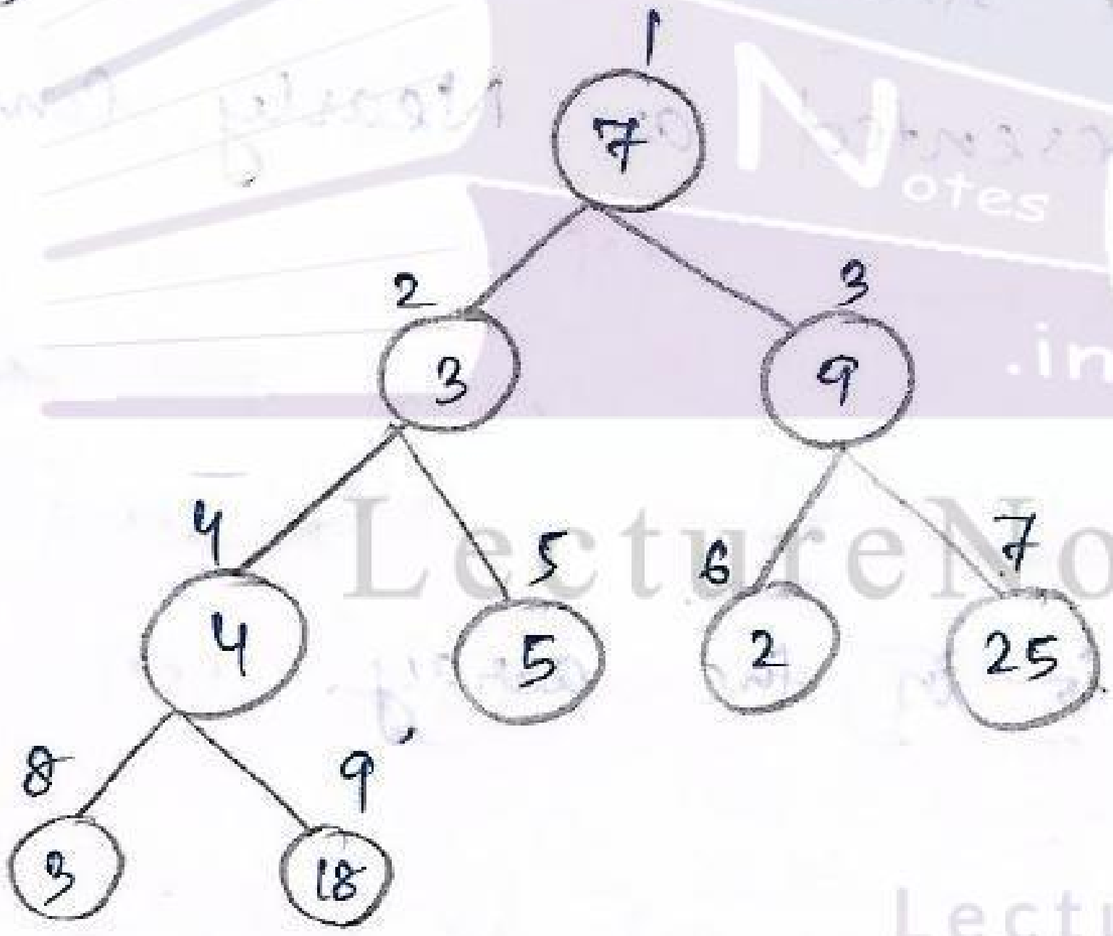
Return 2I + 1

Q. Construct a binary heap of following datas -

7, 3, 9, 4, 5, 2, 25, 3, 18

Sol.

1	2	3	4	5	6	7	8	9
7	3	9	4	5	2	25	3	18



The Child Nodes are at Index 5, 6, 7, 8, 9.

PARENT (I) →

Return $\lfloor \frac{I}{2} \rfloor$

Properties of Binary Heap:

(1) If the Binary heap contains n elements, then the Index of child node are

$\lfloor \frac{n}{2} \rfloor + 1$

$\lfloor \frac{n}{2} \rfloor + 2$

$\lfloor \frac{n}{2} \rfloor + 3$

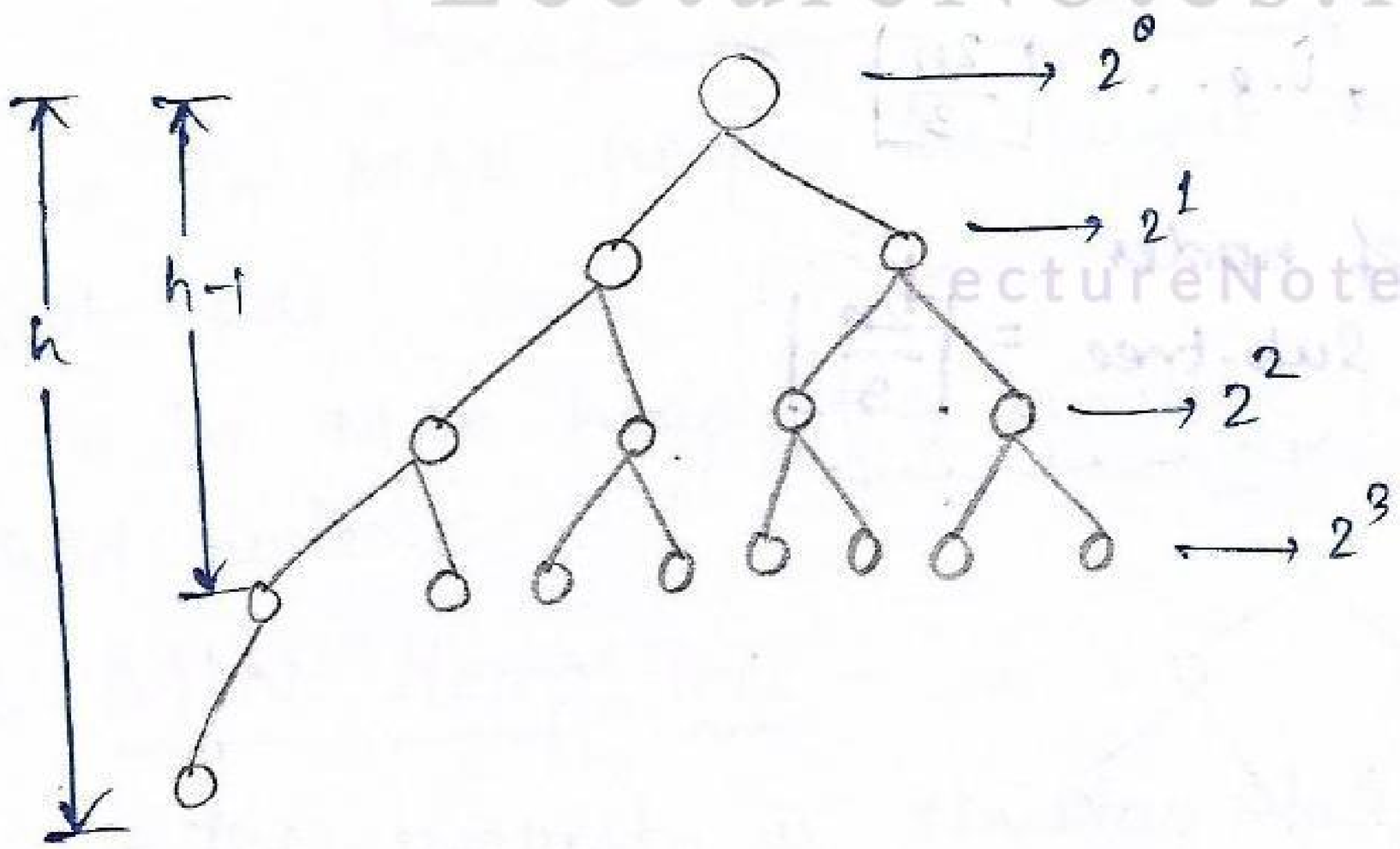
⋮

(2) The height of the heap is with 'n' elements is

$h = \log_2^n$

(3) The Minimum no. of nodes possible in a heap of height, h is

Min no. of nodes $= 2^h$



LectureNotes.in

$$2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{h-1} + 1$$

$$\sum_{i=0}^{h-1} (2)^i + 1$$

$$= 2^{h-1+1} - 1 + 1$$

$$= 2^h$$

(4) The Maximum number of nodes at height h is

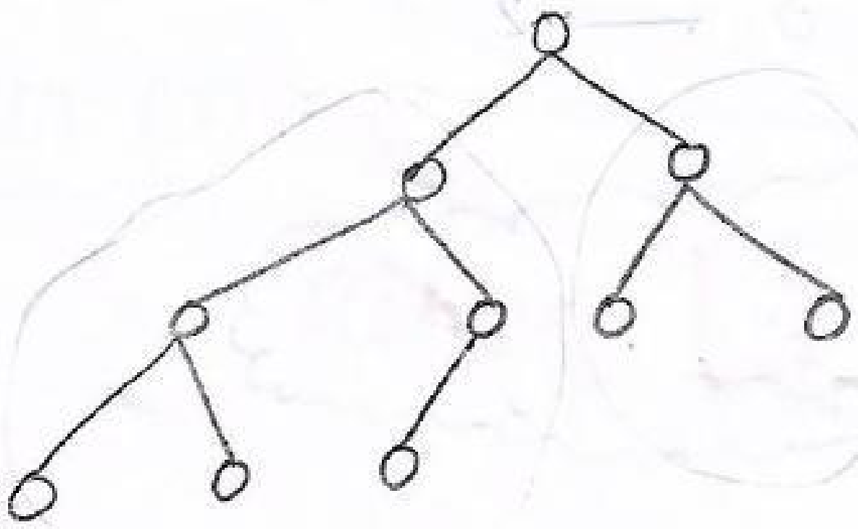
Max no. of nodes = $2^{h+1} - 1$

$$2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^h$$

$$\sum_{i=0}^{h} (2)^i = 2^{h+1} - 1$$

(5) If there are n elements in a tree, then the children of tree has the size at most floor of $\frac{2n}{3}$, i.e., $\left\lfloor \frac{2n}{3} \right\rfloor$

Max. no. of nodes of Sub-tree = $\left\lfloor \frac{2n}{3} \right\rfloor$

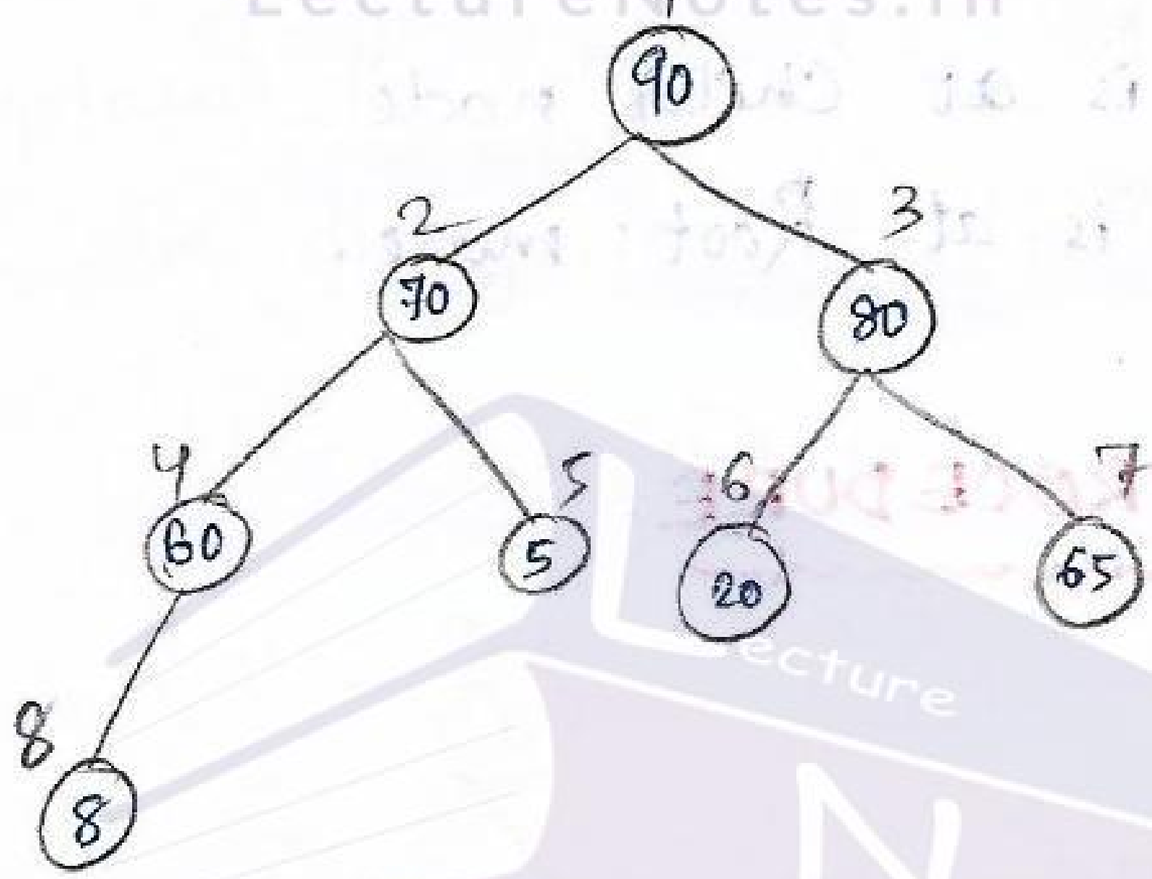


Dt→ 9th Feb 18
Friday

Types of Heap Tree

1. Max Heap Tree
2. Min Heap Tree

1. MAx Heap Tree →



* In MAX Heap, the property is that it is nearly complete Binary tree. in which

$$A[PARENT(I)] \geq A[I]$$

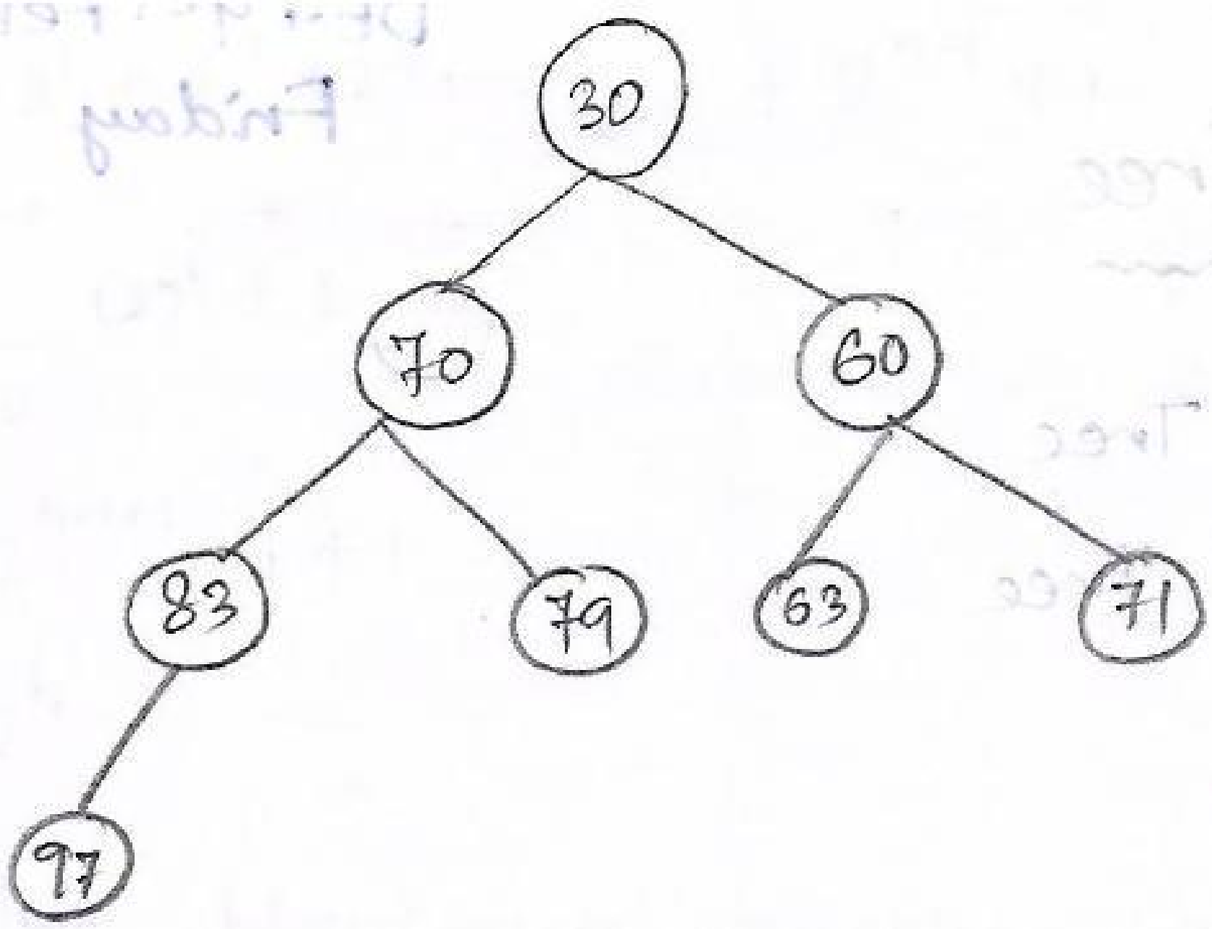
* In MAX heap, the largest node is at Root node.

* In MAX heap, the smallest node is at Child node.

2. MIN Heap Tree →

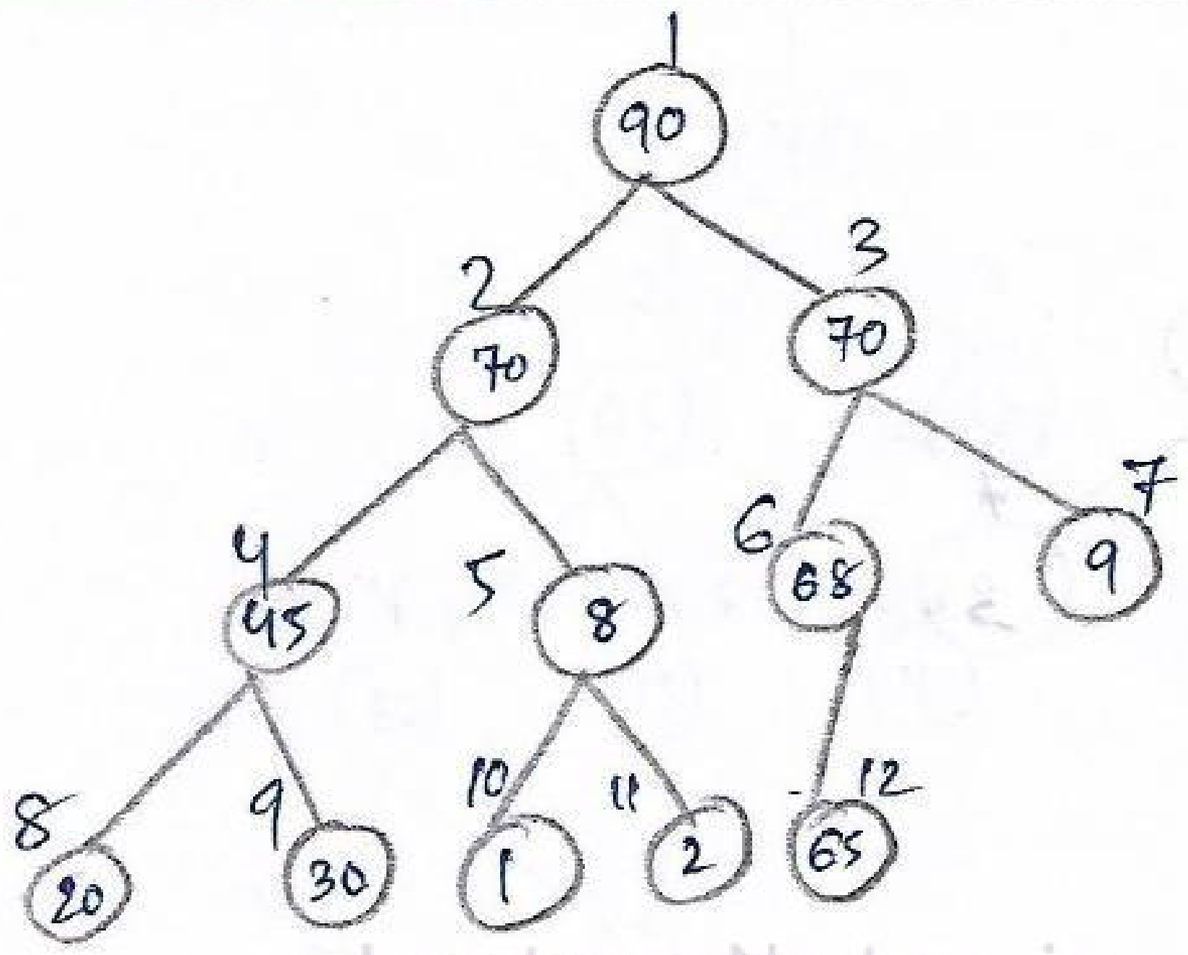
* the property is similar to Nearly Complete Binary tree in which.

$$A[PARENT(I)] \leq A[I]$$



* The largest node is at Child node.
* The smallest node is at Root node.

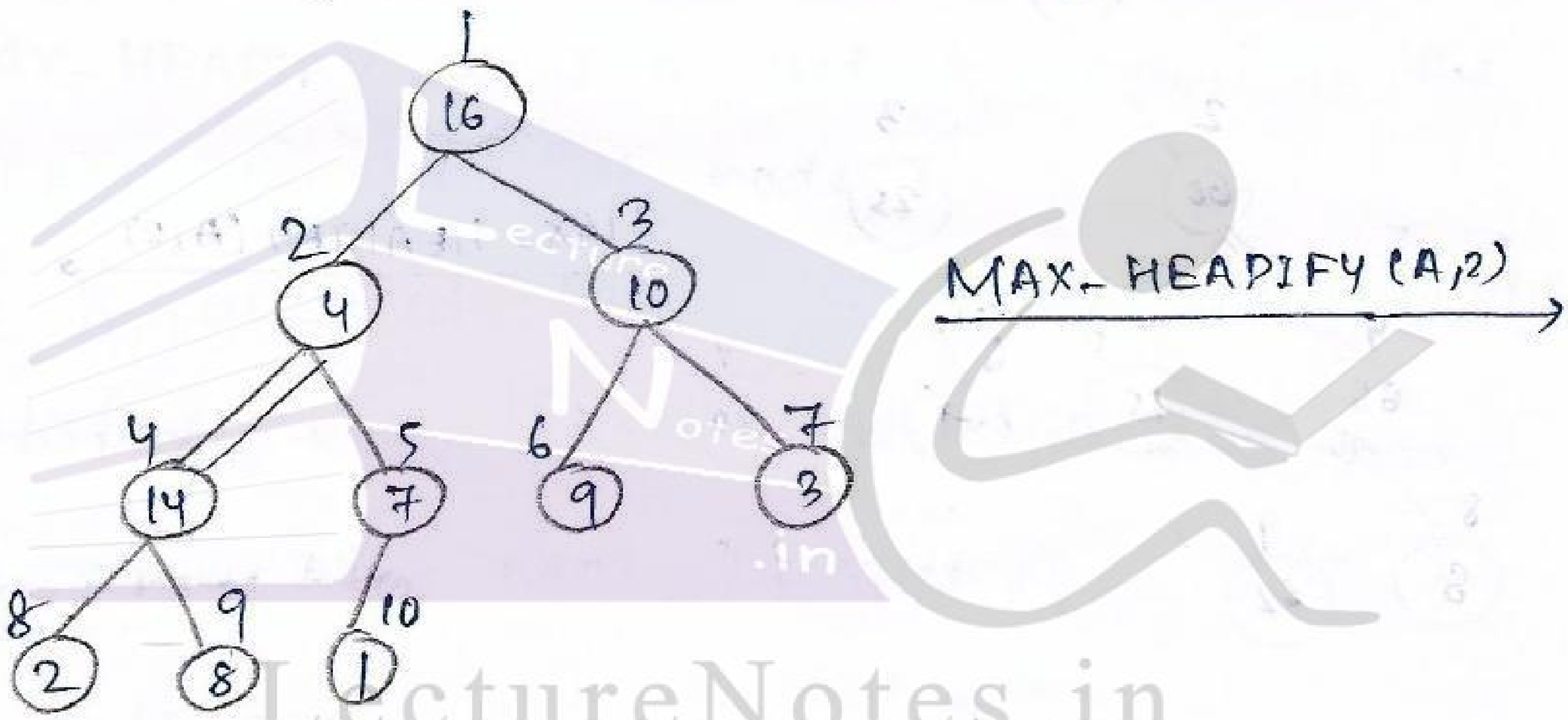
MAX_HEAPIFY PROCEDURE



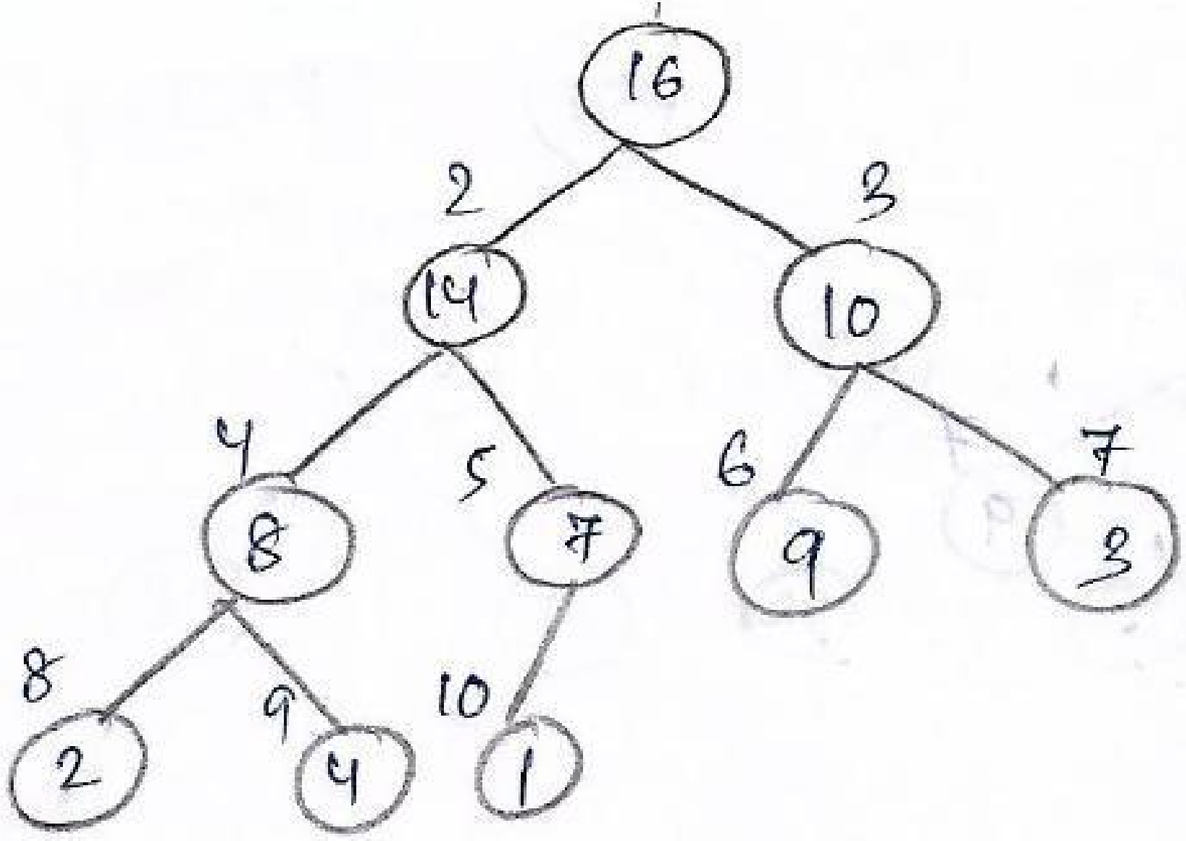
LectureNotes.in

Q. Explain the operation of MAX_HEAPIFY (A, 2). If the array is 16, 4, 10, 14, 7, 9, 3, 2, 8, 1

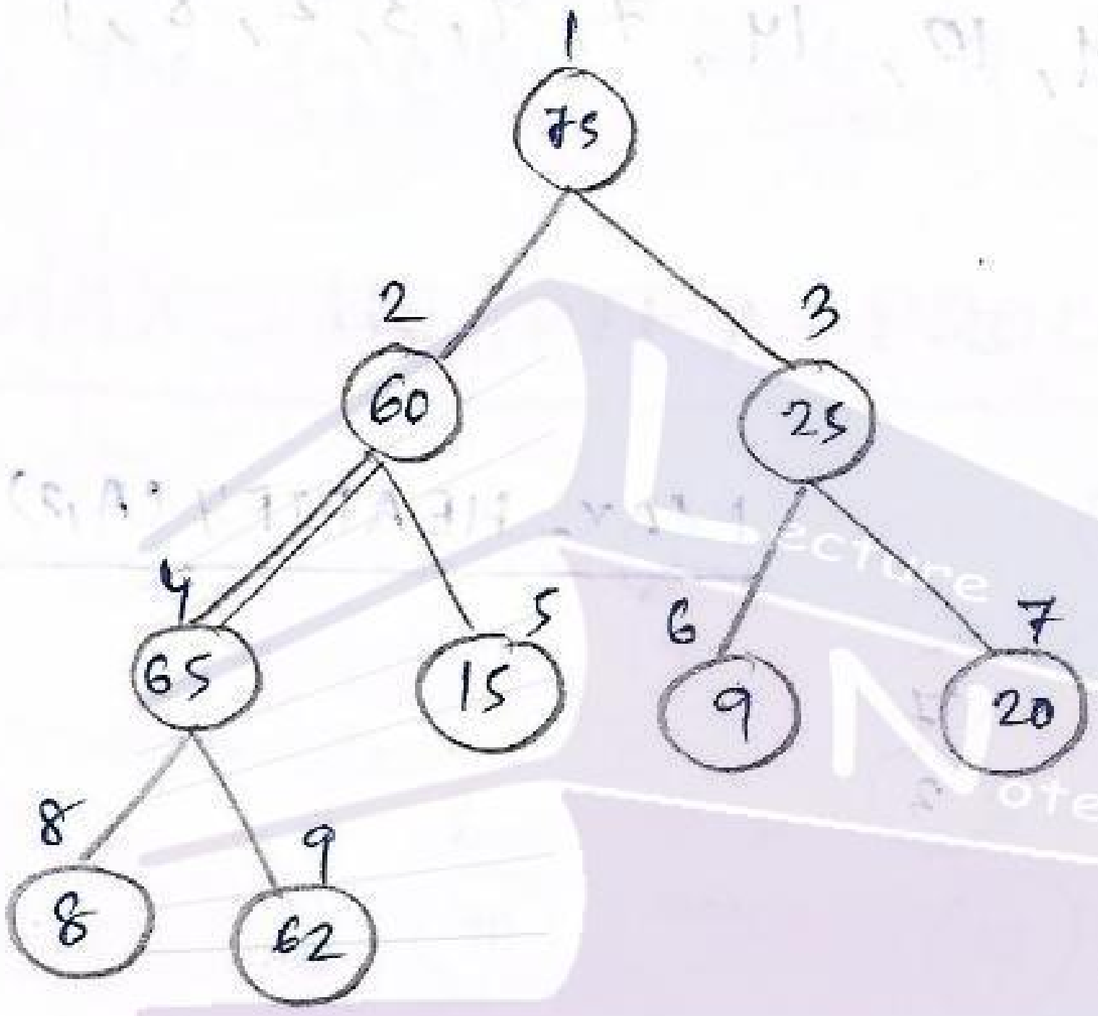
Sol.



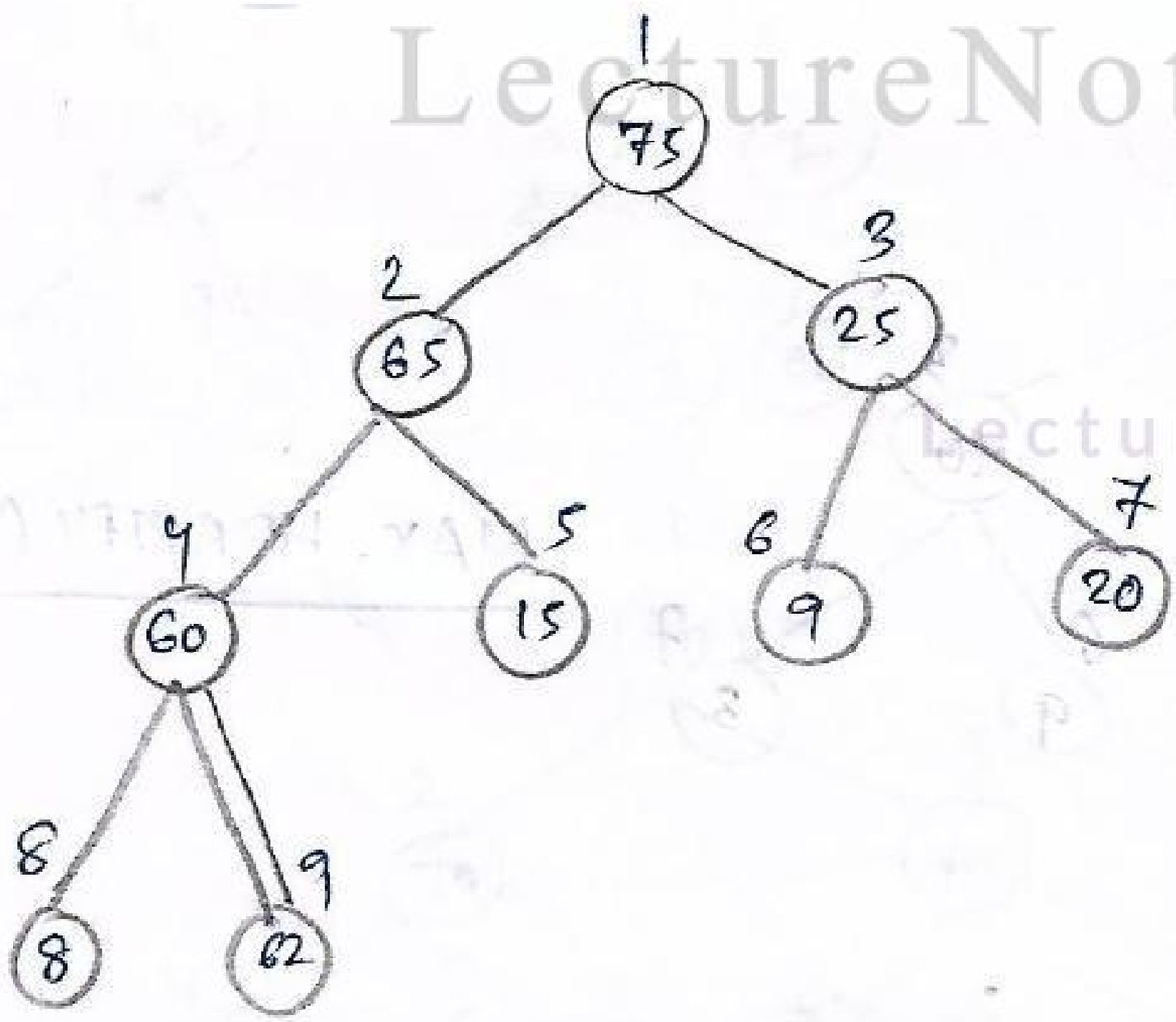
LectureNotes.in



Dt - 10th Feb 18
Saturday

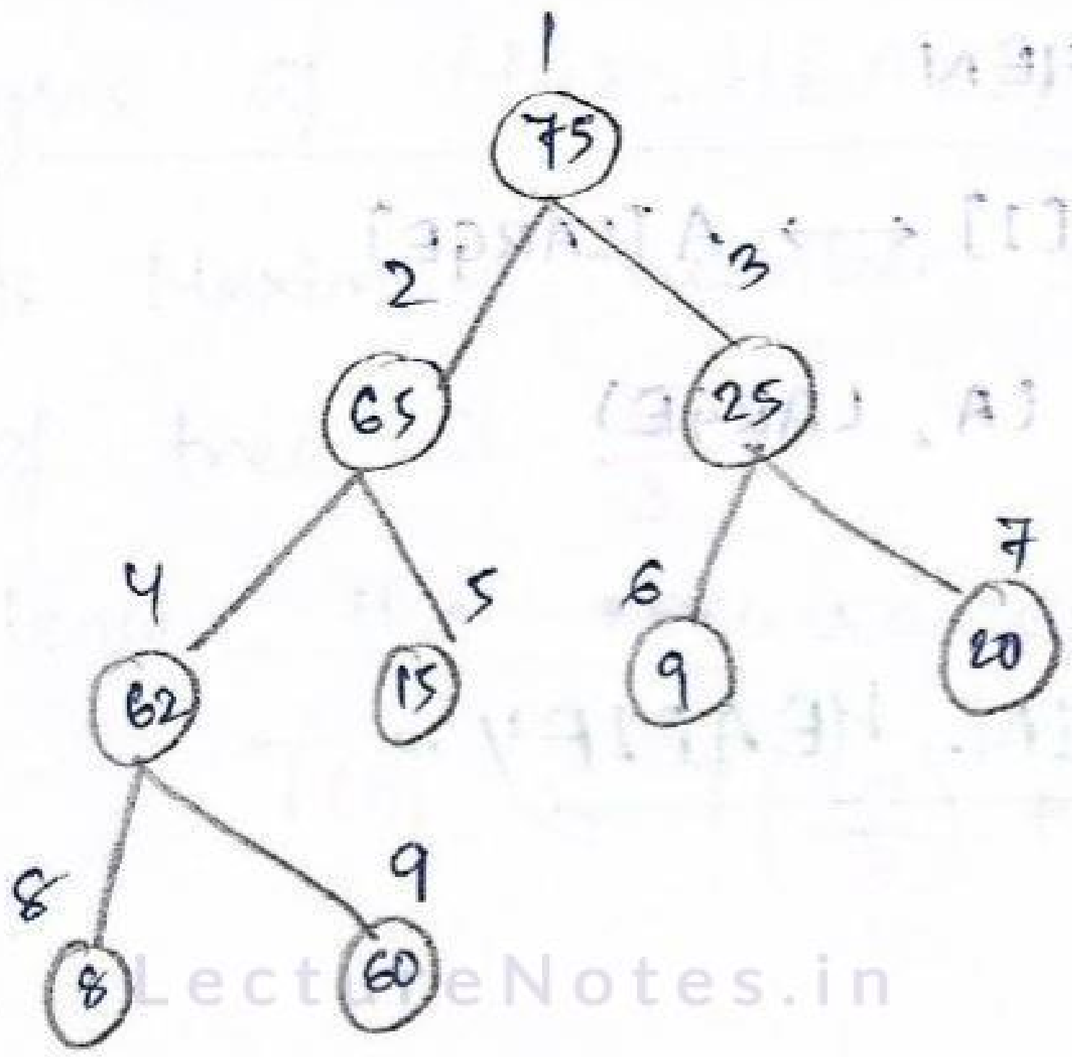


MAX_HEAPIFY (A, 2)



MAX_HEAPIFY (A, 4)

LectureNotes.in



MAX_HEAPIFY (A, I) →

* MAX_HEAPIFY at I is used to maintain the MAX_HEAP property at node I.

* It is assumed that the MAX_HEAP property is satisfied at LEFT(I) and RIGHT(I).

Algorithm → (for MAX_HEADIFY)

1. L ← LEFT(I)
2. R ← RIGHT(I)
3. IF A[L] > A[I] AND L ≤ HEAP_SIZE(A)
4. LARGE ← L

   ELSE
5. LARGE ← I

   [END IF]
6. IF A[R] > A[LARGE] AND R ≤ HEAP_SIZE(A)
7. LARGE ← R

8. IF LARGE ≠ I THEN

9. EXCHANGE A[I] ⟷ A[LARGE]

MAX_HEAPIFY (A, LARGE)

[END IF]

Algorithm for MIN_HEAPIFY :-

1. L ← LEFT(I)
2. R ← RIGHT(I)
3. IF A[L] < A[I] AND L ≤ HEAP_SIZE(A)
4. LARGE ← L

ELSE

5. LARGE ← I

[END IF]

6. IF A[R] < A[LARGE] AND R ≤ HEAP_SIZE(A)
7. LARGE ← R

[END IF]

8. IF LARGE ≠ I THEN
9. EXCHANGE A[I] ⟷ A[LARGE]

MIN_HEAPIFY (A, LARGE)

[END IF]

LectureNotes.in

Analysis of MAX_HEAPIFY Procedure →

→ As Maximum number of nodes in either side of tree is $\frac{2n}{3}$.

→ Hence, the recurrance is

$$T(n) \leq T\left(\frac{2n}{3}\right) + 1$$

Then,

$$T(n) \leq T\left(\frac{n}{3/2}\right) + 1$$

Here, $a=1$, $b=\frac{3}{2}$, $f(n)=1$

$$n^{\log_{3/2}^{1}} = n^0 = 1$$

According to Master Theorem 2, the solution

$$T(n) = \theta(n^{\log_b^a} \log n)$$

$$T(n) = \theta(\log n)$$

But,

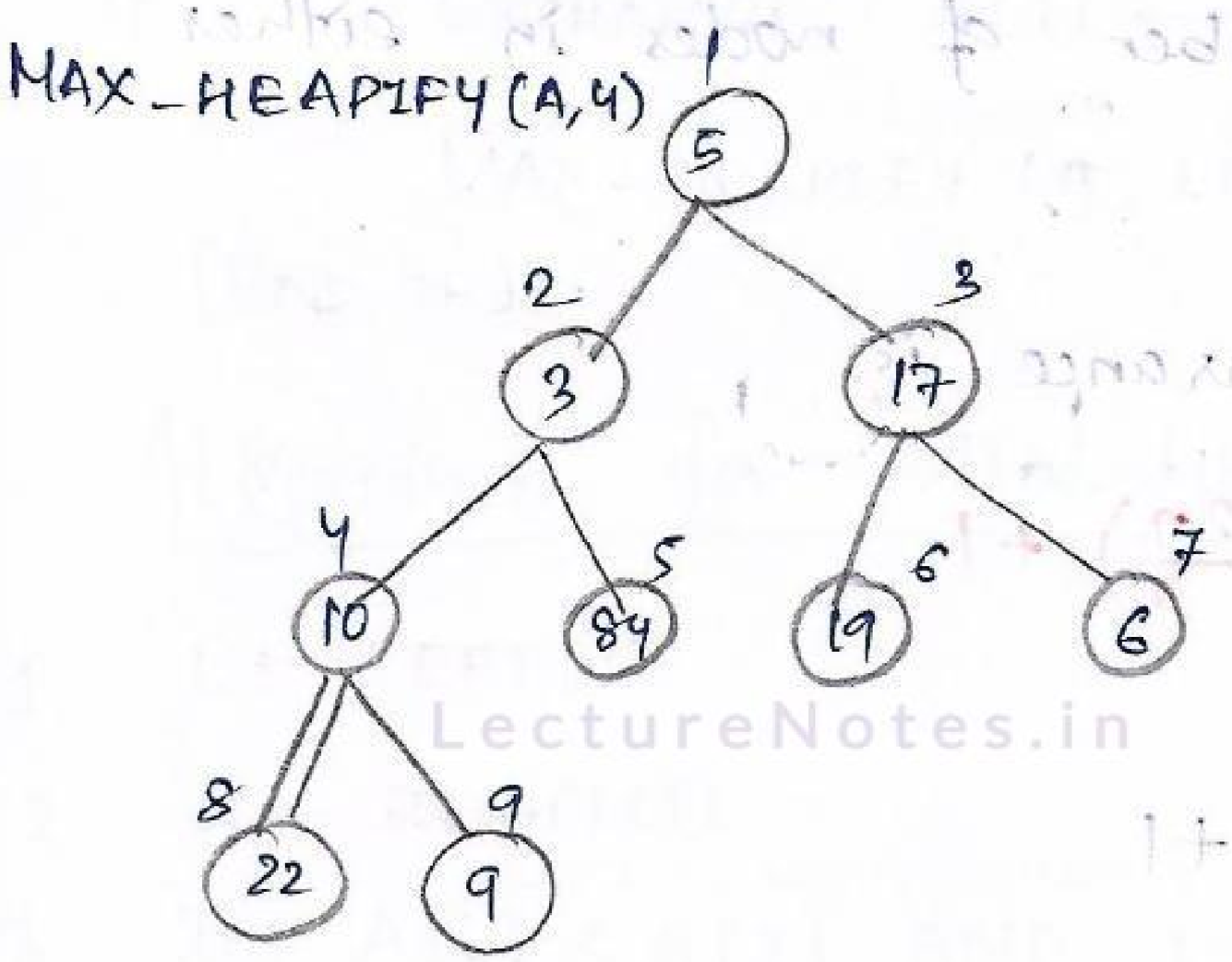
$$T(n) = O(\log n)$$

Q. Explain the operation of BUILD_MAX_HEAP on the array A with elements

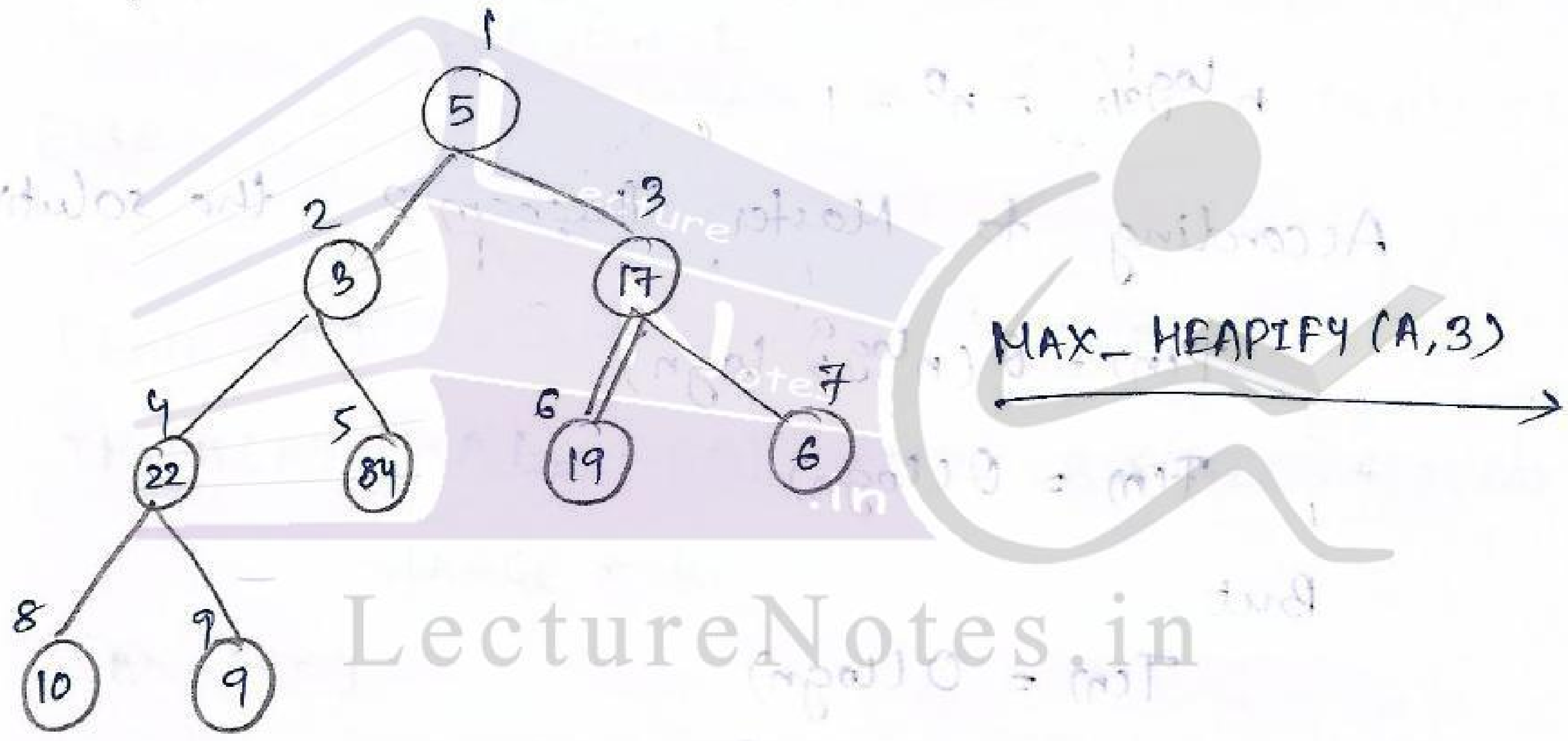
A <5, 3, 17, 10, 84, 19, 6, 22, 9>

P.T.O.

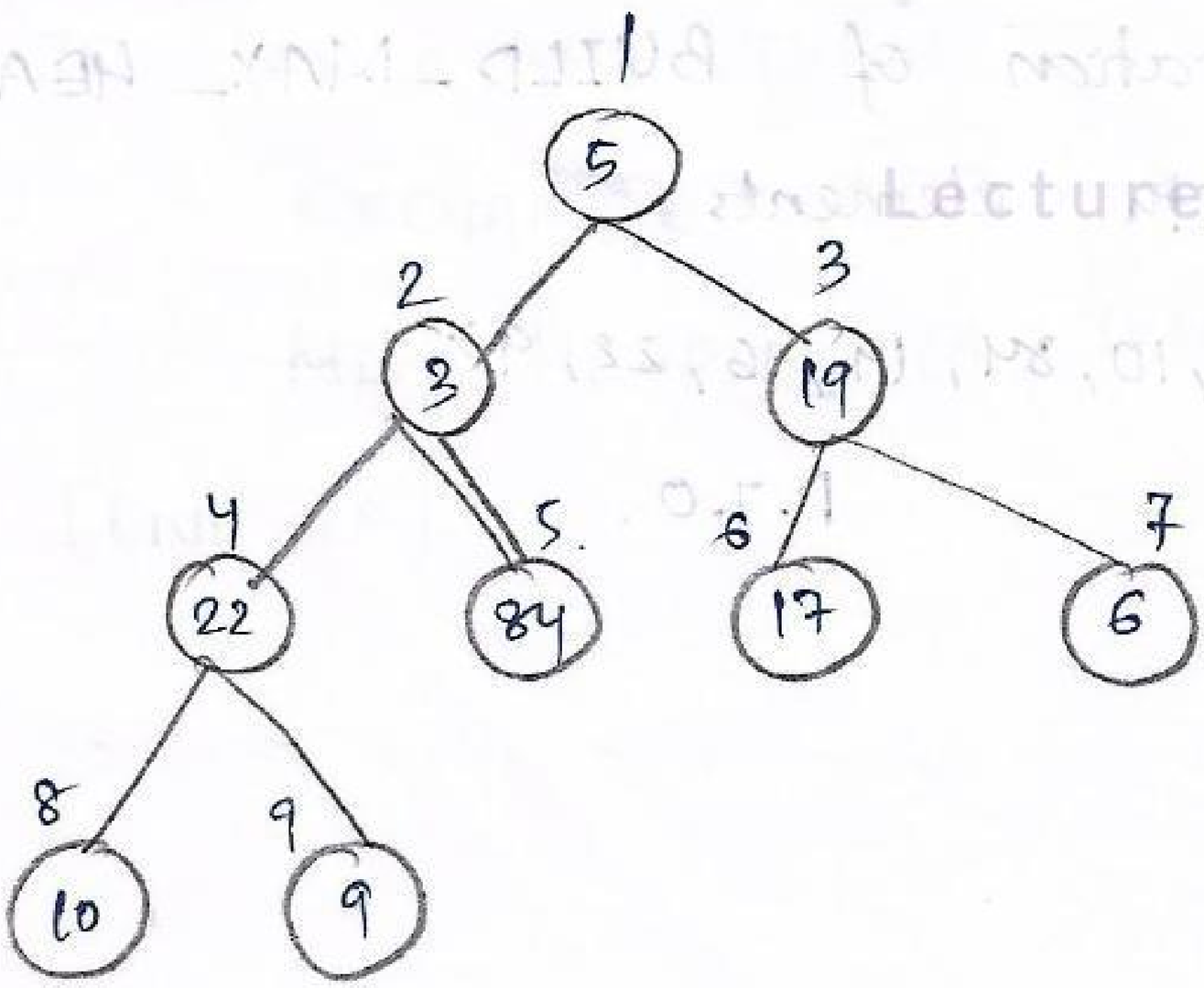
BUILD_MAX_HEAP



MAX_HEAPIFY(A,4)



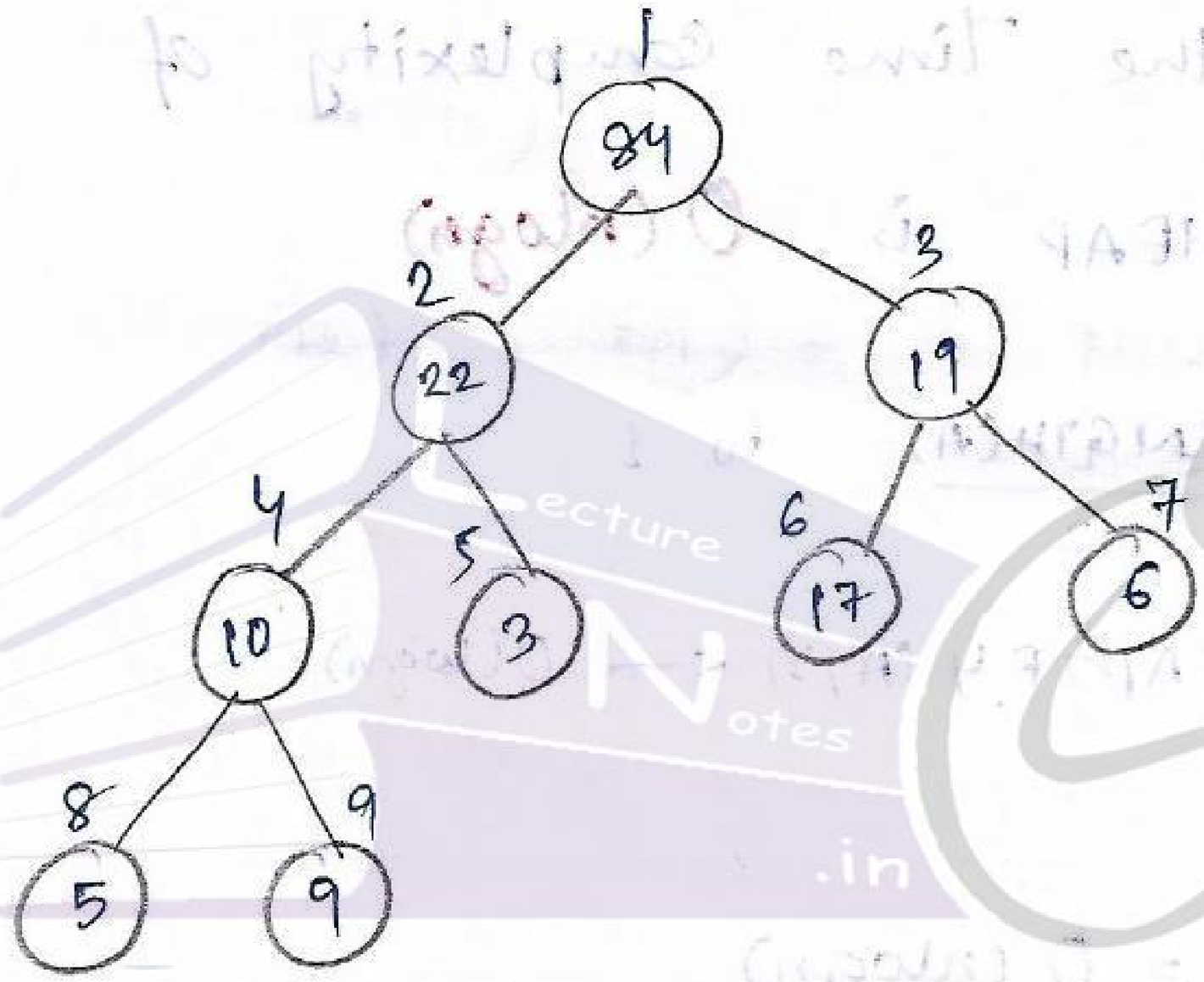
MAX_HEAPIFY(A,3)



MAX_HEAPIFY(A,2)

LectureNotes.in

MAX-
HEAPIFY (A, 1)



BUILD_MAX_HEAP procedure is used to construct MAX_HEAP Tree.

Algorithm of BUILD_MAX_HEAP →

BUILD_MAX_HEAP (A)

1. FOR I ← $\frac{\text{LENGTH (A)}}{2}$. to 1
2. MAX_HEAPIFY (A, I)

LectureNotes.in

Analysis of BUILD_MAX_HEAP →

The Time Complexity of MAX_HEAPIFY(A, I) is $O(\log n)$

This statement 2 will execute for $\frac{n}{2}$ times.

Therefore the time complexity of BUILD_MAX_HEAP is $O(n \log n)$

1. FOR I ← $\frac{\text{LENGTH}(A)}{2}$ to 1
2. MAX-HEAPIFY (A, I) ← $O(\log n)$

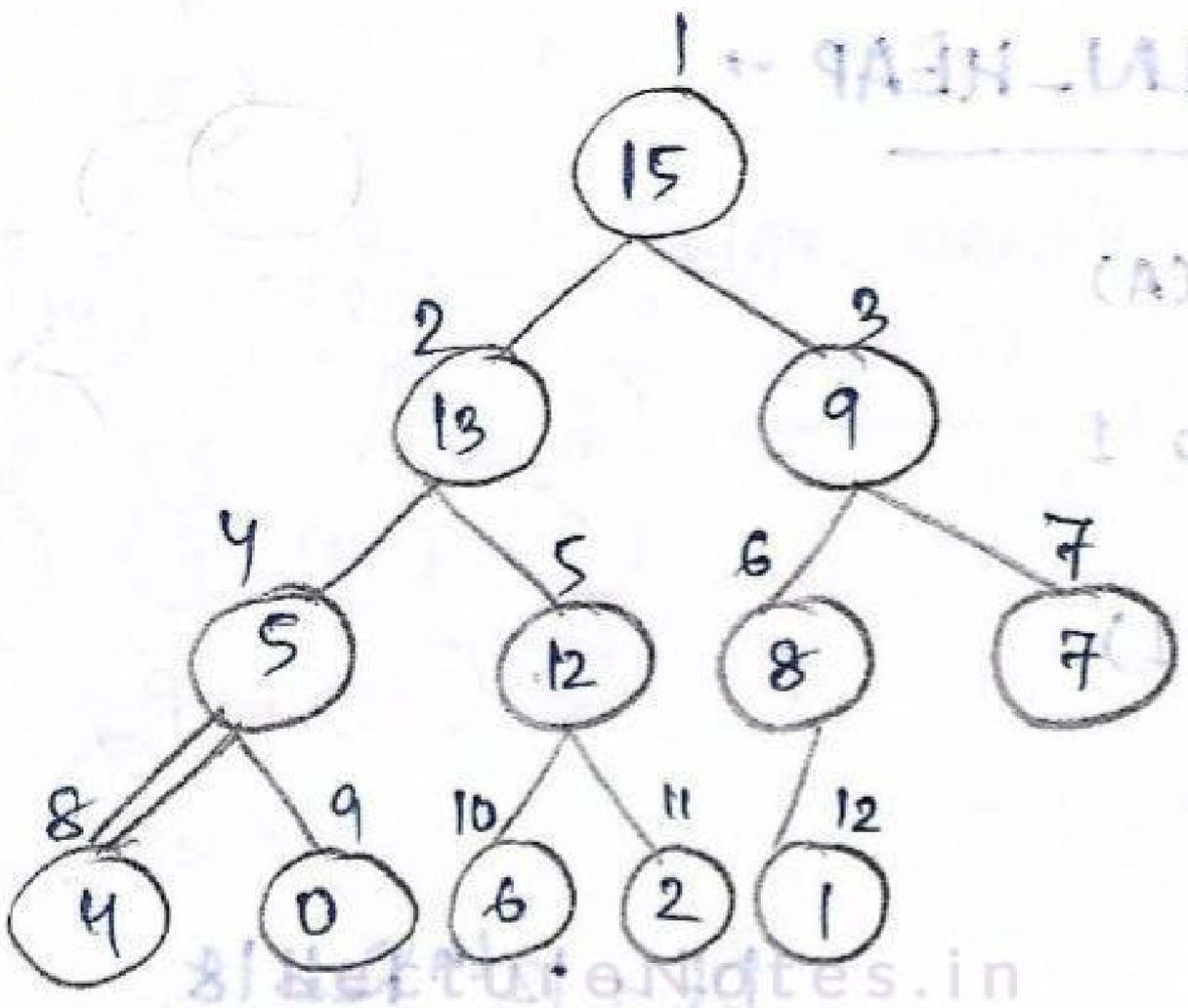
[END FOR]

$T(n) = O(n \log n)$

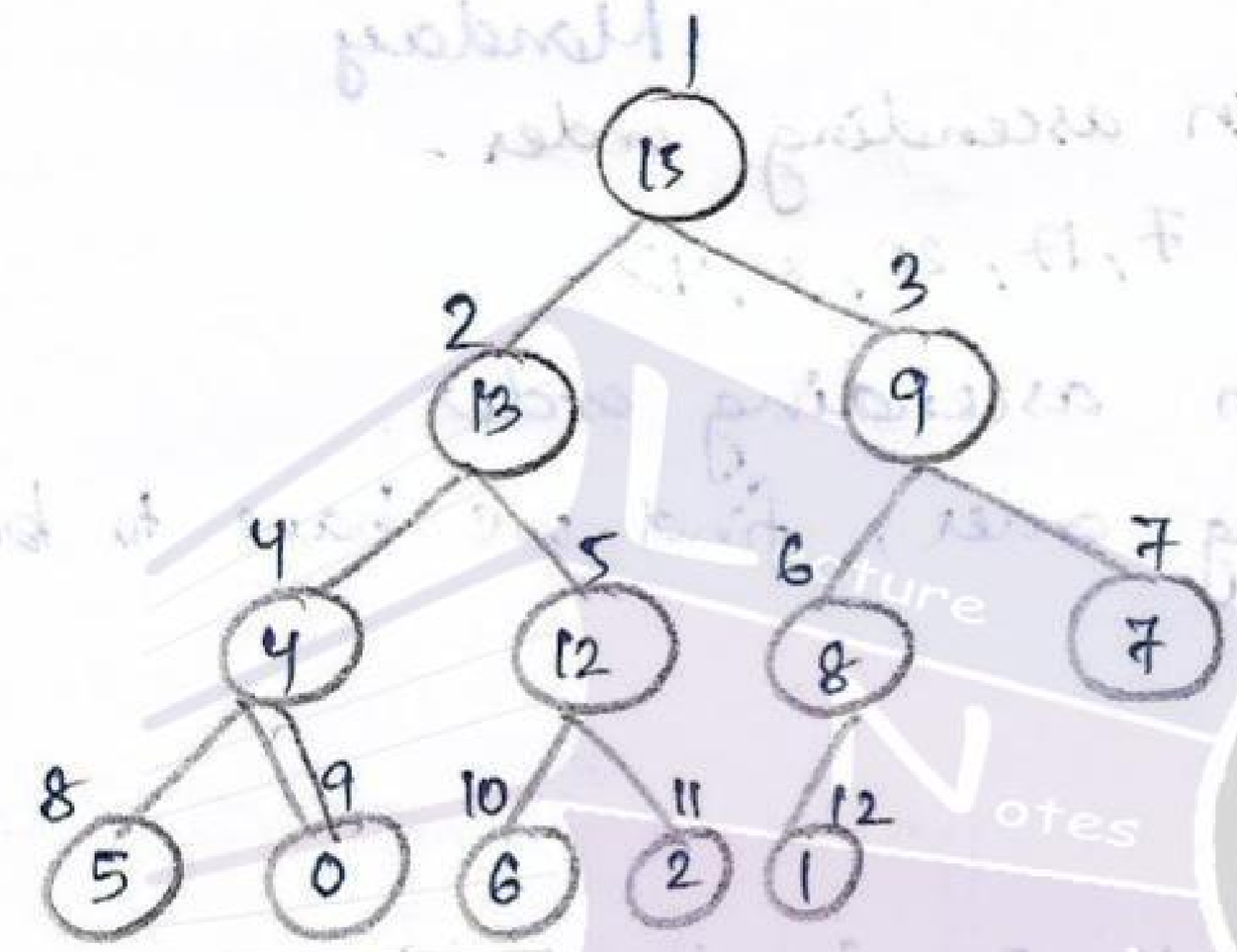
Q. Construct BUILD_MIN_HEAP with the following data - <15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1>

Q. Write the algorithm for MIN. BUILD_MIN_HEAP.

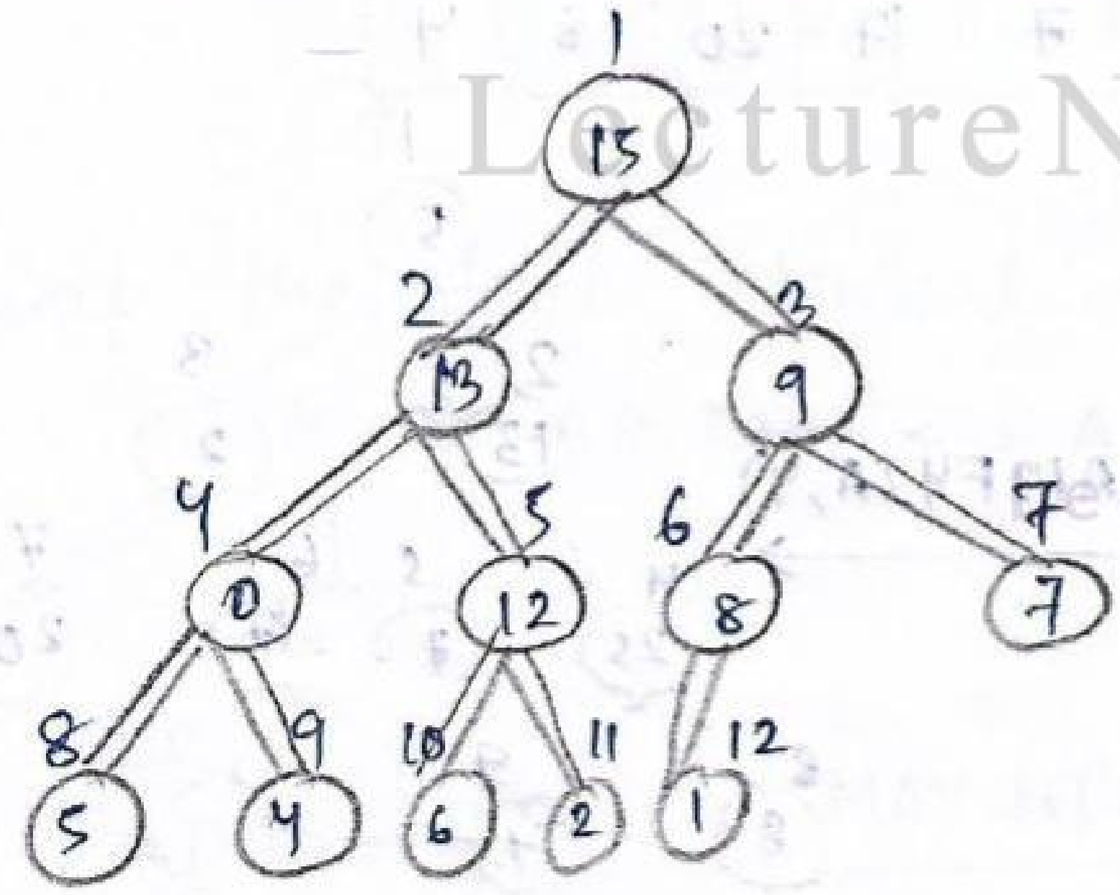
Sol. 1



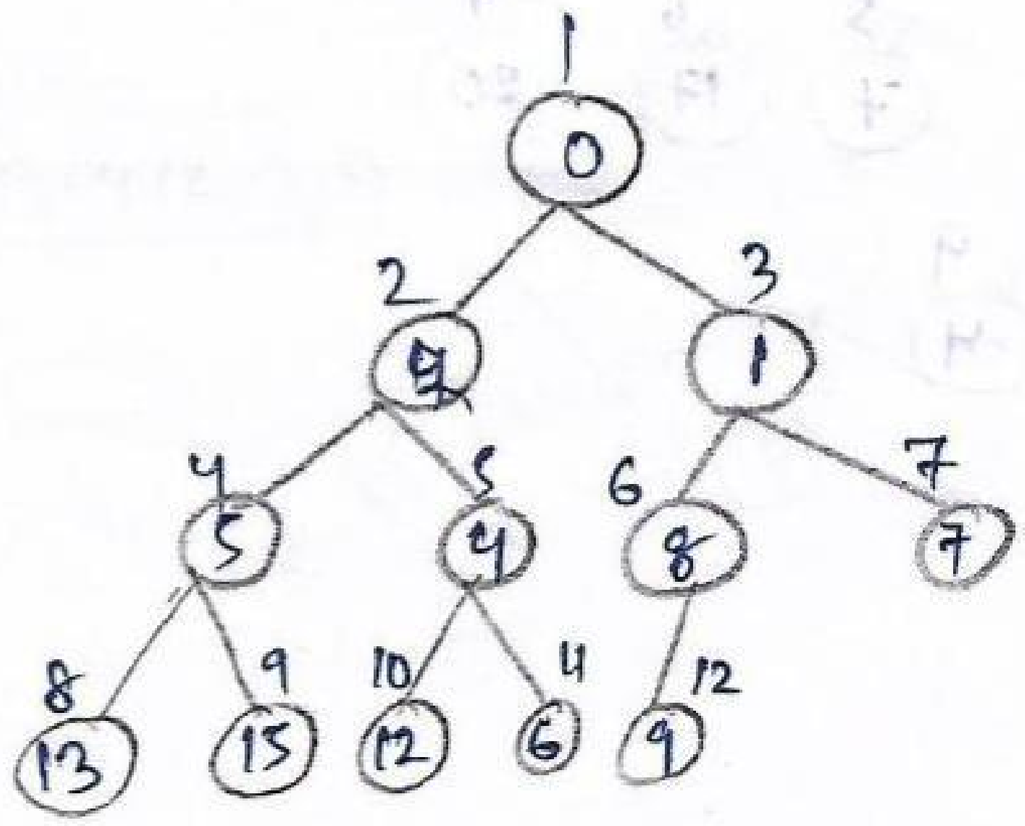
MIN_HEAPIFY(A,4)



MIN_HEAPIFY(A,9)



MIN_HEAPIFY



Algorithm for BUILD_MIN_HEAP →

BUILD_MIN_HEAPIFY (A)

1 FOR I ← $\frac{\text{LENGTH}}{2}$ to 1

2. MIN_HEAPIFY (A, I)

[END FOR]

Dt → 12th Feb 18
Monday

Q. Explain the Heap Sort in ascending order.

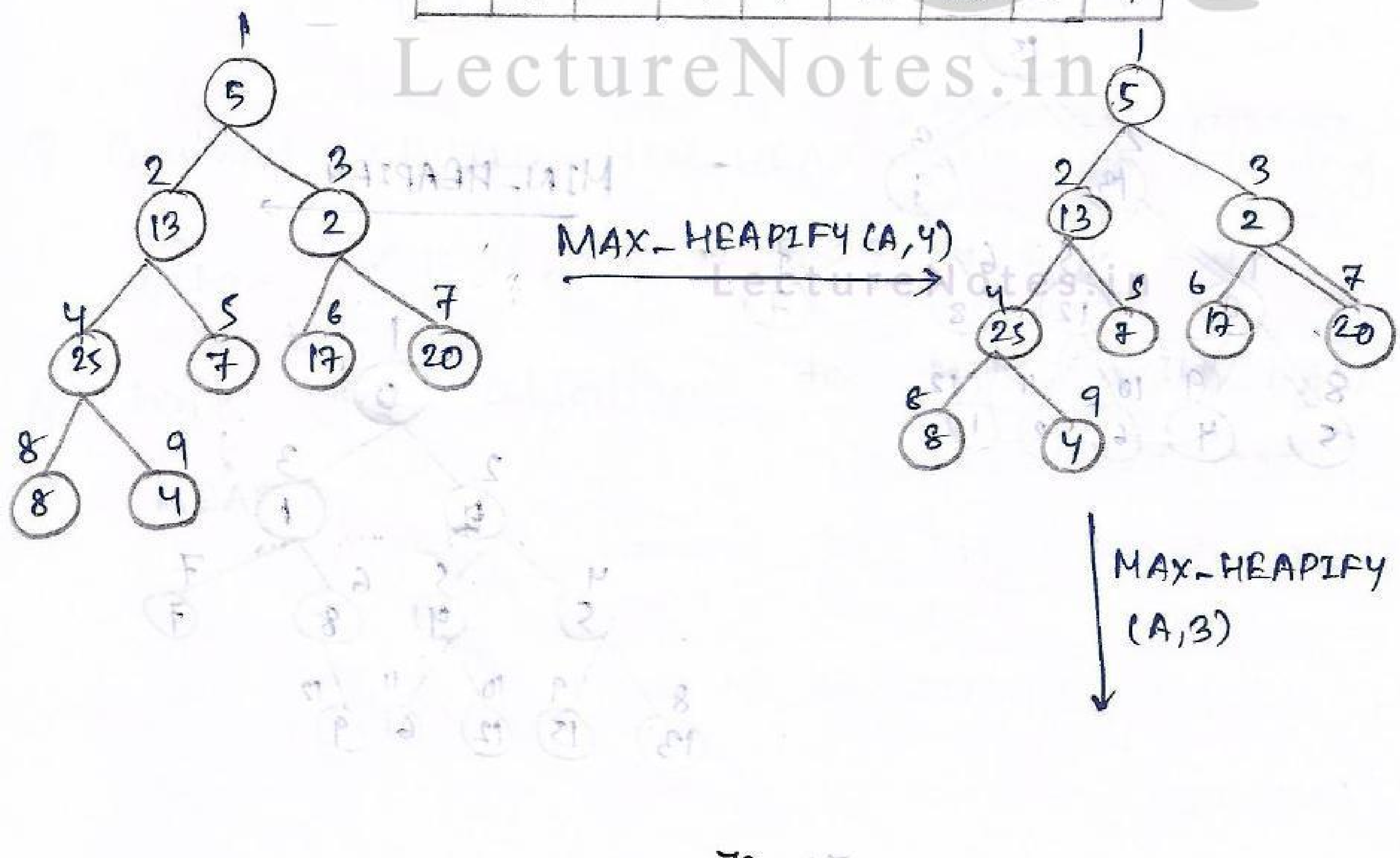
A = < 5, 13, 2, 25, 7, 17, 20, 8, 4>

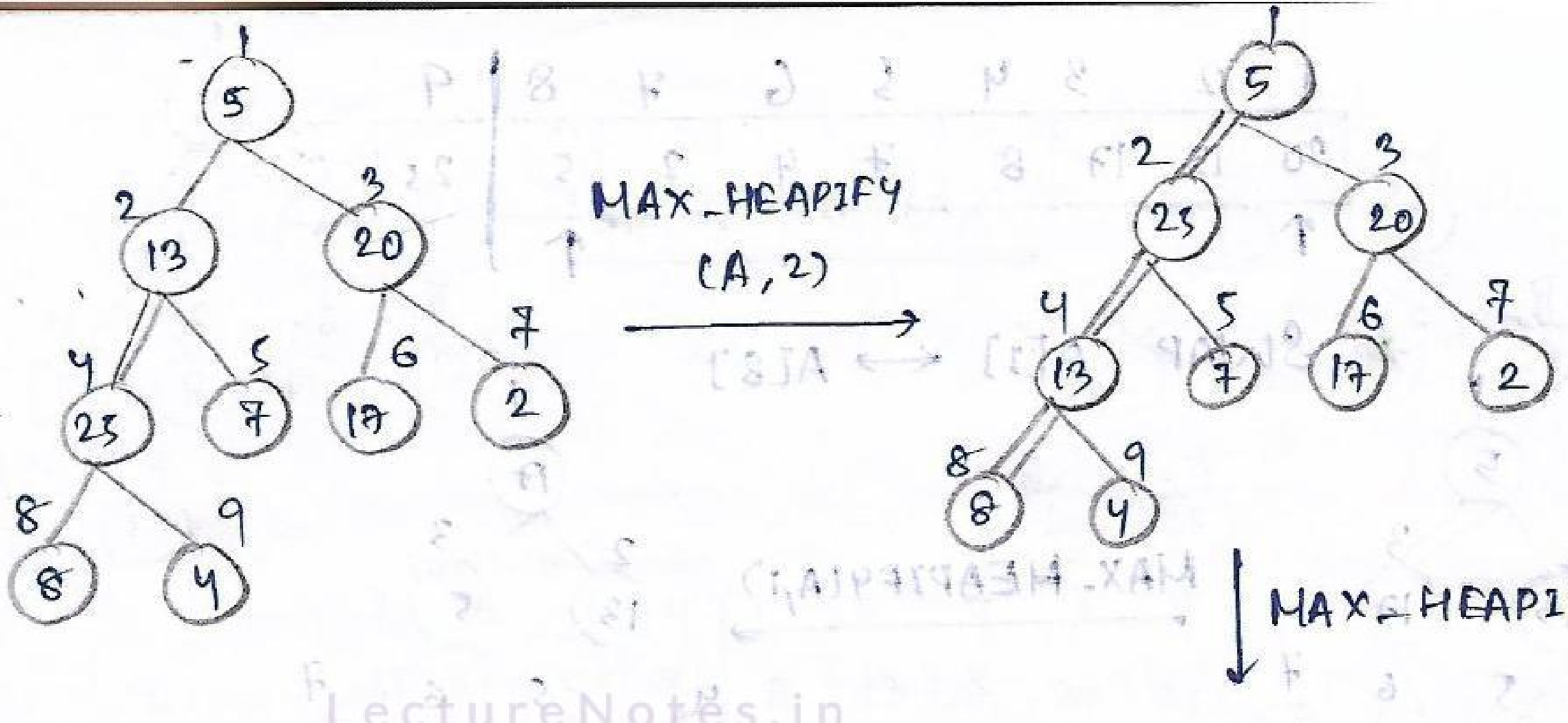
Sol. Let the sort be in ascending order.

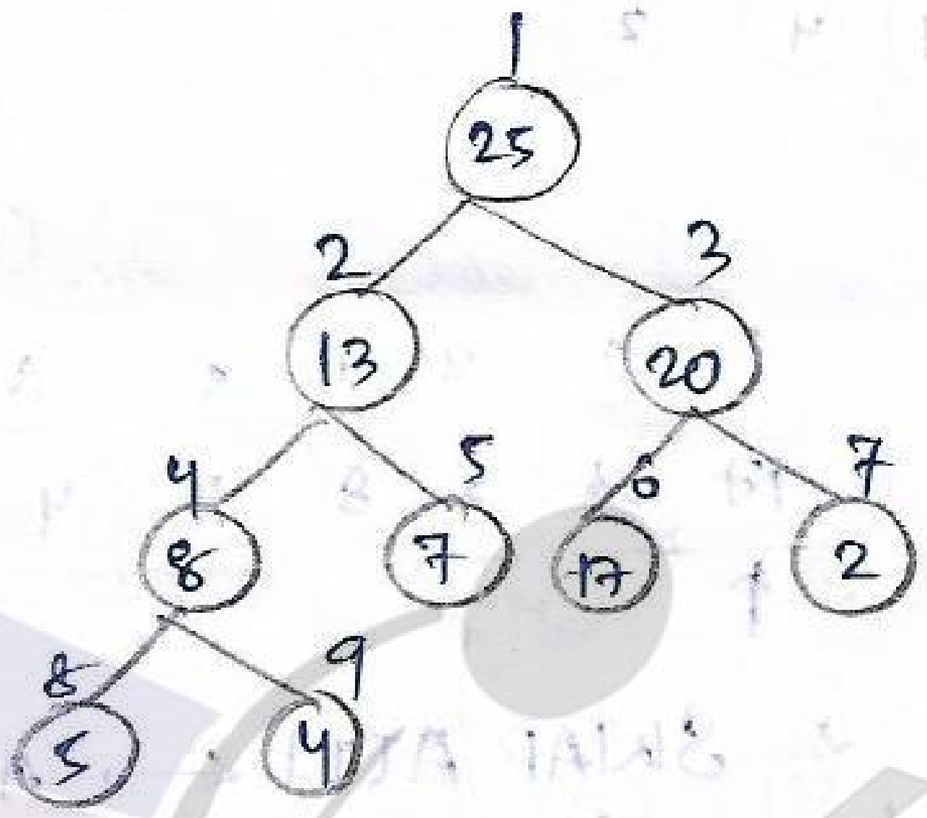
To sort in ascending order, find we have to build MAX_HEAP tree.

BUILD_MAX_HEAP(A) →

1	2	3	4	5	6	7	8	9
5	13	2	25	7	17	20	8	4





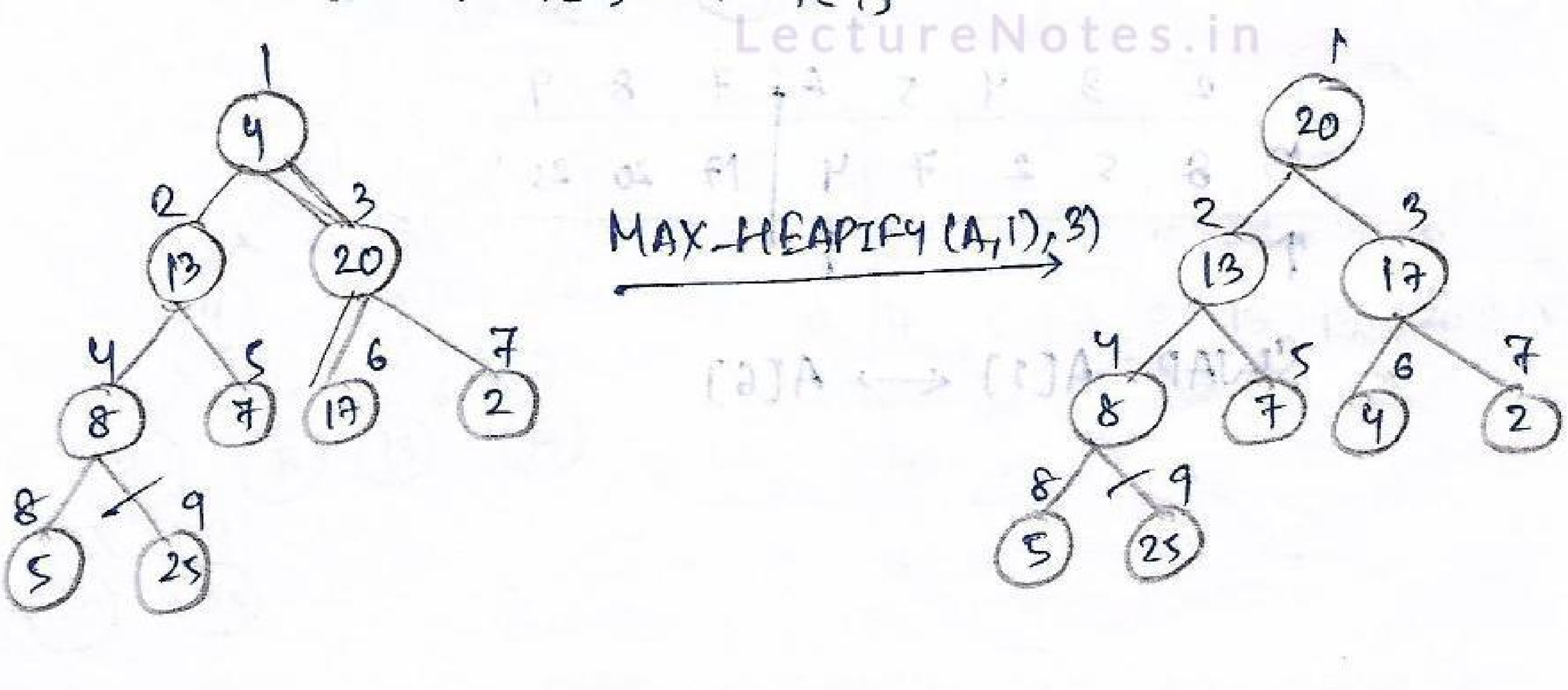


The corresponding array is.

1	2	3	4	5	6	7	8	9
25	13	20	8	7	17	2	5	4

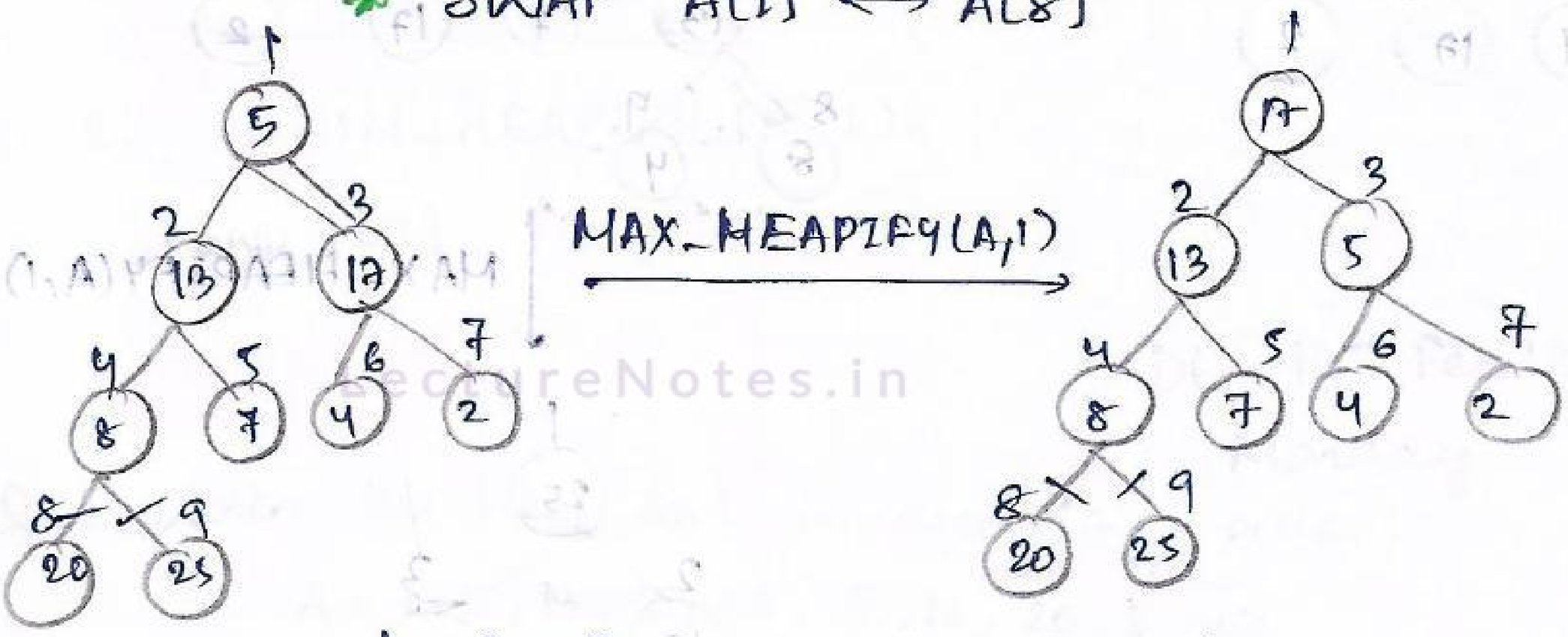
Swap the first element with last element.

* SWAP A[1] ↔ A[9]



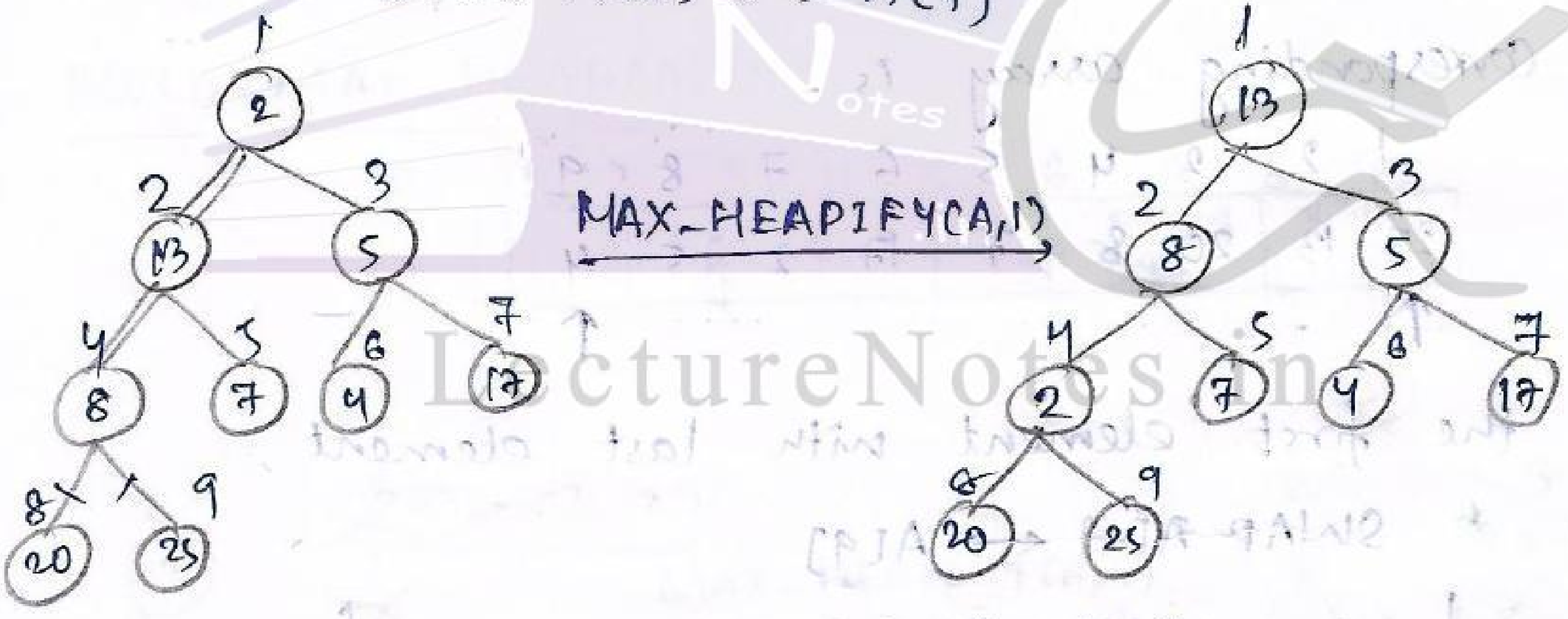
1	2	3	4	5	6	7	8	9
20	13	17	8	7	4	2	5	25

* SWAP A[1] ⟷ A[8]



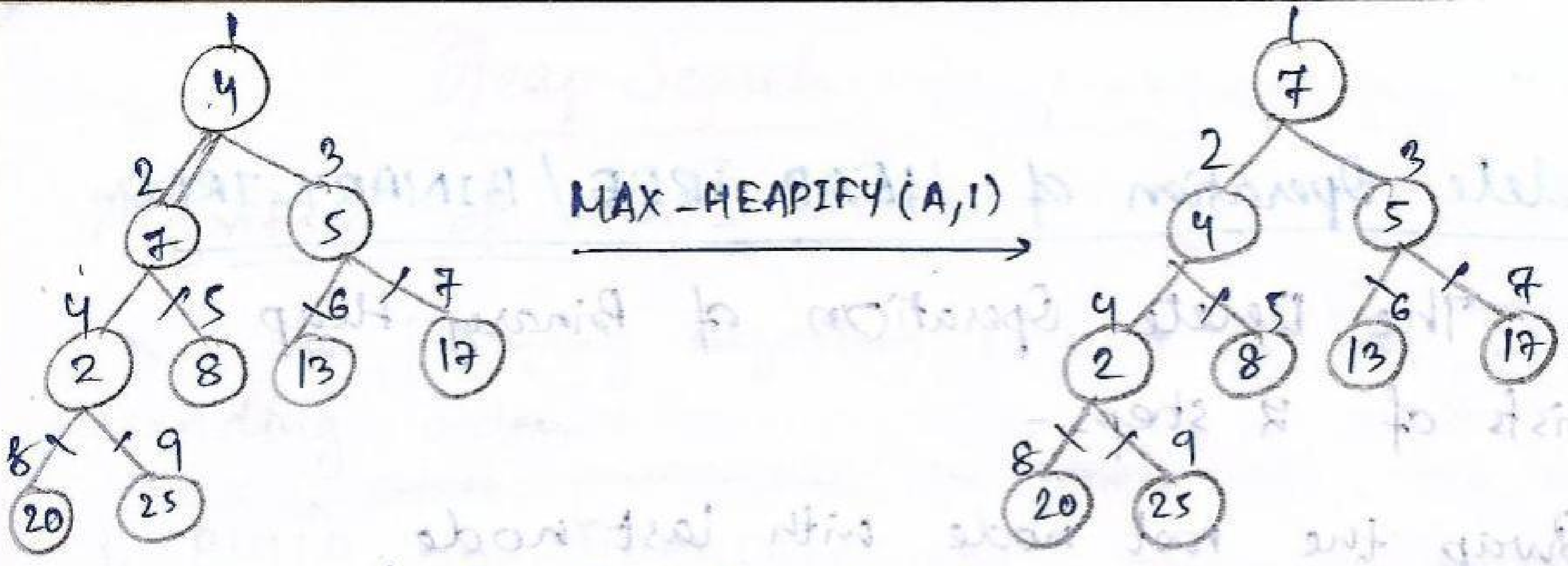
1	2	3	4	5	6	7	8	9
17	13	5	8	7	4	2	20	25

* SWAP A[1] ⟷ A[7]



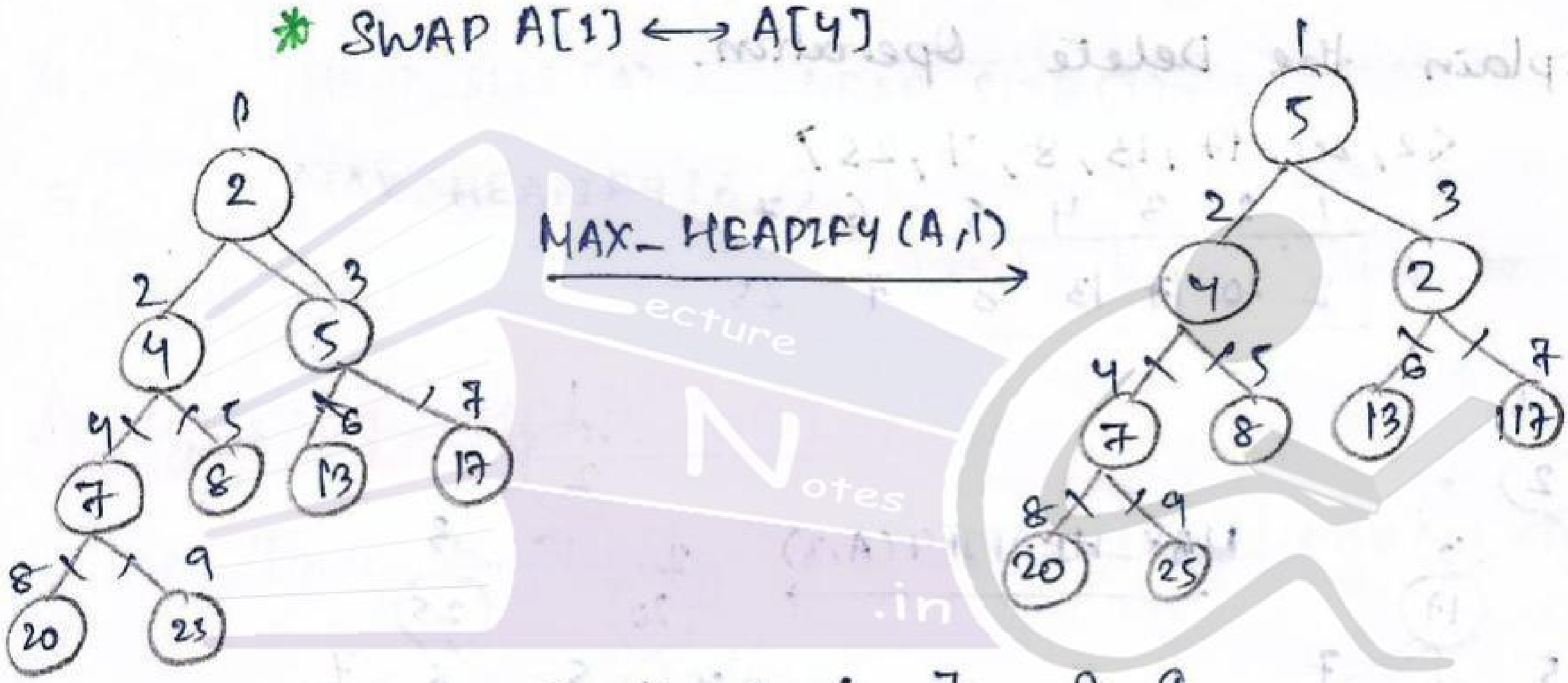
1	2	3	4	5	6	7	8	9
13	8	5	2	7	4	17	20	25

* SWAP A[1] ⟷ A[6]



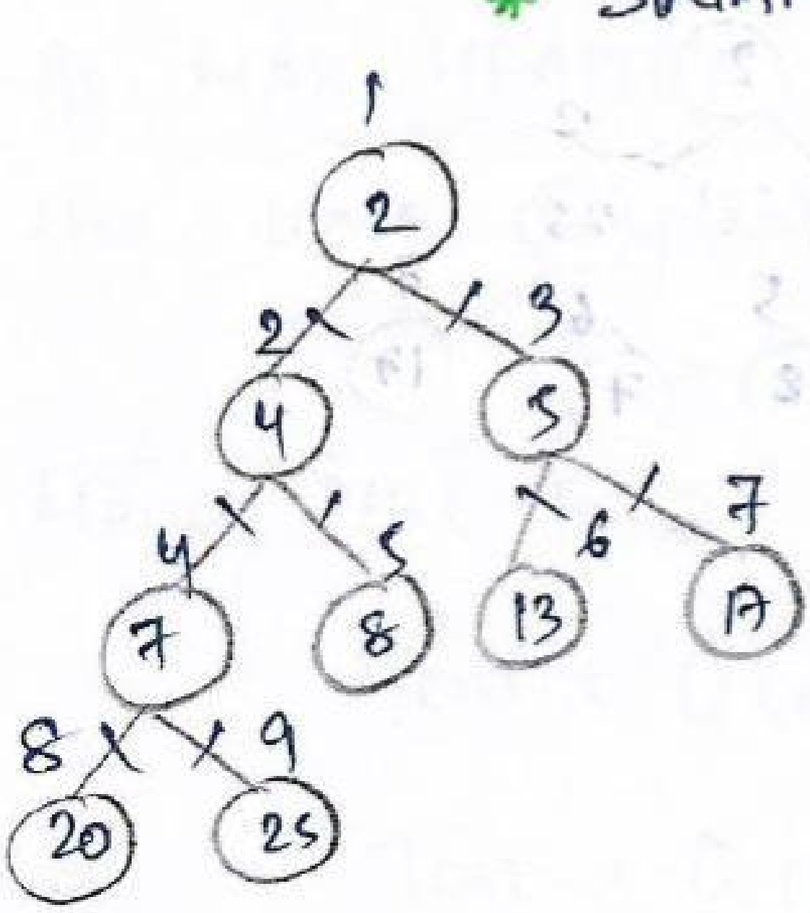
1	2	3	4	5	6	7	8	9
7	4	5	2	8	13	17	20	25

* SWAP A[1] ⟷ A[4]



1	2	3	4	5	6	7	8	9
5	4	2	7	8	13	17	20	25

* SWAP A[1] ⟷ A[3]



1	2	3	4	5	6	7	8	9
2	4	5	7	8	13	17	20	25

NOTE →

Delete Operation of HEAP_TREE / BINARY_TREE →

The Delete Operation of Binary Heap consists of 2 steps -

→ Swap the root node with last node.

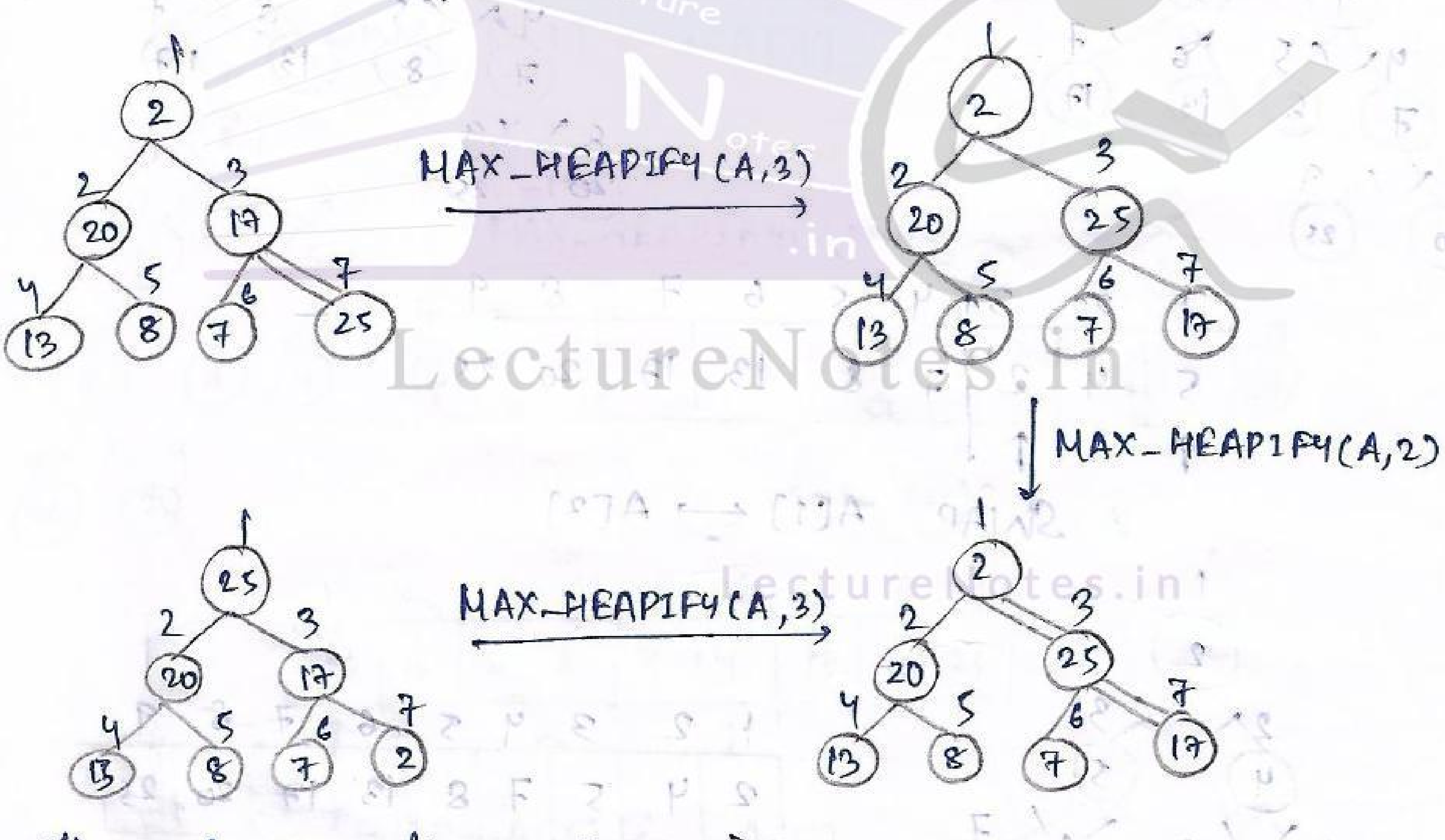
→ Apply the MAX_HEAPIFY A[1] or MIN_HEAPIFY A[1] depending on the tree if MAX_HEAP or MIN_HEAP.

Q. Explain the Delete Operation.

<2, 20, 17, 13, 8, 7, 25>

Sol.

1	2	3	4	5	6	7
2	20	17	13	8	7	25



The Corresponding array is -

1	2	3	4	5	6	7
25	20	17	13	8	7	2

It is sorted in descending order.

LectureNotes.in

Heap Search

Algorithm of HEAP_SEARCH →

The following algorithm is used to sort in ascending order-

1. BUILD_MAX_HEAP(A).
2. FOR I ← $\frac{\text{LENGTH(A)}}{2}$ to 2
3. EXCHANGE A[1] ↔ A[I]
4. HEAP_SIZE[A] ← HEAP_SIZE[1] - 1
5. MAX_HEAPIFY(A, 1)

[END FOR]

Analysis of HEAP_SORT →

The Time Complexity to BUILD_MAX_HEAP(A) is

$$T_B = O(n \log n)$$

The Time required for line no. 5 -

$$T_N = O(\log n)$$

As MAX_HEAPIFY will execute for $\frac{n}{2}$ times. Hence the time complexity for loop 2 is -

$$T_1 = O(n \log n)$$

Hence, the total time complexity is

$$T(n) = O(n \log n) + O(n \log n)$$

$$T(n) = O(n \log n)$$

LectureNotes.in

The time complexity for Heap Sort is

$$T(n) = O(n \log n)$$

NOTES →

To sort the array in descending order, we have to construct MIN_HEAP using BUILD_MIN_HEAP.

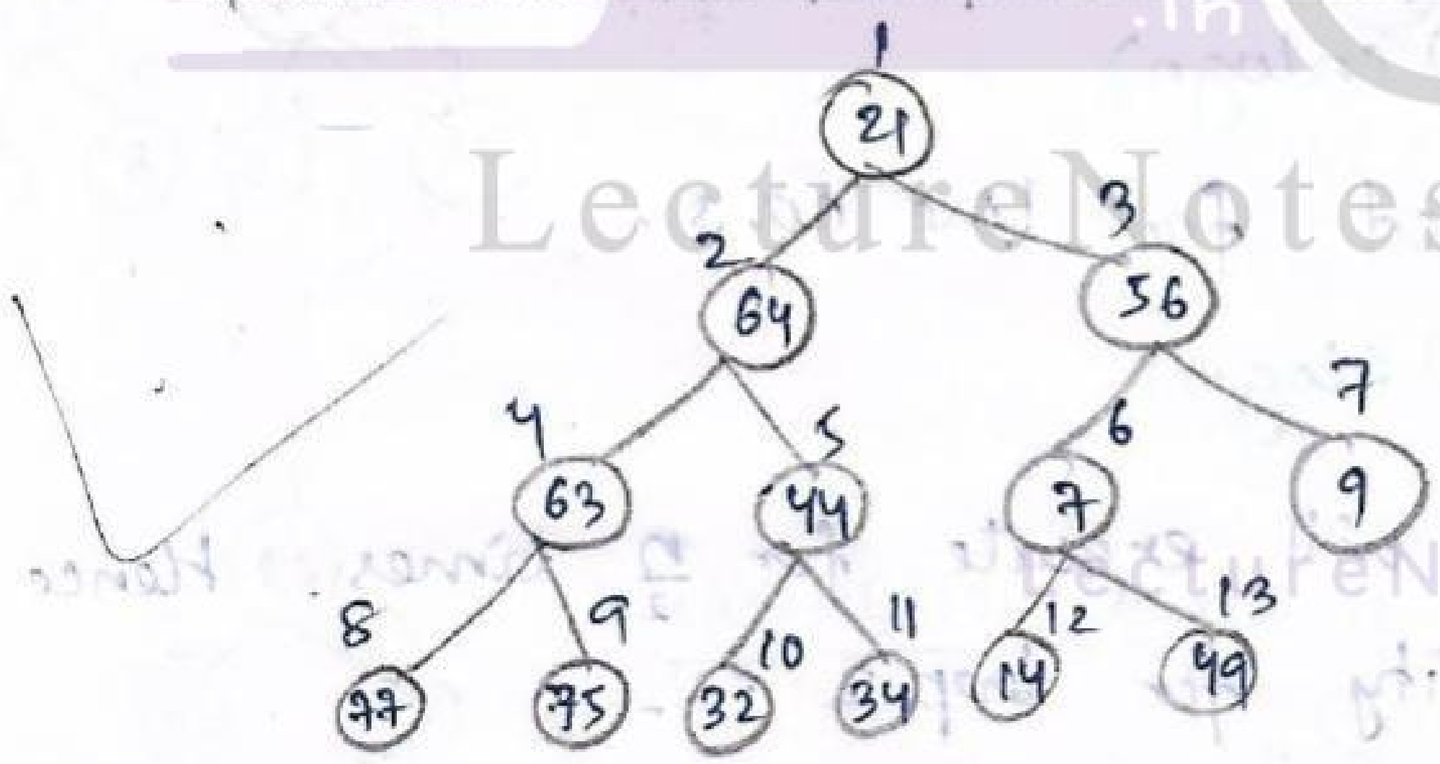
Q1. Explain the process of HEAP_SORT. Write an algorithm to construct MIN_HEAP. Write the algorithm for HEAP_SORT to sort the data in descending order.

Q2. Sort the following data in descending order:-

<21, 64, 56, 63, 44, 7, 9, 77, 75, 32, 34, 14, 49>

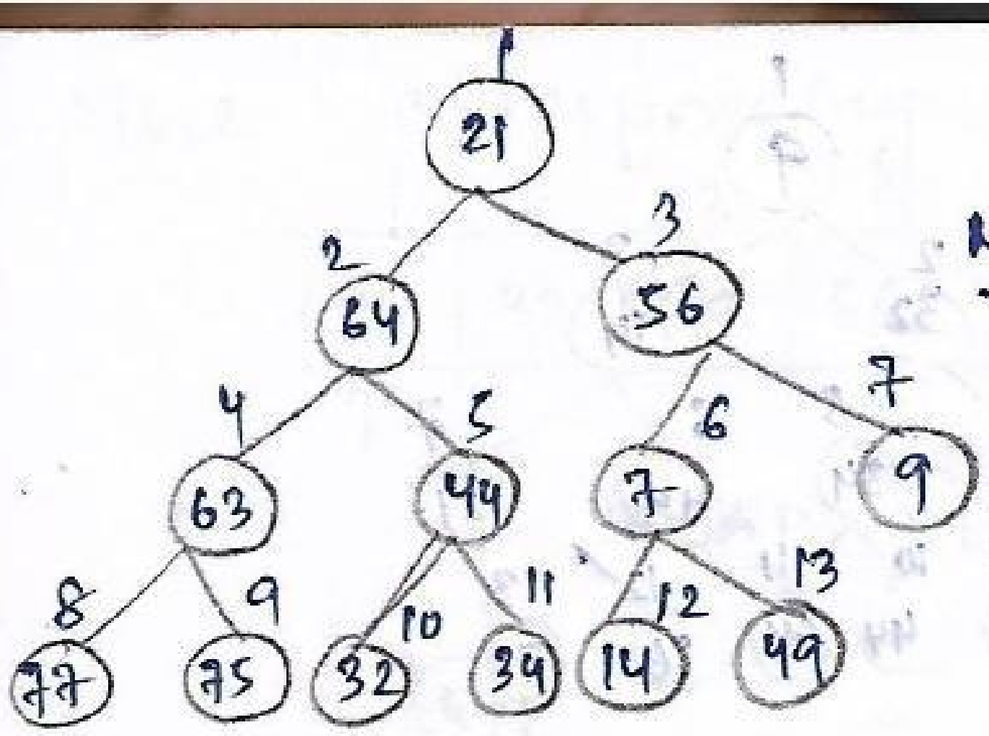
Sol.

1	2	3	4	5	6	7	8	9	10	11	12	13
21	64	56	63	44	7	9	77	75	32	34	14	49

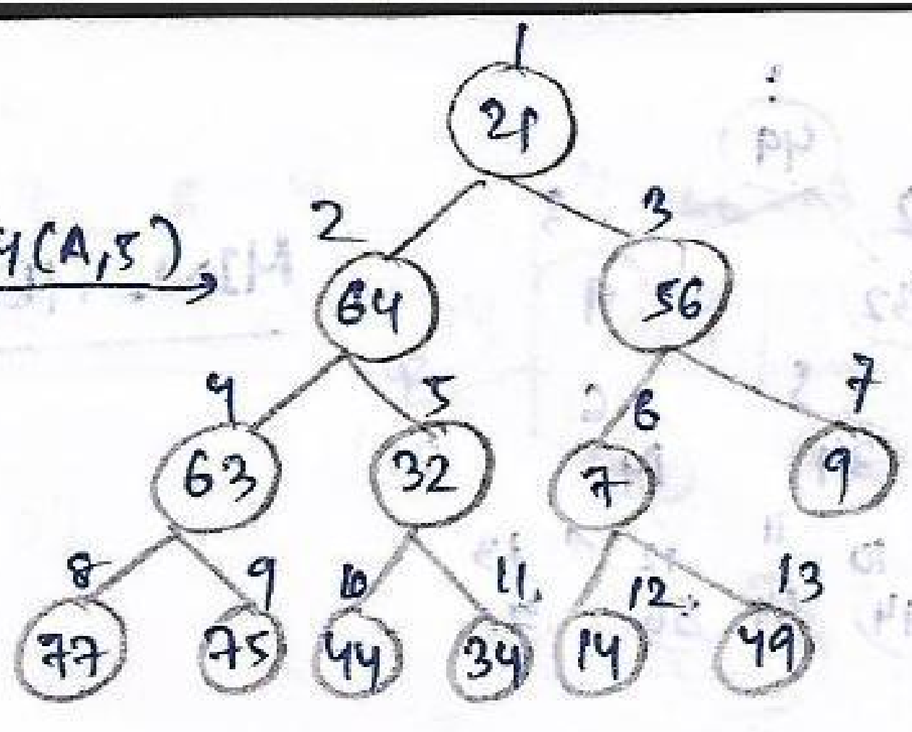


To sort in descending order, first we have to build MIN_HEAP tree.

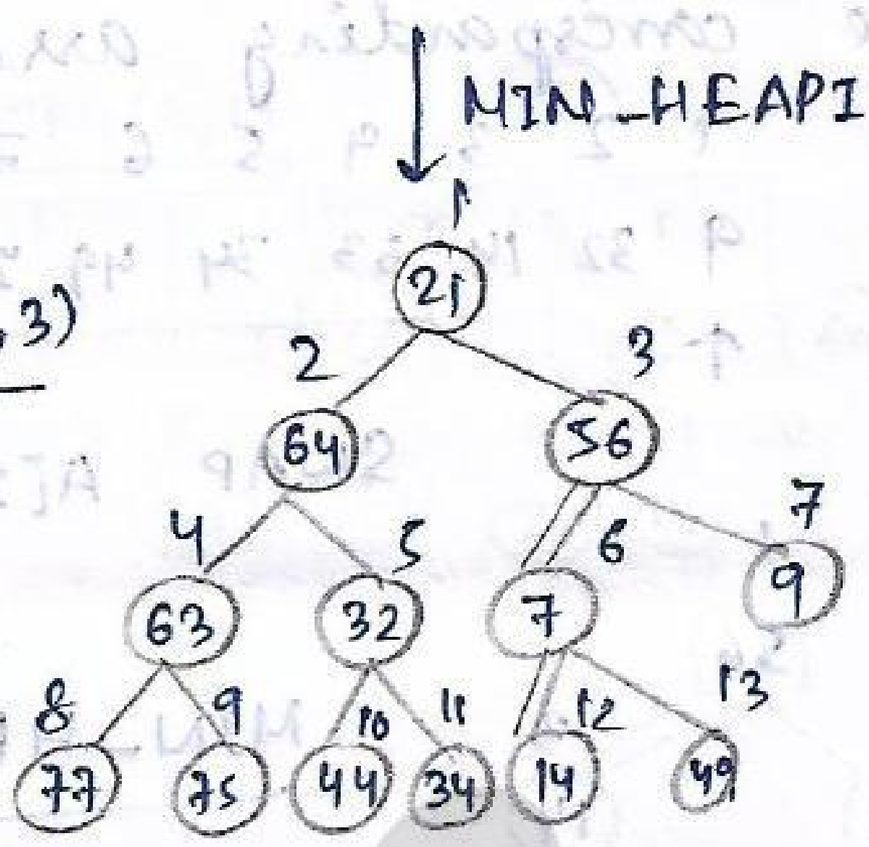
LectureNotes.in



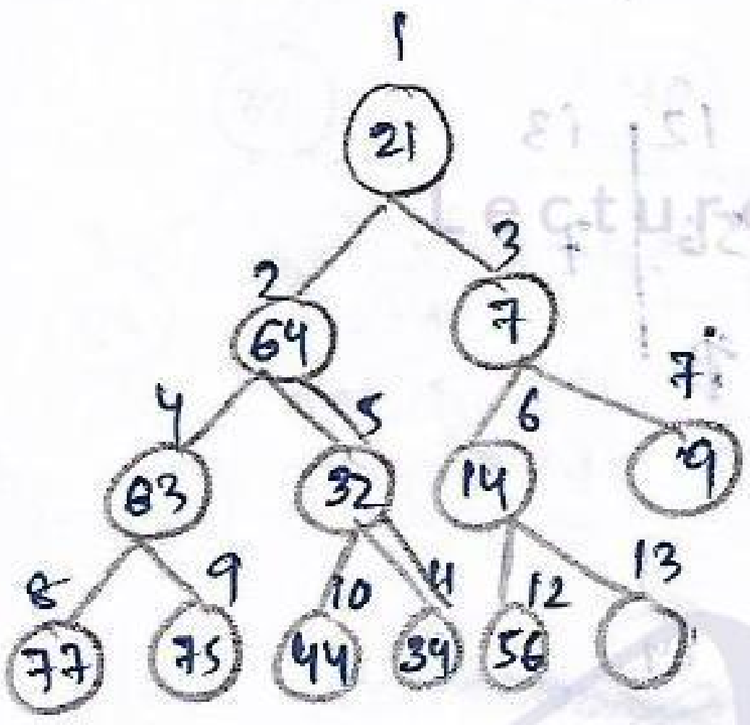
MIN_HEAPIFY(A,5) →



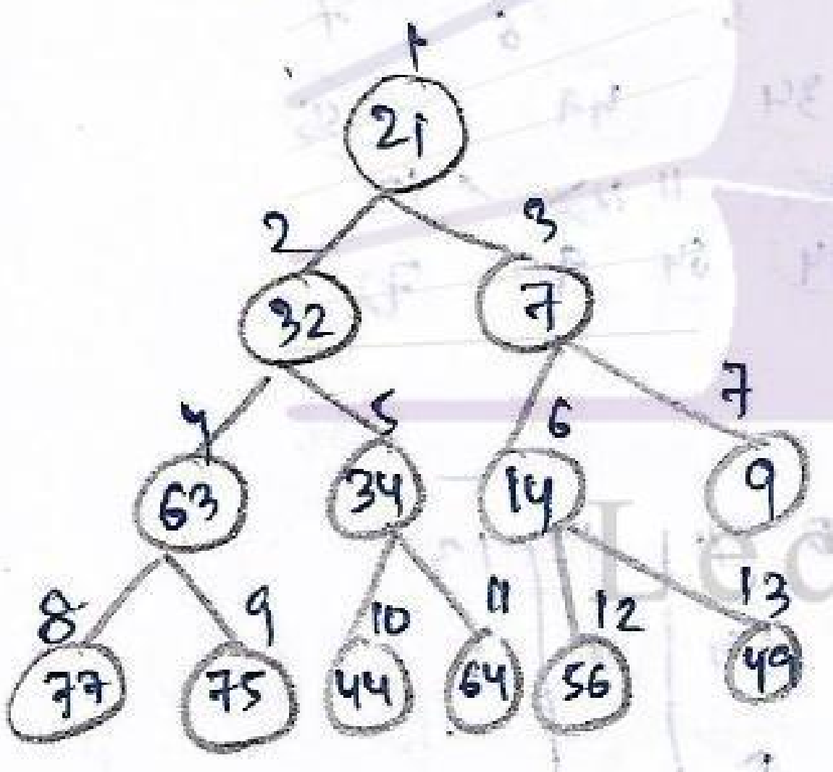
MIN_HEAPIFY(A,4) ↓



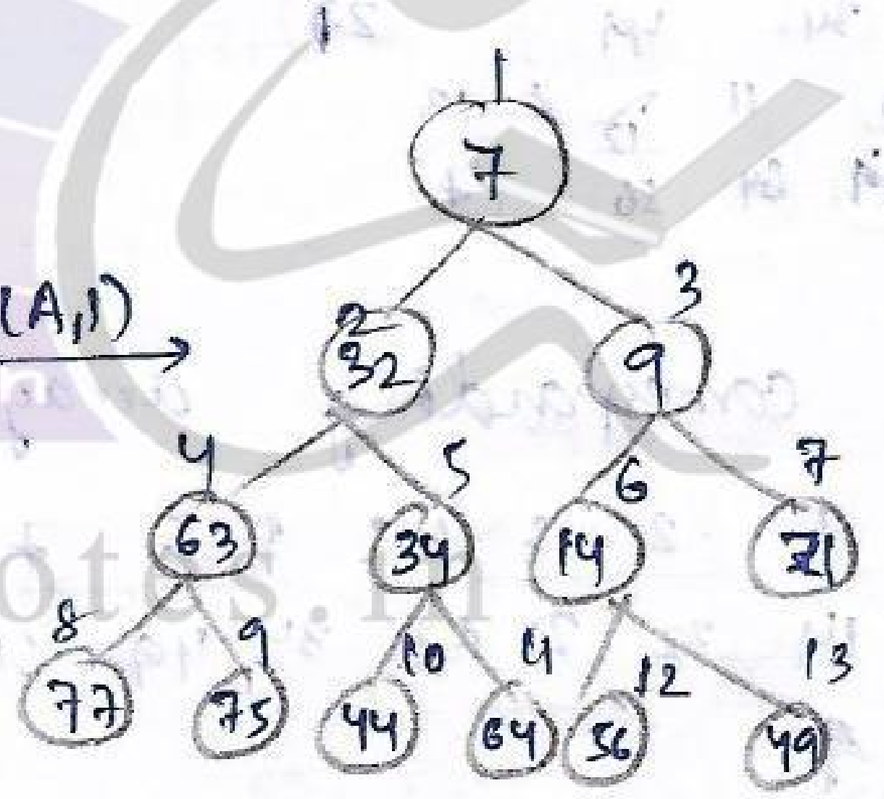
← MIN_HEAPIFY(A,3)



↓ MIN_HEAPIFY(A,2)



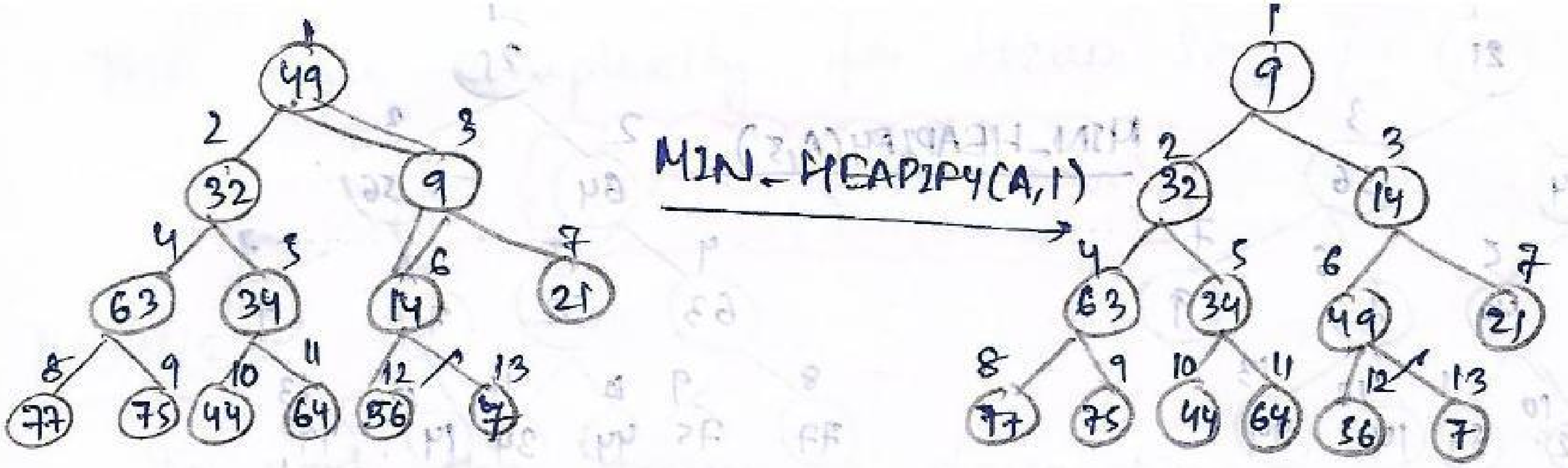
MIN_HEAPIFY(A,1) →



The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
7	32	9	63	34	14	21	77	75	44	64	56	49

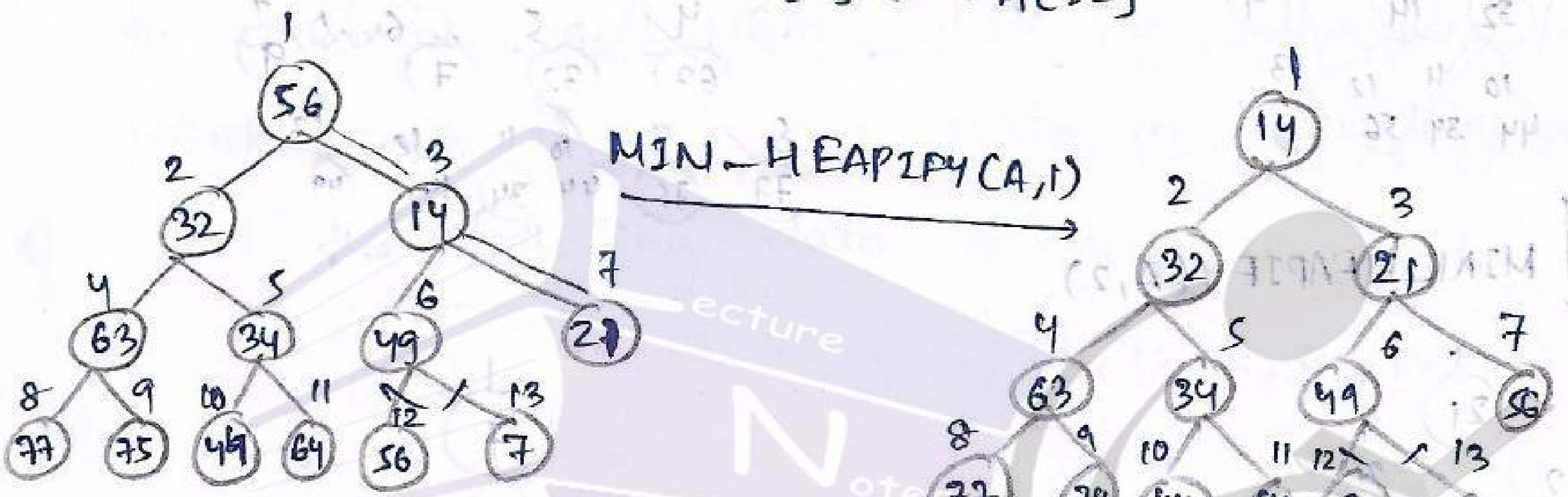
SWAP A[1] ⟷ A[13]



The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
9	32	14	63	34	49	27	77	75	44	64	56	7

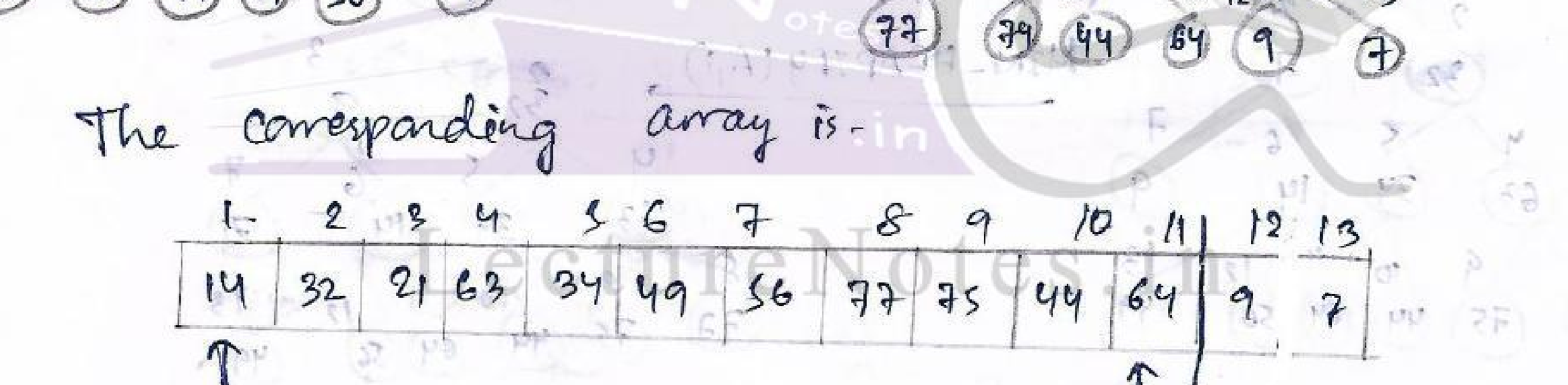
SWAP A[1] ⟷ A[12]



The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
14	32	21	63	34	49	56	77	75	44	64	9	7

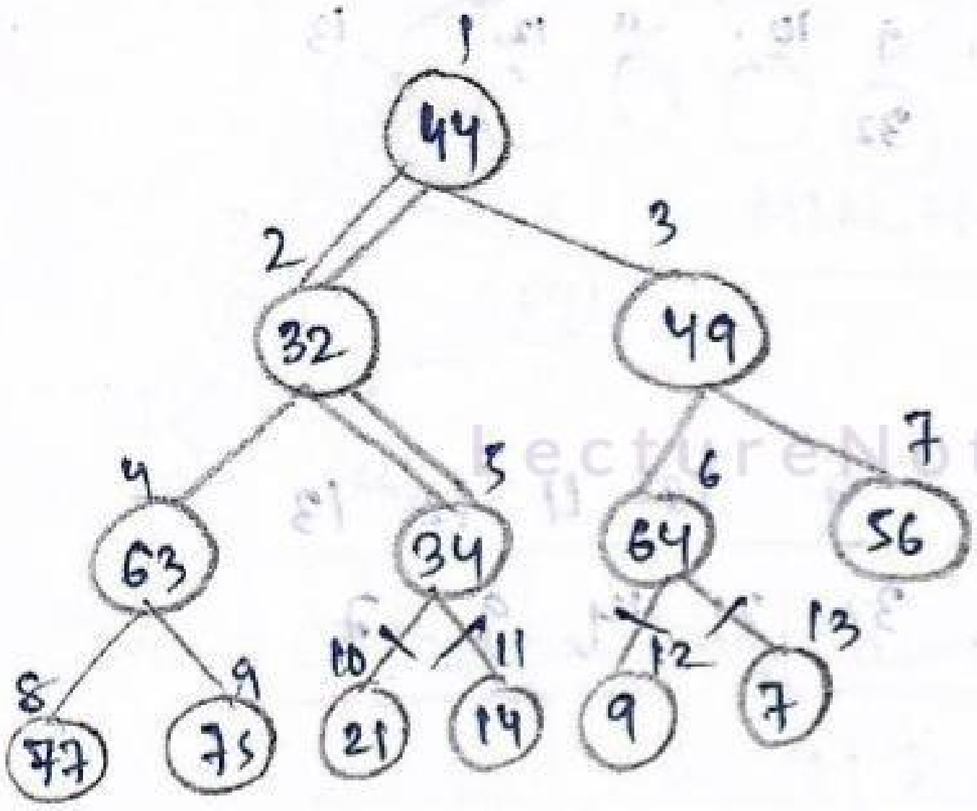
SWAP A[1] ⟷ A[11]



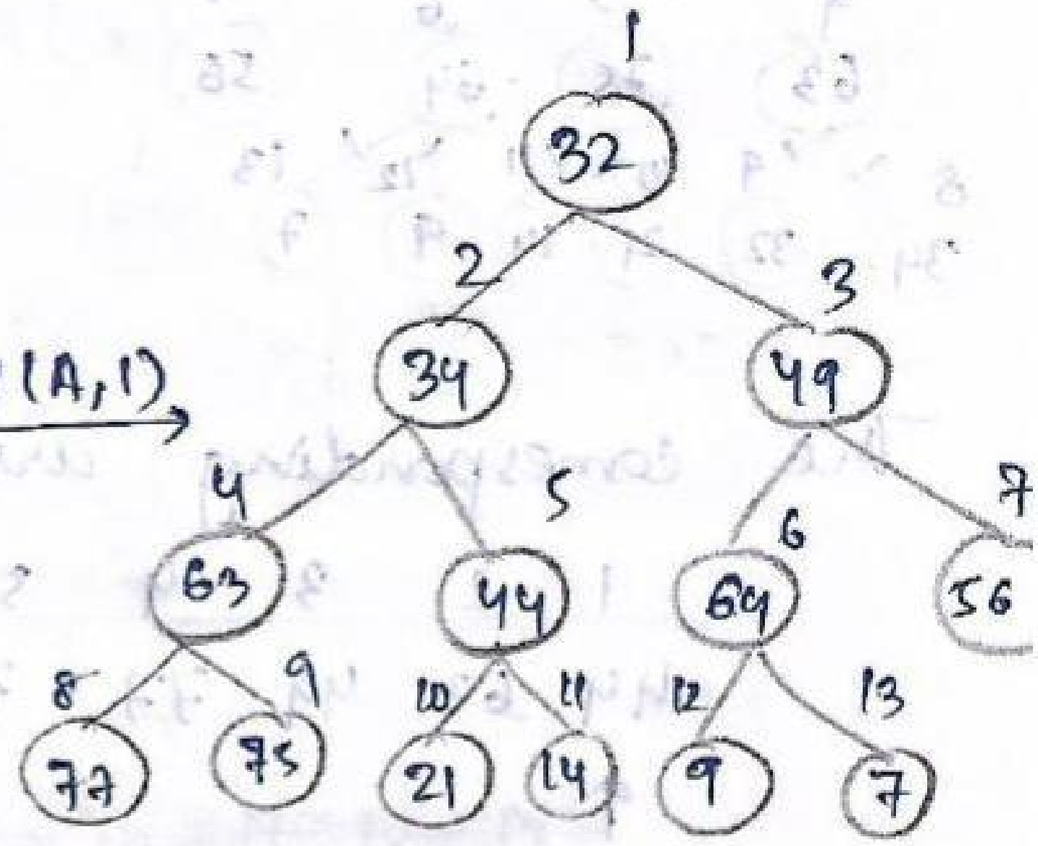
The corresponding array is -

1	2	3	4	5	6	7	8	9	10	11	12	13
21	32	49	63	34	64	56	77	75	44	14	9	7

SWAP A[1] ⟷ A[10]



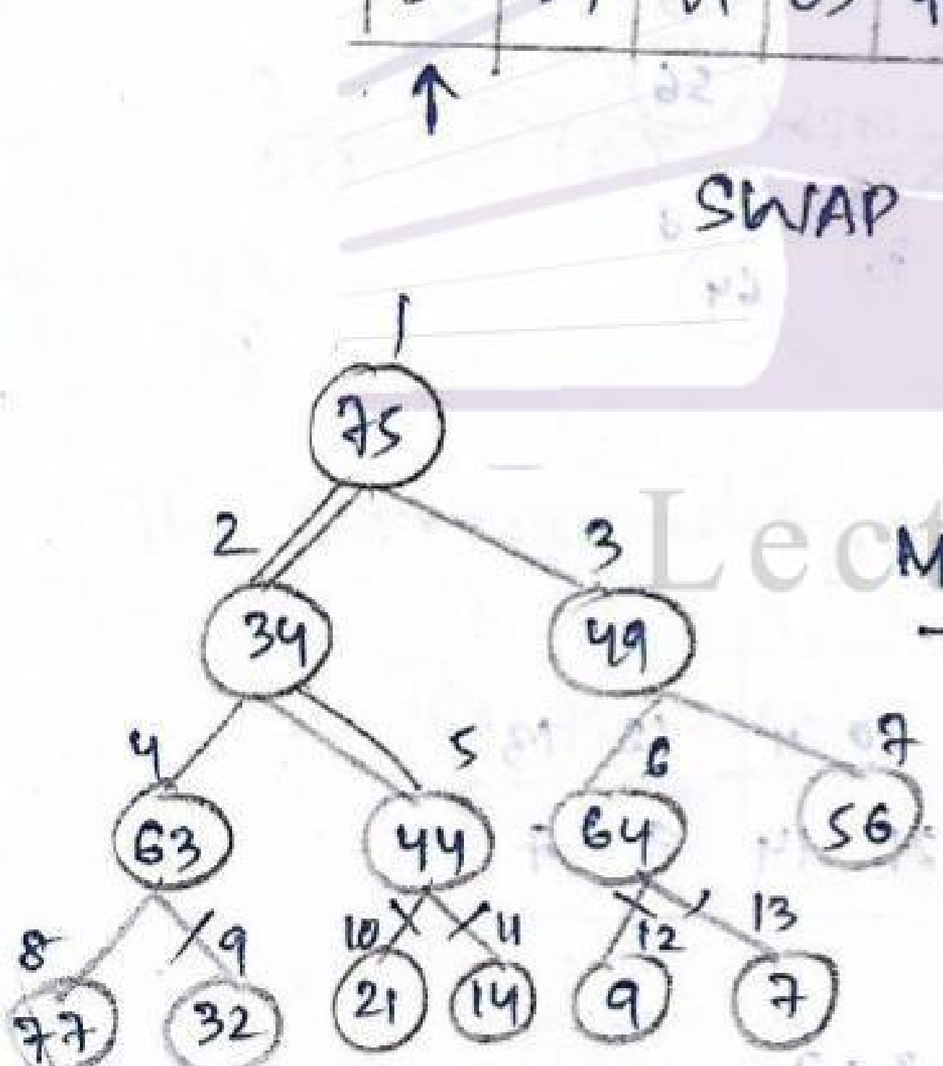
MIN_HEAPIFY(A,1)



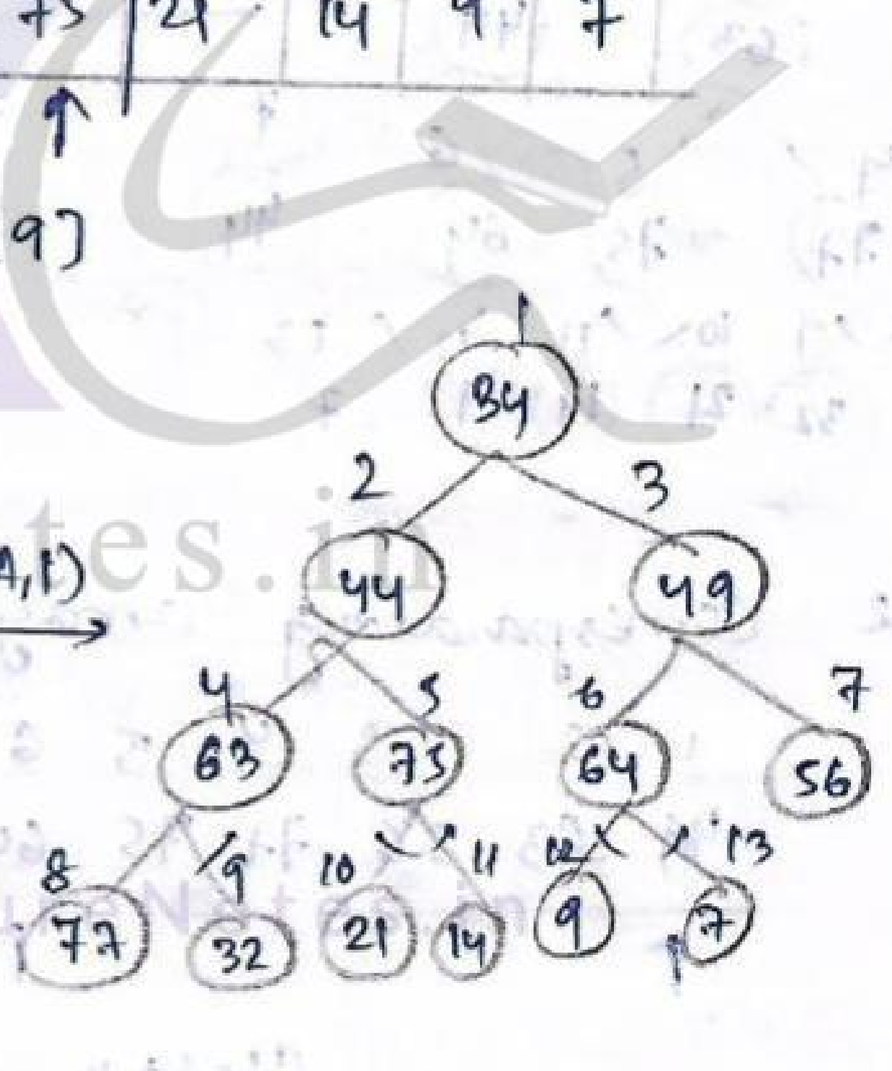
The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
32	34	49	63	44	64	56	77	75	21	14	9	7

SWAP A[1] ⟷ A[9]



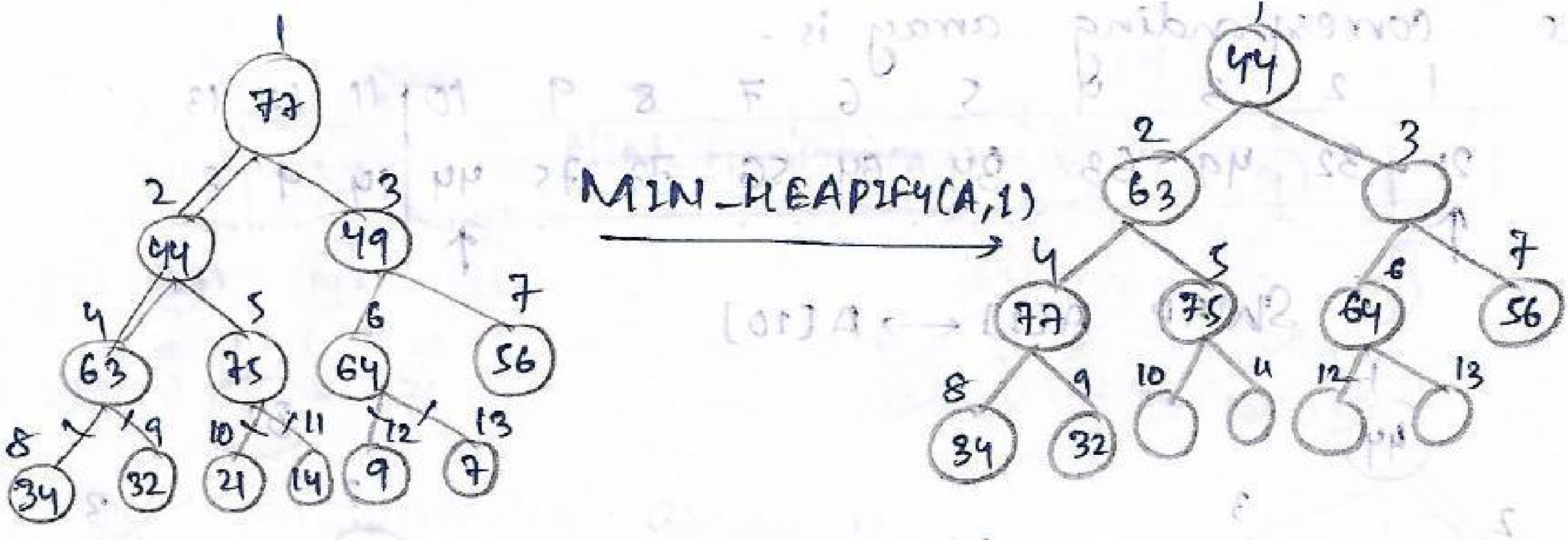
MIN_HEAPIFY(A,1)



The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
34	44	49	63	75	64	56	77	32	21	14	9	7

SWAP A[1] ⟷ A[8]

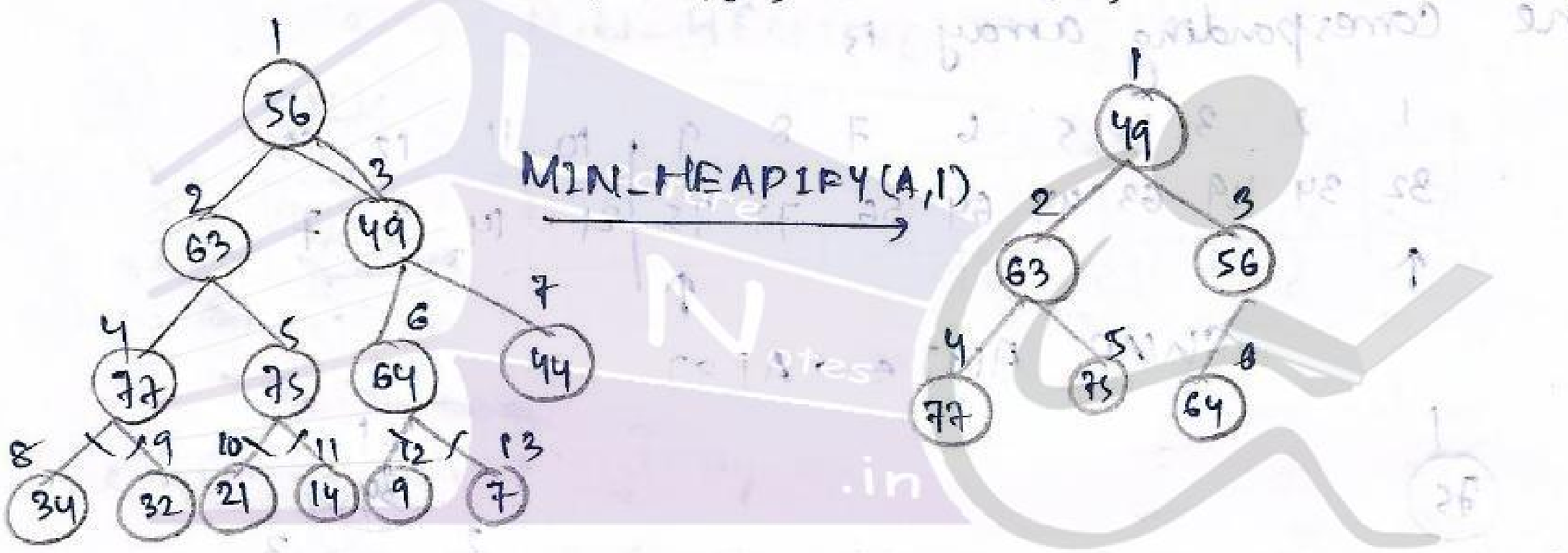


The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
44	63	49	77	75	64	56	34	32	21	14	9	7

↑ ↑

SWAP A[1] ⟷ A[7]

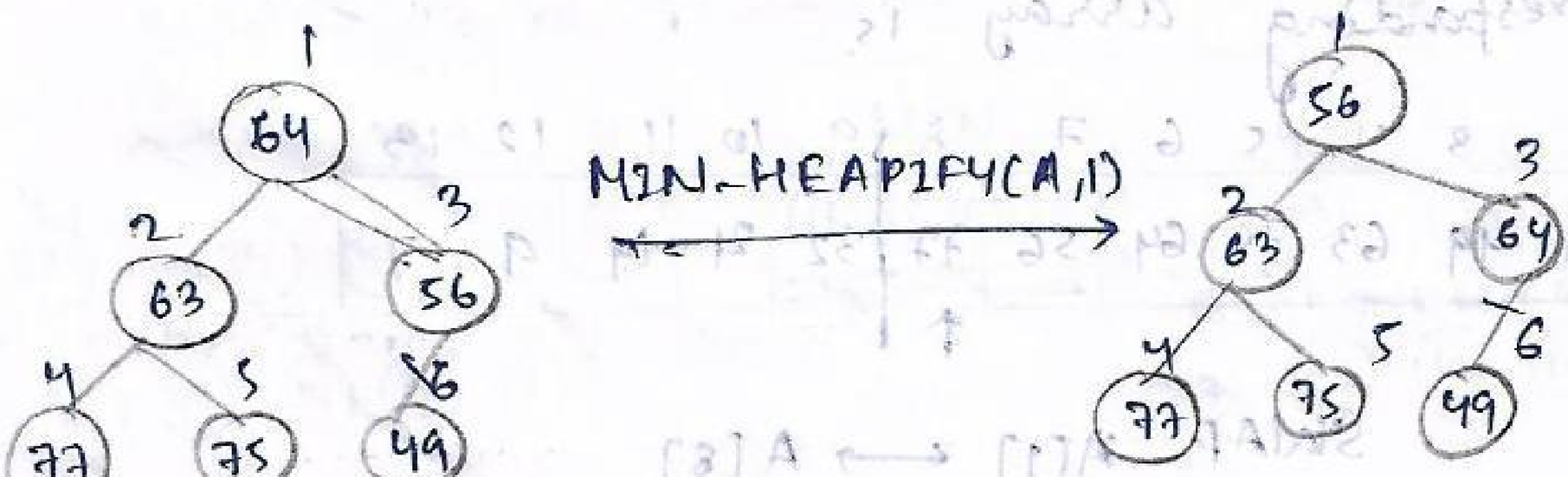


The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
49	63	56	77	75	64	44	34	32	21	14	9	7

↑ ↑

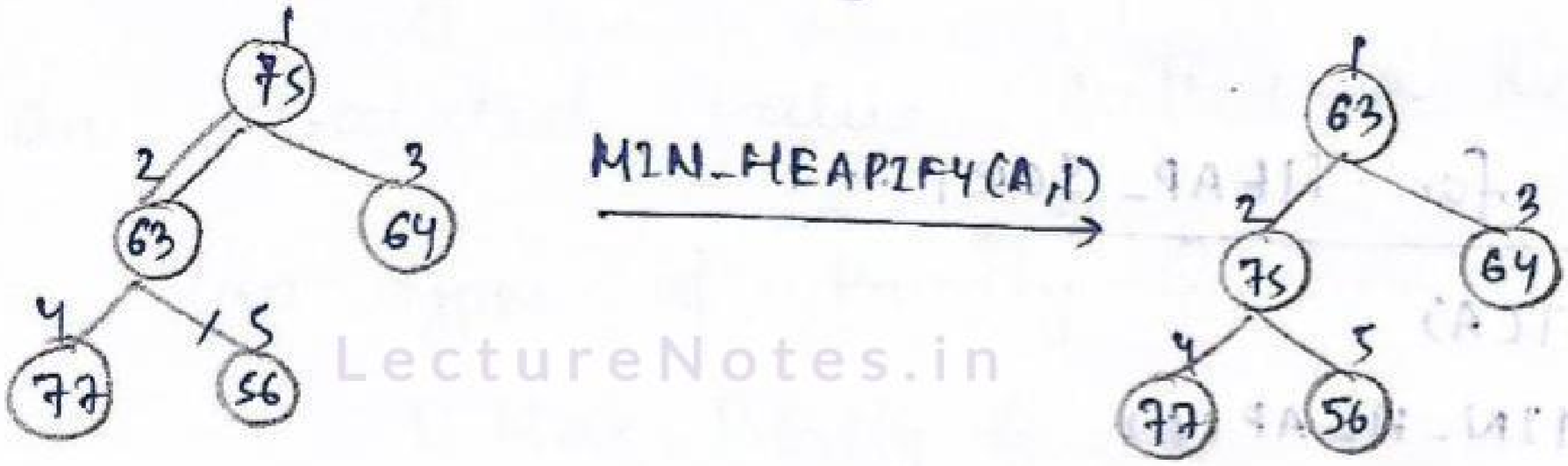
SWAP A[1] ⟷ A[6]



The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
56	63	64	77	75	49	44	34	32	21	14	9	7
↑				↑								

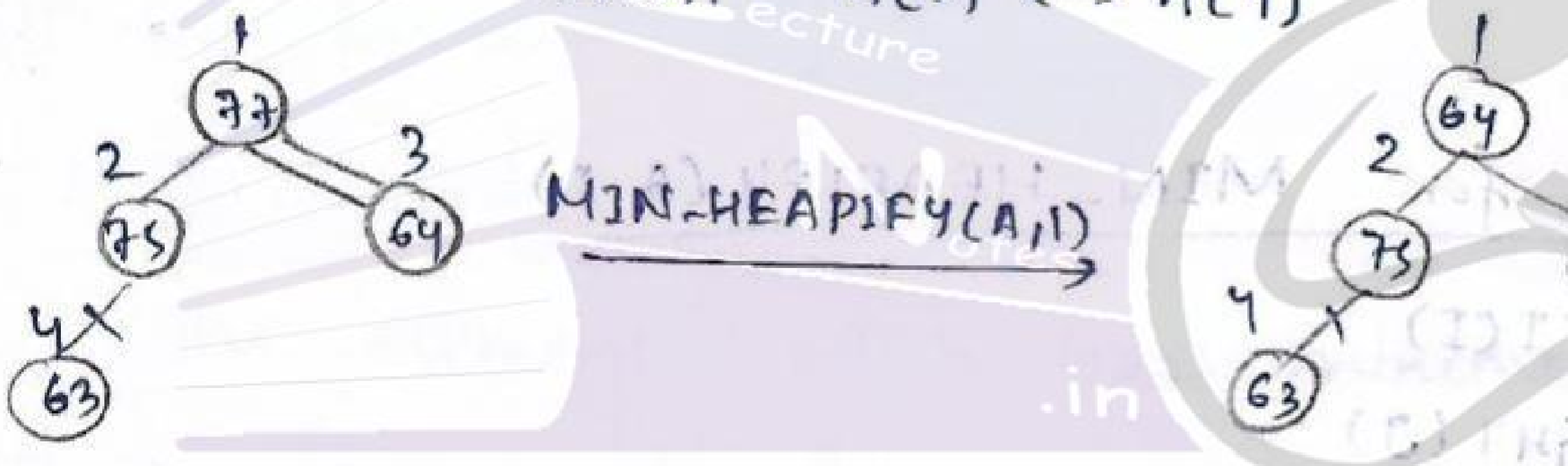
SWAP A[1] ⟷ A[5]



The Corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
63	75	64	77	56	49	44	34	32	21	14	9	7
↑			↑									

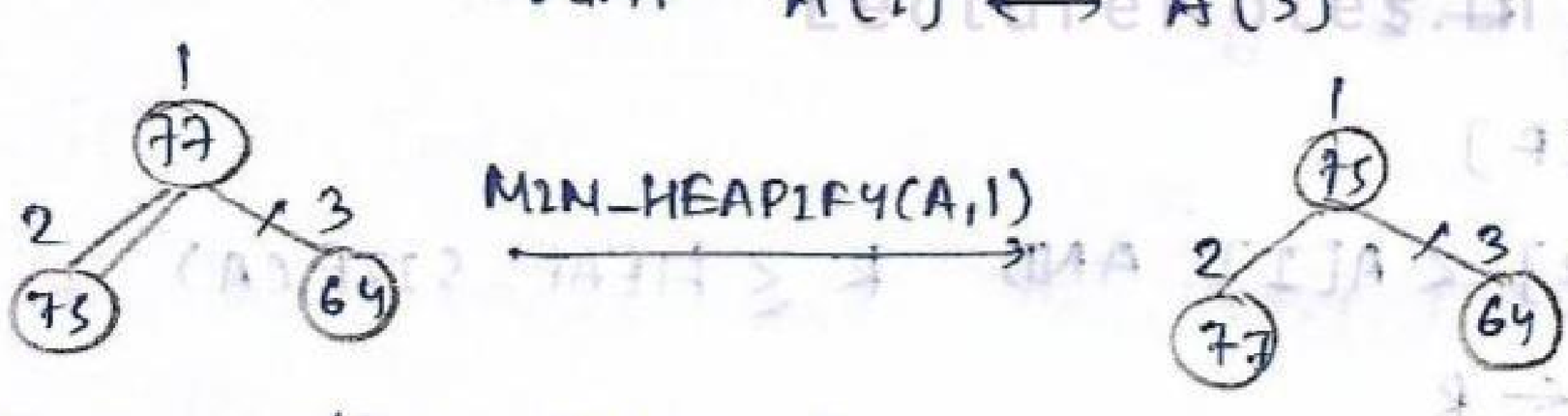
SWAP A[1] ⟷ A[4]



The corresponding array is

1	2	3	4	5	6	7	8	9	10	11	12	13
64	75	77	63	56	49	44	34	32	21	14	9	7
↑		↑										

SWAP A[1] ⟷ A[3]



The corresponding array is.

1	2	3	4	5	6	7	8	9	10	11	12	13
75	77	64	63	56	49	44	34	32	21	14	9	7
↑	↑											

SWAP A[1] ⟷ A[2]

The required array is

1	2	3	4	5	6	7	8	9	10	11	12	13
77	75	64	63	56	49	44	34	32	21	14	9	7

which is sorted in descending order.

Sol.1

Algorithm for HEAP_SORT →

HEAP_SORT(A)

1. BUILD_MIN_HEAP(A)
2. FOR I ← LENGTH(A) / 2 to 2
3. EXCHANGE A[1] ⟷ A[I]
4. HEAP_SIZE[A] ← HEAP_SIZE[A] - 1
5. MIN_HEAPIFY(A, 1)

[END FOR]

Algorithm for MIN_HEAPIFY(A, I)

1. l ← LEFT(I)
2. r ← RIGHT(J)
3. IF A[l] < A[I] AND l ≤ HEAP_SIZE(A)
4. SMALL ← l

ELSE

5. SMALL ← I

[END IF]

6. IF A[R] < A[I] AND R ≤ HEAP_SIZE(A)
7. SMALL ← R

[END IF]

8. IF SMALL ≠ I THEN
9. EXCHANGE A[I] ⟷ A[SMALL]
10. MIN_HEAPIFY(A, SMALL)

[END IF]

LectureNotes.in

Priority Queue

Dt → 13th Feb 18
Tuesday

A Priority Queue is a data structure for maintaining a set S of elements each with an associated value called a Key.

Two types of Priority Queue -

1. Max Priority Queue
2. Min Priority Queue

1. Max Priority Queue

The element which has maximum key value will be processed first.

2. Min Priority Queue

The element which has minimum key value will be processed first.

LectureNotes.in

Implementation of Max Priority Queue →

* It is implemented by using MAX-HEAP Tree.
* The trees are constructed depending on the key.

Some Operations of MAX Priority Queue -

1. HEAP_MAXIMUM(A)

3. HEAP_INCREASE_KEY (A, I, KEY)

4. MAX_HEAP_INSET (A, KEY)

1. HEAP_MAXIMUM(A) →

It displays the element which has maximum priority queue.

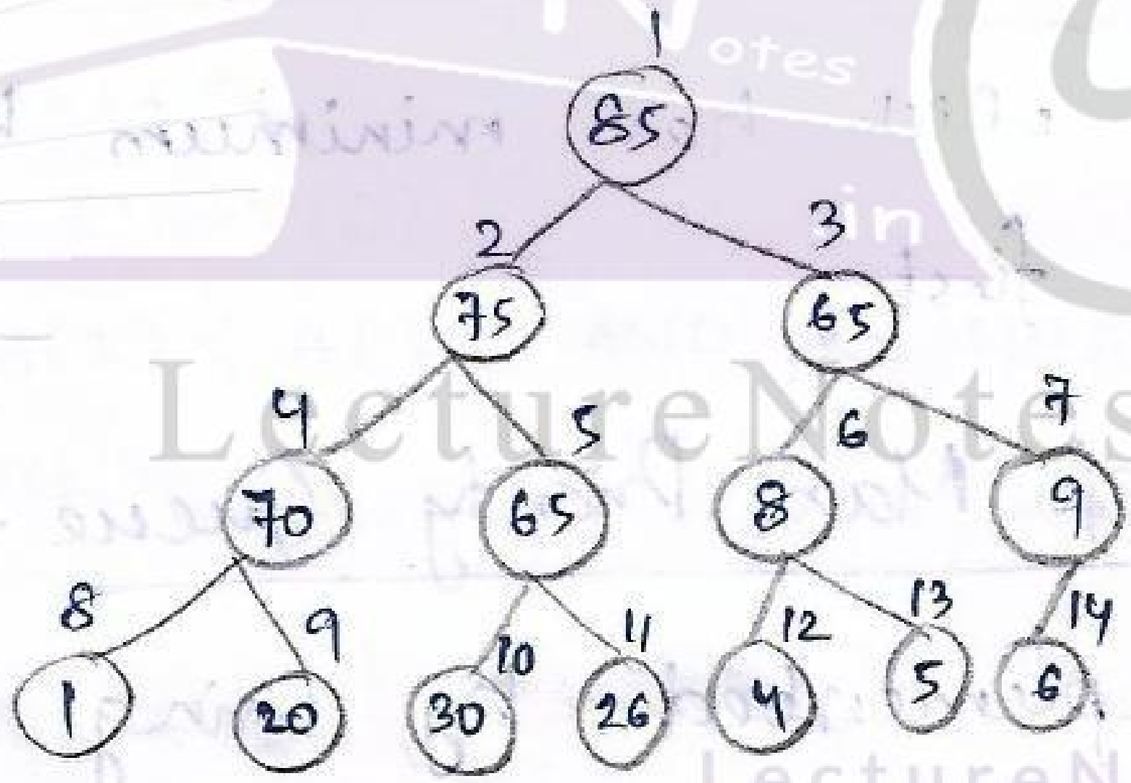
Algorithm →

1. RETURN (A[1])

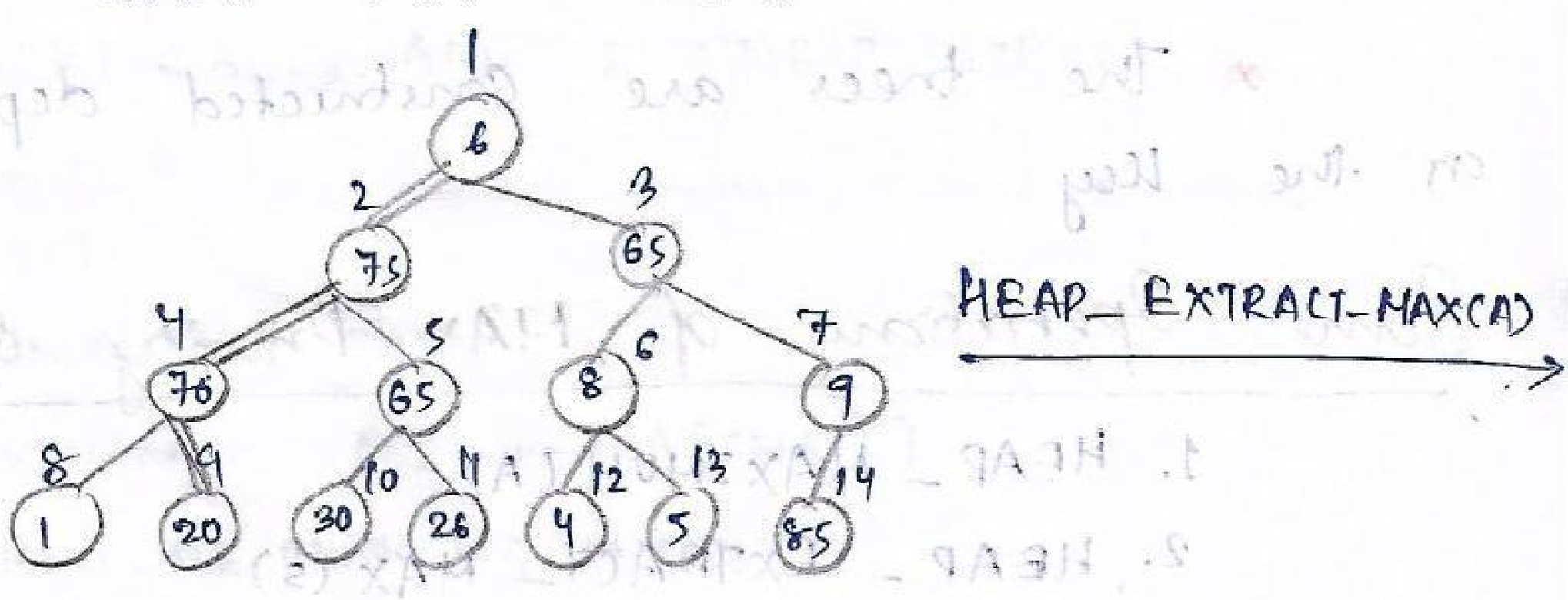
The Time complexity for T(N) is

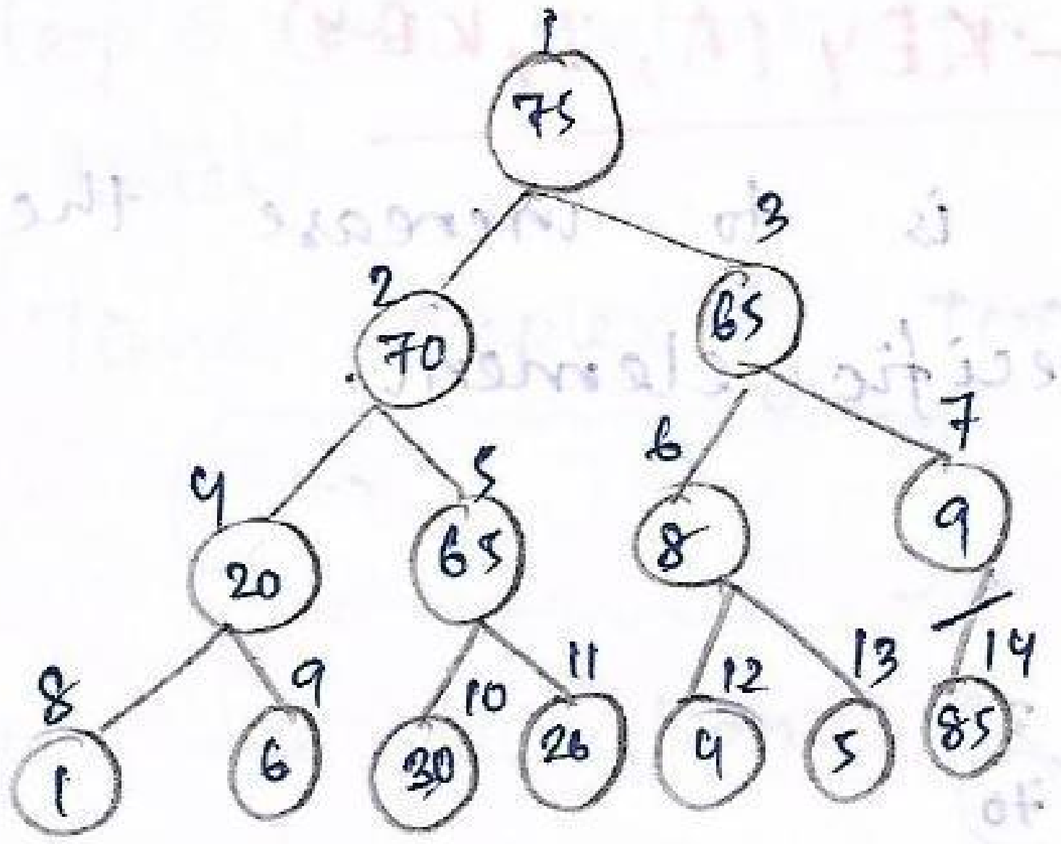
$T(N) = \theta(1)$

2. HEAP_EXTRACT_MAX(A) →



SWAP A[1] ⟷ A[14]





Algorithm →

1. If HEAP_SIZE(A) < 1
2. DISPLAY "Underflow Condition"
3. RETURN

[END IF]

4. MAX ← A[1]
5. SWAP A[1] ↔ A[HEAP_SIZE(A)]
6. HEAP_SIZE(A) ← HEAP_SIZE(A) − 1
7. MAX_HEAPIFY(A, 1)
8. RETURN MAX

The Time Complexity for step 7 is

$$T(n) = O(\log n)$$

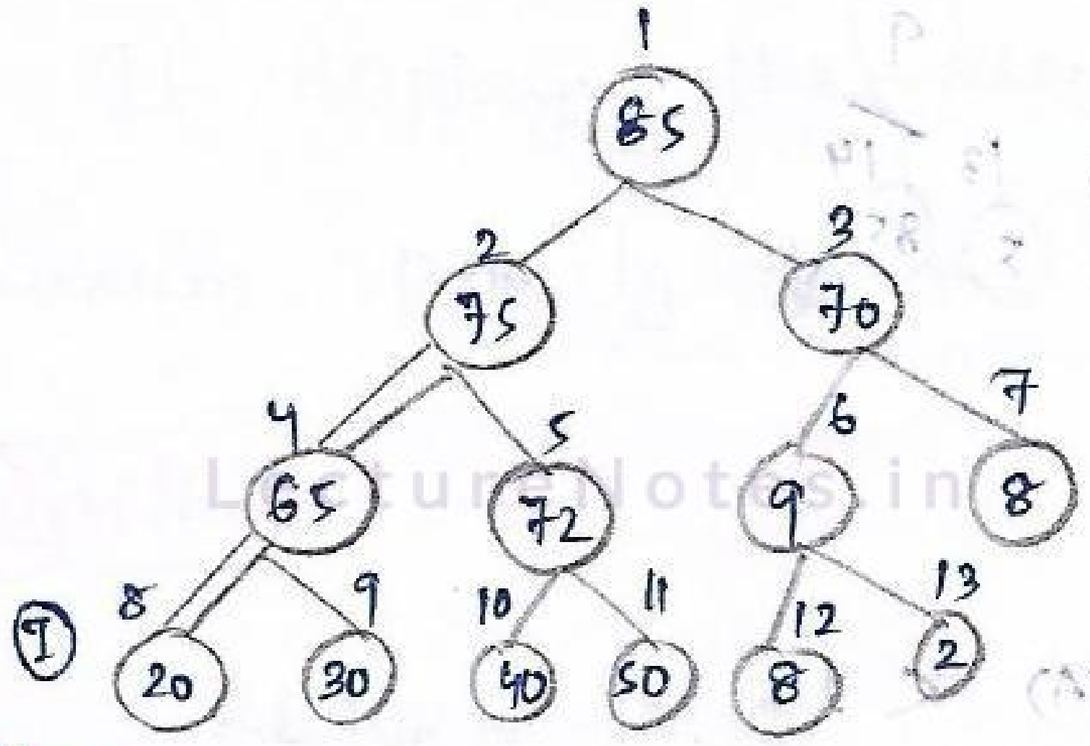
Therefore, the Time Complexity for HEAP_EXTRACT_MAX(A) is

$$T(n) = O(\log n) + c, \quad c > 0$$

$$\boxed{T(n) = O(\log n)}$$

3. HEAP_INCREASE_KEY (A, I, KEY)

The purpose is to increase the key value of specific element.



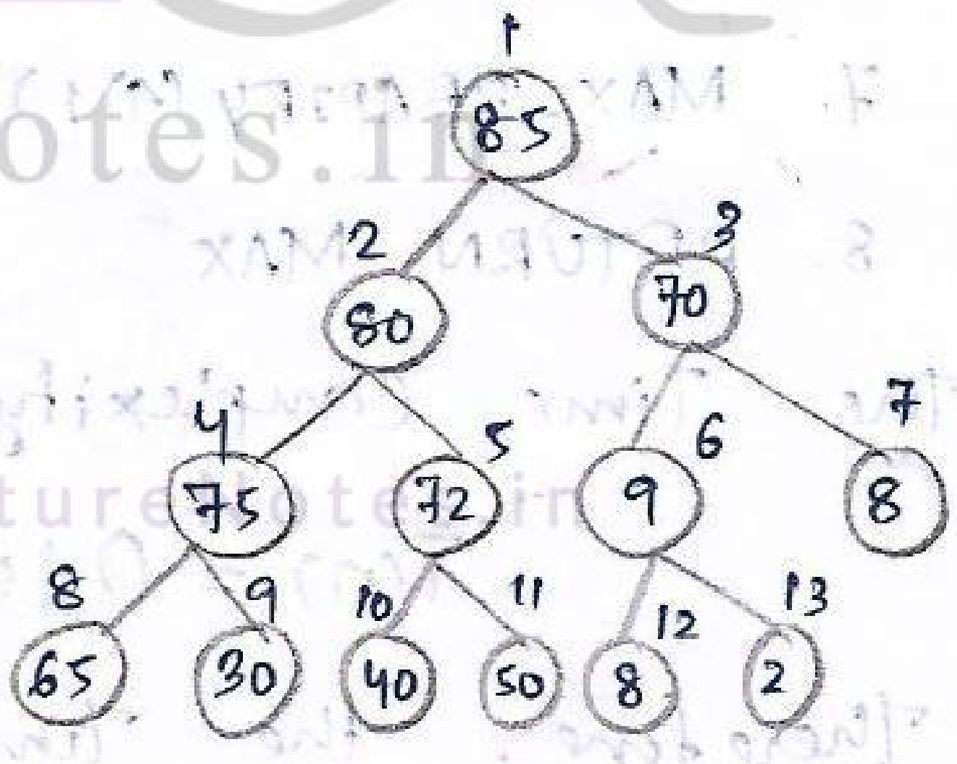
Algorithm -

/* A : Array
I : Index of the element whose key value is increased to the value KEY */

1. IF A[I] > KEY
2. DISPLAY "New key is smaller than old key"
3. RETURN

[END IF]

HEAP_INCREASE_KEY (A, 8, 80) →



4. A[I] ← KEY
5. WHILE I > 1 AND A[PARENT(I)] < A[I]
6. SWAP A[I] ⟷ A[PARENT(I)]
7. I ← PARENT(I)

[END WHILE]

the loop 5 will continue for log n (height of the tree) times.

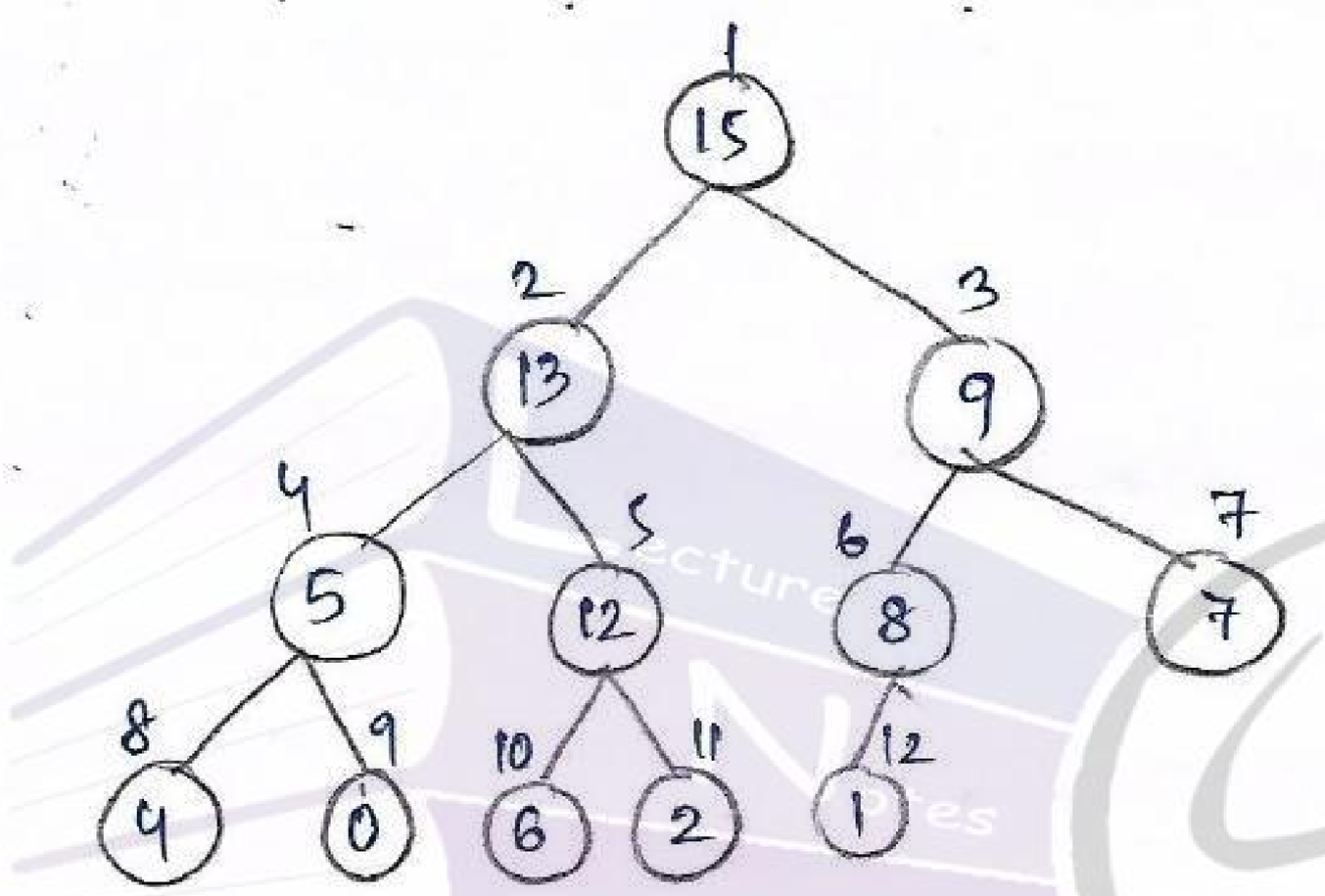
The Time complexity T(n) is

$$T(n) = O(\log n)$$

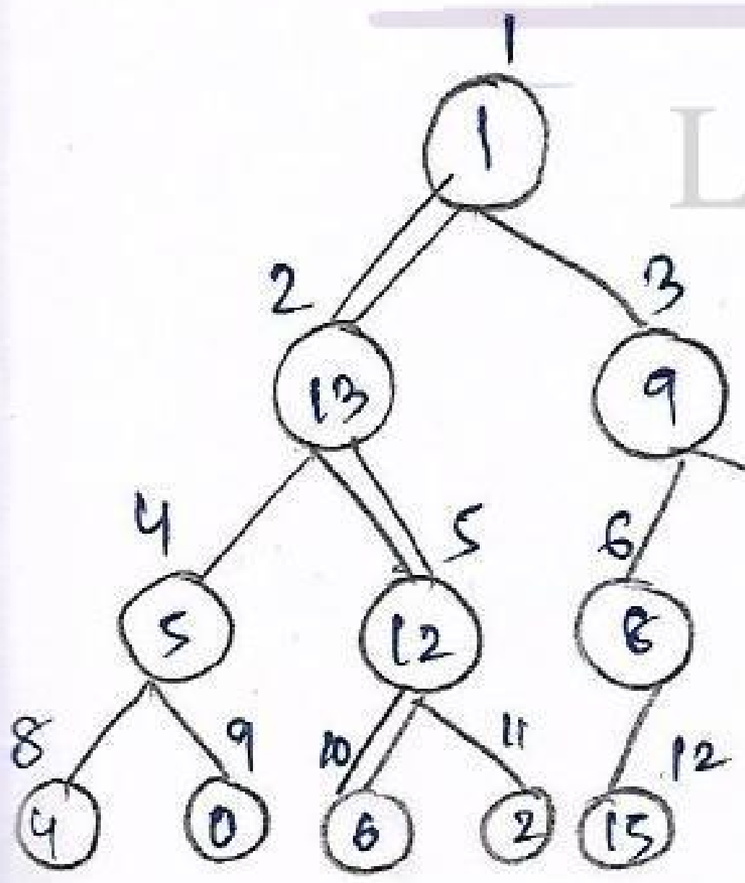
Q. Explain the operation of Heap Extractness on the heap -

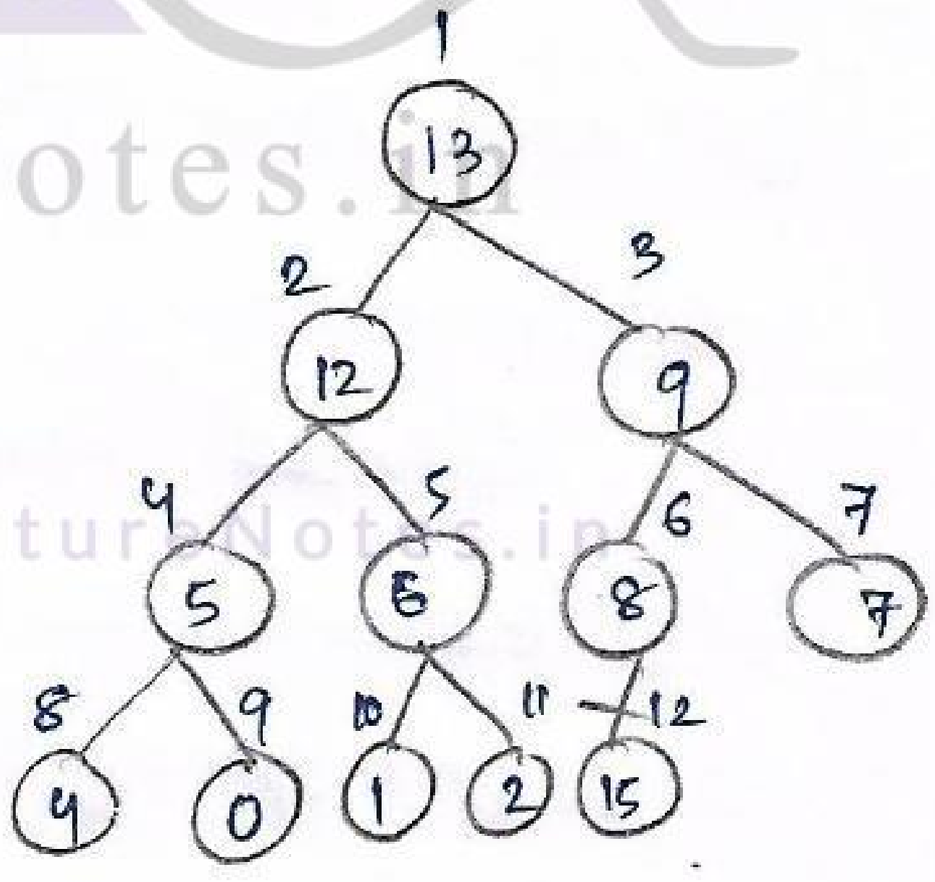
A = <15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1>

Sol.



HEAP_EXTRACT_MAX(A)





Dynamic Programming

* Dynamic Programming is an approach to find best solution among all solutions.

* It uses dividing approach in which the large problem is divided into sub-problems.

* The sub-problems are dependent to each other.

* The value of the sub-problem should be stored in Data Structure.

* Dynamic Programming is used to solve Optimization Problem (finding best solution among different solutions)

* Dynamic approach has following 4 steps.

1. Characterise the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of Optimal solution using bottom-up approach.
4. Construct an optimal solution from computed information.

LectureNotes.in

The following are the few problems where the Dynamic Programming is used →

1. Matrix Chain Multiplication
2. Longest Common Subsequence (LCS)
3. Assembly line Scheduling
4. Knapsack Problem.

1. Matrix Chain Multiplication →

$$[A]_{p \times q} \times [B]_{q \times r} = [M]_{p \times r}$$

The time complexity to find the multiplication of two matrix without using Matrix Chain Algorithm is

$$T(n) = \theta(n^3)$$

Q. Find the number of internal multiplication required to multiply the matrix -

$$[A]_{2 \times 3} \times [B]_{3 \times 5} =$$

Sol. $[A]_{2 \times 3} \times [B]_{3 \times 5} = p \times q \times r = 2 \times 3 \times 5 = 30$

$p \; q \qquad q \; r$

Q. Let A is a matrix of order $[A]_{10 \times 50}$, $[B]_{50 \times 100}$, $[C]_{100 \times 5}$. Multiply ABC. Find internal multiplication required for A, B and C.

Sol. * Finding internal multiplication for $[AB] \times C$

$$[AB]_{10\times100} \times [C]_{100\times5} = [ABC]_{10\times5}$$

* $[A]_{10\times50}\ [BC]_{50\times5}$

(a) Internal multiplication required for

$$[A]_{10\times50} \times [B]_{50\times100} = [AB]_{10\times100} = 10\times50\times100 = 50000$$

Then,

$$[AB]_{10\times100} \times [C]_{100\times5} = 10\times100\times5 = 5000$$

Total internal multiplication is

$$[ABC]_{10\times5} = 50000 + 5000 = 55000$$

(b) Internal multiplication required for

$$[B]_{50\times100} \times [C]_{100\times5} = 50\times100\times5 = 25000$$

Then,

$$[A]_{10\times50} \times [BC]_{50\times5} = 10\times50\times5 = 2500$$

Total internal multiplication is

$$[ABC]_{10\times5} = 25000 + 2500 = 27500$$

In above problem, it is clear that [A] x [BC] require less number of internal multiplications.
Thus, it is clear that the order of multiplication is important to find optimized value in Matrix Chain Multiplication.

Dt → 23rd Feb 18
Friday

Matrix Chain Multiplication Algorithm

* It defines the order of the multiplications so that the number of internal multiplication is less.
* It is solved using Dynamic Programming Approach.

1. Characterise the structure of an optimal solution.

	P_0	P_1	P_2	P_3	P_4
P	5	2	8	4	6
	0	1	2	3	4

$[A_1]_{5\times2}$ $P_0 \times P_1$ $\quad [A_2]_{2\times8}$ $P_1 \times P_2$ $\quad [A_3]_{8\times4}$ $P_2 \times P_3$ $\quad [A_4]_{4\times6}$ $P_3 \times P_4$

$$[A_1 A_2 A_3 A_4]_{5\times6} = [A_{I\ldots J}]_{P_0 \times P_4} = [A_{I\ldots J}]_{P_0 \times}$$

Order of.

$[A_1]_{P_0 \times P_1}$ $\quad [A_2]_{P_1 \times P_2}$ $\quad \ldots$ $\quad [A_I]_{P_{I-1} \times P_I}$

2. Find the recursive definition.

$[A_1 A_2 A_3 A_4]$

(i) $[A_1] \times [A_2 A_3 A_4]$ $K=1$

(ii) $[A_1 A_2] \times [A_3 A_4]$ $K=2$

(iii) $[A_1 A_2 A_3] \times [A_4]$ $K=3$

$A_{I..J} = A_{I...K} \times A_{K+1 ... J}$

$I \leq K < J$

Let $M[I...J]$ be the number of internal multiplication required to multiply the matrix from $A_{I...J}$

Let $M[I...k]$ be the number of internal multiplication required to multiply the matrix from $A_{I...K}$

Let $M[K+1 ... J]$ be the number of internal multiplication required to multiply the matrix from $A_{K+1 ... J}$

$M[I,...J]$: $A_{I..J}$

$M[I,K]$: $A_{I...K}$

$M[K+1, J]$: $A_{K+1...J}$

$A_{I...J} = A_{I...K} \times A_{K+1...J}$

$$M[I,J] = \begin{cases} 0 & \text{if } I=J \\ \underset{I \leq K < J}{MIN}(M[I,K] + M[K+1,J] + P_{I-1} \times P_K \times P_J) \end{cases}$$

5x2 2x8 8x4 4x6

$[A_1\ A_2\ A_3\ A_4]$

$[A_1\ A_2]_{5\times 8}$ × $[A_3\ A_4]_{8\times 6}$

p q r

$p \times q \times r$

$5 \times 8 \times 6$

$P_0\ P_2\ P_4$

$P_{I-1}\ P_K\ P_J$

$I = 1$

$K = 2$

$J = 4$

Q. Find an optimal parenthesis of a matrix chain products whose sequence of dimension is

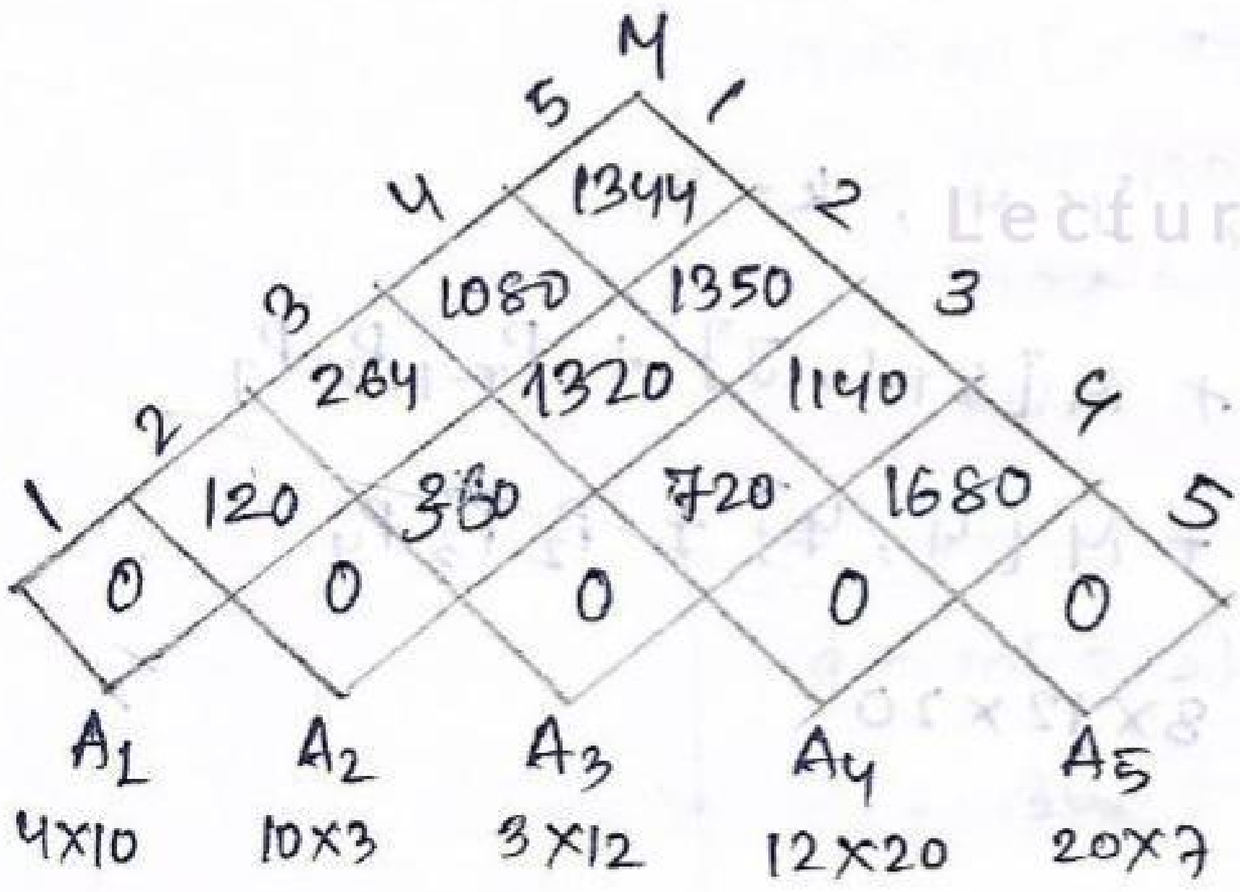
$< 4, 10, 3, 12, 20, 7 >$

Sol.

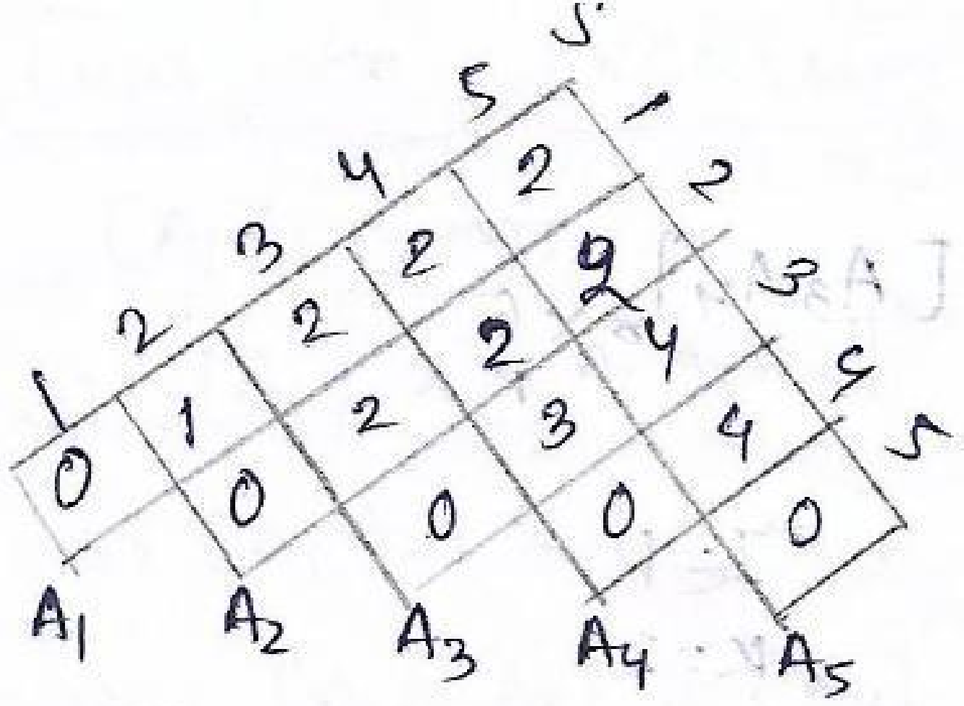
	0	1	2	3	4	5
P	4	10	3	12	20	7
	P_0	P_1	P_2	P_3	P_4	P_5

$[A_1]_{4\times 10}$ $[A_2]_{10\times 3}$ $[A_3]_{3\times 12}$ $[A_4]_{12\times 20}$ $[A_5]_{20\times 7}$

$[A_1\ A_2\ A_3\ A_4\ A_5]$



LectureNotes.in



Step 1 $L = 2$

(a) $[A_1 A_2]$, $I = 1$, $J = 2$, $K = 1$

$$q = M[I,K] + M[K+1,J] + P_0 P_1 P_2$$

$$= M[1,1] + M[2,2] + P_0 P_1 P_2$$

$$= 0 + 0 + 4 \times 10 \times 3$$

$$= 120$$

(b) $[A_2 A_3]$, $I = 2$, $J = 3$, $K = 2$

$$q = M[I,K] + M[K+1,J] + P_1 P_2 P_3$$

$$= M[2,2] + M[3,3] + 10 \times 3 \times 12$$

$$= 0 + 0 + 360$$

$$= 360$$

(c) $[A_3 A_4]$, $I = 3$, $J = 4$, $K = 3$

$$q = M[I,K] + M[K+1,J] + P_{I-1} P_K P_J$$

$$= M[3,3] + M[4,4] + P_2 P_3 P_4$$

$$= 0 + 0 + 3 \times 12 \times 20$$

$$= 720$$

(d) $[A_4 A_5]$; I=4 , J=5 , K=4

$q = M[I,K] + M[K+1, JJ + P_{I-1} P_k P_J$

$= M[4,4] + M[5, 5) + P_3 P_4 P_5$

$= 0 + 0 + 12 \times 20 \times 7$

$= 1680$

Step 2 $L = 3$

(a) $[A_1 A_2 A_3]$, I=1, J=3

$M[1,3] =$ {

K=1

$q = M[1,1] + m[2,3] + P_0 P_1 P_3$

$= 0 + 360 + 4 \times 10 \times 12$

$= 0 + 360 + 480$

$= 840$

K=2

$q = m[1,2] + m[3,3] + P_0 P_2 P_3$

$= 120 + 0 + 4 \times 3 \times 12$

$= 120 + 144$

$= 264$

(b) $[A_2 A_3 A_4]$, I=2 , J=4

$M[2,4] =$ {

K=2

$q = m[2,2] + m[3,4] + P_1 P_2 P_4$

$= 0 + 720 + 10 \times 3 \times 20$

$= 720 + 600$

$= 1320$

K=3

$q = m[2,3] + m[4,4] + P_1 P_3 P_4$

$= 360 + 0 + 10 \times 12 \times 20$

$= 360 + 2400$

$= 2760$

(c) $[A_3 A_4 A_5]$, I=3, J=5

$m[3,5]$ =

K=3

$q = m[3,3] + m[4,5] + P_2 P_3 P_5$

$= 0 + 1680 + 3 \times 12 \times 7$

$= 1680 + 252$

$= 1932$

K=4

$q = m[3,4] + m[5,5] + P_2 P_4 P_5$

$= 720 + 0 + 3 \times 20 \times 7$

$= 720 + 420$

$= 1140$

Step 3 L=4

(a) $[A_1 A_2 A_3 A_4]$, I=1, J=4

$m[1,4]$ =

K=1

$q = m[1,1] + m[2,4] + P_0 P_1 P_4$

$= 0 + 1320 + 4 \times 10 \times 20$

$= 1320 + 800$

$= 2120$

K=2

$q = m[1,2] + m[3,4] + P_0 P_2 P_4$

$= 120 + 720 + 4 \times 3 \times 20$

$= 840 + 240$

$= 1080$

K=3

$q = m[1,3] + m[4,4] + P_0 P_3 P_4$

$= 264 + 0 + 4 \times 12 \times 20$

$= 264 + 960$

$= 1224$

LectureNotes.in

(b) $[A_2 A_3 A_4 A_5]$, $I=2$, $J=5$

$m[2,5] =$

$\underline{K=2}$

$q = m[2,2] + m[3,5] + P_1 P_2 P_5$

$= 0 + 1140 + 10 \times 3 \times 7$

$= 1140 + 210$

$= 1350$

$\underline{K=3}$

$q = m[2,3] + m[3,5] + P_2 P_3 P_5$

$= 360 + 1140 + 3 \times 12 \times 7$

$= 1500 * 252$

$= 1752$

$\underline{K=4}$

$q = m[2,4] + m[5,5] + P_1 P_4 P_5$

$= 1320 + 0 + 10 \times 20 \times 7$

$= 1320 + 1400$

$= 2720$

Step 4 $I=5$

$[A_1 A_2 A_3 A_4 A_5]$, $I=1$, $J=5$

$m[1,5] =$

$\underline{K=1}$

$q = m[1,1] + m[2,5] + P_0 P_1 P_5$

$= 0 + 1350 + 4 \times 10 \times 7$

$= 1350 + 280$

$= 1630$

$\underline{K=2}$

$q = m[1,2] + m[3,5] + P_0 P_2 P_5$

$= 120 + 1140 + 4 \times 3 \times 7$

$= 1260 + 84$

$k=3$

$q = m[1,3] + m[4,5] + P_0 P_3 P_5$

$= 264 + 1680 + 4 \times 12 \times 7$

$= 1944 + 336$

$= 2280$

$k=4$

$q = m[1,4] + m[5,5] + P_0 P_4 P_5$

$= 1080 + 0 + 4 \times 20 \times 7$

$= 1080 + 560$

$= 1640$

The minimum no. of multiplication required to solve the matrix is 1344.

LectureNotes.in

3. Compute the optimal cost.

The procedure follows following two auxiliary 2D array.

m[1...n...1..N]: It stores the optimal value of

m[I, J]

S[1...N ...1...N]: It stores the value of K for which m[1, J] is optimum.

Algorithm →

```
MATRIX_CHAIN_ORDER(P)
/* P is an array which has the dimension
   of matrix */
1.  N ← LENGTH[P] - 1
2.  FOR  I ← 1 to N
3.          m[I, I] ← 0
4.          S[I, I] ← 0
    [END FOR]
5.  FOR  L ← 2 to N
6.      FOR  I ← 1  to  N-L+1
7.          J ← I+L-1
8.          M[I, J] = ∞
9.          FOR  K ← I  to  J-1
10.             q ← m[I,K] + m[K+1, J] + P_{I-1} P_K P_J
```

LectureNotes.in

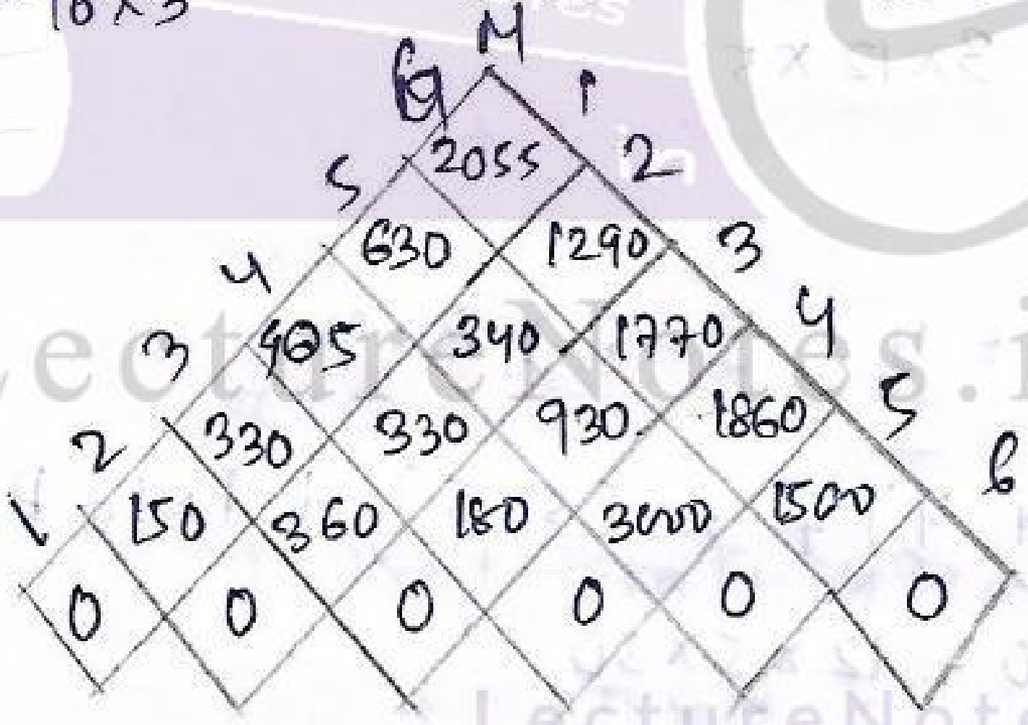
```
11.            IF (q < m[I,J])
12.                m[I,J] ← q
13.                S[I,J] ← K
           [END IF]
      [END FOR 9]
 [END FOR 6]
[END FOR 5]
14. RETURN m, S
```

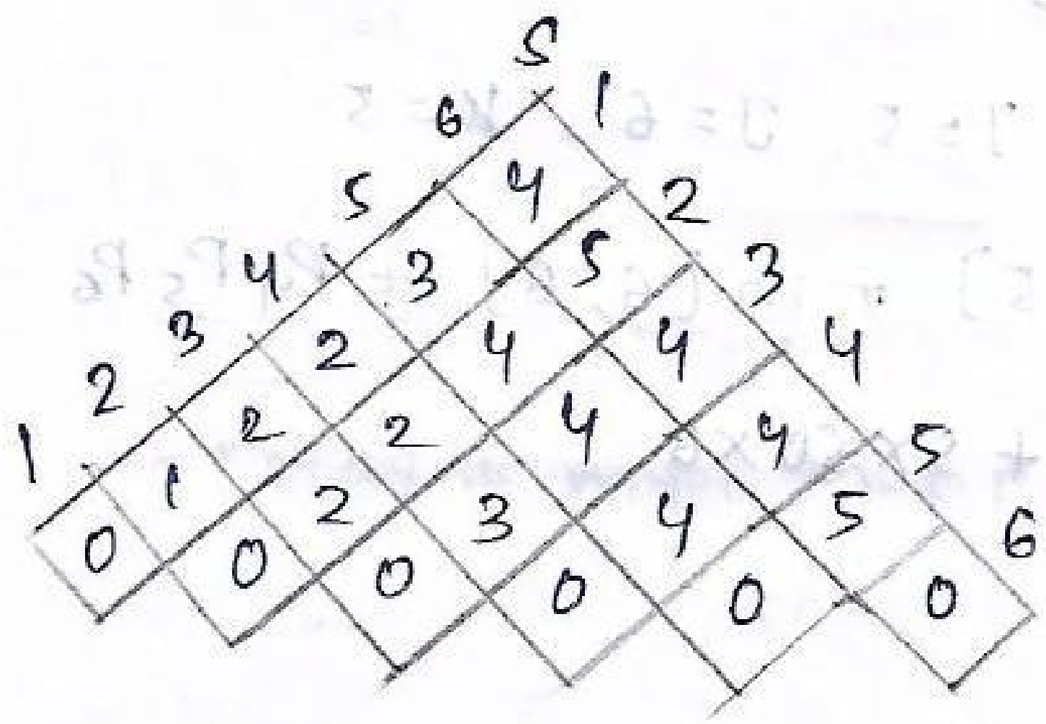
Q. Find an optimal parantheses of matrix chain product whose sequence of dimension is $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$.

Sol.

	0	1	2	3	4	5	6
P	5	10	3	12	5	50	6

$[A_1]_{5\times10}$ $[A_2]_{10\times3}$ $[A_3]_{3\times12}$ $[A_4]_{12\times5}$ $[A_5]_{5\times50}$ $[A_6]_{50\times6}$





Step 1 $L=2$

(a) $[A_1 A_2]$, $I=1$, $J=2$, $K=1$

$q = m[I,k] + m[k+1, J] + P_0 P_1 P_2$

$= m[1,1] + m[2,2] + 5 \times 10 \times 3$

$= 0 + 0 + 150$

$= 150$

(b) $[A_2, A_3]$, $I=2$, $J=3$, $K=2$

$q = m[2,2] + m[3,3] + P_1 P_2 P_3$

$= 0 + 0 + 10 \times 3 \times 12$

$= 360$

(c) $[A_3, A_4]$, $I=3$, $J=4$, $K=3$

$q = m[3,3] + m[4,4] + P_2 P_3 P_4$

$= 0 + 0 + 3 \times 12 \times 5$

$= 180$

(d) $[A_4, A_5]$, $I=4$, $J=5$, $K=4$

$q = m[4,4] + m[5,5] + P_3 P_4 P_5$

$= 0 + 0 + 12 \times 5 \times 50$

$= 3000$

(e) $[A_5, A_6]$, $I=5$, $J=6$, $K=5$

$q = m[5,5] + m[6,6] + P_4 P_5 P_6$

$= 0 + 0 + 5 \times 50 \times 6$

$= 1500$

LectureNotes.in

Step 2 $L = 3$

(a) $[A_1 A_2 A_3]$, $I = 1$, $J = 3$

$m[1,3] =$ $\underline{K=1}$

$= m[1,1] + m[2,3] + P_0 P_1 P_3$

$= 0 + 360 + 5 \times 10 \times 12$

$= 960$

$\underline{K=2}$

$= m[1,2] + m[3,3] + P_0 P_2 P_3$

$= 150 + 0 + 5 \times 3 \times 12$

$= 330$

(b) $[A_2 A_3 A_4]$, $I = 2$, $J = 4$

$m[2,4]$ $\underline{K=2}$

$= m[2,2] + m[3,4] + P_1 P_2 P_4$

$= 0 + 180 + 10 \times 3 \times 5$

$= 330$

$\underline{K=3}$

$= m[2,3] + m[4,4] + P_1 P_3 P_4$

$= 360 + 0 + 10 \times 12 \times 5$

$= 960$

(c) $[A_3 A_4 A_5]$, $I = 3$, $J = 5$

$m[3,5)$ $\underline{K=3}$

$= m[3,3] + m[4,5] + P_2 P_3 P_5$

$= 0 + 3000 + 3 \times 12 \times 50$

$= 4800$

LectureNotes.in

$k=4$

$= m[3,4] + m[5,5] + P_2 P_4 P_5$

$= 180 + 0 + 3 \times 5 \times 50$

$= 930$

(d) $[A_4 A_5 A_6]$, $I=4$, $J=6$

$m[4,6] =$ $\underline{k=4}$

$= m[4,4] + m[5,6] + P_3 P_4 P_6$

$= 0 + 1500 + 12 \times 5 \times 6$

$= 1860$

$\underline{k=5}$

$= m[4,5] + m[6,6] + P_3 P_5 P_6$

$= 3000 + 0 + 12 \times 50 \times 6$

$= 6000$

Step 3 $L=4$

(a) $[A_1 A_2 A_3 A_4]$, $I=1$, $J=4$

$m[1,4]$ $\underline{k=1}$

$= m[1,1] + m[2,4] + P_0 P_1 P_4$

$= 0 + 330 + 5 \times 10 \times 5$

$= 580$

$\underline{k=2}$

$= m[1,2] + m[3,4] + P_0 P_2 P_4$

$= 150 + 180 + 5 \times 3 \times 5$

$= 405$

$\underline{k=3}$

$= m[1,3] + m[4,4] + P_0 P_3 P_4$

(b) $[A_2 A_3 A_4 A_5]$, $I=2$, $J=5$

$m[2,5] =$ $\underline{K=2}$

$= m[2,2] + m[3,5] + P_1 P_2 P_3$

$= 0 + 930 + 10 \times 3 \times 50$

$= 2430$

$\underline{K=3}$

$= m[2,3] + m[4,5] + P_1 P_3 P_5$

$= 360 + 3000 + 10 \times 12 \times 50$

$= 9360$

$\underline{K=4}$

$= m[2,4] + m[5,5] + P_1 P_4 P_5$

$= 330 + 0 + 2 \times 5 \times 6$

$= 390$

(c) $[A_3 A_4 A_5 A_6]$; $I=3$, $J=6$

$m[3,6]$ $\underline{K=3}$

$= m[3,3] + m[4,6] + P_2 P_3 P_6$

$= 0 + 1860 + 3 \times 12 \times 6$

$= 2076$

$\underline{K=4}$

$= m[3,4] + m[5,6] + P_2 P_4 P_6$

$= 180 + 1500 + 3 \times 5 \times 6$

$= 1770$

$\underline{K=5}$

$= m[3,5] + m[6,6] + P_2 P_5 P_6$

$= 930 + 0 + 3 \times 50 \times 6$

Step 4 $l = 5$

(a) $[A_1 A_2 A_3 A_4 A_5]$, $I = 1$, $J = 5$

$m[1,5]$ $\quad k = 1$

$= m[1,1] + m[2,5] + P_0 P_1 P_5$

$= 0 + 390 + 5 \times 10 \times 50$

$= 2890$

$k = 2$

$= m[1,2] + m[3,5] + P_0 P_2 P_5$

$= 150 + 930 + 5 \times 3 \times 30$

$= 1830$

$k = 3$

$= m[1,3] + m[4,5] + P_0 P_3 P_5$

$= 330 + 3000 + 5 \times 12 \times 50$

$= 6330$

$k = 4$

$= m[1,4] + m[5,5] + P_0 P_4 P_5$

$= 405 + 0 + 5 \times 5 \times 50$

$= 1655$

(b) $[A_2 A_3 A_4 A_5 A_6]$, $I = 2$, $J = 6$

$m[2,6]$ $\quad k = 2$

$= m[2,2] + m[3,6] + P_1 P_2 P_6$

$= 0 + 1770 + 10 \times 3 \times 6$

$= 1950$

$k = 3$

$= m[2,3] + m[4,6] + P_1 P_3 P_6$

$K=4$

$= m[2,4] + m[5,6] + P_1 P_4 P_6$

$= 330 + 1500 + 10 \times 5 \times 6$

$= 2136$

$K=5$

$= m[2,5] + m[6,6] + P_2 P_5 P_6$

$= 390 + 0 + 3 \times 50 \times 6$

$= 1290$

Step 3 $L=6$

$[A_1 A_2 A_3 A_4 A_5 A_6)$, $I=1$, $J=6$

$m[1,6]$ $K=1$

$= m[1,1] + m[2,6] + P_0 P_1 P_6$

$= 0 + 1290 + 5 \times 10 \times 6$

$= 1590$

$K=2$

$= m[1,2] + m[3,6] + P_0 P_2 P_6$

$= 150 + 1770 + 5 \times 3 \times 6$

$= 2010$

$K=3$

$= m[1,3] + m[4,6] + P_0 P_3 P_6$

$= 330 + 1860 + 5 \times 12 \times 6 = 2550$

$K=4$

$= m[1,4] + m[5,6] + P_0 P_4 P_6$

$= 405 + 1500 + 5 \times 5 \times 6 = 2055$

$K=5$

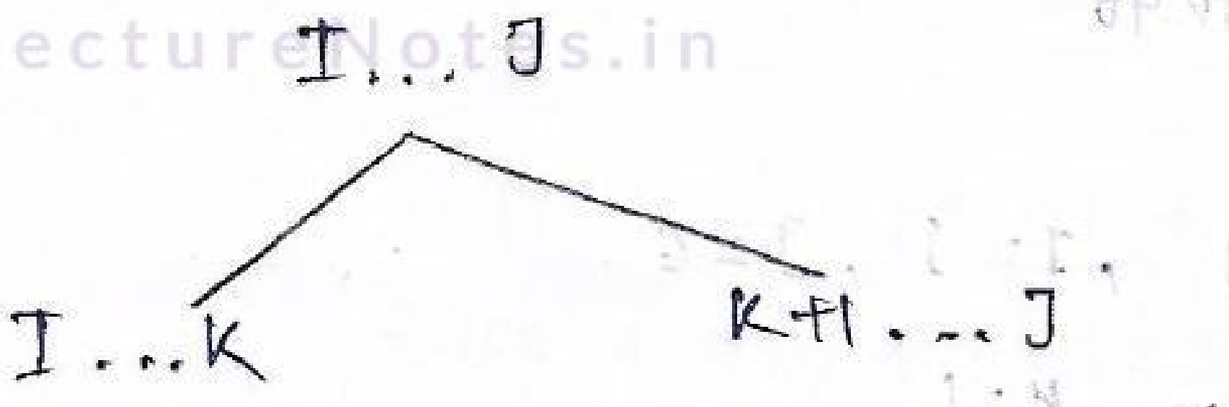
$= m[1,5] + m[6,6] + P_0 P_5 P_6$

$= 630 + 0 + 5 \times 50 \times 6$

Dt → 24th Feb 18
Saterday

4. Construct an Optimal Solution.

The matrix S can be used to find the order of multiplications in Matrix Chain Multiplication.



Alg ... →

PRINT_OPTIMAL-PARANTHESIS (S, I, J)

1. IF (I = J) THEN
2. PRINT A_I
3. ELSE
   PRINT '('
4. PRINT_OPTIMAL_PARANTHESIS (S, I, S[I, J])
5. PRINT_OPTIMAL_PARANTHESIS (S, S[I, J]+1, J)
6. PRINT ')'

[END IF]

LectureNotes.in

Finding Order of Multiplication

	I	J
Initial Call	1	5

1 — 2

3 — 5

3

Call	1	2

1 — 1

2 — 2

3

Call	1	1

4

Call	2	2

Call	3	5

3 — 4

5 — 5

7

Call	3	4

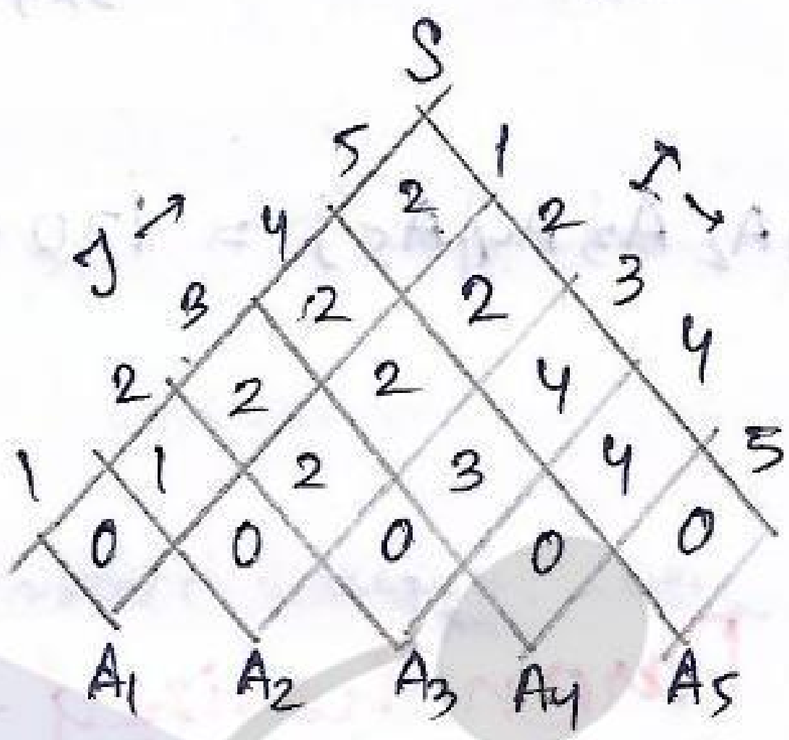
3 — 3

4 — 4

7

Output

$((A_1 A_2)(A_3 A_4) A_5))$



LectureNotes.in

$[A_1]_{4\times10} \times [A_2]_{10\times3} = 4\times10\times3 = 120 = [A_1A_2]_{4\times3}$

$[A_3]_{3\times12} \times [A_4]_{12\times20} = 3\times12\times20 = 720 = [A_3A_4]_{3\times20}$

$[A_3A_4]_{3\times20} \times [A_5]_{20\times7} = 3\times20\times7 = 420 = [A_3A_4A_5]_{3\times7}$

$[A_1A_2]_{4\times3} \times [A_3A_4A_5]_{3\times7} = 4\times3\times7 = 84$

$[A_1A_2A_3A_4A_5] = 120 + 720 + 420 + 84 = 1344$

Dt → 26th Feb 18
Monday

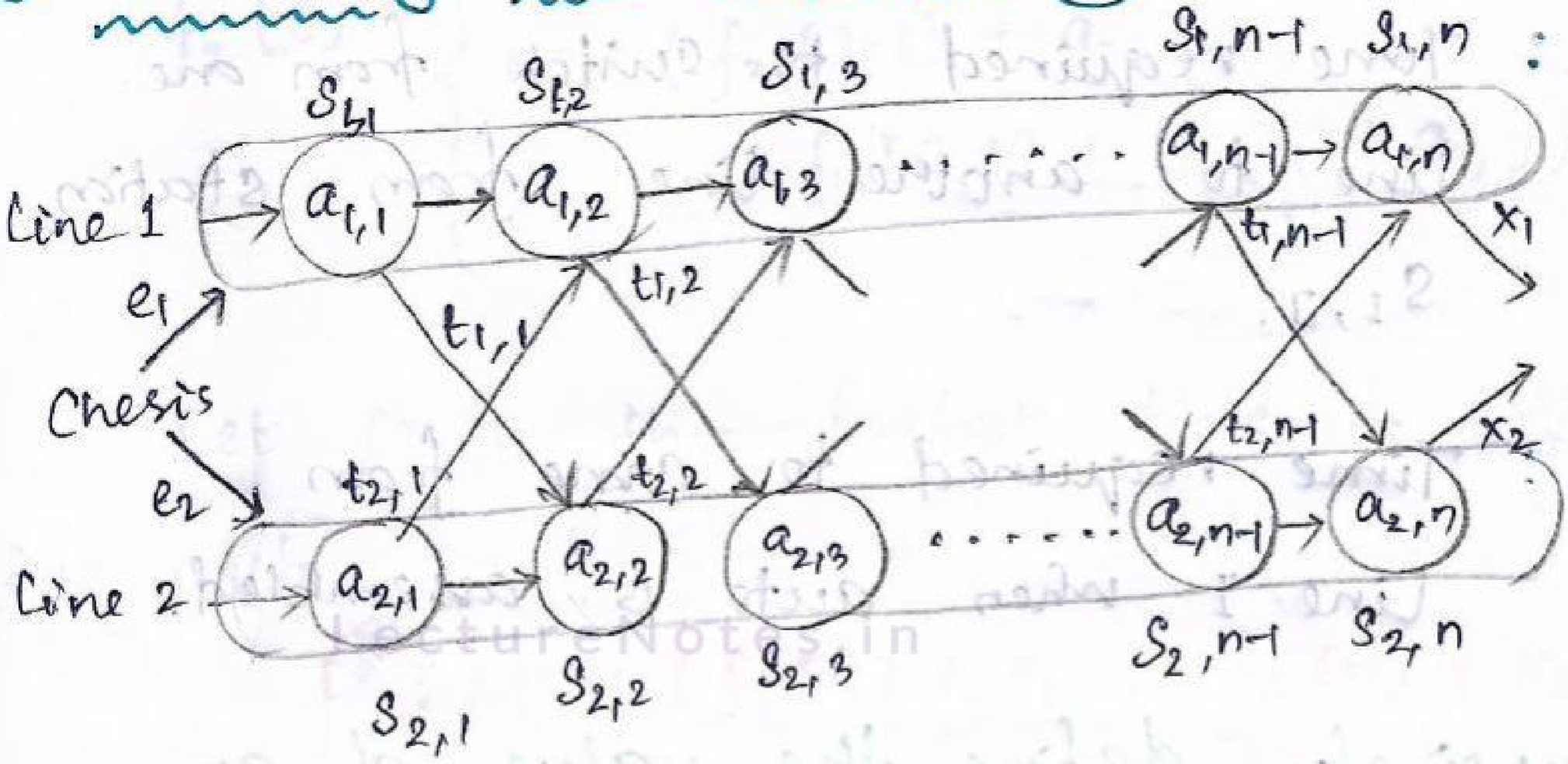
Fully Parenthesized →

1. A_1
2. $(A_1 \times A_2)$
3. $((A_1 \times A_2) \times A_3)$
4. $((A_1A_2)(A_3A_4))$

A product of matrices is fully parenthesized if it is either a single matrix or the product of two fully parenthesized matrix surrounded by parenthesis.

LectureNotes.in

3. Assembly Line Scheduling



* It is used for solving manufacturing problems to find fastest way to assemble the auto.
* The 4 steps of Dynamic approach are -
  1. Characterise the structure of an optimal solution.
  2. Recursively define the value of an optimal solution.
  3. Compute the value of optimal solution using Bottom-up approach.
  4. Construct an optimal solution from computed information.

1. Characterise the structure of optimal solution-

* Let there are two lines

$s_{I,J}$: It denotes line I and station J.

Let I = 1, 2

$a_{I,J}$: The time required at station

$s_{I,J}$

LectureNotes.in

to the assembly line I

$t_{I,J}$: Time required to switch from one line to another line from station $S_{I,J}$.

$X_{I,J}$: Time required to came from Line I when auto is assembled.

2. Recursively define the value of an optimal Solution.

Let $f_1[J]$ is the fastest time at Station $S_{1,J}$

$$f_1[J] = S_{1,J}$$

Let $f_2[J]$ is the fastest time at station $S_{2,J}$

$$f_2[J] = S_{2,J}$$

$$f_1[J] = \begin{cases} e_1 + a_{1,1}, & \text{if } J=1 \\ MIN(f_1[J-1] + a_{1,J}, \\ \quad f_2[J-1] + t_{2,J-1} + a_{1,J}), & \text{if } J>1 \end{cases}$$

$$f_2[J] = \begin{cases} e_2 + a_{2,1} & , \text{ if } J=1 \\ \text{MIN}\,(f_2[J-1] + a_{2,J}, & \\ \qquad f_1[J-1] + t_{1,J-1} + a_{2,J}) & , \text{ if } J>1 \end{cases}$$

Let f^* is the fastest time to assemble the auto.

$$f^* : \text{MIN}\,(f_1[n] + x_1 \;,\; f_2[n] + x_2)$$

3. Algorithm → (Compute the value of Optimal Solution

```
FASTEST_WAY (a, t, e, x, n)
/* a → two dimensional array which defines
       the time
   t → 2D array which defines the time to
       switch from one station to another.
   e → 1D array which have two values
       e1 and e2
   x → 1D array having two values, x1 and
       x2
   n → No. of stations */
1. f1[1] ← e1 + a1,1
2. f2[2] ← e2 + a2,1
3. FOR  J ← 2 to n
4.      IF  f1[J-1] + a1,J ≤ f2[J-1] + t2[J-1] + a1,J)
5.          f1[J] ← f1[J-1] + a1,J
```

6. $L_I[J] \leftarrow 1$

else

7. $f_1[J] \leftarrow f_2[J-1] + t_2[J-1] + a_1, a_{1,J}$

8. $L_I[J] \leftarrow 2$

[END IF]

9. IF ($f_2[J-1] + a_{2,J} \leq f_1[J-1] + t_1[J-1] + a_{2,J}$)

10. $f_2[J] \leftarrow f_2[J-1] + a_{2,J}$

11. $L_I[J] \leftarrow 2$

else

12. $f_2[J] \leftarrow f_1[J-1] + t_1[J-1] + a_{2,J}$

13. $L_I[J] \leftarrow 1$

[END IF]

[END FOR 3]

14. IF $f_1[n] + x_1 \leq f_2[n] + x_2$

15. $f^* = f_1[n] + x_1$

16. $l^* = 1$

else

17. $f^* = f_2[n] + x_2$

18. $l^* = 2$

[END IF]

LectureNotes.in

Q. Using Assembly line, find the fastest way to assemble the auto with the following data.

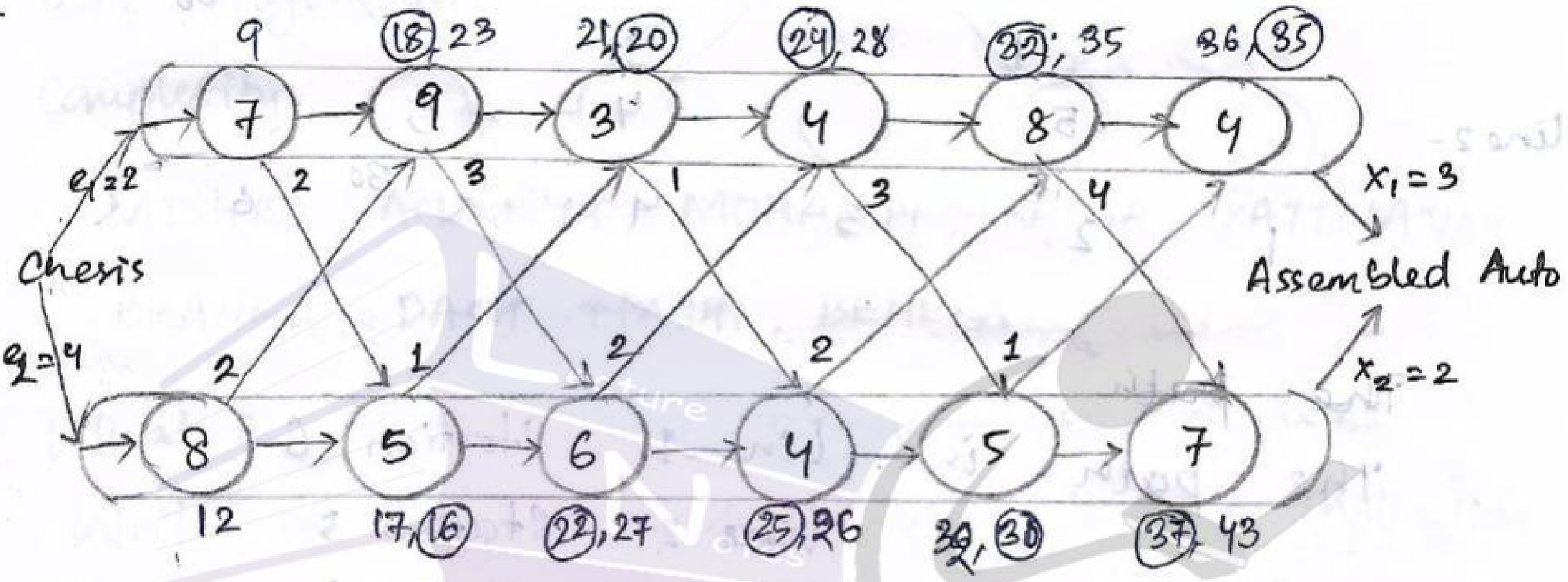
a	1	2	3	4	5	6
1	7	9	3	4	8	4
2	8	5	6	4	5	7

t	1	2	3	4	5
1	2	3	1	3	4
2	2	1	2	2	1

$e_1 = 2$ $\quad x_1 = 3$

$e_2 = 4$ $\quad x_2 = 2$

Sol.



f	1	2	3	4	5	6
1	9	18	20	24	32	35
2	12	16	22	25	30	37

L	2	3	4	5	6
1	1	2	1	1	2
2	1	2	1	2	2

If we exit from line 1, then the time is

$$35 + 3 = 38$$

If we exit from line 2, then time required is

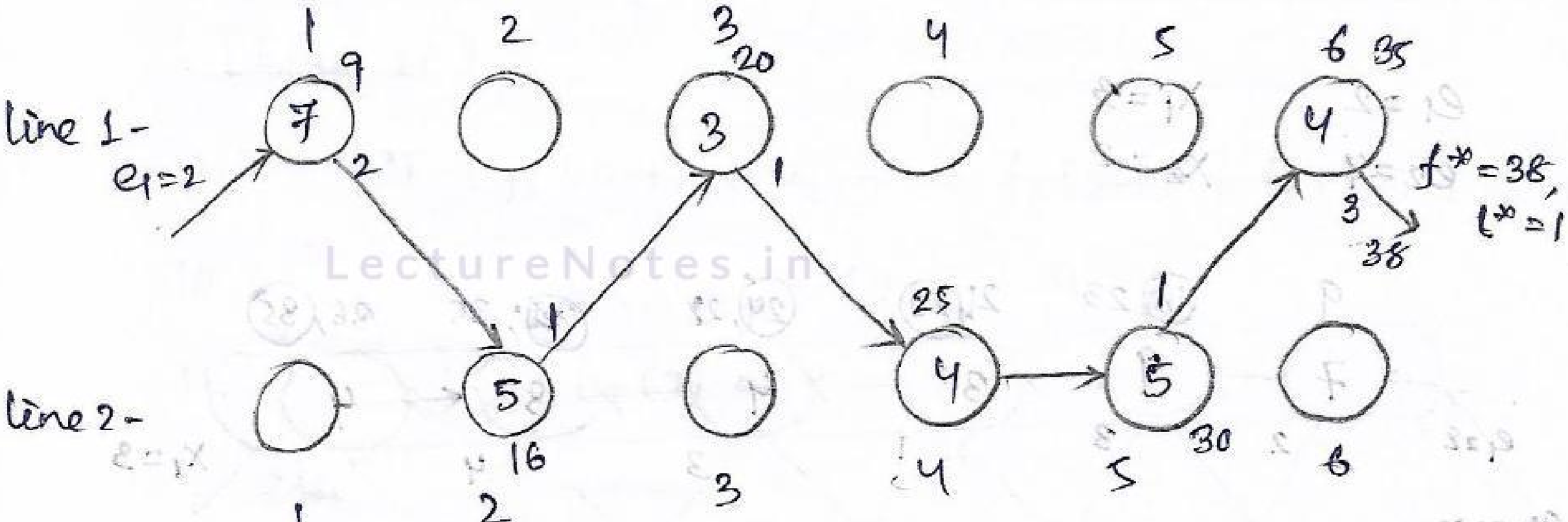
$$37 + 2 = 39$$

Therefore,

$$f^* = 38, \quad L^* = 1$$

4. Finding the path →

L matrix can be used to find the path.



The path is

Line 1	Station 6
Line 2	Station 5
Line 2	Station 4
Line 1	Station 3
Line 2	Station 2
Line 1	Station 1

4. Construct the fastest way or find the way or path.

```
PRINT_STATION (L, N)
/* L → 2D array where line no. is stored.
     N → No. of stations */
1.  I ← L*
2.  PRINT 'Line' I 'station' N
3.  I ← L*
4.  N ← N-1
5.  IF N = 0 THEN
        PRINT 'Line' I 'station' N
    ELSE
        GOTO 2
```

3. FOR J ← N DOWN 2

4. I ← L_I[J]

5. PRINT line 'I' station 'J-1'

[END FOR]

Time required, T(N) = Θ(N)

Q. Sort the given set of surnames using Heap-Sort Algorithm. Find its worst case time complexity.

MISHRA, ACHARYA, MOHANTY, RATHA, PATTNAYAK, BRAHMA, DASH, MAJHI, BEHERA.

Q. What is Matrix Chain Multiplication Problem. Write the algorithm for Matrix Chain Multiplication. Find m and s table computed by the algorithm for the following matrix dimensions -

Matrix	Dimensions
A_1	25 × 35
A_2	35 × 15
A_3	15 × 5
A_4	5 × 40

2. Longest Comman Subsequences

Ex → $X = \langle A, B, C, B, D, A, B \rangle$

Find the subsequence of set x.

Sol. (a) $\langle A, B, D \rangle$

(b) $\langle A, B, A \rangle$

(c) A

(d) $\langle A, B, A, B \rangle$

$\langle A, B, D, C \rangle$ is not a subsequence.

The sequence $Z = \{ z_1, z_2, z_3 \ldots z_k \}$ is a subsequence of $X = \langle x_1, x_2, x_3 \ldots x_n \rangle$ if there exist a strictly increasing sequence $\langle i_1, i_2, i_3 \ldots i \rangle$ of indices of x, such that, for all $J = 1, 2, 3 \ldots K$, we have

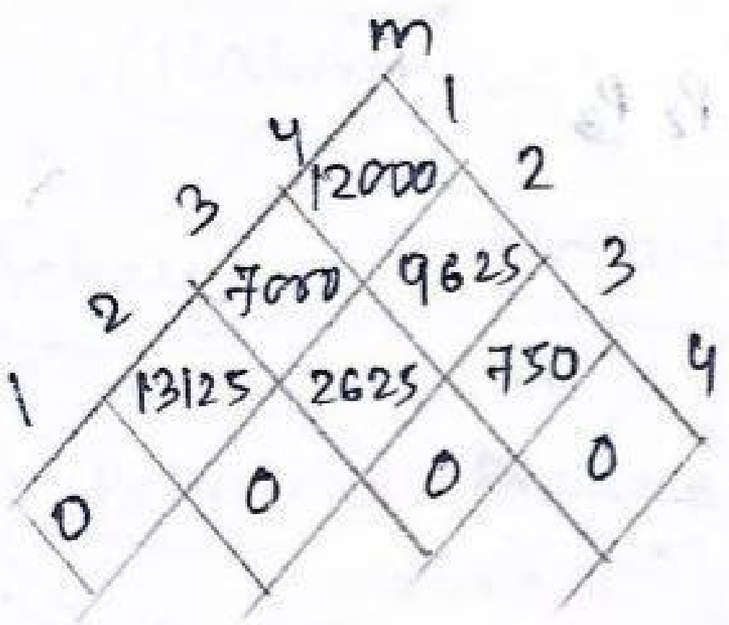
$$x_I = z_J$$

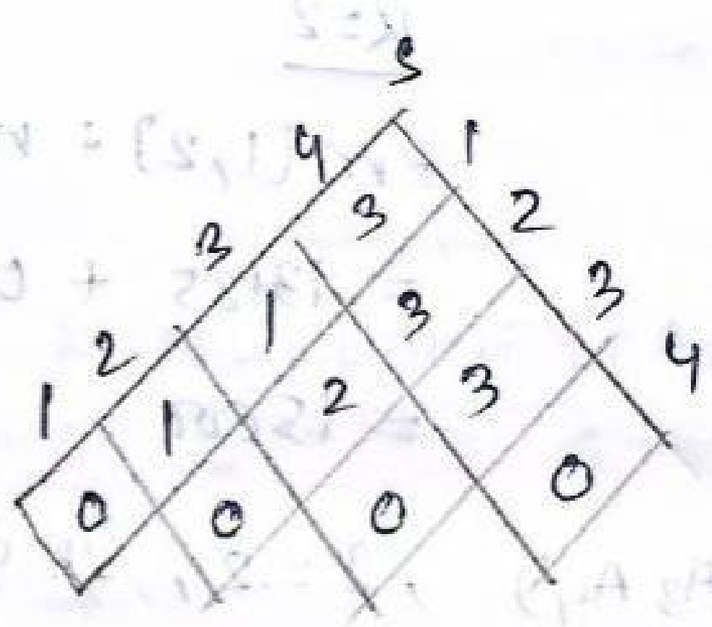
LectureNotes.in

Q.2 $[A_1]_{25 \times 35}$ $[A_2]_{35 \times 15}$ $[A_3]_{15 \times 5}$ $[A_4]_{5 \times 40}$

	0	1	2	3	4
P	25	35	15	5	40

$[A_1 A_2 A_3 A_4]$





Step 1

(a) $[A_1 A_2]$, $I=1$, $J=2$, $K=1$

$m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2$

$= 0 + 0 + 25 \times 35 \times 15$

$= 13125$

(b) $[A_2 A_3]$, $I=2$, $J=3$, $K=2$

$m[2,3] = m[2,2] + m[3,3] + P_1 P_2 P_3$

$= 0 + 0 + 35 \times 15 \times 5$

$= 2625$

(c) $[A_3 A_4]$, $I=3$, $J=4$, $K=3$

$m[3,4] = m[3,3] + m[4,4] + P_2 P_3 P_4$

$= 0 + 0 + 15 \times 5 \times 10$

$= 750$

Step 2

(a) $[A_1 A_2 A_3]$, $I=1$, $J=3$.

$K=1$

$m[1,3] = m[1,1] + m[2,3] + P_0 P_1 P_3$

$= 0 + 2625 + 25 \times 35 \times 5$

$= 7000$

$K=2$

$= m[1,2] + m[3,3] + P_0 P_2 P_3$

$= 13125 + 0 + 25 \times 15 \times 5$

$= 15000$

(b) $[A_2 A_3 A_4)$, $I = 2$, $J = 4$

$K=2$

$m[2,4] = m[2,2] + m[3,4] + P_1 P_2 P_4$

$= 0 + 750 + 35 \times 15 \times 40$

$= 21750$

$K=3$

$= m[2,3] + m[4,4] + P_1 P_3 P_4$

$= 2625 + 0 + 35 \times 5 \times 40$

$= 9625$

(c)

Step 3 : $[A_1 A_2 A_3 A_4]$, $I = 1$, $J = 4$

$K=1$

$m[1,4] = m[1,1] + m[2,4] + P_0 P_1 P_4$

$= 0 + 9625 + 25 \times 35 \times 40$

$= 44625$

$K=2$

$= m[1,2] + m[3,4] + P_0 P_2 P_4$

$= 13125 + 750 + 25 \times 15 \times 40$

$= 28875$

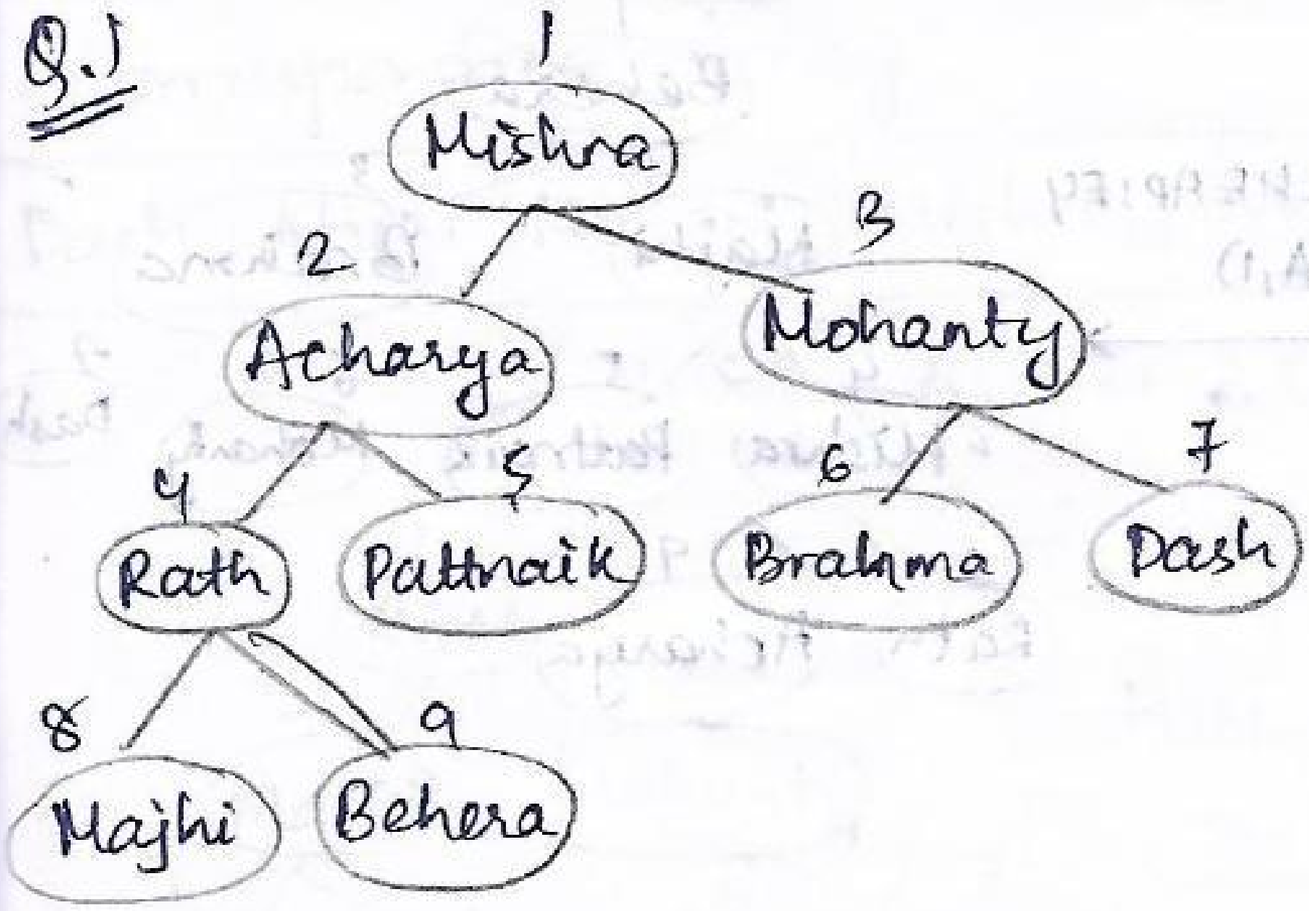
$K=3$

$= m[1,3] + m[4,4] + P_0 P_3 P_4$

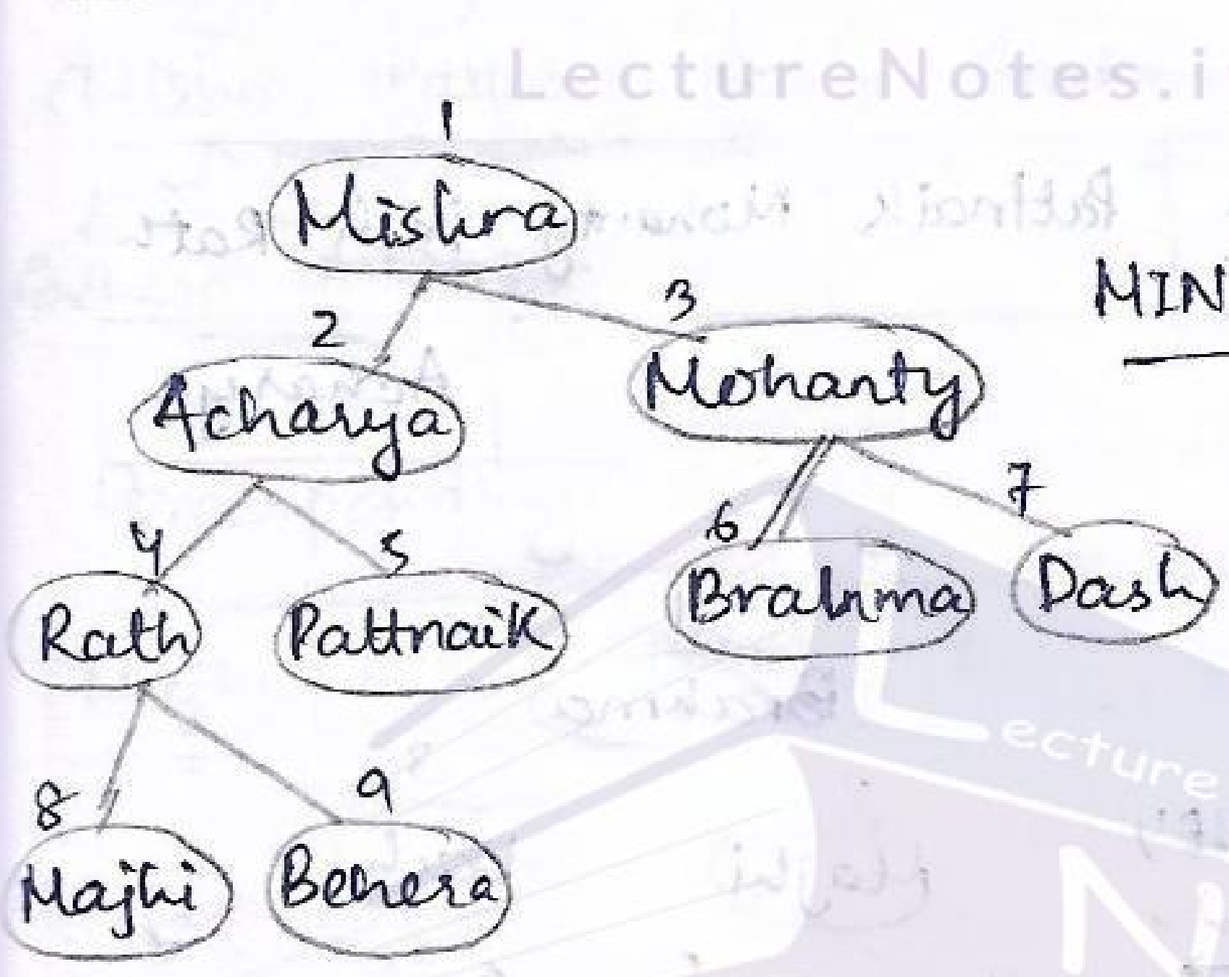
$= 7000 + 0 + 25 \times 5 \times 40$

$= 12000$

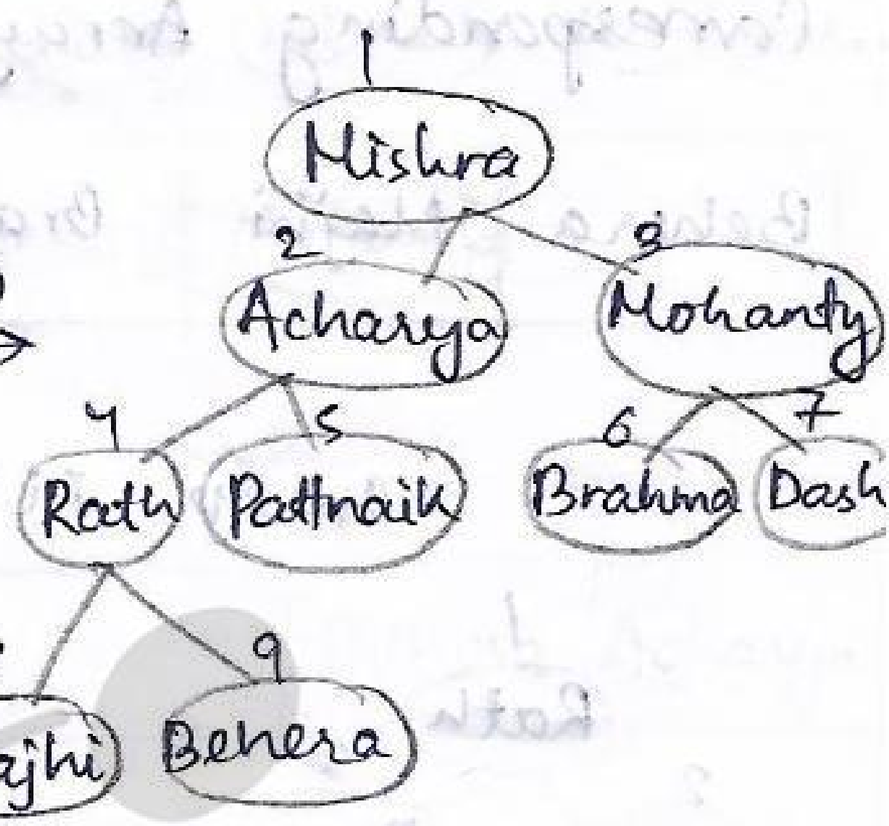
Q.1



MIN_HEAPIFY (A,4)

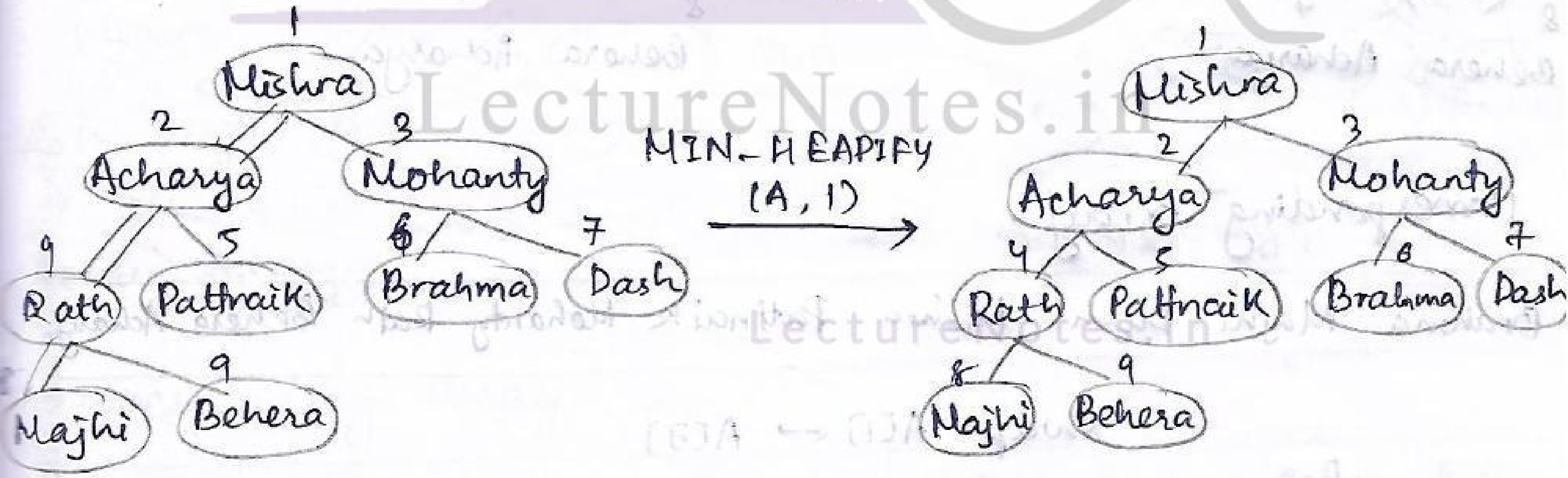


MIN_HEAPIFY (A,3)



MIN-HEAPIFY(A,2)

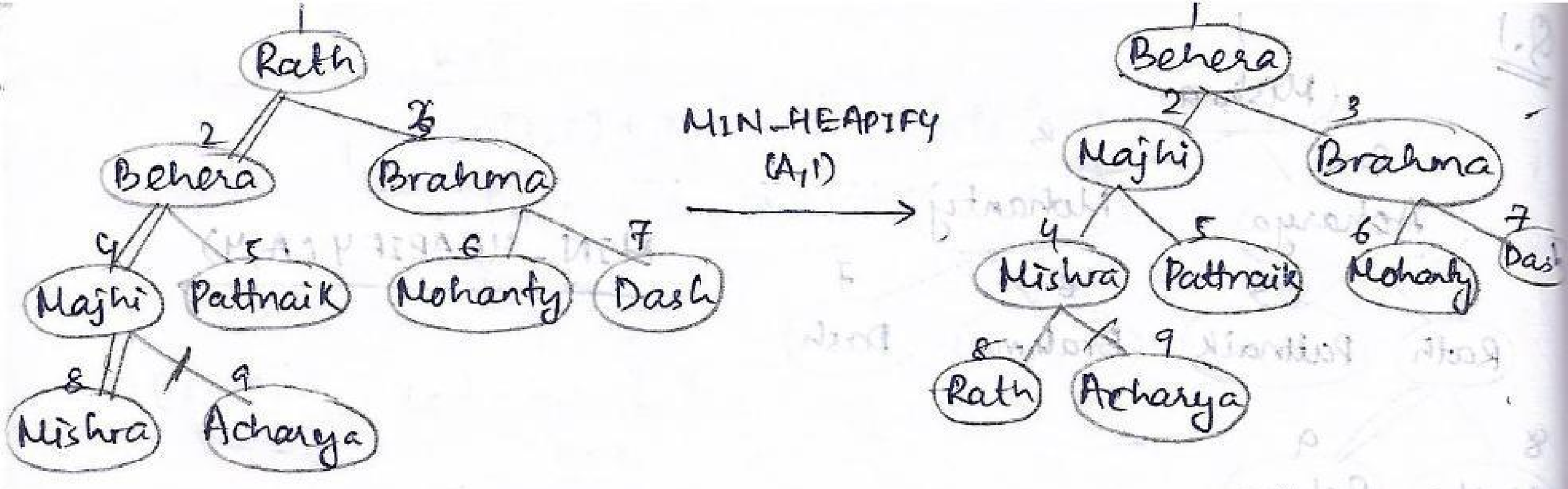
MIN-HEAPIFY (A,1)



Corresponding array →

Acharya	Behera	Brahma	Majhi	Pattnaik	Mohanty	Dash	Mishra	Rath

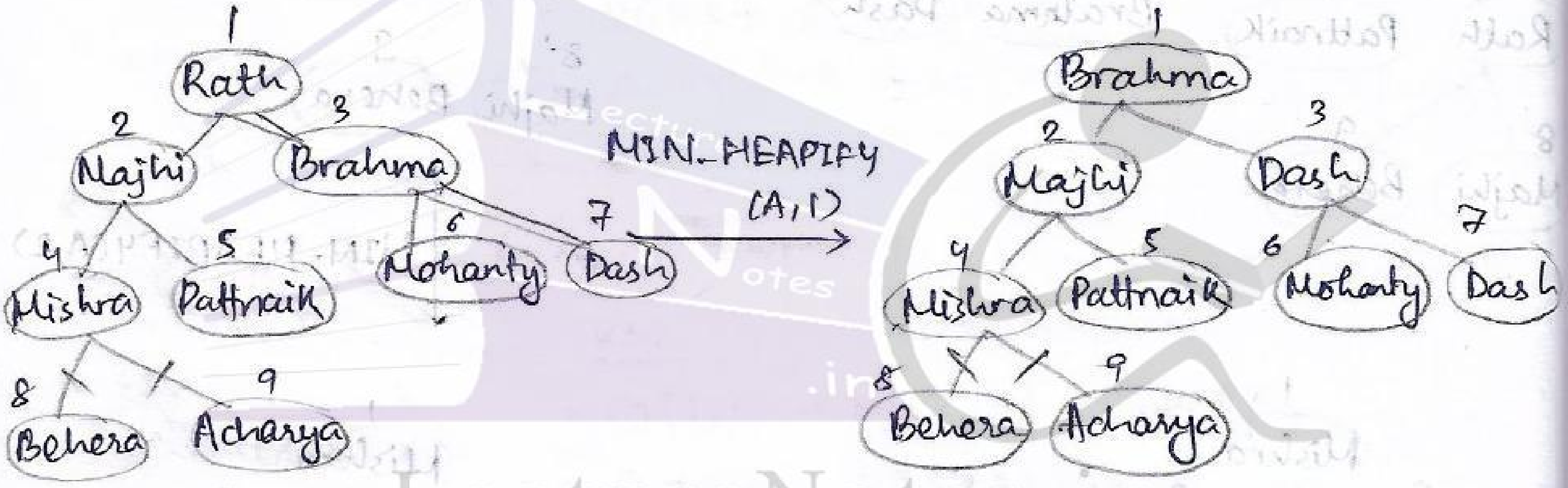
SWAP A[1] ⟷ A[9]



Corresponding Array →

Behera	Majhi	Brahma	Mishra	Pattnaik	Mohanty	Dash	Rath
						Acharya	

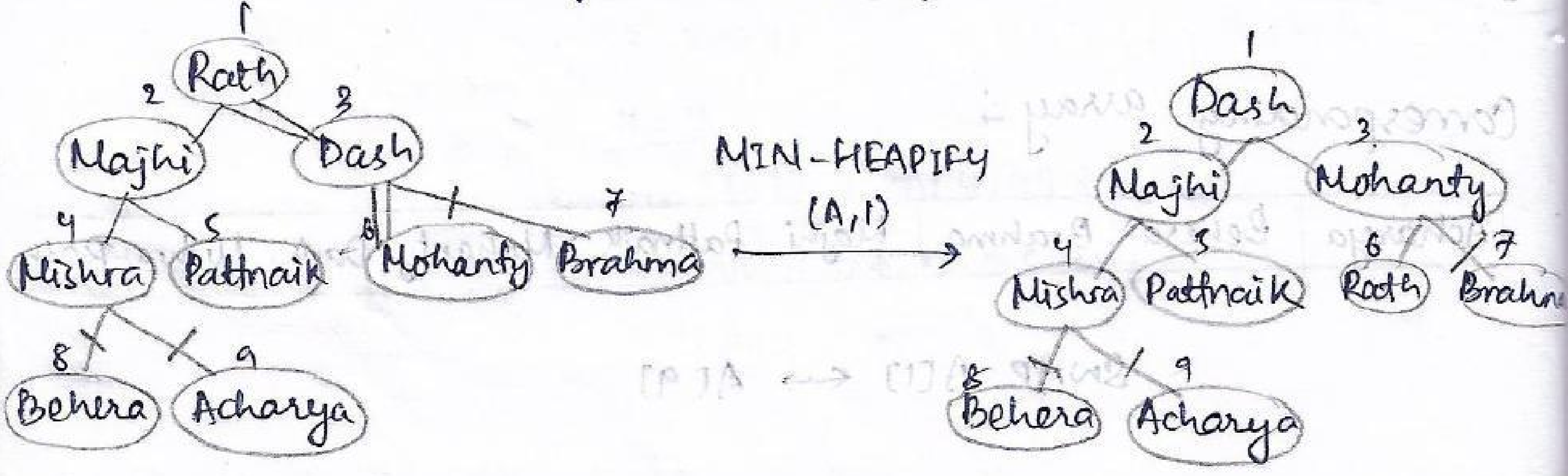
Swap A[1] ⟷ A[8]



Corresponding Array →

Brahma	Majhi	Dash	Mishra	Pattnaik	Mohanty	Rath	Behera	Acharya

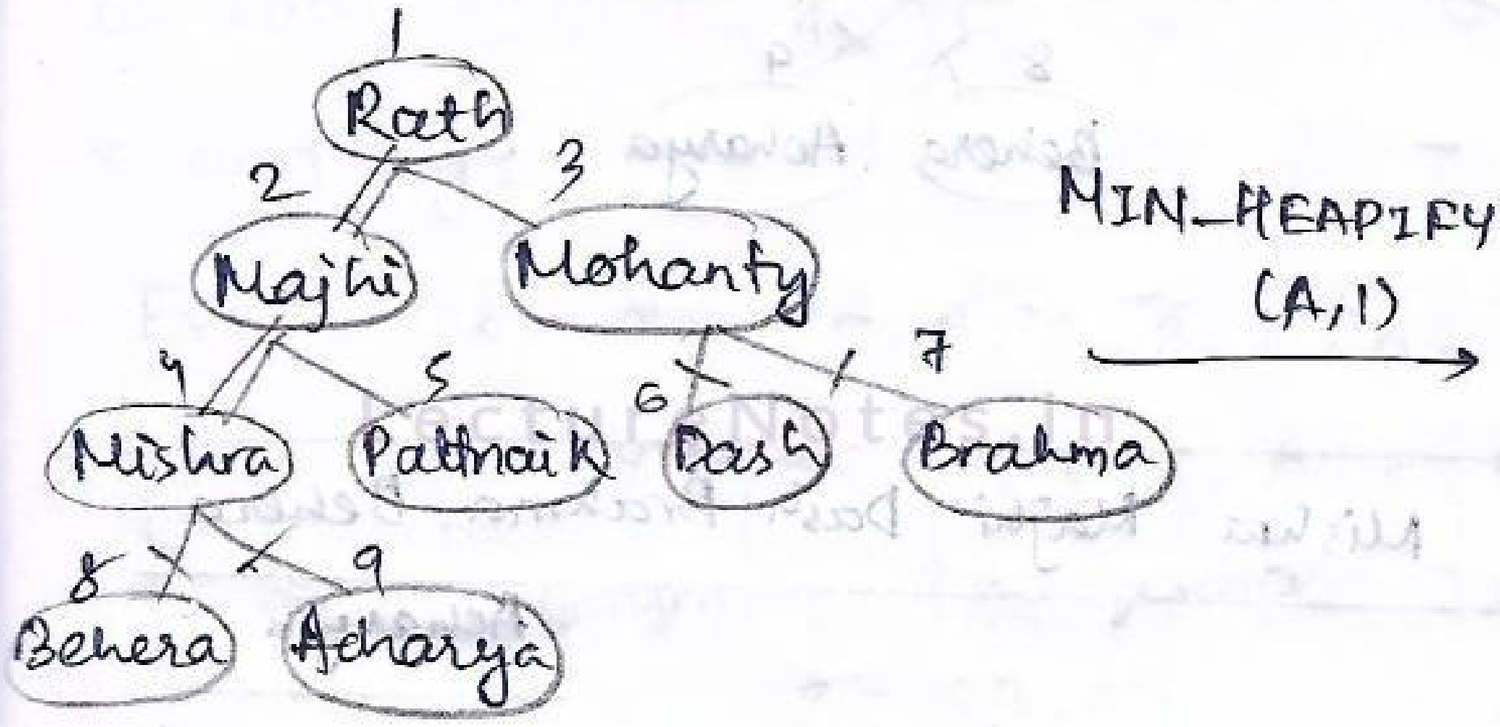
Swap A[1] ⟷ A[3]



Corresponding Array

Dash	Majhi	Mohanty	Mishra	Pattnaik	Rath	Brahma	Behera	Acharya

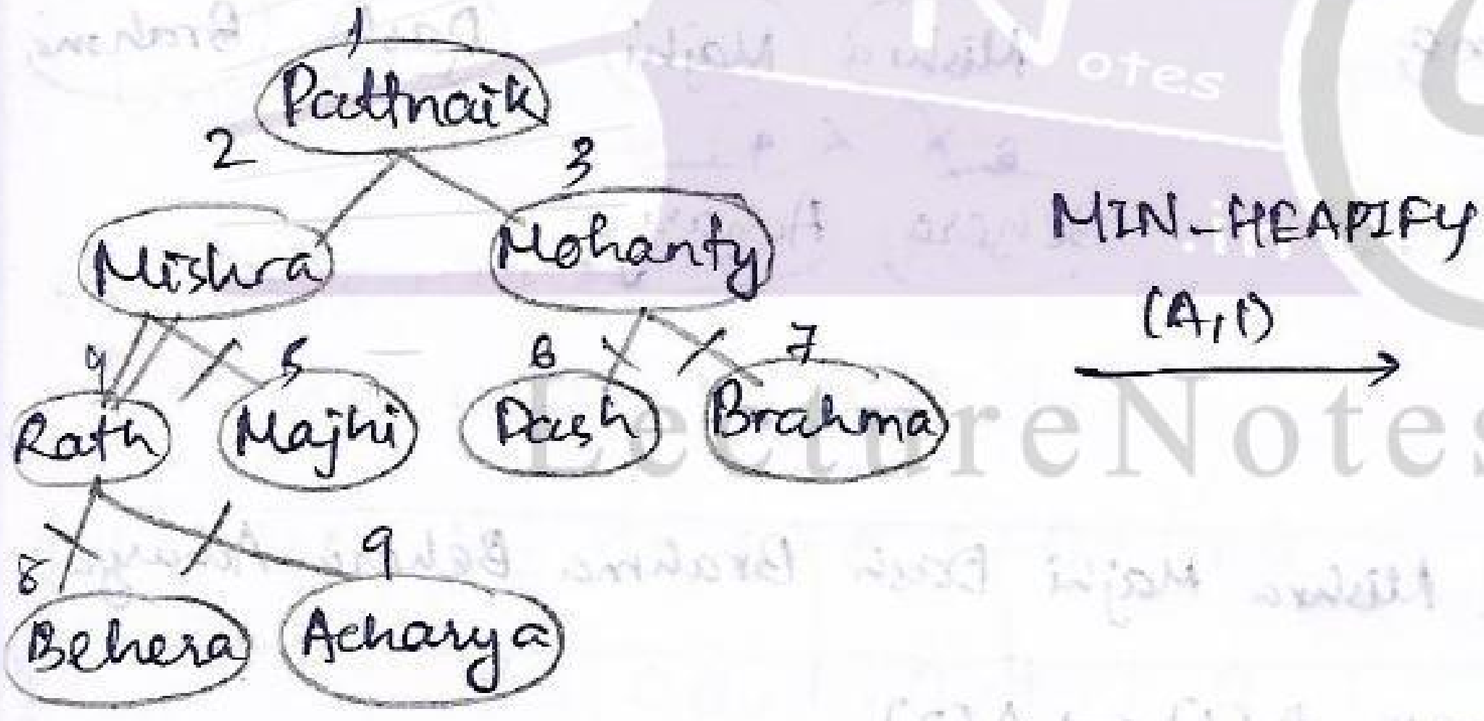
Swap A[1] ↔ A[6]



Corresponding array

Majhi	Mishra	Mohanty	Rath	Pattnaik	Dash	Brahma	Behera	Acharya

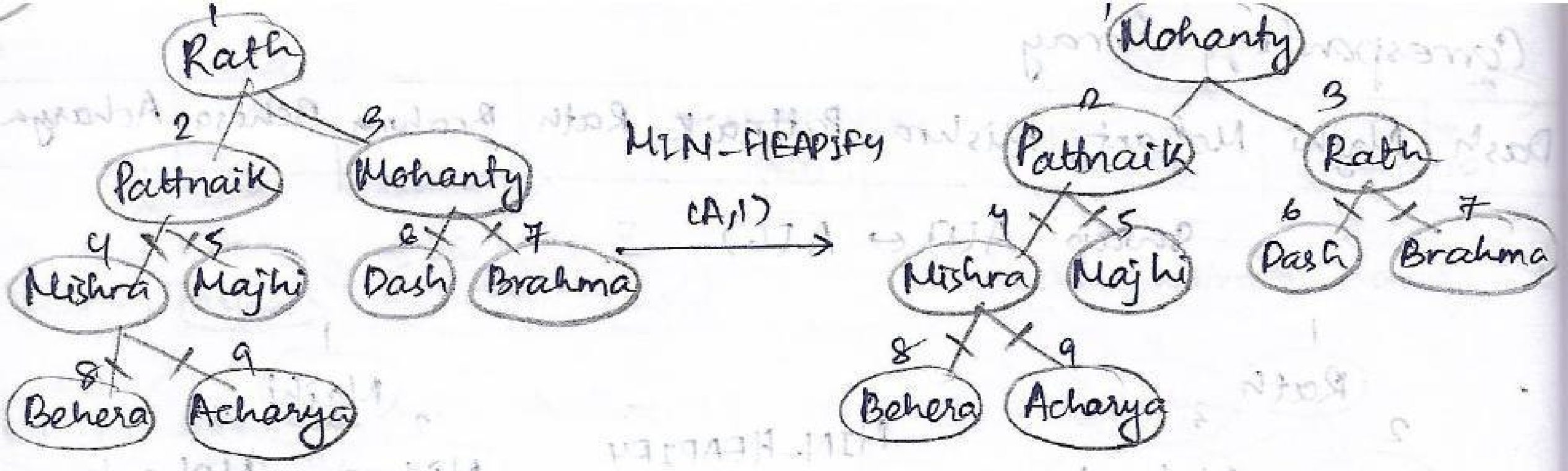
Swap A[1] ↔ A[5]



Corresponding array

Mishra	Pattnaik	Mohanty	Rath	Majhi	Dash	Brahma	Behera	Acharya

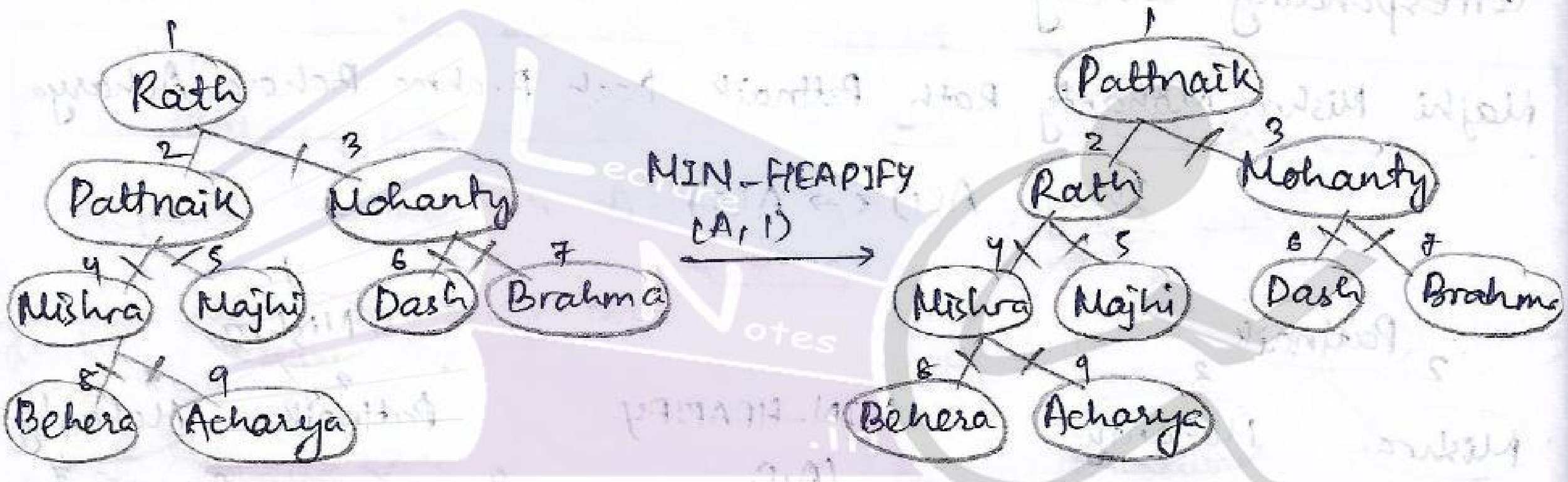
Swap A[1] ↔ A[4]



Corresponding array -

Mohanty	Pattnaik	Rath	Mishra	Majhi	Dash	Brahma	Behera
							Acharya

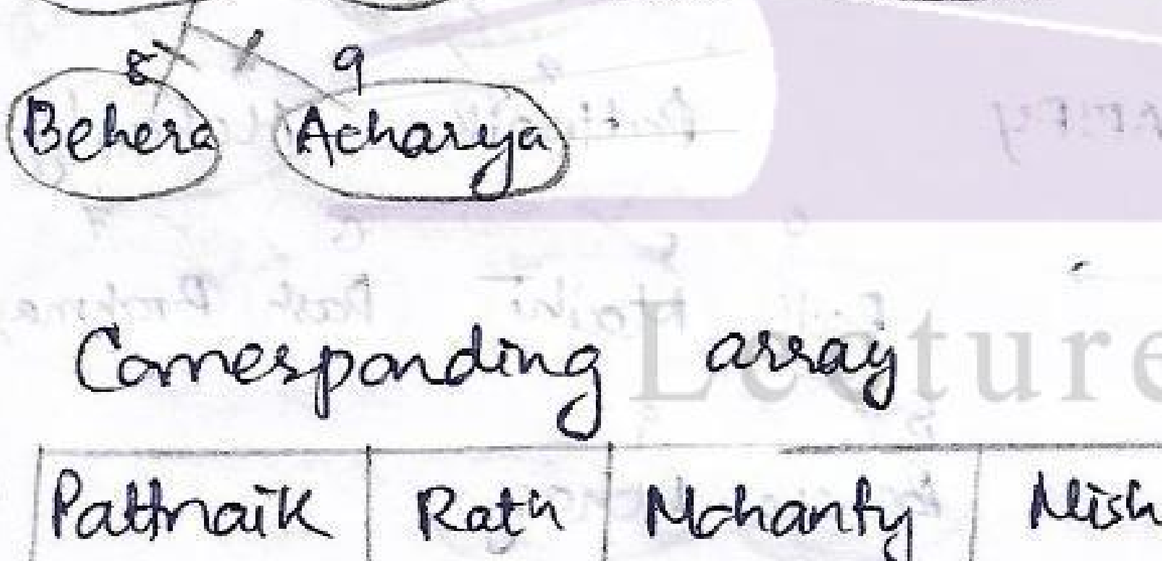
Swap A[1] ↔ A[3]



Corresponding array

Pattnaik	Rath	Mohanty	Mishra	Majhi	Dash	Brahma	Behera	Acharya

Swap A[1] ↔ A[2]



Required array →

Rath	Pattnaik	Mohanty	Mishra	Majhi	Dash	Brahma	Behera	Acharya

DE- 27th Feb 18
Tuesday

Common Subsequence

The sequence z is a common subsequence of both x and y if z is a subsequence of both x and y.

Ex - let $X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

(a) Common Subsequence

$Z = \langle B, C \rangle$

(b) $Z = \langle B, D, A, B \rangle$

Longest Common Subsequence (LCS)

The subsequence which has maximum length among all subsequence is called LCS.

Q. Determine the LCS of $\langle 1, 0, 0, 1, 0, 1, 0, 1 \rangle$ and $\langle 0, 1, 0, 1, 1, 0, 1, 1, 0 \rangle$

Sol. Let $X = \langle 1, 0, 0, 1, 0, 1, 0, 1 \rangle$ and

$Y = \langle 0, 1, 0, 1, 1, 0, 1, 1, 0 \rangle$

LectureNotes.in

J →
Y →

I ↓	X ↓	C	0 (0)	1 (1)	2 (0)	3 (1)	4 (1)	5 (0)	6 (1)	7 (1)	8 (0)	9
		0	0	0	0	0	0	0	0	0	0	0
(1)	1		0	0↑	1↖	1←	1↑	1↖	1←	1↖	1↑	1←
(0)	2		0	1↑	1↑	2↖	2←	2←	2↖	2←	2←	2↑
(0)	3		0	1↑	1↑	2↑	2↑	2↑	3↑	3←	3←	3↑
(1)	4		0	1↑	2↑	2↑	3↑	3↑	3↑	4↑	4↑	4←
(0)	5		0	1↑	2↑	3↑	3↑	3↑	4↑	4↑	4↑	5↑
(1)	6		0	1↑	2↑	3↑	4↑	4↑	4↑	5↑	5↑	5↑
(0)	7		0	1↑	2↑	3↑	4↑	4↑	5↑	5↑	5↑	6↑
(1)	8		0	1↑	2↑	3↑	4↑	5↑	5↑	6↑	6↑	6↑

Same = corner + 1

diff = largest (if same then, largest is upper one)

The length of the largest common sub-sequence is

$C[N_1][N_2] = 6$

The LCS is $\langle 1, 0, 0, 1, 1, 0 \rangle$

NOTE →

There may be another LCS of length 6.

Q. Find the LCS of the sets

$\langle A, B, C, D, B, A, C, D, F \rangle$ and

$\langle C, B, A, F \rangle$

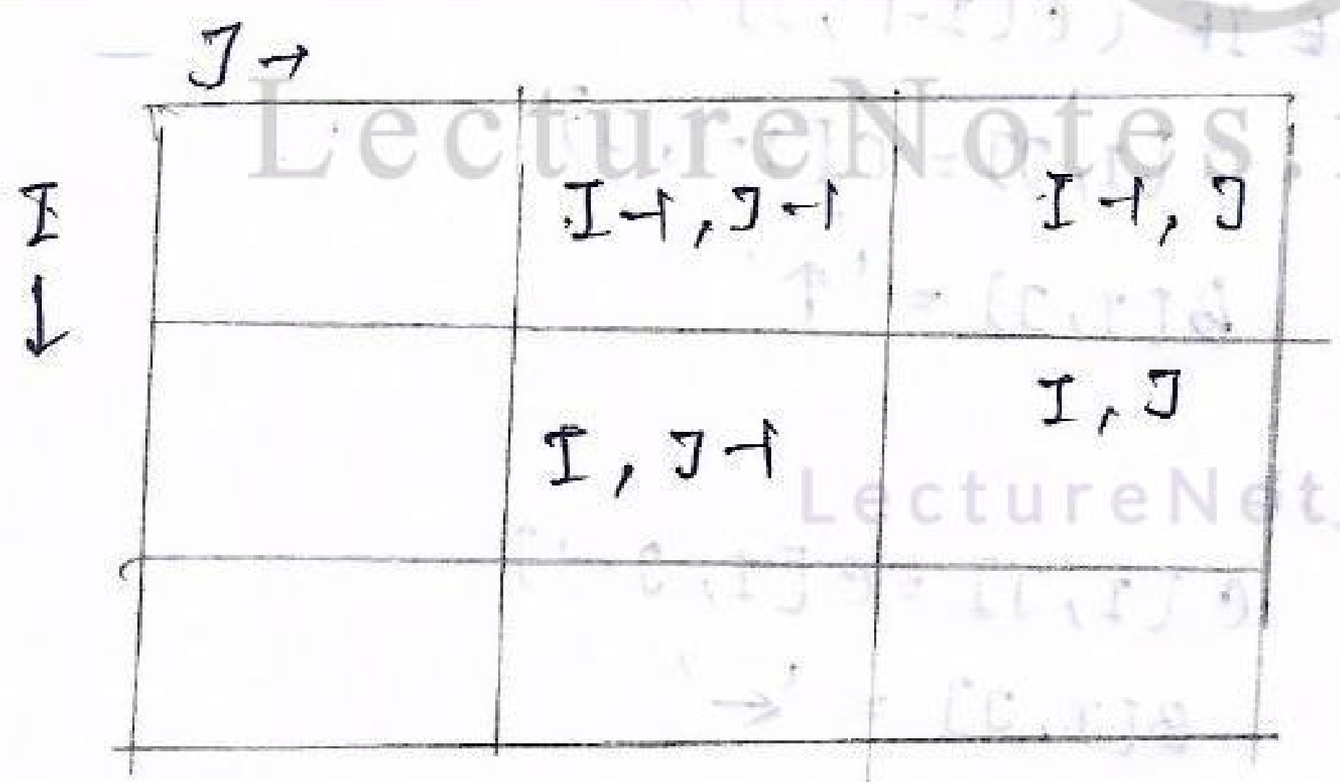
LectureNotes.in

let $X = \langle A, B, C, D, B, A, C, D, F \rangle$

$Y = \langle C, B, A, F \rangle$

			(C) 0	(B) 1	(A) 2	(F) 3	4
	0		0	0	0	0	0
(A)	1		0	0↑	0↑	1↖	1←
(B)	2		0	0↑	1↖	1↑	1↑
(C)	3		0	1↖	1↑	1↑	1↑
(D)	4		0	1↑	1↑	1↑	1↑
(B)	5		0	1↑	2↖	2←	2←
(A)	6		0	1↑	2↑	3↖	3←
(C)	7		0	1↖	2↑	3↑	3↑
(D)	8		0	1↑	2↑	3↑	3↑
(F)	9		0	1↑	2↑	3↑	4↖

The LCS is $\langle C, B, A, F \rangle$



Algorithm →

LCS_LENGTH (X, Y)

1. M ← LENGTH [X]
2. N ← LENGTH [Y]
3. FOR I ← 0 to M
4. C[I, 0] = 0

[END FOR 3]

5. FOR J ← 0 to N
6. RC[0, J] = 0

[END FOR 5]

7. FOR I ← 1 to M
8. FOR J ← 1 to N
9. IF (X[I] = Y[J])
10. C[I, J] = C[I-1, J-1]+1
11. B[I, J] = '↖'
12. ELSE IF (C[I-1, J] >= C[I, J-1])
13. C[I, J] = C[I-1, J]
14. B[I, J] = '↑'

ELSE

15. C[I, J] = C[I, J-1]
16. B[I, J] = '←'

[END IF 9]

[END FOR 8]

[END FOR 7]

17. RETURN C AND B

LectureNotes.in

Time Complexity, $T(n) = \Theta(n^2)$

```
PRINT_LCS (B, X, I, J)
1. IF I=0 OR J=0
2.        RETURN
 [END IF]
3. IF B[I, J] = '↖'
4.        PRINT_LCS (B, X, I-1, J-1);
5.        PRINT X[I]
6. ELSE IF (B[I, J] = '↑')
7.        PRINT_LCS (B, X, I-1, J).
  ELSE
8.        PRINT_LCS (B, X, I, J-1)
[END IF]
```

LectureNotes.in

Dt-17th Mar18
Saturday

Data Structure for Disjoint Sets

Disjoint Sets -

Disjoint set data structure contains set of disjoint sets where

$$S = \{ S1, S2, S3 \ldots Sn \}$$

such that

$$S_I \cap S_J = \phi$$

S1, S2, S3 ... Sn are disjoint sets.

Example -

S1 = {A, B, P} , S2 = { M, N, Q } , S3 = { 7, 8, 9 }

Here, A is the representative of set S1.

Here, M is the representative of set S2.

Here, 7 is the representative of set S3.

Some operations of Disjoint Sets →

1. MAKE_SET(X) :

- It creates a new set whose only member is X.
- X is the representative.

2. UNION (X, Y) :

- It unites the set that contains the element (x and y)

- The Union operation is possible if the representative of the member are not same.

Example -

UNION (B, N) = {A, B, P, M, N, Q}

3. FIND_SET(X):

- It will return the representative of the set in which X is the member.

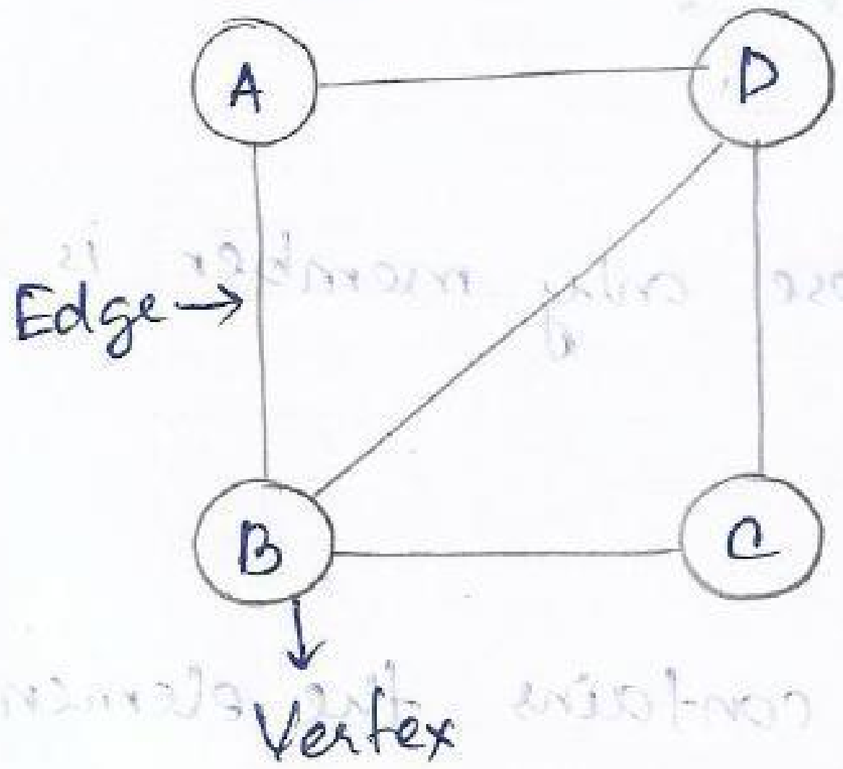
Example -

FIND_SET(N) : M

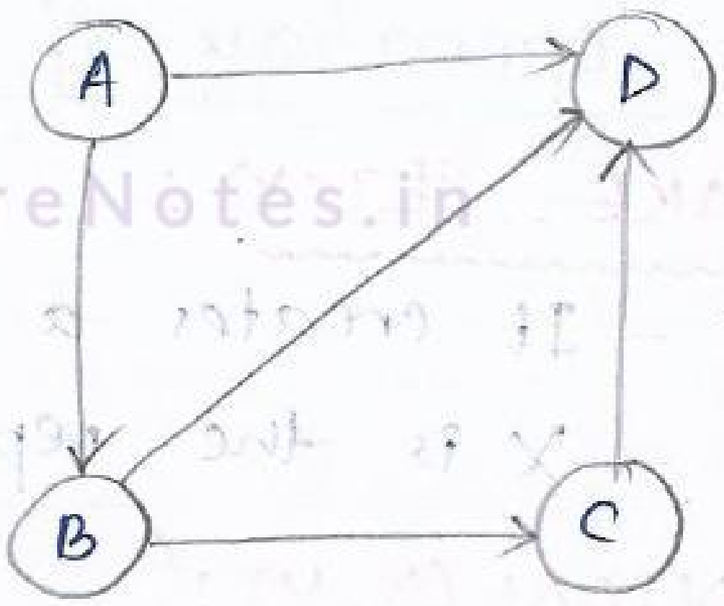
Applications of Disjoint Sets -

One of the application of Disjoint set data structure is Determine connected components of undirected graph.

Undirected Graph



Directed Graph



LectureNotes.in

Q. Find the connected components of undirected graph, G which has a set of vertices, V and edges, E.

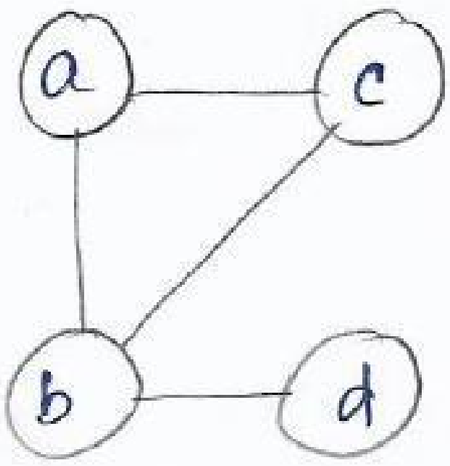
$G = (V, E)$ where $V = \{a, b, c, d, e, f, g, h, i, j\}$

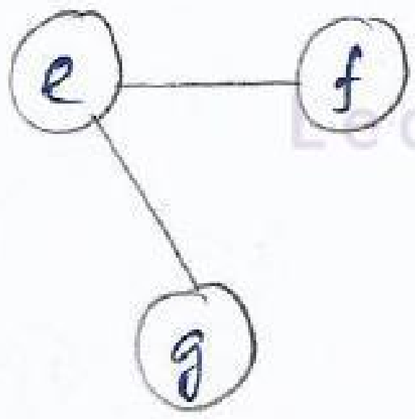
and $E = \{(b,d), (e,g), (a,c), (h,i), (a,b), (e,f), (b,c)\}$

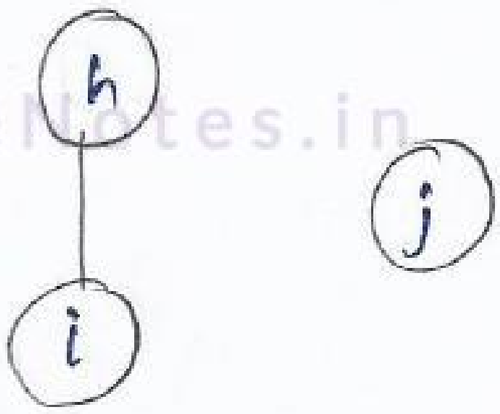
Procedure →

Sol.

Edges Processed	Collection of Disjoint Sets
Initial state	{a} {b} {c} {d} {e} {f} {g} {h} {i} {j}
(b,d)	{a} {b,d} {c} {e} {f} {g} {h} {i} {j}
(e,g)	{a} {b,d} {c} {e,g} {f} {h} {i} {j}
(a,c)	{a,c} {b,d} {e,g} {f} {h} {i} {j}
(h,i)	{a,c} {b,d} {e,g} {f} {h,i} {j}
(a,b)	{a,c,b,d} {e,g} {f} {h,i} {j}
(e,f)	{a,c,b,d} {e,g,f} {h,i} {j}
(b,c)	{a,c,b,d} {e,g,f} {h,i} {j}







Graph-1
Tree-1

Graph-2
Tree-2

Graph-3
Tree-3

Graph-4
Tree-4

This is called Disjoint Set of Forest.

This is called Disjoint Set of Forest.

Algorithm →

CONNECTED_COMPONENT(G)

1. FOR EACH VERTEX V ∈ V[G]
2. MAKE-SET(V)

[END FOR]

3. FOR EACH EDGE (U,V) ∈ E[G]
4. IF FIND_SET(U) ≠ FIND_SET(V)
5. UNION(U,V)

[END IF]

[END FOR]

SAME_COMPONENT(U, V)

1. IF FIND_SET(U) = FIND_SET(V)

RETURN TRUE

ELSE

RETURN FALSE

LectureNotes.in

Q. Find the Hoffmann code for each character of the following text -

WORK IS WORKSHIP OF GOD

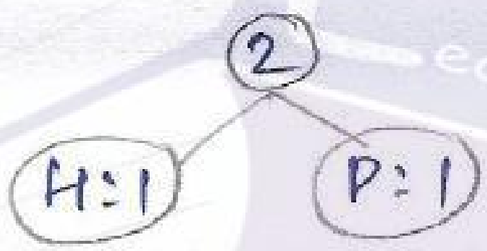
Sol. The characters and their frequencies are

W:2, O:4, R:2, K:2, Space:4, I:2, S:2, H:1, P:1, F:1, G:1, D:1

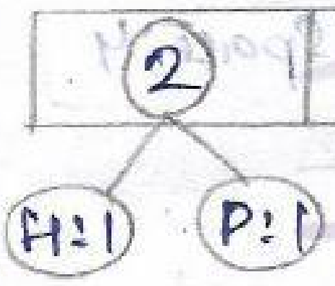
H:1	P:1	F:1	G:1	D:1	W:2	R:2	K:2	I:2	S:2	O:4	Space:4

Step 1 -

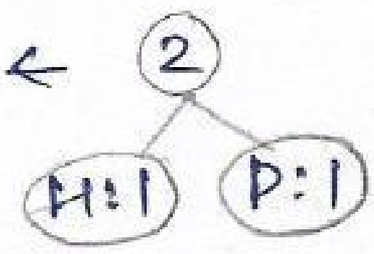
$X \leftarrow H:1$, $Y \leftarrow P:1$, $f[z] = 2$



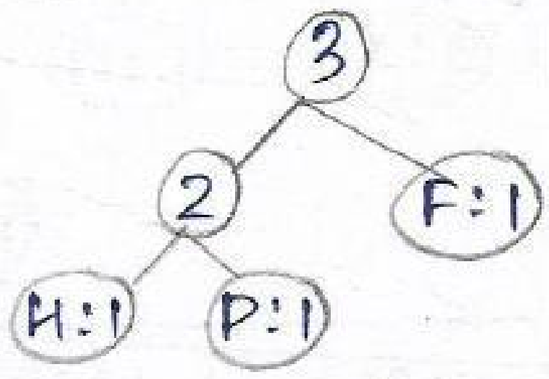
2	F:1	G:1	D:1	W:2	R:2	K:2	I:2	S:2	O:4	Space:4



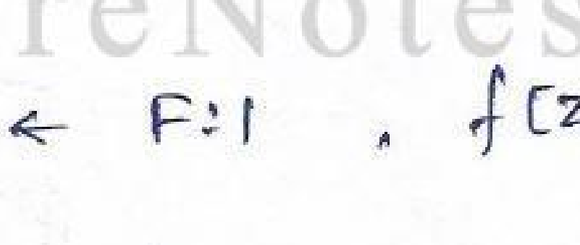
Step 2 -

$X \leftarrow$ 

, $Y \leftarrow F:1$, $f[z] = 3$

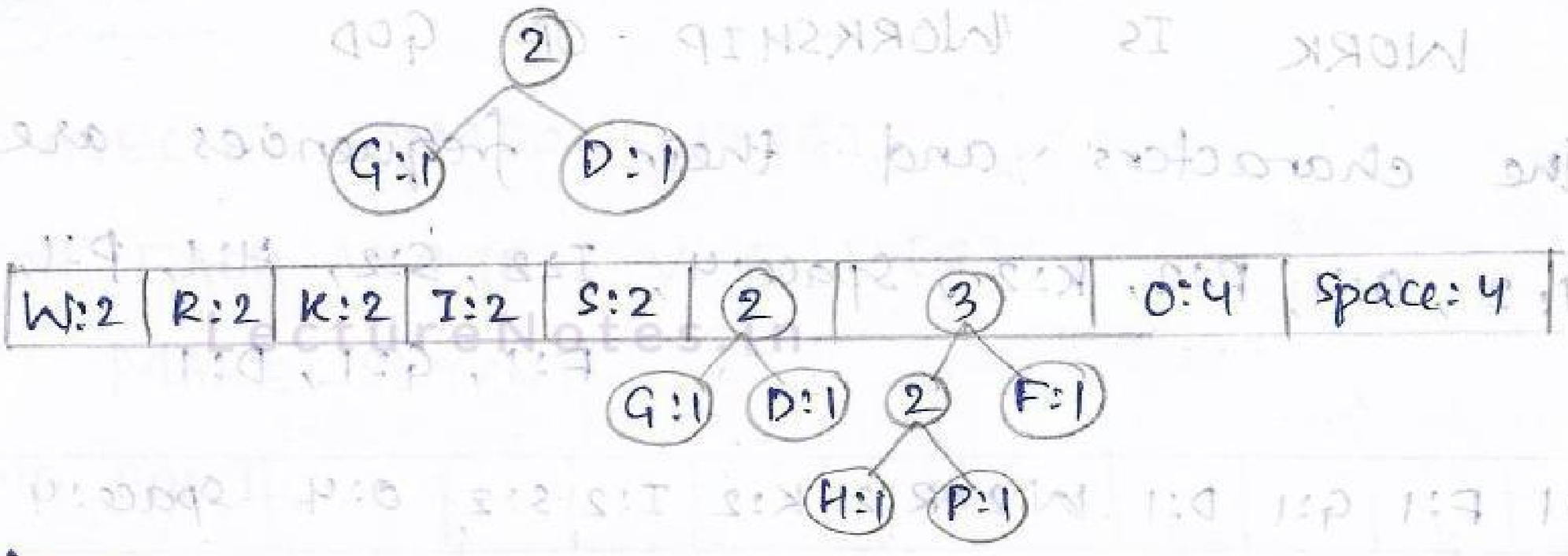


G:1	D:1	W:2	R:2	K:2	I:2	S:2	3	O:4	Space:4



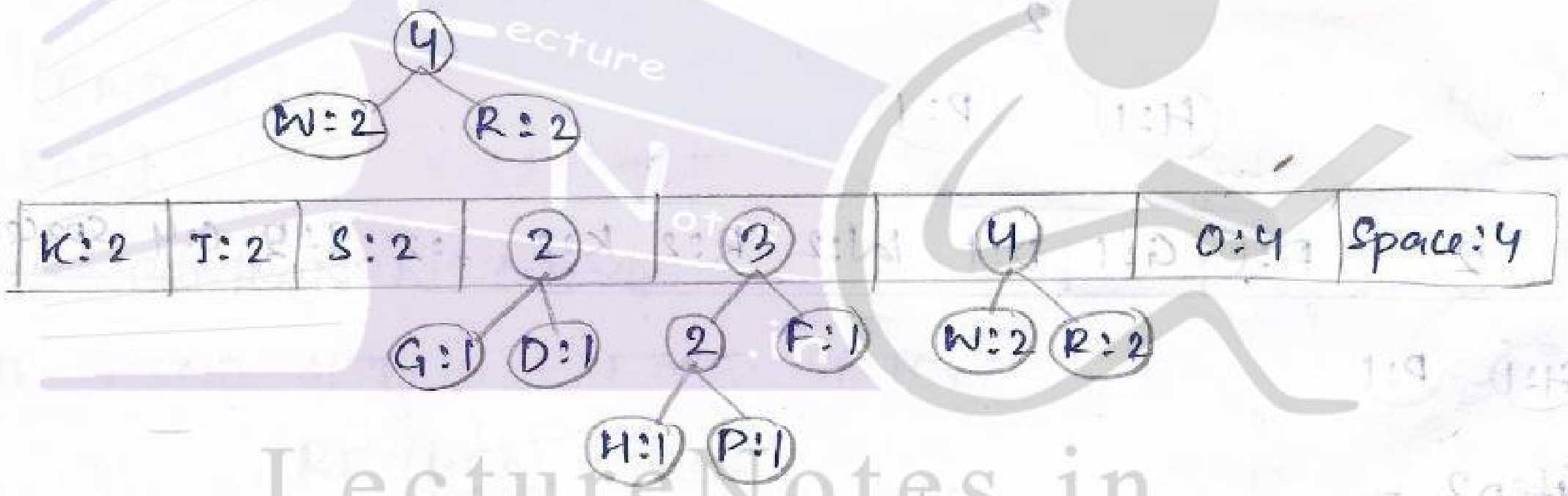
Step 3 →

$x \leftarrow G:1$, $y \leftarrow D:1$, $f[z] = 2$



Step 4 →

$x \leftarrow W:2$, $y \leftarrow R:2$, $f[z] = 4$

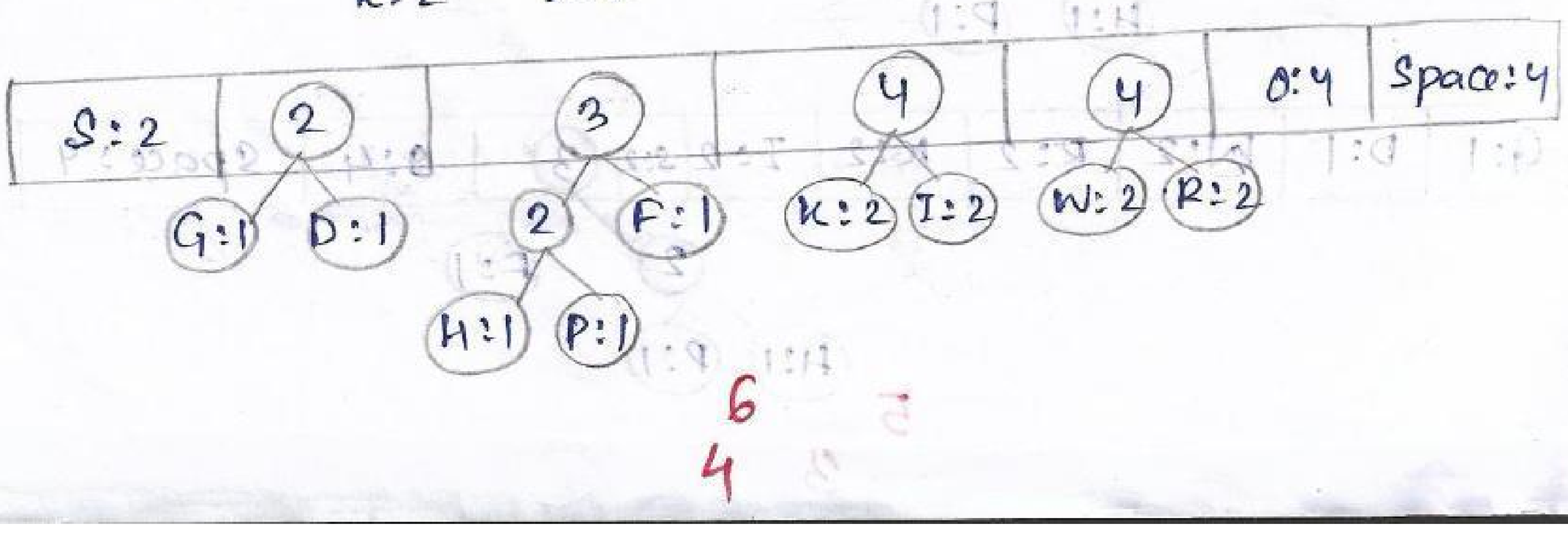


Step 5 →

$x \leftarrow K:2$, $y \leftarrow I:2$, $f[z] = 4$

4

K:2 I:2

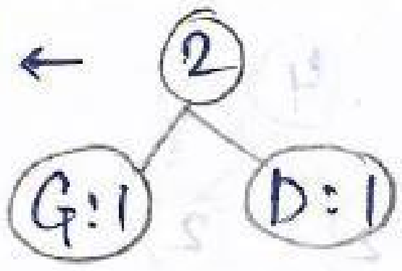


6

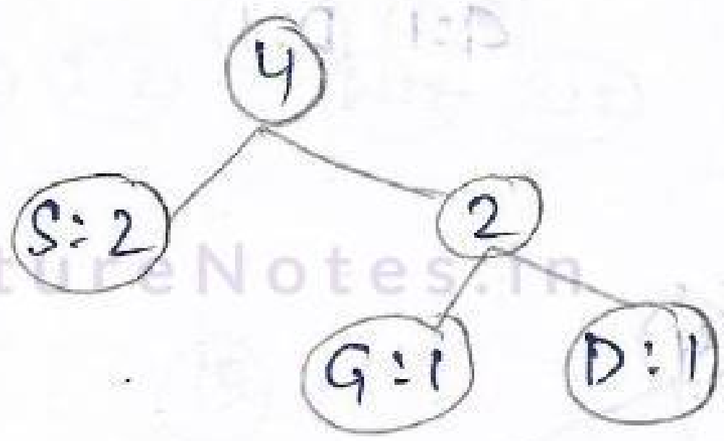
4

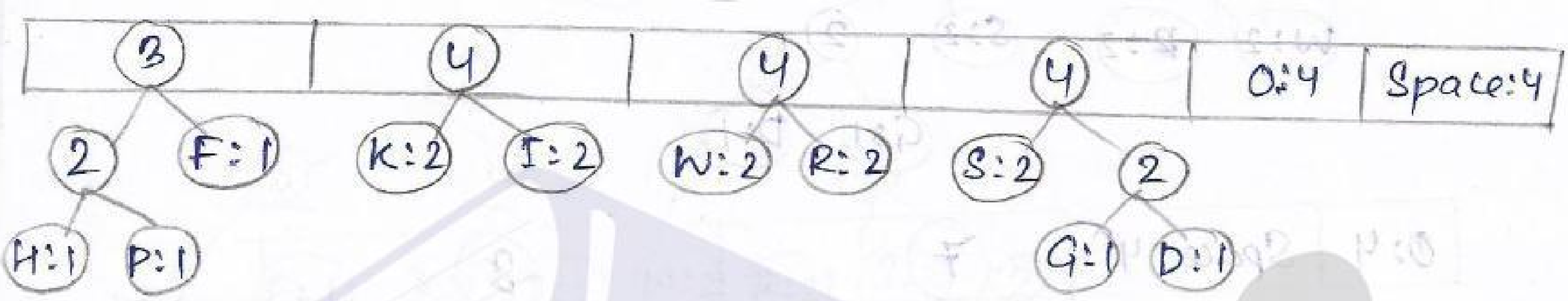
LectureNotes.in

Step 6 →

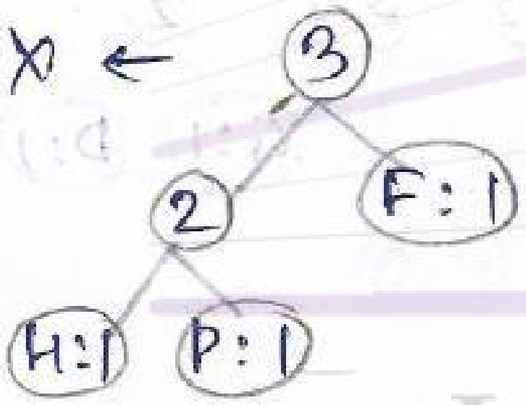
X ← S:2 , Y ← 

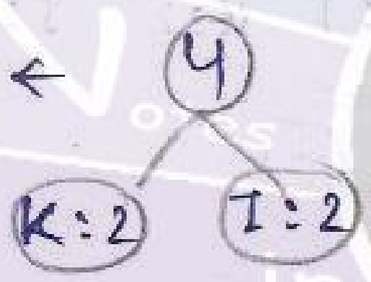
, f[z] = 4



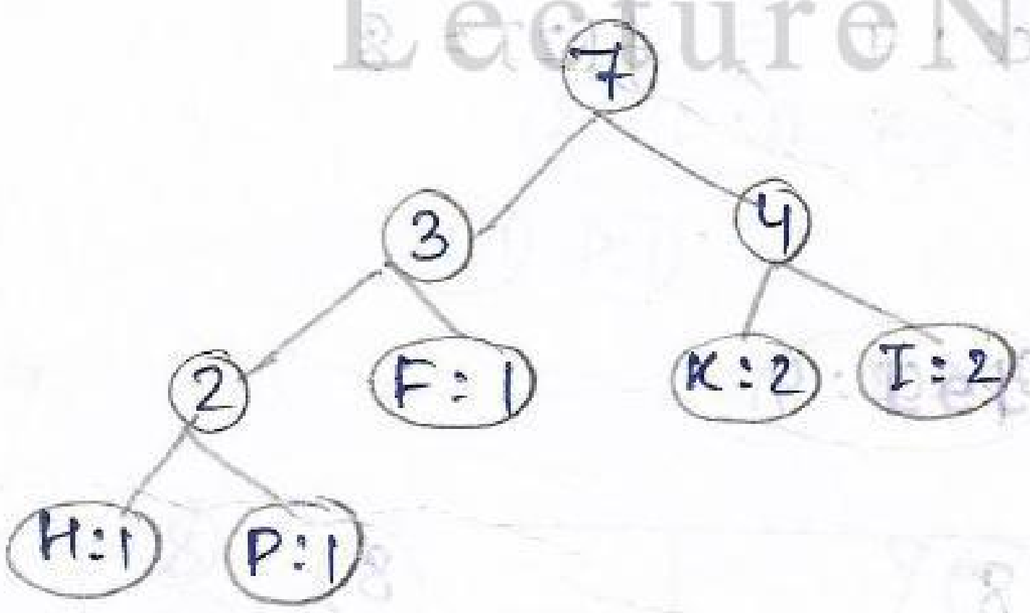


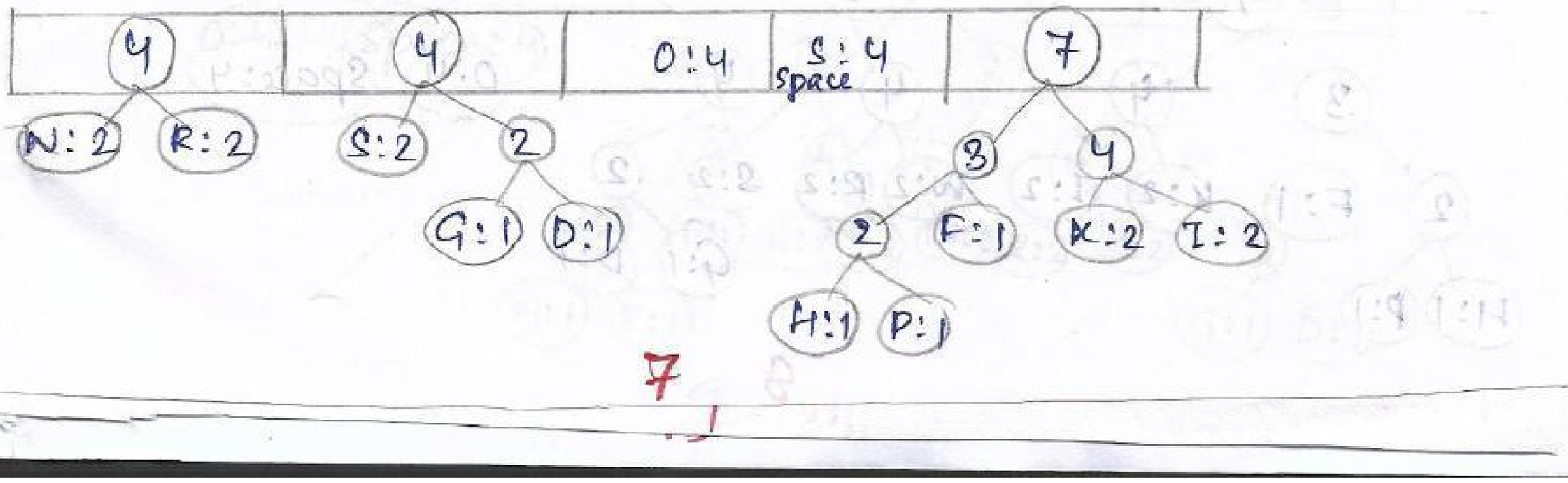
Step 7 →

X ← 

, Y ← 

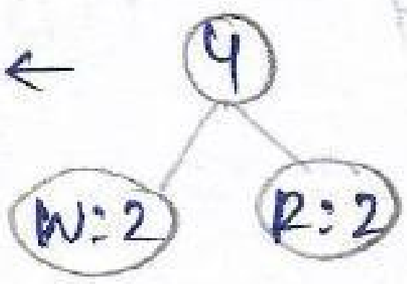
, f[z] = 7

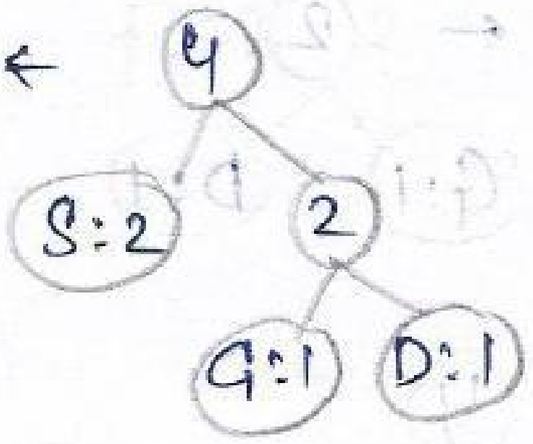




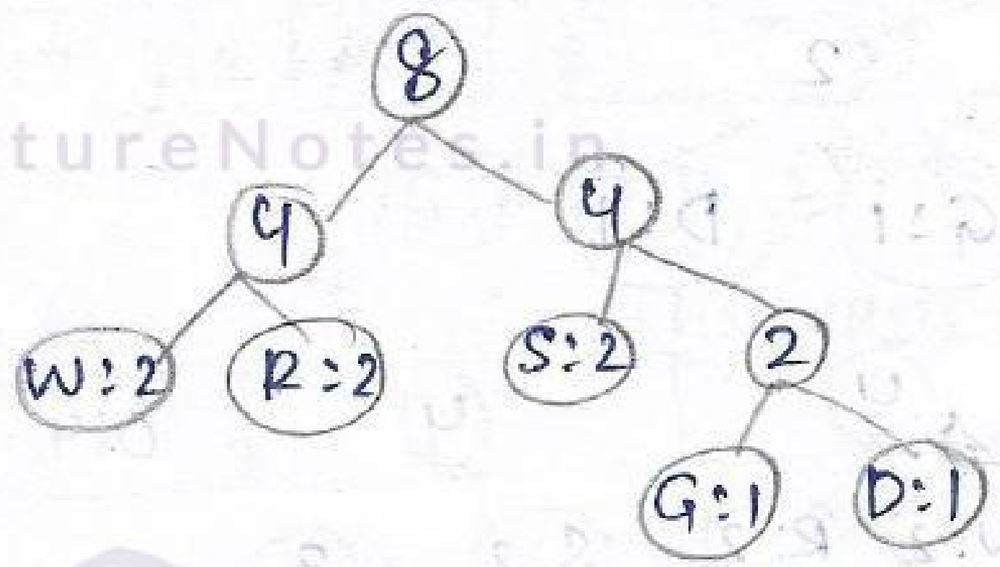
7

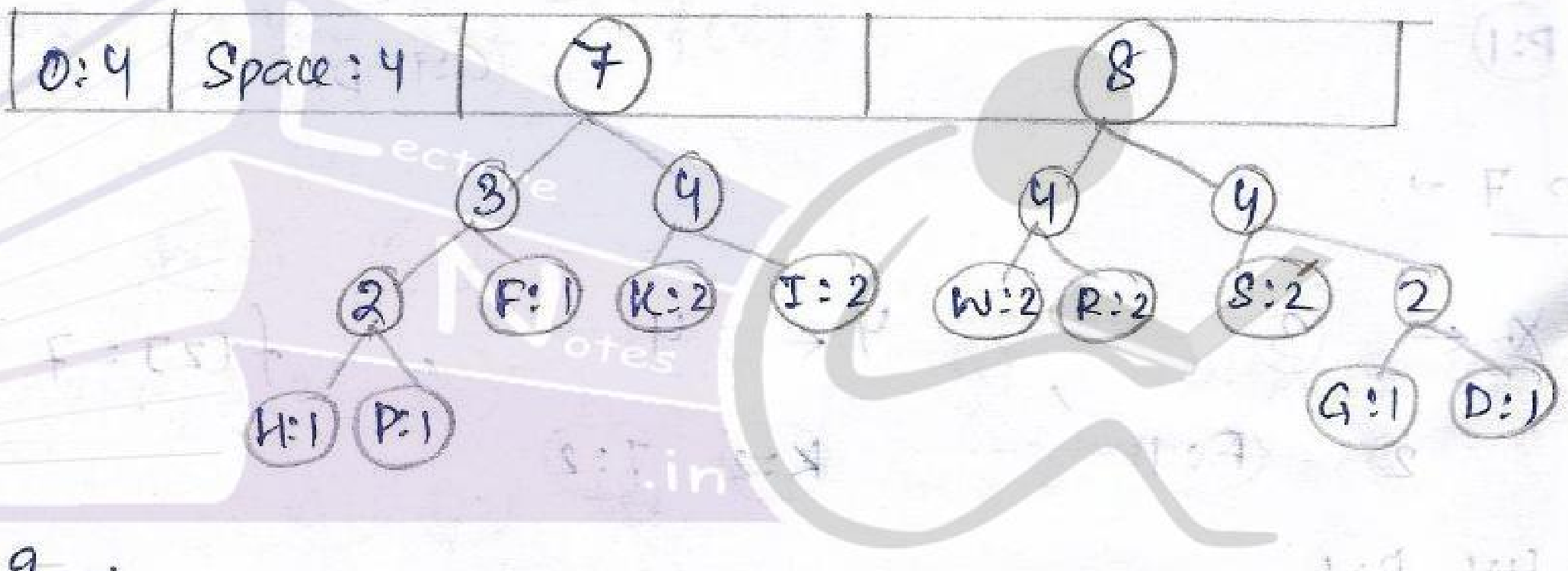
Step 8 →

x ← 

, y ← 

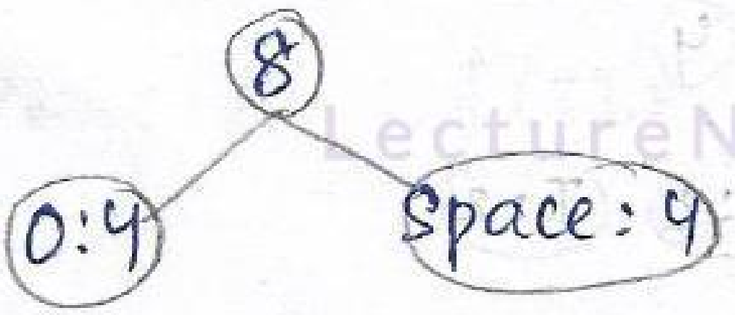
, f[z] = 8

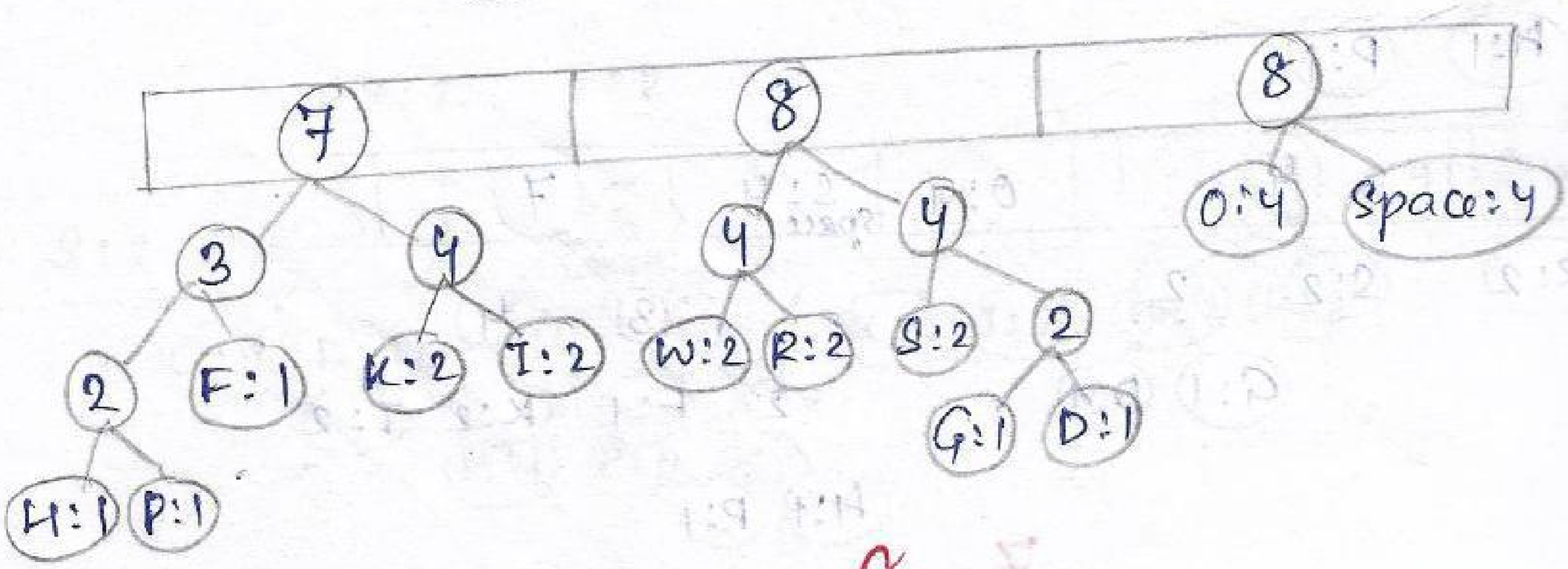




Step 9 →

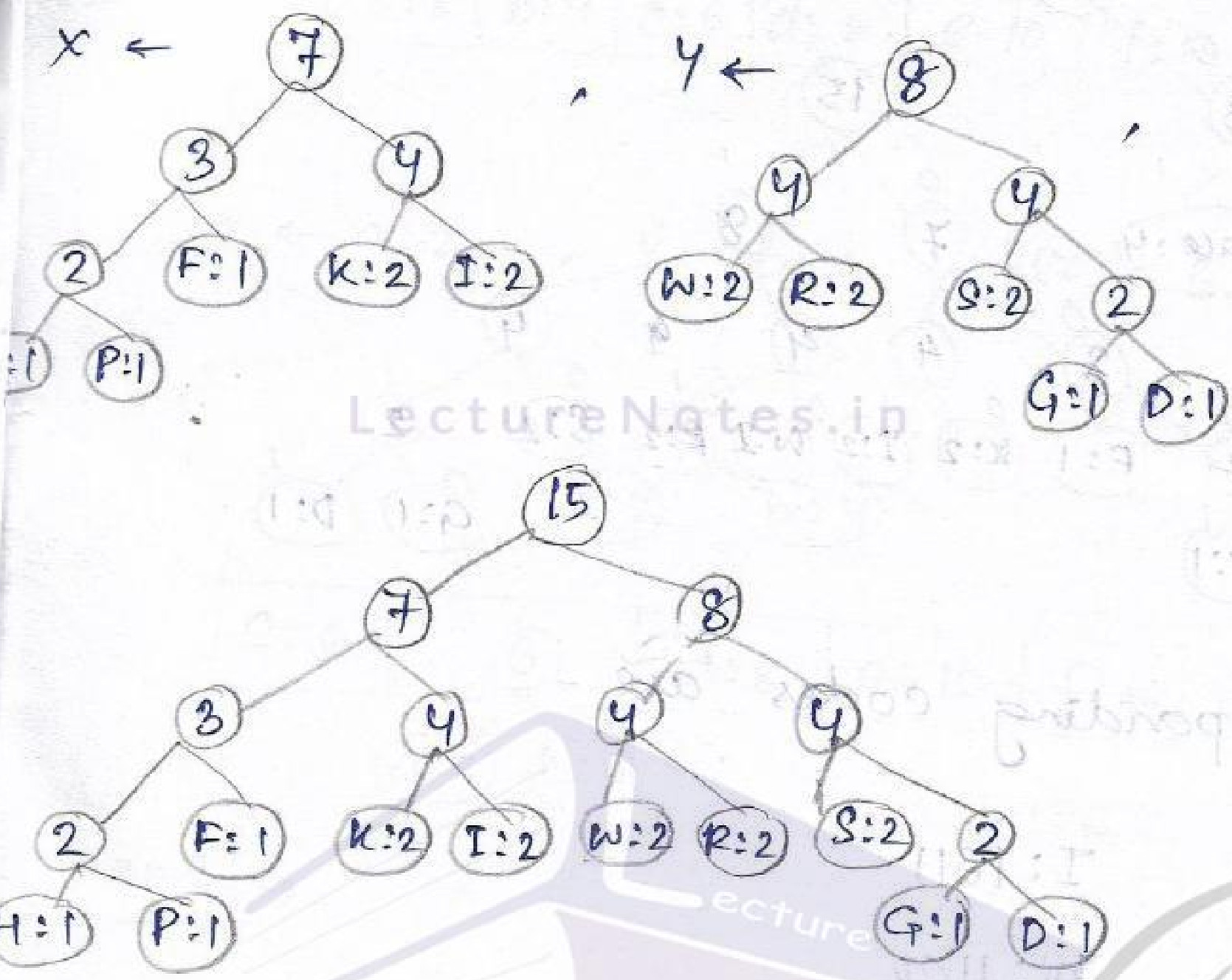
$x \leftarrow 0:4$, $y \leftarrow$ Space : 4 , $f[z] = 8$

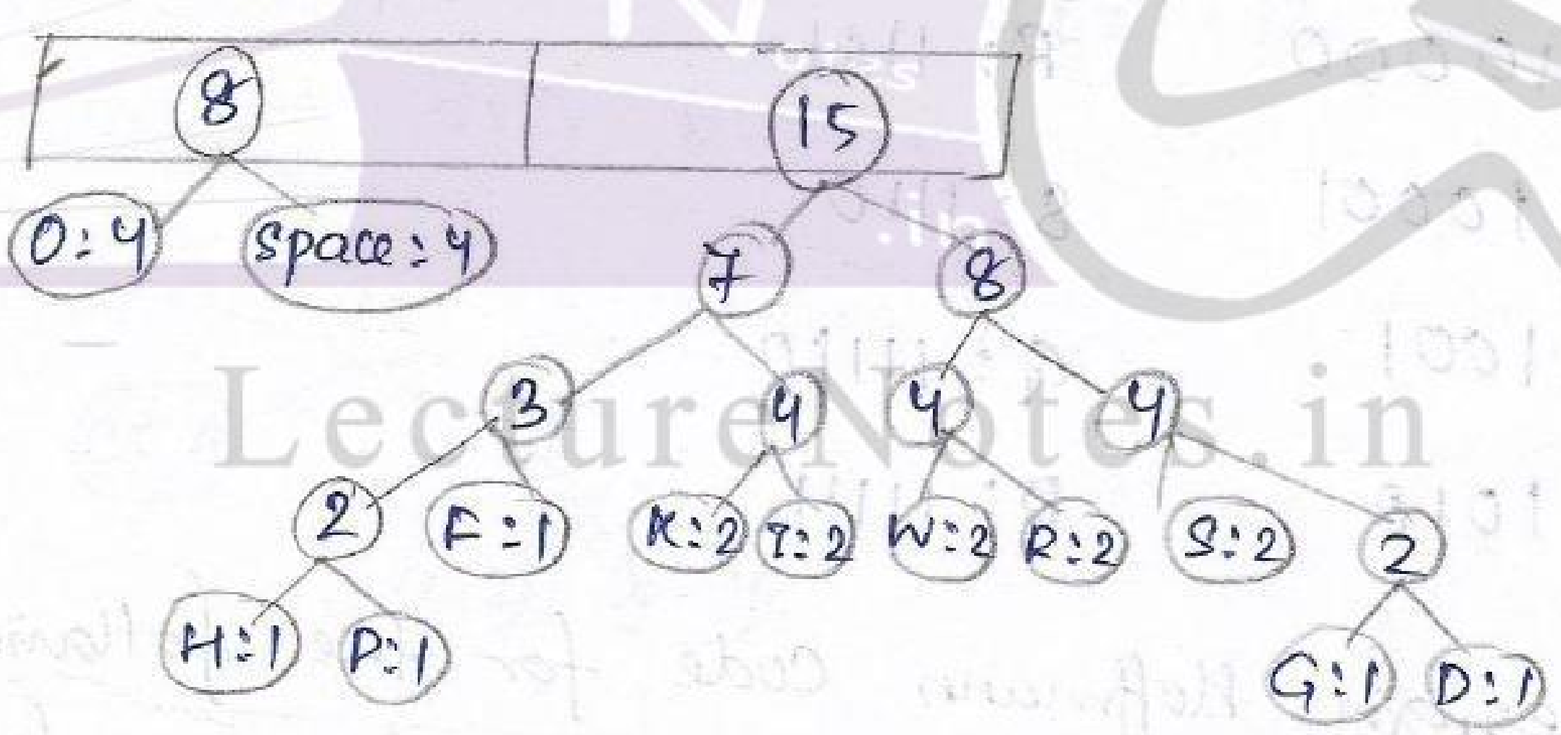




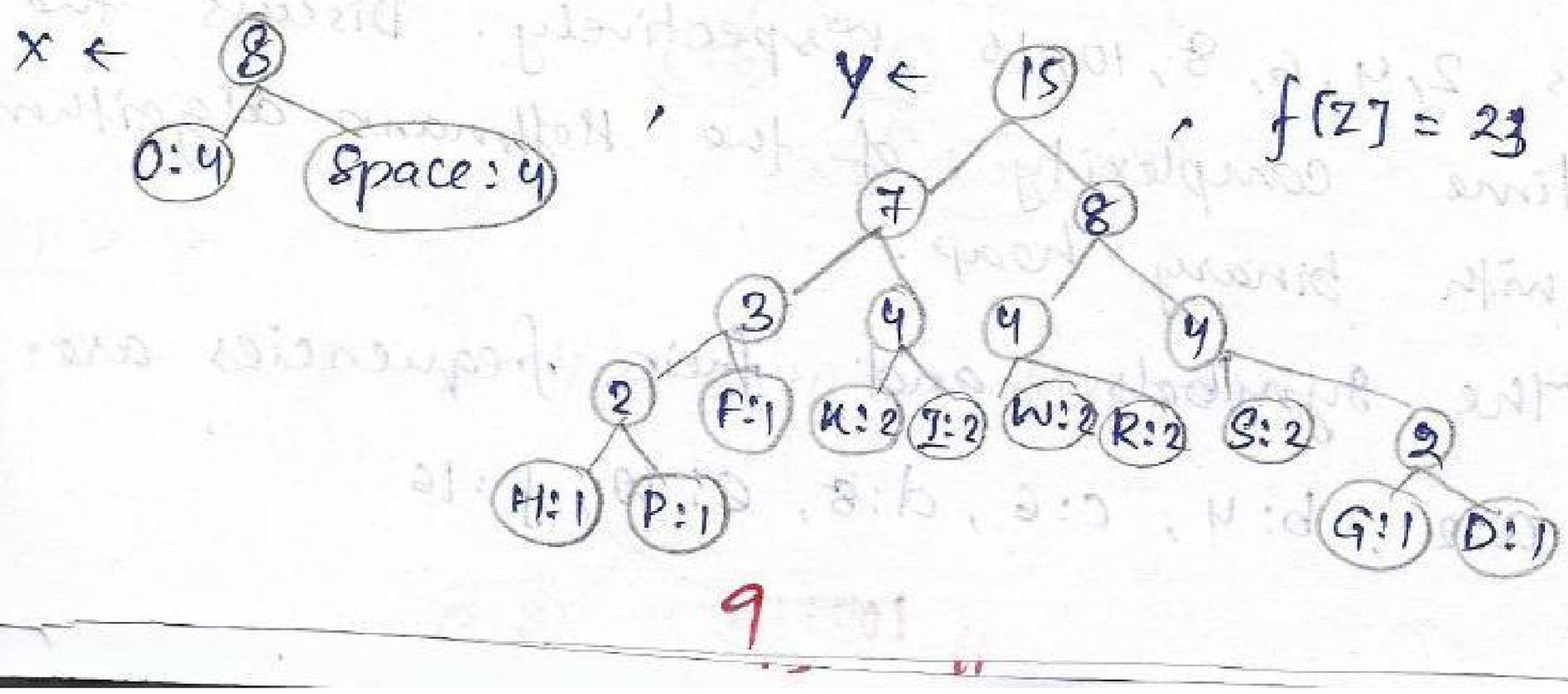
7 8

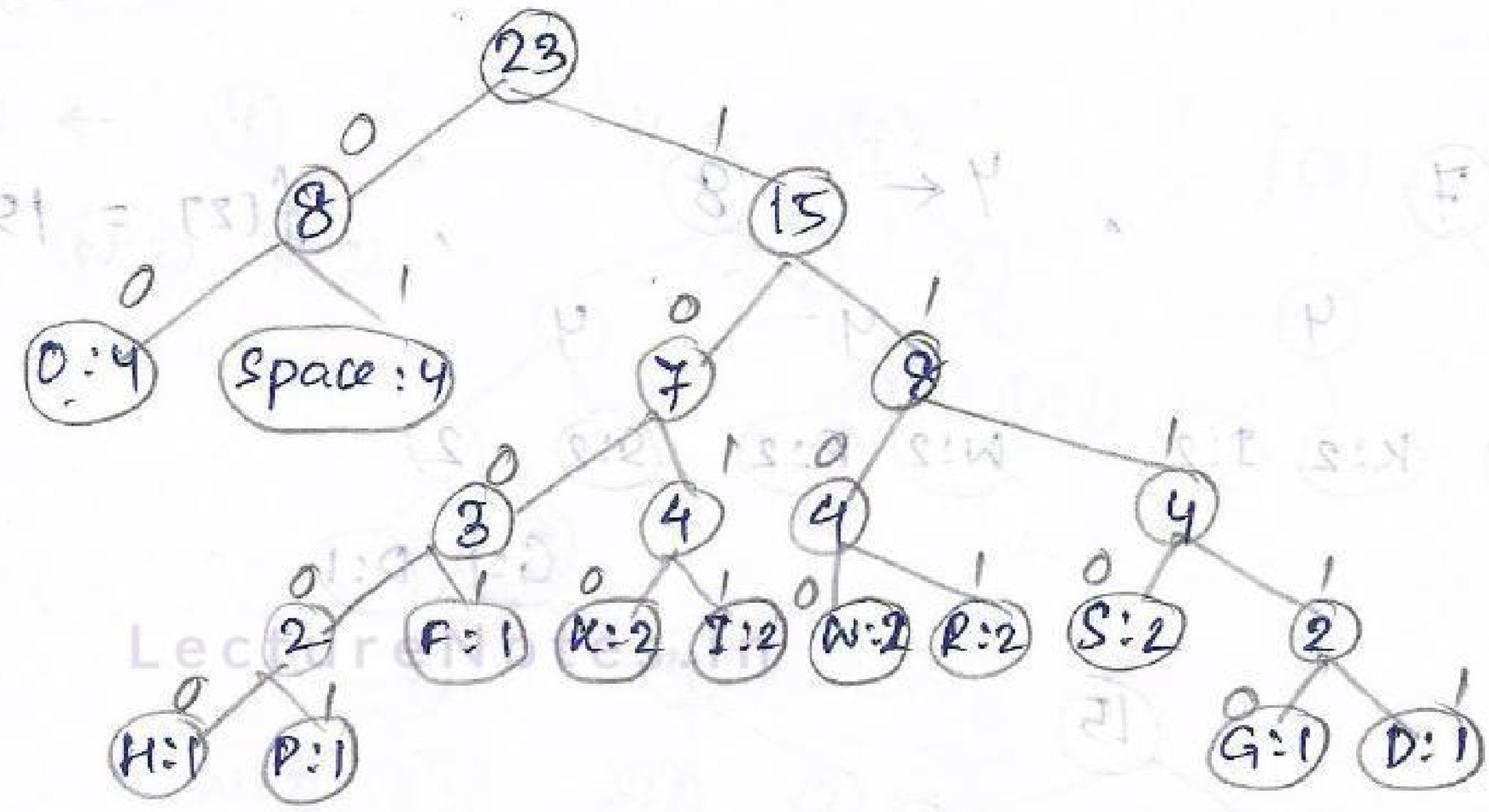
Step 10 →





Step 11 →





The corresponding codes are -

0 : 00	I : 1011
Space : 01	W : 1100
H : 10000	R : 1101
P : 10001	S : 1110
F : 1001	G : 11110
K : 1010	D : 11111

Q. Design Hoffmann code for the following symbol. a, b, c, d, e, f. The corresponding frequency is 2, 4, 6, 8, 10, 16 respectively. Discuss the time complexity of the Hoffmann algorithm with binary heap.

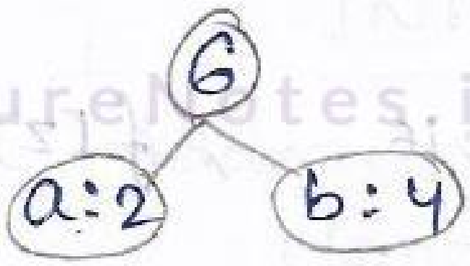
Sol. The symbols and their frequencies are:

a : 2 , b : 4 , c : 6 , d : 8 , e : 10 , f : 16

a:2	b:4	c:6	d:8	e:10	f:16

Step 1 →

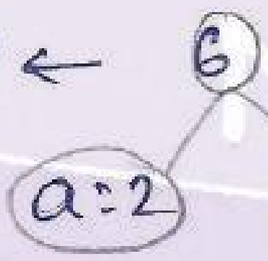
$x \leftarrow a:2$, $y \leftarrow b:4$, $f[z] = 6$



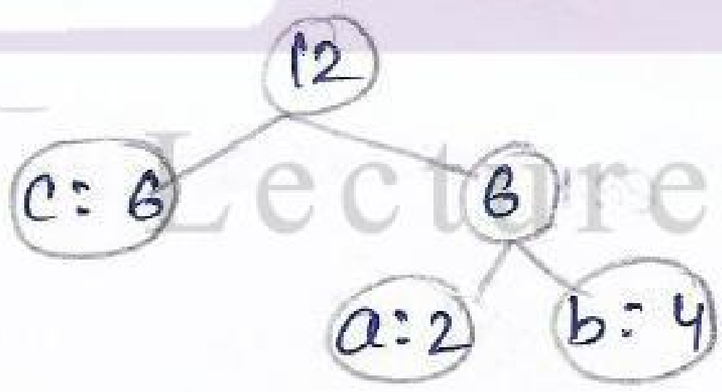
c:6	6	d:8	e:10	f:16



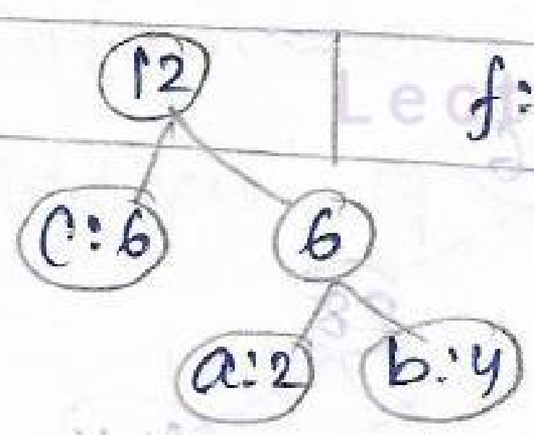
Step 2 →

$x \leftarrow c:6$, $y \leftarrow$ 

, $f[z] = 12$

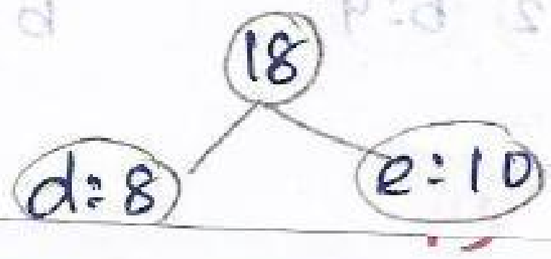


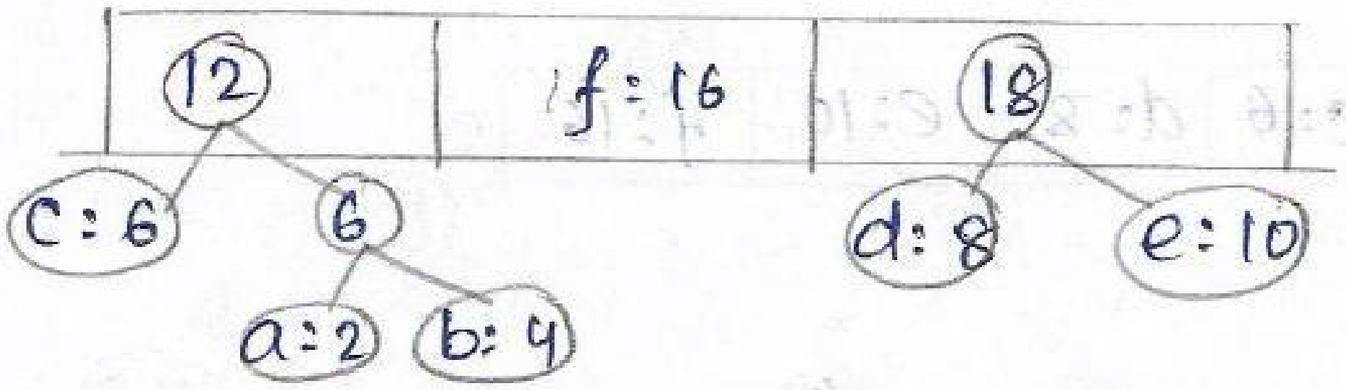
d:8	e:10	12	f:16



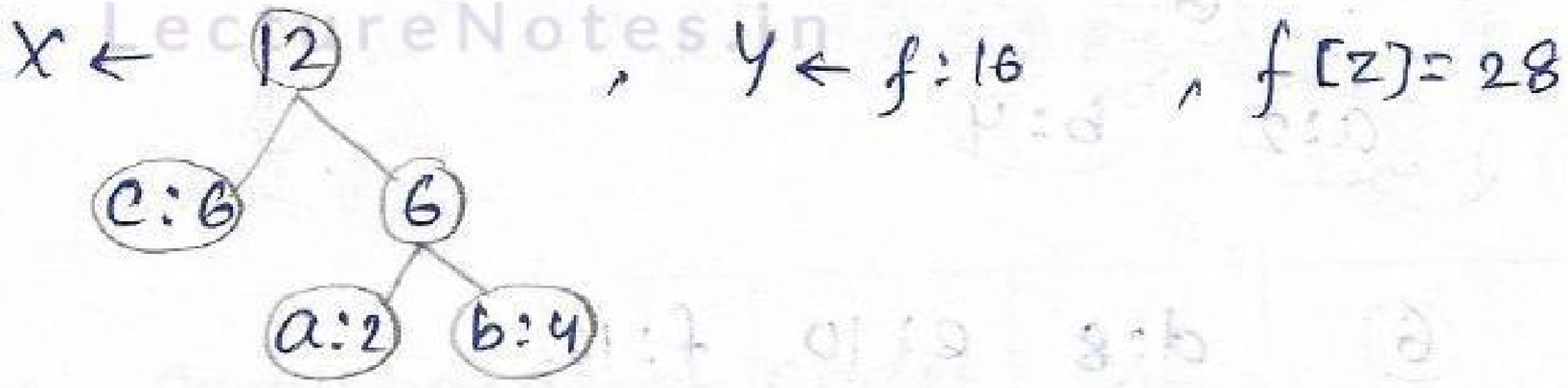
Step 3 →

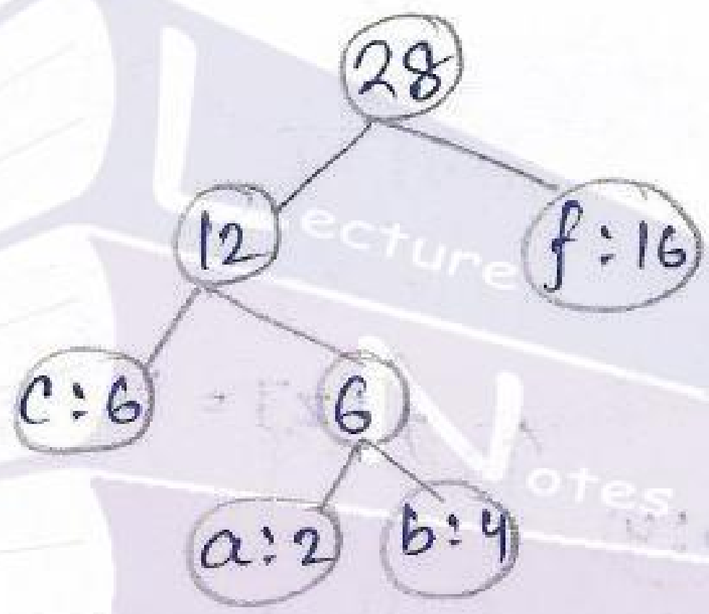
$x \leftarrow d:8$, $y \leftarrow e:10$, $f(z) = 18$

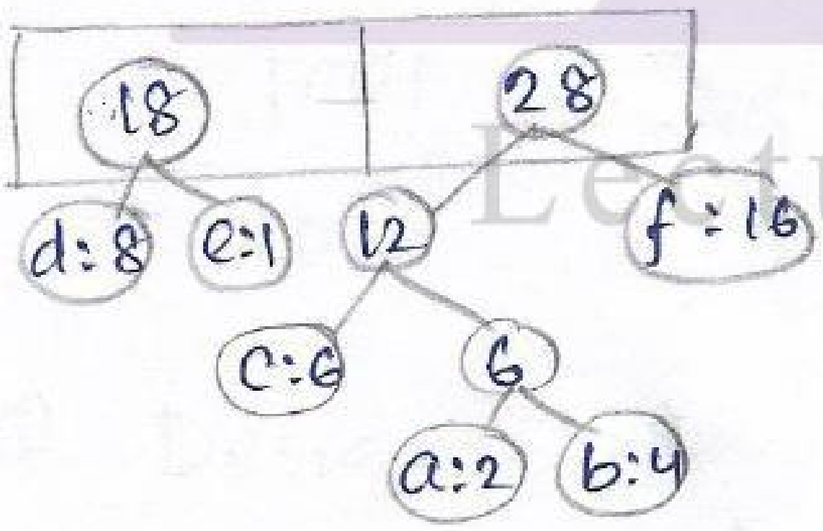




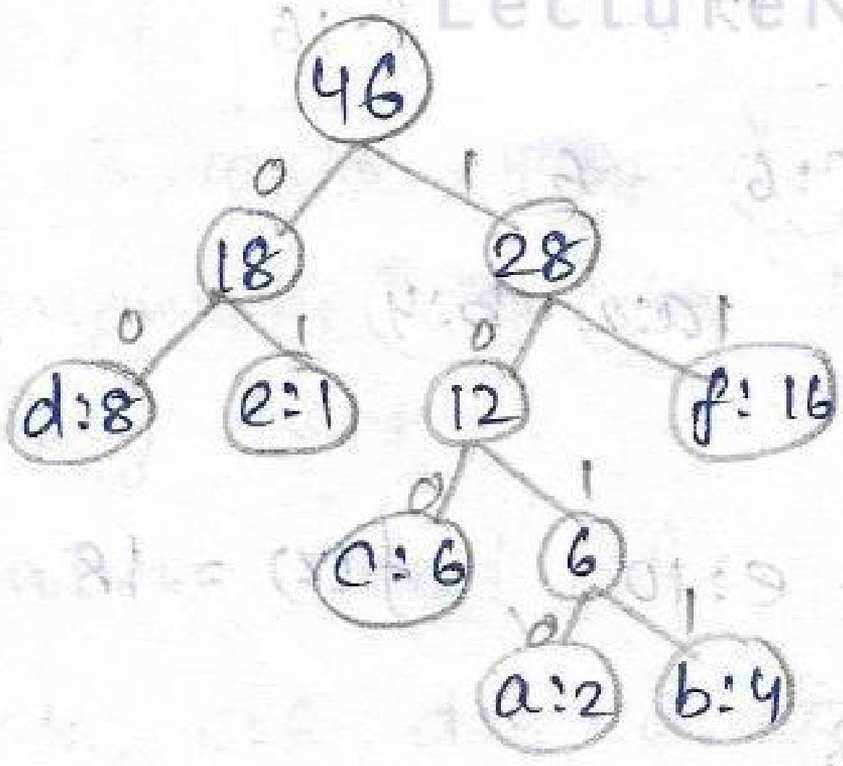
Step 4 →







Step 5 →

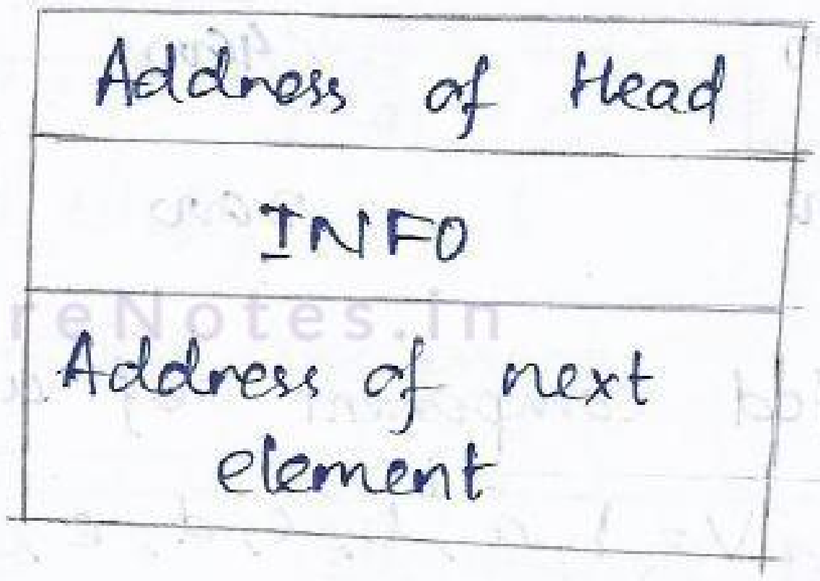


Corresponding Codes -

d : 00
e : 01
c : 100
f : 11
a : 1010
b : 1011

Dt→. 19th Mar 18
Monday

Linked List Representation of Disjoint Set



Head node → (HEAD)

Head holds address of the representative of the set.

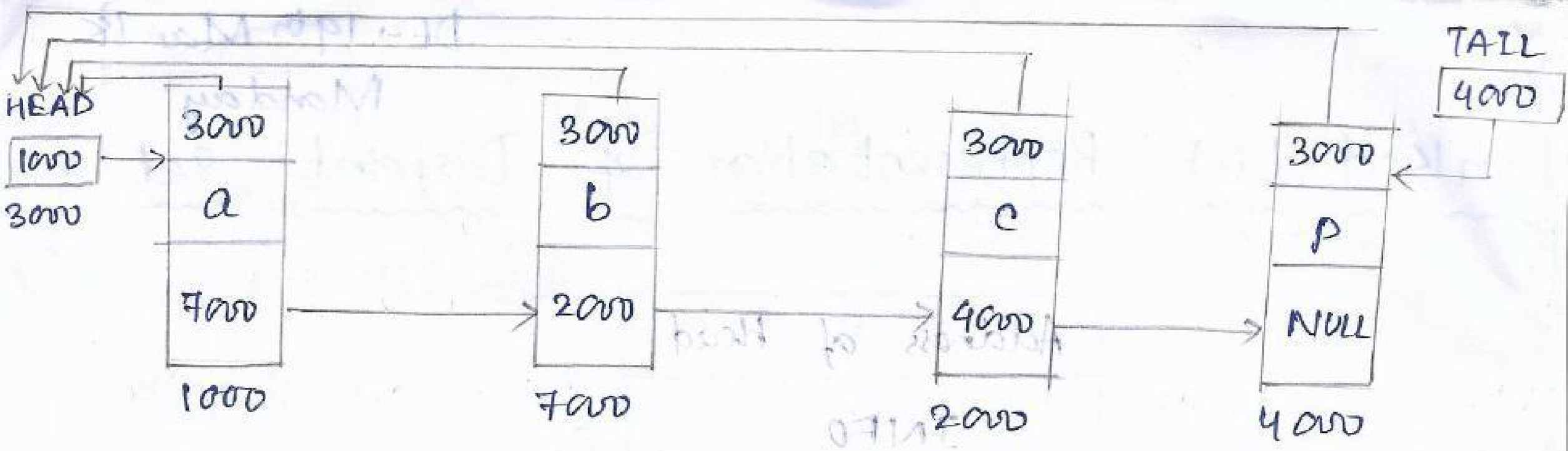
TAIL →

Tail holds the adress of last element of disjoint set.

* Here, Each element consists of 3 parts -
  - → Address of Head
  - → Information of Head
  - → Address of next element.

Example →

$S_1 = \{ a, b, c, p \}$



Q. Suppose the connected component of undirected graph, G = (V, E) where V = { a, b, c, d, e, f, g, h, i, j, k } and E = { (d, i), (f, k), (g, i), (b, g), (a, b), (i, j), (d, k), (b, j), (d, f), (a, e) }

Find the connected components.

Elements of Greedy Strategy

* Determine the optimal structure of the problem.
* Prove that at any stage, the optimal choice is the Greedy choice.
* Show that one of the sub-problem is empty.
* Develope a recursive algorithm that implements the Greedy strategy.
* Convert the recursive algorithm into iterative algorithm.

LectureNotes.in

Q. Suppose the connected component of undirected graph, G

$G = (V, E)$ where

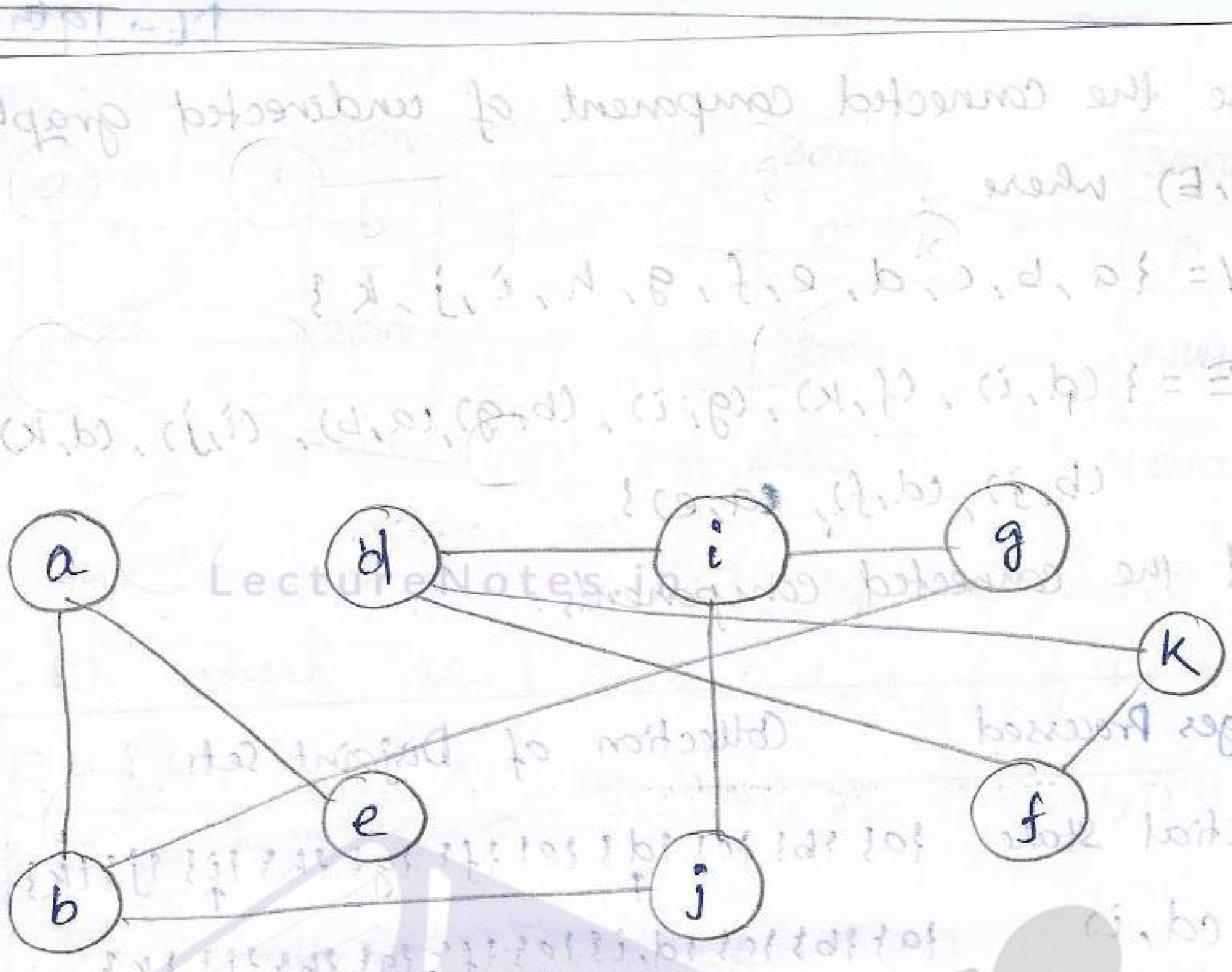
$V = \{a, b, c, d, e, f, g, h, i, j, k\}$

and $E = \{(d,i), (f,k), (g,i), (b,g), (a,b), (i,j), (d,k), (b,j), (d,f), (a,e)\}$

Find the connected components.

Sol.

Edges Processed	Collection of Disjoint Sets
Initial State	{a} {b} {c} {d} {e} {f} {g} {h} {i} {j} {k}
(d, i)	{a} {b} {c} {d, i} {e} {f} {g} {h} {j} {k}
(f, k)	{a} {b} {c} {d, i} {e} {f, k} {g} {h} {j}
(g, i)	{a} {b} {c} {d, i, g} {e} {f, k} {h} {j}
(b, g)	{a} {c} {b, d, i, g} {e} {f, k} {h} {j}
(a, b)	{a, b, d, i, g} {c} {e} {f, k} {h} {j}
(i, j)	{a, b, d, i, j, g} {c} {e} {f, k} {h}
(d, k)	{a, b, d, i, j, g, f, k} {c} {e} {h}
(b, j)	{a, b, d, i, j, g, f, k} {c} {e} {h}
(d, f)	{a, b, d, i, j, g, f, k} {c} {e} {h}
(a, e)	{a, b, d, e, i, j, g, f, k} {c} {h}



Graph - 1

Tree - 1

(c)

Graph - 2

Tree - 2

(b)

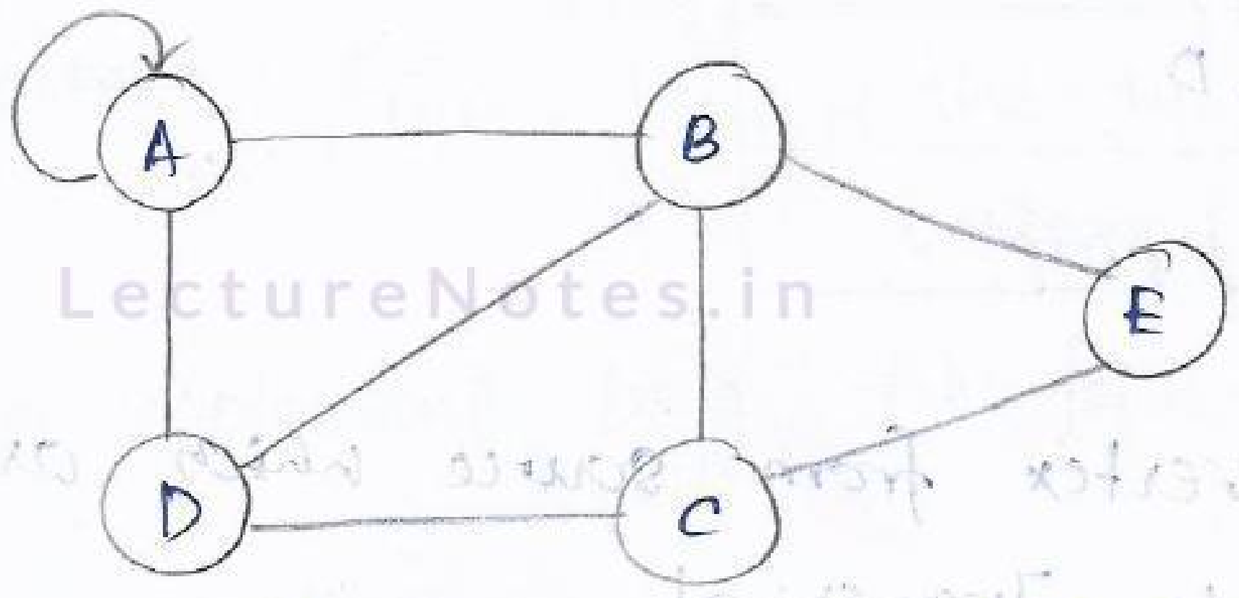
Graph - 3

Tree - 3

LectureNotes.in

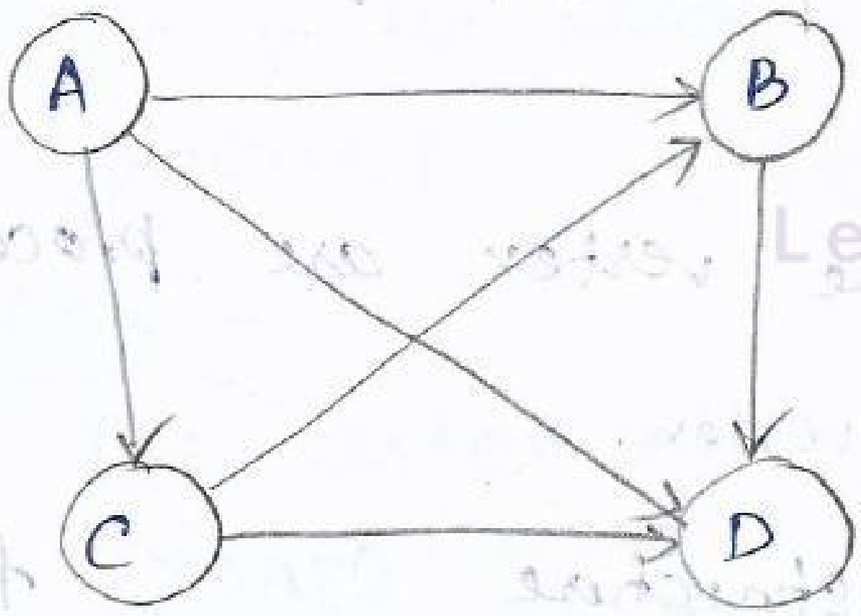
Graph Traversal →

Adjacent of Vertex -



Graph 1 (Undirected Graph)

Vertex	
A	A, B, D
B	A, C, D, E
C	B, D, E
D	A, B, C
E	B, C



Graph 2 (Directed Graph)

LectureNotes.in

Vertex	Adjacent Vertex
A	B, C, D
B	D
C	B, D
D	—

LectureNotes.in

* Visit all the vertex from source which are reachab is called Graph Traversal.

Graph Traversal

- DFS (Depth First Search)
- BFS (Best First Search)

1. DFS (Depth First Search)

* In this technique, most recently vertex v the highest level nodes are processed first before the lowest level nodes.

* the adjacent of the vertex are processed first before the parent vertex.

* It uses the data structure STACK for implementation.

* DFS can be used in both directed and undirected graph.

* In DFS, each vertex V has 2 timestamps.

1. d(vertex) : Start time - The time when the vertex is discovered

2. f(vertex) : Finish time - the time at which the vertex is processed.

LectureNotes.in

* To implement DFA the following 3 colors are used -

1. WHITE
2. GRAY
3. BLACK

1. WHITE →
   - the vertex which are not processed, not discovered.
   - Initially, all vertex are in White color.

2. GRAY →
   - The vertex which are discovered but not processed.

3. BLACK →
   - The vertex which is processed

(Insert from Right side, Delete from Right side)

Algorithm -

DFS (G)

1. FOR EACH VERTEX $u \in V[G]$
2. COLOR [u] ← WHITE
3. $\pi[u]$ ← NULL

[END FOR]

4. TIME ← 0
5. FOR EACH VERTEX $u \in V[G]$
6. IF COLOR[u] ← WHITE
7. DFS_VISIT (u)

[END IF]

[END FOR]

DFS_VISIT (u)

1. TIME ← TIME + 1
2. COLOR [u] ← GRAY
3. d[u] ← TIME
4. FOR EACH VERTEX $v \in adj[u]$
5. IF COLOR [v] ← WHITE
6. DFS_VISIT (v)

[END IF]

[END FOR]

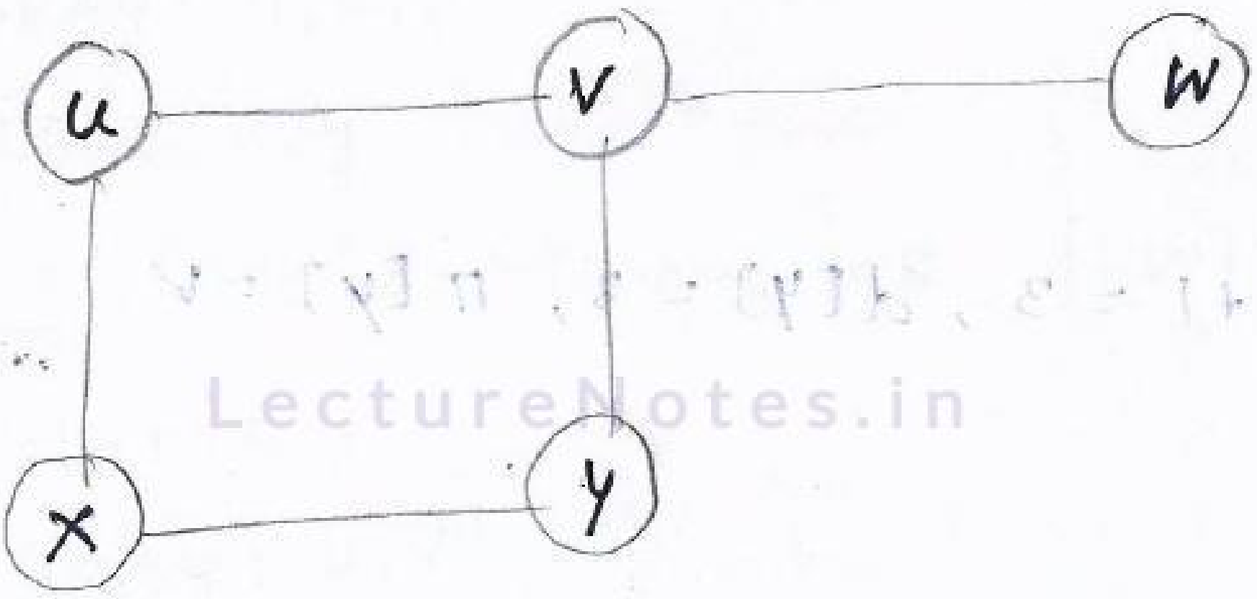
7. COLOR[u] ← BLACK

f[u] ← TIME

LectureNotes.in

Procedure →

Q. Traverse the following graph using DFA.



LectureNotes.in

Parent of Vertex ↑

Vertex	Ad-jacent of Vertex	d(vertex)	f(vertex)	π(vertex)
u	v, x	1	10	NULL
v	u, y, w	2	9	u
w	v	7	8	v
x	u, y	4	5	y
y	x, v	3	6	v

Initially,

Time: 0

WHITE: u, v, w, x, y

(STACK) GRAY:

BLACK:

Step 1 - Time = Time + 1 = 1, d[u] = 1, π[u] = NULL.

WHITE: v, w, x, y

GRAY: u

BLACK:

Step 2 - Time = Time + 1 = 2, d[v] = 2, π[v] = u

WHITE : w, x, y

GRAY : u, v

BLACK :

Step 3 - Time = Time + 1 = 3, d[y] = 3, π[y] = v

WHITE : w, x

GRAY : u, v, y

BLACK :

Step 4 - Time = Time + 1 = 4, d[x] = 4, π[x] = y

WHITE : w

GRAY : u, v, y, x

BLACK :

Step 5 - Time = Time + 1 = 5, f[x] = 5

WHITE : w

GRAY : u, v, y,

BLACK : x

Step 6 - Time = Time + 1 = 6, f[y] = 6

WHITE : w

GRAY : u, v

BLACK : x, y

LectureNotes.in

Step 7: Time = Time + 1 = 7 , d[w] = 7 , π[w] = v

WHITE:

GRAY: u, v, w

BLACK: x, y

Step 8 - Time = Time + 1 = 8 , f[w] = 8

WHITE :

GRAY: u, v.

BLACK: x, y, w

Step-9 - Time = Time + 1 = 9, f[v] = 9

WHITE :

GRAY: u

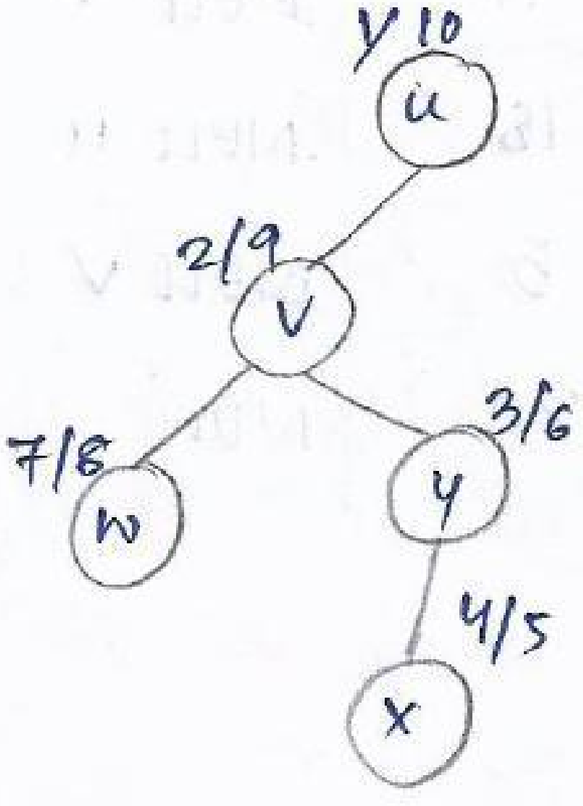
BLACK: x, y, w, v

Step 10 - Time = Time + 1 = 10 , f[u] = 10

WHITE:

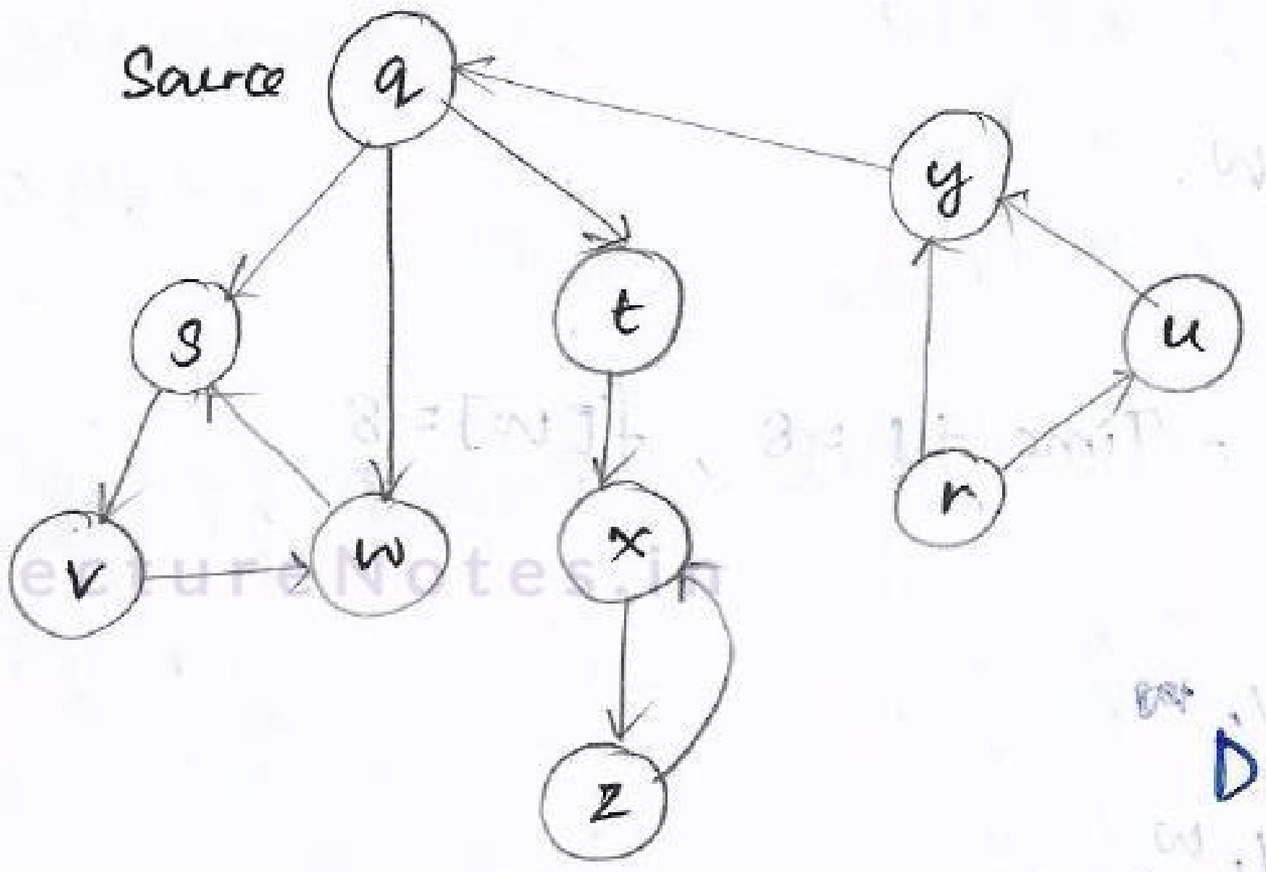
GRAY:

BLACK: x, y, w, v, u



$x \to y \to w \to v \to u$

Q. Traverse the following graph using DFS.



Dt → 20th Mar 18
Tuesday

Sol.

Vertex	Adj (Vertex)	d(Vertex)	f(Vertex)	π(Vertex)
q	s, w, t	1	14	NULL
s	v	2	7	~~NULL~~ q
v	w	3	6	~~NULL~~ s
u	y	16	19	~~NULL~~ r
t	x	8	13	~~NULL~~ q
x	z	9	12	~~NULL~~ t
z	x	10	11	~~NULL~~ x
y	q	17	18	~~NULL~~ u
w	s	4	5	~~NULL~~ v
r	y, u	15	20	NULL

LectureNotes.in

Initially, Time = 0

White: q, s, v, u, t, x, z, y, w, r

Gray:

Black:

Step 1 - Time = 1, d[q] = 1

White: s, v, u, t, x, z, y, w, r

Gray: q

Black:

Step 2 - Time = 2, d[s] = 2

White: v, u, t, x, z, y, w, r

Gray: q, s

Black:

Step 3 - Time = 3, d[v] = 3

White: u, t, x, z, y, w, r

Gray: q, s, v

Black:

Step 4 - Time = 4, d[w] = 4

White: u, t, x, z, y, r

Gray: q, s, v, w

Black:

Step 5 - Time = 5, f[w] = 5

White: u, t, x, z, y, r

Gray: q, s, v

Black: w

Step 6 - Time = 6, f[v] = 6

White: u, t, x, z, y, r

Gray: q, s

Black: w, v

Step 7 - Time = 7, f[s] = 7

White: u, t, x, z, y, r

Gray: q

Black: w, v, s

Step 8 - Time = 8, d[t] = 8

White: u, x, z, y, r

Gray: q, t

Black: w, v, s

Step 9 - Time = 9, d[x] = 9

White: u, z, y, r

Gray: q, t, x

Black: w, v, s

LectureNotes.in

Step 10 Time = 10, d[z] = 10

White: u, y, r

Gray: q, t, x, z

Black: w, v, s

Step 11 Time = 11, f[z] = 11

White: u, y, r

Gray: q, t, x

Black: w, v, s, z

Step 12 Time = 12, f[x] = 12

White: u, y, r

Gray: q, t

Black: w, v, s, z, x

Step 13 Time = 13, f[t] = 13

White: u, y, r

Gray: q

Black: w, v, s, z, x, t

Step 14 Time = 14, f[q] = 14

White: u, y, r

Gray:

Black: w, v, s, z, x, t, q

Let the vertex r be another source vertex.

Step 15 Time = 15, d[r] = 15

White: u, y

Gray: r

Black: w, v, s, z, x, t, q

Step 16 Time = 16, d[u] = 16

White: y

Gray: r, u

Black: w, v, s, z, x, t, q

Step 17 Time = 17, d[y] = 17

White:

Gray: r, u, y

Black: w, v, s, z, x, t, q

Step 18 Time = 18, f[y] = 18

White:

Gray: r, u

Black: w, v, s, z, x, t, q, y

Step 19 Time = 19, f[u] = 19

White:

Gray: r

Black: w, v, s, z, x, t, q, y, u

Step 20 Time = 20, f[r] = 20

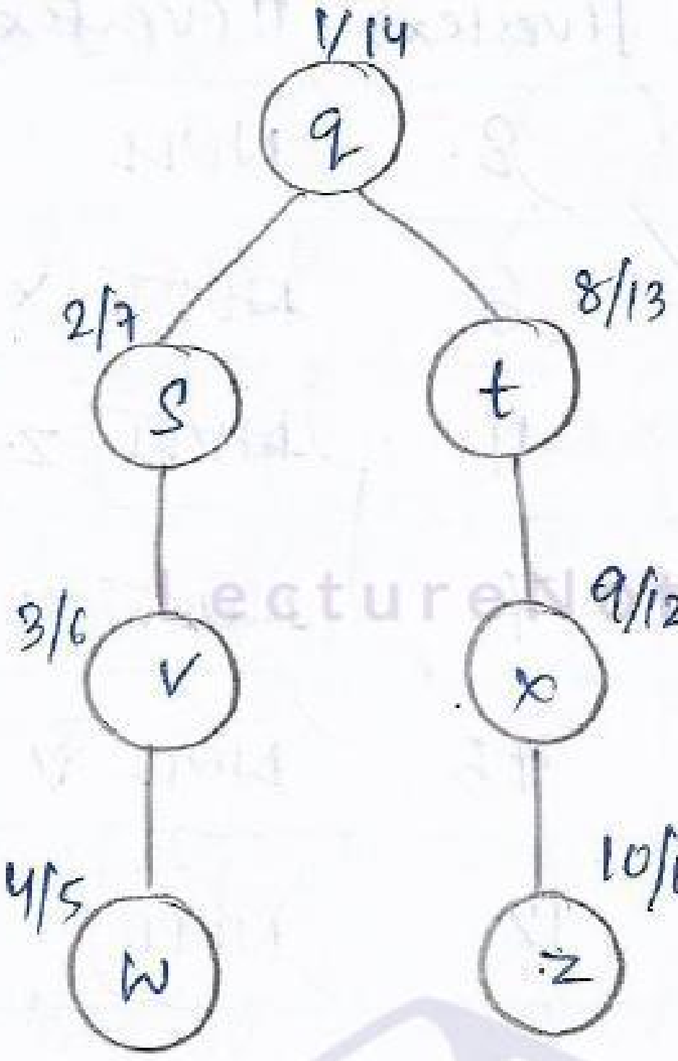
White:

Gray:

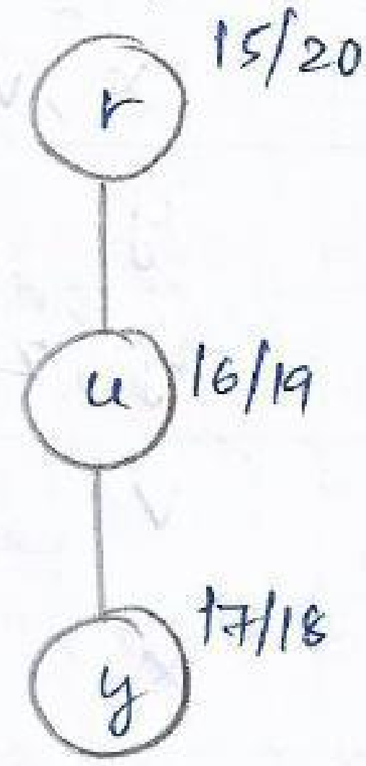
Black: w, v, s, z, x, t, q, y, u, r

LectureNotes.in

DFS forest tree →

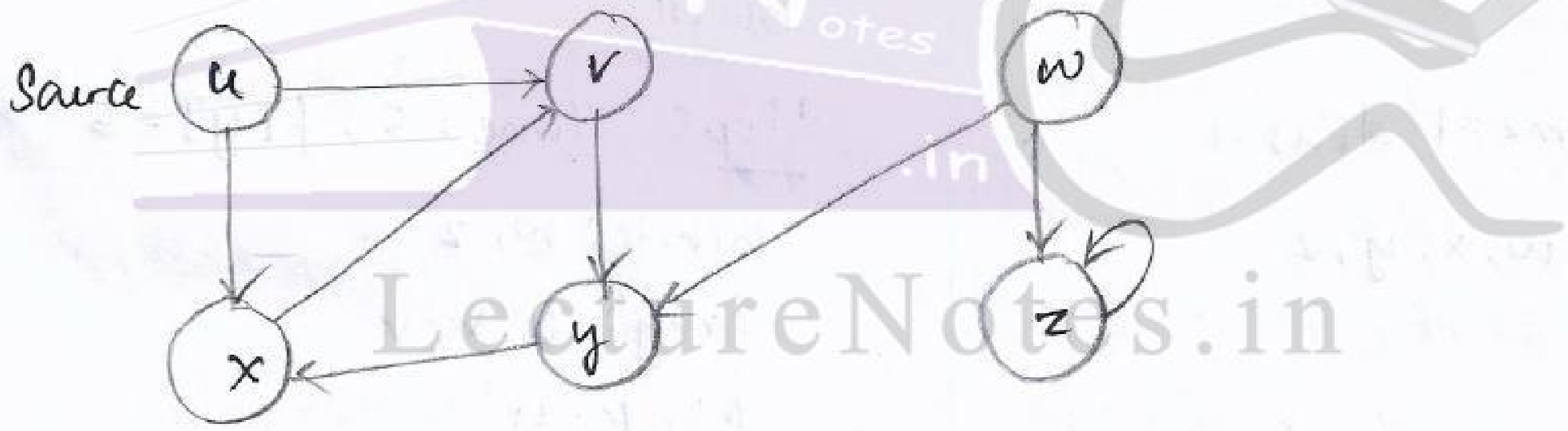


DFS Tree 1



DFS Tree 2

Q. Explain the DFS of following graph.



LectureNotes.in

Sol.

Vertex	Adj(Vertex)	d(Vertex)	f(Vertex)	π(Vertex)
u	x, v	1	8	NULL
v	y	3	6	~~NULL~~ x
w	y, z	10	11	~~NULL~~ z
x	v	2	7	~~NULL~~ u
y	x	4	5	~~NULL~~ v
z	z	9	12	NULL

Initially, Time = 0

White: u, v, w, x, y, z

Gray:

Black:

Step 1, Time = 1, d[u] = 1

White: v, w, x, y, z

Gray: u

Black:

Step 2 Time = 2, d[x] = 2

White: v, w, y, z

Gray: u, x

Black:

Step 3 Time = 3, d[v] = 3

White: w, y, z

Gray: u, x, v

Black:

Step 4 Time = 4, d[y] = 4

White: w, z

Gray: u, x, v, y

Black:

Step 5 Time = 5, f[y] = 5

White: w, z

Gray: u, x, v

Black: y

Step 6: Time = 6, f[v] = 6

White: w, z

Gray: u, x

Black: y, v

Step 7: Time = 7, f[x] = 7

White: w, z

Gray: u

Black: y, v, x

LectureNotes.in

Step 8 Time = 8, f[u] = 8

White: w, z

Gray:

Black: y, v, x, u

let the vertex z be source vertex.

Step 9 Time = 9, d[z] = 9

White: w

Gray: z

Black: y, v, x, u

Step 10 Time = 10, d[w] = 10

White:

Gray: z, w

Black: y, v, x, u.

Step 11 Time = 11, f[w] = 11

White:

Gray: z

Black: y, v, x, u, w

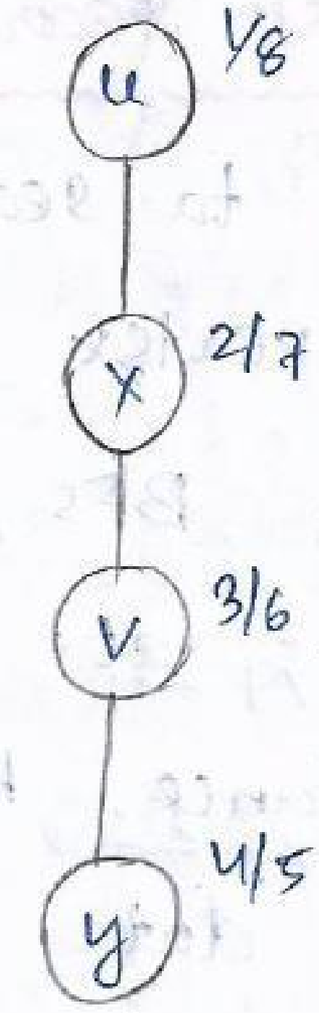
Step 12 Time = 12, f[z] = 12

White:

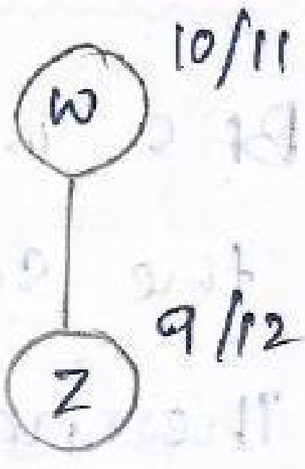
Gray:

Black: y, v, x, u, w, z

DFS forest Tree



DFS Tree 1



DFS Tree 2

LectureNotes.in

Dt → 23rd Mar 18
Friday

2. BFS (Best First Search)

* BFS is used to search a node in a graph G = (V, E) from the source vertex S.

* The name is BFS because the algorithm process the vertex at distance K before to process all the vertex at distance K+1.

* It uses the data structure Q to implement BFS.

d(v): Start time of vertex
d(v) = ∞ - The vertex V is not reachable from the source vertex
π(v) - Parent of vertex V.

* The algorithm uses the following 3 colors -

1. White
2. Gray (Queue)
3. Black

1. White →

Initially, all vertex are white ex Source vertex

2. Gray (Queue) →

The vertex which is discovered first time.

Initially, Source vertex is Gray in color.

3. **Black** →

The vertex which are already processed are in Black Color.

* BFS algorithm can be used in both directed and un-directed graph..

(Insert from Right-side, Delete from Left-side)

BFS Algorithm

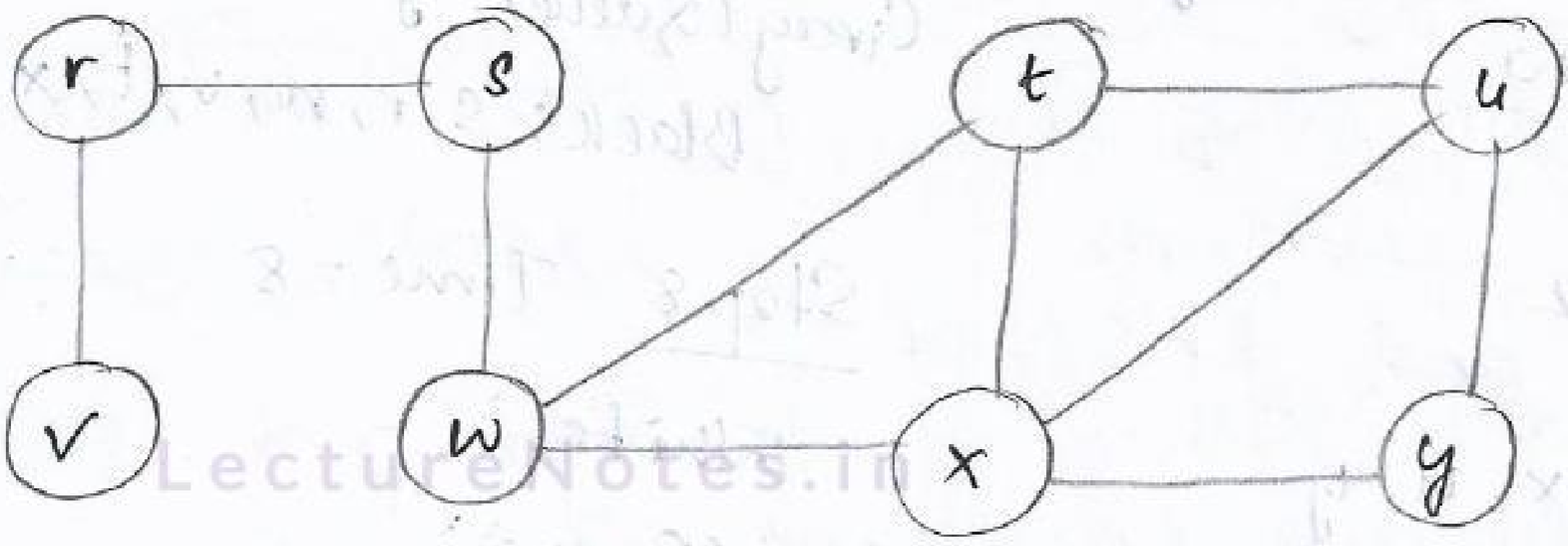
```
BFS (G, S)
1. FOR EACH VERTEX u ∈ V[G] - {s}
2.         d[u] ← ∞          COLOR[u] ← WHITE
3.
4.         π[u] ← NULL
   [END FOR]
5.  d[s] ← 0
6.  π[s] ← NULL
7.  COLOR[s] ← GRAY
8.  Q ← ϕ
9.  ENQUEUE(Q, s) // ENQUEUE is an operation to
                     insert an element into
                     a queue
```

LectureNotes.in

```
10. WHILE Q IS NOT EMPTY
11.     u ← DQUEUE(Q) // DQUEUE: is an operation to
                          delete from queue
12.     FOR EACH VERTEX V ∈ Adj(u)
13.         IF (COLOR[V] ← WHITE)
14.             ENQUEUE (Q, V)
                d[V] ← d[u]+1
15.             π[V] ← u
16.
                COLOR[V] ← GRAY
17.
            [END IF]
        [END FOR]
18.     COLOR[u] ← BLACK
  [END WHILE]

PRINT-PATH (G, S, V)
/* G=(V, E), S: Source Vertex, V: Destination */
1. IF (S=V)
2.     PRINT S
   ELSE
3. IF (π[V]= NULL)
4.     PRINT "NO PATH"
.  ELSE
5.     PRINT_PATH (G, S, π(V))
6.     PRINT(V)
   [END IF]
```

Q. Find the BFS of the following graph from the source vertex s



Sol.

	Vertex	Adj(v)	d(v)	π(v)
Source →	s	r, w	0	NULL
	r	v, s	~~∞~~ 1	~~NULL~~ s
	v	r	~~∞~~ 2	~~NULL~~ r
	w	s, t, x	~~∞~~ 1	~~NULL~~ s
	t	w, x, u	~~∞~~ 2	~~NULL~~ w
	x	w, t, u, y	~~∞~~ 2	NULL w
	u	t, x, y	~~∞~~ 3	~~NULL~~ t
	y	u, x	~~∞~~ 3	~~NULL~~ x

Initial Step Time = 0

White : r, v, w, t, x, u, y

Gray (Queue) : s

Black :

LectureNotes.in

Step-1 Time = time + 1 = 1

White : v, t, x, u, y
Gray (Queue) : r, w
Black : s

Step-2 Time = 2

White : t, x, u, y
Gray (Queue) : w, v
Black : s, r

Step-3 Time = 3

White : u, y
Gray (Queue) : v, t, x
Black : s, r, w

Step-4 Time = 4

White : u, y
Gray (Queue) : t, x
Black : s, r, w, v

Step-5 Time = 5

White : y
Gray (Queue) : x, u
Black : s, r, w, v, t

Step-6 Time = 6

White :
Gray (Queue) : u, y
Black : s, r, w, v, t, x

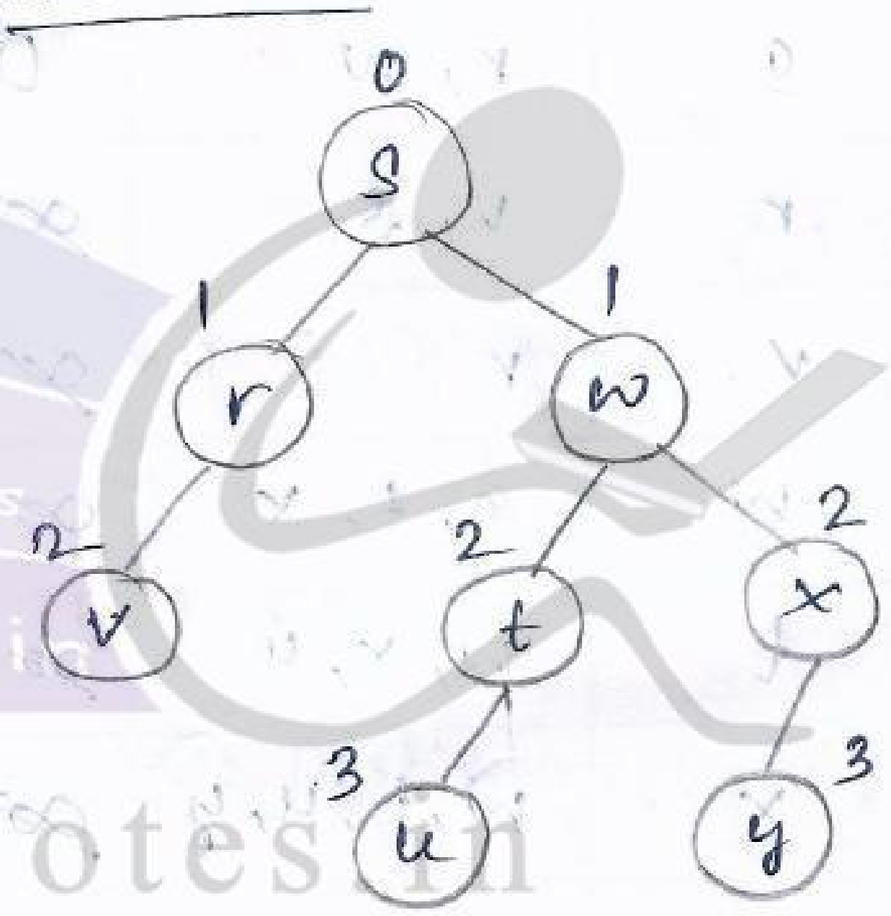
Step-7 Time = 7

White :
Gray (Queue) : y
Black : s, r, w, v, t, x, u

Step 8 Time = 8

White :
Gray (Queue) :
Black : s, r, w, v, t, x, u, y

BFS Tree



LectureNotes.in

Difference between DFS and BFS →

DFS	BFS
→ It uses the data Structure 'Stack.'	→ It uses the data structure 'Queue'.
→ In DFS, the node at distance k+1 is processed first before to process the node at distance k.	→ In BFS, the node at distance k is processed first before to process the node at distance k+1.
→ Here, we can have Forest Tree. because if Stack is empty, then another node is considered as Source node to continue the process.	→ Here, we can specify only 1 Tree because if Queue is empty, then also same nodes are in white node, another vertex can't be considered as Source node.

NOTE :

→ BFS algorithm can be used to find the shortest path (the number of edges) between source to all other vertex.

→ Hence, BFS algorithm can also be used to solve Single Source Shortest Path Problem. if the weight of all edges is same.

Analysis of BFS

From the Algorithm in Pg. 31,

Loop 1 will be executed for mode of $(V-1)$.

DQUEUE and ENQUEUE are the contant steps.

The loop 10 will process for all edges

Therefore, $T_n = T(V, E) = O(V, E)$

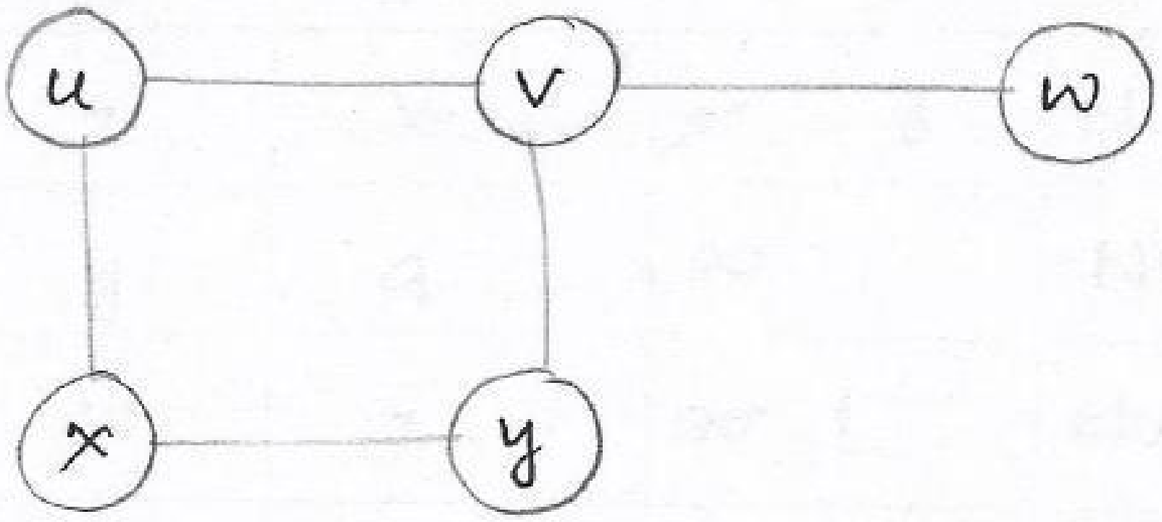
Time complexity for BFS is

$$T(V, E) = O(V, E)$$

LectureNotes.in

LectureNotes.in

Q. Find the BFS of the following graph from the source u.



LectureNotes.in

Sol.

Vertex	Adj(v)	d(v)	π(v)
u	v, x	0	NULL
v	u, y, w	~~∞~~ 1	~~NULL~~ u
w	v	~~∞~~ 2	~~NULL~~ v
x	u, y	~~∞~~ 1	~~NULL~~ u
y	x, v	~~∞~~ 2	~~NULL~~ v

Initial Step, Time = 0

White : v, w, x, y

Gray (Queue) : u

Black :

Step 1 Time = 1

White : w, y

Gray (Queue) : v, x

Black : u

Step 2 Time = 2

White :

Gray (Queue) : x, y, w

Black : u, v

Step 3 Time = 3

White :

Gray (Queue) : y, w

Black : u, v, x

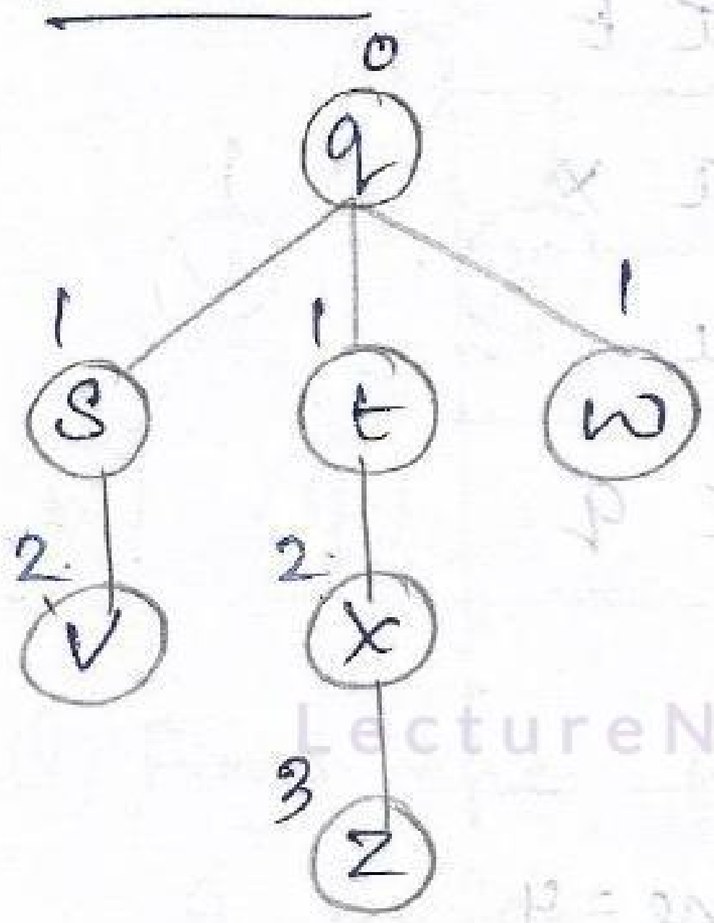
Step 4 Time = 4

White :

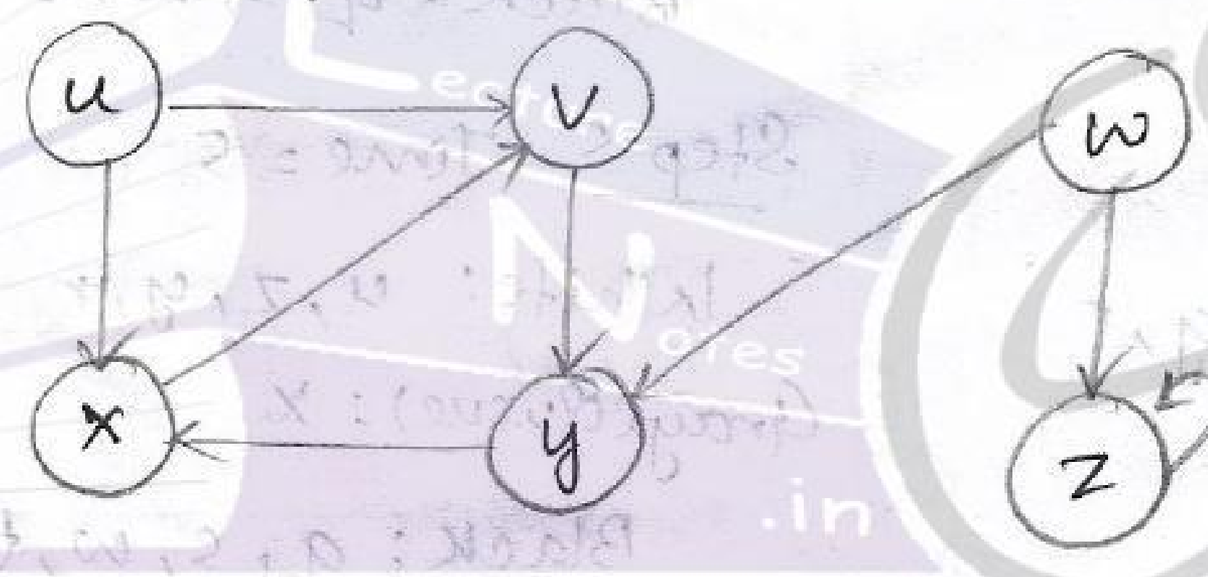
Gray (Queue) :

Black : u, v, x, y, w

BFS Tree



Q. Find the BFS of the following graph with source as u.



Sol.

Vertex	Adj(v)	d(v)	π(v)
u	x, v	0	NULL
v	y	~~∞~~ 1	~~NULL~~ u
w	y, z	∞	NULL
x	v	~~∞~~ 1	~~NULL~~ u
y	x	~~∞~~ 2	~~NULL~~ v
z	z	∞	NULL

Let the source vertex be u

Initial Step, Time=0

White: v, x, y, w, z

Gray (Queue): u

Black:

Step 1 Time = 1

White: y, w, z

Gray (Queue): v, x

Black: u

Step 2 Time = 2

White: w, z

Gray (Queue): x, y

Black: u, v

Step 3 Time = 3

White: w, z

Gray (Queue): y

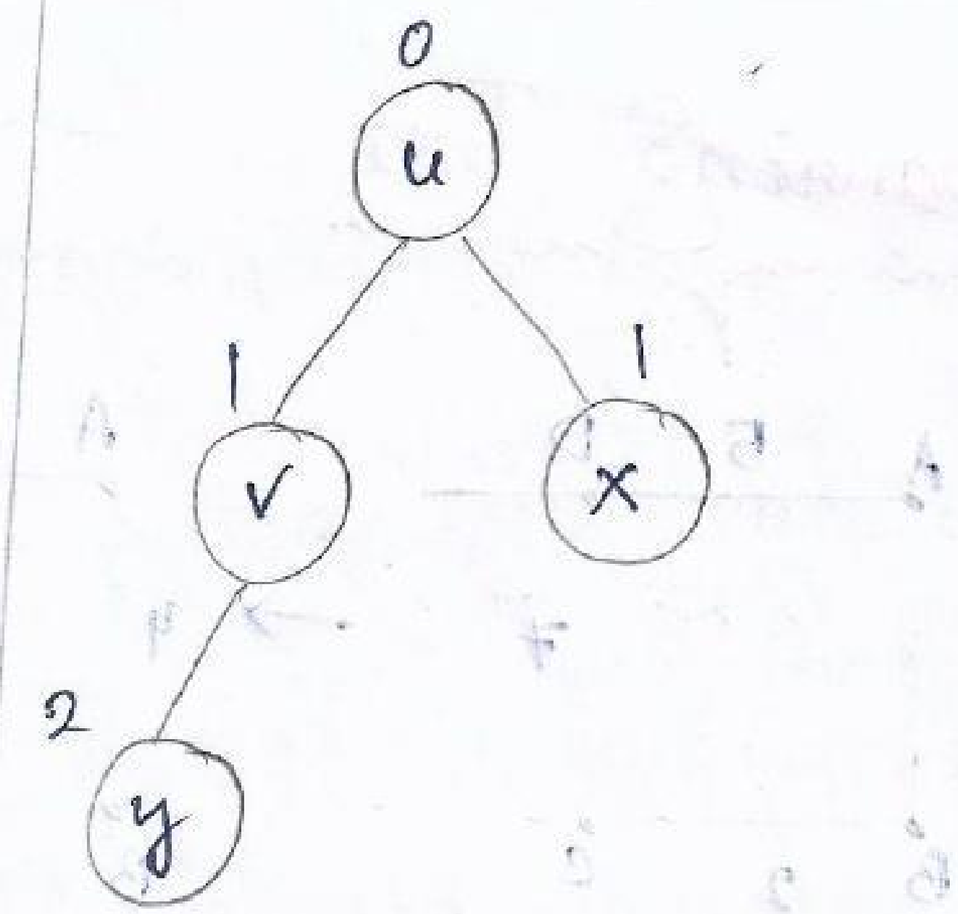
Black: u, v, x

Step 4 Time = 4

White: w, z

Gray (Queue):

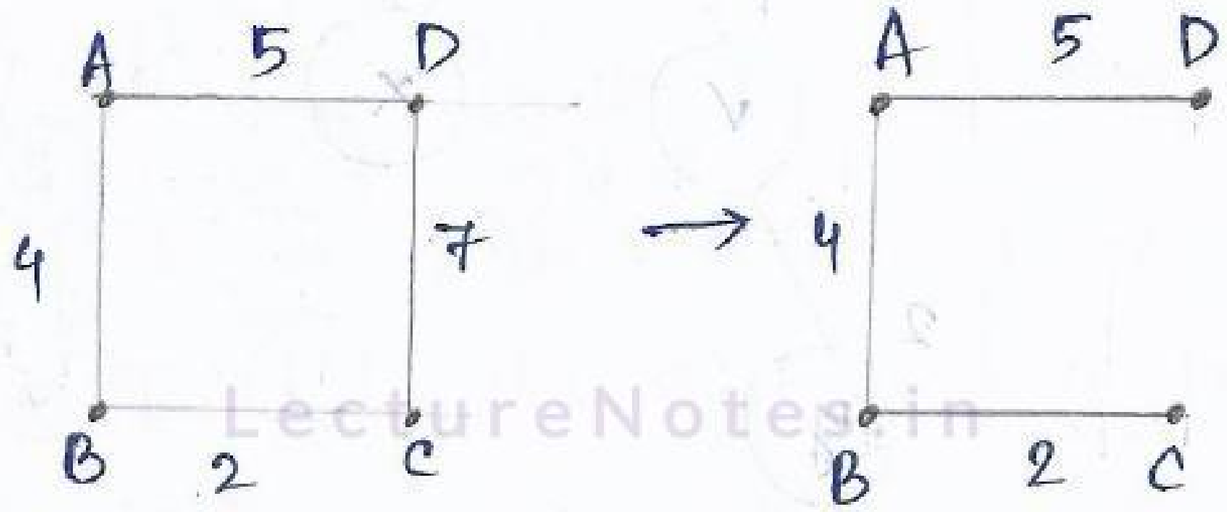
Black: u, v, x, y



LectureNotes.in

Spanning Tree

Dt → 29th Mar 18
Thursday

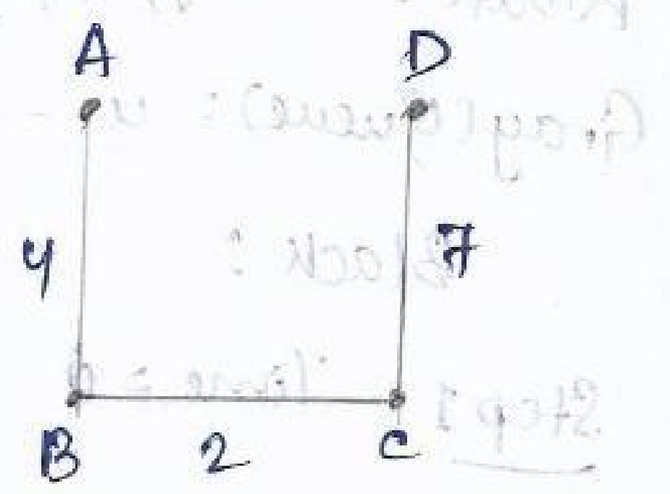


(Graph-1)

A → D → C → B → A

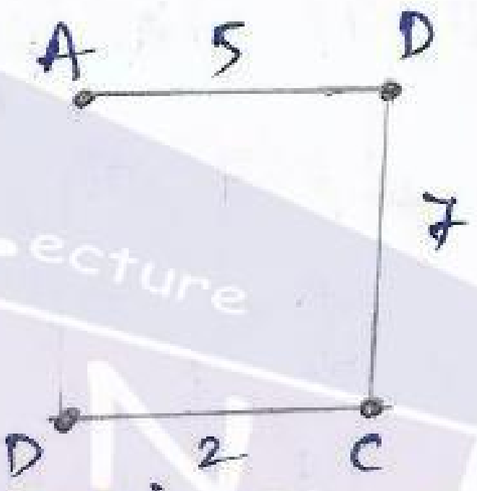
Spanning Tree-1

$W(T_1) = 11$



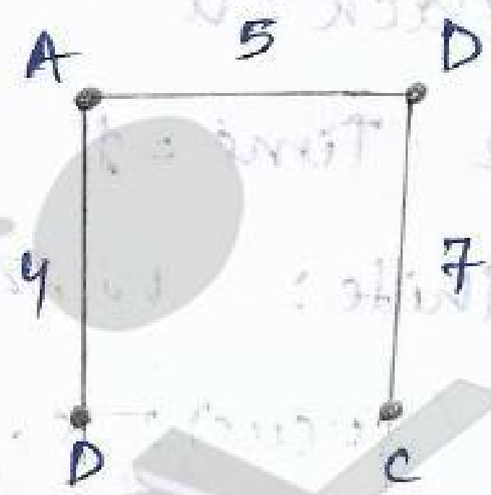
Spanning Tree = 2

$W(T_2) = 13$



Spanning Tree-3

$W(T_3) = 14$



Spanning Tree-4

$W(T_4) = 16$

In the above diagrams, the minimum Spanning Tree is Spanning Tree-1.

* In undirected graph, $G = (V, E)$ is a set of vertices and edges.
* Spanning Tree is a acyclic graph (The graph which doesnot have any cycles) which connects all vertex.

LectureNotes.in

Minimum Spanning Tree → (MST)

→ In undirected graph, $G = (V, E)$ is a set of vertices and edges.

→ We need to find an acyclic subset, $T \subseteq E$ where T forms a spanning Tree, such that, $W(T)$ is minimum.

$$W(T) = \sum_{(u,v) \in T} w(u,v)$$

→ The following two algorithms can be used to find minimum spanning tree of an undirected graph-

1. Kruskal's Algoritam
2. Prim's Algoritam

Safe Edge →

If an edge is added to form a Minimum Spanning Tree 'A', then that edge is called Safe Edge.

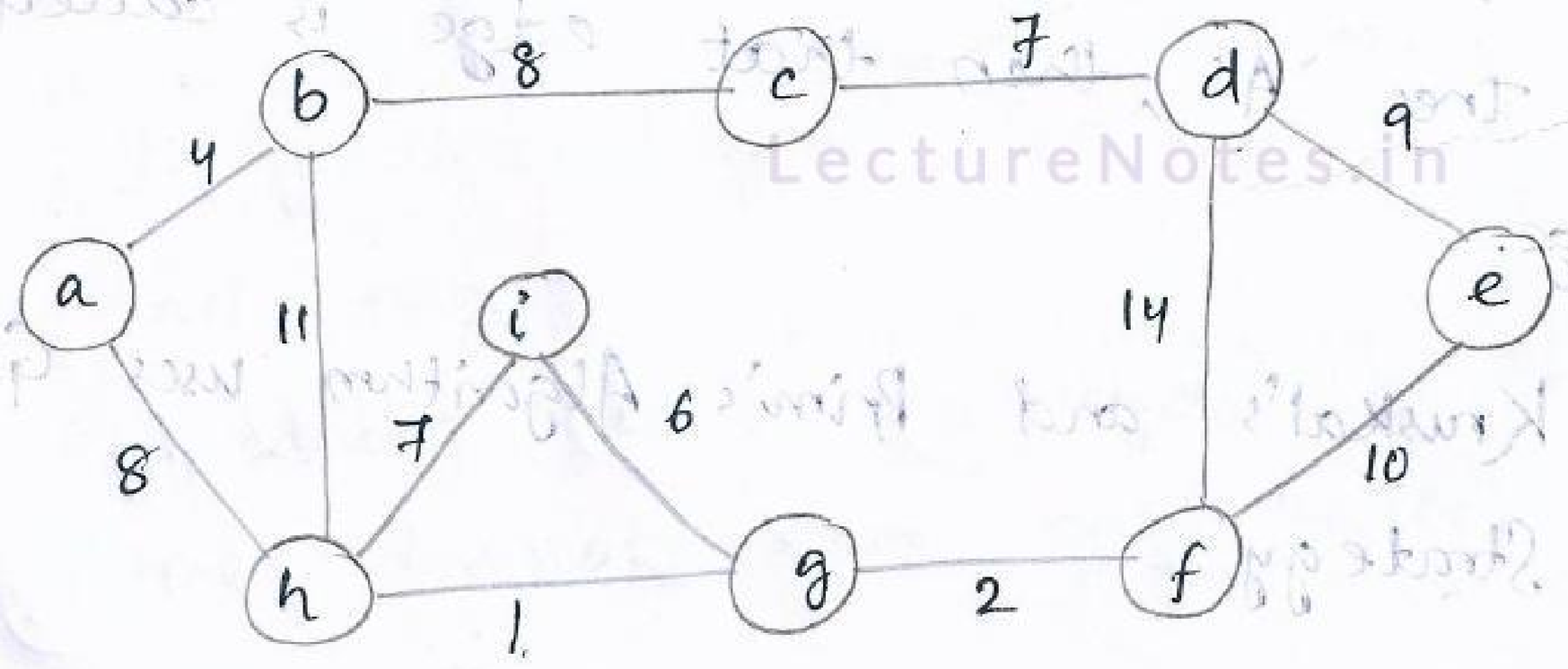
NOTE: Kruskal's and Prim's Algorithm uses Greedy Strategy.

LectureNotes.in

1. Kruskal's Algorithm (Kruskal's MST Algorithm)

* Kruskal's Algorithm, can be used to construct a minimum spanning tree.
* It finds the subset of edges that form a tree that includes every vertex where the total weight of all the edges in the tree is minimum.
* If the graph is not connected, then it finds the Minimum Spanning Tree Forest. (It will form spanning tree for each connected components.)
* Disjoint data set operations are used in Kruskal's Algorithm.

Q. Find the Minimum Spanning Tree of the graph by using Kruskal's Algorithm.



Sol. $A = \{\phi\}$

Edges	Weight
(h, g)	1
(g, f)	2
(a, b)	4
(i, g)	6
(h, i)	7
(c, d)	7
(a, h)	8
(b, c)	8
(d, e)	9
(e, f)	10
(b, h)	11
(d, f)	14

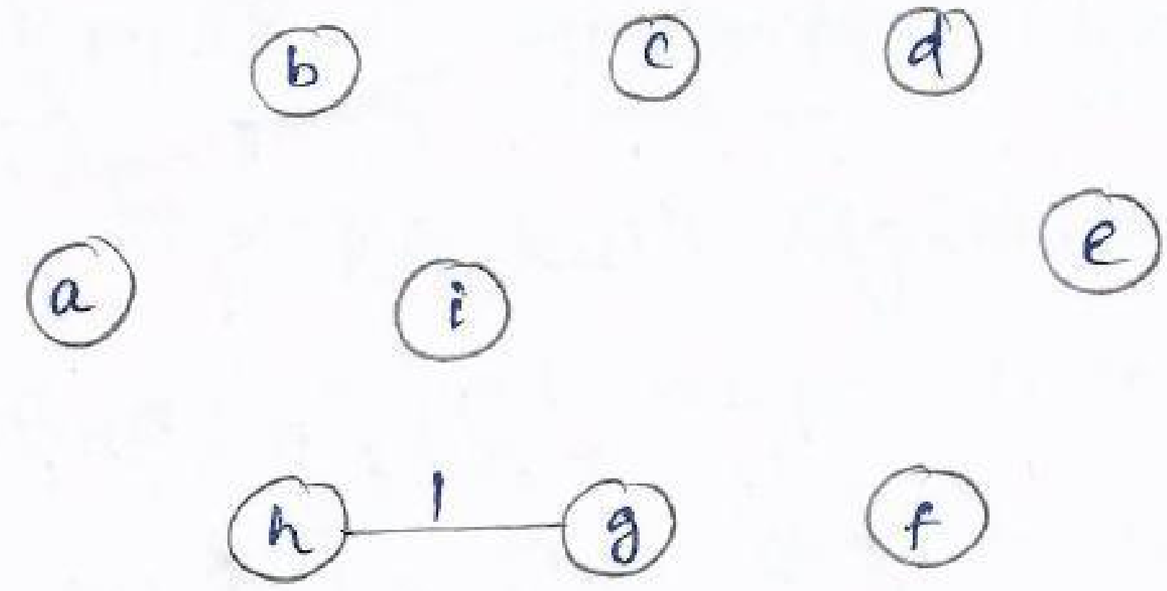
Initially, applying the Make set operation for all vertices.

$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$

Step 1 Edge : (h, g), {a} {b} {c} {d} {e} {f} {g, h} {i}

$A = A \cup (h, g)$

$A = \{(h, g)\}$

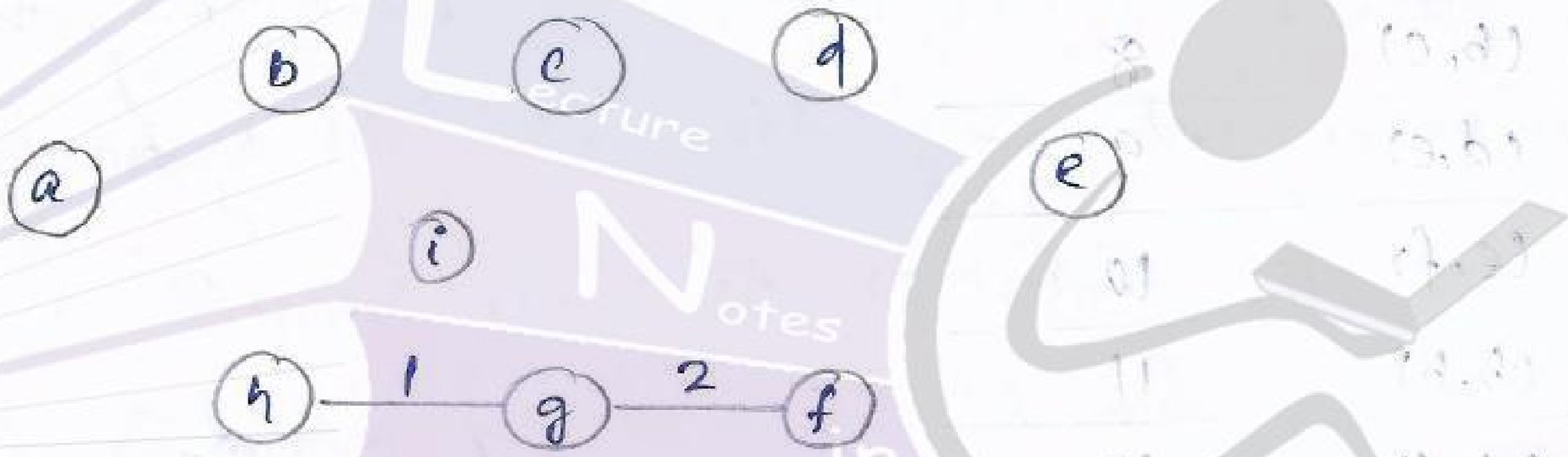


Step 2

Edge : (g, f) {a} {b} {c} {d} {e} {f, g, h} {i}

A = A ∪ {g, f}

A = {(h, g), (g, f)}

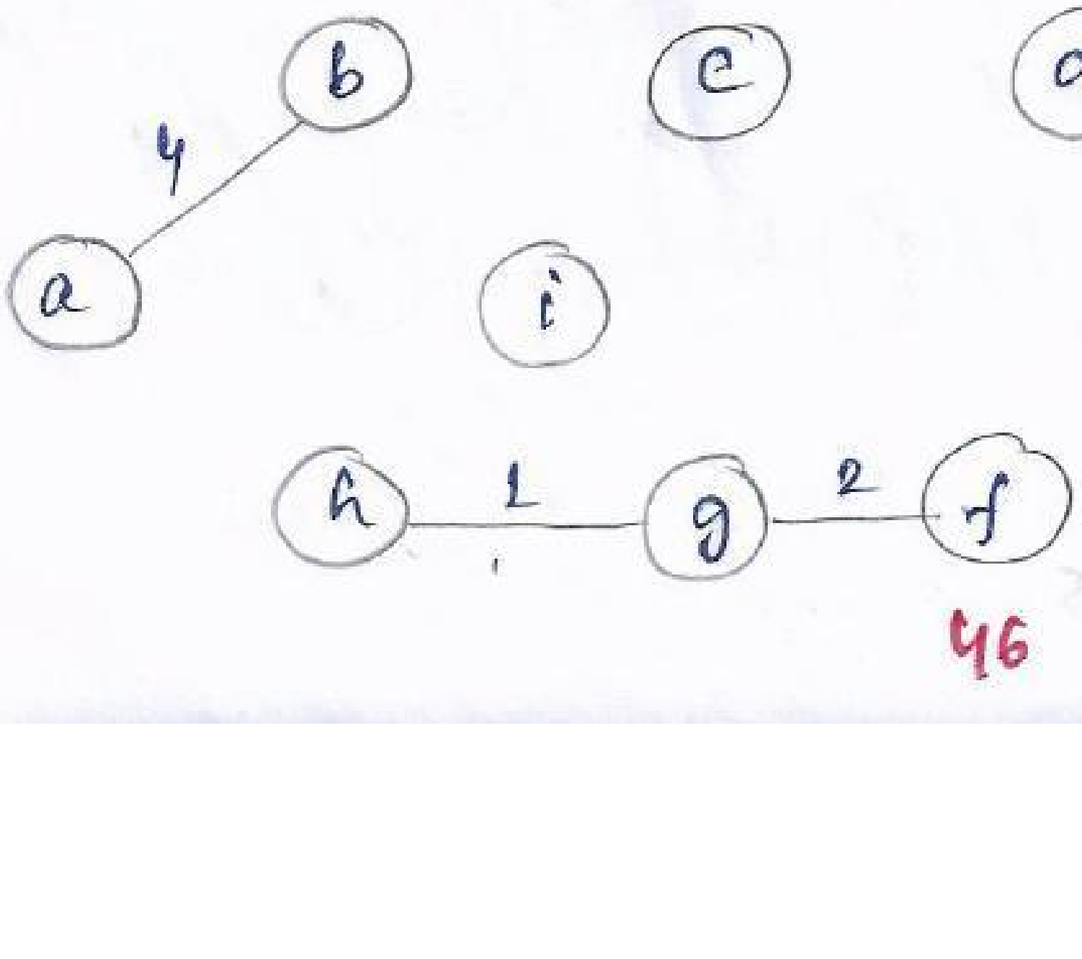


Step 3

Edge : (a, b) {a, b} {c} {d} {e} {f, g, h} {i}

A = A ∪ {a, b}

A = {(h, g) (g, f) (a, b)}

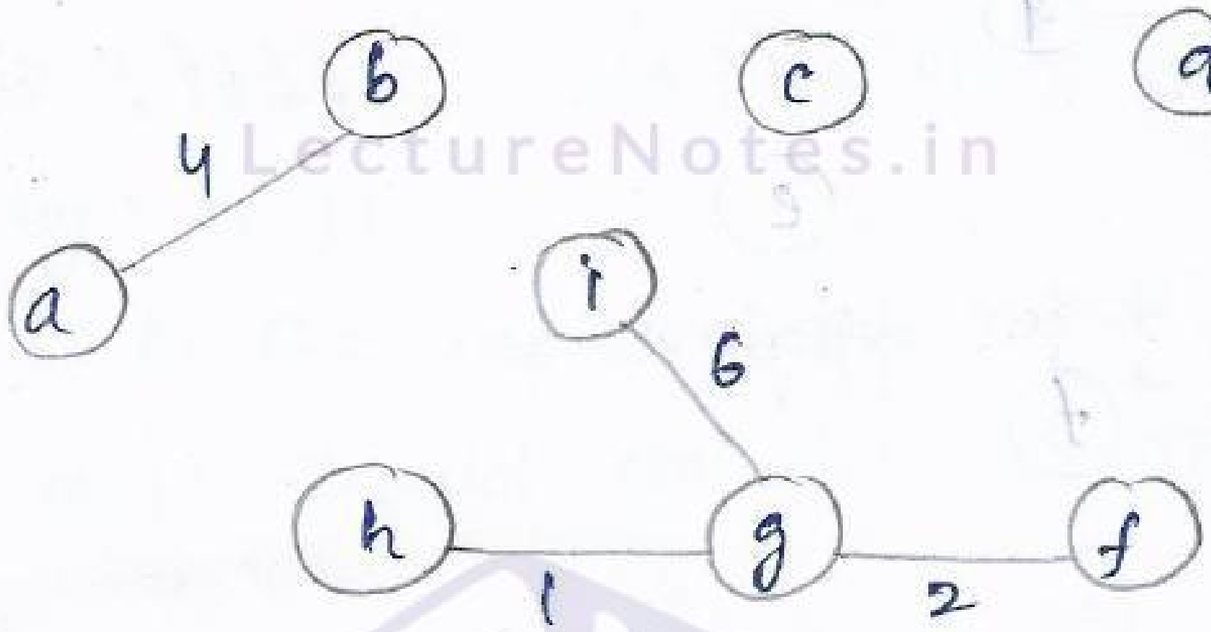


Step 4.

Edge : (i, g) {a, b} {c} {d} {e} {f, g, h, i}

$A = A \cup \{i, g\}$

$A = \{(h, g)\ (g, f)\ (a, b)\ (i, g)\}$



Step 5

Edge : (h, i)

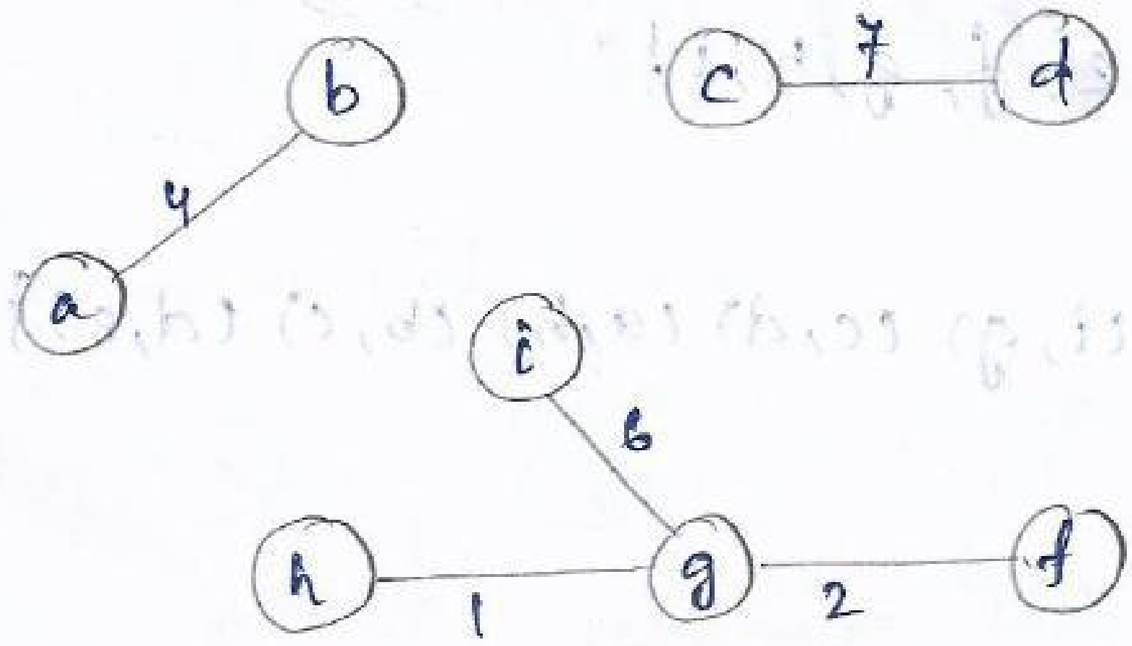
As the representative of h and i are in same set, the (h, i) should not be considered.

Step 6

Edge : (c, d) {a, b} {c, d} {e} {f, g, h, i}

$A = A \cup \{c, d\}$

$A = \{(h, g)\ (g, f)\ (a, b)\ (i, g)\ (c, d)\}$

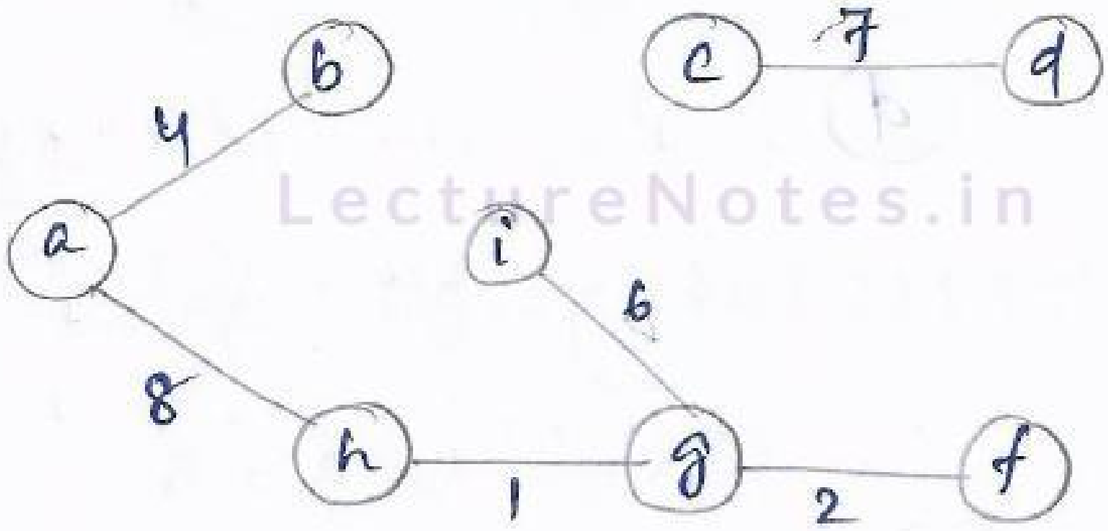


Step 7

Edge : (a, h) {a, h, b, f, g, i} {c, d} {e}

$A = A \cup (a, h)$

$A = \{(h, g)\ (g, f)\ (a, b)\ (i, g)\ (c, d)\ (a, h)\}$

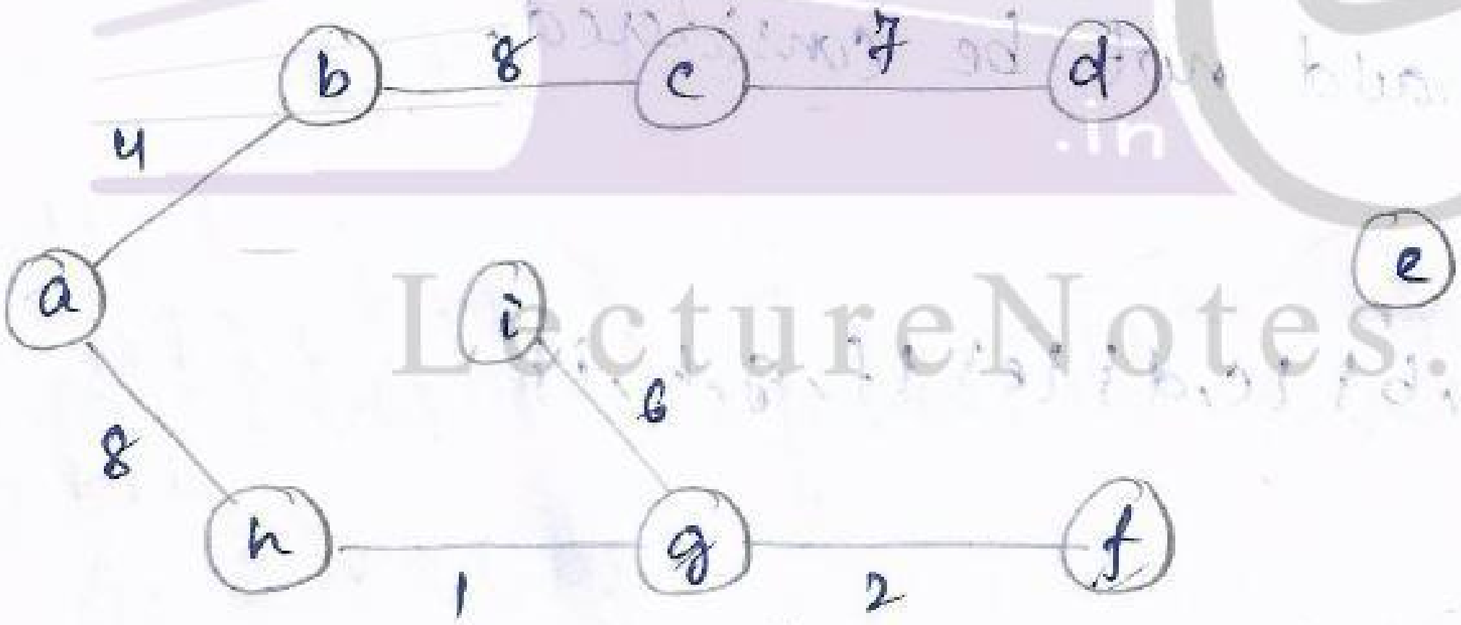


Step 8

Edge : (b, c) {a, b, c, h, f, g, i, d} {e}

$A = A \cup (b, c)$

$A = \{(h, g)\ (g, f)\ (a, b)\ (i, g)\ (c, d)\ (a, h)\ (b, c)\}$

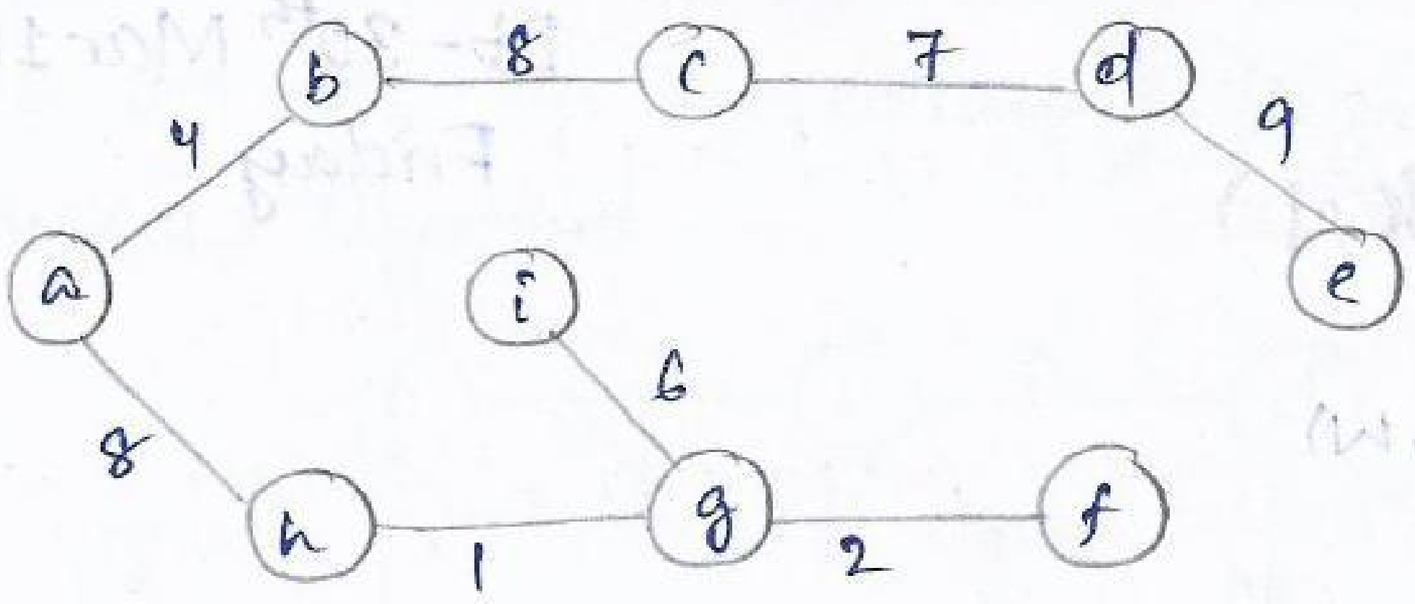


Step 9

Edge : (d, e) {a, b, c, d, e, f, g, h, i}

$A = A \cup (d, e)$

$A = \{(h, g)\ (g, f)\ (a, b)\ (i, g)\ (c, d)\ (a, h)\ (b, c)\ (d, e)\}$



Step 10

Edge : (e, f)

As the representative of e and f are in same set, the (e, f) should not be considered.

Step 11

Edge : (b, h)

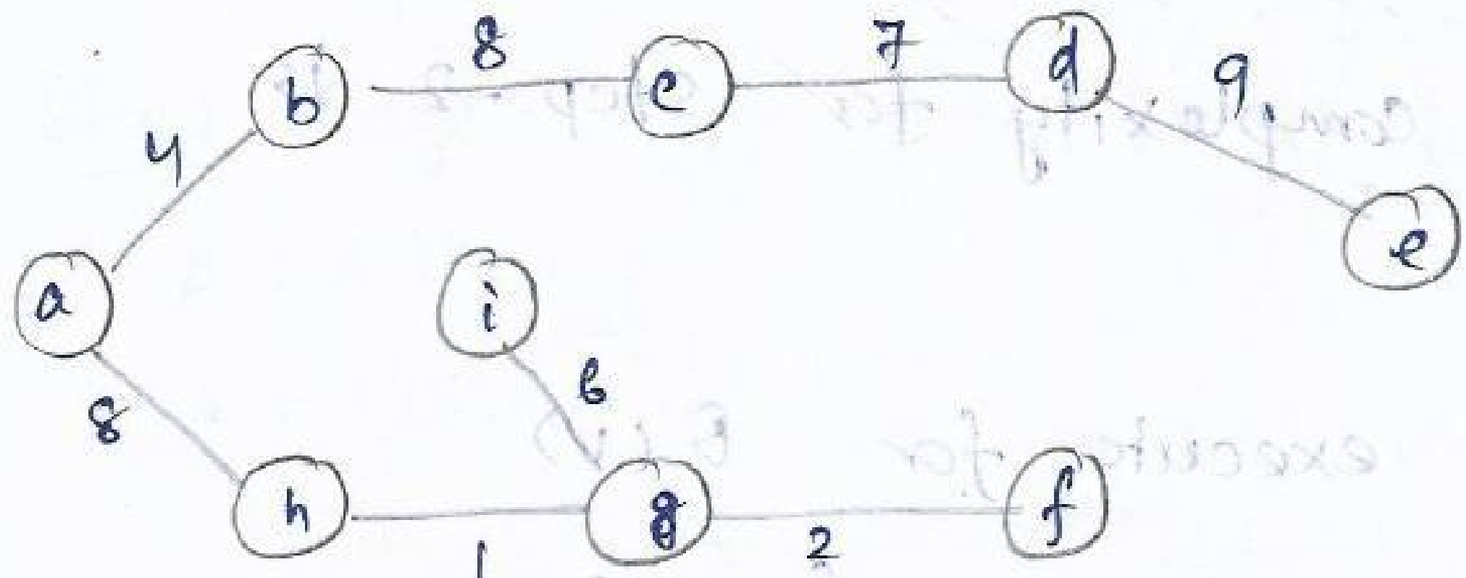
As the representative of b and h are in same set, the (b, h) should not be considered.

Step 12

Edge : (d, f)

As the representative of d and f are in same set, the (d, f) should not be considered.

So, the final diagram is -



So, the minimum cost is 45.

Dt- 30th Mar 18
Friday

Algorithm (Kruskal)

MST_KRUSKAL (G, W)

1. A ← ϕ
2. ARRANGE ALL THE EDGES IN ASCENDING ORDER OF THE WEIGHT
3. FOR EACH VERTEX u ∈ V[G]
4. MAKE_SET(u)

[END FOR]

5. FOR EACH EDGE (u, v) ∈ E[G]
6. IF FIND_SET(u) ≠ FIND_SET(v)
7. A ← A ∪ (u, v)
8. UNION (u, v)

[END IF]

[END FOR]

9. RETURN A

Time Complexity of Kruskal's Algorithm →

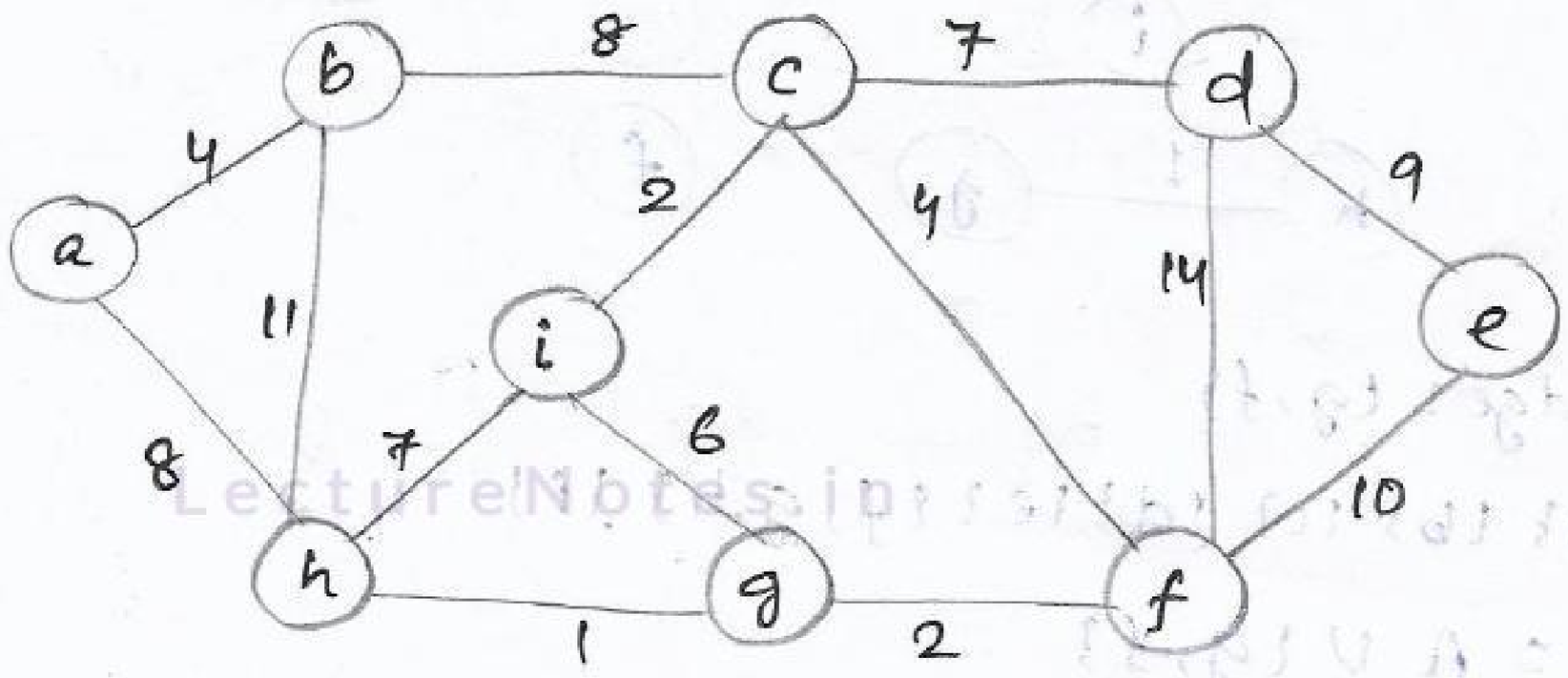
The time complexity for step-2 is

O(E log E)

The loop 3 will execute for O(V)

The loop 5 will execute for O(E)

). Find the MST of the following diagram using Kruskal's Algorithm



Sol.

Edges	Weight
(h, g)	1
(g, f)	2
(i, c)	2
(a, b)	4
(c, f)	4
(i, g)	6
(h, i)	7

Edges	Weight
(c, d)	7
(b, c)	8
(a, h)	8
(d, e)	9
(f, e)	10
(b, h)	11
(d, f)	14

Initially, applying the Make set Operation for all vertices.

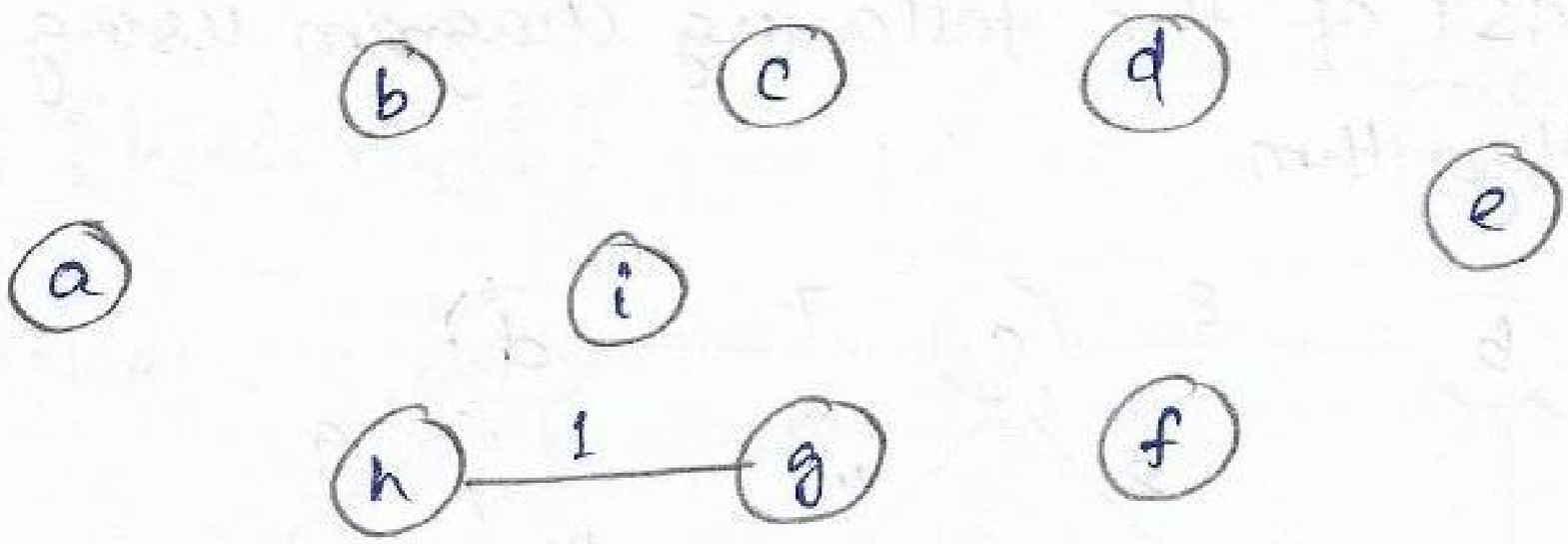
{a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}

Step 1 Edge : (h, g) {a} {b} {c} {d} {e} {f} {g, h} {i}

$A = A \cup (h, g)$

$A = \{(h, g)\}$

HW-1

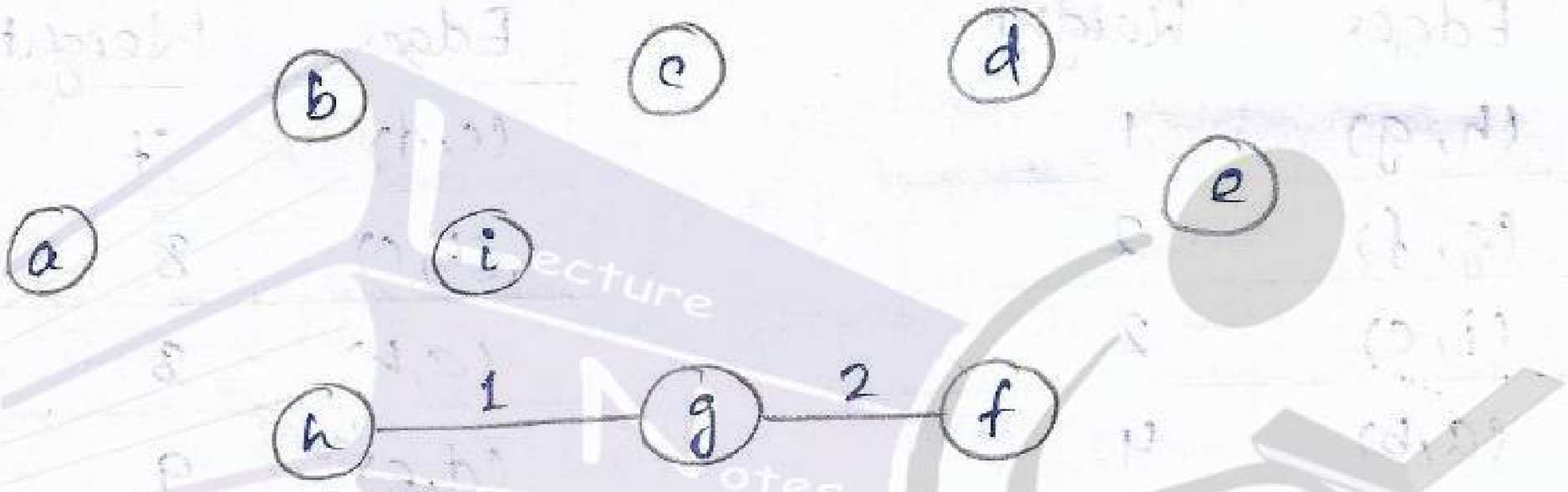


Step 2 Edge : (g, f)

{a} {b} {c} {d} {e} {f, g, h} {i}

A = A ∪ {g, f}

A = {(h, g) (g, f)}

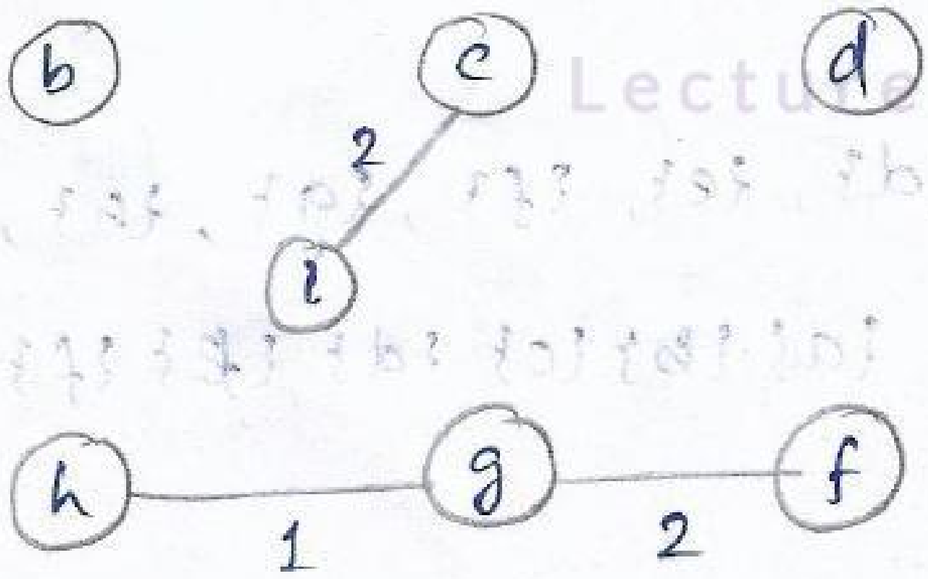


Step 3

Edge : (i, c) {a} {b} {c, i} {d} {e} {f, g, h}

A = A ∪ {i, c}

A = {(h, g) (g, f) (i, c)}

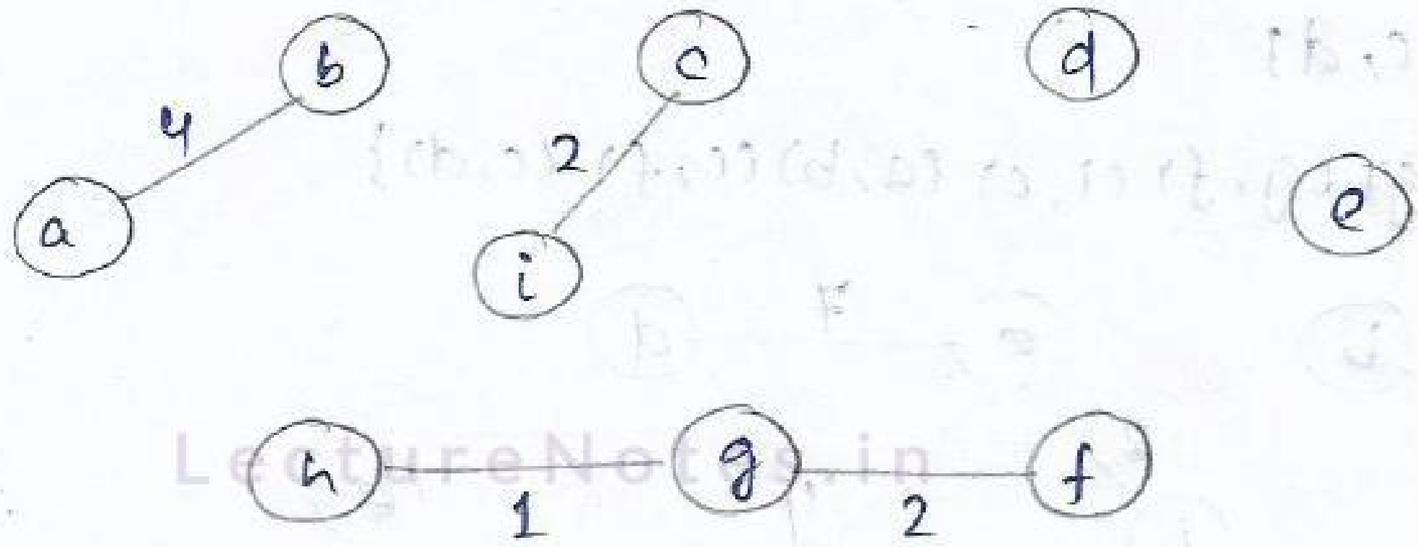


Step 4

Edge : (a, b) {a, b} {c, i} {d} {e} {f, g, h}

$A = A \cup \{a, b\}$

$A = \{(h, g)\ (g, f)\ (i, c)\ (a, b)\}$



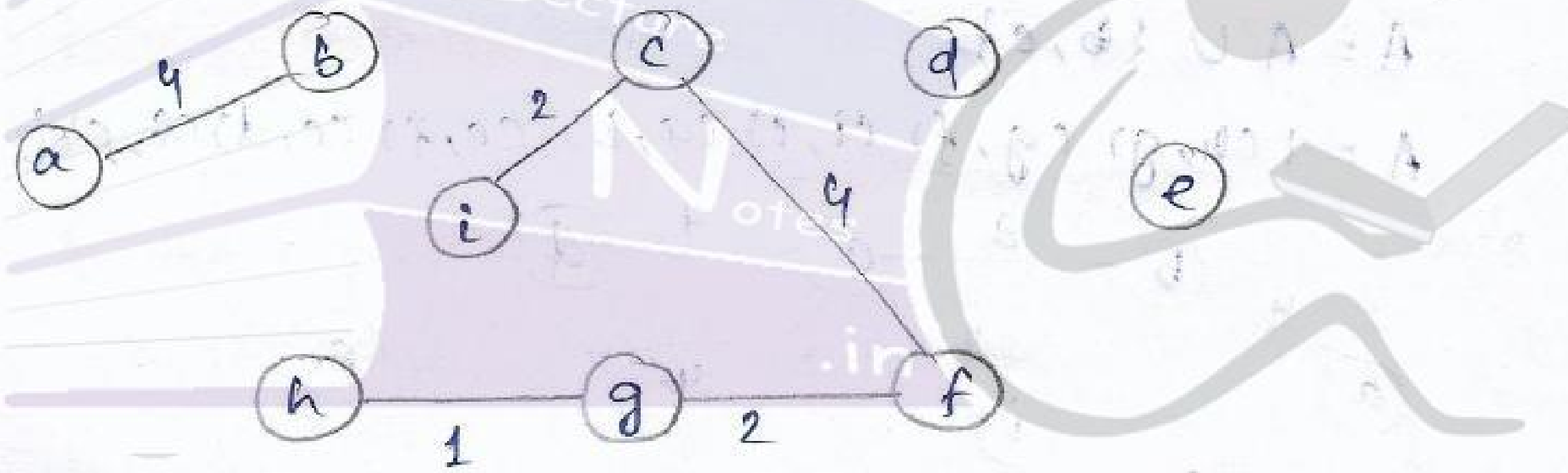
ical's

Step 5

Edge : (c, f) {a, b} {c, i, f, g, h} {d} {e}

$A = A \cup \{c, f\}$

$A = \{(h, g)\ (g, f)\ (i, c)\ (a, b)\ (c, f)\}$



LectureNotes.in

Step 6

Edge : (i, g)

As the representative of i and g are in same set, (i, g) edge should not be considered.

Step 7

Edge : (h, i)

As the representative of h and i are in same set, (h, i) edge should not be considered.

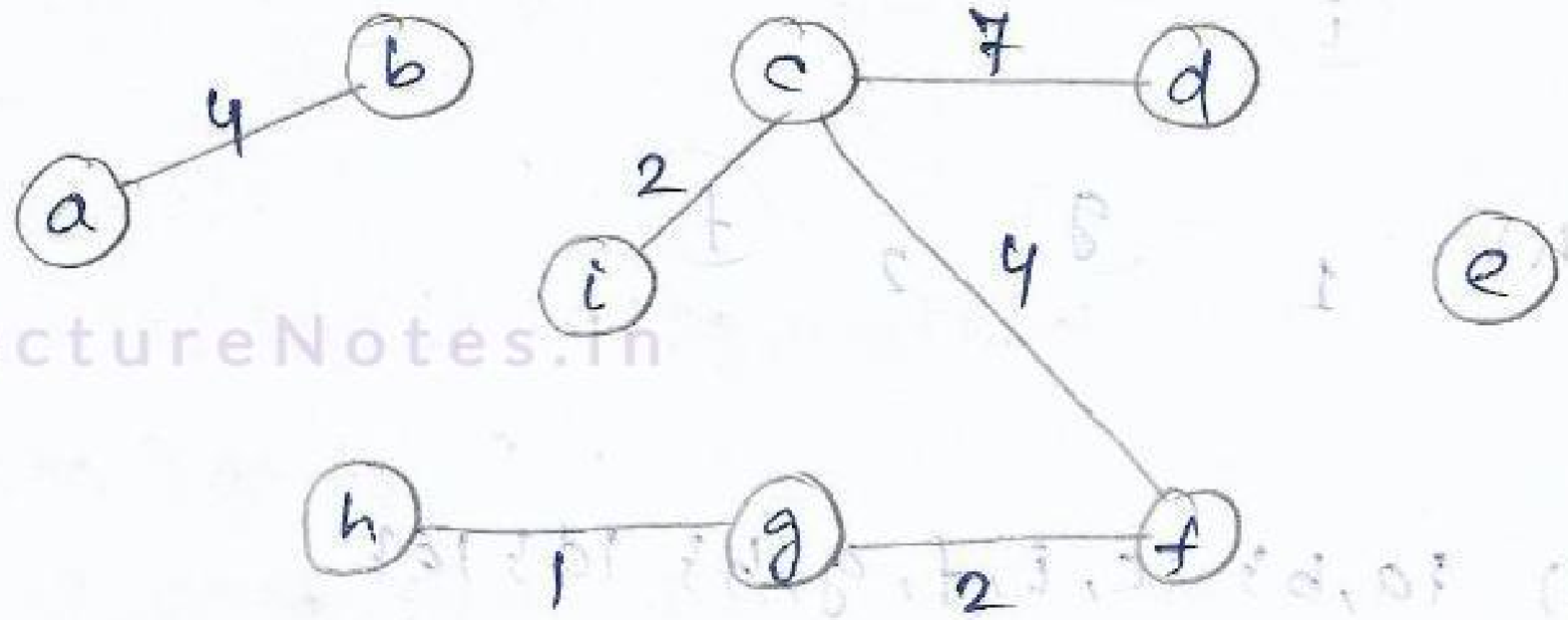
HW-3

Step 8

Edge : (c,d) {a,b} {c,d,i,f,g,h} {e}

$A = A \cup \{c,d\}$

$A = \{(h,g)\ (g,f)\ (i,c)\ (a,b)\ (c,f)\ (c,d)\}$

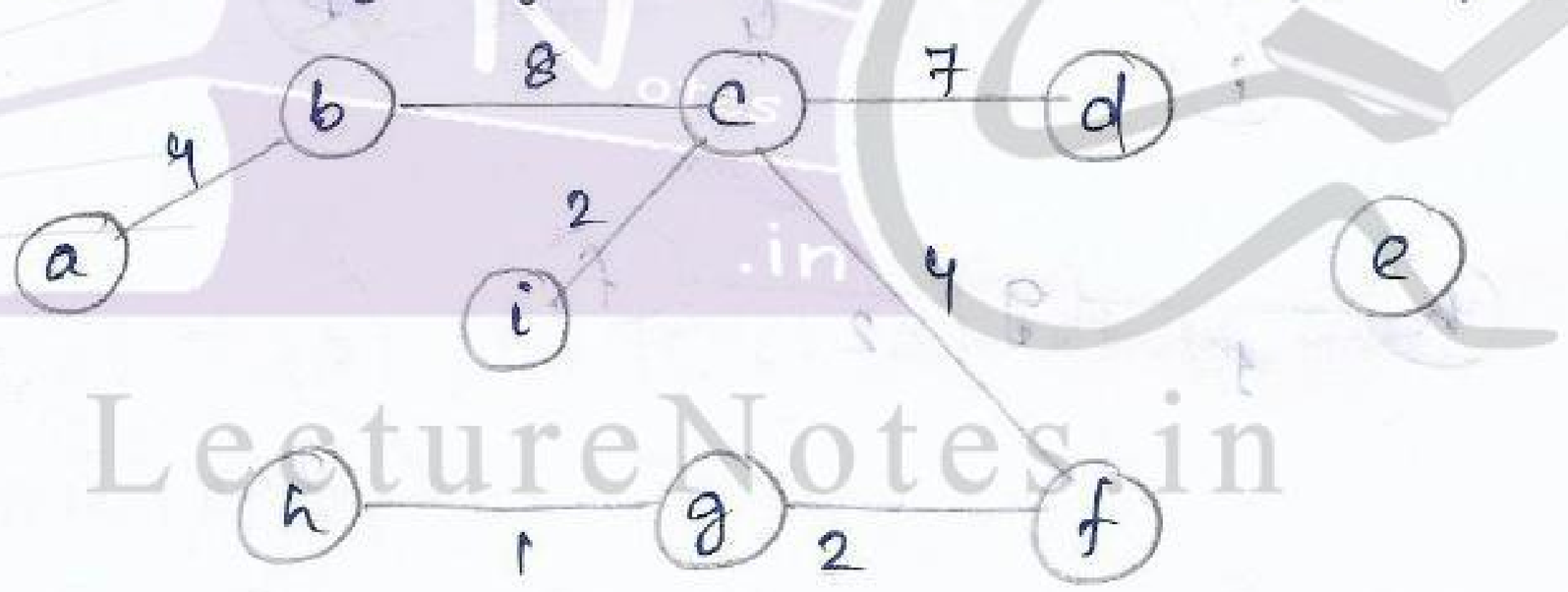


Step 9

Edge : (b,c) {a,b,c,d,i,f,g,h} {e}

$A = A \cup \{b,c\}$

$A = \{(h,g)\ (g,f)\ (i,c)\ (a,b)\ (c,f)\ (c,d)\ (b,c)\}$



Step 10

Edge : (a,b)

As the representative of a and h are in s
set, then (a,b) set should not be considered.

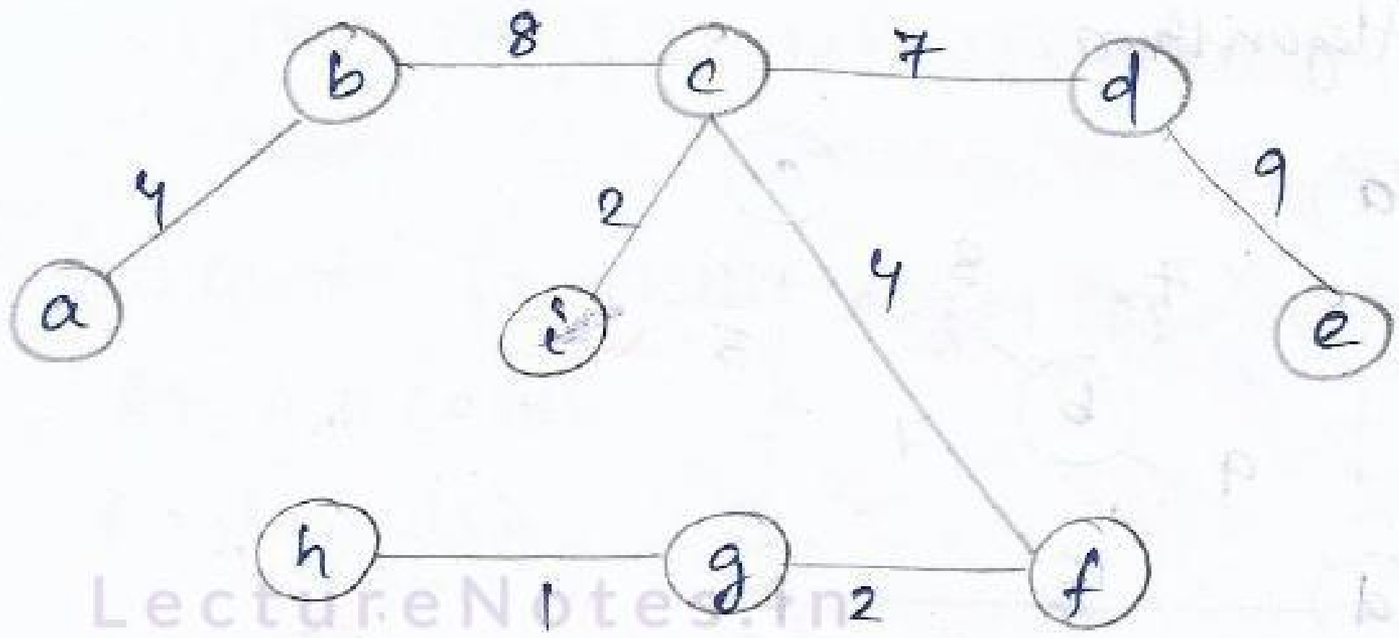
Step 11

Edge : (d,e) {a,b,c,d,e,f,g,h,i}

$A = A \cup \{d,e\}$

HW-4

$A = \{(h,g)\ (g,f)\ (i,c)\ (a,b)\ (c,f)\ (c,d)\ (b,c)\ (d,e)\}$



LectureNotes.in

Step 12

Edge : (f, e)

As the representative of f and e are in same set, so (f, e) edge should not be considered.

Step 13

Edge : (b, h)

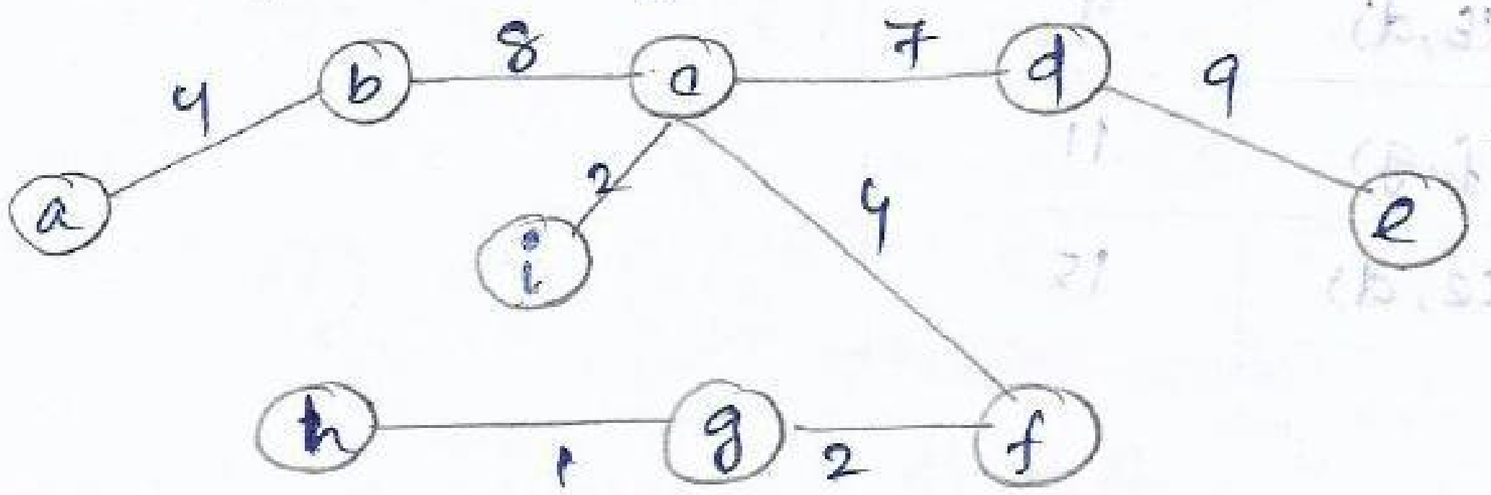
As the representative of b and h are in same set, (b, h) edge should not be considered.

Step 14

Edge : (d, f)

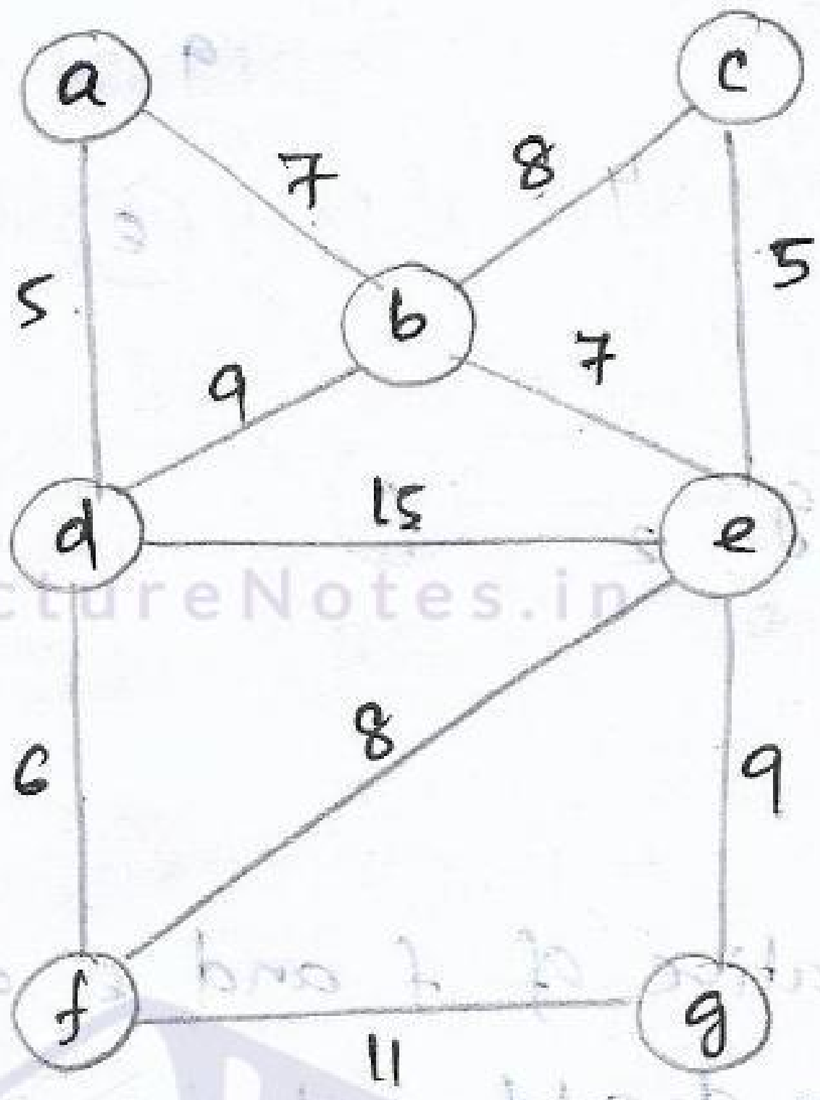
As the representative of d and f are in same set, (d, f) edge should not be considered.

Thus, the final diagram is



The minimum weight is 37.

Q. Find the MST of the following diagram using Kruskal's Algorithm.



Sol.

Edges	Weight
(a,d)	5
(c,e)	5
(d,f)	6
(a,b)	7
(b,e)	7
(b,c)	8
(e,f)	8
(e,g)	9
(b,d)	9
(f,g)	11
(e,d)	15

LectureNotes.in

HW-6

Initially, applying the Make Set operation for all vertex.

{a}, {b}, {c}, {d}, {e}, {f}, {g}

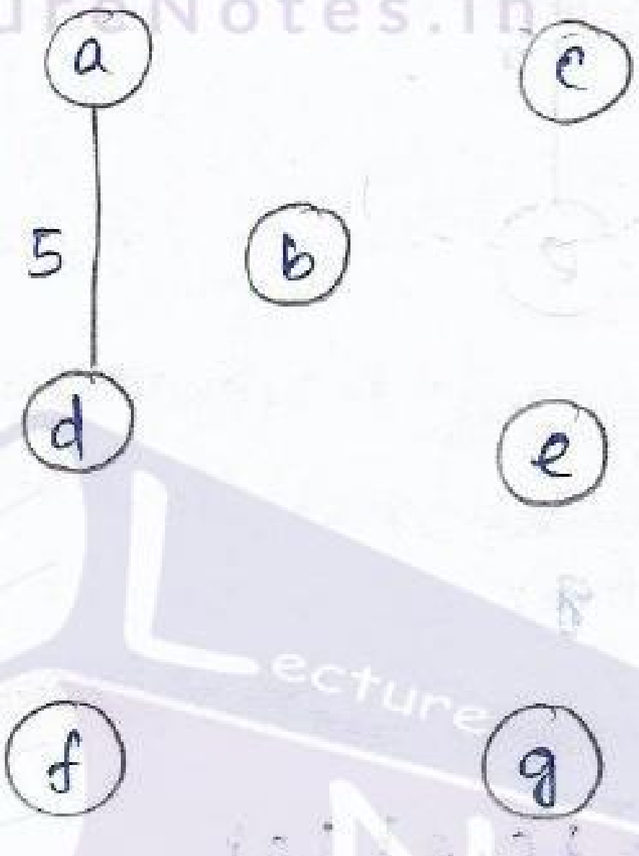
Step 1

Edge : (a, d) {a, d} {b} {c} {e} {f} {g}

$A = A \cup (a, d)$

$A = \{(a, d)\}$

LectureNotes.in



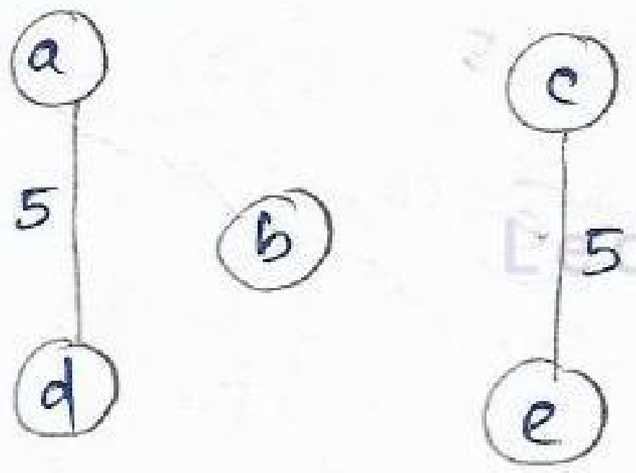
Step 2

Edge : (c, e) {a, d} {b} {c, e} {f} {g}

$A = A \cup (c, e)$

$A = \{(a, d) (c, e)\}$

LectureNotes.in



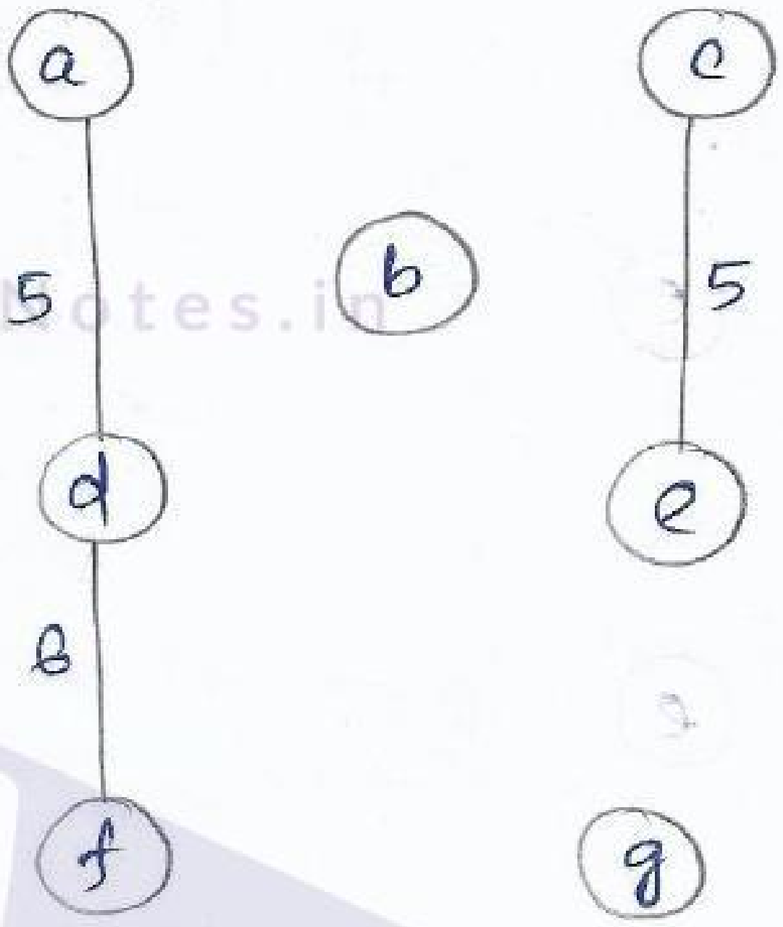
HW-7

Q. 1 Step 3

Edge : (d, f) {a, d, f} {b} {c, e} {g}

$A = A \cup (d, f)$

$A = \{(a, d) (c, e) (d, f)\}$



E Sol.

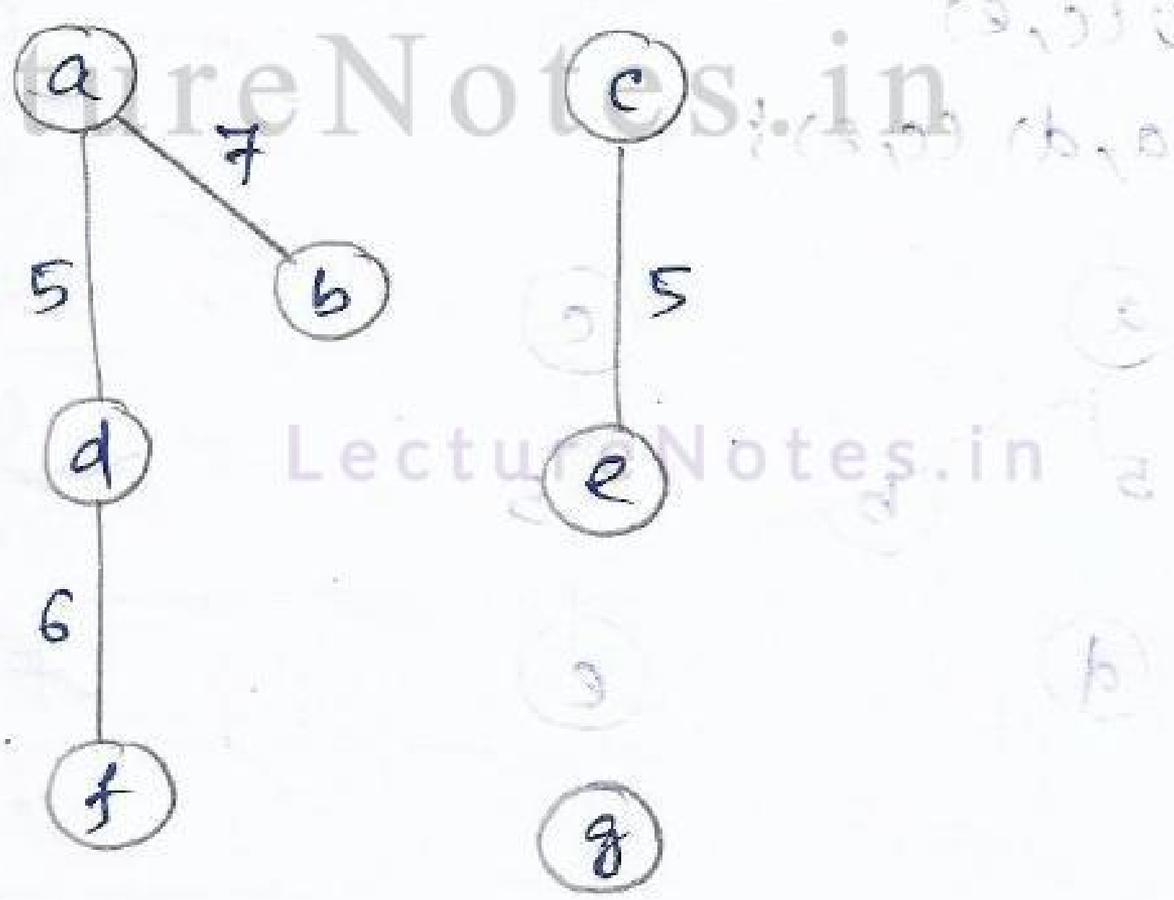
1

Step 4

Edge : (a, b) {a, b, d, f} {c, e} {g}

$A = A \cup (a, b)$

$A = \{(a, d) (c, e) (d, f) (a, b)\}$

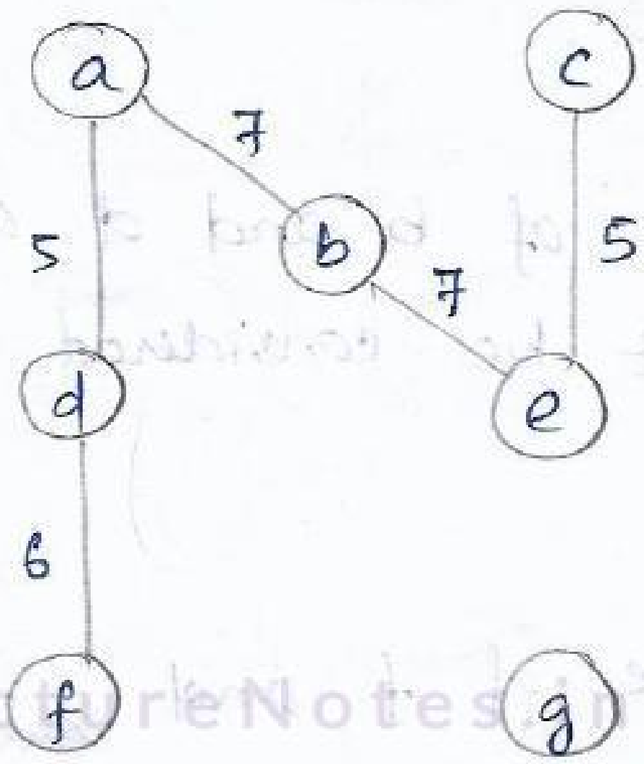


Step 5

Edge : (b, e) {a, b, d, e, f, c} {g}

$A = A \cup (b, e)$

HW-8 $A = \{(a, d) (c, e) (d, f) (a, b) (b, e)\}$



Step 6

Edge : (b, c)

As the representative of b and c are in same set, (b, c) edge should not be considered.

Step 7

Edge : (e, f) {a, b, c, d, e, f} {g}

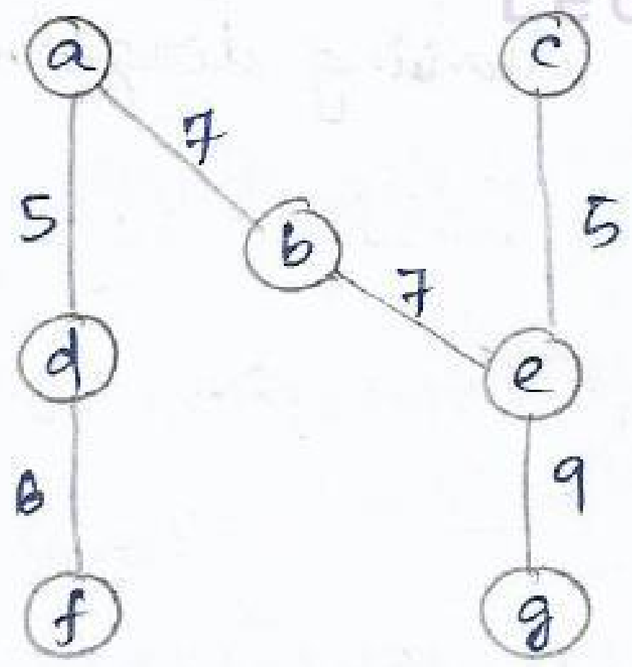
As the representative of e and f are in same set, (e, f) edge should not be considered.

Step 8

Edge : (e, g) {a, b, c, d, e, f, g}

$A = A \cup (e, g)$

$A = \{(a, d) (c, e) (d, f) (a, b) (b, e) (e, g)\}$



HW-9
HW-13

Step 9

Edge : (b, d)

As the representative of b and d are in same set, (b, d) edge should not be considered.

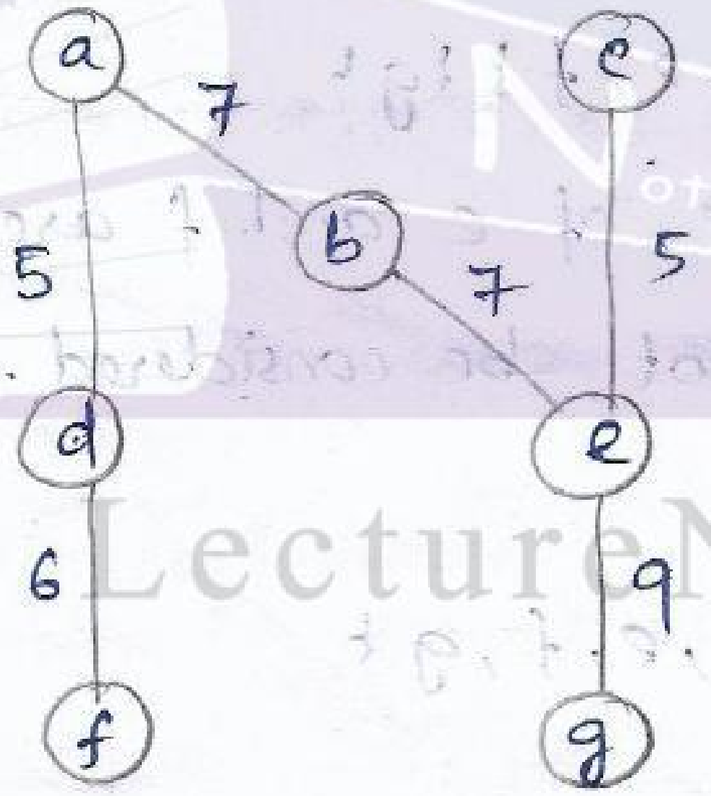
Step 10 Edge : (f, g)

As the representative of f and g are in same set, (f, g) edge should not be considered.

Step 11 Edge : (e, d)

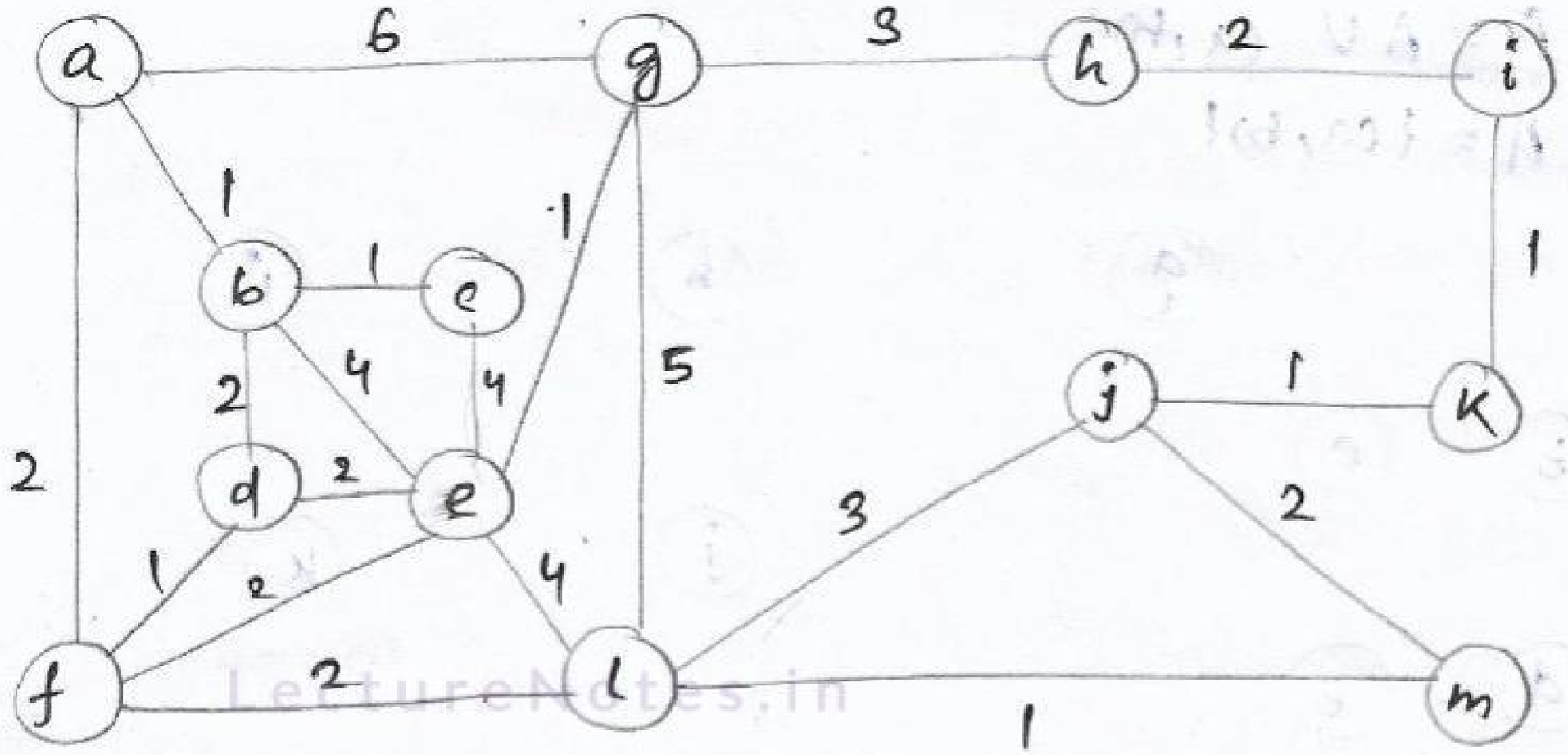
As the representative of e and d are in same set, (e, d) edge should not be considered.

So, the final diagram is.



The minimum weight is 39.

Q. Find the MST of the following diagram using Kruskal's Algorithm.



Sol.

Edges	Weight
(a, b)	1
(b, c)	1
(e, g)	1
(d, f)	1
(l, m)	1
(j, k)	1
(i, k)	1
(a, f)	2
(f, l)	2
(f, e)	2
(d, e)	2

Edges	Weight
(h, i)	2
(j, m)	2
(b, d)	2
(l, j)	3
(g, h)	3
(e, l)	4
(c, e)	4
(b, e)	4
(g, l)	5
(a, g)	6

Initially, applying the Make Set Operation for all vertex.

{a} {b} {c} {d} {e} {f} {g} {h} {i} {j} {k} {l} {m}

Step 1 Edge: (a, b)

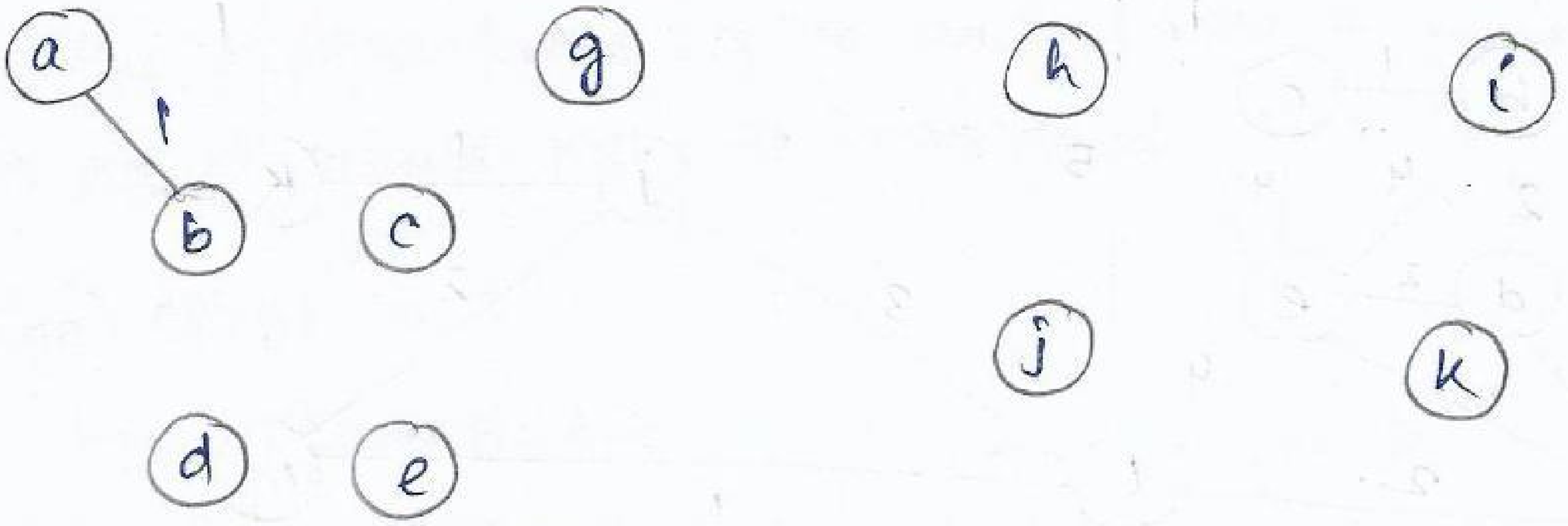
{a, b} {c} {d} {e} {f} {g} {h} {i} {j} {k} {l} {m}

LectureNotes.in

HW-11
HW-13

$A = A \cup (a, b)$

$A = \{(a, b)\}$



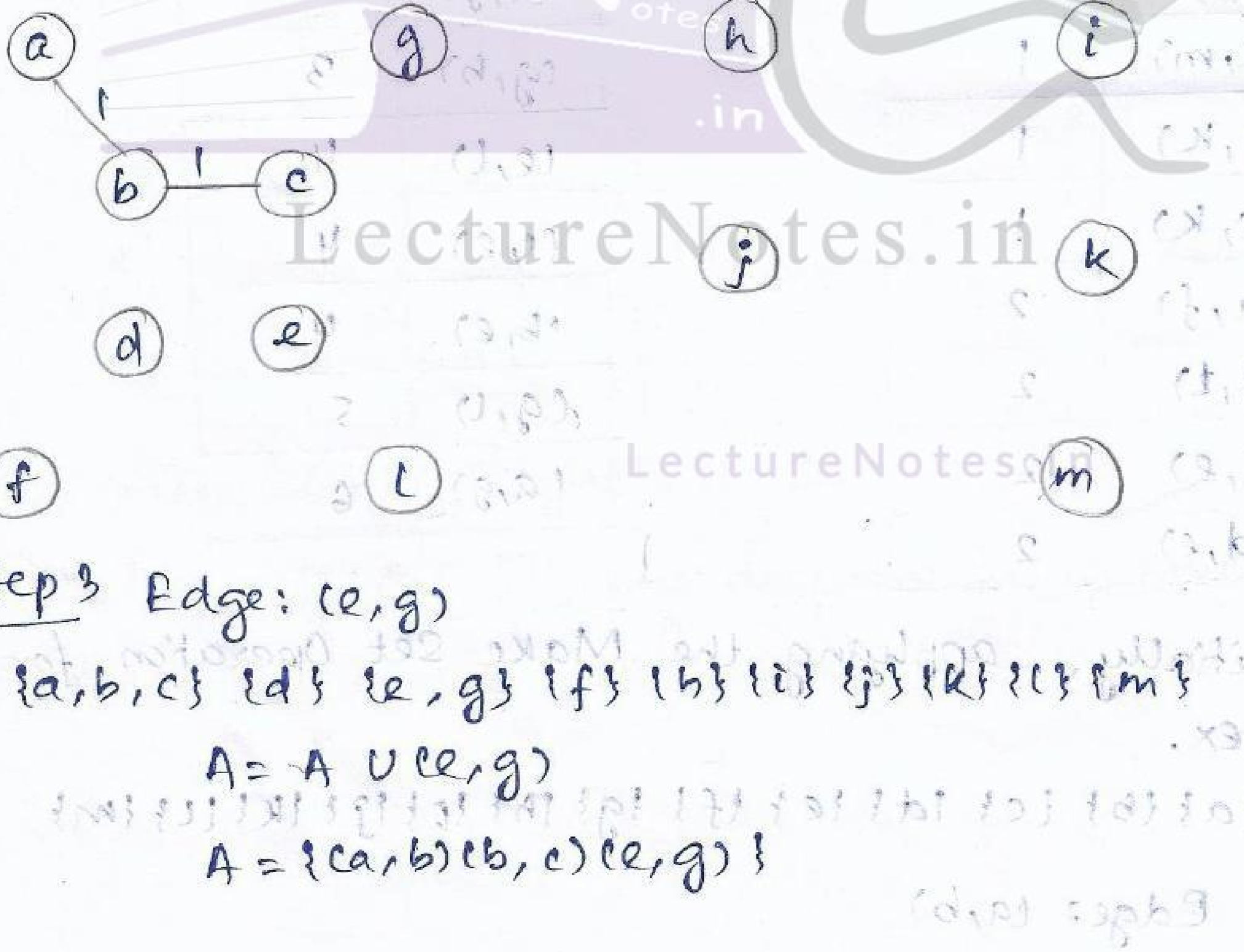
LectureNotes.in

Step 2 Edge : (b, c)

{a, b, c} {d} {e} {f} {g} {h} {i} {j} {k} {l} {m}

$A = A \cup (b, c)$

$A = \{(a, b) (b, c)\}$



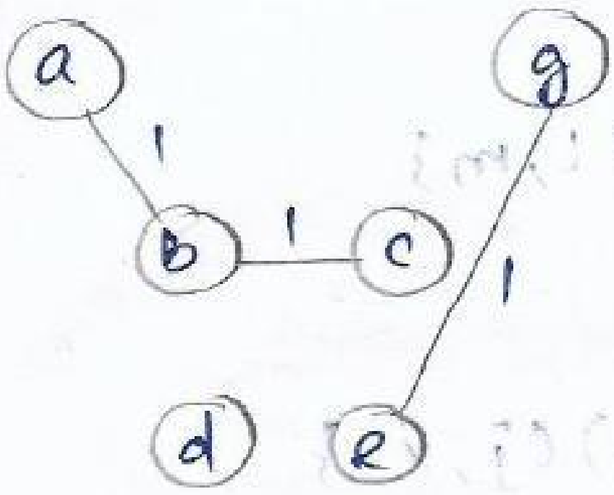
LectureNotes.in

Step 3 Edge : (e, g)

{a, b, c} {d} {e, g} {f} {h} {i} {j} {k} {l} {m}

$A = A \cup (e, g)$

$A = \{(a, b) (b, c) (e, g)\}$

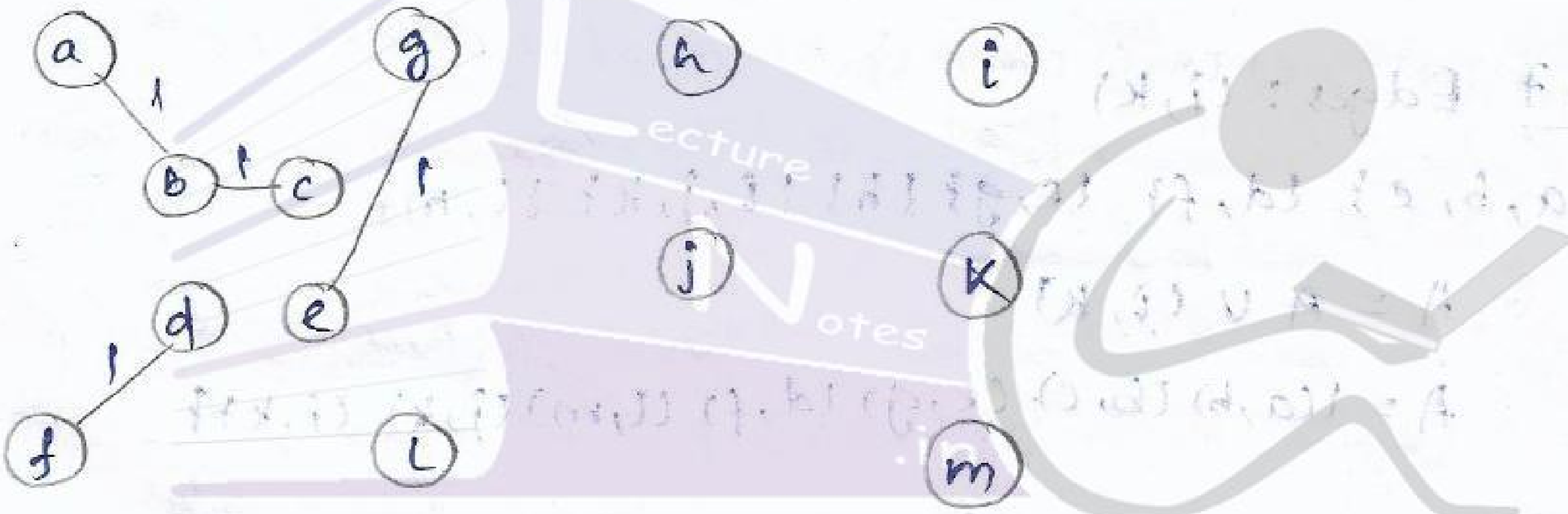


Step 4 Edge : (d, f)

{a, b, c} {d, f} {e, g} {h} {i} {j} {k} {l} {m}

$A = A \cup (d, f)$

$A = \{(a, b)\ (b, c)\ (e, g)\ (d, f)\}$

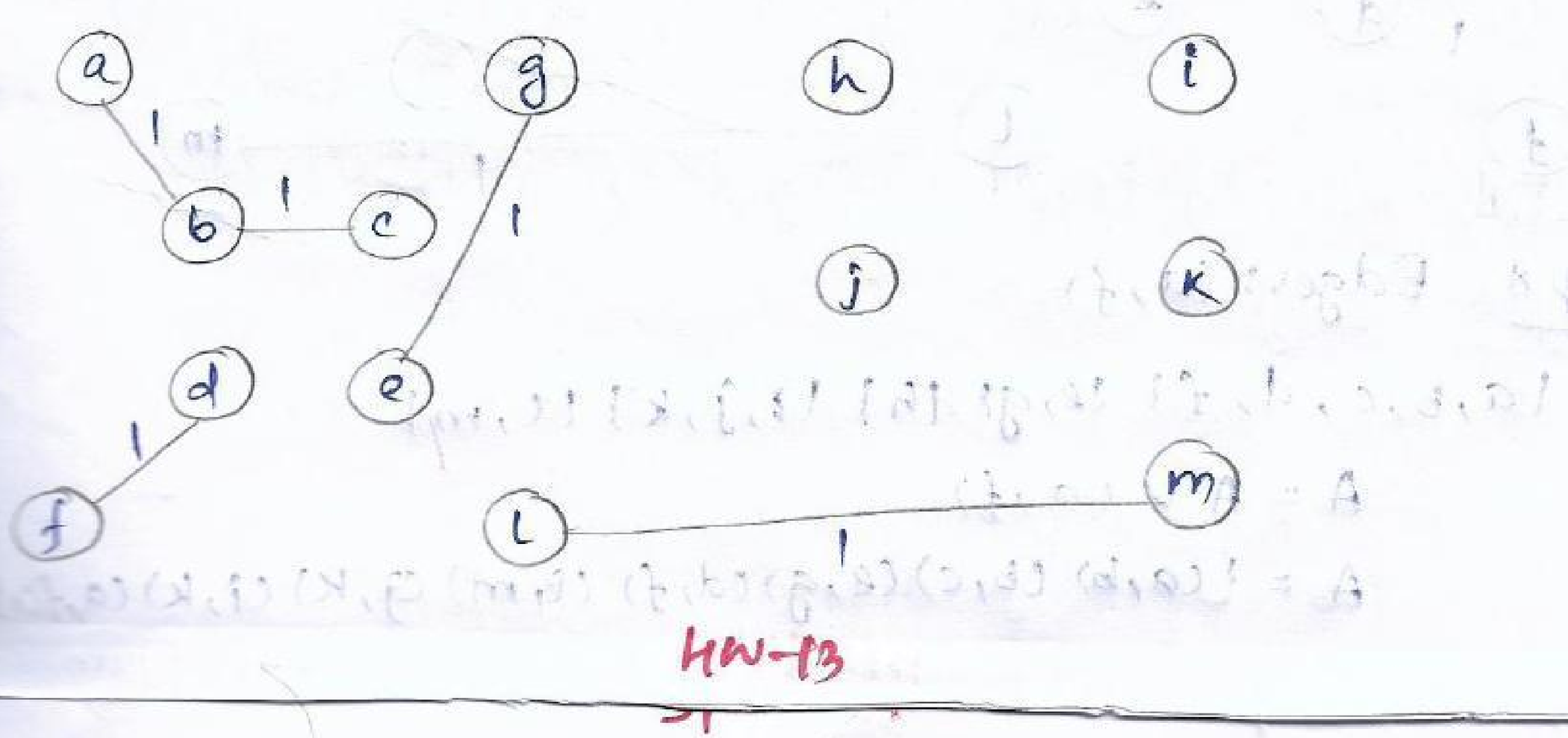


Step 5 Edge : (l, m)

{a, b, c} {d, f} {e, g} {h} {i} {j} {k} {l, m}

$A = A \cup (l, m)$

$A = \{(a, b)\ (b, c)\ (e, g)\ (d, f)\ (l, m)\}$



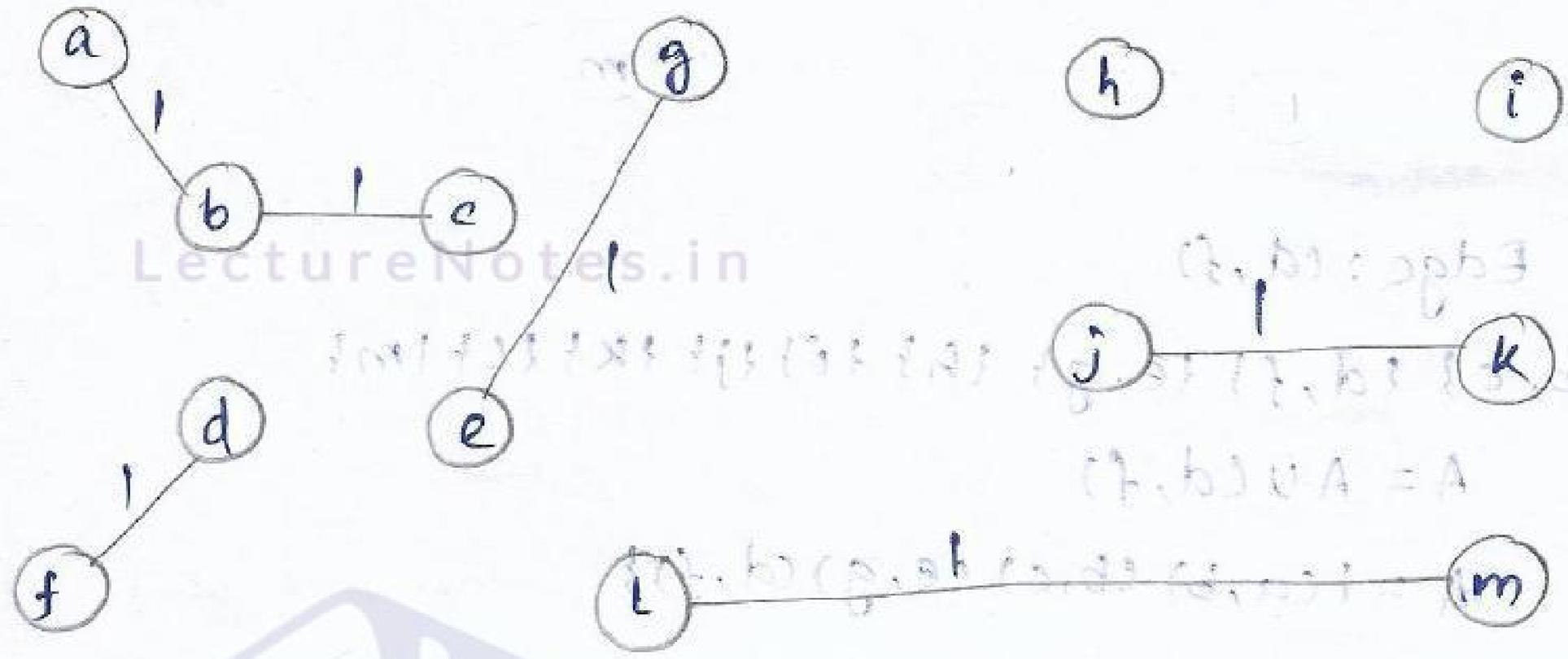
HW-13

Step 6 - Edge - (j, k)

{a, b, c} {d, f} {e, g} {h} {i} {j, k} {l, m}

A = A ∪ (j, k)

A = {(a, b), (b, c), (e, g), (d, f) (l, m) (j, k)}

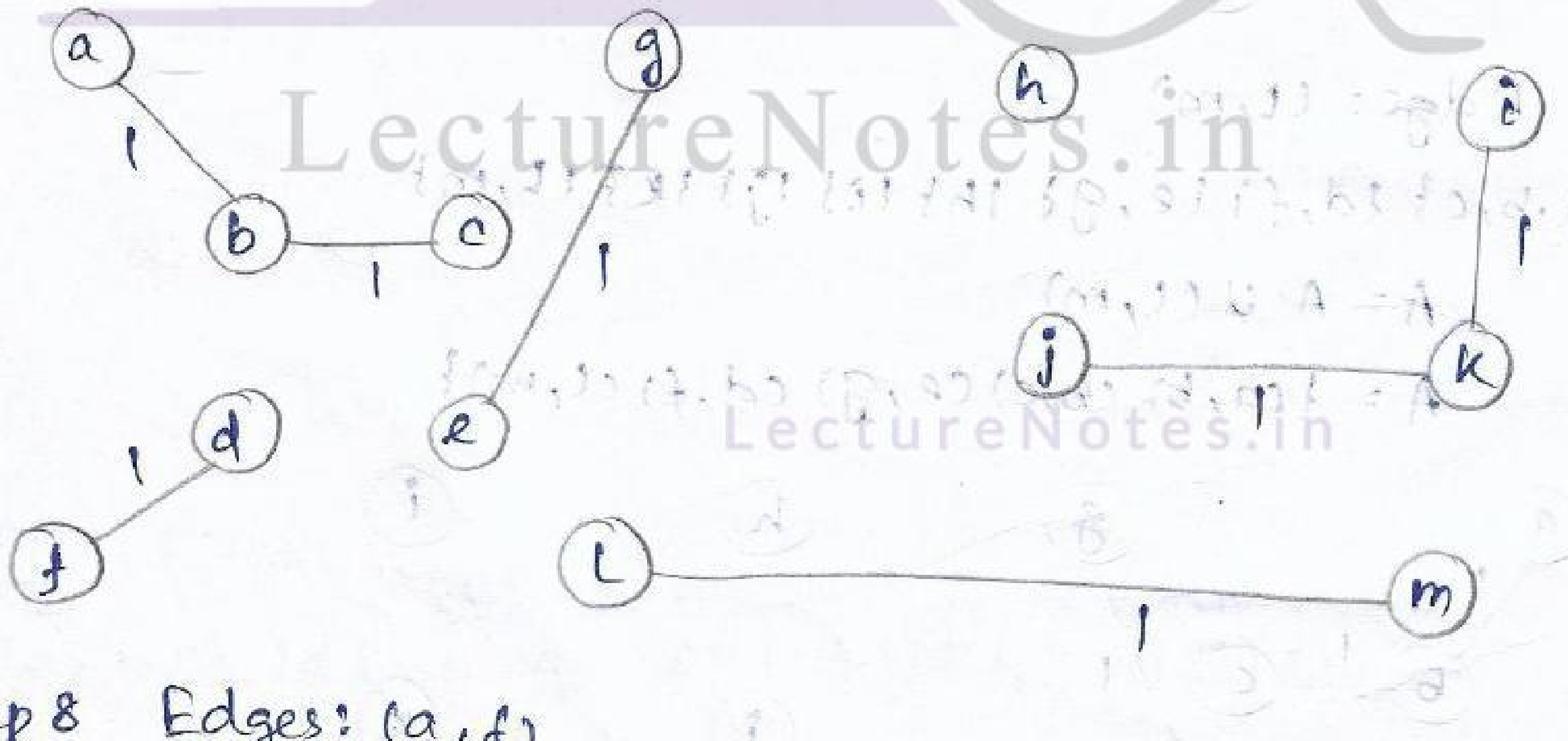


Step 7 Edges : (i, k)

{a, b, c} {d, f} {e, g} {h} {i, j, k} {l, m}

A = A ∪ (i, k)

A = {(a, b) (b, c) (e, g) (d, f) (l, m) (j, k) (i, k)}



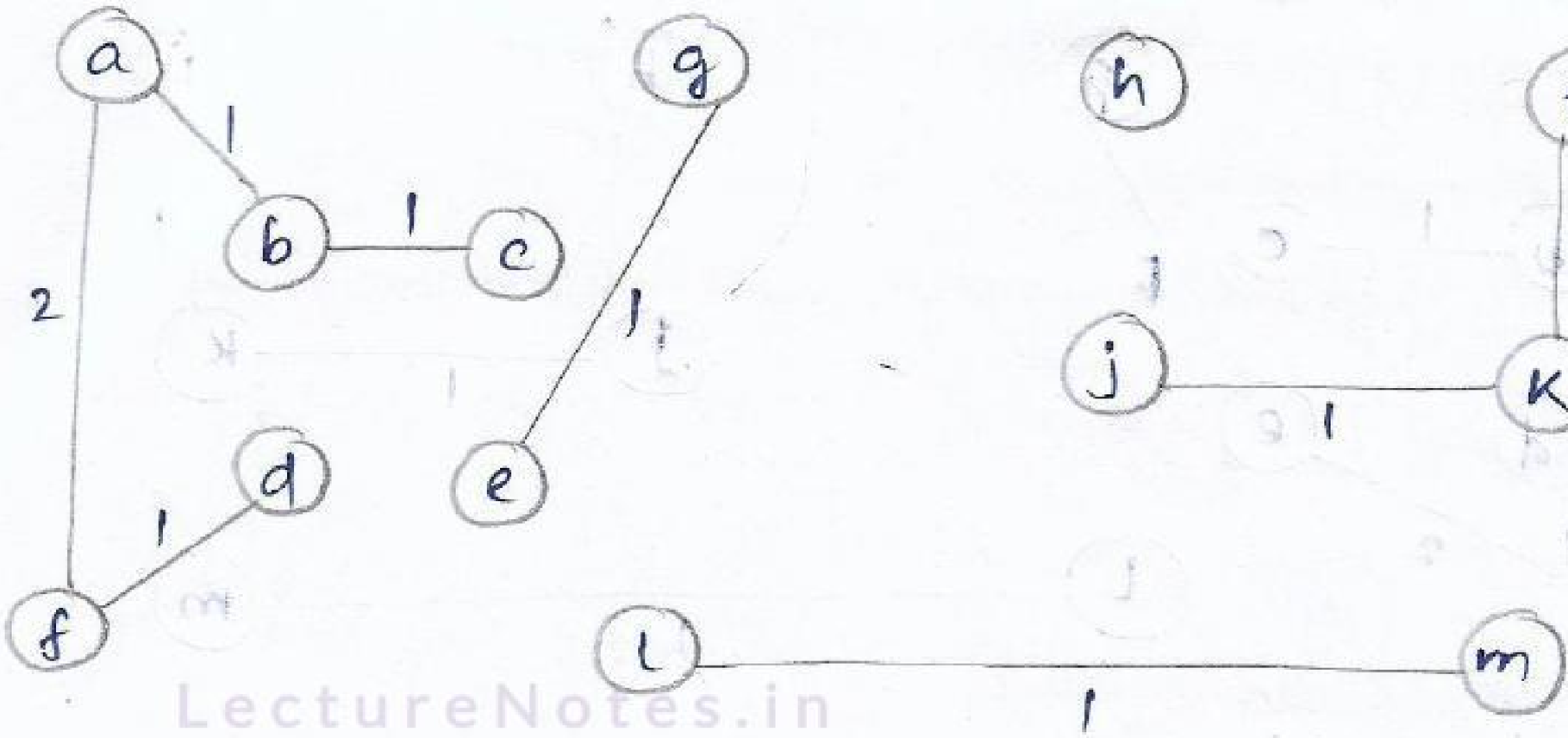
Step 8 Edges: (a, f)

{a, b, c, d, f} {e, g} {h} {i, j, k} {l, m}

A = A ∪ (a, f)

A = {(a, b) (b, c) (e, g) (d, f) (l, m) (j, k) (i, k) (a, f)}

HW-14



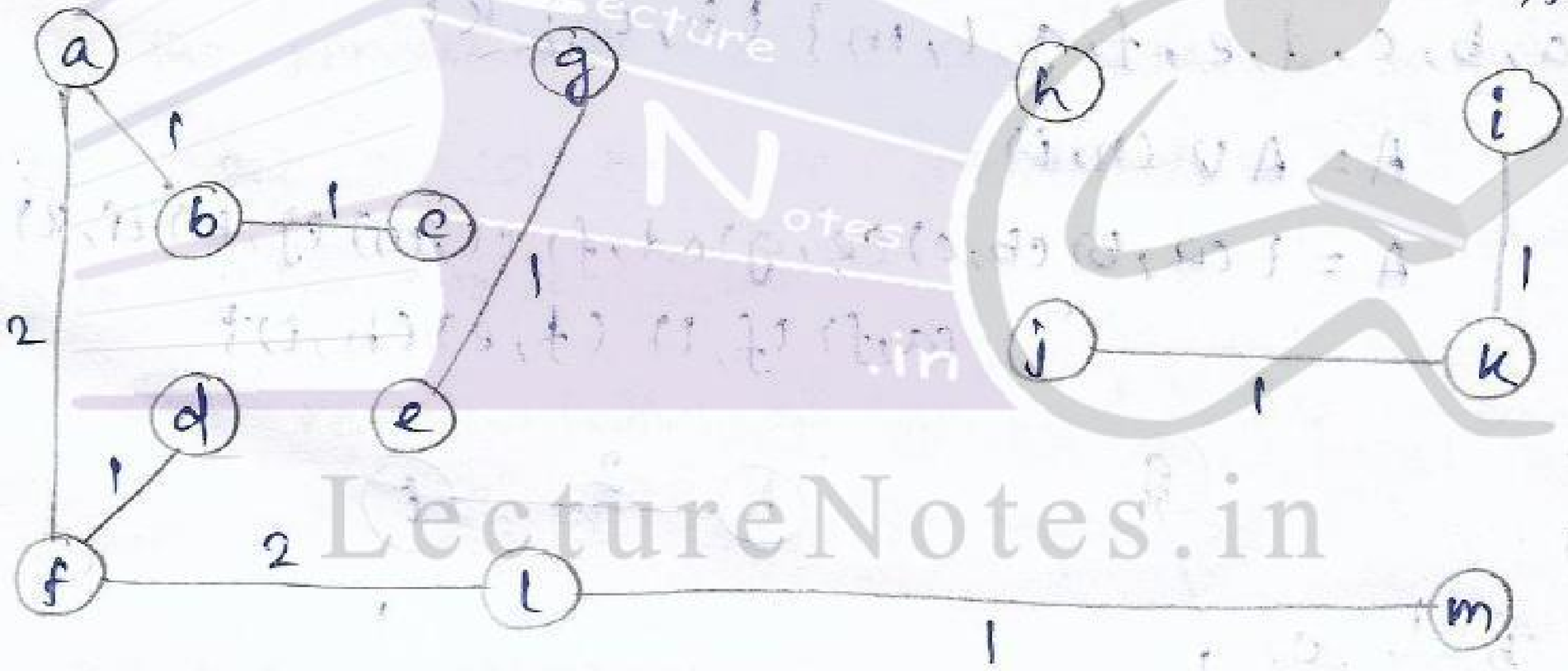
LectureNotes.in

Step 9 Edges: (f, l)

{a, b, c, d, f, l, m} {e, g} {h} {i, j, k}

A = A ∪ (f, l)

A = {(a,b) (b,c) (e,g) (d,f) (l,m) (j,k) (i,k) (a,f) (f,l)}



LectureNotes.in

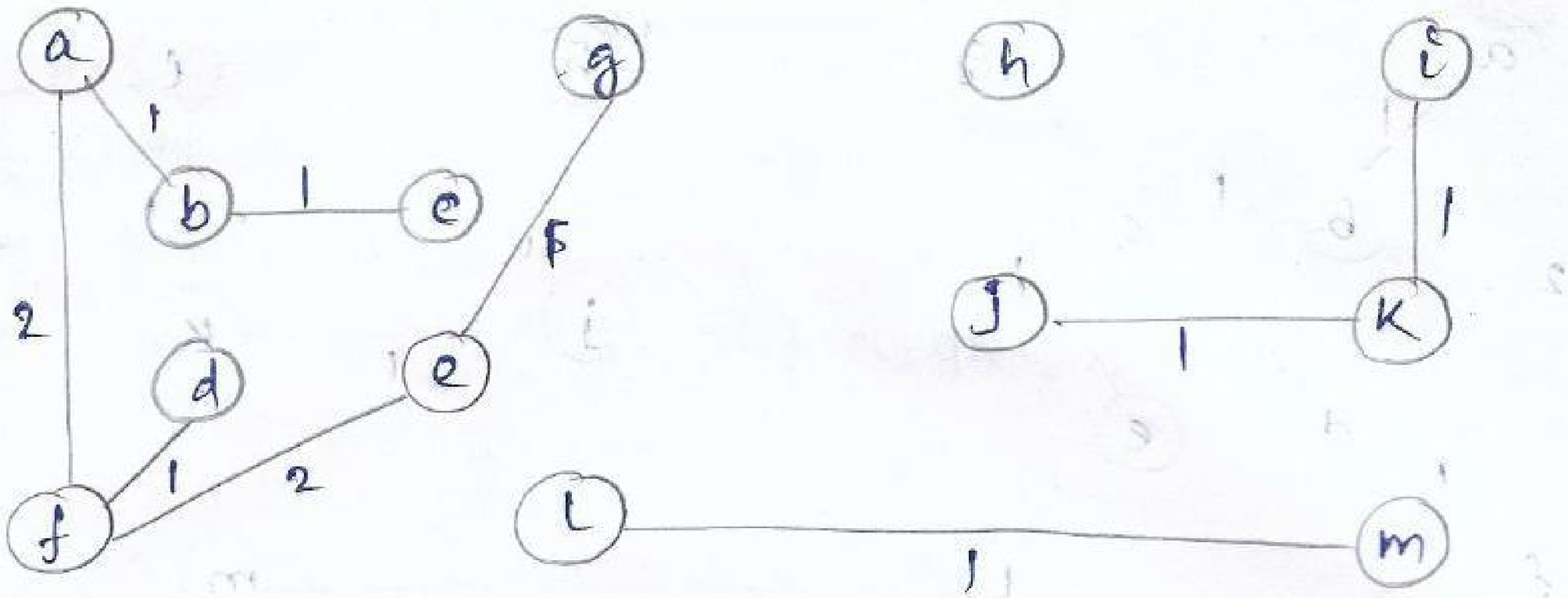
Step 10 Edge: (f, e)

{a, b, c, d, e, f, g, l, m} {h} {i, j, k}

A = A ∪ (f, e)

A = {(a,b) (b,c) (e,g) (d,f) (l,m) (j,k) (i,k) (a,f) (f,l) (f,e)}

Step



Step II Edges : (d, e)

As the representative of d and h are in same set, (d, e) set should not be considered.

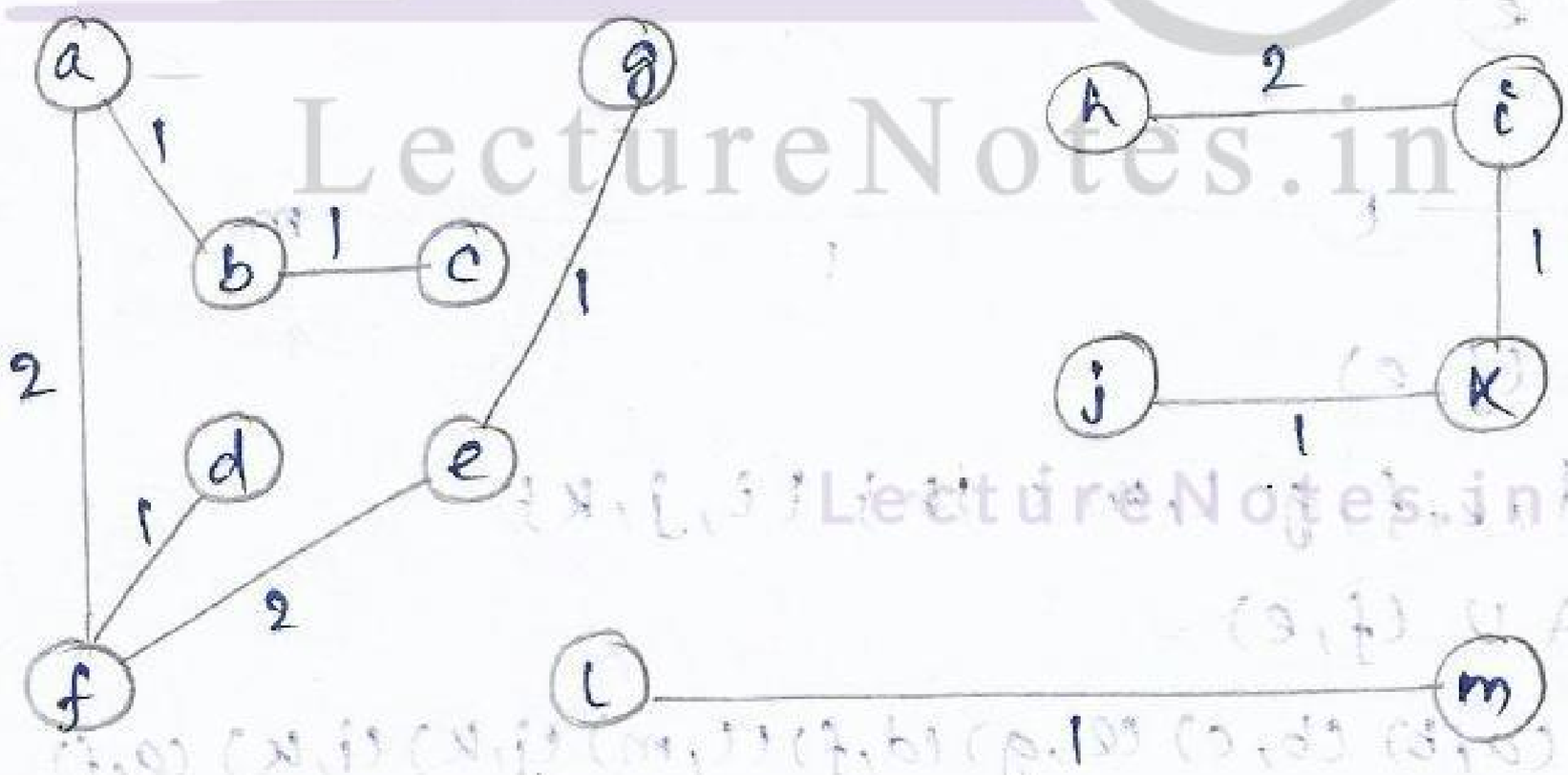
Step 12 Edges : (h, i)

Step

{ {a, b, c, d, e, f, g, l, m} {h, i, j, k}

$A = A \cup (h, i)$

$A = \{(a, b)(b, c)(e, g)(d, f)(l, m)(j, k)(i, k)$
$(a, f)(f, l)(f, e)(h, i)\}$



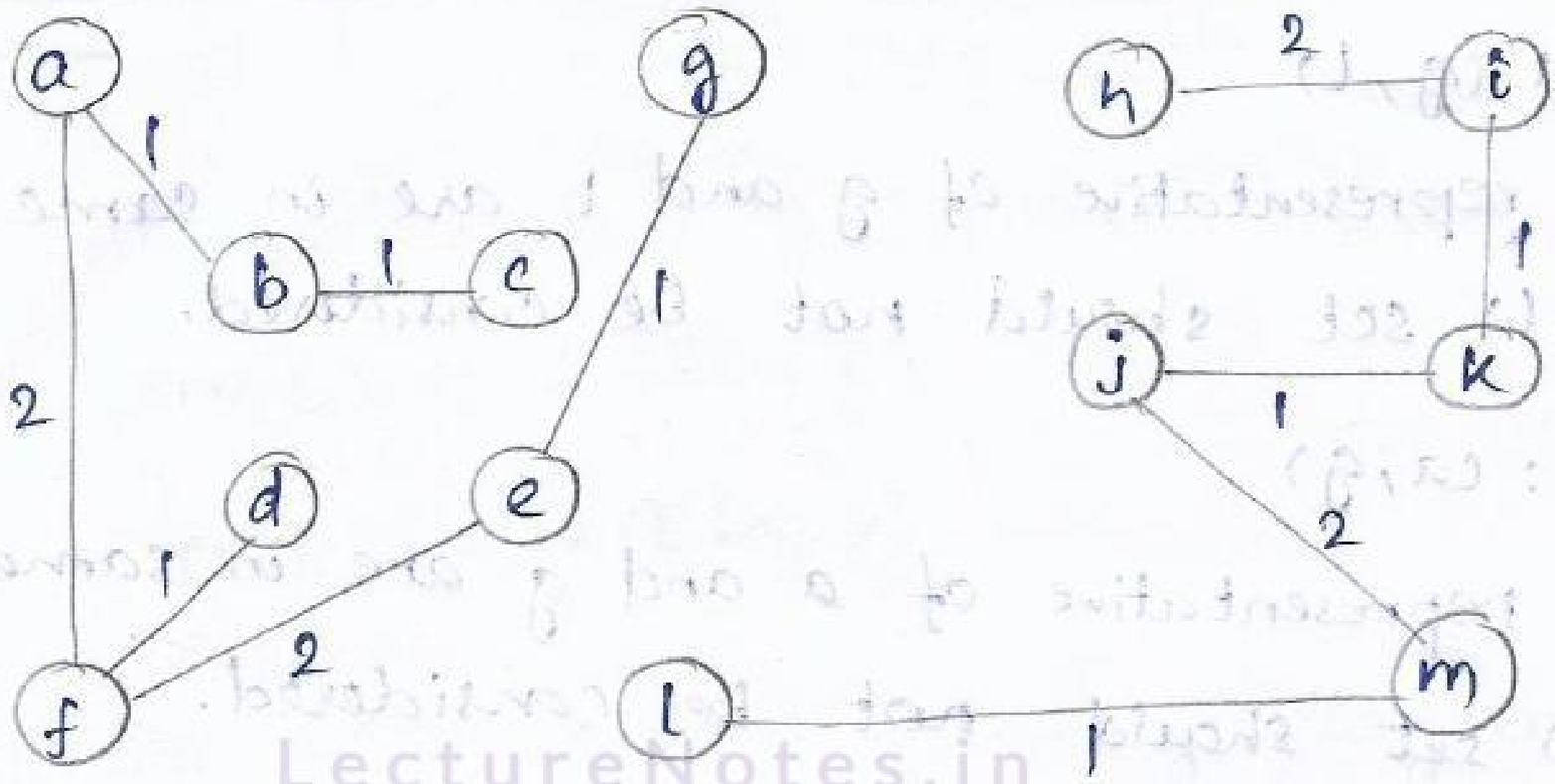
Step Step 13 Edges : (j, m)

{a, b, c, d, e, f, g, h, i, j, k, l, m}

$A = A \cup (j, m)$

$A = \{(a, b)(b, c)(e, g)(d, f)(l, m)(j, k)(i, k)(a, f)(f, l)(f, e)$
$(h, i)(j, m)\}$

HW-16



cal's

Step 14 Edges : (b, d)

As the representative of b and d are in same set, (b, d) set should not be considered.

Step 15 Edges : (l, j)

As the representative of l and j are in same set, (l, j) set should not be considered.

Step 16 Edges : (g, h)

As the representative of g and h are in same set, (g, h) set should not be considered.

Step 17 Edges : (e, l)

As the representative of e and l are in same set, (e, l) set should not be considered.

Step 18 Edges : (c, e)

As the representative of c and e are in same set, (c, e) set should not be considered.

Step 19 Edges: (b, e)

e)

As the representative of b and e are in same set, (b, e) set should not be considered. HW-17

LectureNotes.in

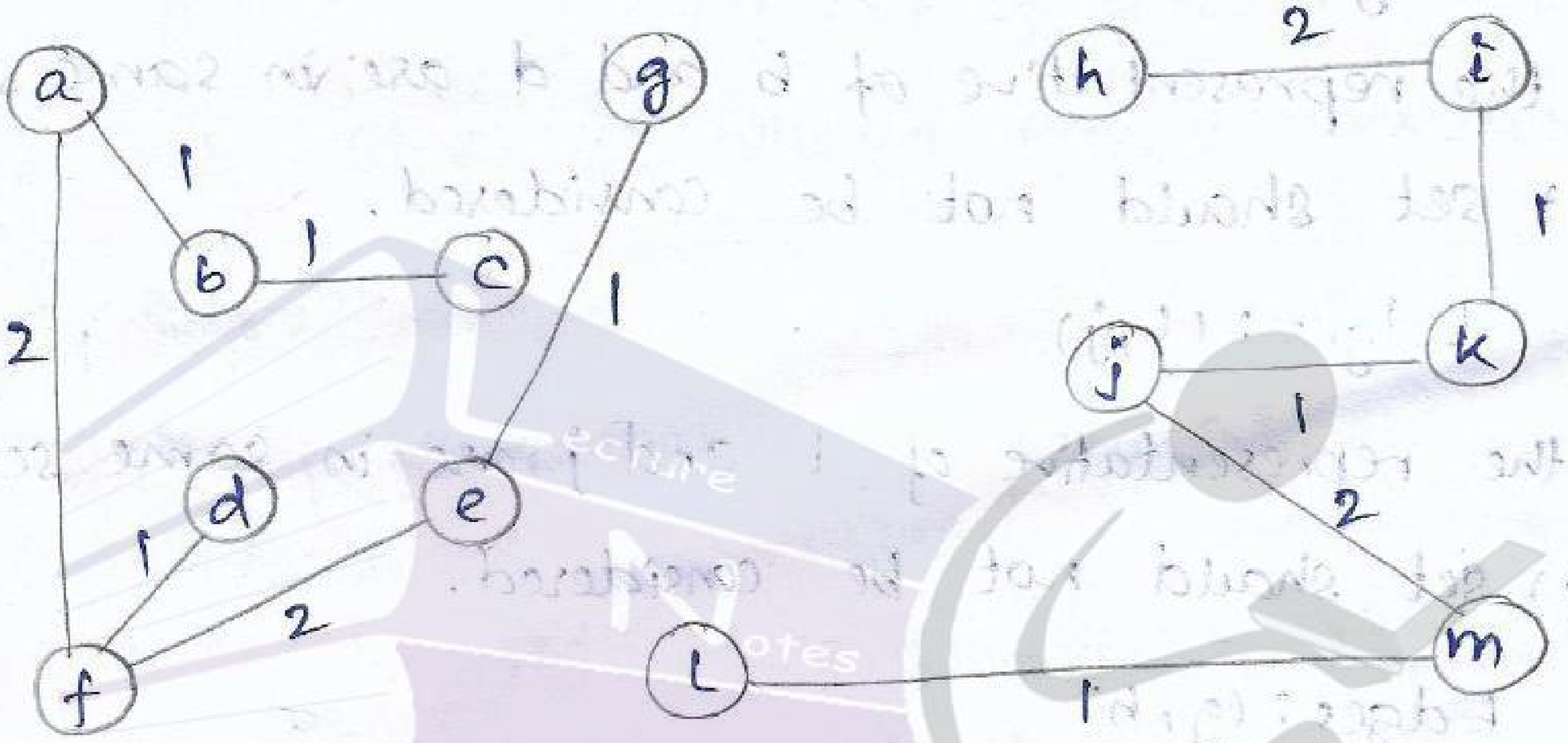
Step 20 Edges : (g, L)

As the representative of g and L are in same set, (g, L) set should not be considered.

Step 21 Edges : (a, g)

As the representative of a and g are in same set, (a, g) set should not be considered.

Thus, the final diagram is



The minimum weight is 15.

LectureNotes.in

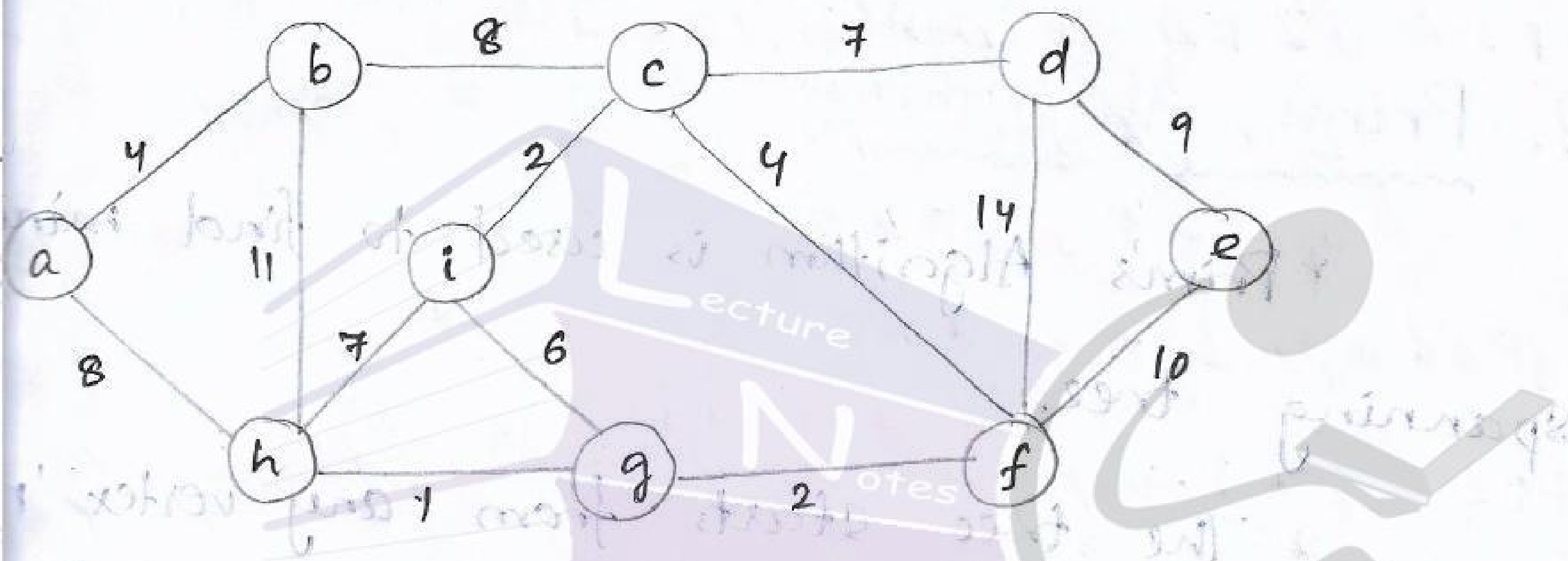
total time complexity is

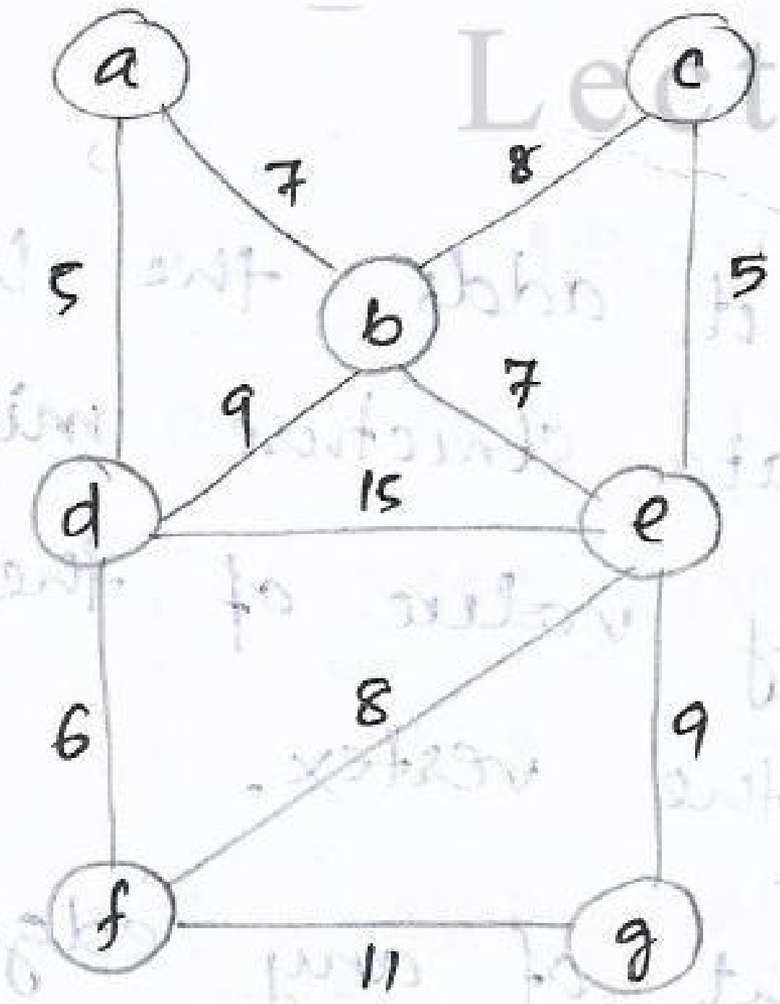
$$T(V, E) = \Theta(E \log E) + \Theta(V) + \Theta(E)$$

$$T(V, E) = \Theta(E \log E)$$

$$T(V, E) = O(E \log V)$$

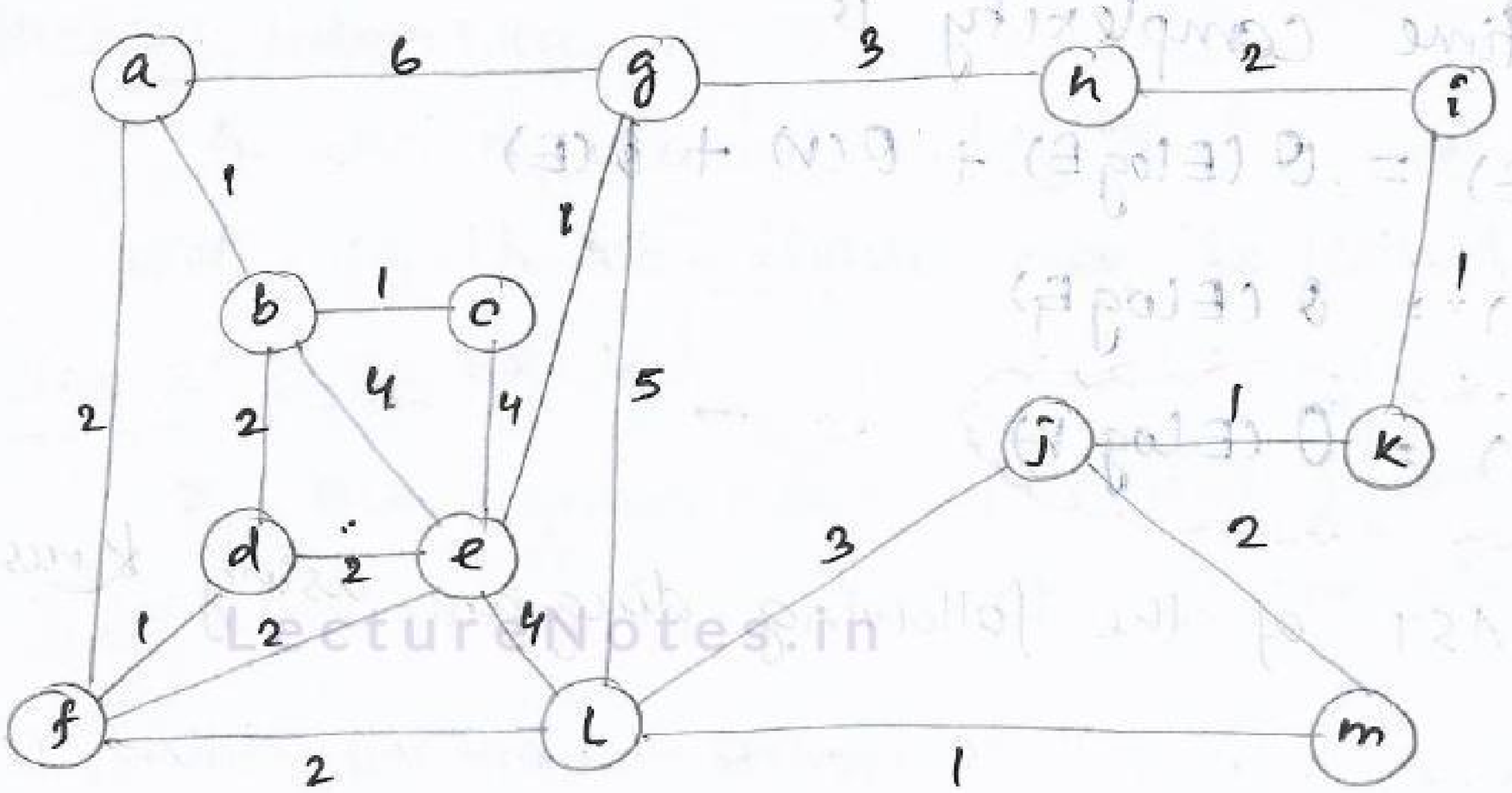
the MST of the following diagram using Kruskal's algorithm.





LectureNotes.in

(3)



2. Prim's Algorithm →

* Prim's Algorithm is used to find minimum Spanning tree.

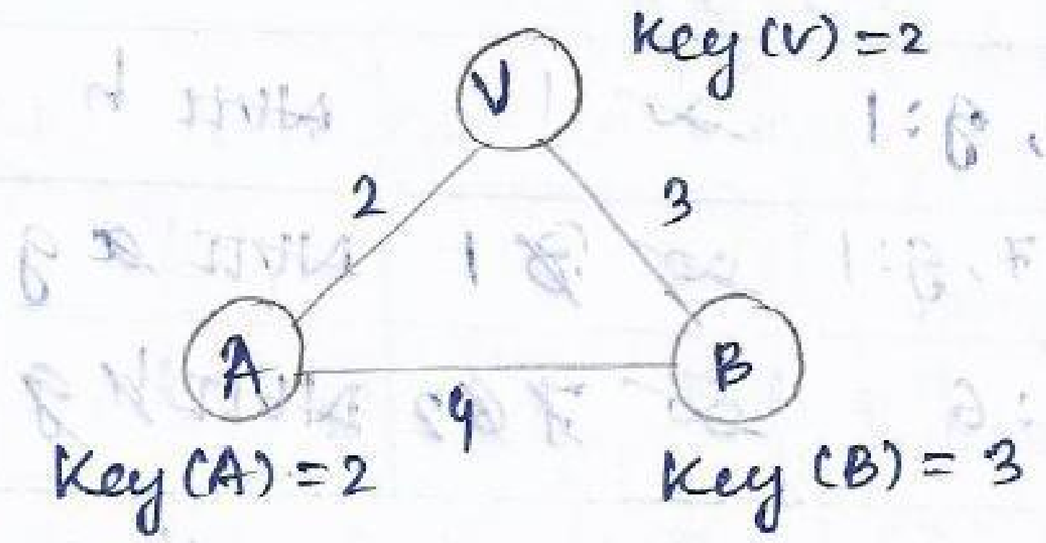
* The tree starts from any vertex 'r' and it grows until the tree's spans all vertex in 'V'.

* At each step, it adds the light edge.

* It uses the data structure min-priority queue. where the priority value of the vertex is the 'key' value of the vertex.

Key(v): Minimum weight of any edge connecting to v.

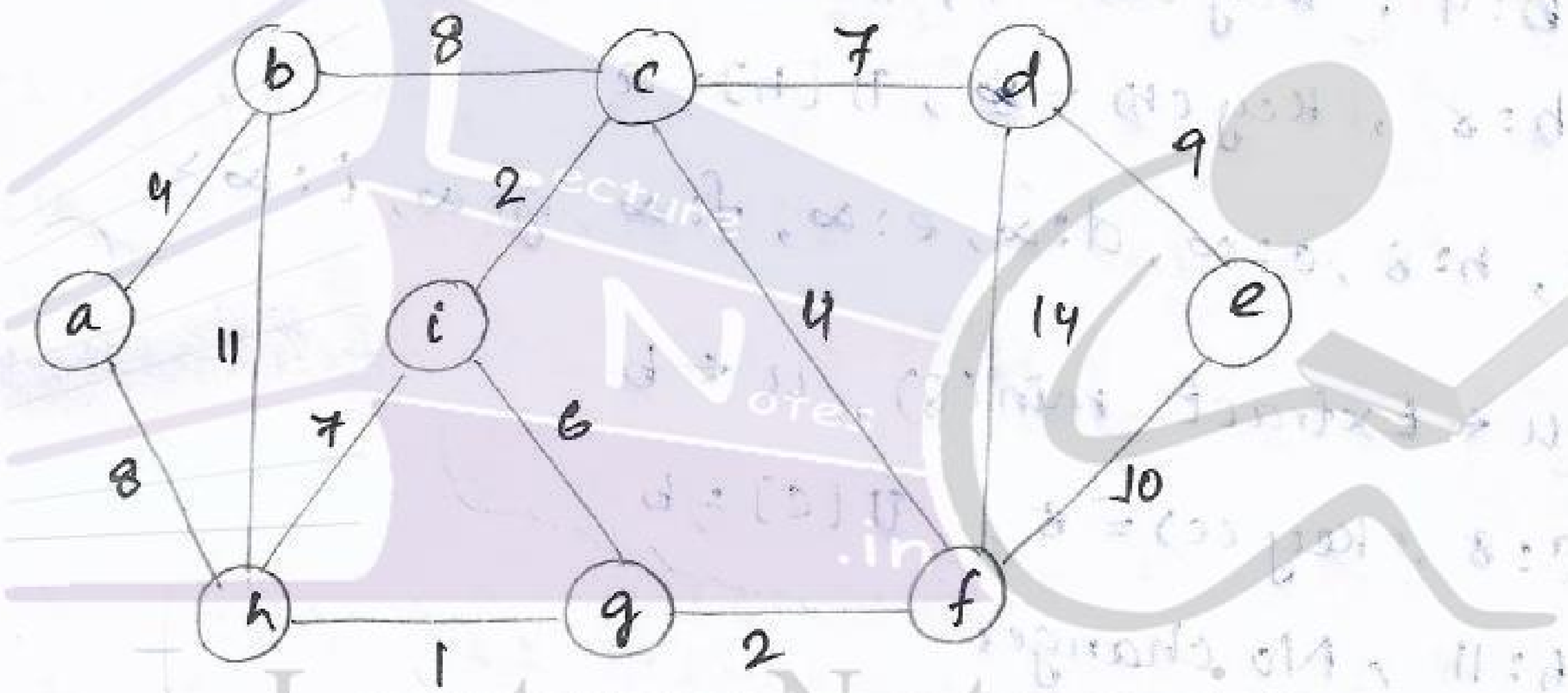
LectureNotes.in



KEY[v] = ∞ : There is no such edge connected to v.

π[v] : Parent of the vertex v.

Q. Find MST of the following graph using Prim's Algorithm. Consider a is the root.



Sol:

Vertex	Adj(v)	Key(v)	π(v)
a	b:4 , h:8	0	NULL
b	c:8 , h:11 , a:4	~~∞~~ 4	~~NULL~~ a
c	b:8 , d:7 , i:2 , f:4	~~∞~~ ~~8~~ 4	~~NULL~~ b f
d	c:7 , f:14 , e:9	~~∞~~ ~~14~~ 7	~~NULL~~ f c
e	d:9 , f:10	~~∞~~ ~~10~~ 9	~~NULL~~ f d
f	d:14 , e:10 , c:4 , g:2	~~∞~~ 2	~~NULL~~ g

LectureNotes.in

g	h:1, i:6, f:2	~~∞~~ 1	~~NULL~~ h
h	a:6, b:11, i:7, g:1	~~∞~~ 8	~~NULL~~ a
i	c:2, h:7, g:6	~~∞~~ 7 6 2	~~NULL~~ h g c

Dt → 31st Mar 18
Saturday

Q: <a:0, b:∞, c:∞, d:∞, e:∞, f:∞, g:∞, h:∞, i:∞>

Step 1 → u ← Extract_min(Q), u ← a

b:4, Key(b) = 4, π[b] = a

h:8, Key(h) = 8, π[h] = a

Q: <b:4, h:8, c:∞, d:∞, e:∞, f:∞, g:∞, i:∞>

Step 2 → u ← Extract_min(Q), u ← b

c:8, Key(c) = 8, π[c] = b

h:11, No changes

a:4, a is not in queue

Q: <h:8, c:6, d:∞, e:∞, f:∞, g:∞, i:∞>

Step 3 → u ← Extract_min(Q), u ← h

i:7, Key(i) = 7, π[i] = h

g:1, Key(g) = 1, π[g] = h

b:11, 'b is not in queue

a:8, a is not in queue

Q: <g:1, i:7, c:8, d:∞, e:∞, f:∞>

Step 4 → u ← Extract_min(Q), u ← g

h:1, h is not in queue

i:6, key(i) = 6, π[i] = g

f:2, key(f) = 2, π[f] = g

Q: <f:2, i:6, c:8, d:∞, e:∞>

Step 5 → u ← Extract_min(Q), u ← f

e:10, key(e) = 10, π[e] = f

d:14, key(d) = 14, π[d] = f

c:4, key(c) = 4, π[c] = f

g:2, g is not in queue.

Q: <c:4, i:6, e:10, d:14>

Step 6 → u ← Extract_min(Q), u ← c

b:8, b in not in queue.

i:2, key(i) = 2, π[i] = c

f:4, f is not in queue.

d:7, key(d) = 7, π[d] = c

Q: <i:2, d:7, e:10>

Step 7 → u ← Extract_min(Q), u ← i

c:2, c is not in queue

h:7, h is not in queue

g:6, g is not in queue

Q: <d:7, e:10>

LectureNotes.in

Step 8 → u ← Extract_min(Q), u ← d

c:7, c is not in queue

f:14, f is not in queue

e:9, Key(e) = 9, π(e) = d

Q: <e:9>

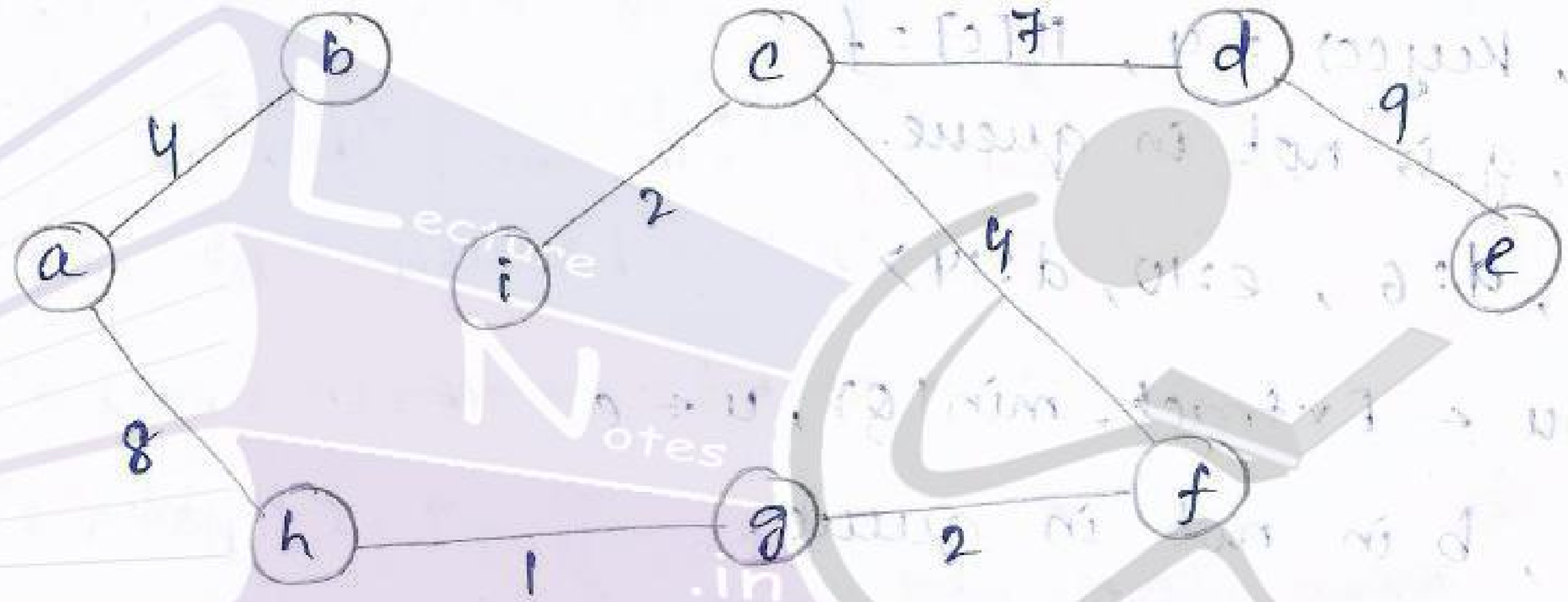
Step 9 → u ← Extract_min(Q), u ← e

Q: d:9, d is not in queue

Step f:10. f is not in queue

Q:

Step



The minimum weight is 37.

Q: Algorithm (Prim's Algorithm)

Step

MST_PRIM (G, w, r)

/* G = (V, E)

w ←

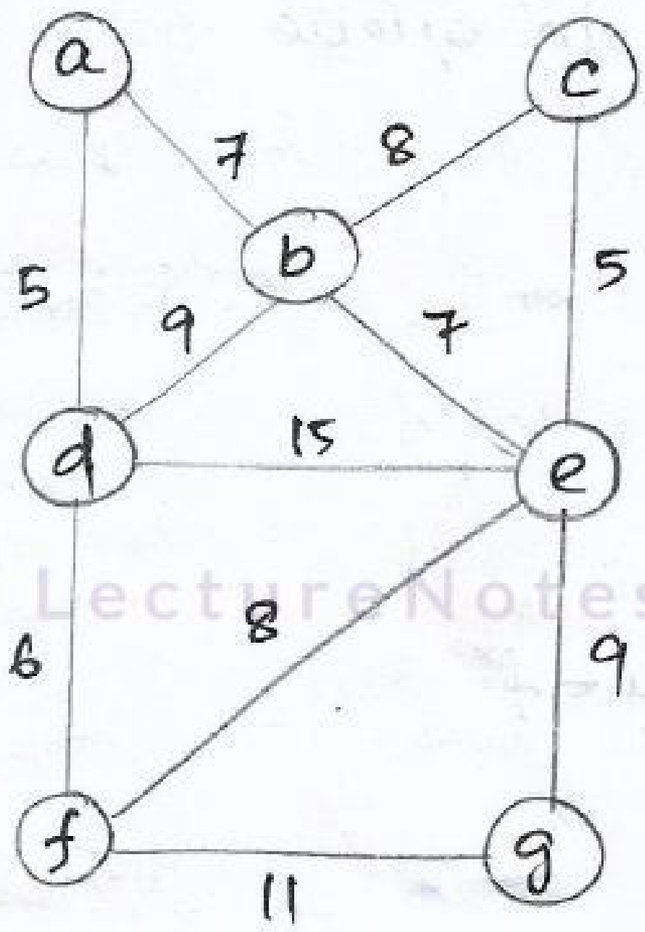
r ← Source */

Q:

1. FOR EACH VERTEX u ∈ V[G] ← Q(v)
2. KEY[u] ← ∞
3. π[u] ← NULL

[END FOR]

3. Find MST of the following graph using Prim's Algorithm



Sol.

Vertex	Adj(v)	key(v)	π(v)
a	d:5, b:7	0	NULL
b	a:7, c:8, d:9, e:7	~~∞~~ 7	~~NULL~~ a
c	b:8, e:5	~~∞~~ 8 5	~~NULL~~ b e
d	a:5, b:9, e:15, f:6	~~∞~~ 5	~~NULL~~ a
e	c:5, b:7, d:15, f:8, g:9	~~∞~~ 15 8 7	~~NULL~~ ~~X~~ ~~X~~ b
f	d:6, e:8, g:11	~~∞~~ 6	~~NULL~~ d
g	f:11, e:9	~~∞~~ ~~X~~ 9	~~NULL~~ ~~X~~ e

Q: < a:0, b:∞, c:∞, d:∞, e:∞, f:∞, g:∞ >

Step 1 - u ← Extract_min(Q), u ← a

d:5, Key(d) = 5, π[d] = a

b:7, Key(b) = 7, π[b] = a

Q: < d:5, b:7, c:∞, e:∞, f:∞, g:∞ >

Step 8

Step 2 - u ← Extract_min(Q), u ← d

a:5, No change as a is not in queue

b:9, No change

e:15, Key(e) = 15, π[e] ← d

f:6, Key(f) = 6, π[f] ← d

Step c Q: < f:6, b:7, e:15, c:∞, g:∞ >

Q

Step 3 - u ← Extract_min(Q), u ← f

Step

d:6, d is not in queue

e:8, Key(e) = 8, π[e] ← f

g:11, Key(g) = 11, π[g] ← f

Q: Q: < b:7, e:8, g:11, c:∞ >

(

Step 4 - u ← Extract_min(Q), u ← b

Step

a:7, a is not in queue

c:8, Key(c) = 8, π[c] ← b

d:9, d is not in queue

e:7, Key(e) = 7, π[e] ← b

Q: Ans Q: < e:7, c:8, g:11 >

Step 5 - u ← Extract_min(Q), u ← e

Step

M

c:5, Key(c) = 5, π[c] ← e

/ b:7, b is not in queue

d:15, d is not in queue

f:8, f is not in queue

1. g:9, Key(g) = 9, π[g] ← e

Q:

2. Q: < c:5, g:9 >

3.

HW-20

Step 6 - $u \leftarrow$ Extract_min(Q), $u \leftarrow c$

b:8, b is not in queue

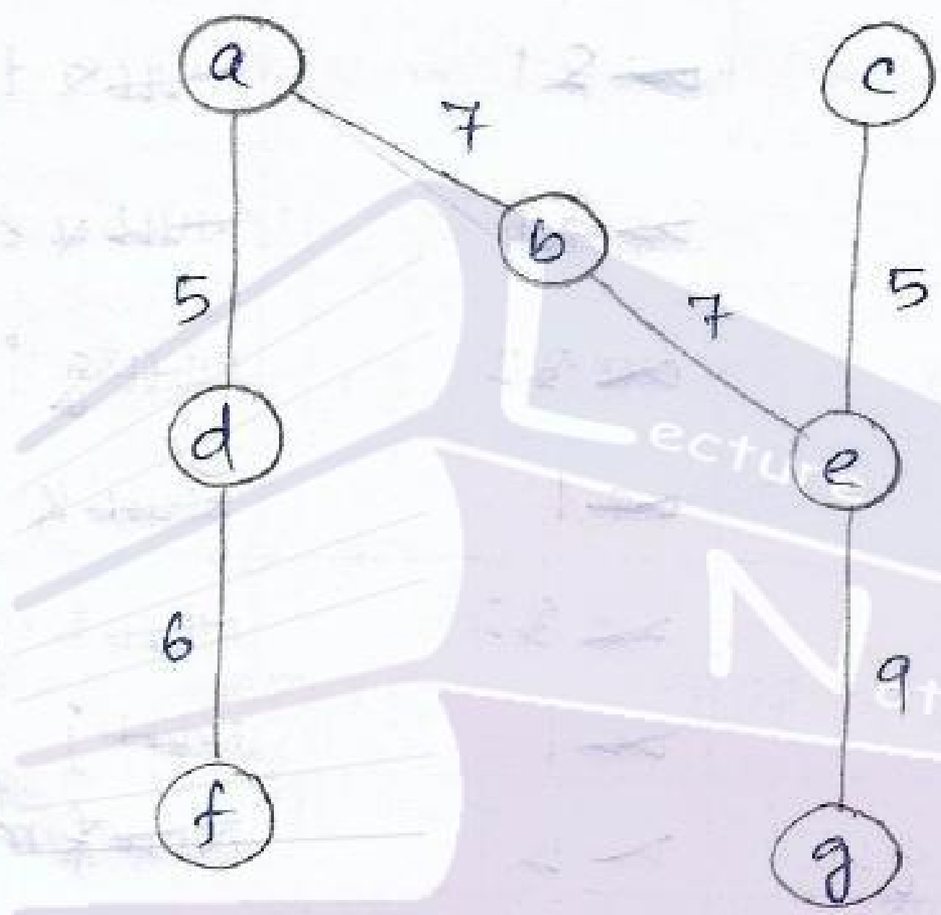
e:5, e is not in queue

Q: <g:9>

Step 7 - $u \leftarrow$ Extract_min(Q), $u \leftarrow g$

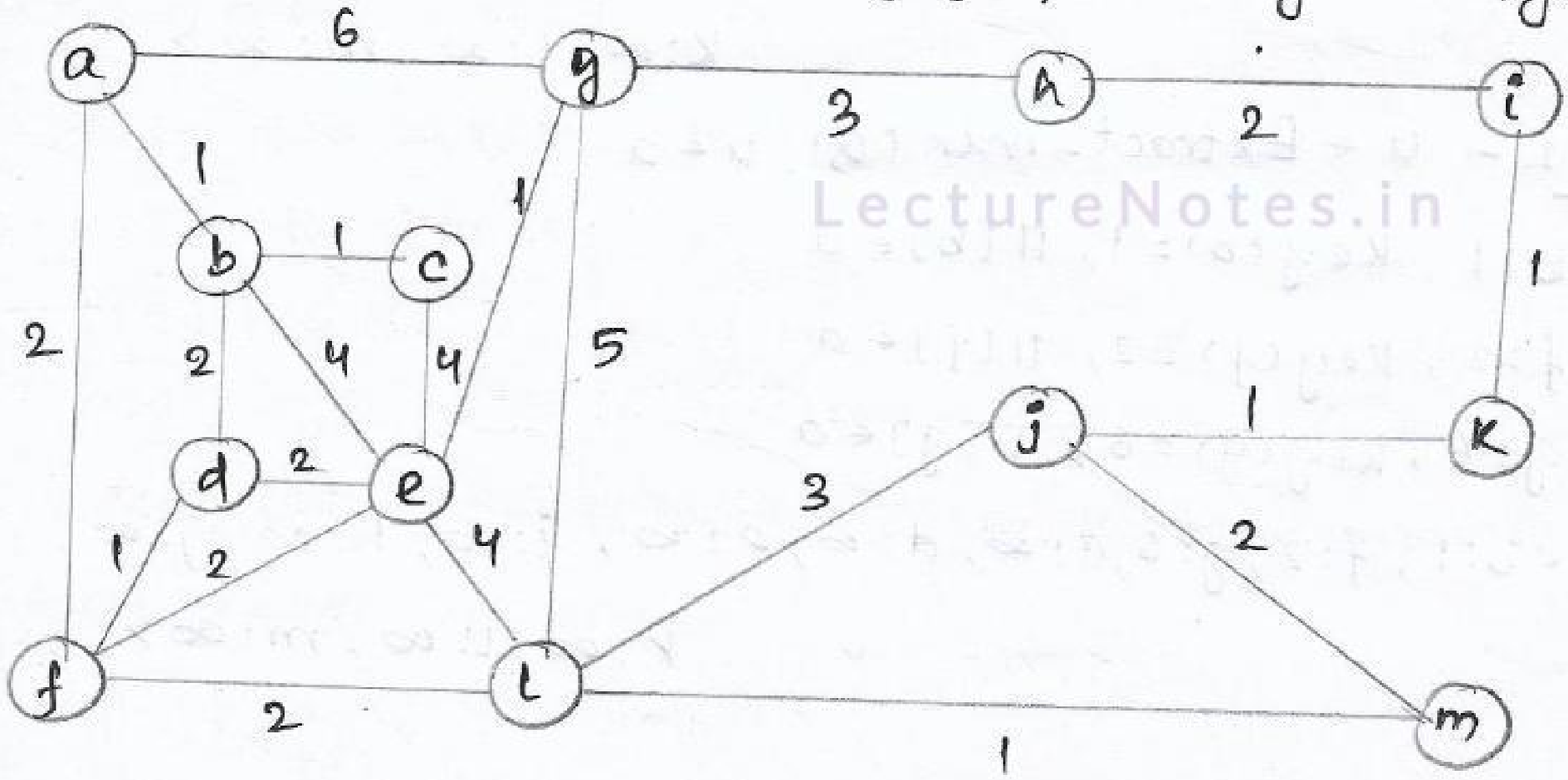
f:11, f is not in queue

e:9, e is not in queue



The minimum weight is 39

Q. Find MST of the following graph using Prim's Algorithm ne



HW-21

Step 9

Vertex	Adj(v)	Key(v)	π(v)
a	b:1, f:2, g:6	0	NULL
b	a:1, c:1, d:2, e:4	~~∞~~ 1	~~NULL~~ a
c	b:1, e:4	~~∞~~ 1	~~NULL~~ b
d	f:1, b:2, e:2	~~∞~~ 2	~~NULL~~ b
e	g:1, d:2, f:2, b:4, c:4, l:4	~~∞~~ ~~4~~ 2	~~NULL~~ bd
f	d:1, e:2, l:2, a:2	~~∞~~ ~~2~~ 1	~~NULL~~ ~~d~~ d
g	e:1, l:5, h:3, a:6	~~∞~~ ~~6~~ 1	~~NULL~~ ~~e~~ e
h	g:3, i:2	~~∞~~ ~~3~~ 2	~~NULL~~ ~~g~~ i
i	h:2, k:1	~~∞~~ 1	~~NULL~~ k
j	k:1, m:2, l:3	~~∞~~ ~~3~~ 2	~~NULL~~ ~~l~~ m
k	i:1, j:1	~~∞~~ 1	~~NULL~~ j
l	m:1, f:2, j:3, e:4, g:5	~~∞~~ ~~2~~ 1	~~NULL~~ ~~f~~ m
m	l:1, j:2	~~∞~~ 1	~~NULL~~ l

Step c

Q: <a:0, b:∞, c:∞, d:∞, e:∞, f:∞, g:∞, h:∞, i:∞, j:∞, k:∞, l:∞, m:∞>

Step 1 – u ← Extract-min(Q), u ← a

b:1, key(b) = 1, π[b] ← a

f:2, key(f) = 2, π[f] ← a

g:6, key(g) = 6, π[g] ← a

Q: <b:1, f:2, g:6, c:∞, d:∞, e:∞, i:∞, h:∞, j:∞, k:∞, l:∞, m:∞>

1.

2.

3.

HW-22

Step 2 $u \leftarrow$ Extract_min(Q), $u \leftarrow b$

a:1, a is not in queue

c:1, key(c) = 1, $\pi[c] \leftarrow b$

d:2, key(d) = 2, $\pi[d] \leftarrow b$

e:4, key(e) = 4, $\pi[e] \leftarrow b$

Q: < c:1, d:2, f:2, e:4, g:6, h:∞, i:∞, j:∞, k:∞, l:∞, m:∞ >

Step 3 $u \leftarrow c$

b:1, b is not in queue

e:4, No change

Q: < d:2, f:2, e:4, g:6, h:∞, i:∞, j:∞, k:∞, l:∞, m:∞ >

Step 4 $u \leftarrow d$

f:1, key(f) = 1, $\pi[f] \leftarrow d$

b:2, b is not in queue

e:2, key(e) = 2, $\pi[e] \leftarrow d$

Q: < f:1, e:2, g:6, h:∞, i:∞, j:∞, k:∞, l:∞, m:∞ >

Step 5 $u \leftarrow f$

d:1, d is not in queue

e:2, No change

l:2, key(l) = 2, $\pi[l] \leftarrow f$

a:2, a is not in queue

Q: < e:2, l:2, g:6, h:∞, i:∞, j:∞, k:∞, m:∞ >

Step 6 $u \leftarrow e$

g:1, key(g) = 1, $\pi[g] \leftarrow e$

d:2, d is not in queue

f:2, f is not in queue

HW-23

LectureNotes.in

Sfc4 b:4, b is not in queue

c:4, c is not in queue

l:4, No change

Q: < g:1, l:2, h:∞, i:∞, j:∞, k:∞, m:∞ >

Step 7 u ← g

Sfe e:1, e is not in queue

l:5, No change

h:3, key(h) = 3, π[h] ← g

a:6, a is not in queue

Q: < l:2, h:3, i:∞, j:∞, k:∞, m:∞ >

Step 8 u ← l

m:1, key(m) = 1, π[m] ← l

f:2, f is not in queue

j:3, key(j) = 3, π[j] ← l

e:4, e is not in queue

g:5, g is not in queue

Q: < m:1, j:3, h:3, i:∞, k:∞ >

Step 9 u ← m

l:1, l is not in queue

j:2, key(j) = 2, π[j] ← m

Q: < j:2, h:3, i:∞, k:∞ >

Step 10 u ← j

k:1, key(k) = 1, π[k] ← j

m:2, m is not in queue

l:3, l is not in queue

Q: < k:1, h:3, i:∞ >

HW-24

LectureNotes.in

Step 11 $u \leftarrow k:1$

$i:1$, $key(i)=1$, $\pi[i] \leftarrow k$

$j:1$, j is not in queue

$Q: < i:1, h:3 >$

Step 12 $u \leftarrow i$

$h:2$, $key(h)=2$, $\pi[h] \leftarrow i$

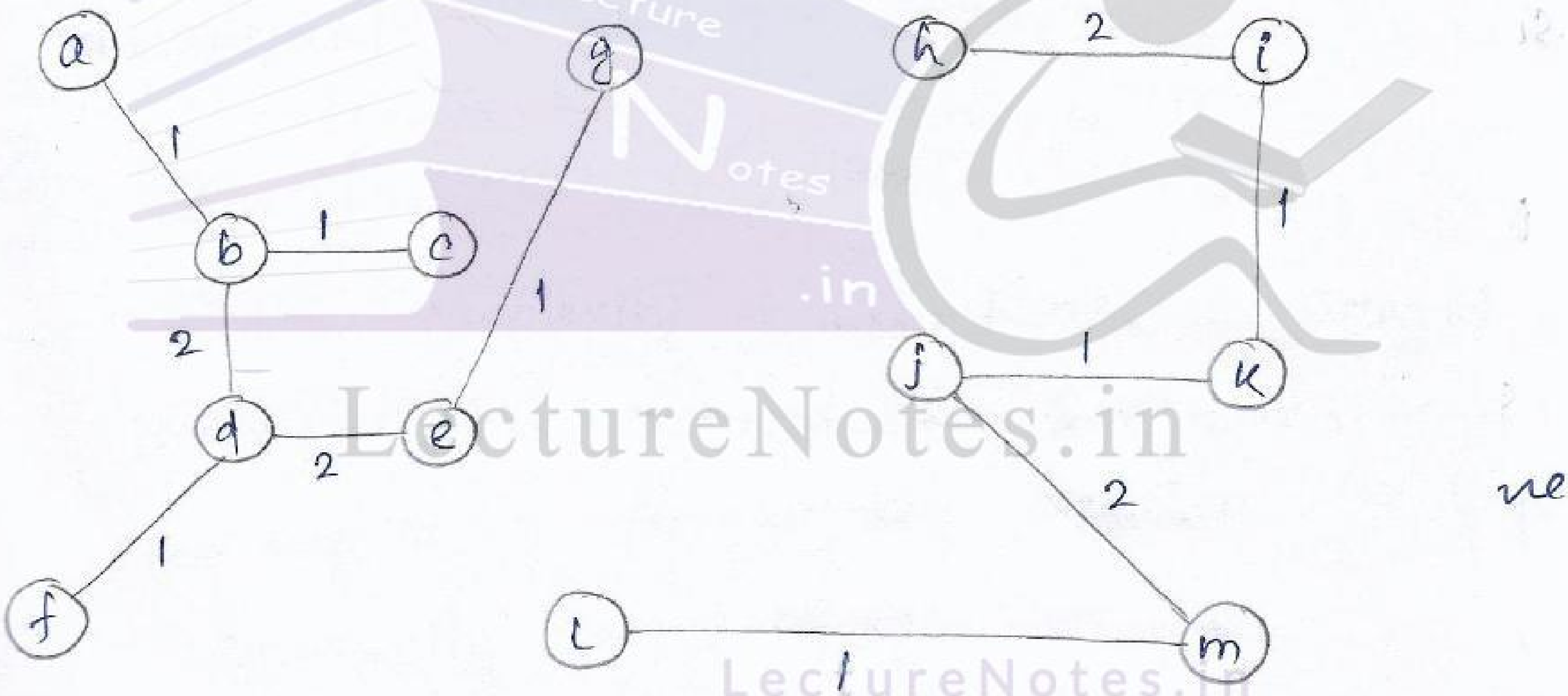
$k:1$, k is not in queue

$Q: < h:2 >$

Step 13 $u \leftarrow h$

$g:3$, g is not in queue

$i:2$, i is not in queue



The minimum weight is 15.

4. KEY[r] ← 0

5. Q ← V[G] // Q is a Min Priority Queue where Priority is the key value of vertex.

6. WHILE Q IS NOT EMPTY

7. u ← EXTRACT_MIN(Q)

8. FOR EACH VERTEX v ∈ Adj(u)

9. IF v ∈ Q AND w(u,v) < KEY(v)

10. KEY[v] ← w(u,v) ← O(log v)

11. π[v] ← u

[END IF]

[END FOR]

[END]WHILE]

Analysis of Prim's Algorithm

The time complexity of Extract_min is O(log v)

loop 1 will be executed for O(v) times.

The step 10 is decrease key operation, the time complexity is O(log v)

The loop 6 will execute for [v] times

Hence, the time complexity of step 6 is

O(v log v)

So, the total time complexity is

$T(v, E) = \theta(v) + O(v \log v)$

$T(V, E) = O(V \log V)$

Shortest Path Problems

There are 2 types of Shortest Path Problems -

1. Single Source Shortest Path Problems
2. All Pair Shortest Path Problems

1. Single Source Shortest Path Problems →

* It will find the shortest path from the source to all other vertex.

* BFS is used if the weight of all edges is same.

* There are 3 algorithms used to solve single source shortest path problem -

(i) BFS

(ii) Bellman Ford Algorithm

(iii) Dijekstri's Algorithm

2. All Pair Shortest Path Problems →

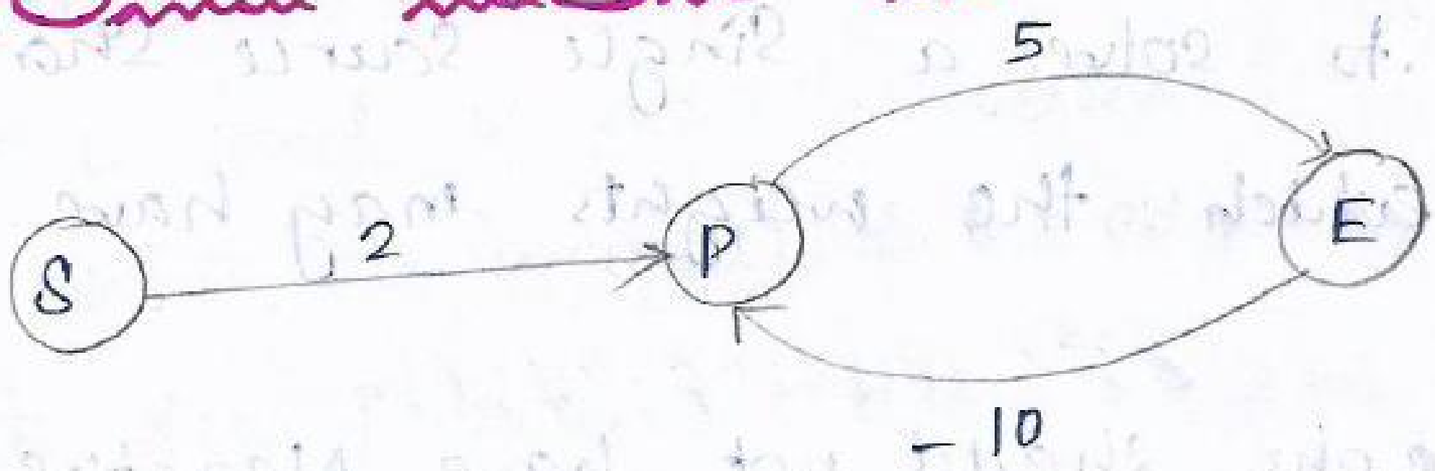
* It can be used to find the shortest path between any two pair of vertex.

* Floyd- Warshall Algorithm is used to solve this.

LectureNotes.in

Dt → 2nd Apr 18
Monday

Negative Weight Cycle →



Path from S → E

S → P → E = 5 + 2 = 7

S → P → E → P → E = 2 + 5 - 10 + 5 = 2

S → P → E → P → E → P → E = -3

When the weight of the edge is negative value, then it is called Negative Weight Edge.

Negative Weight Cycle →

If the graph contains some edges with negative weight and the total weight may get reduced in each cycle. This is called Negative Weight Cycle.

NOTE-

→ If there is a Negative weight cycle on some path from S to V, we find the cost from S to V is $-\infty$. $\delta(s, v) = -\infty$.

→ If there is no path exist from S to V, the cost from S to V is ∞ ($\delta(s, v) \infty$).

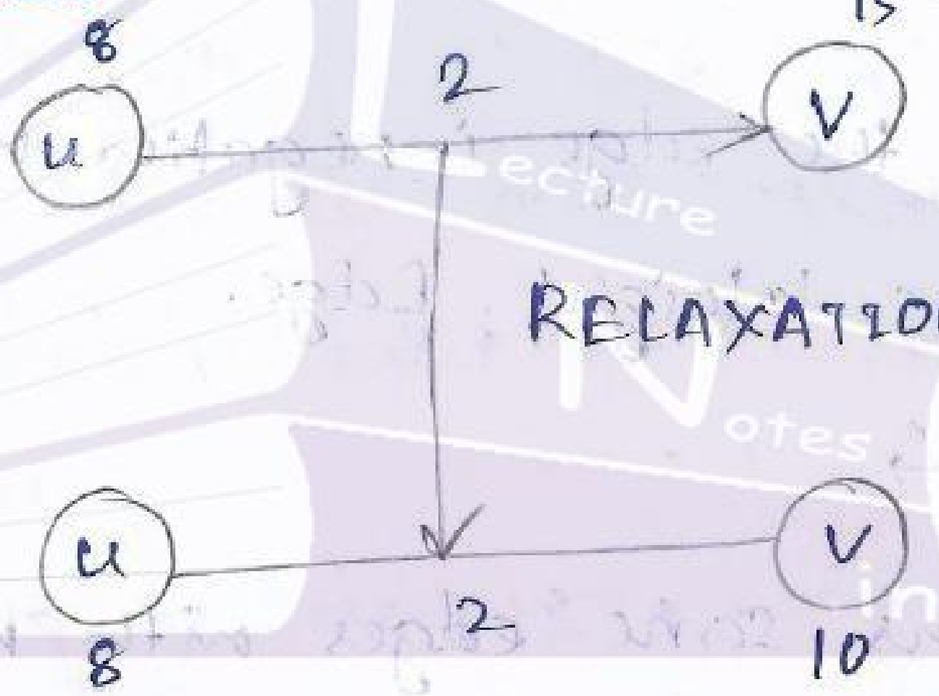
(ii) Bellman Ford Algorithm

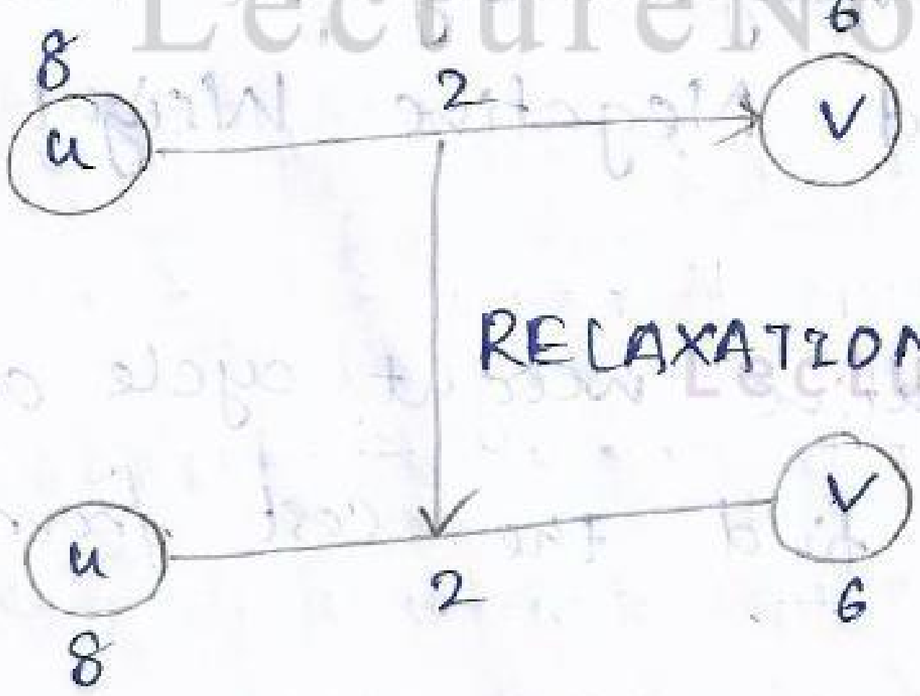
* It is used to solve a Single Source Shortest Path Problem in which the weights may have negative values.

* But the graphs should not have Negative weight Cycle.

* It uses the procedure **Relaxation**.

Relaxation -





LectureNotes.in

Relaxation Procedure →

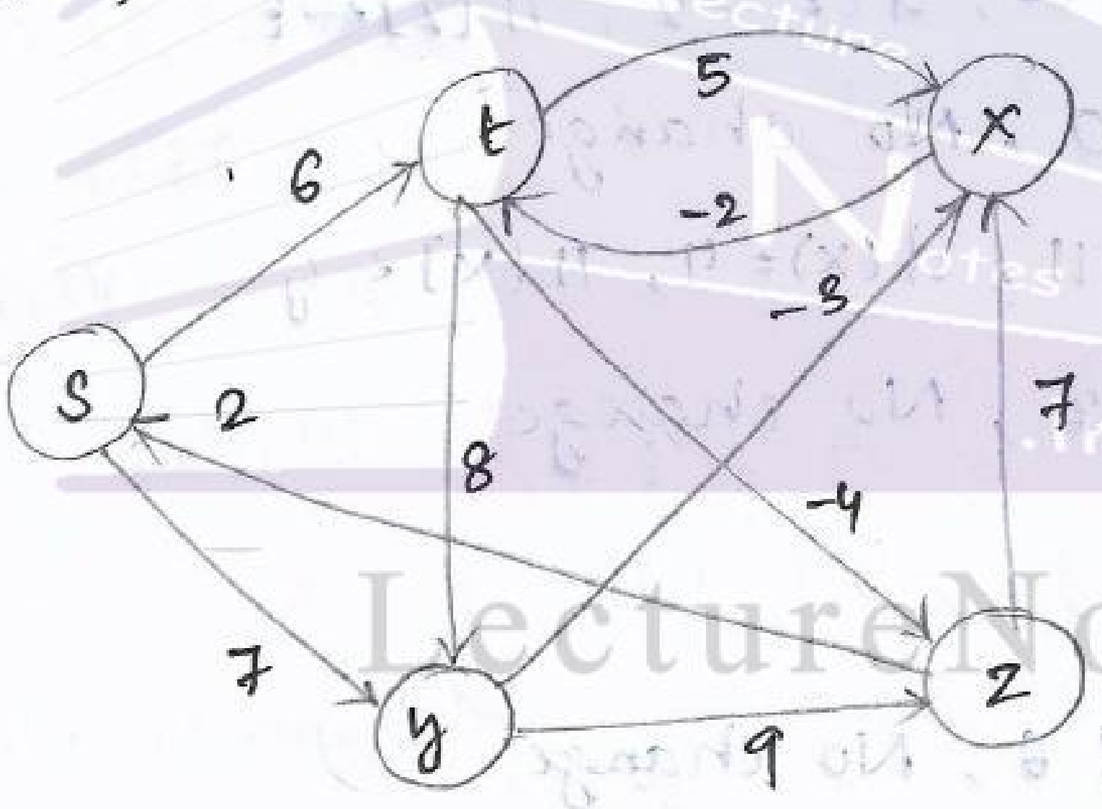
RELAX (u, v, w)

/* u and v are two vertex
w is the weight of the edge (u, v) */

1. IF $d(u) + w(u,v) < d(v)$
2. $d[v] \leftarrow d(u) + w(u,v)$
3. $\pi[v] \leftarrow u$

[END IF]

Q. Using Bellman-Ford Algoritum, find the shortest path from vertex s to all other vertex.



Sol.

Vertex	d(v)	π(v)
s	0	NULL
x	~~∞~~ 4	~~NULL~~ y
t	~~∞~~ 2	~~NULL~~ x
y	~~∞~~ 7	~~NULL~~ s
z	~~∞~~ -2	~~NULL~~ t

If there are 5 vertex, then 4 times we have to relax all the edges.

Step 1

RELAX (s, t, 6) = 0+6 < ∞, d(t) = 6, π[t] ← s

RELAX (t, x, 5) = 6+5 < ∞, d(x) = 11, π[x] ← t

RELAX (x, t, -2) = 11-2 ≮ 6, No change

RELAX (s, y, 7) = 0+7 < ∞, d(y) = 7, π[y] ← s

RELAX (t, y, 8) = 6+8 ≮ 7, No change

RELAX (z, x, 7) = ∞+7 ≮ 11, No change

RELAX (t, z, -4) = 6-4 < ∞, d(z) = 2, π[z] ← t

RELAX (z, s, 2) = 2+2 ≮ 0, No change

RELAX (y, x, -3) = 7-3 < 11, d(x) = 4, π[x] ← y

RELAX (y, z, 9) = 7+9 ≮ 2, No change

Step 2

RELAX (s, t, 6) = 0+6 ≮ 6, No change

RELAX (t, x, 5) = 6+5 ≮ 4, No change

RELAX (x, t, -2) = 4-2 < 6, d(t) = 2, π[t] ← x

RELAX (s, y, 7) = 0+7 ≮ 7, No change

RELAX (t, y, 8) = 2+8 ≮ 7, No change

RELAX (z, x, 7) = 2+7 ≮ 4, No change

RELAX (t, z, -4) = 2-4 < 2, d(z) = -2, π[z] ← t

LectureNotes.in

RELAX (z, s, 2) = -2+2 ≯ 0, No change

RELAX (y, x, -3) = 7-3 ≯ 4, No change

RELAX (y, z, 9) = 7+9 ≯ -2, No change

Step 3

RELAX (s, t, 6) = 0+6 ≯ 2, No change

RELAX (t, x, 5) = 2+5 ≯ 4, No change

RELAX (x, t, -2) = 4-2 ≯ 2, No change

RELAX (s, y, 7) = 0+7 ≯ 7, No change

RELAX (t, y, 8) = 2+8 ≯ 7, No change

RELAX (z, x, 7) = -2+7 ≯ 4, No change

RELAX (t, z, -4) = 2-4 ≯ -2, No change

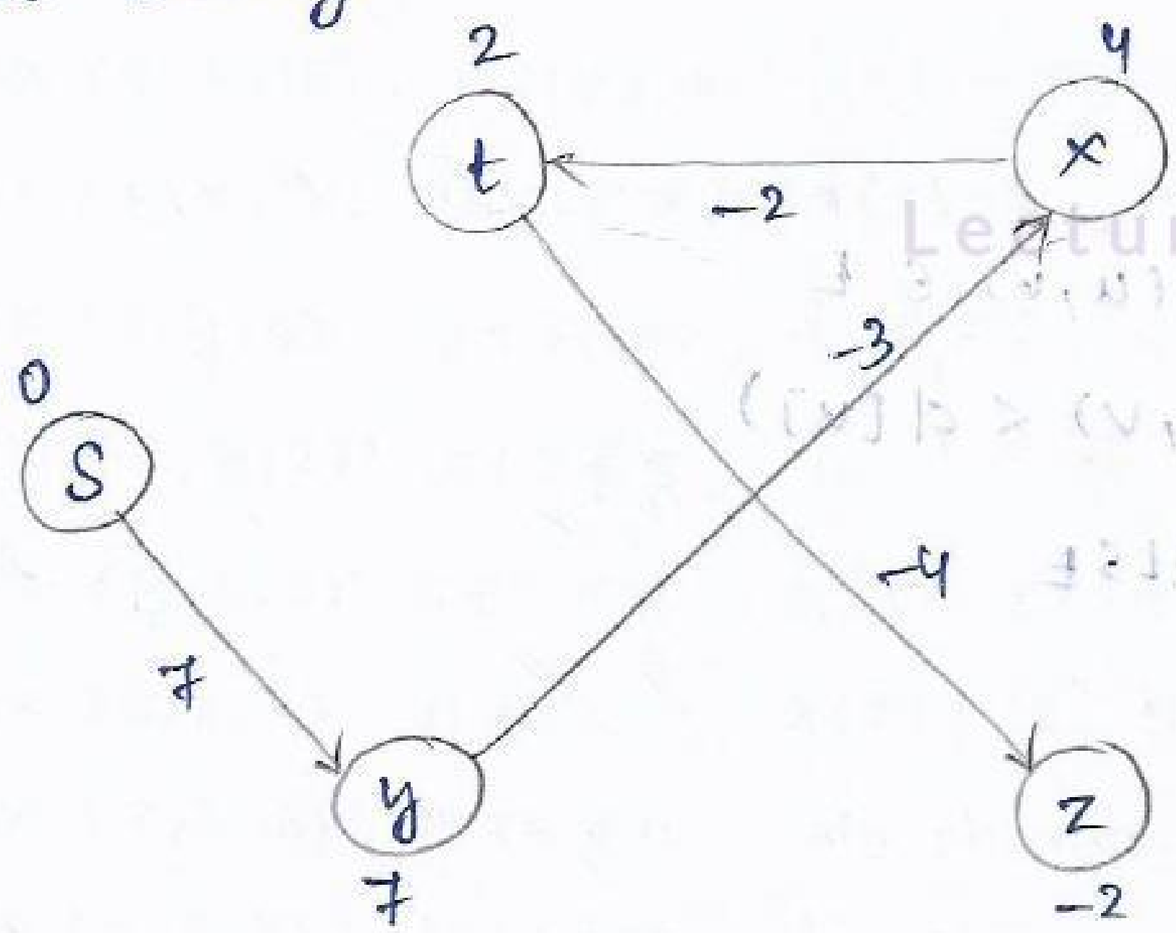
RELAX (z, s, 2) = -2+2 ≯ 0, No change

RELAX (y, x, -3) = 7-3 ≯ 4, No change

RELAX (y, z, 9) = 7+9 ≯ -2, No change

Step 4

No change



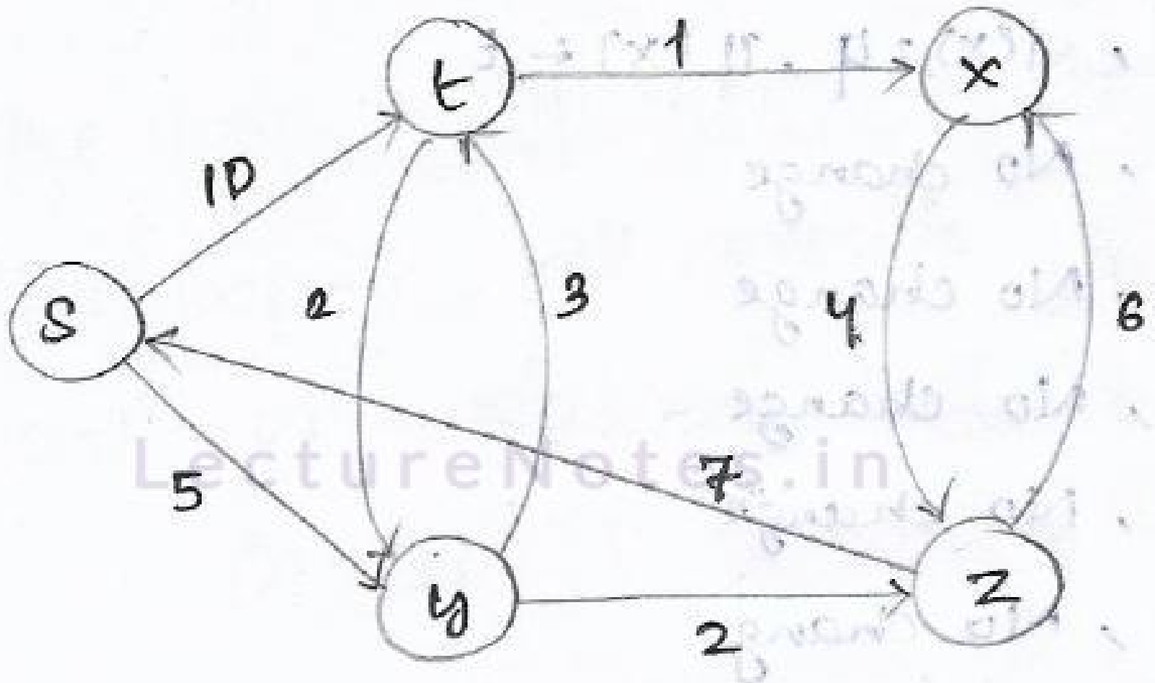
LectureNotes.in

Algorithm (Bellman-Ford Algorithm)

```
BELLMAN_FORD (G, W, S)
/* G is the graph with set of vertex and edges
        G = (V, E)
   W is the weight of each edge.
   S is source vertex */
1. FOR EACH VERTEX u ∈ V[G]
2.        d[u] ← ∞
3.        π[u] ← NULL
   [END FOR]
4. d[s] ← 0
5. FOR I ← 1 to |V| - 1
6.        FOR EACH EDGE (u, v) ∈ E[G]
7.                RELAX (u, v, w)
          [END FOR]
   [END FOR]
8. FOR EACH EDGE (u, v) ∈ E
9.        IF (d(u) + w(u, v) < d[v])
10.               RETURN FALSE
          [END IF]
   [END FOR]
```

LectureNotes.in

Q. Using Bellman-Ford Algorithm, find the shortest path from vertex s to all other vertex.



Sol.

Vertex	d(v)	π(v)
s	0	NULL
x	~~∞~~ 11	~~NULL~~ t
t	~~∞~~ 10	~~NULL~~ s
y	~~∞~~ 5	~~NULL~~ s
z	~~∞~~ 7	~~NULL~~ y

Step 1 -

RELAX (s, t, 10): 0+10 < ∞, d(t) = 10, π[t] ← s

RELAX (t, x, 1): 10+1 < ∞, d(x) = 11, π[x] ← t

RELAX (s, y, 5): 0+5 < ∞, d(y) = 5, π[y] ← s

RELAX (t, y, 2): 10+2 ≮ 5, No change

RELAX (y, t, 3): 5+3 < 10, d(t) = 8, π[t] ← y

RELAX (x, z, 4): 11+4 < ∞, d(z) = 15, π[z] ← x

RELAX (z, x, 6): 15+6 ≮ 11, No change

RELAX (z, s, 7): 15+7 ≮ 0, No change

RELAX (y, z, 2): 5+2 < 15, d(z) = 7, π[z] ← y

Step 2

RELAX (s, t, 10): 0+10 ⊀ 8, No change

RELAX (t, x, 1): 8+1 < 11, d(x) = 9, π[x] ← t

RELAX (s, y, 5): 0+5 ⊀ 5, No change

RELAX (t, y, 2): 8+2 ⊀ 5, No change

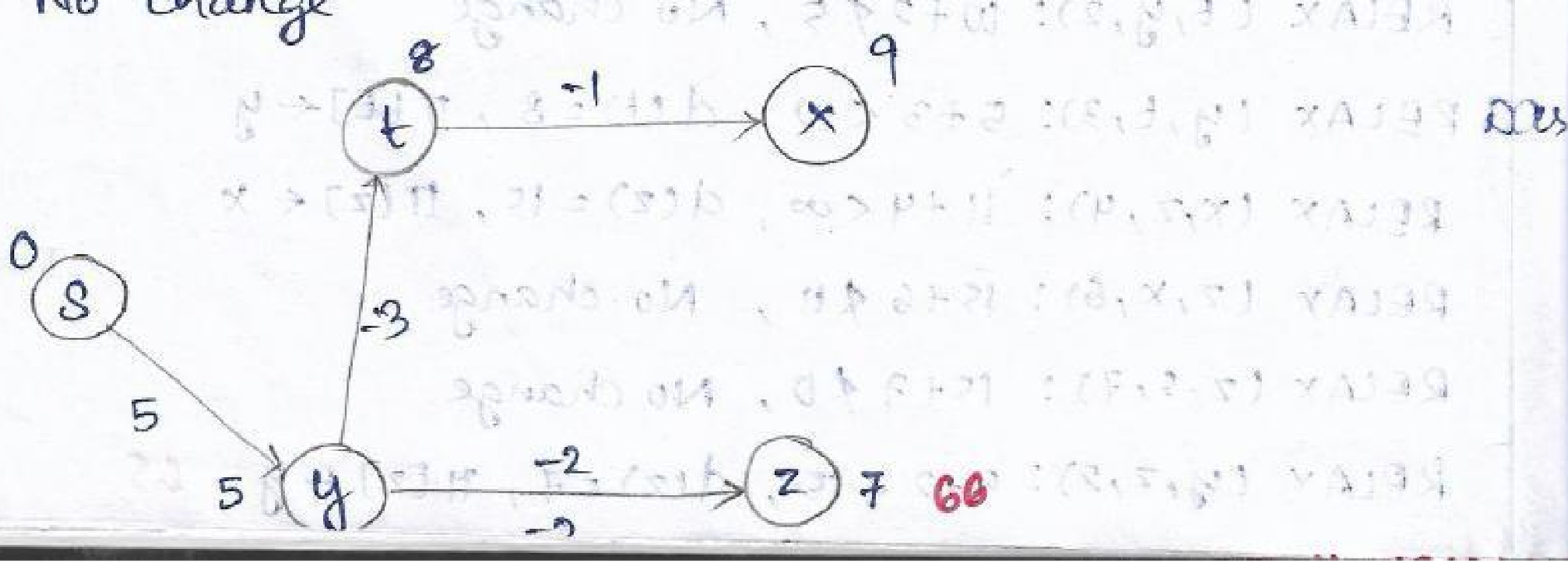
RELAX (y, t, 3): 5+3 ⊀ 8, No change

RELAX (x, z, 4): 9+4 ⊀ 7, No change

RELAX (z, x, 6): 7+6 ⊀ 9, No change

RELAX (z, s, 7): 7+7 ⊀ 0, No change

1. RELAX (y, z, 2): 5+2 ⊀ 7, No change
2. Step 3
3. RELAX (s, t, 10): 0+10 ⊀ 8, No change
4. RELAX (t, x, 1): 8+1 ⊀ 9, No change
5. RELAX (s, y, 5): 0+5 ⊀ 5, No change
6. RELAX (t, y, 2): 8+2 ⊀ 5, No change
7. RELAX (y, t, 3): 5+3 ⊀ 8, No change
8. RELAX (x, z, 4): 9+4 ⊀ 7, No change
   RELAX (z, x, 6): 7+6 ⊀ 9, No change
   RELAX (z, s, 7): 7+7 ⊀ 0, No change
   RELAX (y, z, 2): 5+2 ⊀ 7, No change
9. Step 4
   No change
10.



Time Complexity

The Step-4 has time complexity as $O(V)$.

The loop-5 will execute for $O(V)$ times.

The loop-6 will execute for $O(E)$ times.

The cost of loop-5 is (As loop-6 is inside loop-5)

$O(VE)$.

The total cost is

$$T(V,E) = \theta(V) + O(V,E)$$

$$T(V,E) = O(V,E)$$

ii) Dijekstri's Algorithm →

→ It can be used to solve a Single Source Shortest Path Problem. of a directed graph. (directed weighted graph)

→ $G = (V, E)$ for the case in which all edges weights is non negative, i.e., $w(u,v) \geq 0$

→ Dijekstri's Algorithm has less time complexity as compared to Bellman-Ford Algorithm.

→ It uses Min-Priority-Queue for implementation

LectureNotes.in

Algorithm (Dijekstri's Algorithm)

DIJEKSTRI's (G, w, s)

/* G is the graph with set of vertex and edges

$G = (V, E)$

w is the weight of each edge.

s is the source vertex */

1. FOR EACH VERTEX $u \in V[G]$
2. $d[u] \leftarrow \infty$
3. $\Pi[u] \leftarrow$ NULL

[END FOR]

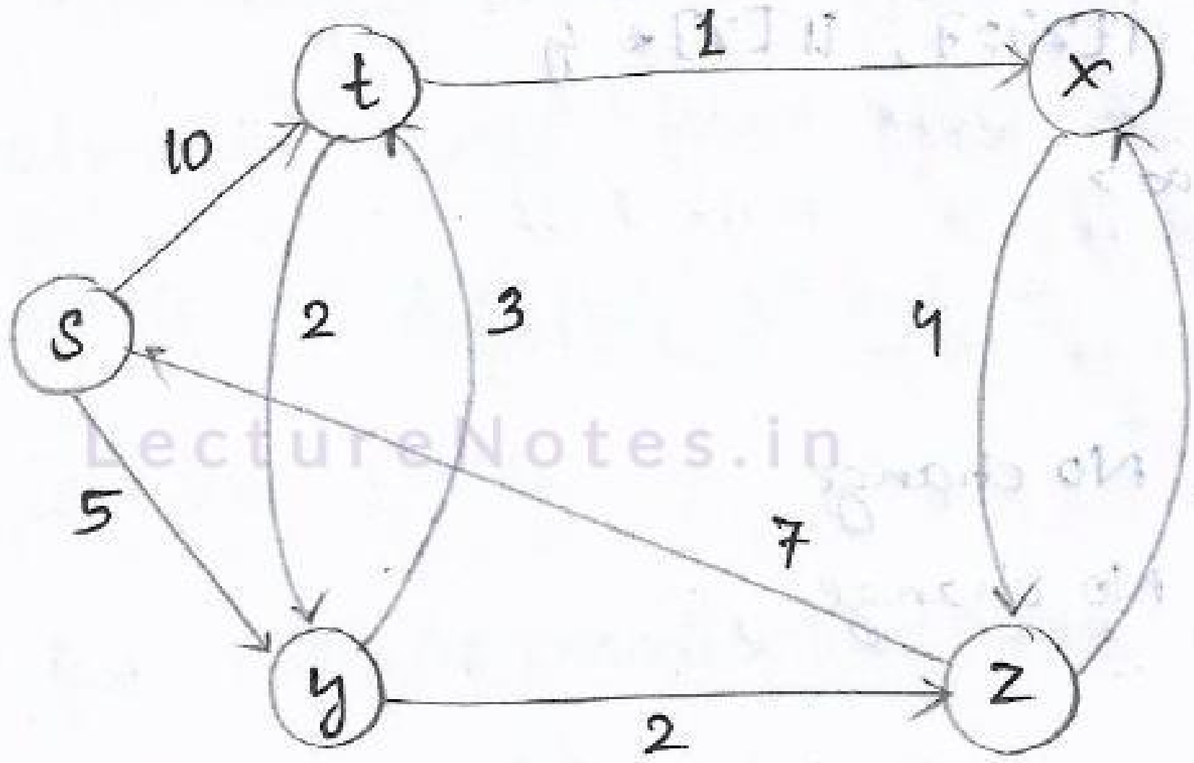
4. $d[s] \leftarrow 0$
5. $T \leftarrow \{\phi\}$
6. $Q \leftarrow V[G]$ // Q is a min-priority-queue where the cost of vertex is key value
7. WHILE Q IS NOT EMPTY
8. $u \leftarrow$ EXTRACT_MIN(Q)
9. $T \leftarrow T \cup \{u\}$
10. FOR EACH VERTEX $v \in Adj(u)$ AND $v \in Q$
11. RELAX (u, v, w)

[END FOR]

[END WHILE]

LectureNotes.in

Q. Find the shortest path from the vertex to all other vertex using Dijekstri's Algorithm.



Sol.

Vertex	Adj(v)	d(v)	π[v]
s	t:10, y:5	0	NULL
t	x:1, y:2	~~∞~~ ~~10~~ 8	~~NULL~~ s y
x	z:4	~~∞~~ 9	~~NULL~~ t
y	t:3, z:2	~~∞~~ 5	~~NULL~~ s
z	x:6, s:7	∞ 7	~~NULL~~ y

Q : < s:0, t:∞, x:∞, y:∞, z:∞ >

Step 1

u ← s

RELAX (s, t, 10) : 0+10 < ∞ , d[t] = 10 , π[t] ← s

RELAX (s, y, 5) : 0+5 < ∞ , d[y] = 5 , π[y] ← s

Q : < y:5, t:10, x:∞, z:∞ >

Step 2 $u \leftarrow y$

RELAX $(y, t, 3)$: $5+3 < 10$, $d[t] = 8$, $\pi[t] \leftarrow y$

RELAX $(y, z, 2)$: $5+2 < \infty$, $d[z] = 7$, $\pi[z] \leftarrow y$

$Q : \langle z:7,\ t:8,\ x:\infty \rangle$

Step 3 $u \leftarrow z$

RELAX $(z, x, 6)$: $\infty + 6 \not< 4$, No change

RELAX $(z, s, 7)$: $\infty + 7 \not< 0$, No change

$Q : \langle t:8,\ x:\infty \rangle$

Step 4 $u \leftarrow t$

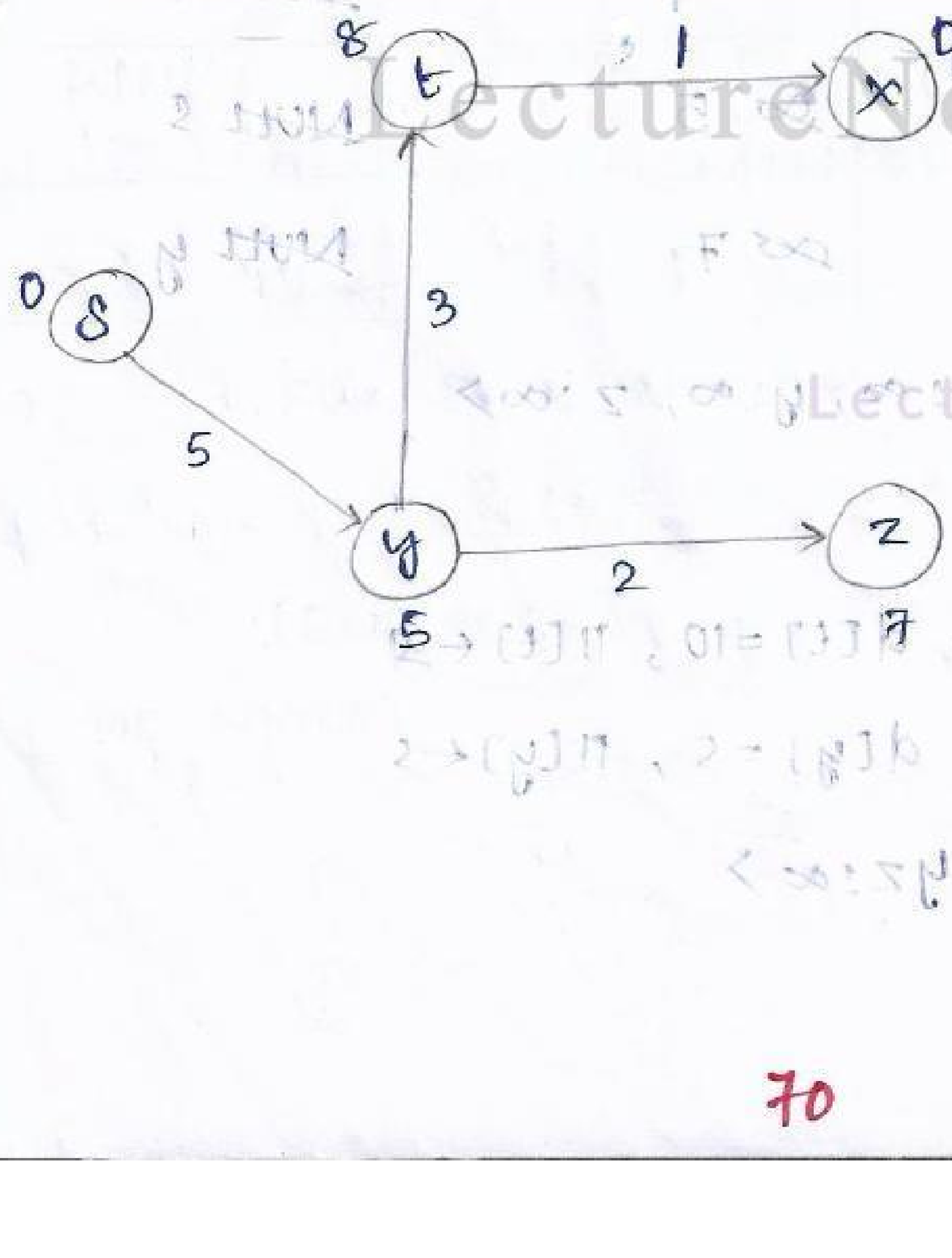
RELAX $(t, x, 1)$: $8+1 < \infty$, $d[x] = 9$, $\pi[x] \leftarrow t$

RELAX $(t, y, 2)$: $8+2 \not< 5$, No change

$Q : \langle x:\infty \rangle$

Step 5 $u \leftarrow x$

RELAX $(x, z, 4)$: $9+4 \not< 7$, No change



LectureNotes.in

Floyd - Warshall Algorithm

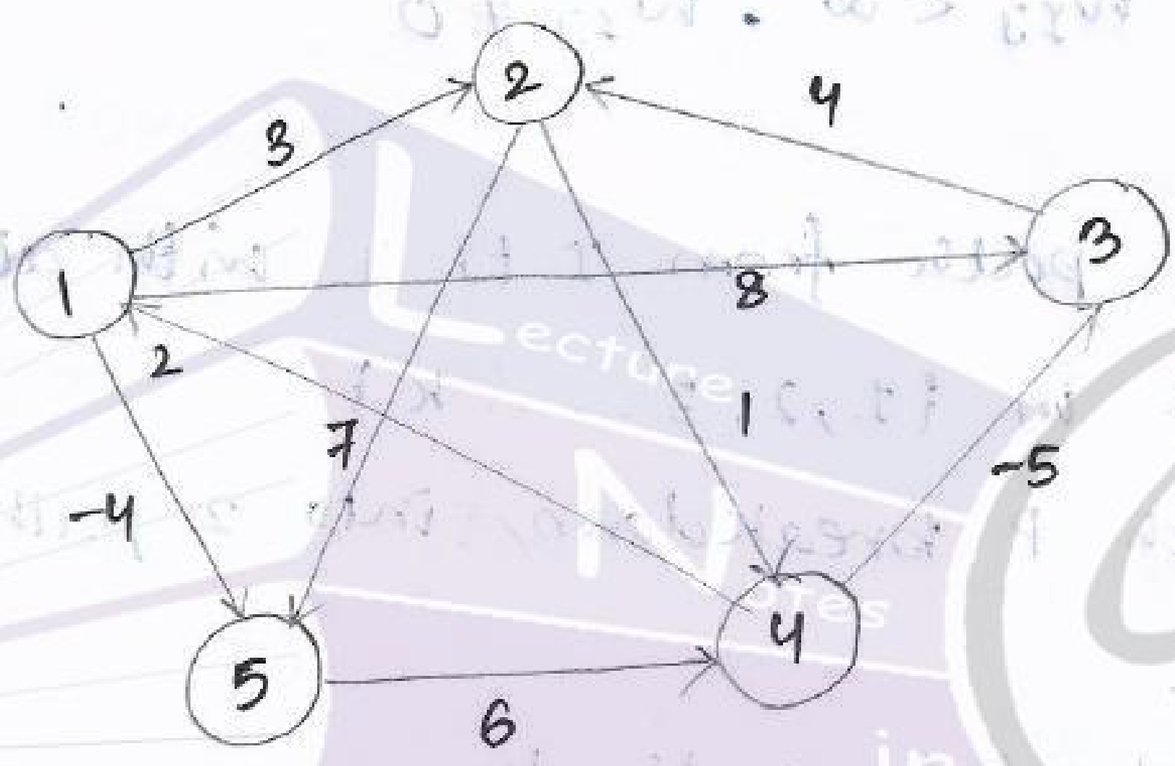
Dt → 5th Apr 18
Thursday

→ This algorithm is used to find the shortest path between any two pair of vertex.

→ It is a All Pair Shortest Path Problem.

→ Let the vertex of graph G be 1, 2, 3, ... n.

Q. Find the weight matrix of the following data.



Weight Matrix →

The weight matrix represents the cost between two vertices with no intermediate vertices.

Sol.

$$W = D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \end{array}$$

$$\Pi^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \text{NULL} & 1 & 1 & \text{NULL} & 1 \\ \text{NULL} & \text{NULL} & \text{NULL} & 2 & 2 \\ \text{NULL} & 3 & \text{NULL} & \text{NULL} & \text{NULL} \\ 4 & \text{NULL} & 4 & \text{NULL} & \text{NULL} \\ \text{NULL} & \text{NULL} & \text{NULL} & 5 & \text{NULL} \end{bmatrix} \end{array}$$

$$\Pi_{ij}^{(0)} = \begin{cases} \text{NULL}, & \text{when } I = J, \ W_{IJ} = \infty \\ I, & W_{IJ} < \infty, \ W_{IJ} \neq 0 \end{cases}$$

Let P is the path from I to J with all intermediate vertices in $\{1, 2, 3 \ldots k\}$

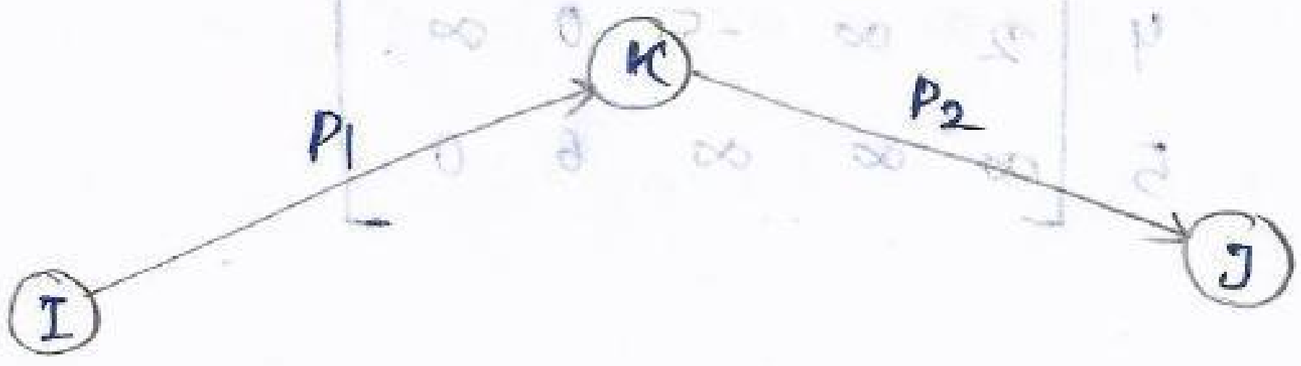
Let the Path P breakdown into 2 paths through intermediate vertex.

P_1 is the shortest path from I to k with all intermediate vertices in the set.

$P_1 : \{1, 2, 3 \ldots k\}$

P_2 is the shortest path from k to J with all intermediate vertices in the set.

$P_2 : \{1, 2, 3 \ldots k\}$



$$d_{IJ}^{(K)} = \begin{cases} W_{IJ}, & K = 0 \\ MIN\left(d_{IJ}^{(K-1)}, d_{IK}^{(K-1)} + d_{KJ}^{(K-1)}\right) \end{cases}$$

Algorithm (Floyd-Warshall Algorithm)

LectureNotes.in

```
FLOYD_WARSHALL (W)
/* W is the weight matrix */
1. N ← ROWS[W]
2. D[0] ← W
3. FOR K ← 1 to N
4.     FOR I ← I to N
5.         FOR J ← 1 to N
6.             d_IJ^(K) = MIN (d_IJ^(K-1), d_IK^(K-1) + d_KJ^(K-1))
           [END FOR 5]
       [END FOR 4]
   [END FOR 3]
7. RETURN D
```

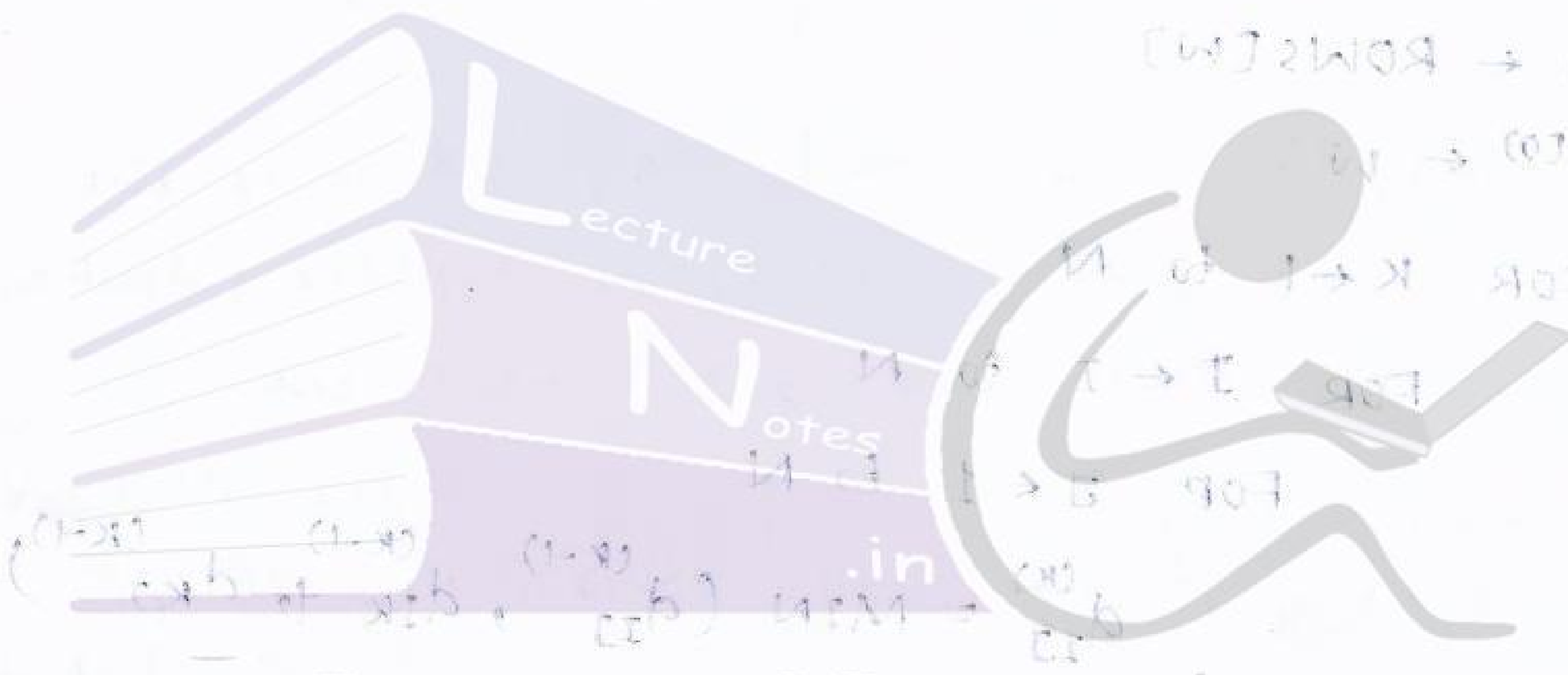
LectureNotes.in

Path Matrix →

$$\Pi_{IJ}^{(K)} = \begin{cases} \Pi_{IJ}^{(K-1)}, & \text{IF } d_{IK}^{(K-1)} + d_{KJ}^{(K-1)} > d_{IJ}^{(K-1)} \\ \Pi_{KJ}^{(K-1)}, & \text{IF } d_{IK}^{(K-1)} + d_{KJ}^{(K-1)} < d_{IJ}^{(K-1)} \end{cases}$$

Q. Find the shortest path between any two pair of vertices

LectureNotes.in



LectureNotes.in

Dt → 6th Apr 18
Friday

String Pattern Matching Algorithm

* In the problem, we will have text T and pattern P. We have to search the pattern P in the text T. If it finds, it will return the index position of text T where the pattern matches.

* The following 3 algorithms can be used for String Pattern Matching:-

1. Naive String Matching Algorithm (Brute Force Approach)
2. Rabin - Karp Algorithm.
3. Knuth - Morris - Pratt Algorithm. (KMP)

1. Naive String Matching Algorithm →

T : a a b c a b a d e f a b c
    1 2 3 4 5 6 7 8 9 10 11 12 13

P : a d e f

Each Shift = 4 comparision.

(N - M + 1) M

Let the text T contains N characters and the pattern P contains M characters. Then the number of comparison is

(N - M + 1) M

In every comparision, we have to do M comparisions.

. So, the time complexity of this Algorithm is

$$T(N, M) = O(NM)$$

* This algorithm search the string in Polynomial time.

Algorithm (Naive String Matching Algorithm)

```
NAIVE_STRING_ PRAT (T, P)
1. N ← LENGTH [T]
2. M ← LENGTH [P]
3. FOR  S ← 0  to  (N-M)
4.          IF  P[1...M] = T[S+1...S+M]
5.                  PRINT "Pattern occurs at shift"
            [END IF]
   [END FOR]
```

Drawbacks →

The time complexity of this algorithm is Polynomial time, whereas Knuth - Morris - Pratt Algorithm claims that pattern can be searched in a linear time.

3. Knuth - Morris - Pratt Algorithm →

This algorithm can be used to search a pattern in a string in linear time.

Prefix and Suffix →

Let the pattern P be P: a b c d a b c

The different prefix of P are: a, ab, abc, abcd, abcda, abcdab, abcdabc.

The different suffix of P are: c, bc, abc, dabc, cdabc, bcdabc, abcdabc

→ abc and abcdabc is a prefix as well as suffix.

Prefix Array of the Pattern -

Prefix Function / Prefix Array 'π' for a pattern specifies the longest suffix which is also prefix

Q. Find the Prefix array of the pattern given below -

Pattern: a b c d a b c

Sol.

M	K	Q
7	0	0 1 2 3 0 1 2
	1	
	2	
	3	

P

i	1	2	3	4	5	6	7
P[i]	a	b	c	d	a	b	c
π[i]	0	0	0	0	1	2	3

Algorithm

COMPUTE_PREFIX_FUNCTION (P)

1. M ← LENGTH [P]
2. π[1] ← 0
3. K ← 0
4. FOR q ← 2 to M
5. WHILE K > 0 AND P[K+1] ≠ P[q]
6. K ← π[K]
   [END IF]
7. IF P[K+1] = P[q]
8. K ← K+1
   [END IF]
9. π[q] ← K
   [END FOR]
10.

LectureNotes.in

Q. Find the prefix function of the following pattern -

P: a c a c a b a c a c a b a c a c

Sol.

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
P[I]	a	c	a	c	a	b	a	c	a	c	a	b	a	c	a	c	a	c
Π[I]	0	0	1	2	3	0	1	2	3	4	5	6	7	8	9	10	11	4

M = 18 K = 0

q = 2

Q. Find the prefix function of the following pattern -

P: a a b a a b a a a

Sol.

I	1	2	3	4	5	6	7	8	9
P[I]	a	a	b	a	a	b	a	a	a
Π[I]	0	1	0	1	2	3	4	5	2

Q. Find the prefix array of the following patterns -

1. P: a b c d a b c a

2. P: a b c d a b c y

Sol.

1. P: a b c d a b c a

I	1	2	3	4	5	6	7	8
P[I]	a	b	c	d	a	b	c	a
Π[I]	0	0	0	0	1	2	3	4

2. P: abcdabcy

I	1	2	3	4	5	6	7	8
P[I]	a	b	c	d	a	b	c	y
Π[I]	0	0	0	0	1	2	3	0

Dt → 10th Apr 18
Tuesday

Q. Let the text be

T: abcxabcdabxabcdabcdabcy

p: abcdabcy

Find the prefix array of pattern P.

Sol. K=0

I	1	2	3	4	5	6	7	8
P[I]	a	b	c	d	a	b	c	y
Π[I]	0	0	0	0	1	2	3	0

Step 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
T	a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y

q=0

	a	b	c	d	a	b	c	y
Π[q]								
Π[I]	0	0	0	0	1	2	3	0
	1	2	3	4	5	6	7	8

MISMATCH at I=4, q=3

q ← Π[q] = 0

Step 2

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
T	a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y

q=0

	1	2	3	4	5	6	7	8
π[q]	a	b	c	d	a	b	c	y
π[I]	0	0	0	0	1	2	3	0

MISMATCH at I=11, q=6

$q \leftarrow \pi[q] = 2$

$P[3] \neq T[11]$

$q \leftarrow \pi[q] = 0$

Step 3

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
T	a	b	c	x	a	b	c	d	a	b	x	a	b	c	d	a	b	c	d	a	b	c	y

q=0

	1	2	3	4	5	6	7	8
π[q]	a	b	c	d	a	b	c	y
π[i]	0	0	0	0	1	2	3	0

MISMATCH at I=19, q=7

$q \leftarrow \pi[q] = 3$

[END WHILE]

8. IF (P[q+1] = T[i])

9. q ← q+1

[END IF]

10. IF (q = M)

11. PRINT "Pattern found at Index" I-M

[END IF]

[END FOR]

Time Complexity -

Time Complexity in step 3, Compute prefix function is O(M)

According to Amotized Analysis, the time complexity of Knuth-Morris Pratt Algorithm is

T(n) = O(N+M)

LectureNotes.in

2. Rabin Karp Algorithm

Q. Consider T = "3141 5926 53 589793" and P = "26". Let q be the prime number, q=11.

Sol. T = 3141 5926 53 589793

P = 26

q = 11

P % q = 26 % 11 = 4

T0 = 31, 31 % 11 = 9

T1 = 14, 14 % 11 = 3

T2 = 41, 41 % 11 = 8

T3 = 15, 15 % 11 = 4 (T3 <> P) Sprious Hit

Compare the two characters of T3 with P.

This is called Sprious Hit. because remainder is same but the string or character (T3 and T2) is not same.

T4 = 59, 59 % 11 = 4 (T4 <> P) Sprious Hit

T5 = 92, 92 % 11 = 4 (T5 <> P) Sprious Hit

T6 = 26, 26 % 11 = 4 (T6 = P) Valid Hit

T7 = 65, 65 % 11 = 10 (T7 <> P)

T8 = 53, 53 % 11 = 9 (T8 <> P)

T9 = 35, 35 % 11 = 2 (T9 <> P)

T10 = 58, 58 % 11 = 3 (T10 <> P)

T11 = 89, 89 % 11 = 1 (T11 <> P)

T12 = 97, 97 % 11 = 9 (T12 <> P)

$T_{13} = 79$, $79 \% 11 = 2$ ($T_{13} <> P$)

$T_{14} = 93$, $93 \% 11 = 5$ ($T_{14} <> P$)

Sprias Hit = 3, Valid Hit = 1

* In Naire String Matching Algoritum, in each shift, we have to compare with M characters of T with M characters of P.

* But in Robin Karp Algorithm, we have to compare only when the remainder is same.

* In the previous example, the number of spriaus Hit is 3 and the valid hit is 1, i.e., only 4 times we have to compare each M characters of T with P.

* Rabin and karp has proposed a String Matching Algorithm in effective way as compared to Naive String Matching Algorithm.

* Let T[1..N] - T is a text with N characters.

P[1..M] - P is a pattern with M characters.

S - is a shift starting from 0.

Ts - is a decimal value of length M, i.e., from T[s+1 ... s+M] for $s = 0, 1, 2 \ldots (N-M)$

Q. Let T = 235902314152673992 1 and P = 26739
Consider q as prime number. q = 13.
Find the number of Sprious Hit and Valid Hit.

Sol. T = 2359023141526739921

P = 26739

q = 13

P % q = 26739 % 13 =

t_0 = 23590

t_1 = 35902

$$t_{s+1} = d[t_s - d^{M-1}(T[s+0])] + T[s+M+1]$$

where d = 2 (for Binary number)

d = 8 (for Octal number)

d = 10 (for decimal number)

d = 16 (for Hexadecimal number)

$t_1 = t_{0+1} = 10[t_0 - 10^4 \times T[1]] + T[0+5+1]$

$= 10[23590 - 10^4 \times 2] + T[6]$

$= 10[23590 - 20000] + 2$

$= 35900 + 2$

$t_1 = 35902$

for (i=1 to M)

$t_0 \leftarrow (d t_0 + T[i]) \bmod 13$

i	t_0
1	0
2	2
3	10
4	1
5	6
	8

Algorithm (Rabin-Karp Algorithm)

RABIN_KARP_MATCHER (T, P, D, Q)

1. N = LENGTH [T]
2. M = LENGTH [P]
3. $H = D^{M-1} \text{ MOD } q$
4. $t_0 = 0$
5. $P = 0$
6. FOR $I \leftarrow 1$ to M
7. $t_0 \leftarrow (d t_0 + T[I]) \bmod q$
8. $P \leftarrow (dP + P[I]) \bmod q$

   [END FOR]
9. FOR $S \leftarrow 0$ to N-M
10. IF $P = t_S$
11. IF $P[1..M] = T[S+1..S+M]$
12. PRINT "Pattern occurs at Shift" S

    [END IF]

[END IF]

LectureNotes.in

```
13.     IF S < (N-M) THEN
14.         ts+1 = d[ts - h[T∈(S+1)]) + T(S+M+1).
        [END IF]
    [END FOR]
```

Time Complexity

The loop 6 will execute for N times and the time complexity for loop 6 is $O(N)$

The loop 9 will execute for $(N-M+1)$ times. o

In Step 11, to compare M characters from T with M character from P, the time complexity is $O(M)$.

In Best case, there is only one valid match,

∴ The Time Complexity in Best case is

$$T(N) = O(M) + O(N-M+1) \times 1$$

$$T(N) = O(M) \quad \text{Best Case}$$

In Worst case, the time complexity is

$$T(N) = O(M) + O(N-M+1) \times M$$

$$T(N,M) = O(N,M) \quad \text{Worst Case}$$

Backtracking

Dt → 11th Apr 18
Wednesday

* It traverse the search space by DFS (Depth First Search).
* If the given constraint fails, it undoes the last choice and backtrack to the previous choice.
* It searches the State Space Tree until search is successful.
* It cannot be used to solve optimization problem.
* The following problems can be solved using Backtracking Approach -

1. N Queen Problem
2. Subset Sum Problem

1. N Queen Problem using Backtracking →

→ There is a board with N rows and N columns.

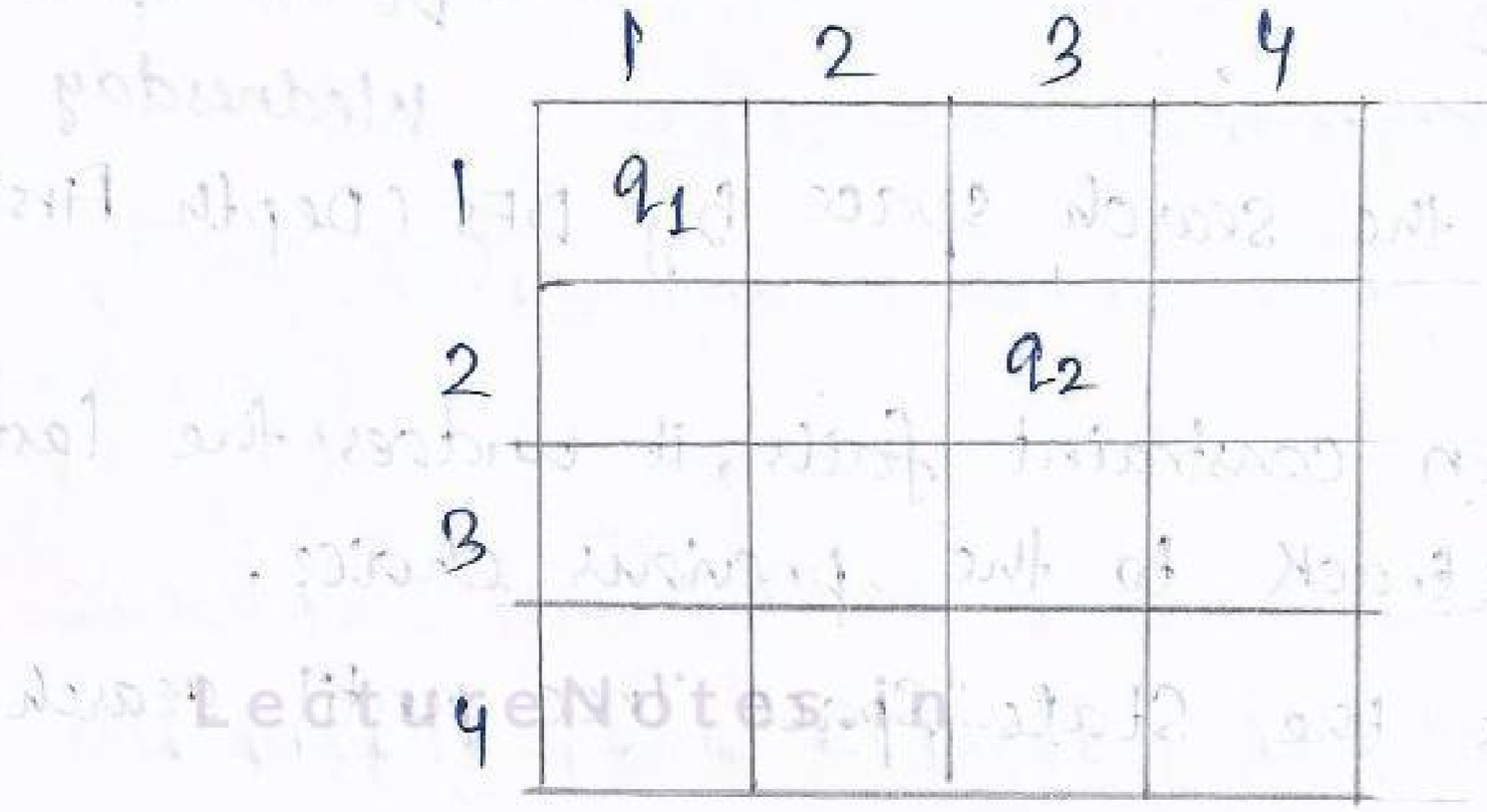
→ The queen should be placed in such a way that no queen should attack to other, i.e., the queen should not be in same row.

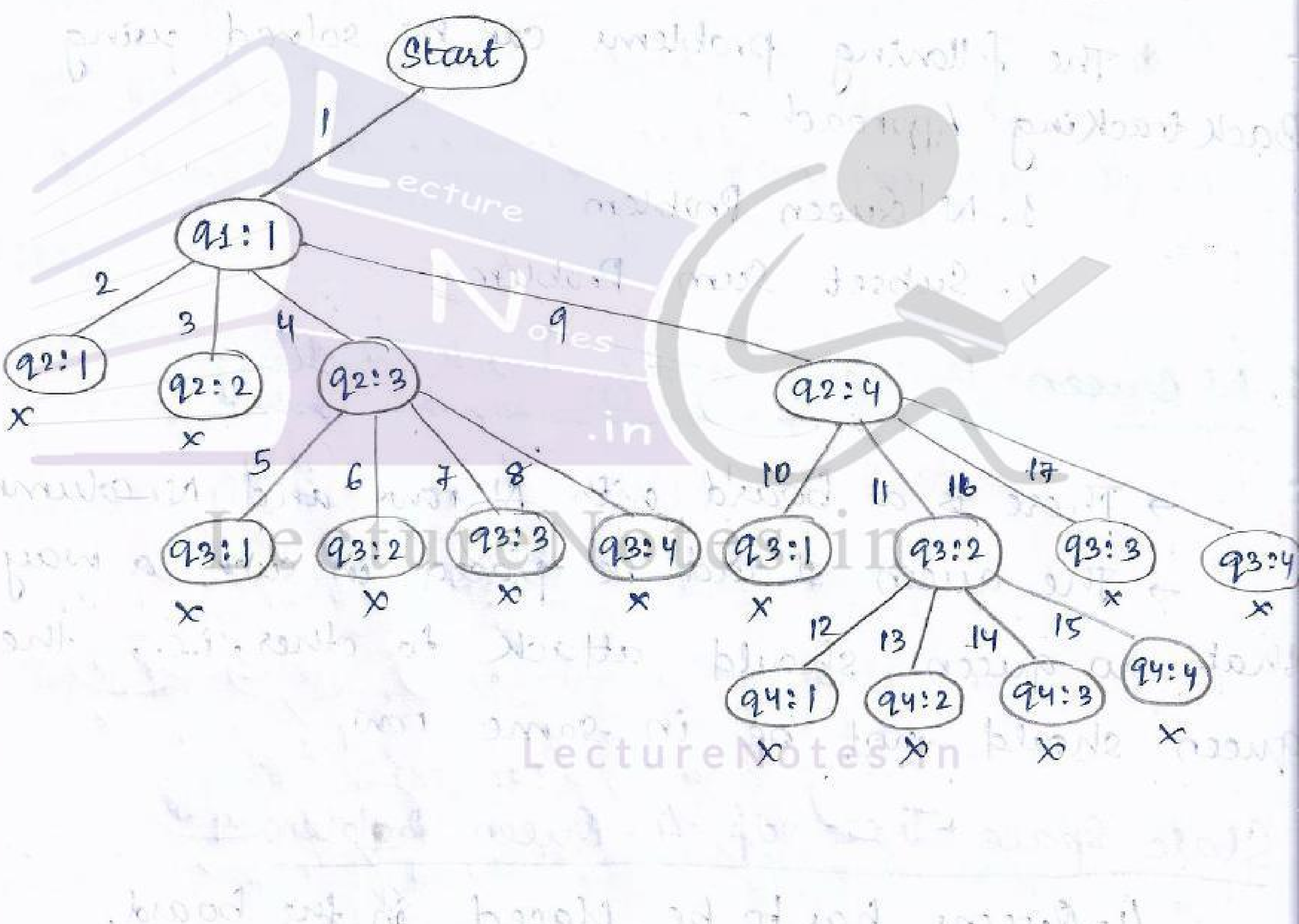
State Space Tree of 4-Queen Problem →

4-Queens has to be placed in the board.

q_1, q_2, q_3, q_4

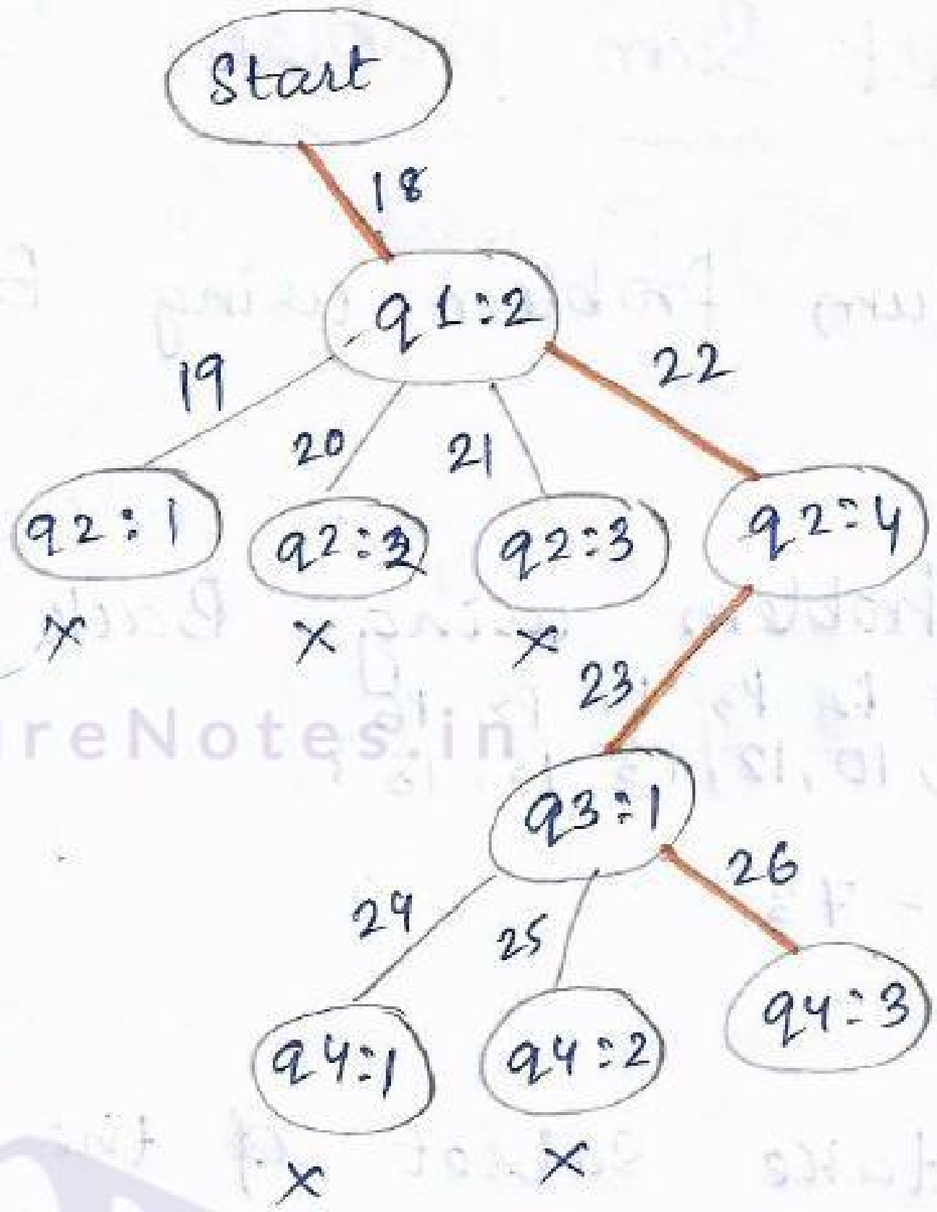
LectureNotes.in

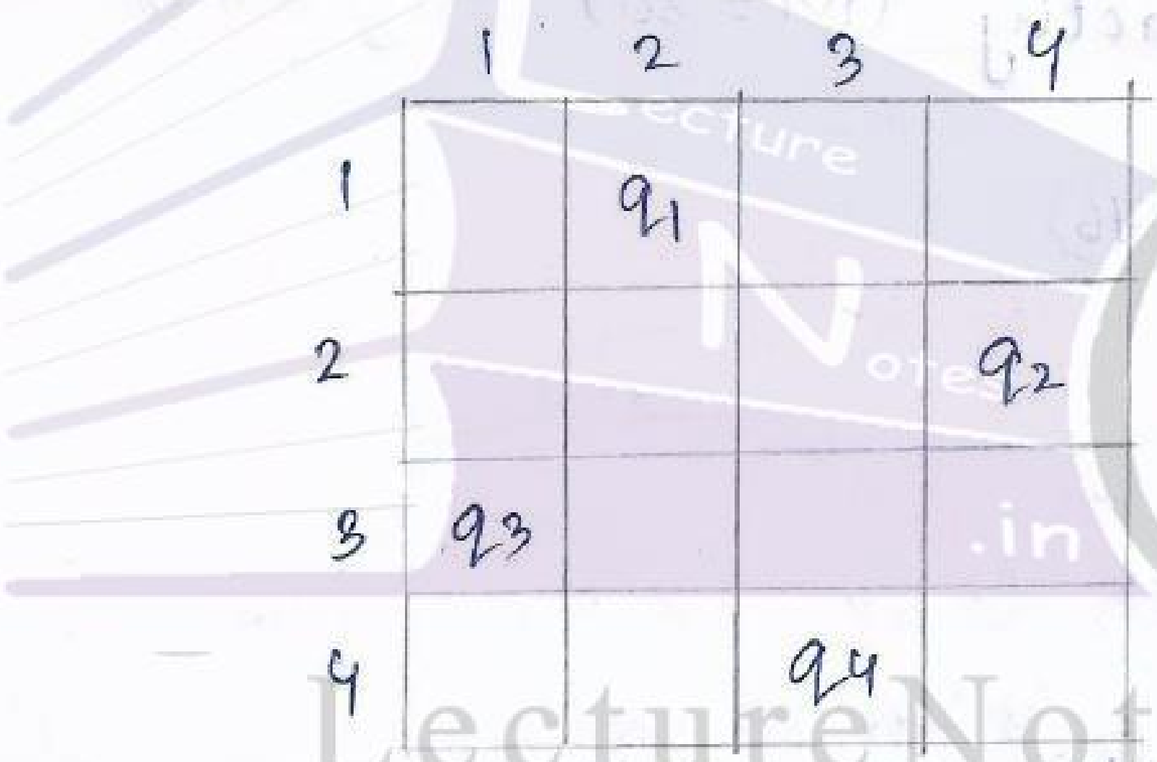




LectureNotes.in

Dt → 13th Apr 18
Friday





	1	2	3	4
q	q3	q1	q4	q2

Q. Derive 8 queen problem using Back Tracking Process.

LectureNotes.in

Subset Sum Problem

Q. Solve Subset Sum Problem using Back Tracking approach.

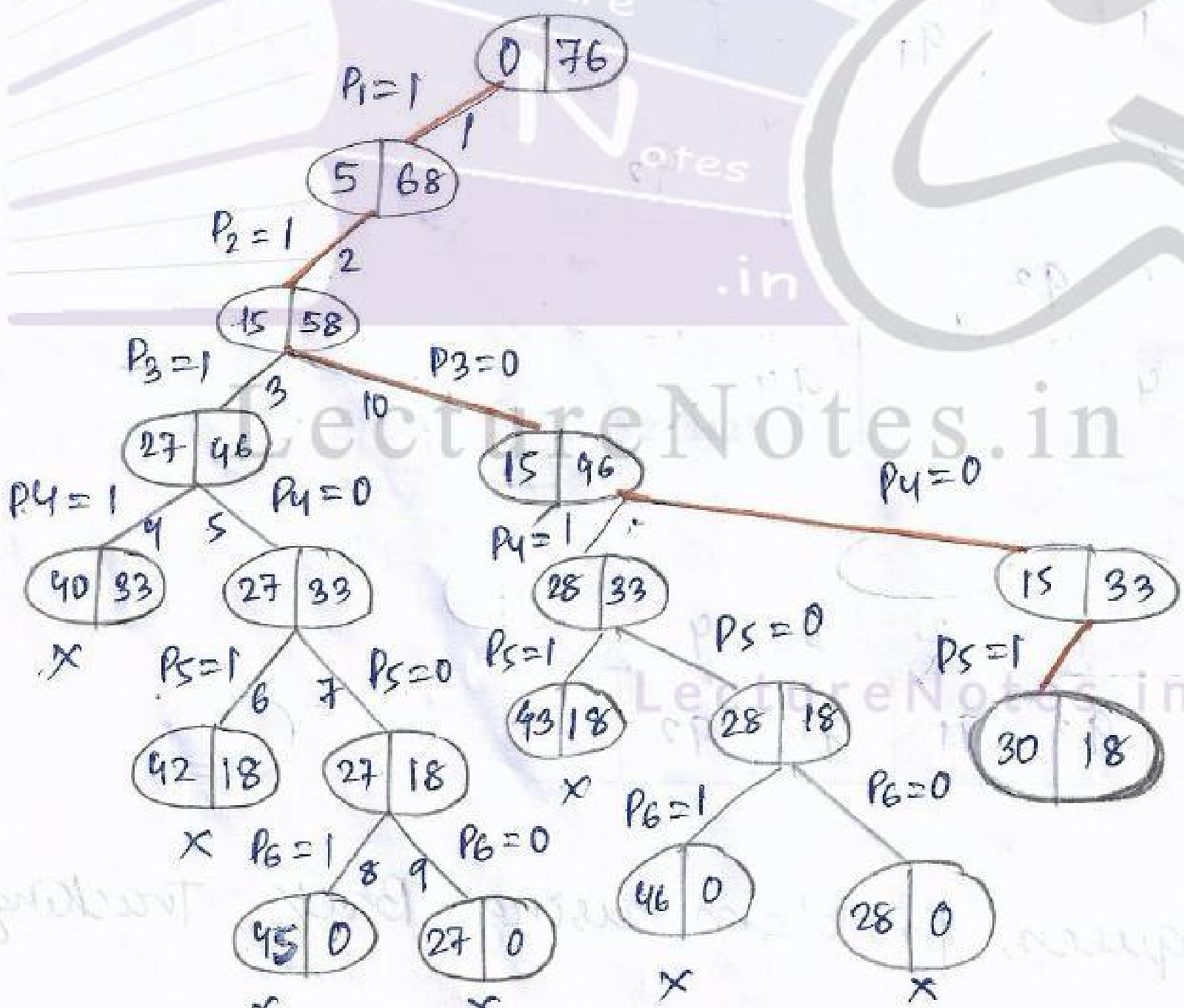
Sol. Subset Sum Problem using Back Tracking →

$$\begin{matrix} & P1 & P2 & P3 & P4 & P5 & P6 \\ w[1,6] = \{ & 5, & 10, & 12, & 13, & 15, & 18 \} \end{matrix}$$

Total weight = 73

M = 30

We have to take subset of the set such that their sum is exactly (M = 30).



	P_1	P_2	P_3	P_4	P_5	P_6
P	1	1	0	0	1	0

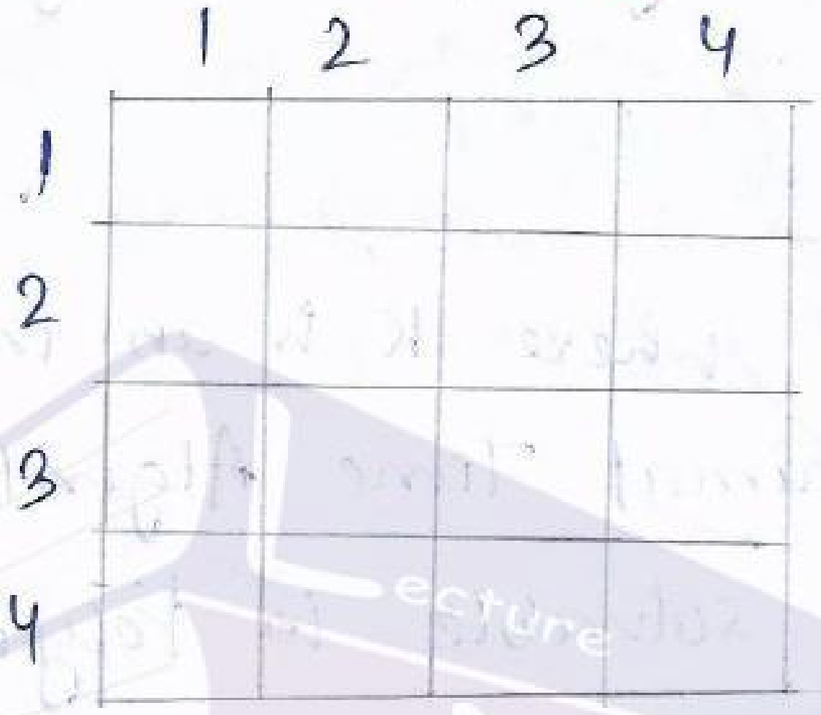
$$\sum_{i=1}^{6} P_i w_i = 30$$

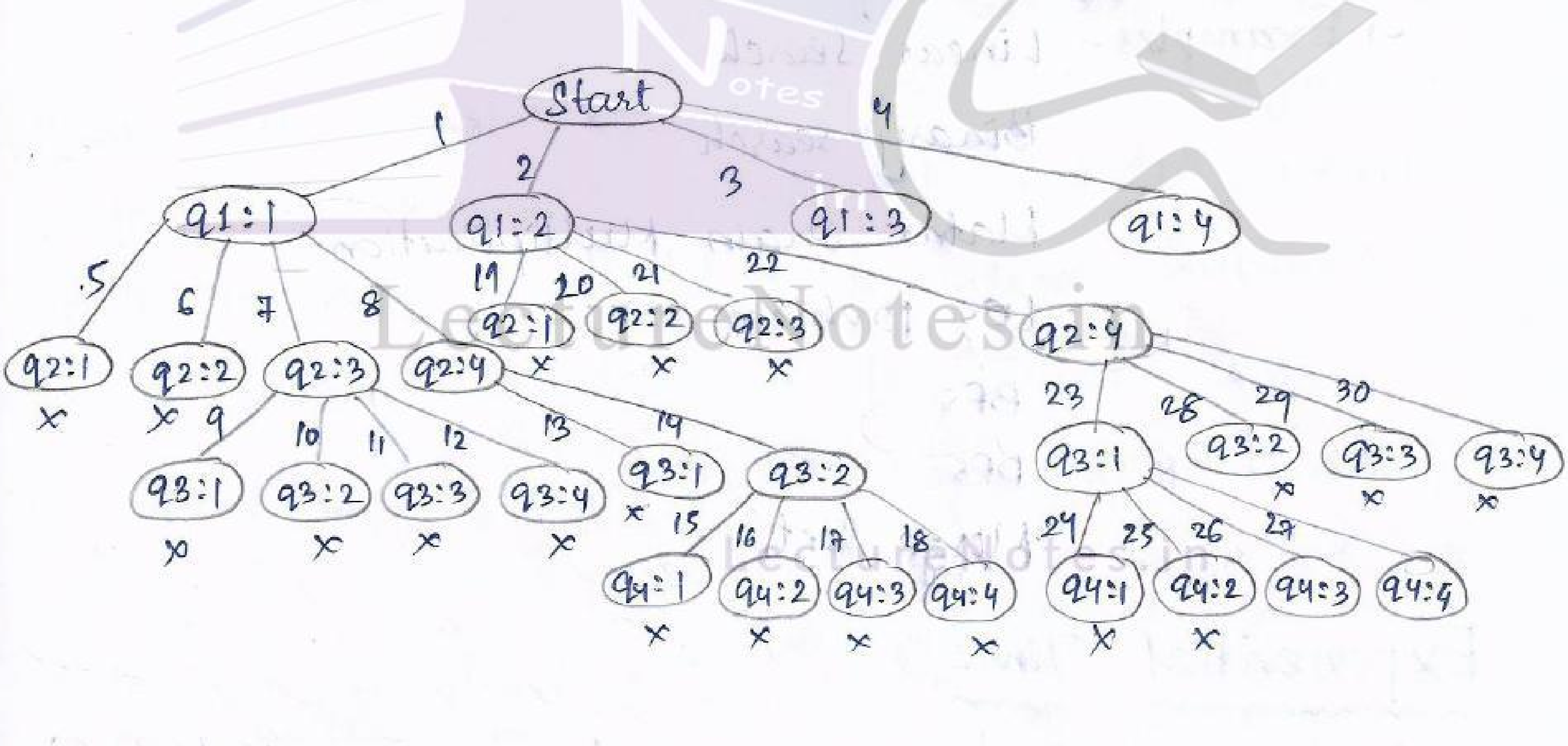
96

Branch and Baund Technique

* It explains the trees in BFS.
* This technique can be used to solve optimization problem (Minimization Only)

4-Queen Problem using Branch & Baund Technique →





Dt-14th Apr 18
Saturday

P, NP, NP hard, NP Complete

$2^n > n^k$

Polynomial Time

→ The time complexity of an algorithm is

$$T(n) = O(n^k)$$

where K is an integer

is called Polynomial Time Algorithm.

→ The Algorithm is solvable in Polynomial time.

→ **Examples** - Linear Search
Binary Search
Matrix Chain Multiplication
LCS Problem
BFS
DFS
Merge Sort

Exponential Time

The time complexity for minimal time is

$$T(n) = \Omega(a^n)$$

LectureNotes.in

Problems

Problems are of 2 types

1. Decidable → Deterministic
   → Non-Deterministic
2. Non-Decidable

Decision Table →

→ A Decision Table is one with answer Yes or No.

→ Example- Does a graph G have Minimum Spanning Tree (MST) of weight ≤ W

MST ≤ W

P Class Problems

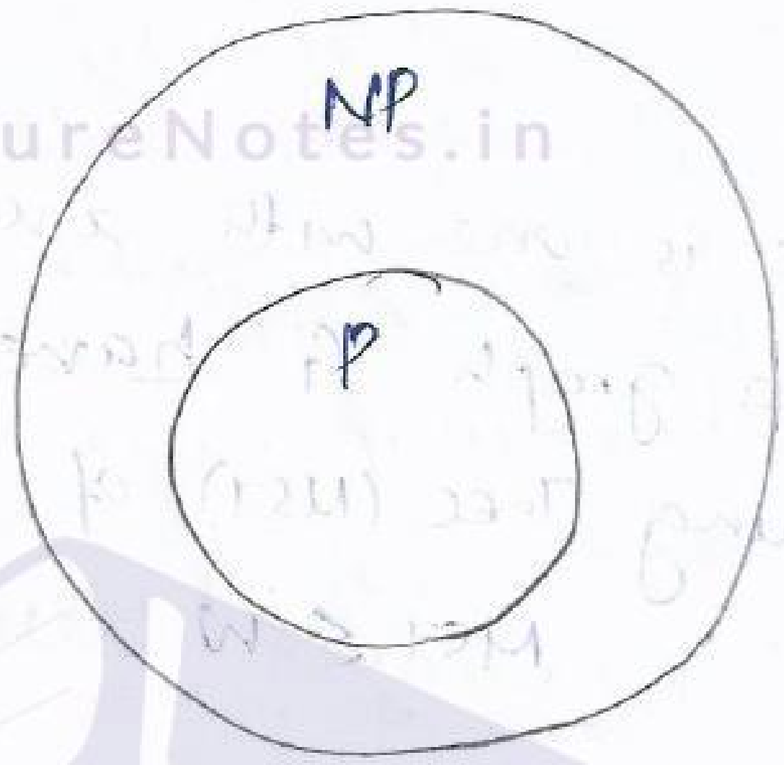
* The set of Decision Problems are said to be in Class P if there exist a Deterministic Algorithm which is solvable in Polynomial Time (n^k, where n is the input size and k is integer constant)
  * P class stands for Polynomial class.
  * Example- Linear Search, Binary Search, MCM, LCS problem, BFS, DFS, Merge Sort, etc.

NP Class Problems

* NP stands for Non-Deterministic Polynomial Problems.
* The set of Decision Problems are said to be in Class NP if there exist a Non-Deterministic

Algorithm in Polynomial Time

* If any decision problem is verifiable in Polynomial time, then the problem belongs to the class NP.





* P is a sub-set of NP

LectureNotes.in

Satisfiability

DL→ 16th Apr 18
Monday

$x_i = \{x_1, x_2, x_3\}$

$CNF = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) = 1$

(Conjunction Normal Form)

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
1	0	0
1	0	1
1	1	0
1	1	1
0	1	1

2^3

CNF − a^n

L1
L2
L3
L4
L5

LectureNotes.in

CNF $\propto$ L_2

CNF $\propto$ 0/1 Knapsack

NP hard NP hard

0/1 Knapsack problem is reduced from CNF in Polynomial time.

If one is NP hard, then the another is also NP hard.

Satisfiability →

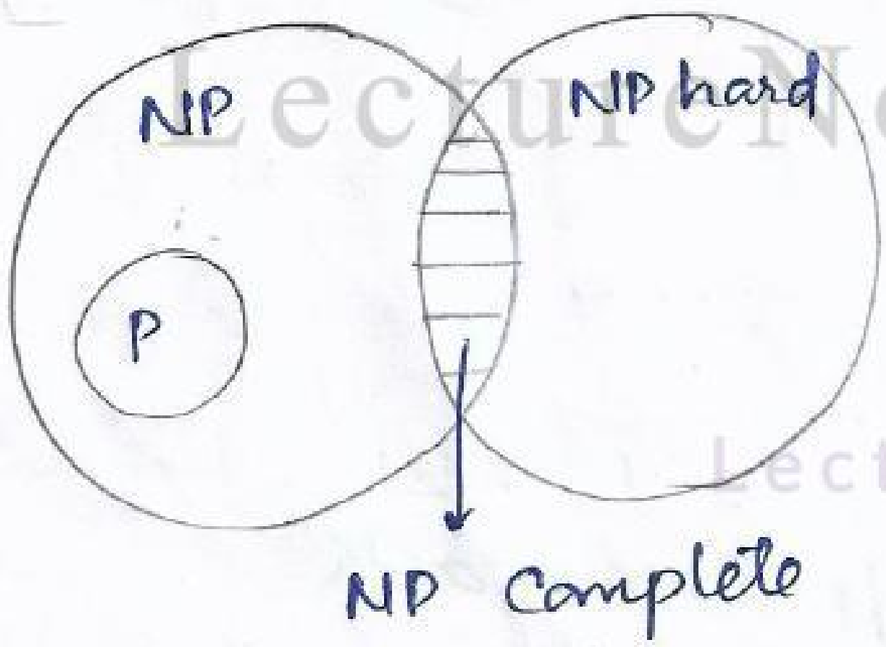
For a Boolean variable, the formula should result 1.

NP Hard Problem

A problem, L is said to be NP Hard if it is reduced from other NP Hard Problem in a Polynomial time.



where P → Polynomial Time



NP Complete →

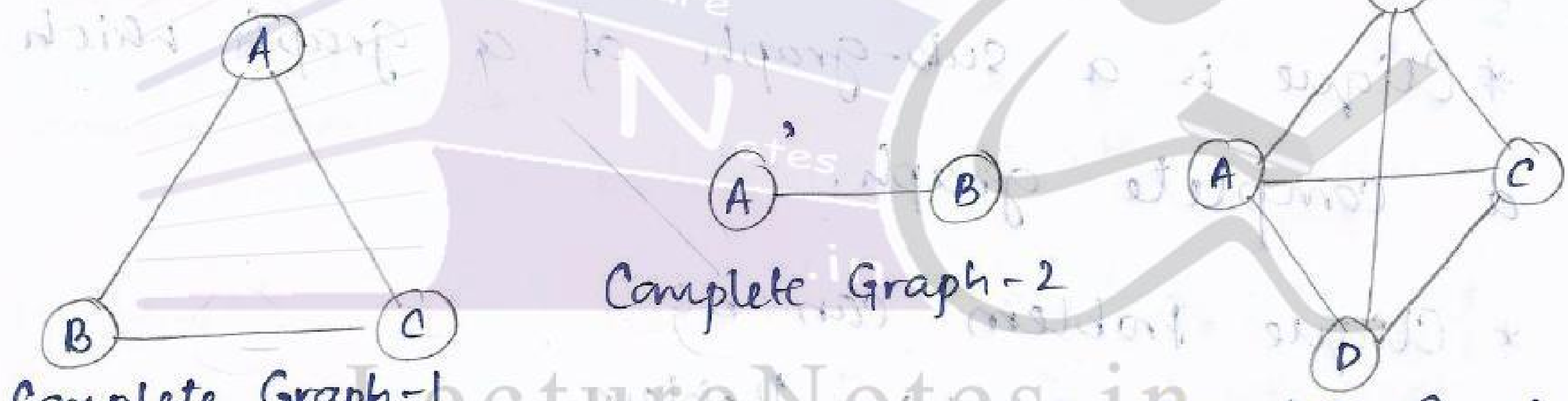
A problem L is said to be NP Complete if the problem is NP hard as well as NP. (We will have Non-Deterministic Algorithm.

in Polynomial Time)

Some NP Hard Problems -

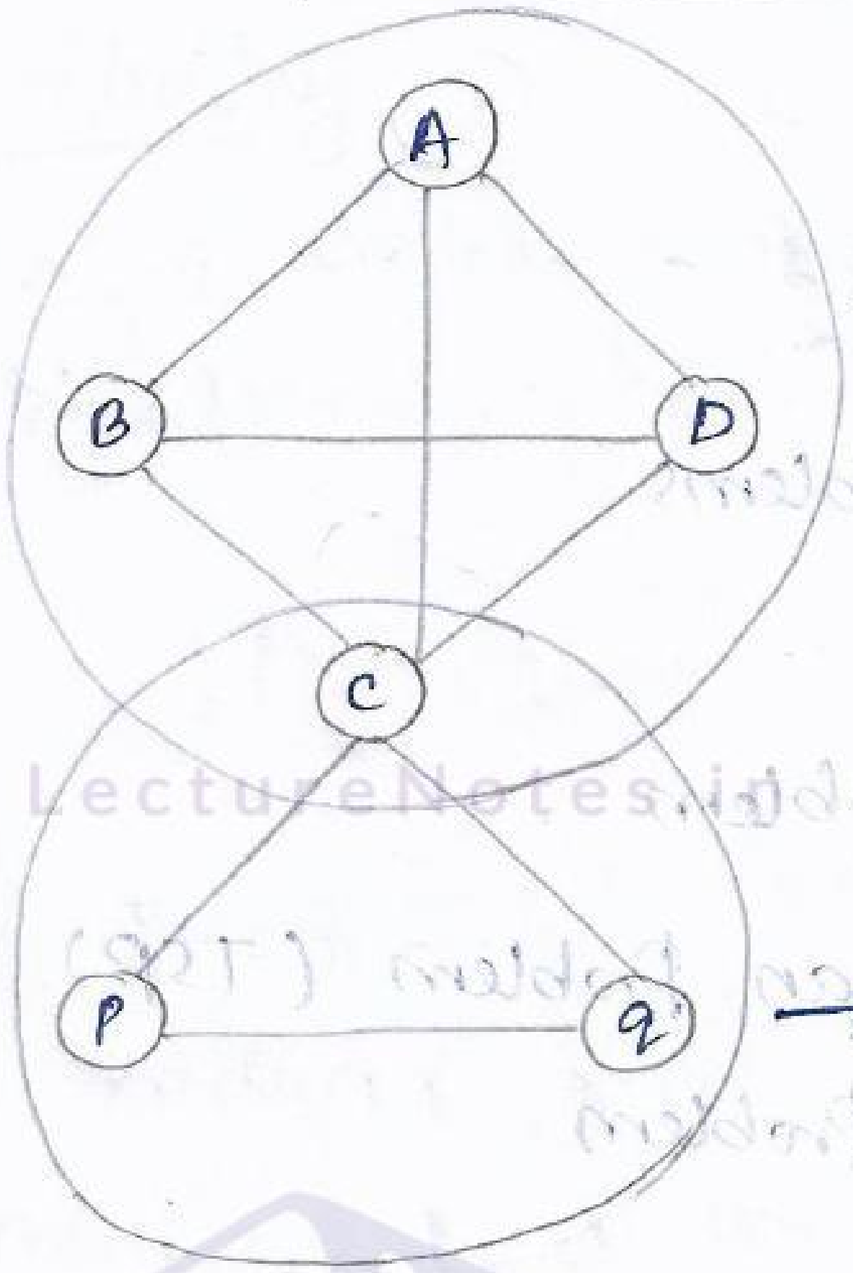
1. Subset Sum Problems
2. Clique Problem
3. Vertex Cover Problem
4. Traveling Salesman Problem (TSP)
5. Graph Coloring Problem

2. Clique Problem →



Complete Graph -

It is an undirected graph G if we have an edge from each vertex to all other vertex.



← Complete Graph
{A, B, C, D}
Clique of size = 4

→ Clique of size = 3

Clique :-

* Clique is a sub-graph of a graph which is a complete graph.

* Clique Problem can be
  1. Decision Problem
  2. Optimization Problem

1. Decision Problem →

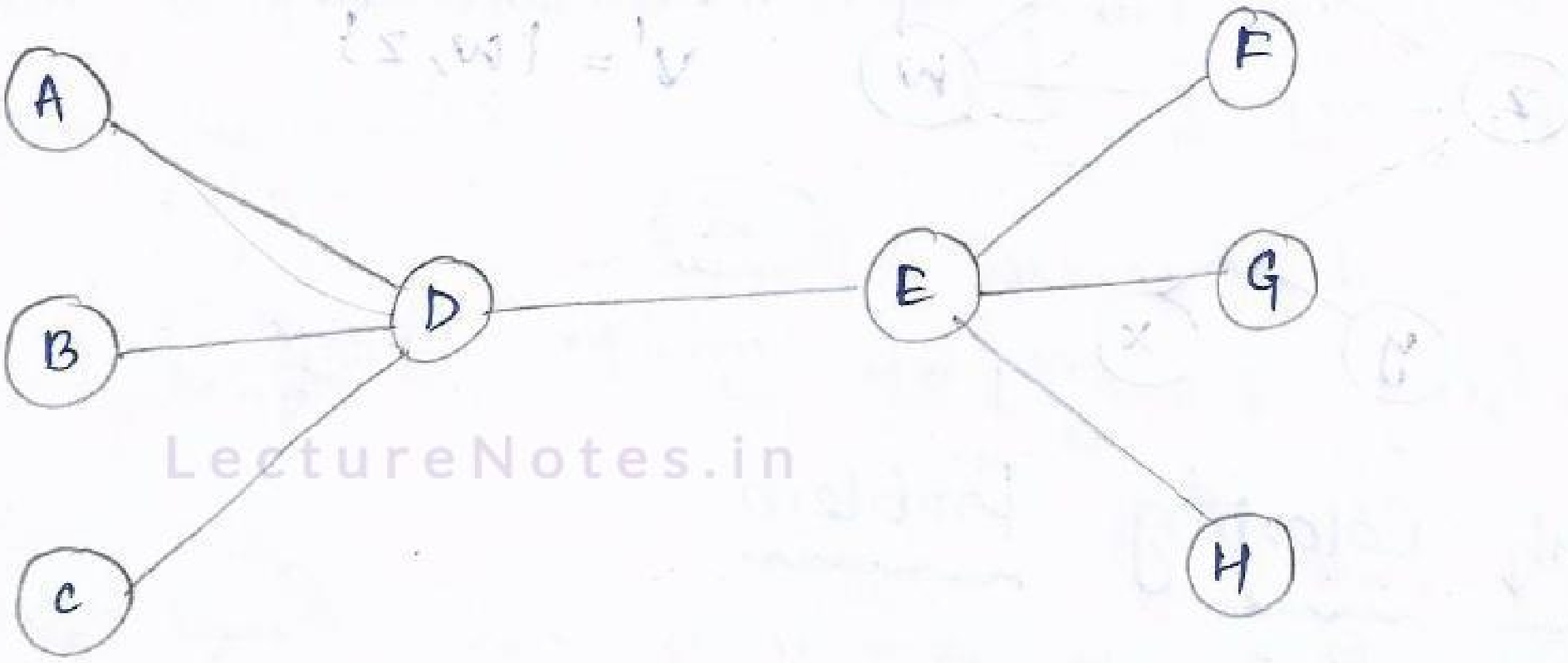
If the graph contains Clique, it is called Decision Problem.

2. Optimization Problem →

To find the maximum clique of a graph.

LectureNotes.in

3. Vertex Cover Problem



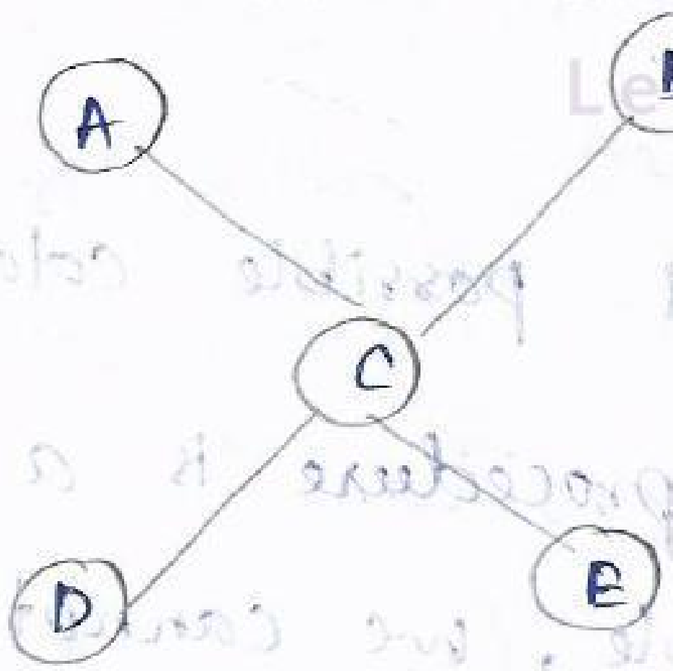
$V = \{A, B, C, D, E, F, G, H\}$ $V' = \{D, E\}$

The edges are $E = \{(A,D), (B,D), (C,D), (D,E), (E,F), (E,G), (E,H)\}$

Vertex Cover -

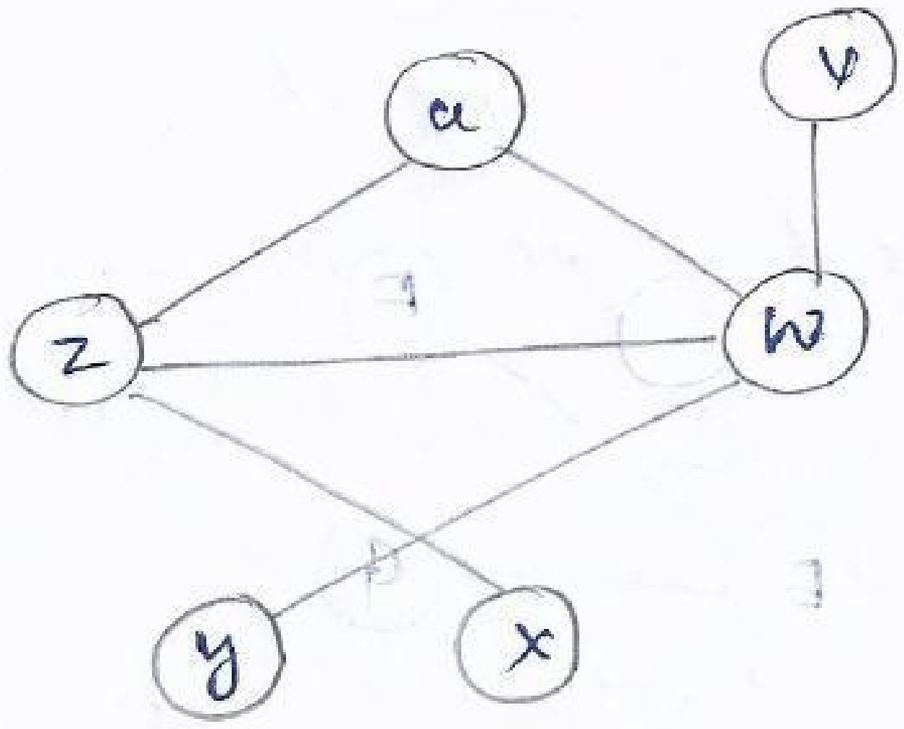
Given an undirected graph, $G = (V, E)$.

A subset $V' \subseteq V$ is called Vertex Cover of G for every edge $e \in E$, e has at least 1 end point incident at V'



$V' = \{C\}$

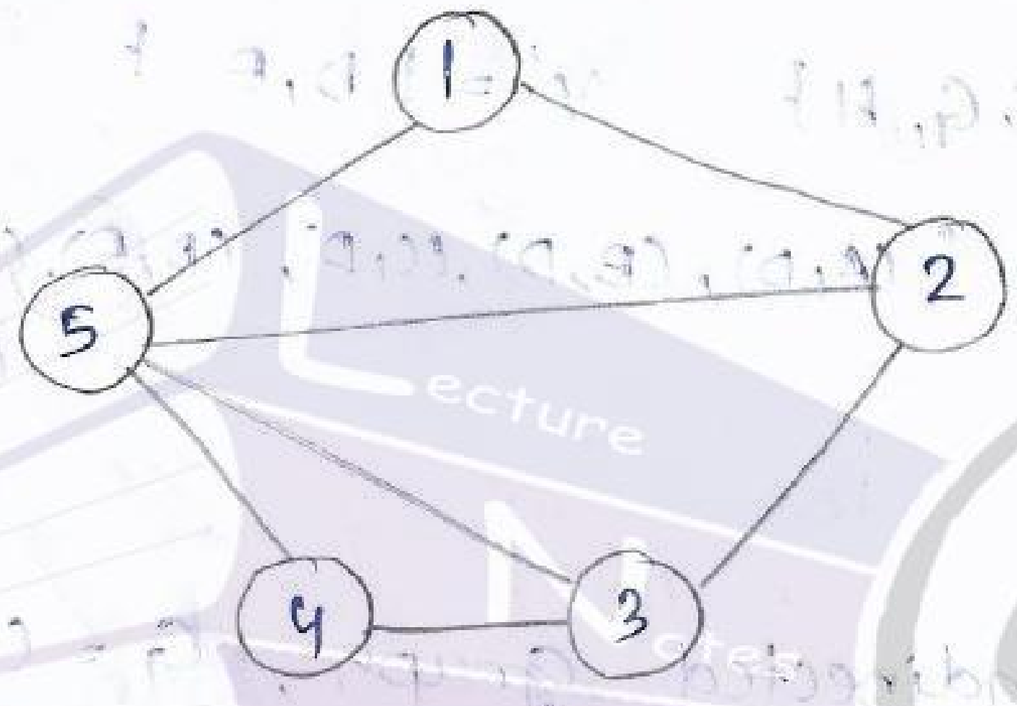
LectureNotes.in



$V' = \{w, z\}$

LectureNotes.in

5. Graph Coloring Problem



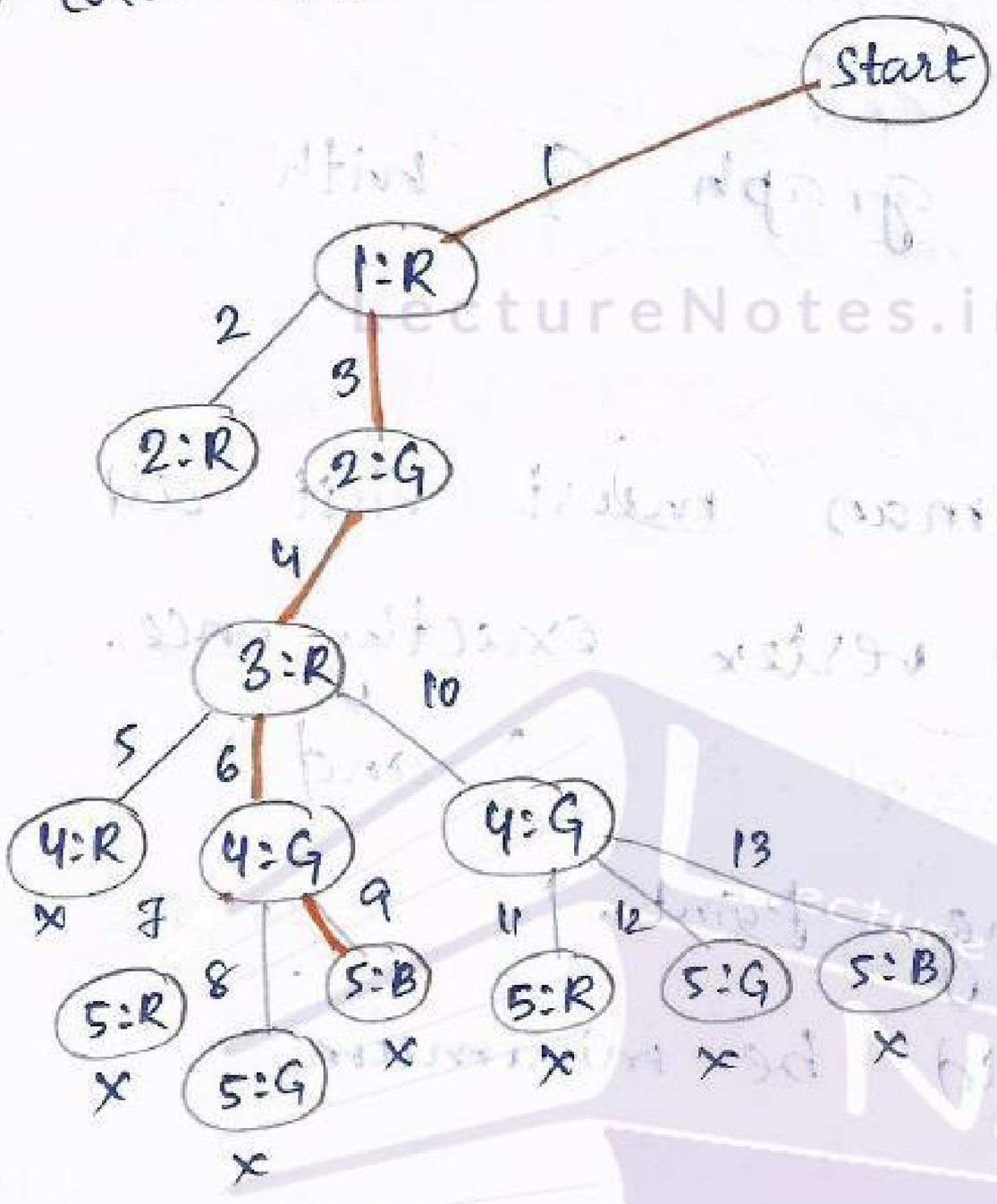
$M = 3 = \{R, G, B\}$

→ We have to color the vertex of the graph such that no two adjacent vertex will have same color.

→ We have to find all possible colors.

→ M graph coloring problem is a Decision Problem. (In above example, we cannot color the graph with 2 colors)

→ M graph coloring problem is an Optimization problem to find minimum number of colors required to color the vertex.



(Graph Coloring Problem using Back Tracking Problem)

Approximation Algorithm

An algorithm that returns near to optimal solution is called Approximation Algorithm.

Few Approximation Algorithms are -

1. Vector Coloring Algorithm
2. Traveling Salesman Problem

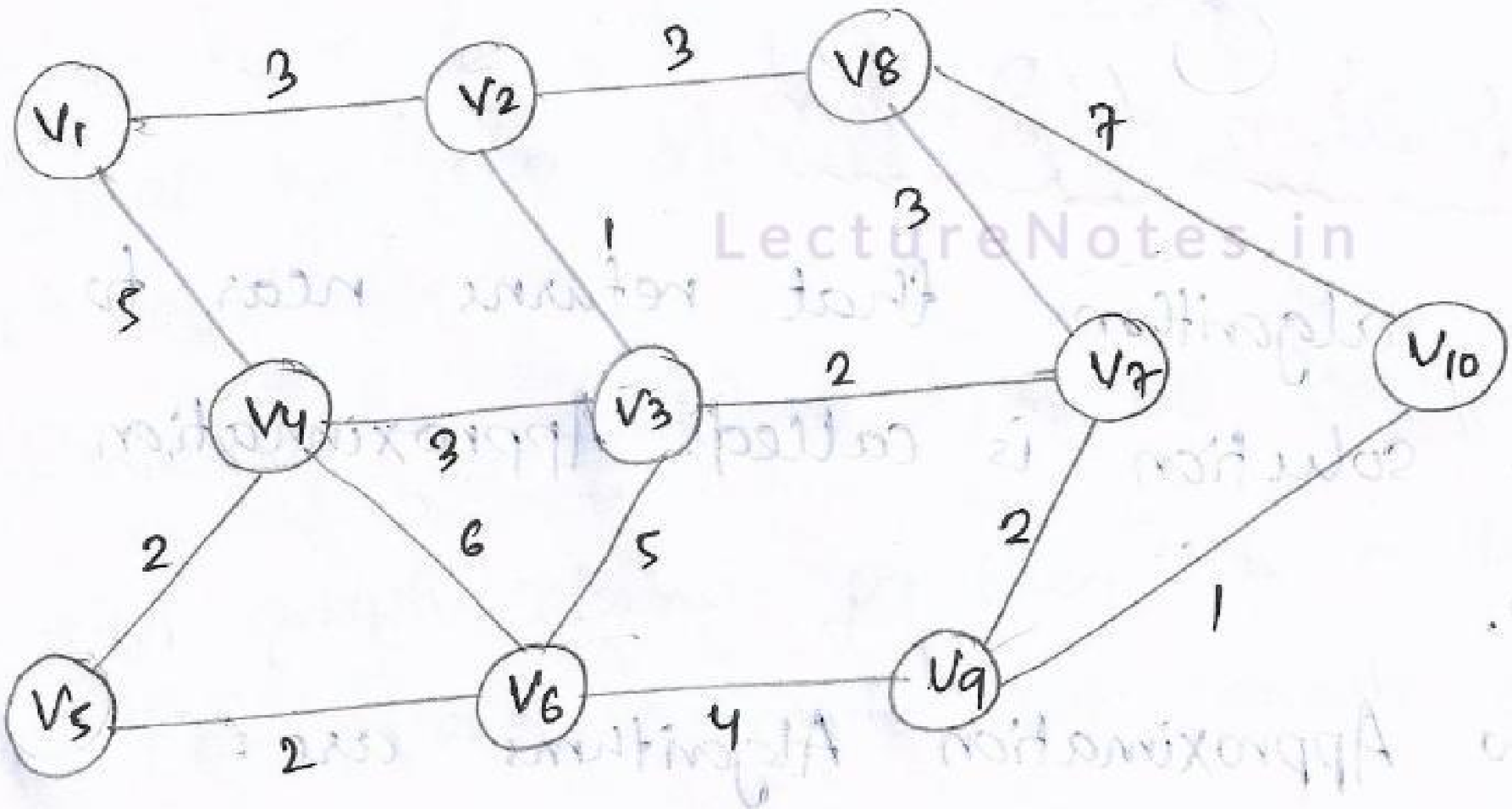
4. Traveling Salesman Problem

→ Given a complete graph G with non-negative edge.

→ In TSP, a salesman must visit all vertex (cities), visiting each vertex exactly once. (This is called Hamiltonial Cycle) and finish the city at starting pont.

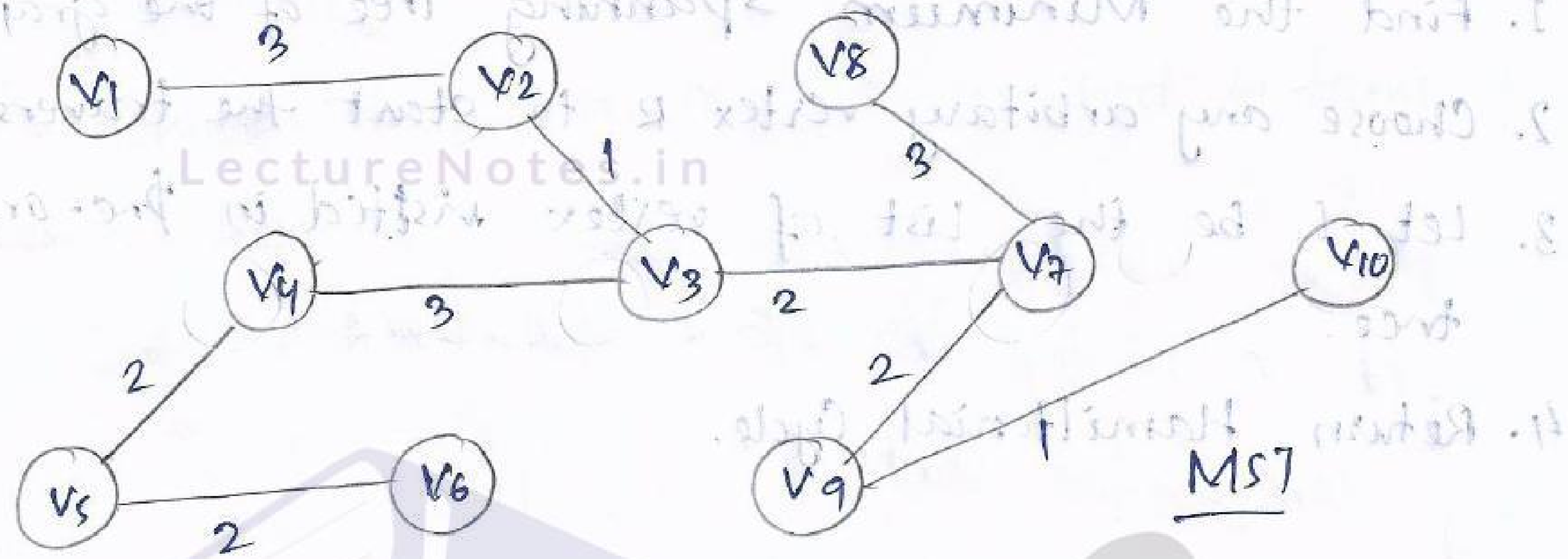
→ the total cost should be minimum.

Q. Solve TSP of following graph using TSP Approximation Algorithm.



Sol. Step 1

Using Prim's or Kruskal's Algorithm, find the Minimum Spanning Tree.

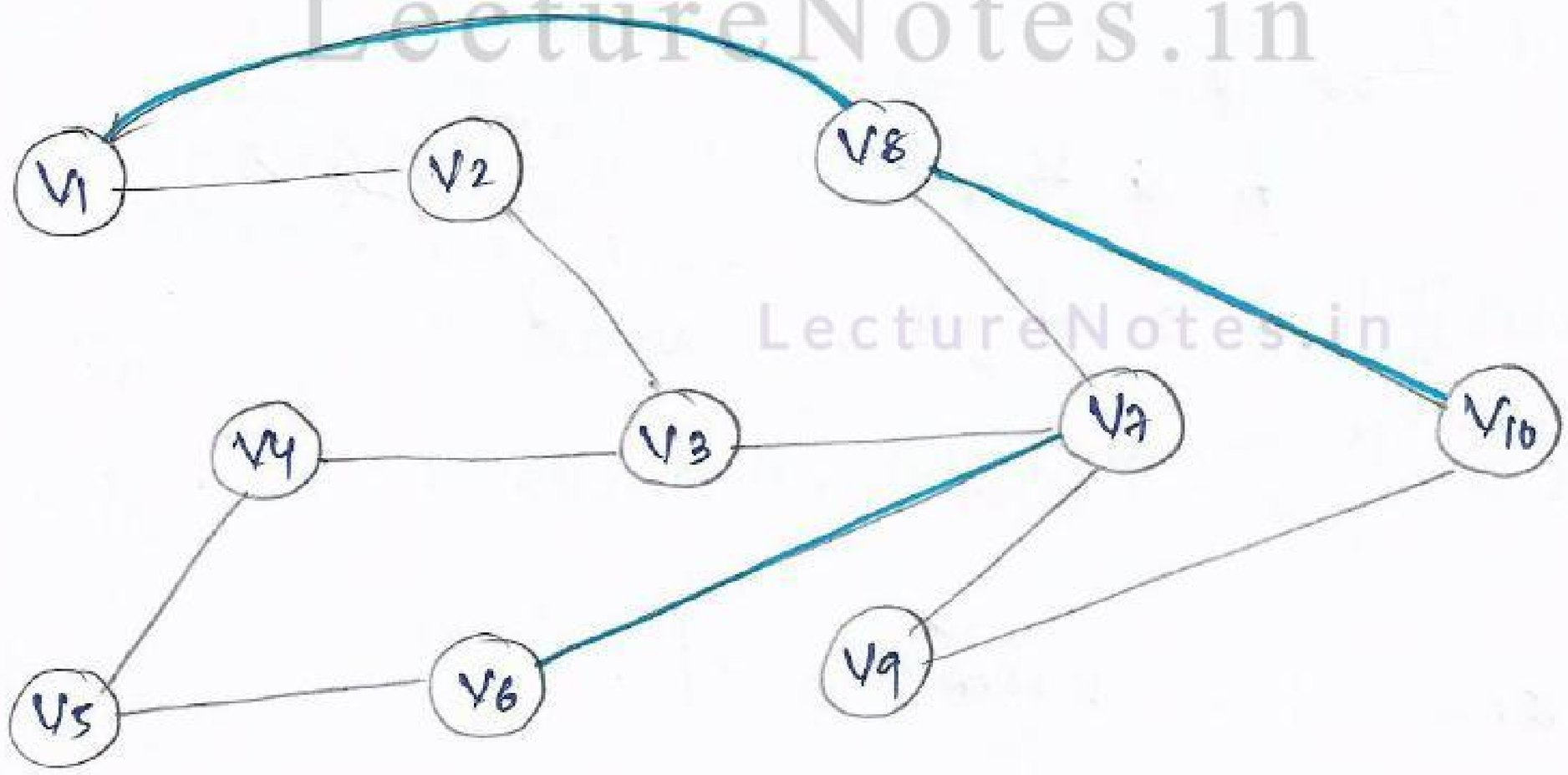


MST

Step 2

Find a cycle. Traverse a pre-order MST assuming the source vertex is V_1

$L = \{V_1 \to V_2 \to V_3 \to V_4 \to V_5 \to V_6 \to V_7 \to V_9 \to V_{10} \to V_8\}$



Algorithm (TSP using Approximation Algorithm)

APPOX-TSP-TOUR(G)

1. Find the Minimum Spanning Tree of the graph.
2. Choose any arbitary vertex R to start the traverse.
3. Let L be the list of vertex visited in Pre-order tree.
4. Return Hamiltonial Cycle.

LectureNotes.in

LectureNotes.in

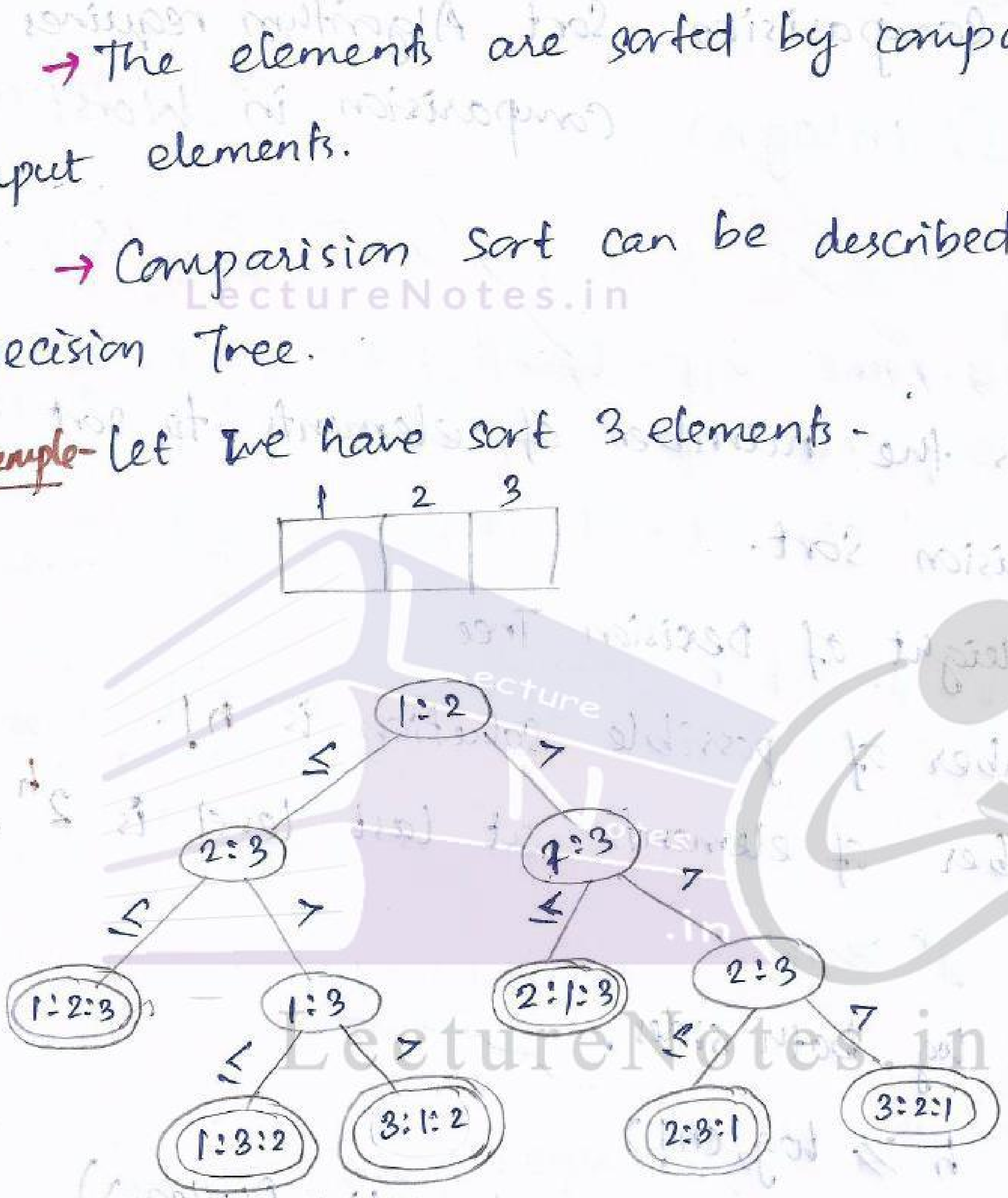
Comparision Sorts

Dt- 17th Apr 18
Tuesday

→ The elements are sorted by comparing between input elements.

→ Comparision sort can be described in terms of Decision Tree.

Example- Let we have sort 3 elements -



→ To sort 3 elements, we have 3 different solution.

→ To sort n elements, we require n! different solutions.

2nd Theorem (Lower-Band of Sorting)

Any Comparision Sort Algoritum requires $\Omega(n \log n)$ Comparision in Worst case.

LectureNotes.in

Proof -

Let n is the number of elements to sort using Comparision Sort.

h → Height of Decision Tree

The number of possible solution is $n!$.

The number of elements at last level is 2^h.

$$2^h \geq n!$$

Taking log both sides,

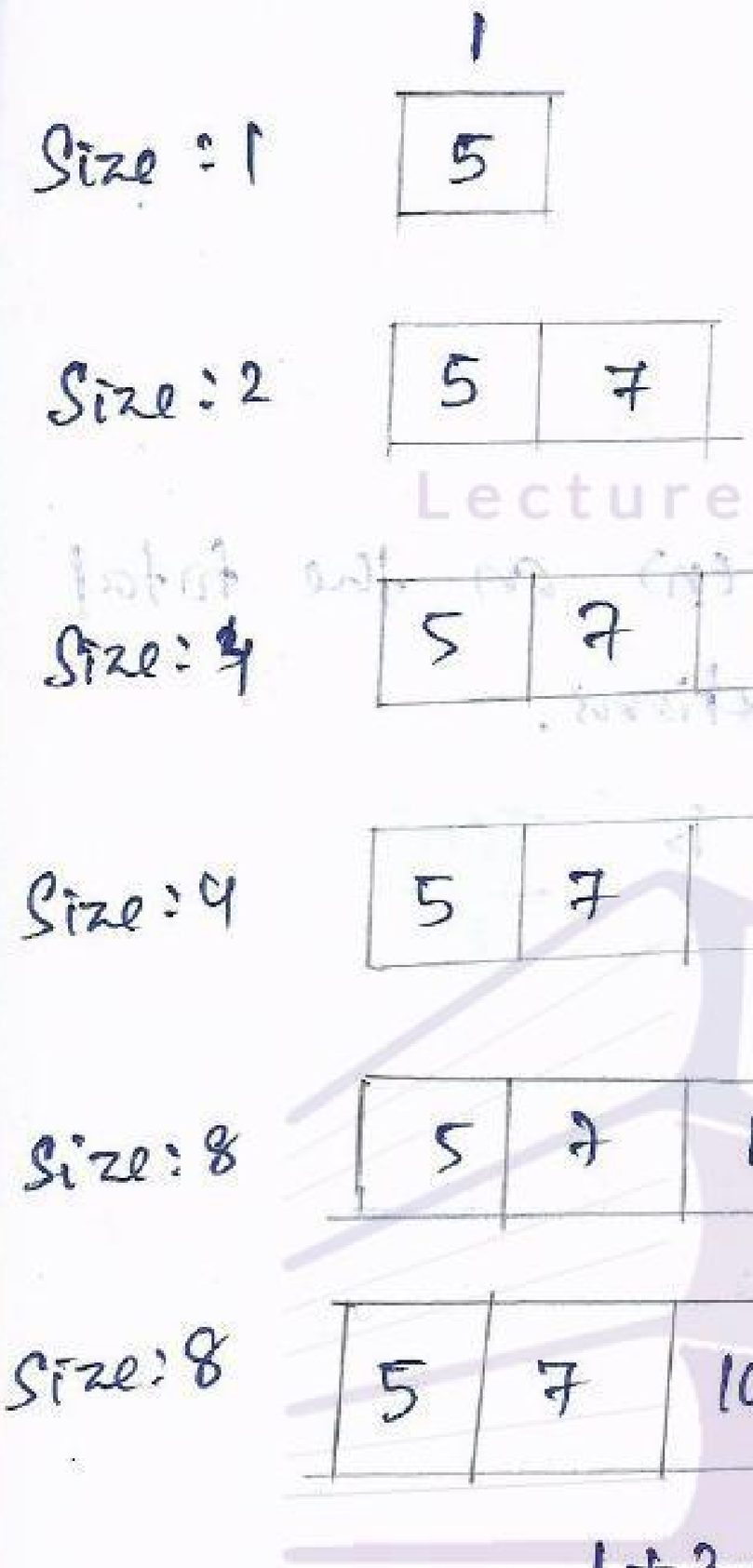
$$h \geq \log_2(n!)$$

$$h \geq n \log n \quad (\because \log(n!) = \theta(n \log n))$$

$$h = \Omega(n \log n)$$

Hence, the number of Comparision in Worst case is $\Omega(n \log n)$.

Amortized Analysis



LectureNotes.in

* The time required to perform a sequence of data structure operation is averaged over all the operations performed

* Amortized Analysis can be used to show that average cost of an operation is small.

1. Aggregation Method

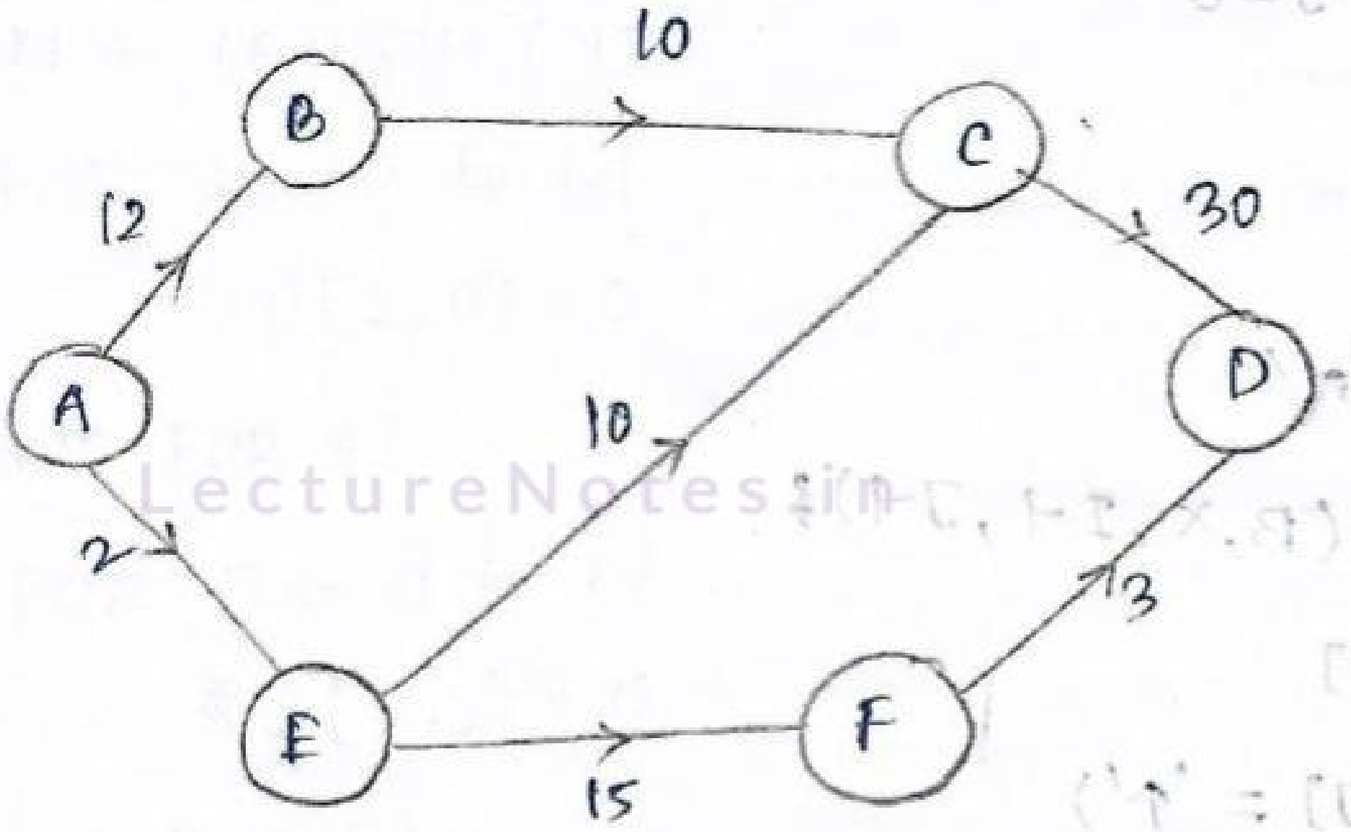
1. Aggregation Method →

* We determine upper bound T(n) on the total cost of a sequence of n operations.
* Then, the Amortized Cost is $\frac{T(n)}{n}$
* Example - Insert n elements

LectureNotes.in

Dt → 6th Mar 18
Tuesday

Greedy Algorithm →



$A \xrightarrow[2]{} E \xrightarrow[10]{} C \xrightarrow[30]{} D = 42$

* A Greedy Algorithm gets optimal solution to a problem by making sequence of Greedy choice (best in the present state)

* In the above diagram, the Greedy choice is $A \rightarrow E \rightarrow C \rightarrow D$. and the total cost is 42 which not minimum.

* So, Greedy Algorithm may not gurantee or give Optimal Solution for all problems.

LectureNotes.in

Dynamic Programming

* The choice in Dynamic Programming depends on the solution of problem.
* It always give Optimal Solution.
* The solution of the sub-problem is stored in a data structure.
* It is solved using Bottom-up Approach.
* The problems which can be solved by Dynamic Programming are-
  - Matrix Chain Multiplication
  - Longest Common Subsequence
  - Assembly line Scheduling
  - 0-1 Knapsack Problem.

Greedy Approach

* The choice in Greedy Algorithm is best at that moment which doesnot depend on future value.
* It may not give Optimal Solution.
* The solution of sub-problem is not stored in a data structure.
* It is solved by using Top-down Approach.
* The problems which can be solved by using Greedy Approach are-
  - Fractional Knapsack Problem.
  - Activity Selection Problem
  - Hoffmand Codes Problem.

LectureNotes.in

1. Fractional Knapsack Problem →

In the Knapsack Problem, a person wants to choose some items in a bag from the store such that the cost of the bag is maximum.

It is of 2 types -

1. Fractional Knapsack Problem
   (Fraction of the item can be chosed)
2. 0-1 Knapsack Problem
   (Either the complete item you have to choose or don't choose)

Solving Fractional Knapsack Problem →

Greedy Approach can be used to solve a Fractional Knapsack Problem.

Q. Let weight, $w = <40, 10, 30, 10, 40>$ and the corresponding cost, $v = <200, 60, 120, 10, 80>$.
The maximum weight we can carry in Knapsack is $w_K = 85$. Find the cost of the Knapsack using Fractional Knapsack Approach.

	1	2	3	4	5
Weight (w_i) kg	40	10	30	10	40
Cost (v_i)	200	60	120	10	80
cost/kg	5	6	4	1	2

Arranging the cost/kg in descending order,

	1	2	3	4	5
w_i	10	40	30	40	10
v_i	60	200	120	80	10
v_i/w_i	6	5	4	2	1

x_1	x_2	x_3	x_4	x_5
1	1	1	1/8	0

$\frac{1}{8} \times 80 = 10$ | $\frac{1}{8} \times 40 = 5$ | $w_k = 5 - 5 = 0$

120 | 30kg | $w_k = 35 - 30 = 5$

200 | 40kg | $w_k = 75 - 40 = 35$

60 | 10kg | $w_k = 85 - 10 = 75$

Knapsack

LectureNotes.in

The total cost is

$$x_1 v_1 + x_2 v_2 + x_3 v_3 + x_4 v_4 + x_5 v_5$$

$$= 1 \times 60 + 1 \times 200 + 1 \times 120 + \frac{1}{8} \times 80 + 0 \times 10$$

$$= 60 + 200 + 120 + 10 + 0$$

$$= ₹ 390$$

Dt- 10th Mar 18

The fractional Knapsack problem can be solved by using Greedy Algorithm but the Knapsack cannot be solved using Greedy Algorithm to get the optimal value.

Q. $w_i = \langle 10, 20, 30 \rangle$

$v_i = \langle 60, 100, 120 \rangle$

$w_u = 50kg$

	1	2	3
w_i	10	20	30
v_i	60	100	120
Cost	6	5	4

→

1	2	3
10	20	30
60	100	120
6	5	4

x_1	x_2	x_3
$\emptyset_1$	$\emptyset_1$	$\emptyset$-2/3

120	30	$w_K = 20-20 = 0$
100	20	$w_K = 40-20 = 20$
60	10	$w_K = 50-10 = 40$
	$w_K = 50$	

Total cost

$= x_1 v_1 + x_2 v_2 + x_3 v_3$

Algorithm for Fractional Knapsack Problem →

f_knapsack (w, v, n, w_k)

/* w is 1-D array represents weight of each item.
v represents cost of each item
n is number of items.
w_k is weight of knapsack */.

1. COMPUTE $\frac{v_i}{w_i}$ AND ARRANGE IN DESC. ORDER.
2. FOR I ← 1 to N
3. $X_I \leftarrow 0$
. [END FOR]
4. FOR I ← 1 to N
5. IF $W_I \leq W_K$
6. $X_I \leftarrow 1$
7. $W_K \leftarrow W_K - W_I$

ELSE

8. $X_I \leftarrow W_K / W_I$
9. $W_K \leftarrow 0$

[END IF 5]

[END FOR 4]

10. TOTAL_COST ← 0

LectureNotes.in

12. TOTAL_COST ← TOTAL_COST + $X_I \times V_I$

[END FOR]

13. RETURN TOTAL-COST

0-1 Knapsack Problem →

If Greedy Algorithm is used to solve 0-1 Knapsack problem, the cost of Knapsack is

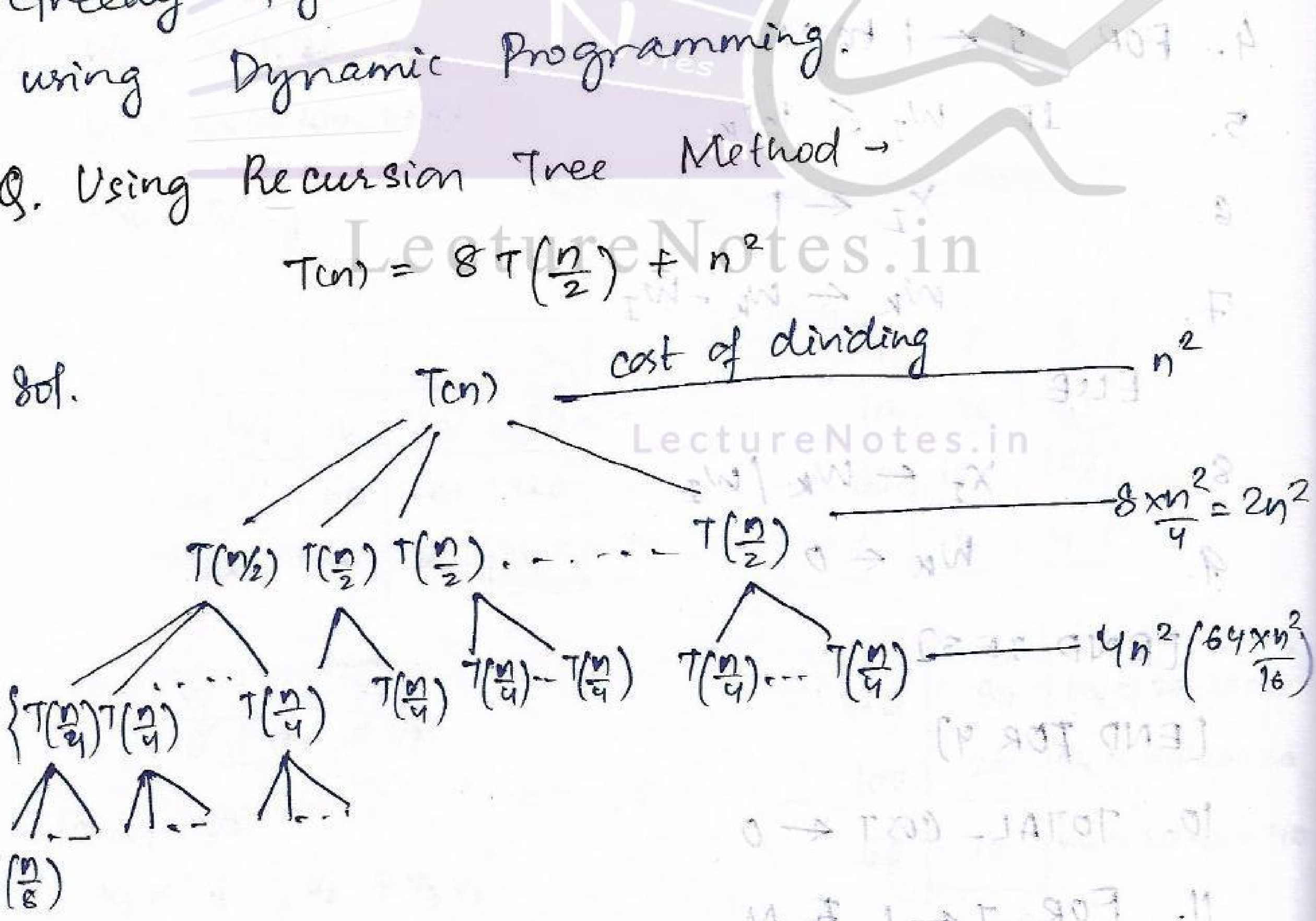
$$1 \times 60 + 1 \times 100 = 160$$

which is not an optimal value. Hence, 0-1 Knapsack problem can't be solved by using Greedy Algorithm but can be solved by using Dynamic Programming.

Q. Using Recursion Tree Method →

$$T(n) = 8T\left(\frac{n}{2}\right) + n^2$$

Sol.



$= n^2 + 2n^2 + 4n^2 + 8n^2 + \cdots + T(1) \times n^{\log_2^9} \longrightarrow n^3$

$= n^2 + 2n^2 + 4n^2 + 8n^2 + \cdots + n^3$

$= n^2 [1 + 2 + 4 + \cdots] + n^3$

$= n^2 \sum_{i=0}^{\log_2^n - 1} (2)^i + n^3$

$= n^2 \left(\frac{2^{\log_2^n} - 1}{2 - 1} \right) + n^3$

$= n^2 (2^{\log_2^n} - 1) + n^3$

$= n^3 - n^2 + n^3$

$= 2n^3 - n^2$

$\boxed{T(n) = O(n^3)}$

DE → 11th Mar 18

Activity Selection Problem

$S = \langle a_1, a_2, a_3, a_4, \cdots a_n \rangle$

$s = \langle s_1, s_2, s_3, s_4, \cdots s_n \rangle$

$f = \langle f_1, f_2, f_3, f_4, \cdots f_n \rangle$

Let set 'S' is a set of activities.

let s_I is the starting time of the activity a_I

Let F_I is the finish time of the activity a_I.

Activity a_I and a_J are comparable if the

$[s_I, f_I]$ and $[s_J, f_J]$

are not overlapped when $s_I < f_I$ and

$s_J < f_J$ and

$s_J \geq f_I$

The activity selection problem is a problem to select max number of compactable activities. It can be solved by using Greedy Algorithm.

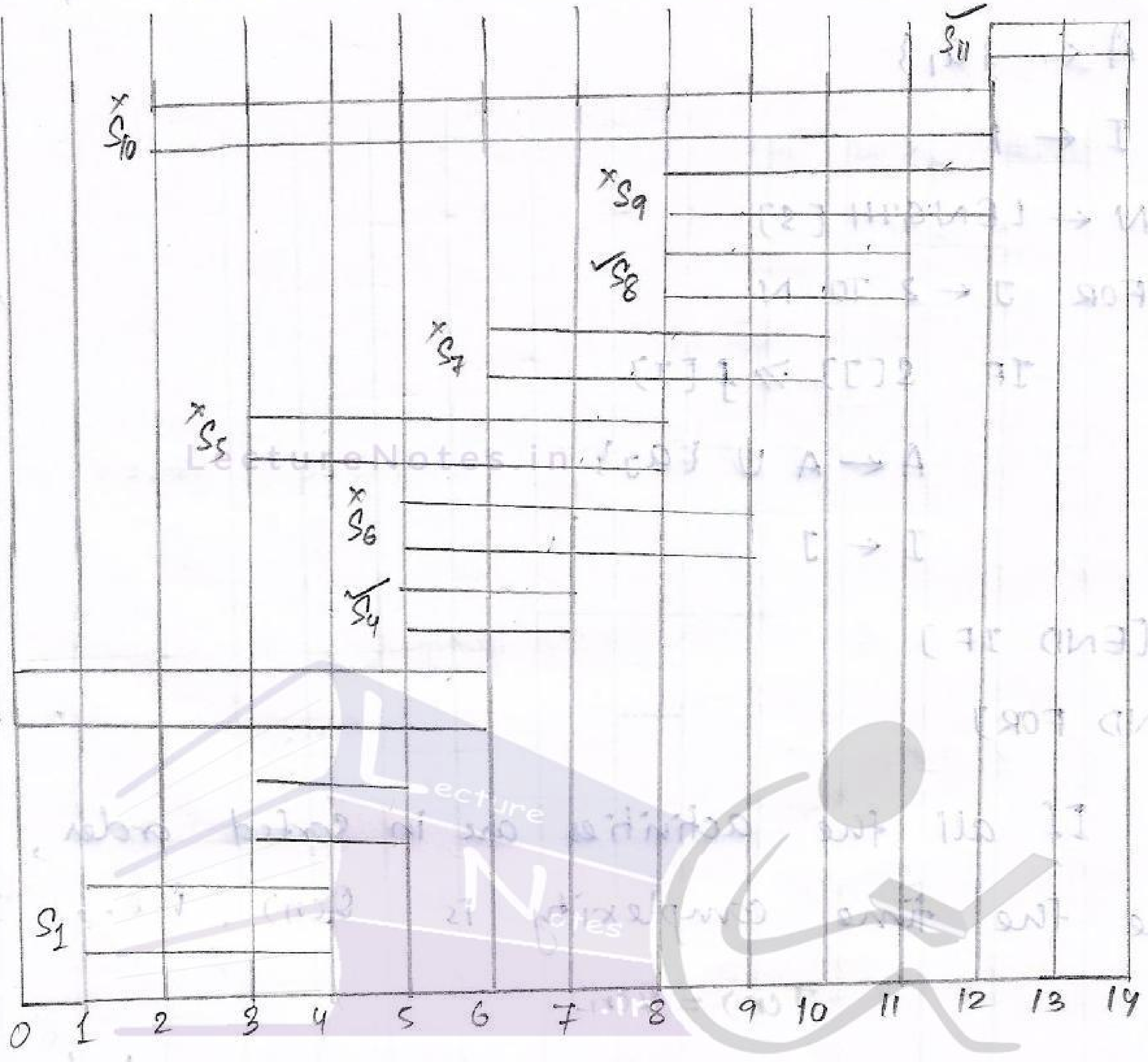
Q. Using Activity selection Algorithm, solve the following problem

$s_I = <1, 0, 3, 3, 5, 5, 6, 8, 8, 2, 12>$

$f_I = <4, 6, 5, 8, 7, 9, 10, 11, 12, 13, 14>$

Sol. Arranging all the activities in increasing order of finish time-

a_i	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	3	14



$$A = \langle S_1, S_4, S_8, S_{11} \rangle$$

$$a_1, a_4, a_8, a_{11}$$

Greedy Algorithm of Activity Selection Proble

<Using loop (Non-Recursive)>

```
GREEDY_ACTIVITY_SELECTOR(S,F)
  /* S is 1D array represents the start time
     F is 1D array represents final time of
```

1. ARRANGE ALL ACTIVITIES IN INCREASING ORDER
2. $A \leftarrow \{a_1\}$
3. $I \leftarrow 1$
4. $N \leftarrow$ LENGTH [S]
5. FOR $J \leftarrow 2$ TO N
6. IF $S[J] \geq f[I]$
7. $A \leftarrow A \cup \{a_J\}$
8. $I \leftarrow J$

[END IF]

[END FOR]

If all the activities are in sorted order, the the time complexity is $\Theta(n)$, i.e.,

$$T(n) = \Theta(n)$$

If all the activities are not in sorted form then the time complexity is $\Theta(n \log n)$

i.e.,

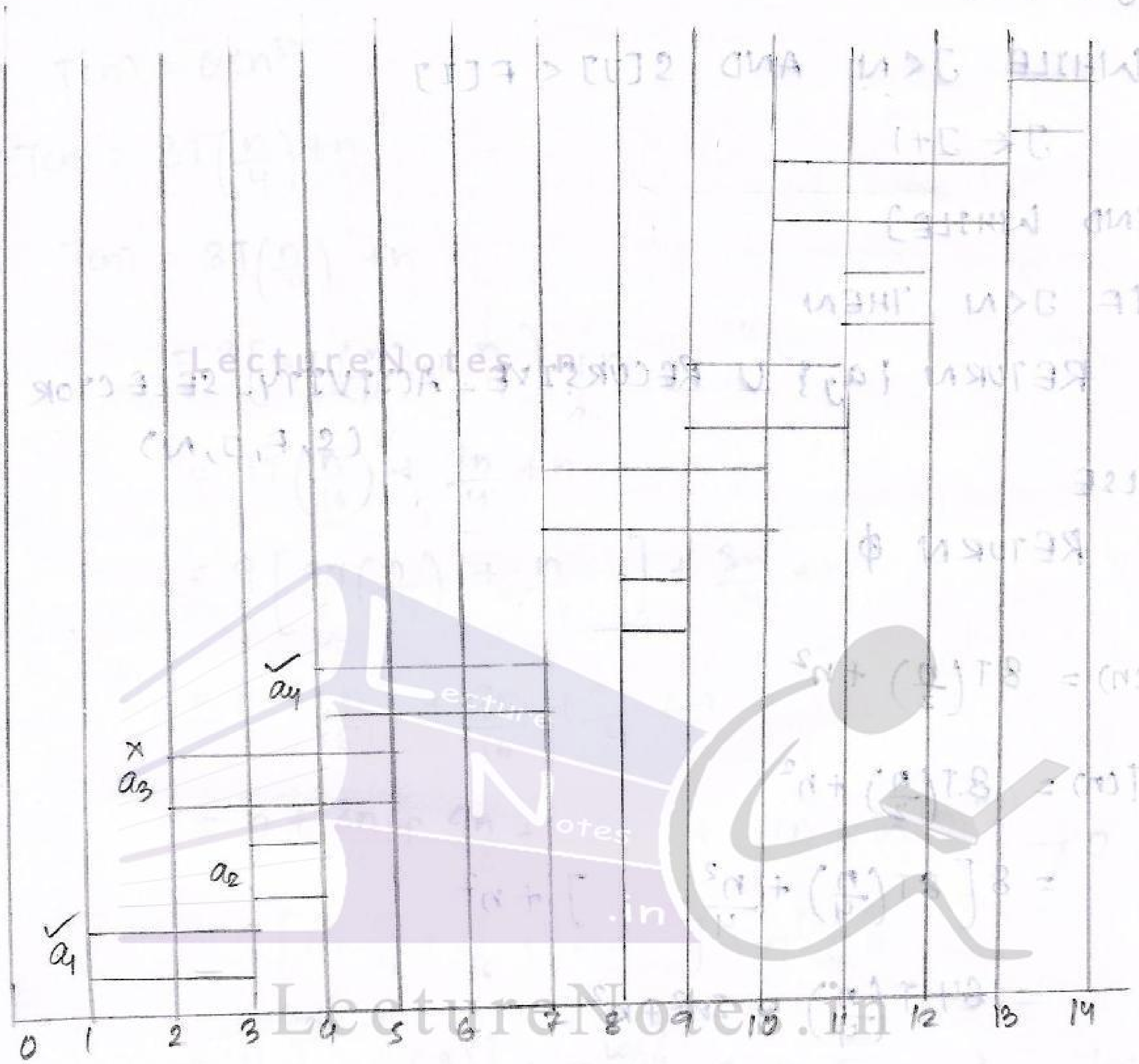
$$T(n) = \Theta(n \log n)$$

$$\Rightarrow T(n) = \Theta(n \log n) + \Theta(n)$$

Q. Given are 10 activities with start & finish time.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	8	9	10	11	12	13	14

$A = \langle a_1, a_2, a_4, a_5, a_7, a_8, a_{10} \rangle$



Recursive Algorithm of Activity Selection Problem

(Using Recursion)

```
RECURSIVE_ACTIVITY_SELECTOR (S, F, I, N)
/* S is a 1-D array represents start time
   F is the finish time of the activity (1-D array)
   N is the no. of activity */

/* RECURSIVE_ACTIVITY_SELECTOR (S, F, 0, N+1) */
```

1. ARRANGE ALL ACTIVITY IN INC. ORDER OF FINISH TIME.

2. J ← I+1

3. WHILE J<N AND S[J] < F[I]

4. J ← J+1

[END WHILE]

5. IF J<N THEN

6. RETURN {a_J} U RECURSIVE_ACTIVITY_SELECTOR (S, F, J, N)

ELSE

7. RETURN ϕ

8. $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

Sol. $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

$= 8\left[8T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right] + n^2$

$= 64\,T\left(\frac{n}{4}\right) + 2n^2 + n^2$

$= 64\left[8T\left(\frac{n}{8}\right) + \frac{n^2}{16}\right] + 2n^2 + n^2$

$= 512\,T\left(\frac{n}{8}\right) + 4n^2 + 2n^2 + n^2 + \cdots$

$= n^2 + 2n^2 + 4n^2 + \cdots + T(1) \times n^{\log_b^a} \rightarrow n^3$

$= n^2[1 + 2 + 4 + \cdots] + n^3$

$= n^2 \sum_{i=0}^{\log_2^n - 1} (2)^i + n^3 \quad \left[\because \sum_{i=0}^{n} (x)^i = \frac{x^{(n+1)} - 1}{x-1} \text{ where } x > 1\right]$

$= n^2 [2^{\log_2^n} - 1] + n^3$

$= n^2[n-1] + n^3$

$= 2n^3 - n^2$

$T(n) = \Theta(n^3)$

Q. $T(n) = 3T\left(\frac{n}{4}\right) + n$

Sol. $T(n) = 3T\left(\frac{n}{4}\right) + n$

$= 3\left[3T\left(\frac{n}{16}\right) + \frac{n}{4}\right] + n$

$= 9T\left(\frac{n}{16}\right) + \frac{3n}{4} + n$

$= 9\left[9T\left(\frac{n}{64}\right) + \frac{n}{16}\right] + \frac{3n}{4} + n$

$= 81T\left(\frac{n}{64}\right) + \frac{9n}{16} + \frac{3n}{4} + n$

$= n + \frac{3n}{4} + \frac{9n}{16} + \cdots + T(1) \times (n^{\log_b^a}) \to 0$

$= n\left[1 + \frac{3}{4} + \frac{9}{16} + \cdots\right] + n^{\log_b^a}$

$= n \sum_{i=0}^{\log_4^n - 1} \left(\frac{3}{4}\right)^i + n^{\log_4^3} \leq n \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + n^{\log_4^3}$

$\leq n \frac{1}{1-\frac{3}{4}} + n^{\log_4^3}$

$\leq 4n + n^{\log_4^3}$

$T(n) = O(n)$

LectureNotes.in

Dt → 13th Mar 18
Tuesday

Hoffman Code

Q.

abaab ....
a = 800
b: 200
} 1000 characters

Encoding →
a:1, b:0

Text : size : 1000byte

Sol.

10110 ....

size : 1000 bit

The size is reduced to 8 times.

Q.

abaabcc ....
a: 700
b: 250
c: 50
} 1000 characters

Encoding →
a: 00
b: 01
c: 10

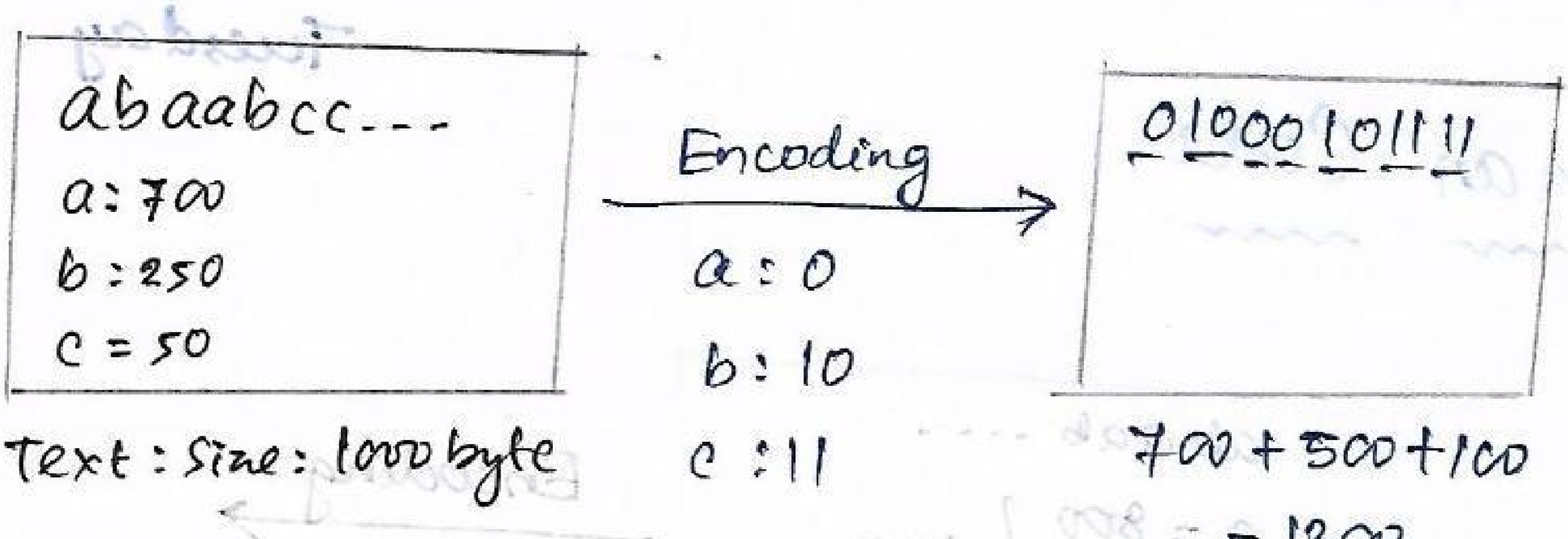
0001000011010...

size : 2000 bit

Text : size : 1000 byte

Q. If there are n different characters then the number of bits required to represent a character is what?

bit = $\lceil \log_2 n \rceil$



General Techniques to Encode the data →

The following two techniques can be used to encode the data -

1. Fixed length code
2. Variable length code

1. Fixed Length Code →

* Assigning same number of bits to represent a character.
* If there are n different characters, the size of the code word is equal to

$$\text{Size of code word} = \lceil \log_2 n \rceil$$

Q. If there are 33 different characters then the length of the code word is what?

Sol. $\lceil \log_2^{33} \rceil = 6$

2. Variable Length code →

* Assigning different number of bits to each character is called Variable Length Code Technique.

* If the character has more frequency, then it should have a code word of less length.

abaabcc...
a = 700
b = 250
c = 50

Text : Size : 1000 byte

Encoding →

a : 0
b : 01
c : 10

0 01 0 0 01 1010
a b a a b c c

We will get ambiguity while decoding.

Prefix Code →

The code where no code word is a prefix of some other code is called Prefix Code.

Example - 0, 101, 100, 111, 1101, 1100

LectureNotes.in

→ Hoffman code is a Prefix code.

→ Hoffman defined an algorithm to generate prefix code.

Dt → 16th Mar 18
Friday

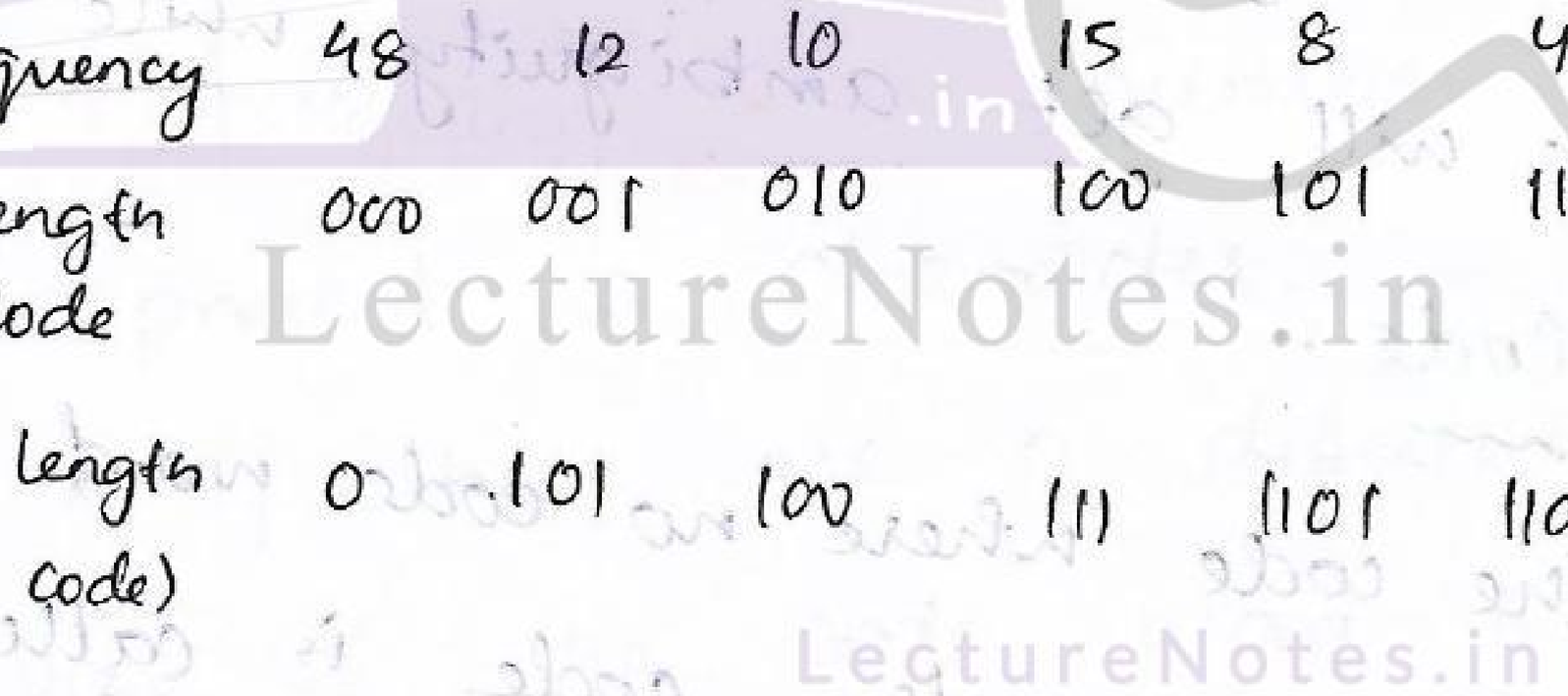
Hoffman Tree →

Hoffman tree is a binary tree in which '0' represents 'go to left' and '1' represents 'go to right'.

Q. Construct a Hoffman Tree corresponding to the following set of tree -

	a	b	c	d	e	f
Frequency	48	12	10	15	8	4
Fixed length code	000	001	010	100	101	110
Variable length code (Prefix code)	0	101	100	111	1101	1100

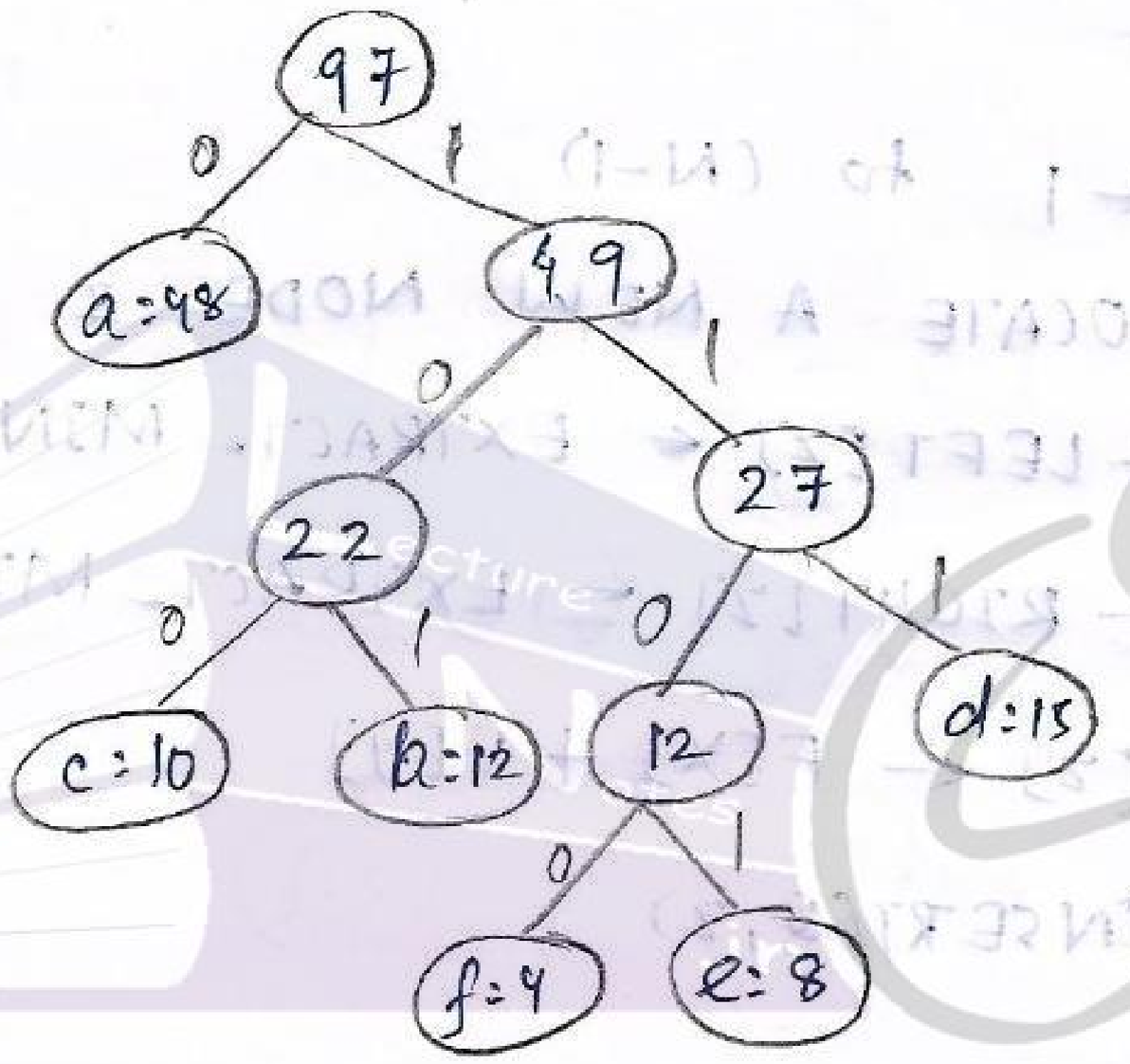
Sol. Hoffman Tree for Fixed length code →



→ The root-node represents the total number of code, i.e., 97.

→ All the characters are in child nodes and they are in same level because it is fixed length code.

Hoffman Tree for Variable length code →



→ All the characters are in child node but in different levels.

* Hoffmann Code Algorithm is used to generate a Hoffmann Tree.

Hoffmann Algorithm →

```
HOFFMANN CODE (C)
    /* C is different character set with
                                    frequency */
1.  N ← |C|
2.  BUILD_MIN_HEAP_TREE
3.  Q ← C
4.  FOR   I ← 1   to  (N-1)
5.          ALLOCATE A NEW NODE z
6.          X ← LEFT[z] ← EXTRACT_MIN(Q)
7.          Y ← RIGHT[z] ← EXTRACT_MIN(Q)
8.          F[z] ← F[X] + F[Y]
9.          INSERT(Q, z)
    [END FOR]
```

LectureNotes.in

Q. What is the Optimal Hoffmann Code for the following set of frequencies -

a:45, b:13, c:12, d:16, e:9, f:5

Sol. Arrange all characters in the ascending order of frequency and store in Q.

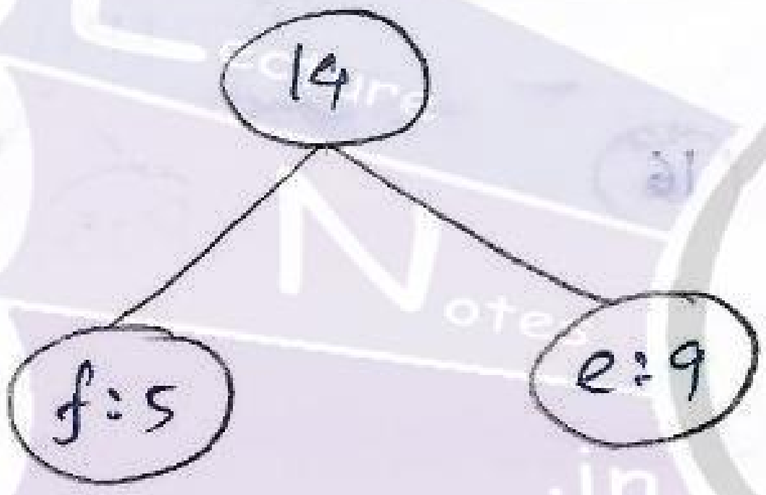
Q

f:5	e:9	c:12	b:13	d:16	a:45

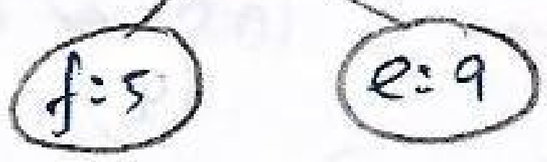
FRONT REAR

Step 1

$x \leftarrow f:5$, $y \leftarrow e:9$, $f[z] = 14$

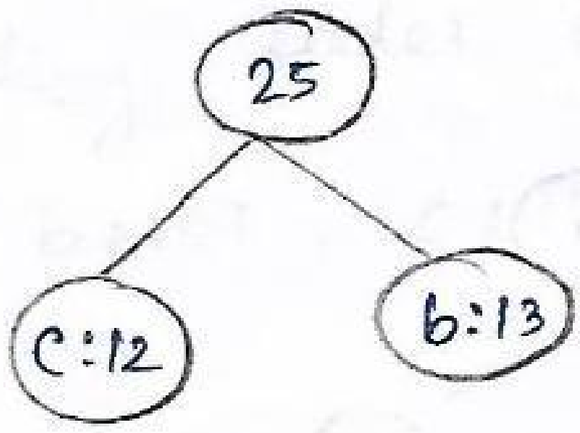


c:12	b:13	14	d:16	a:45



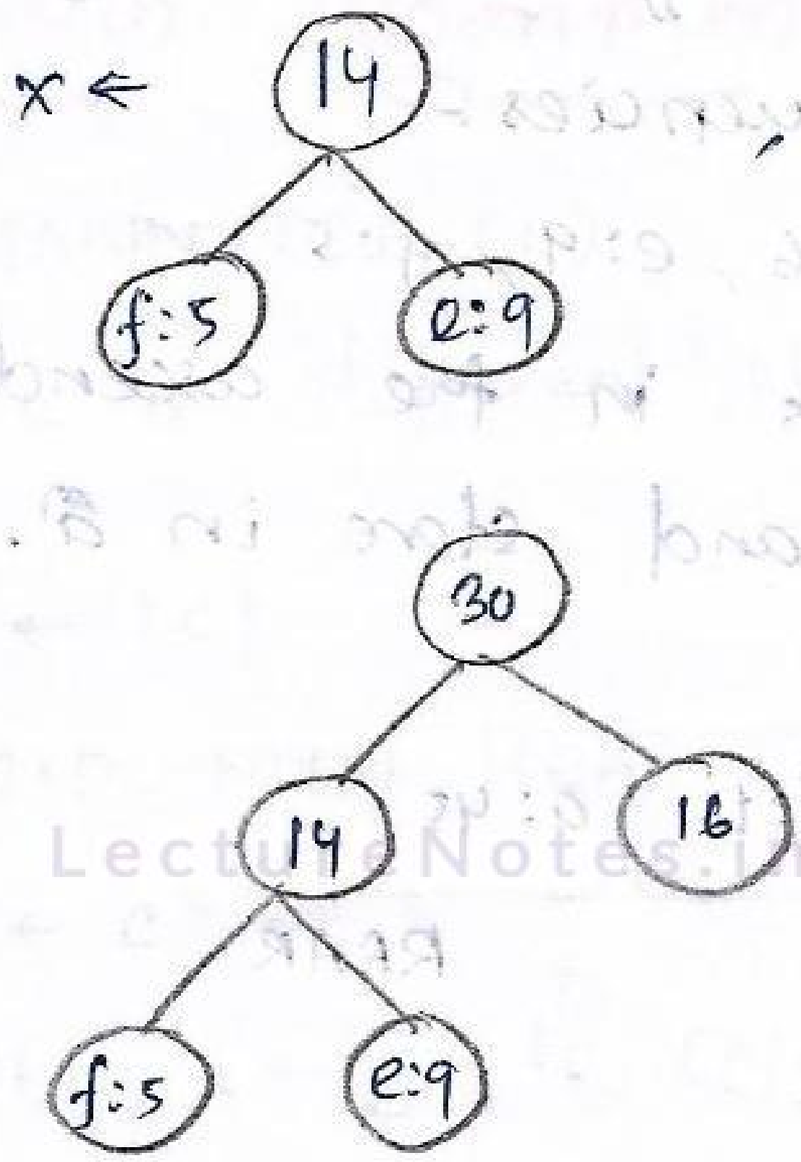
Step 2

$x \leftarrow c:12$, $y \leftarrow b:13$, $f[z] = 25$

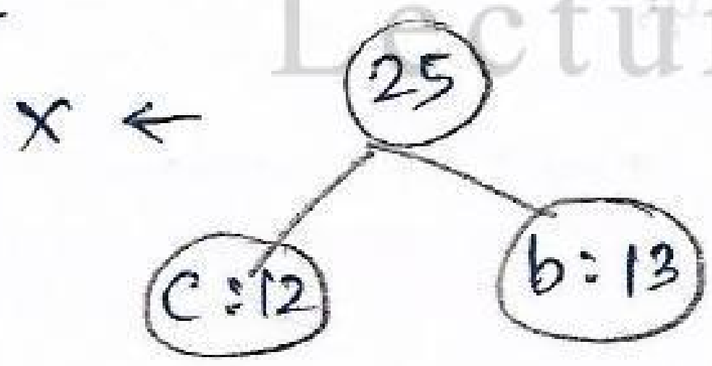


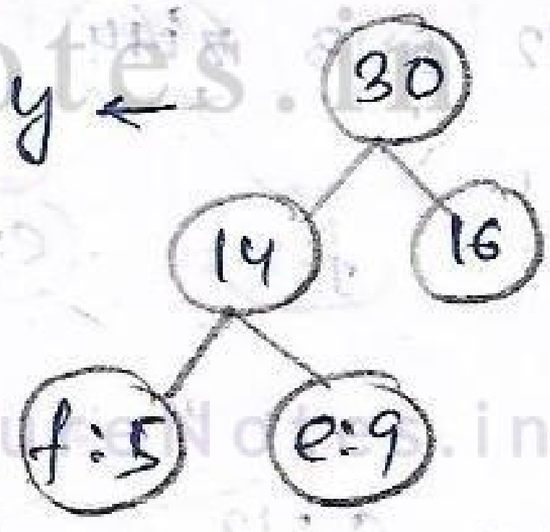
14	d:16	25	a:45

Step 3

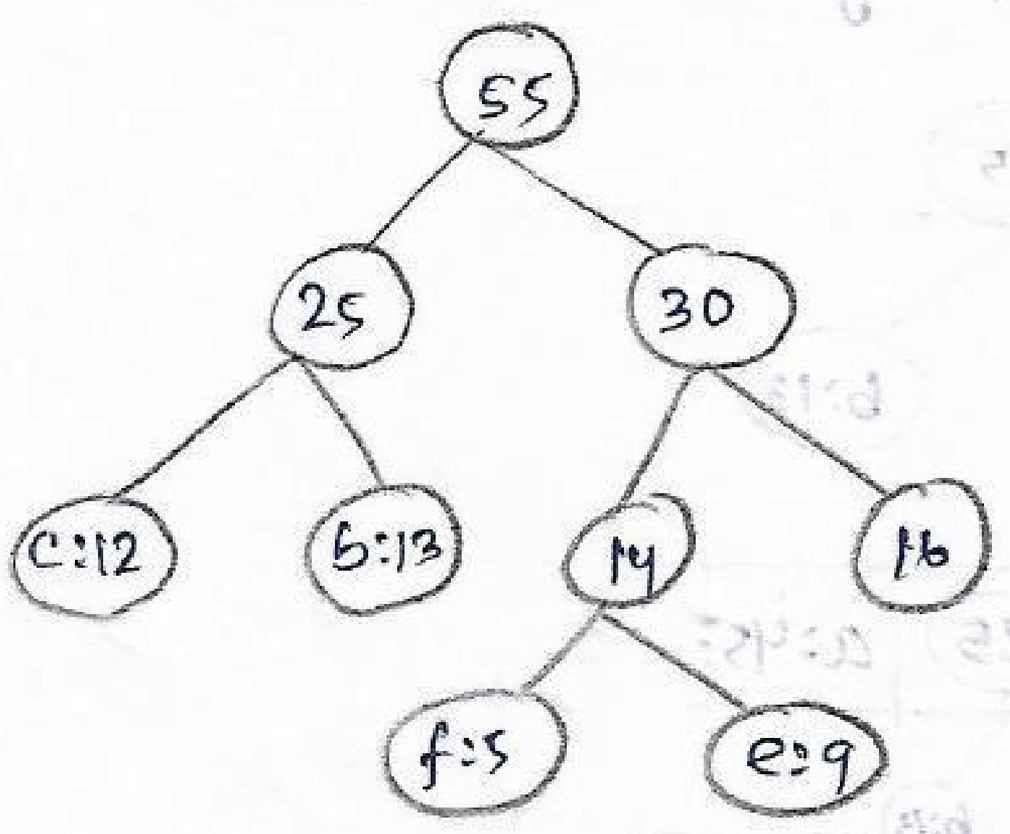


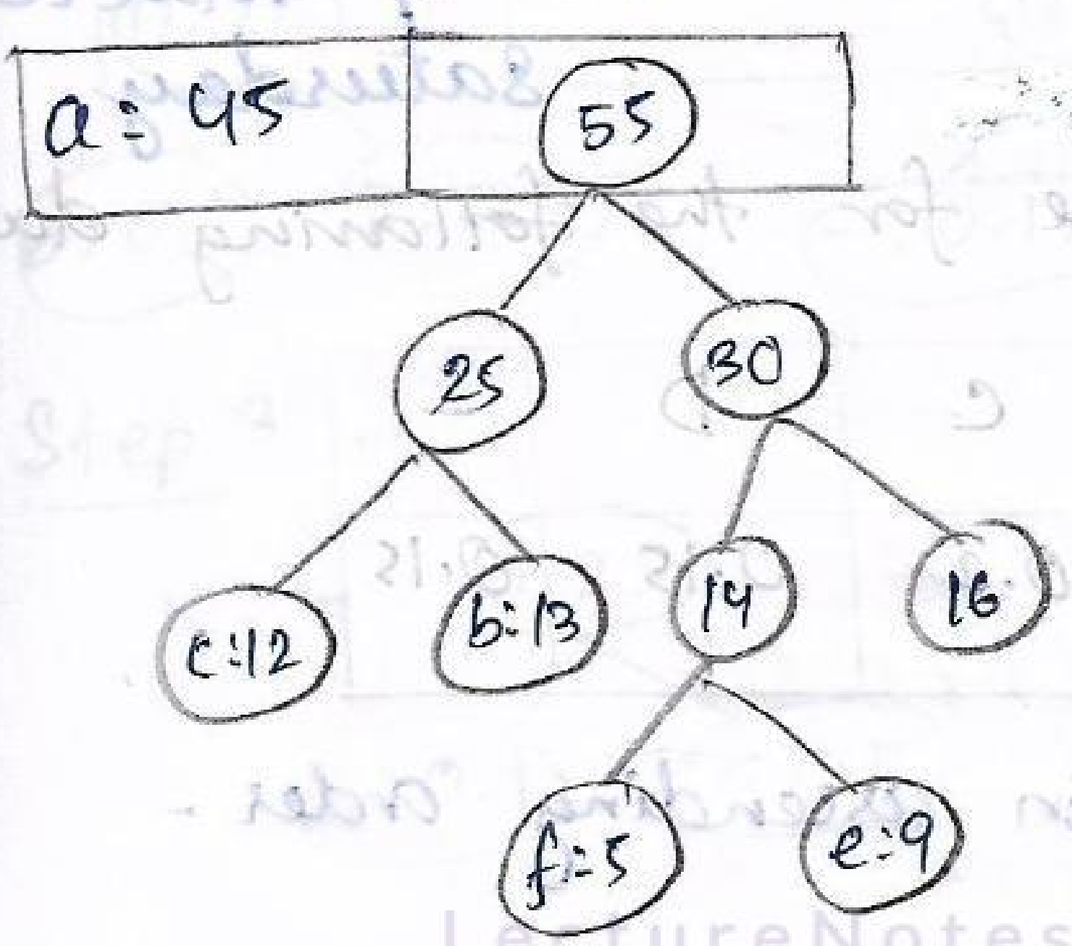
Step 4



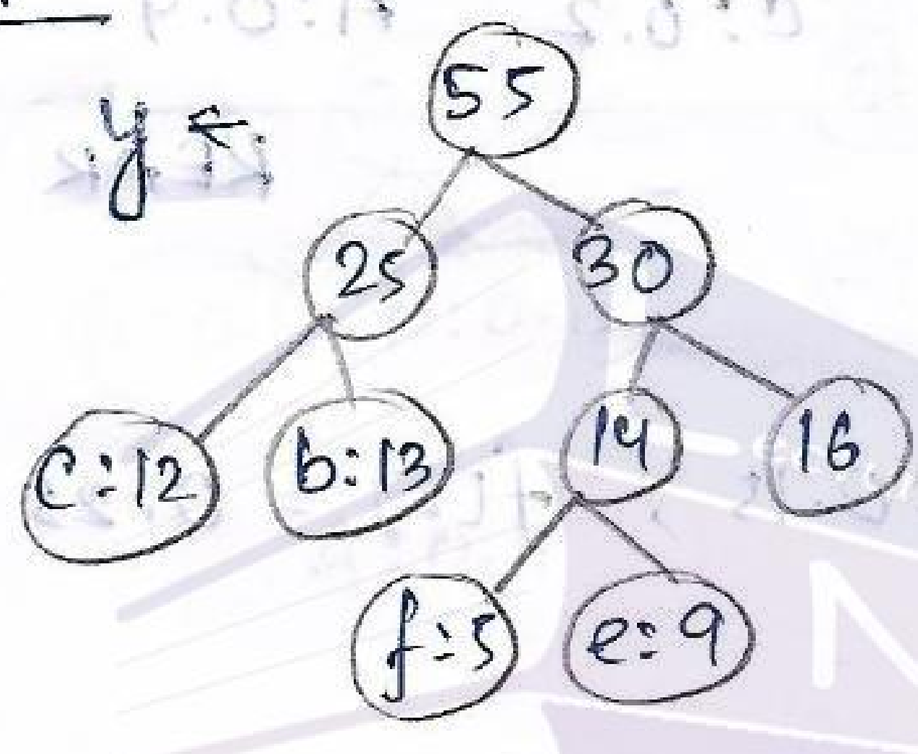


f(z) = 55

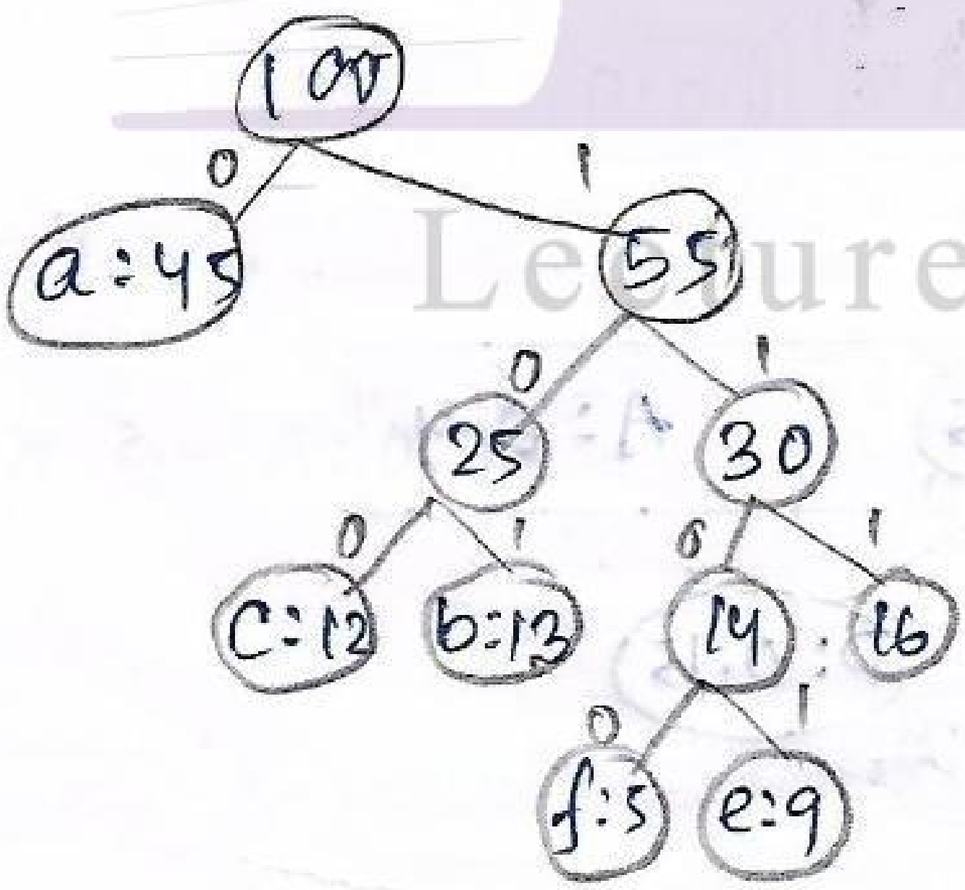




Step 5



x ← a:45 , f(z) = 100



The corresponding codes are -

a:0 , b:101 , c:100, d:111, e:1101, f:1100

Dt → 17th Mar 18
Saturday

Q. Construct a Hoffmann Code for the following data
(a)

Character	A	B	C	D	–
Probability	0.4	0.1	0.2	0.15	0.15

Sol. Arranging all characters in ascending order and store it in Q.

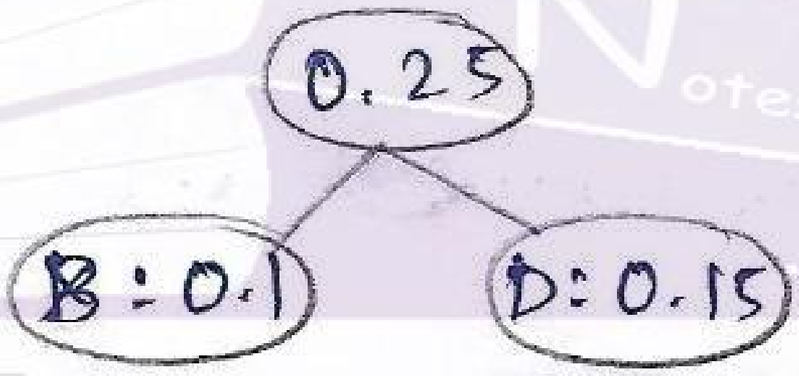
Q

B:0.1	D:0.15	–:0.15	C:0.2	A:0.4

FRONT REAR

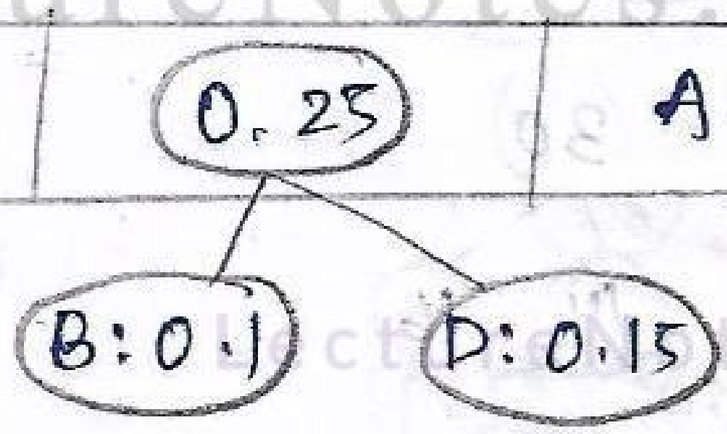
Step 1 -

$x \leftarrow$ B:0.1 , $y \leftarrow$ D:0.15 , $f[z] = 0.25$



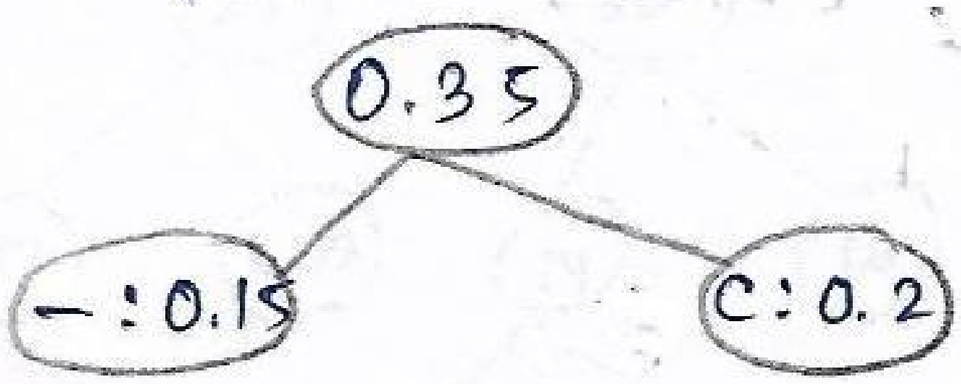
Q

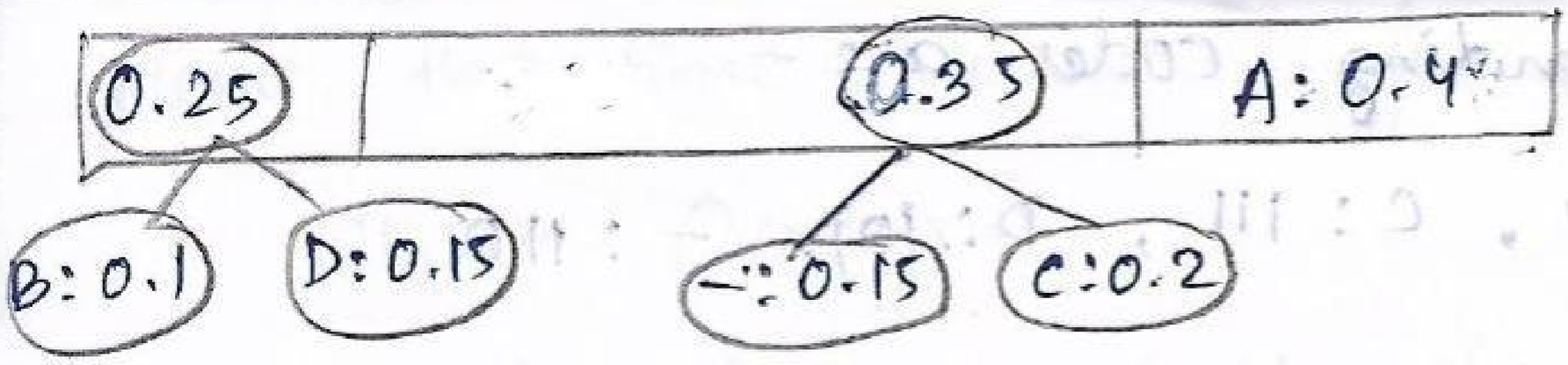
–:0.15	C:0.2	0.25	A:0.4



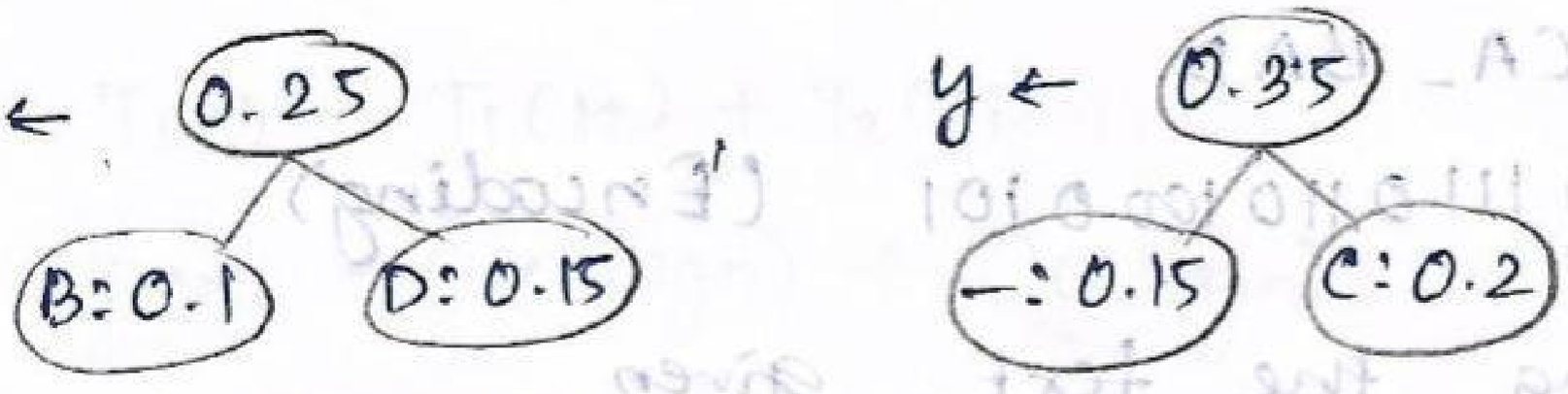
Step 2 →

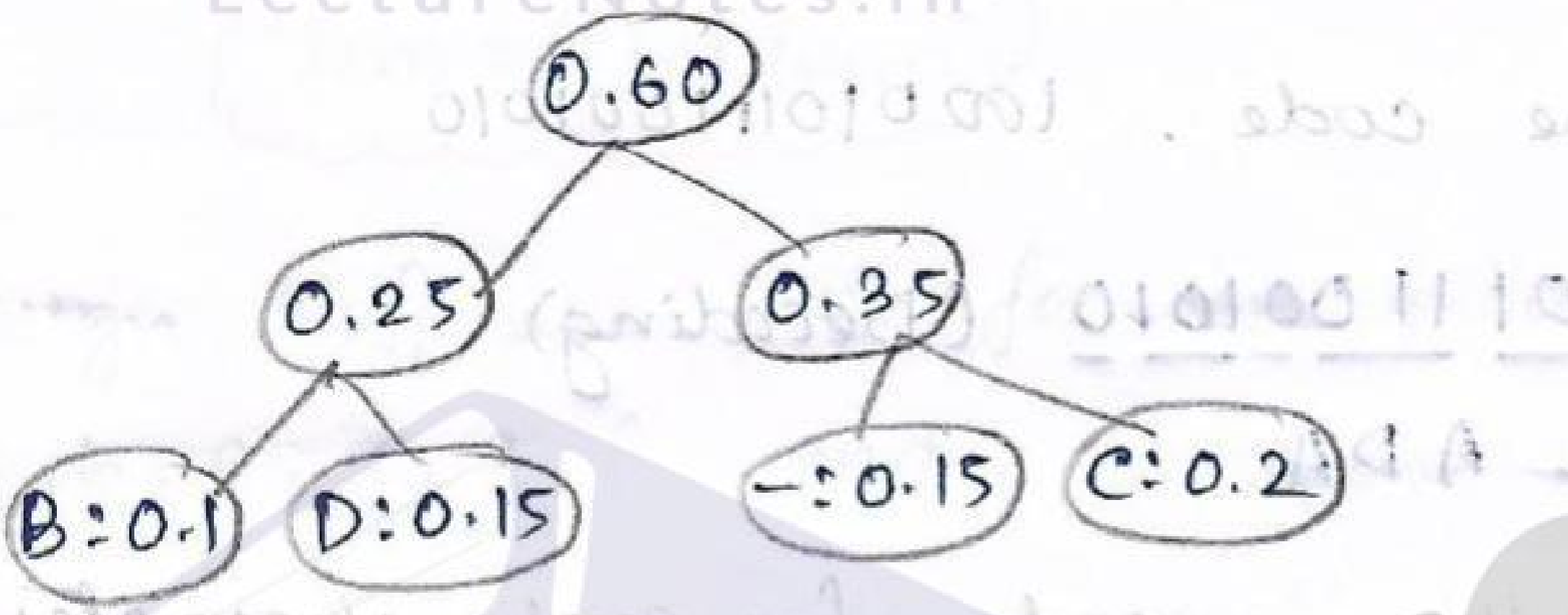
$x \leftarrow$ –:0.15 $y \leftarrow$ C:0.2 , $f(z) = 0.35$

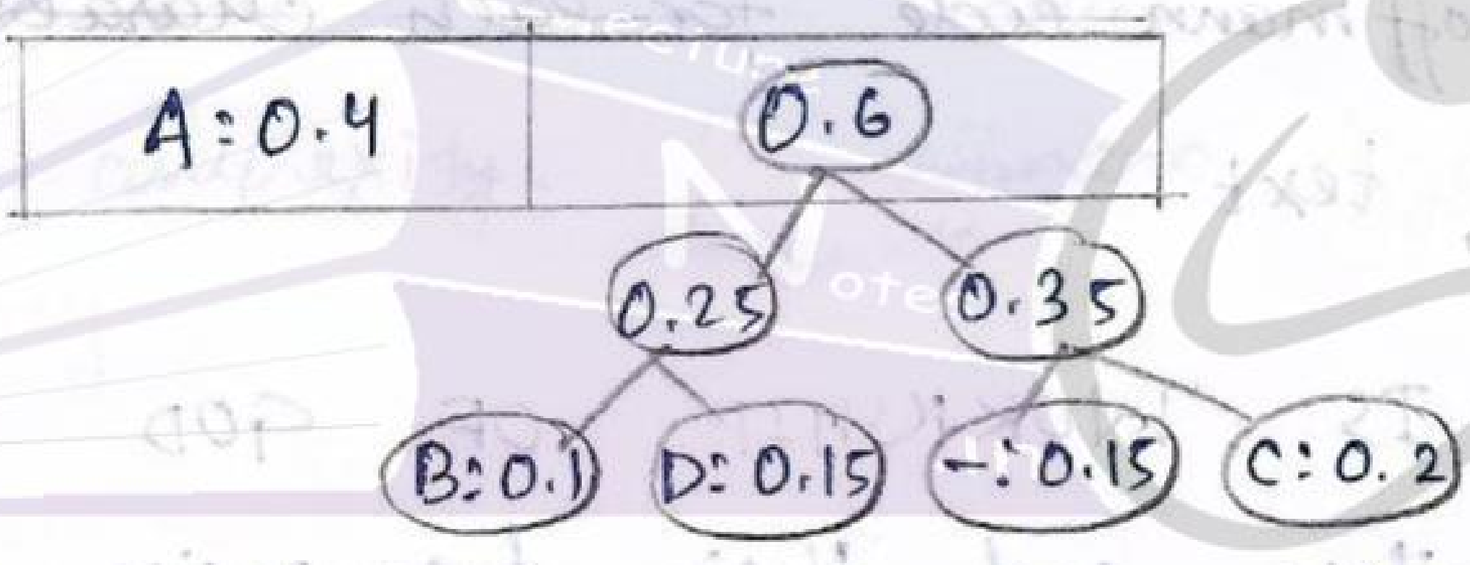



0.25
0.35
A: 0.4
B: 0.1
D: 0.15
—: 0.15
C: 0.2

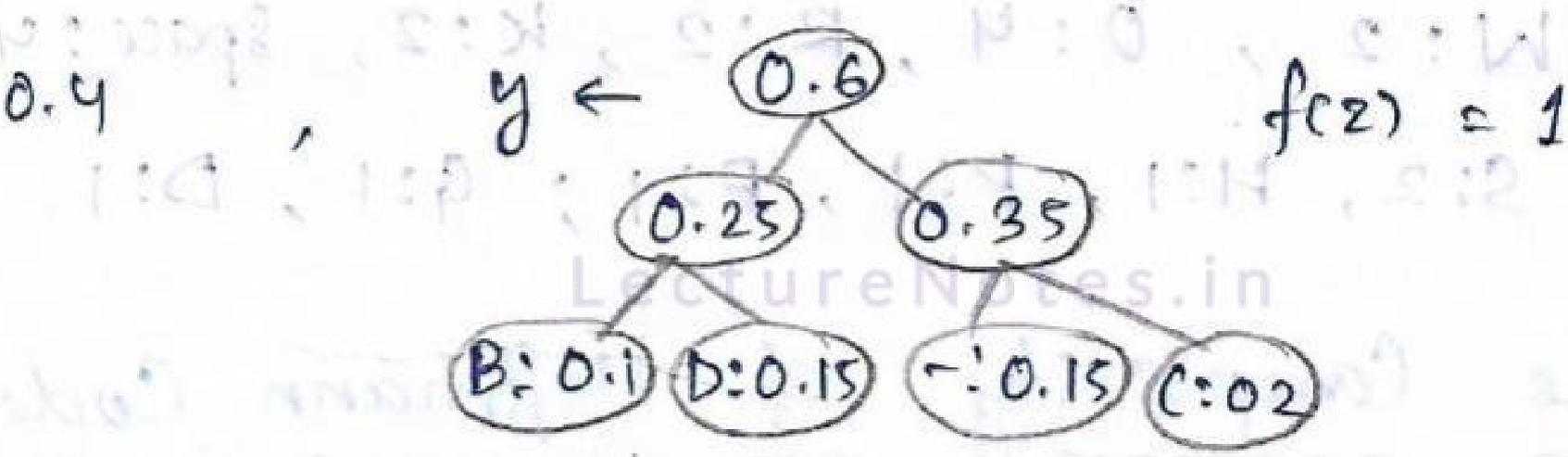
Step 3 →

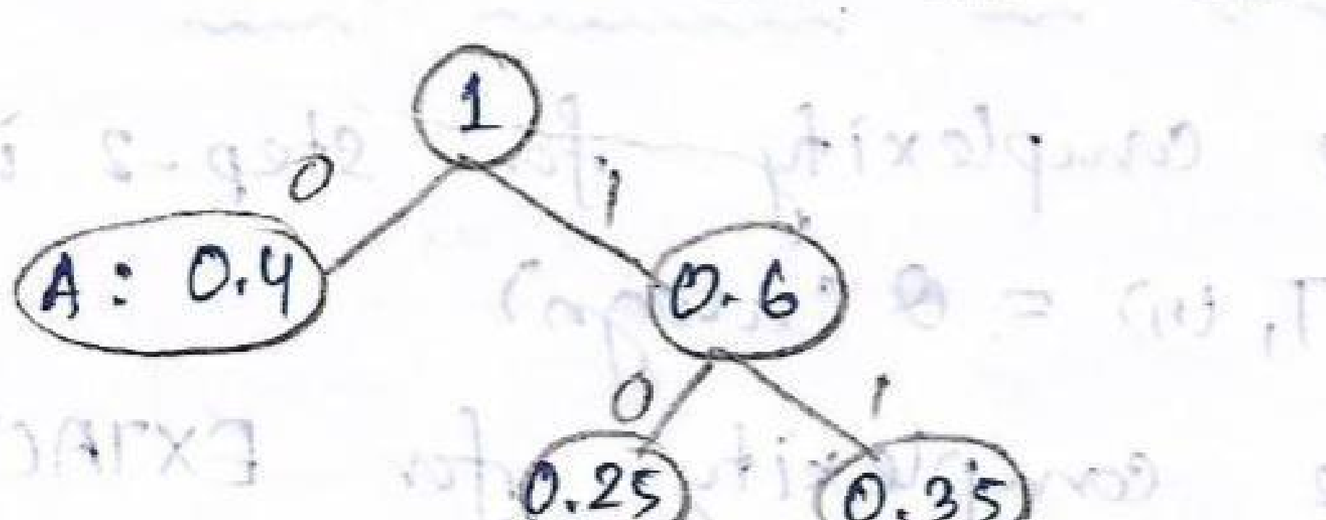

x ← 0.25
B: 0.1
D: 0.15
y ← 0.35
—: 0.15
C: 0.2
f(z) = 0.60


0.60
0.25
0.35
B: 0.1
D: 0.15
—: 0.15
C: 0.2


A: 0.4
0.6
0.25
0.35
B: 0.1
D: 0.15
—: 0.15
C: 0.2

Step 4 →


x ← A: 0.4
y ← 0.6
f(z) = 1
0.25
0.35
B: 0.1
D: 0.15
—: 0.15
C: 02


1
0
1
A: 0.4
0.6
0
1
0.25
0.35
0
1
0
1
B: 0.1
D: 0.15
—: 0.15
C: 02

The corresponding codes are -

A: 0 , B: 100 , C: 111 , D: 101 , — : 110

Q(b) Encode the text ABACA_BAD.

Sol. ABACA_BAD

0100 111 0 110 100 0 101 (Encoding)

Q(c) Decode the code. 100 0 101 111 001010

Sol. 100 0 101 111 001010 (Decoding)

B A D _ A D A

Q. Find the Hoffmann code for each character of the following text -

WORK IS WORKSHIP OF GOD

Sol. The characters and their frequencies are -

W: 2 , O: 4 , R: 2 , K: 2 , Space: 4 , I: 2 ,

S: 2 , H: 1 , P: 1 , F: 1 ; G: 1 , D: 1

Time Complexity of Hoffmann Code →

The Time complexity for step-2 in Algorithm is $T_1(n) = O(n \log n)$

the Time complexity for EXTRACT_MIN Operation is $O(\log n)$.

Hence, the time complexity for loop 4 is

$$T_2(n) = O(n \log n)$$

Therefore, the time complexity of Hoffmann code is

$$T(n) = T_1(n) + T_2(n)$$

$$T(n) = O(n \log n) + O(n \log n)$$

$$T(n) = O(n \log n)$$

Q. Design Hoffmann code for the following symbols - a, b, c, d, e, f and coresponding frequencies are 2, 4, 6, 8, 10, 16 respectively. Discuss the time complexity of Hoffmann Algorithm with binary heap.

LectureNotes.in