

## $\Rightarrow$ Unit - 3 :- Decrease and Conquer :-

- \* The decrease and conquer technique is based on exploiting the relationship between a solution to a given instance of a problem and solution to its smaller instance.
- \* It can be done in both Top-Down @ Bottom-Up approach

### \* Incremental approach :-

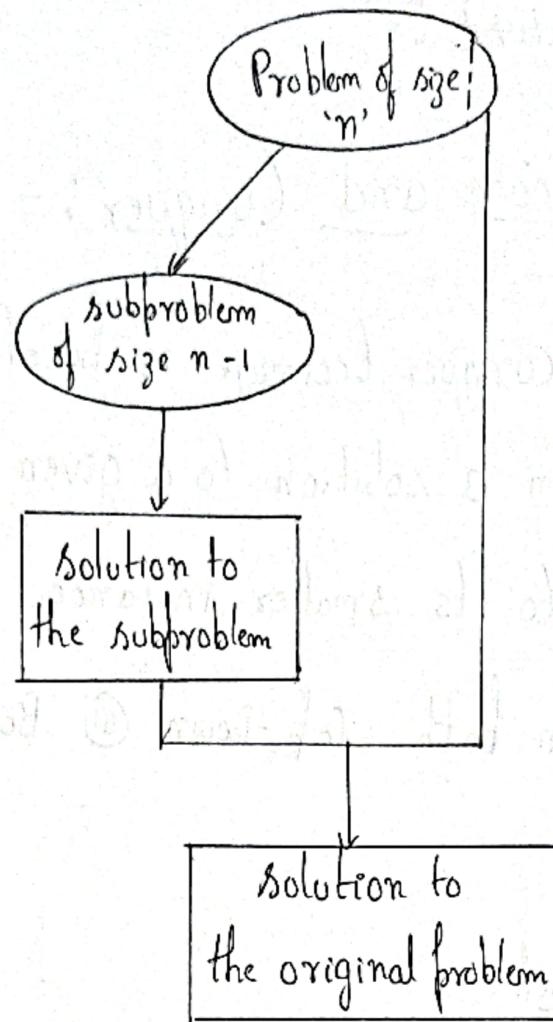
Starting with a solution to smallest instance of problem, and then moving iteratively upwards.

## $\Rightarrow$ Three major variations of Decrease and Conquer :-

### i) Decrease by a constant :-

Size of an instance is reduced by the same constant on each iteration of algorithm.

Typically, this constant is equal to 1.

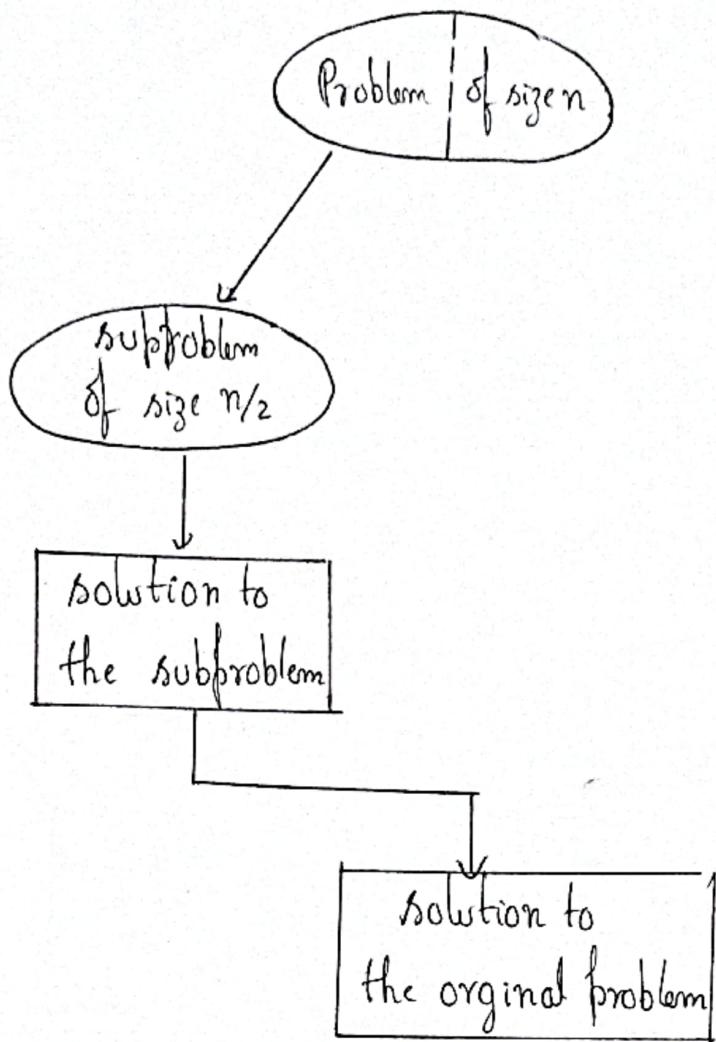


Computing  $a^n$ :

$$f(n) = \begin{cases} f(n-1) \cdot a, & \text{if } n > 0 \\ 1, & \text{if } n = 0 \end{cases}$$

### (ii) Decrease-by-constant factor:

- \* Reducing a problem instance by the same constant factor on each iteration
- \* Typically, this factor is equal to two. (Decrease-by half)



$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & , \text{ if } n=0 \end{cases}$$

(iii) Decrease by variable size factor :-

The size-reduction pattern changes from one iteration of algorithm to another.

Ex:- Euclid's GCD Algorithm.

Decrease by variable size

Ex:- GCD of 2 numbers

$$\text{gcd}(60, 42)$$



$$\text{gcd}(42, 18)$$



$$\text{gcd}(18, 6)$$



$$\text{gcd}(6, 0)$$

⇒ Insertion Sort :- Inplace & Stable Algorithm

$$8 \leftarrow 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad (1)$$

$$7 \leftarrow 8 \leftarrow 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad (2)$$

$$6 \leftarrow 7 \leftarrow 8 \leftarrow 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad (3)$$

$$5 \leftarrow 6 \leftarrow 7 \leftarrow 8 \leftarrow 4 \quad 3 \quad 2 \quad 1 \quad (4)$$

$$4 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 8 \leftarrow 3 \quad 2 \quad 1 \quad (5)$$

$$3 \leftarrow 4 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 8 \leftarrow 2 \quad 1 \quad (6)$$

$$2 \leftarrow 3 \leftarrow 4 \leftarrow 5 \leftarrow 6 \leftarrow 7 \leftarrow 8 \leftarrow 1 \quad (7)$$

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8$$

Algorithm :-

Insertionsort ( $A[0 \dots n-1]$ )

|| I/p :- An array  $A[0 \dots n-1]$  of  $n$  elements

|| o/p :- The sorted array in ascending order

for  $i \leftarrow 1$  to  $n-1$

$t \leftarrow A[i]$

$j \leftarrow i-1$

    while ( $j \geq 0$  And  $A[j] > t$

$A[j+1] \leftarrow A[j]$ ;  $j \leftarrow j-1$

    end while

$A[j+1] \leftarrow t$

end for

Analysis :-

Input Size  $\Rightarrow n$

Basic ~~comparison~~ <sup>operation</sup>  $\Rightarrow$  Comparison

$\Rightarrow$  Best Case Scenario :-

When the array is already sorted in ascending order, then only 1 comparison occurs in each iteration

$$= \sum_{i=1}^{n-1} 1$$

$$\therefore C_{\text{Best}}(n) = n-1 \in \Theta(n)$$

$\Rightarrow$  Worst Case Scenario :-

When the array is sorted in descending order

Maximum number of comparison made in each iteration

$$C_{\text{worst}}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=1}^{n-1} i-1+1$$

$$= \sum_{i=1}^{n-1} i$$

$$= \frac{n(n-1)}{2}$$

$$\therefore C_{\text{worst}}(n) \in \Theta(n^2)$$

$\Rightarrow$  Average Case :-

The input array is some random input.

$$C_{\text{avg}}(n) = O(n^2)$$

# INSERTION

Ex:-

I<sup>j</sup> N<sup>i</sup> S E R T I, O N<sub>z</sub>, ①

I, N<sup>j</sup> I<sup>i</sup> S E R T I, O N<sub>z</sub>, ①

I<sup>i</sup> N<sup>j</sup> S T E R T I, O N<sub>z</sub>, ③

E I, N, S<sup>j</sup> T<sup>i</sup> R T I, O N<sub>z</sub>, ①

E I, N, R S<sup>j</sup> T<sup>i</sup> I, O N<sub>z</sub>, ①

E I, N, R S T<sup>j</sup> T<sup>i</sup> I, O N<sub>z</sub>, ⑤

E E I, N, R S T<sup>j</sup> I, O N<sub>z</sub>, ④

E E I, I, N, O R S T<sup>j</sup> I, O N<sub>z</sub>, ⑤

E E I, I, N, N, O R S T

$\Rightarrow$  Computing  $a^n$  using Decrease & Conquer :-

$\text{apowern}(a, n)$

{

if  $n = 1$

return  $a$

$\text{temp} \leftarrow \text{apowern}(a, n/2)$

if  $n \% 2 = 0$

return  $\text{temp} \times \text{temp}$

Else

return  $a \times \text{temp}$

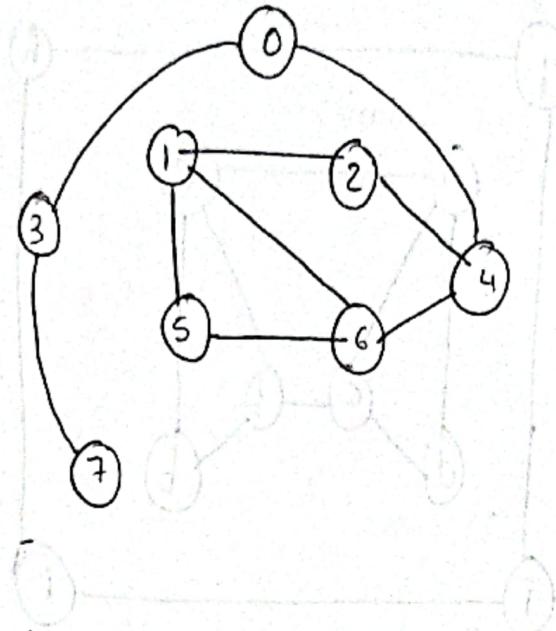
}

$\Rightarrow$  Depth First Search (DFS) :-

Applications :-

- i To check whether graph is connected or not
- ii To check cycle in a graph
- iii To check the articulation point of a graph

The vertex on removal of which, graph becomes disconnected.



DFS :- 0 3 7 4 2 1 5 6

Visited array :-

0	1	2	3	4	5	6	7	8	X
1	6	5	2	4	7	8	3		

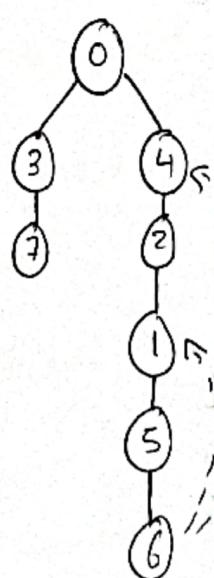
Order in which  
vertices are  
visited.

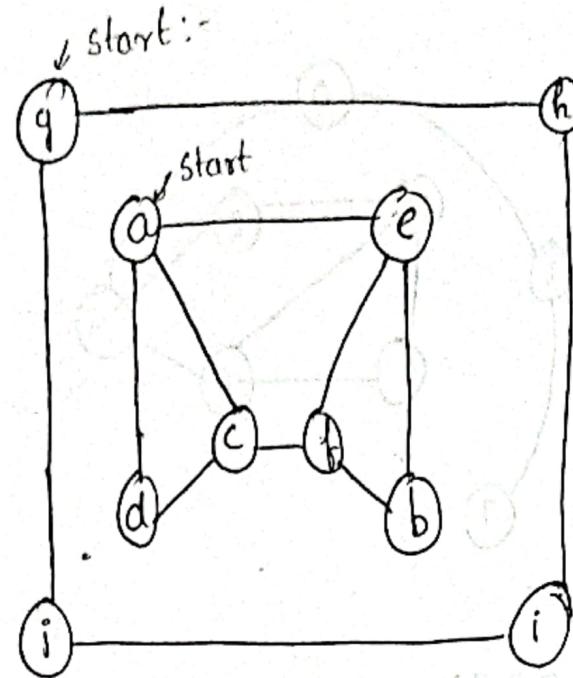
Stack :-

dfs(6)	3
dfs(5)	4
dfs(7)	5
dfs(2)	6
dfs(4)	7
dfs(7)	1
dfs(5)	2
dfs(0)	8

Order in which  
vertices become dead end

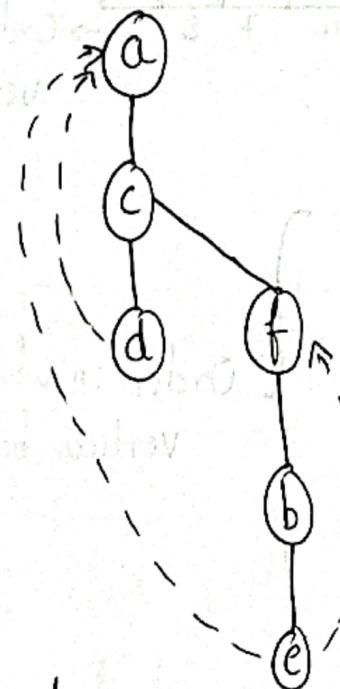
Spanning Tree :-





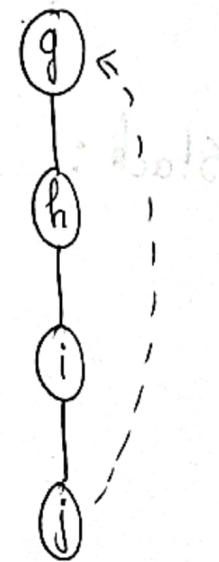
Spanning Trees :-

a

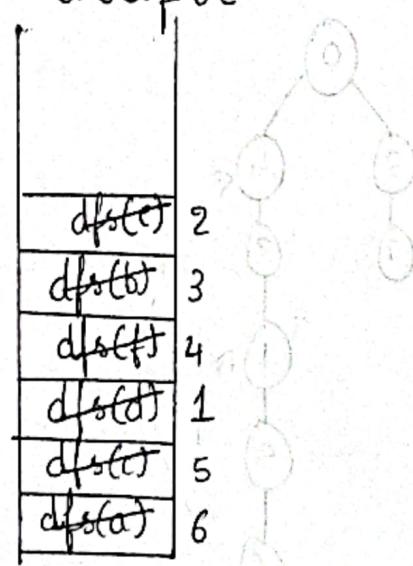


Order of Visit :- acdfbe

g



Order of Visit :- ghij



Algorithm :-

DFS(G)

// Implements a depth first search traversal of a given graph.

// IIP :- Graph G(V, E)

//lop :- Graph G with its vertices marked with consecutive integers in the order they have been visited by DFS traversal.

count  $\leftarrow 0$

for each vertex v in V do

if v is marked with 0

dfs(v)

dfs(v)

// Visit recursively all the unvisited vertices connected to vertex v by a path and number them in the order they are encountered via global variable count

count  $\leftarrow$  count + 1 ; mark v with count

for each vertex w in v adjacent to v do

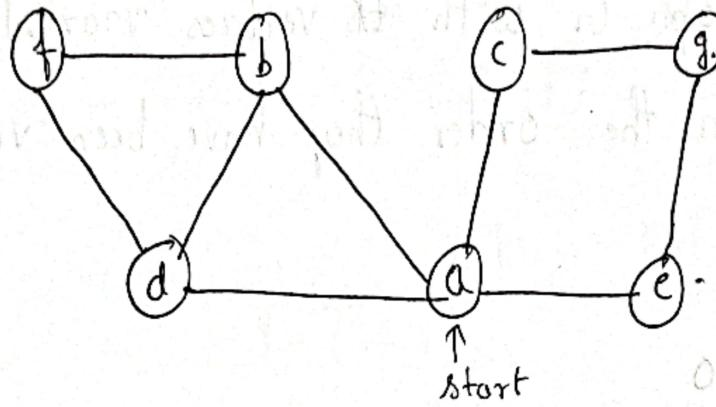
if w is marked with 0

dfs(w)

Ex:- For the given graph,

i) Write adjacency matrix & list

ii) Starting from vertex 'a', construct dfs tree and give the order in which vertices were reached first time and order in which vertices become dead end



Soln:-

Order of visit = abdfcge

dfs(e)	4
dfs(g)	5
dfs(c)	6
dfs(f)	1
dfs(d)	2
dfs(b)	3
dfs(a)	7

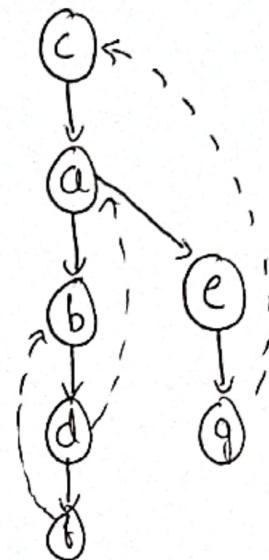
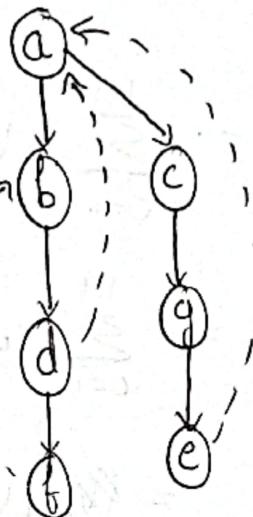
Order of Dead-End:-

fdbegca

Adjacency matrix:-

	a	b	c	d	e	f	g
a	0	1	1	1	1	0	0
b	1	0	0	1	0	1	0
c	1	0	0	0	1	0	1
d	1	1	0	0	0	1	0
e	1	0	0	0	0	0	1
f	0	1	0	1	0	0	0
g	0	0	1	0	1	0	0

## Spanning Tree :-



① Let ' $G$ ' be a graph with ' $n$ ' vertices and ' $m$ ' edges. If all its DFS forest (for traversal from different vertices), all have same number of trees.  $\Rightarrow$  True

② All its dfs forest will have same number of tree edges & back edges.  $\Rightarrow$  True

Soln:- For any connected undirected graph with ' $v$ ' vertices then the number of Tree edges in the graph.

$$|E^{(\text{tree})}| = |V| - 1$$

$$\begin{aligned} |E^{(\text{back})}| &= |E| - |E^{(\text{tree})}| \\ &= |E| - (|V| - 1) \\ &= |E| - |V| + 1 \end{aligned}$$

## Disconnected Graph :-

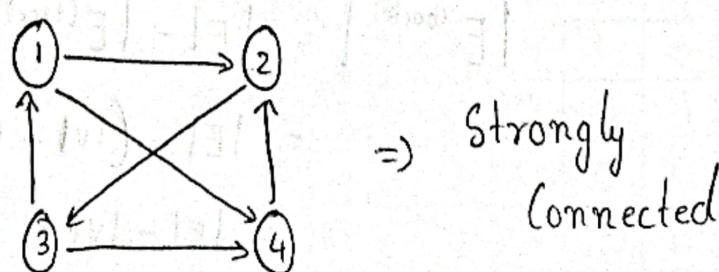
$$\begin{aligned}
 |E^{(\text{tree})}| &= \sum_{c=1}^{|C|} |E_c^{(\text{tree})}| \\
 &= \sum_{c=1}^{|C|} |V_c| - 1 \\
 &= \sum_{c=1}^{|C|} |V_c| - \sum_{c=1}^{|C|} 1 \\
 &= |V| - |C|
 \end{aligned}$$

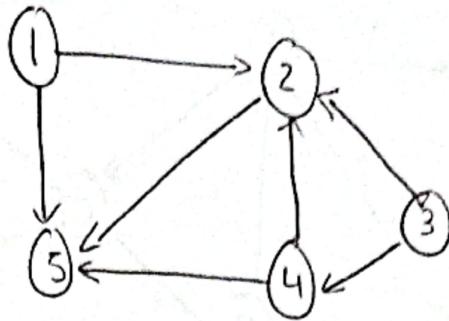
$$\begin{aligned}
 |E^{(\text{back})}| &= |E| - (|V| - |C|) \\
 &= |E| - |V| + |C|
 \end{aligned}$$

## Strongly Connected Graphs / Components :-

A graph is called Strongly connected if there is a path from vertex 'u' to vertex 'v'. for each vertices pair  $u, v \in V$

Ex:-



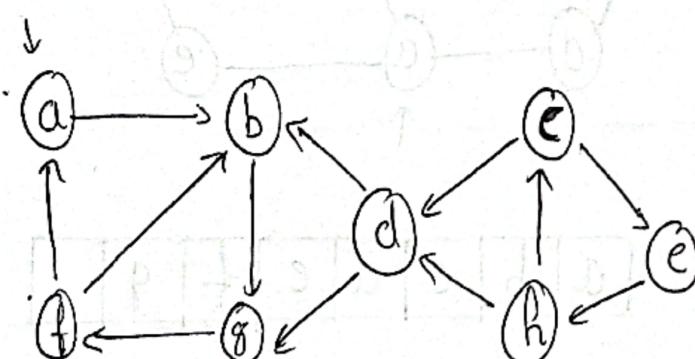


DFS :- 1 2 5  
 4 3 } 3 - components  
 3 2 }

$\Rightarrow$  Kosaraju's Algorithm:-

$\Rightarrow$  Steps to find strongly connected components:-

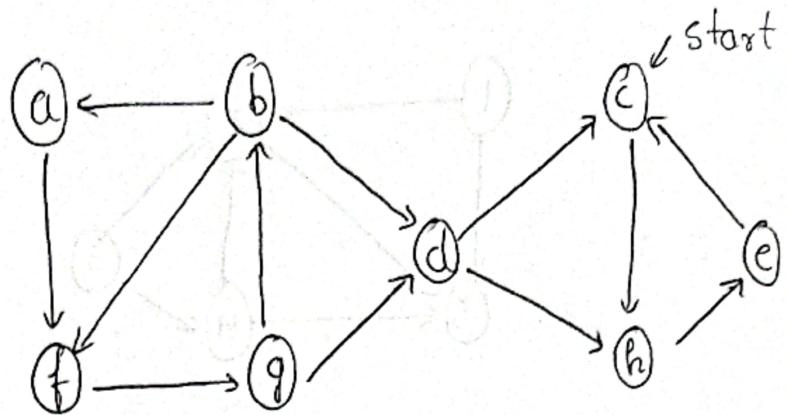
- (i) Conduct the DFS and note order of dead end
- (ii) Change direction of all edges of digraph and do dfs on the new digraph starting from the vertex that became dead end in last.



DFS:- abgf (fbga)  $\rightarrow$  order of dead end  
 cehd (dhec)  $\rightarrow$  order of dead end

Descending order of dead end :- cehdabgf

Step 2:-



DFS:- ched (ehc)

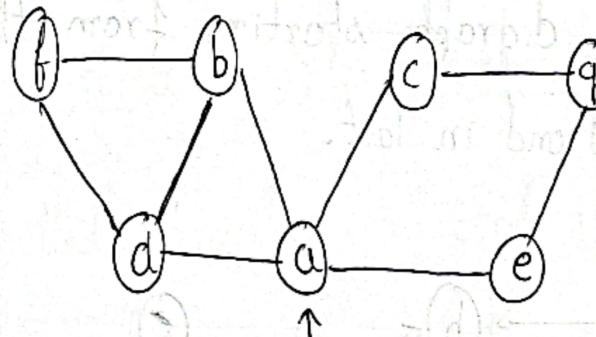
d

afgb (bgfa)

} 3-strongly connected components.

Breadth First Search :-

Ex ① :-

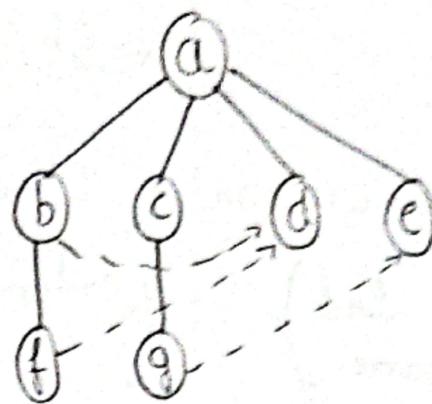


Queue :

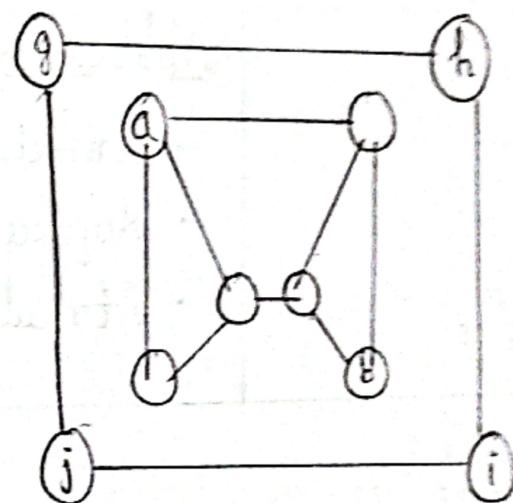
a	b	c	d	e	f	g
---	---	---	---	---	---	---

Order of visit :- abcdefg

## BFS Spanning Tree:-



Ex ② :-



## BFS

- \* Queue is used
- \* It is called one-ordering  
(Order of visit and dead end in same)
- \* It has Tree edge, cross edge & back edge

→ Application :-

- Connectivity
- Acyclicity
- Minimum path edge

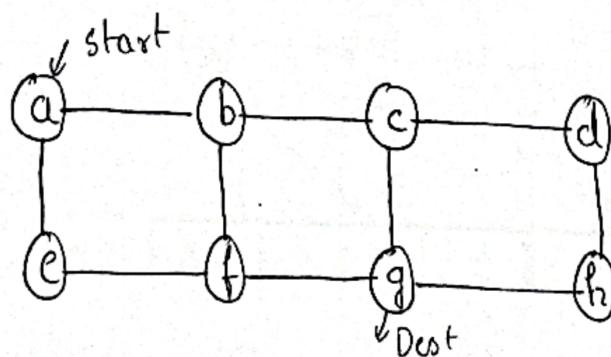
## DFS

- \* Stack is used
- \* It is Two-ordering  
(Order of visit and dead end is different)
- \* It has Tree edge & Cross back edge

→ Application :-

- Connectivity
- Acyclicity
- Articulation Point

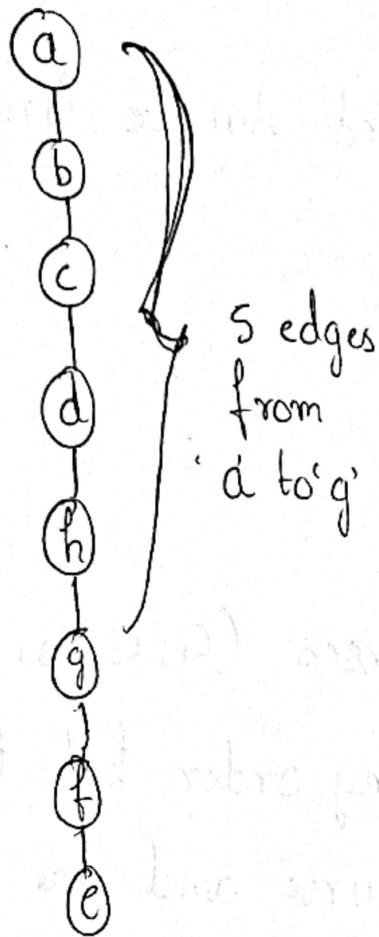
⇒ Minimum Path Edge :-



DFS :- abc dhgfe

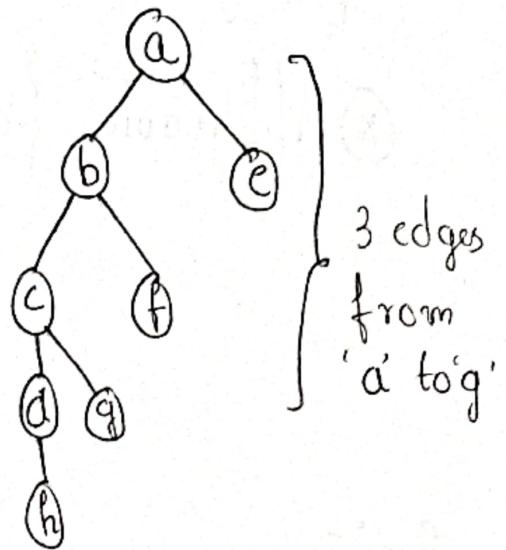
BFS :- abecfgdh

DFS



5 edges  
from  
'a to 'g'

BFS



3 edges  
from  
'a to 'g'

- \* The efficiency of BFS & DFS is dependent on the data structure used to implement the graph.

$\Rightarrow$  Topological Sorting :- (Directed graph)

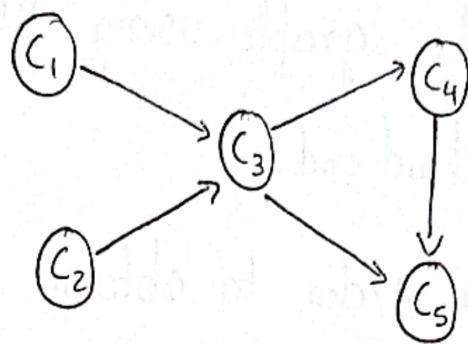
\* Applicable for Directed Acyclic Graph (DAG)

Ex:-

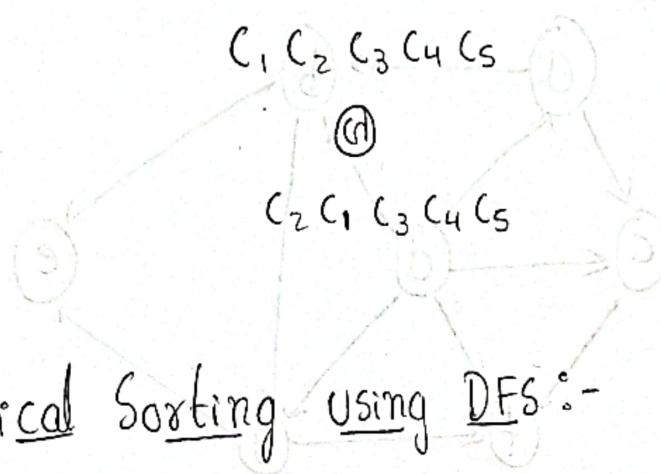
Consider a set of 5 courses ( $C_1, C_2, C_3, C_4, C_5$ ), the courses can be taken in any order but there are prerequisites for ~~each~~ <sup>some</sup> course and one course per term ~~in~~ each in max.

Course	Prerequisites
$C_1 \& C_2$	None
$C_3$	$C_1 \& C_2$
$C_4$	$C_3$
$C_5$	$C_4, C_3$

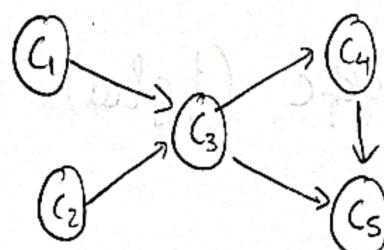
The graph is written as,



Tobological ordering :-



=> Topological Sorting using DFS :-



DFS :-  $C_1, C_3, C_4, C_5$  ( $C_5, C_4, C_3, C_1$ )

$C_2$

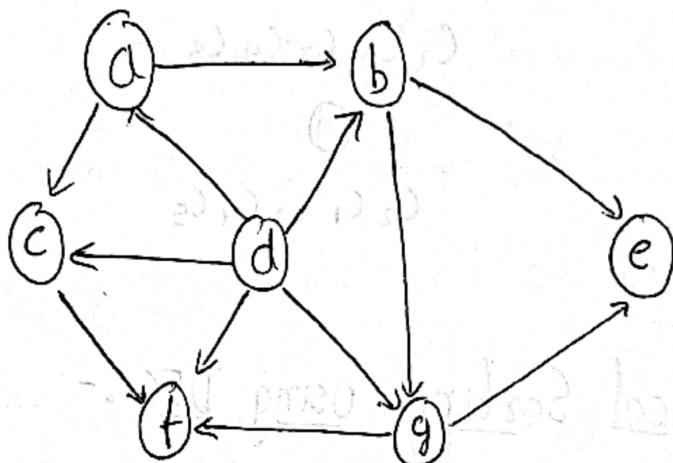
Order of dead end :-  $C_5, C_4, C_3, C_1, C_2$

$\therefore$  Topological Sorting :-  $C_2, C_1, C_3, C_4, C_5$

$\Rightarrow$  Topological Sorting using DFS :-

- ① Traverse the graph using DFS & note down order of dead end
- ② Reverse the order to obtain Topological sorting

Ex(2):-



DFS:- abgefca (fgbca)

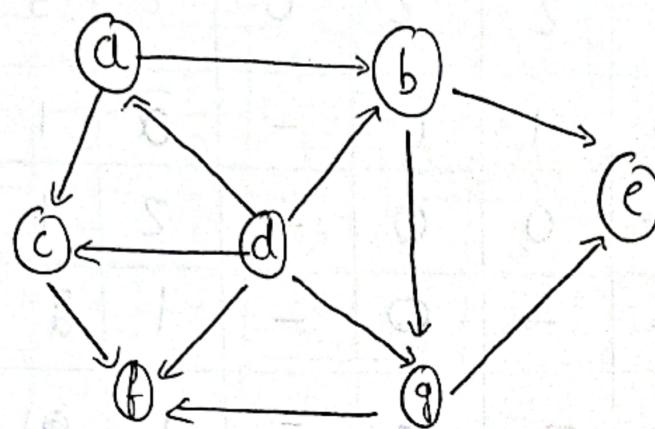
d

e

Order of dead end:- fgbcad

$\therefore$  Topological Sorting:- dacbgfe

$\Rightarrow$  Source removal method for Topological Sorting:-



Adjacency Matrix :-

	a	b	c	d	e	f	g
a	0	1	1	0	0	0	0
b	0	0	0	0	1	0	1
c	0	0	0	0	0	1	0
d	1	1	1	0	0	1	1
e	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0
g	0	0	0	0	1	1	0

Indegrees	1	2	2	0	2	3	2

- \* There should be atleast one vertex with zero <sup>in</sup>degree

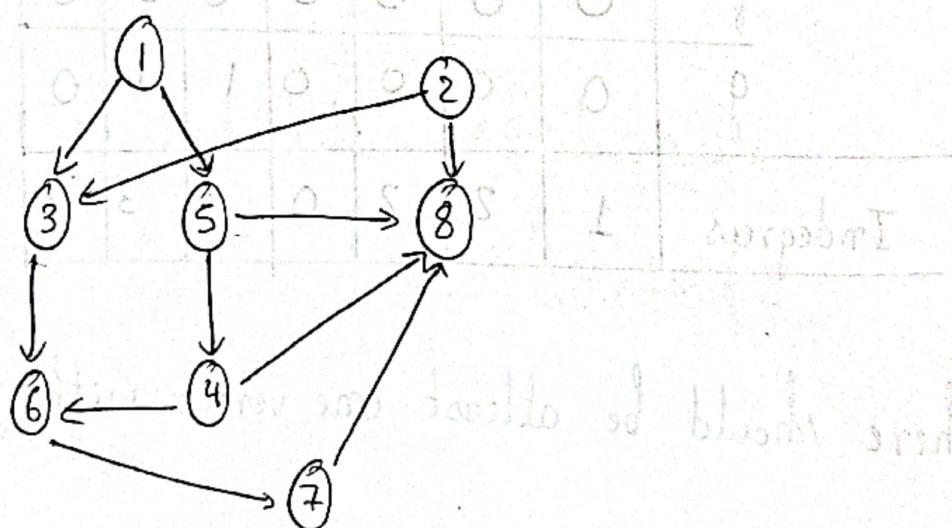
Indegrees :-

a	b	c	d	e	f	g
1	2	2	0	2	3	2
0	1	1	-	a	1	1
-	0	0	-	2	1	1
-	-	0	-	1	a	0
-	-	-	-	1	0	0
-	-	-	-	0	0	-

Queue  $\rightarrow$  d a b c g e f

$\therefore$  Topological order :- dab c g e f

Ex:- ② :-



1	2	3	4	5	6	7	8
0	0	2	1	1	2	1	4
-	0	1	1	0	a	1	34
-	-	0	1	0	02	1	3
-	-	0	0	-	a	1	a
-	-	-	0	-	01	1	a
-	-	-	-	-	0	0	1
-	-	-	-	-	-	0	01
-	-	-	-	-	-	-	0

Queue :-

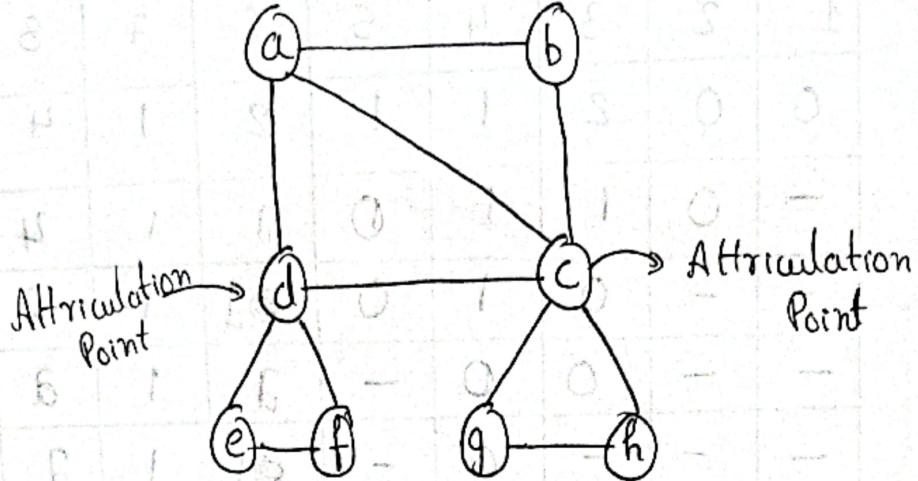
1	2	5	3	4	6	7	8
---	---	---	---	---	---	---	---

Topological order :- 1 2 5 3 4 6 7 8

⇒ Attribution Point & Biconnected Components :-

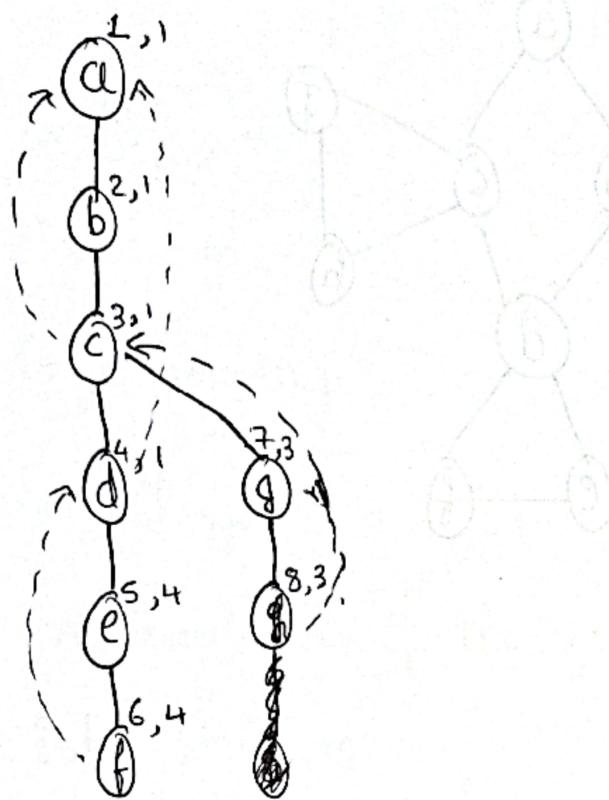
\* It is the node in a graph which on removal makes the graph disconnected.

\* It is not desirable in graph.



Steps :-

- ① Traverse the graph using DFS
- ② While traversing, number each vertex of tree with two numbers
  - (i) First number (L-Value) is the order in which vertex was visited.
  - (ii) Second number (R-Value) is the lowest numbered vertex
- ③ Root is an articulation point, if it has 2 children
- ④ Any other vertex other than root is articulation point  
 if  $R\text{Value}[\text{child}] \geq L\text{Value}[\text{Parent}]$   
 then Parent is an Articulation Point



	a	b	c	d	e	f	g	h
L-V	1	2	3	4	5	6	7	8
R-V	1	1	1	1	4	4	3	3

$R[c] < L[b] \Rightarrow b$  is not articulation point

$R[d] \leq L(c)$

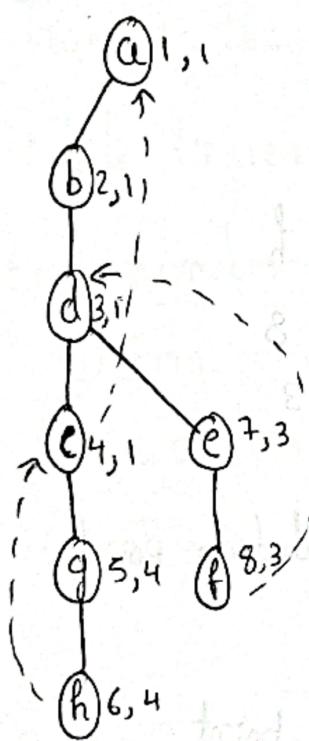
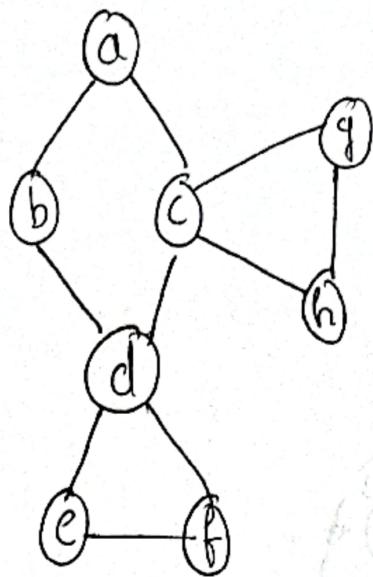
$R[g] \geq L(c) \Rightarrow c$  is articulation point

$R[e] \geq L(d) \Rightarrow d$  is articulation point

$R[f] < L(e) \Rightarrow e$  is not articulation point

$R[h] < L(g) \Rightarrow g$  is not articulation point

Ex-②:-



	a	b	c	d	e	f	g	h
LV	1	2	4	3	7	8	5	6
RV	1	1	1	1	3	3	4	4

## $\Rightarrow$ Transform and Conquer :-

### ① Instance Simplification :-

Given instance is simplified into a simple ② convenient form. Ex:- Presorting

### ② Representation Change :-

③ The representation of the instance is changed, i.e.

changing Data Structure

### ③ Problem Reduction :-

Transformation to a form to which the algorithm is already available.

## $\Rightarrow$ Instance Simplification :-

### Finding Uniqueness of array by presorting

Unique( $A[0 \dots n-1]$ )

|| I/P :- An ordered array  $A[0 \dots n-1]$

|| O/P :- True if A is unique false otherwise

for i ← 0 to  $n-2$  do

    if  $A[i] = A[i+1]$

        return false

    end for

    return True

→ Explanations for PreSorting :-

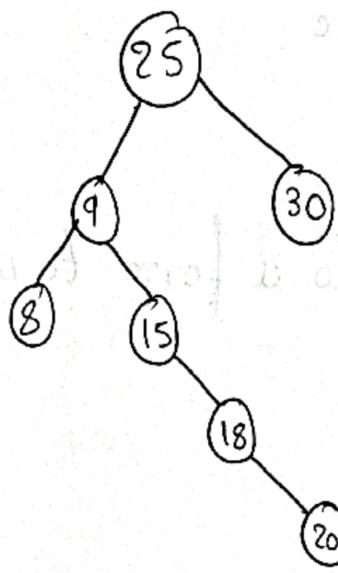
- Uniqueness array
- Binary Search
- Computing mode in an array

⇒ Representation Change :-

Array :- 25, 30, 9, 15, 18, 20, 8 (No of comparison = 7)

Binary Search :-

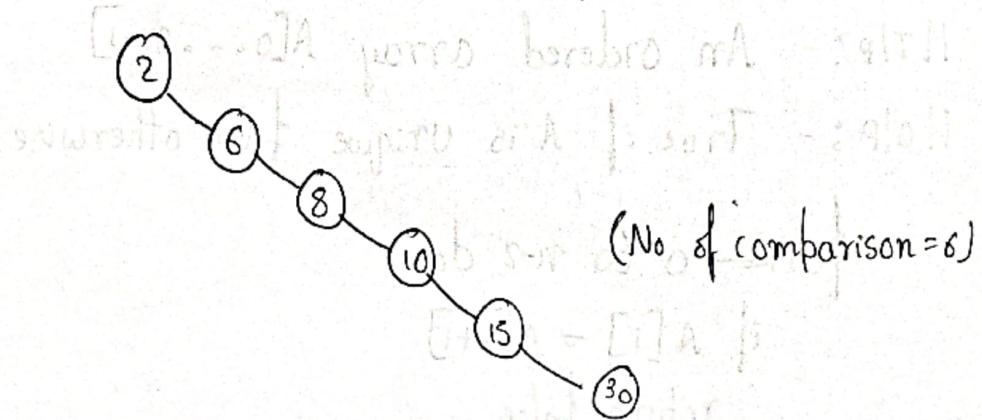
Tree



(No of comparison = 3)

Issue :-

Array :- 2 6 8 10 15 30 (No of comparison = 6)

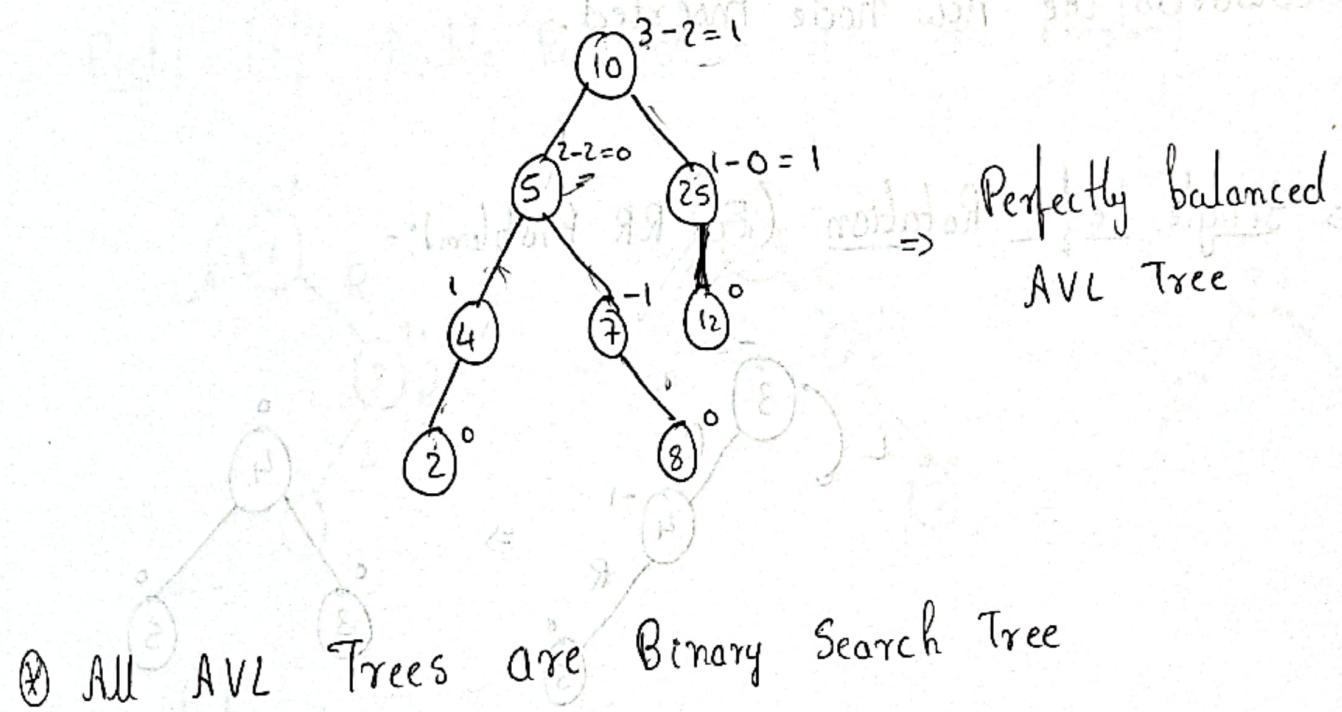


(No of comparison = 6)

→ Converting unbalanced Tree to a balanced Tree :-

→ AVL Tree :-

It is A Binary Search Tree in which balance factor of every node defined as difference b/w height of left and right subtree is either 0 @ -1 @ +1



① All AVL Trees are Binary Search Tree

→ AVL Rotations / Local Transformations :-

(i) Single Rotation

Left

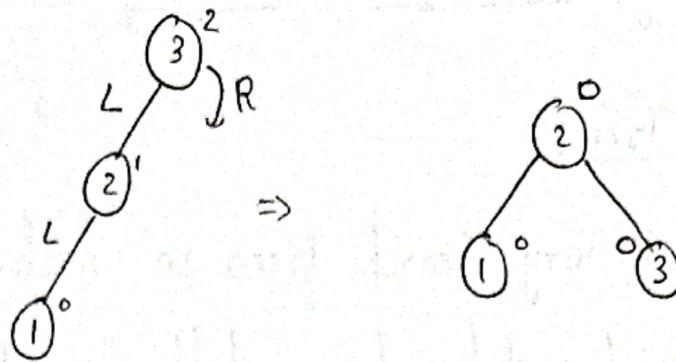
Right

(ii) Double Rotation

Left-Right

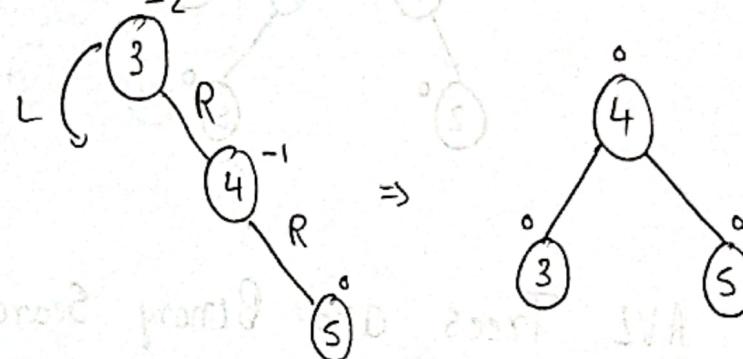
Right-Left

→ Single Right Rotation (For LL Problem):-

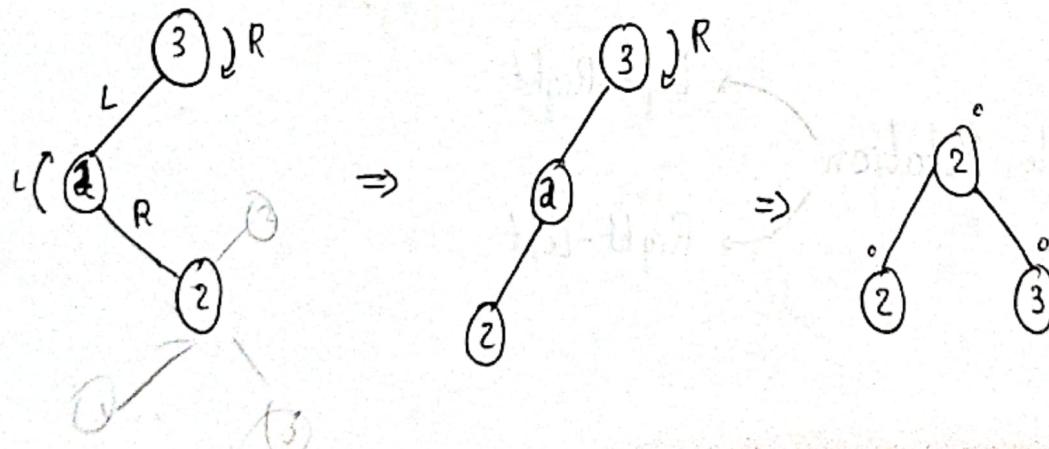


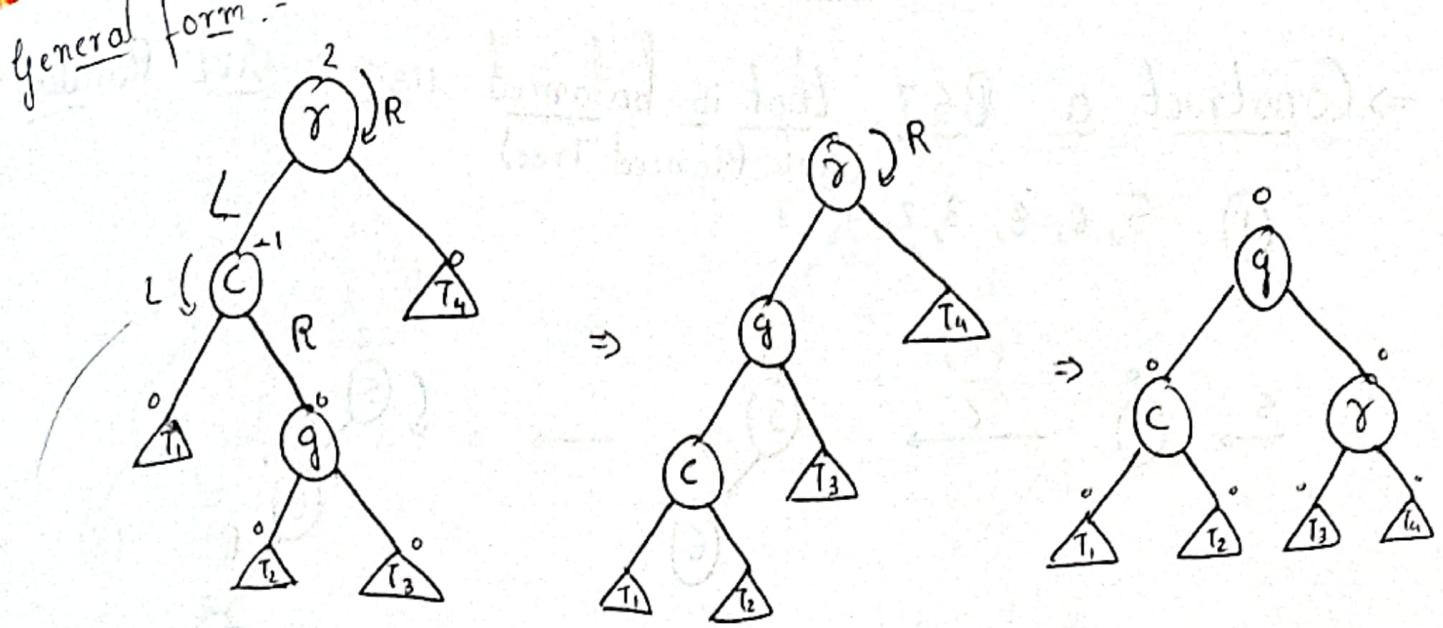
From the node that is imbalanced move 2 steps  
towards the new node inserted.

→ Single Left Rotation (For RR Problem):-

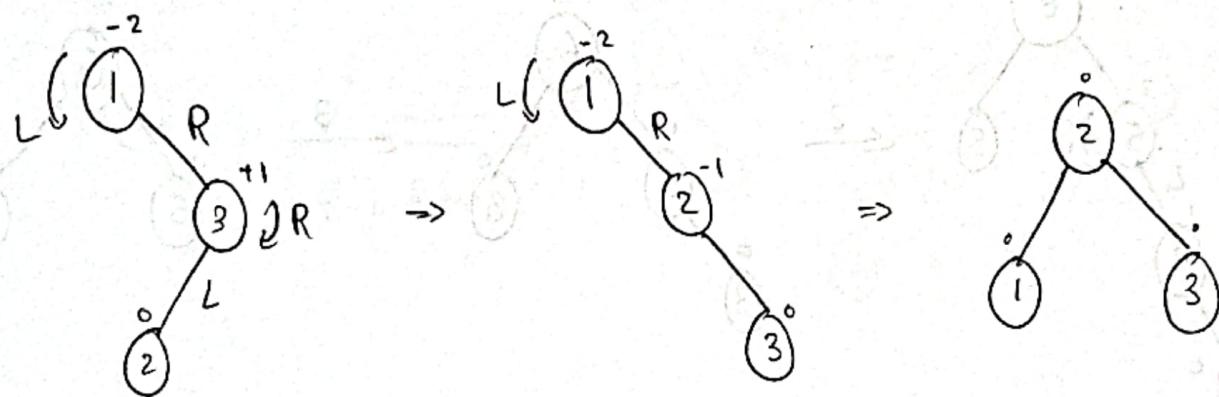


→ Left-Right Double Rotation (For LR Problem):-

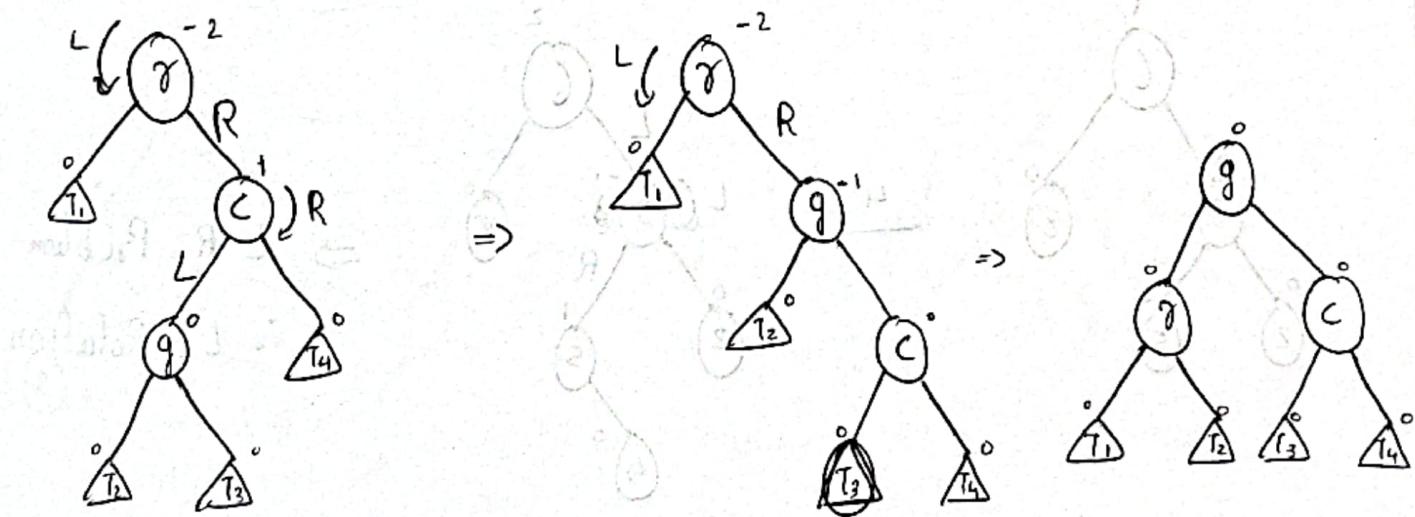




→ Right-Left Double Rotation:- (For RL Problem)



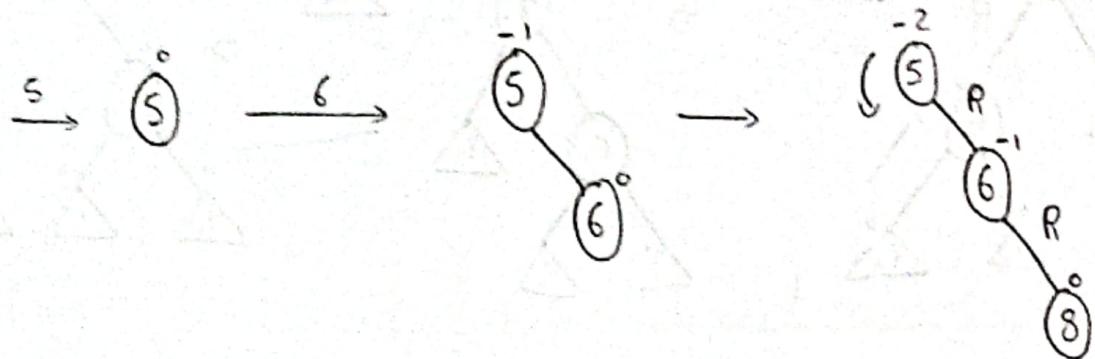
→ General form:-



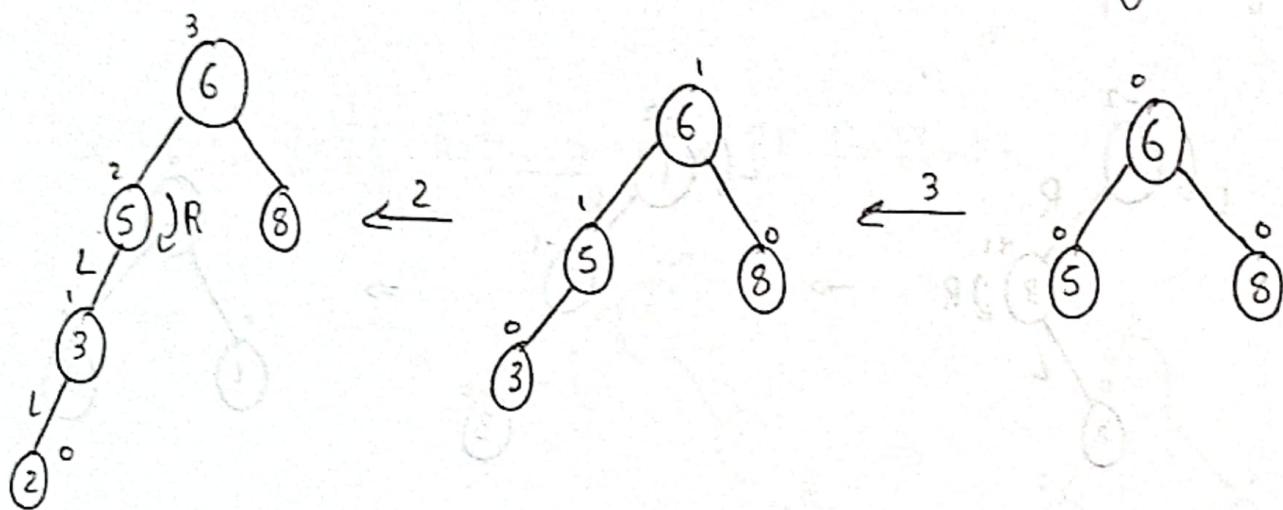
$\Rightarrow$  Construct a BST that is balanced using AVL Rotation

(AVL Balanced Tree)

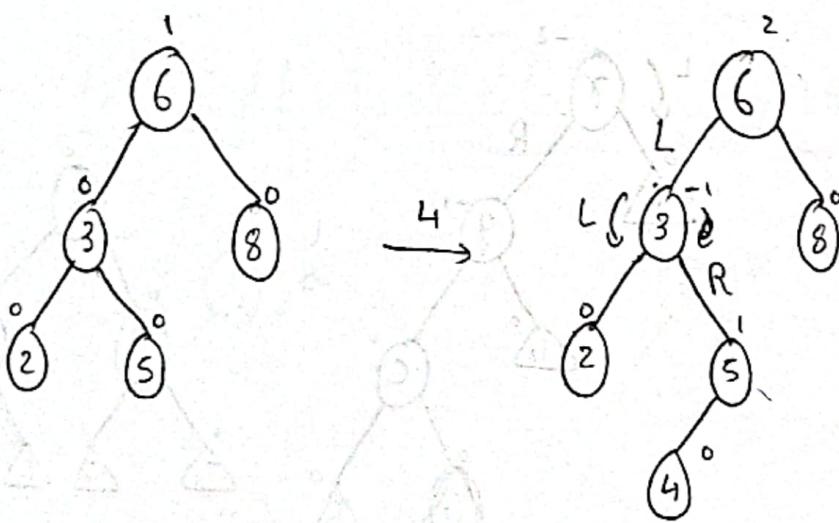
① 5, 6, 8, 3, 2, 4, 7



URotation of 5

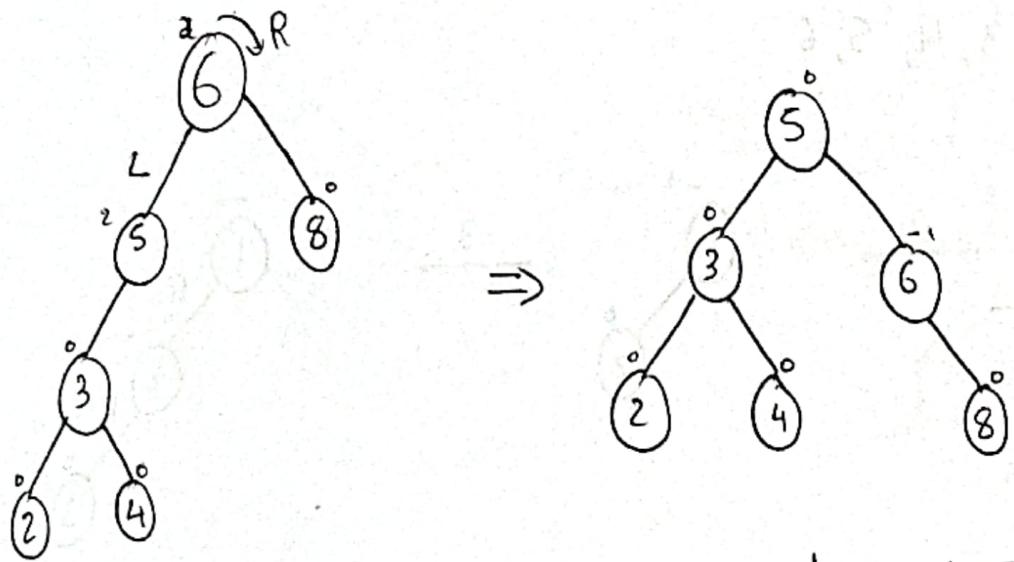


LL Problem of 5 (Right Rotation)

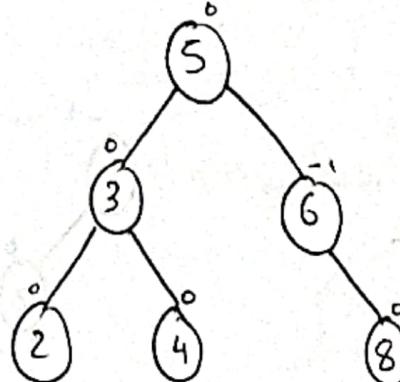


$\Rightarrow$  L-R Problem of 6

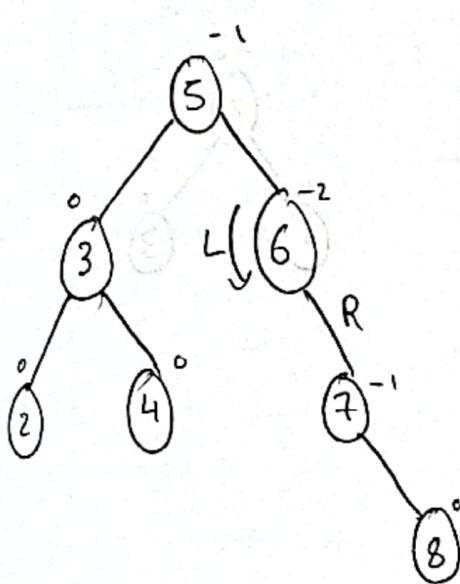
$\therefore$  LR Rotation



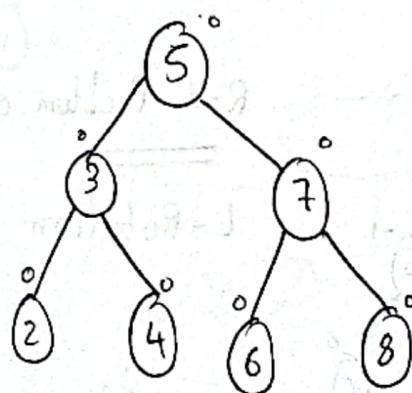
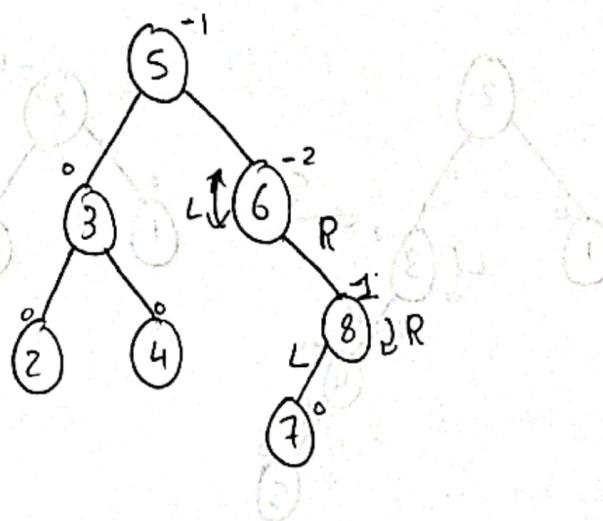
$\Rightarrow$



↓ Insert 7



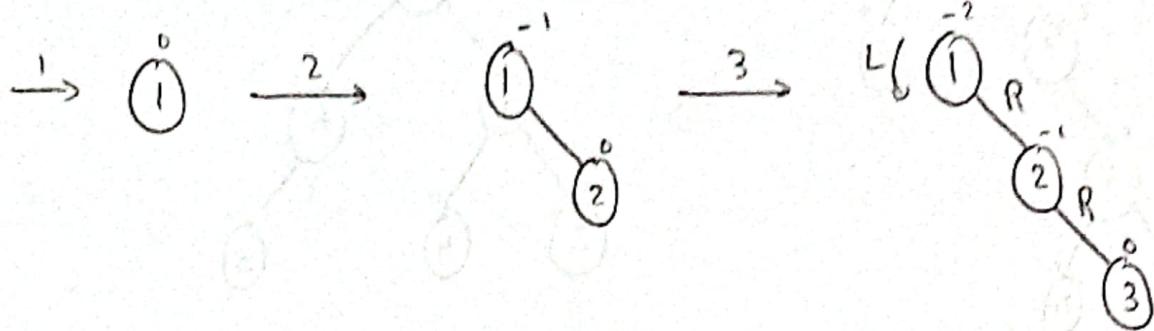
R-L Problem of 6  
R-L Rotation



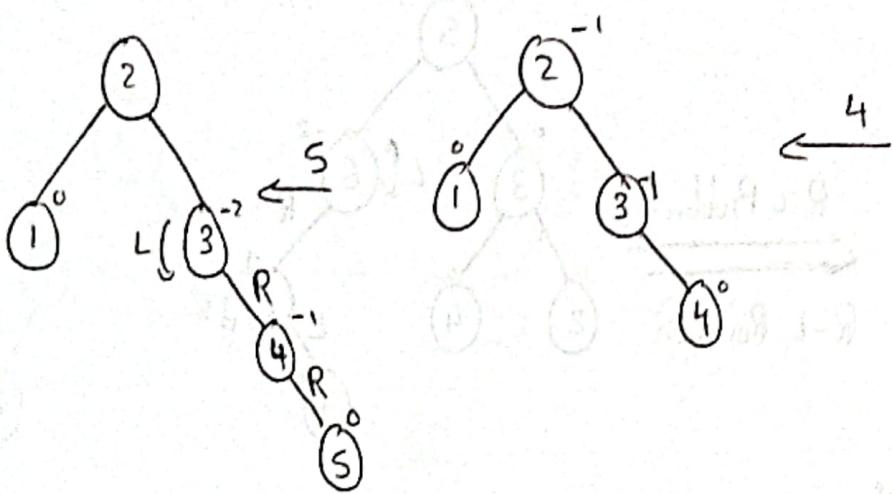
Fully balanced Binary Search Tree

ii

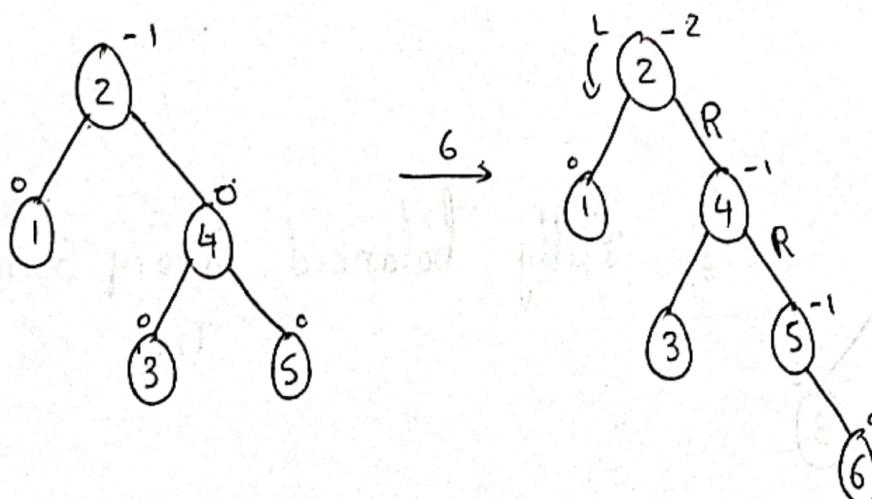
1, 2, 3, 4, 5, 6



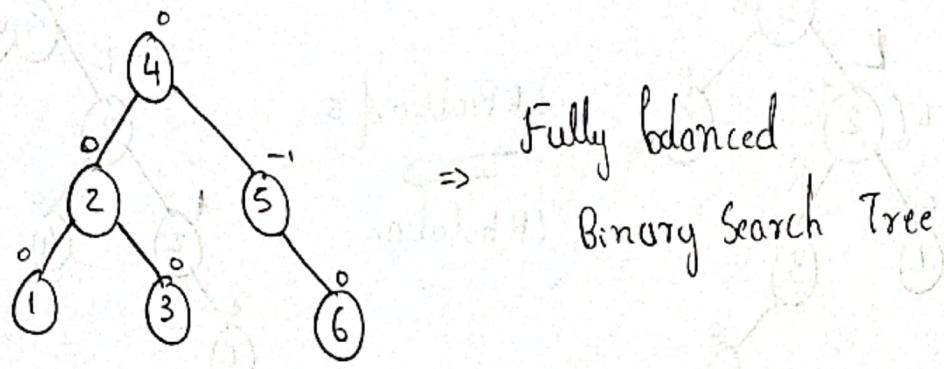
↓ RR Problem of '1'  
L-Rotation



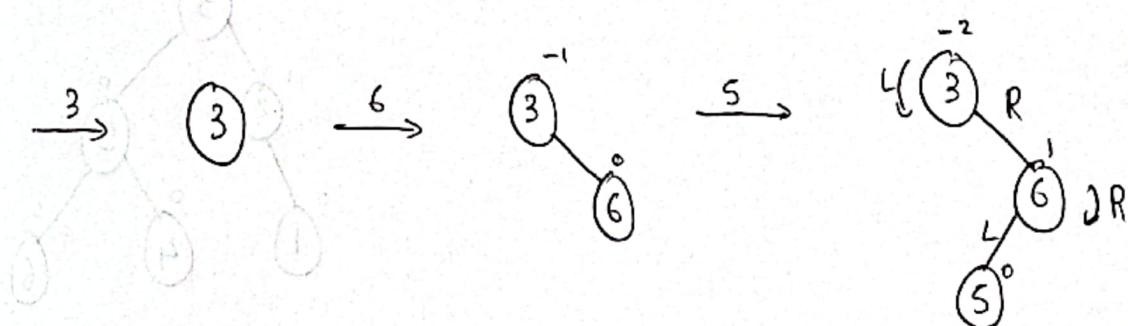
↓ RR Problem of '3'  
L-Rotation



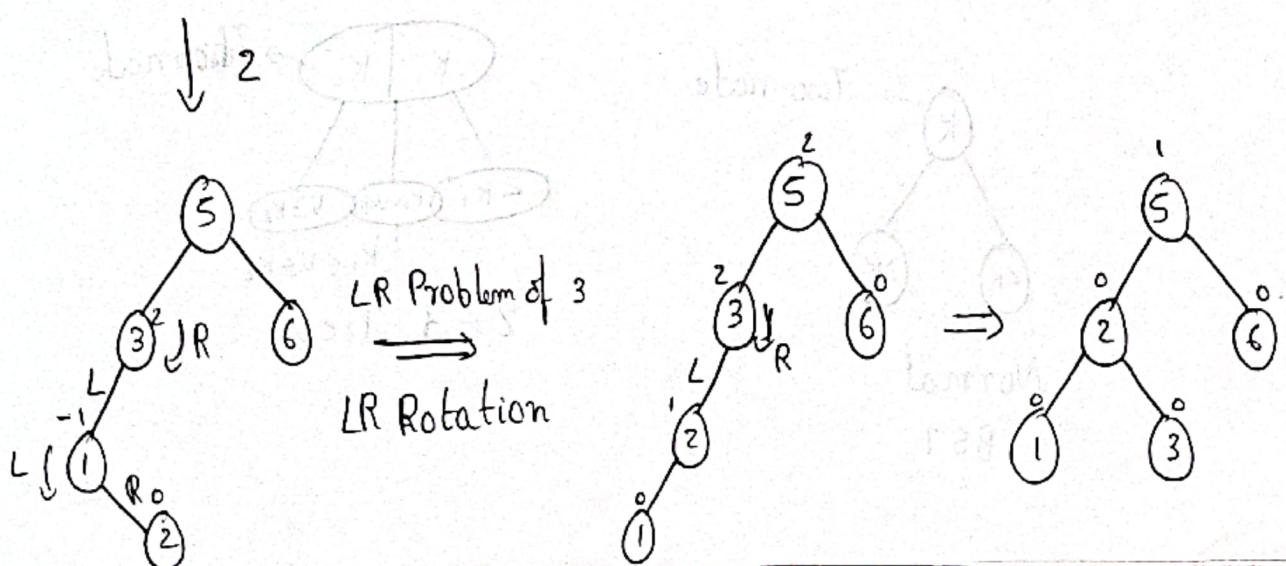
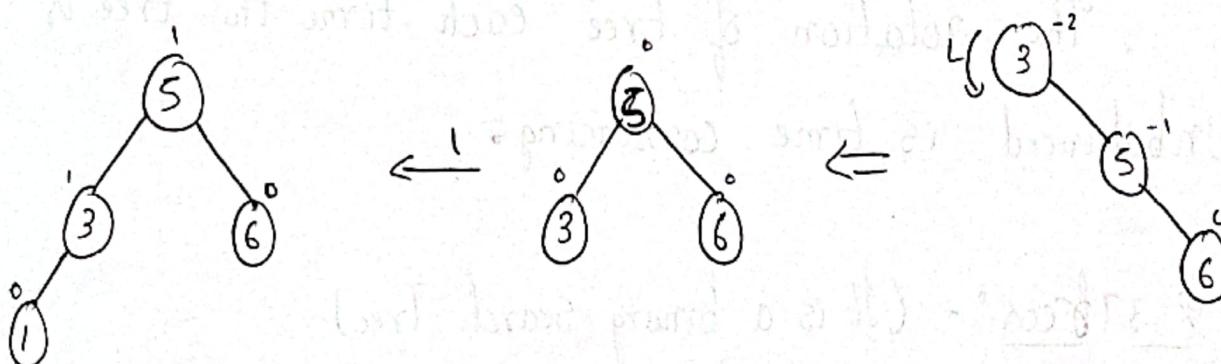
R-R Problem of '2'  
L-Rotation

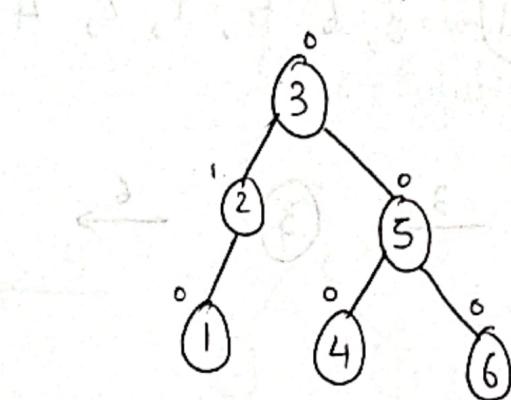
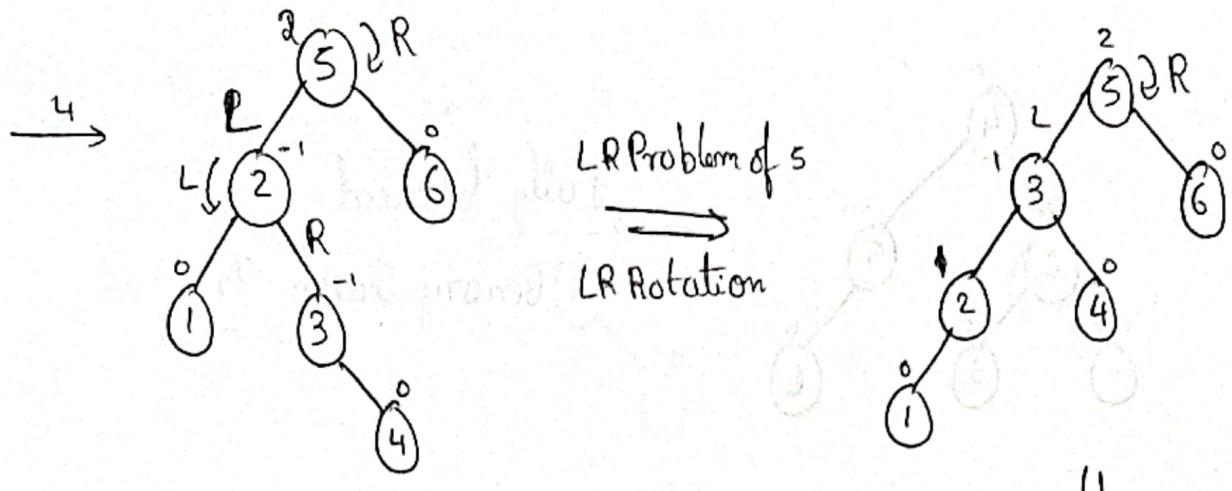


iii) 3, 6, 5, 1, 2, 4



RL Problem of 3  
RL Rotation

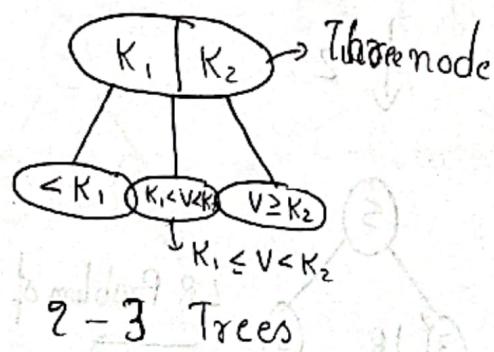
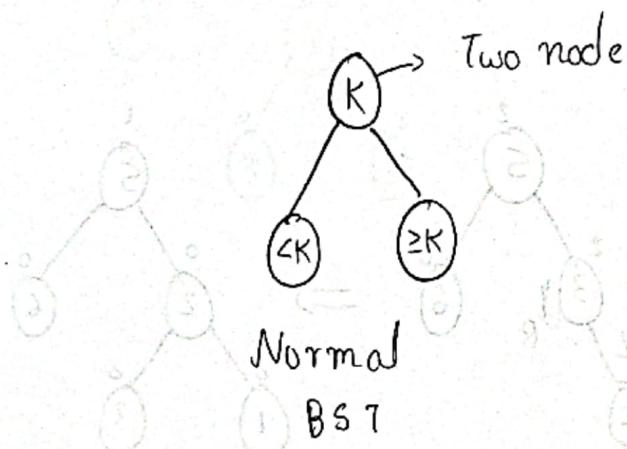




→ Major disadvantage of AVL Trees :-

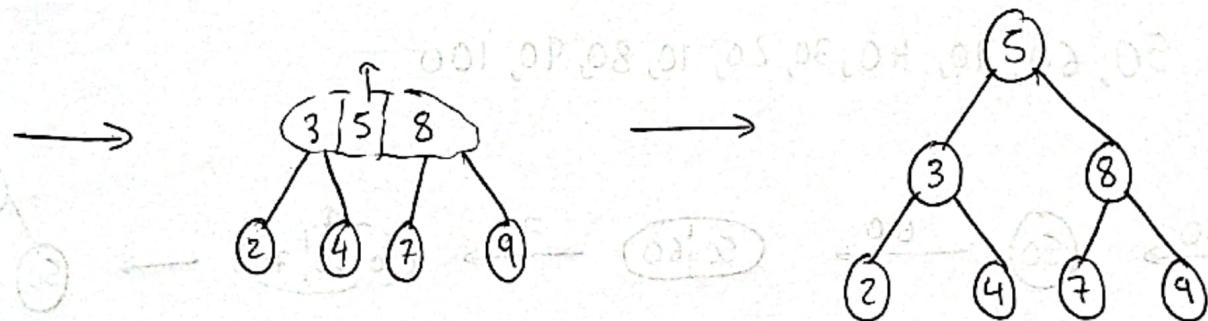
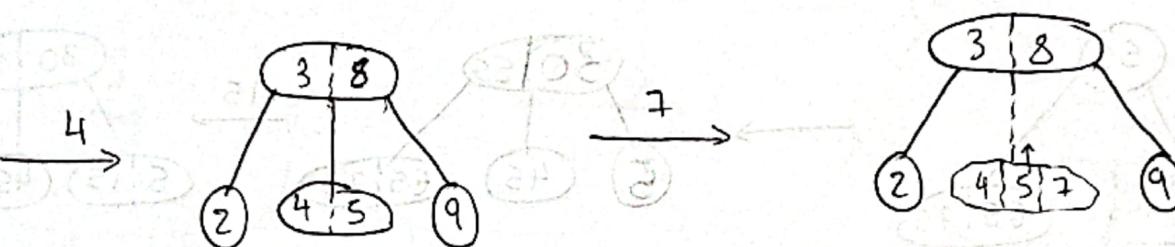
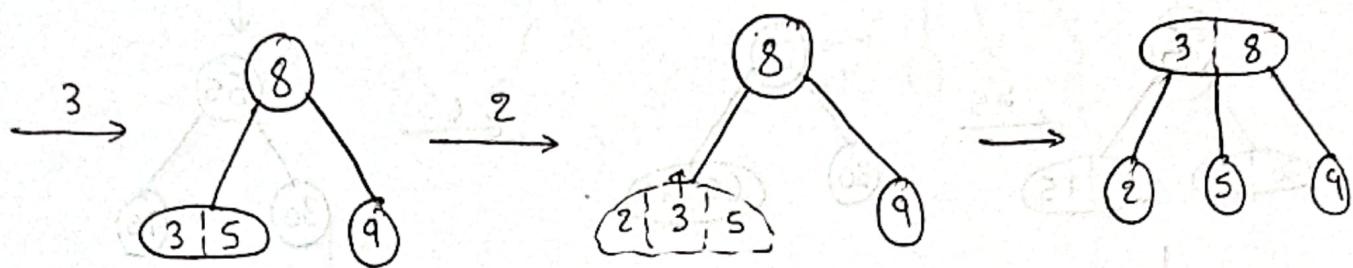
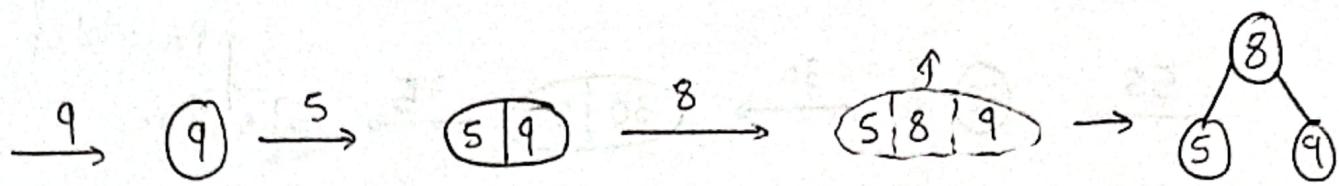
The rotation of tree each time the tree is unbalanced is time consuming.

→ 2-3 Trees :- (It is a binary search Tree)



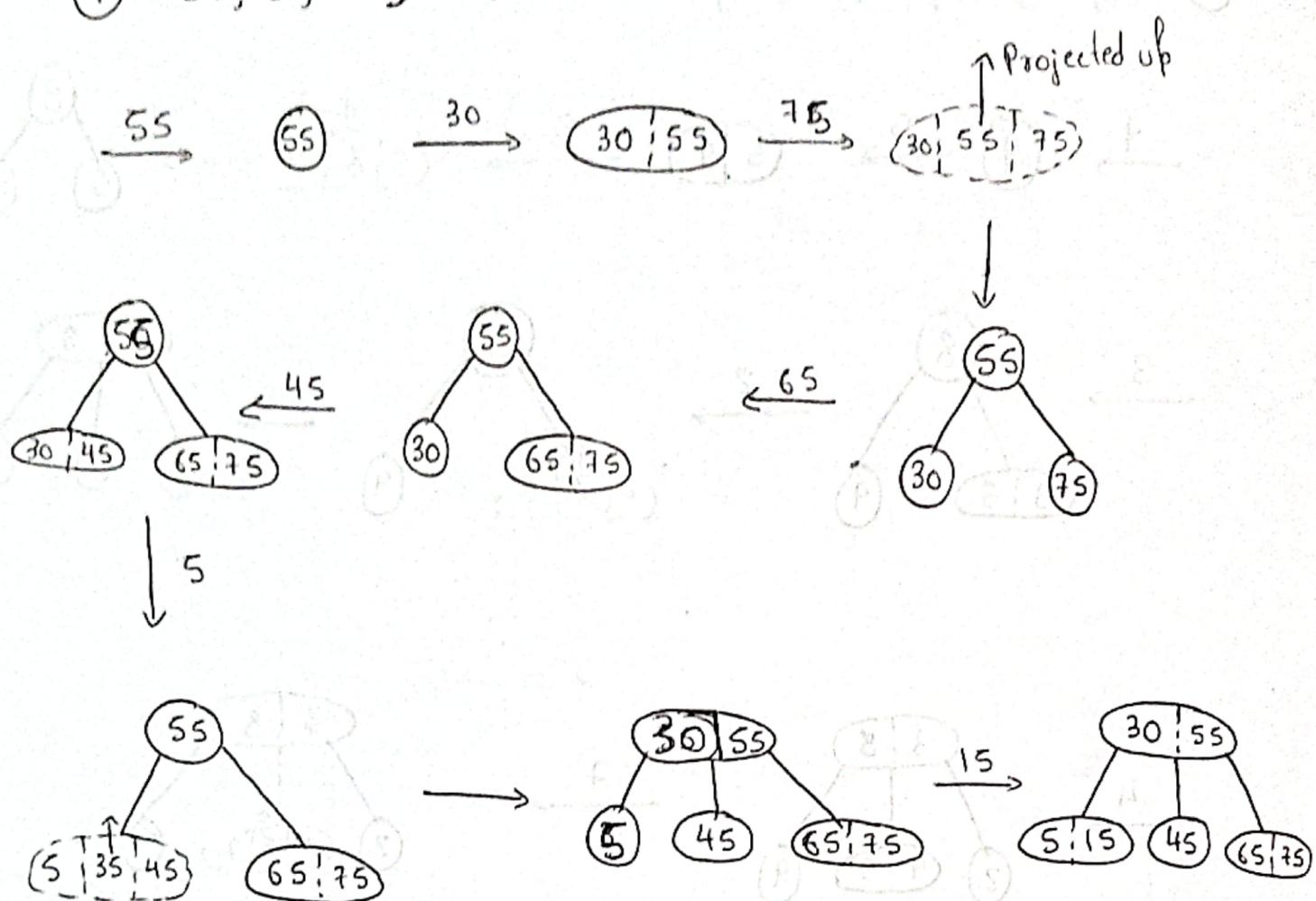
→ Construct a 2-3 Tree :-

- i) 9, 5, 8, 3, 2, 4, 7

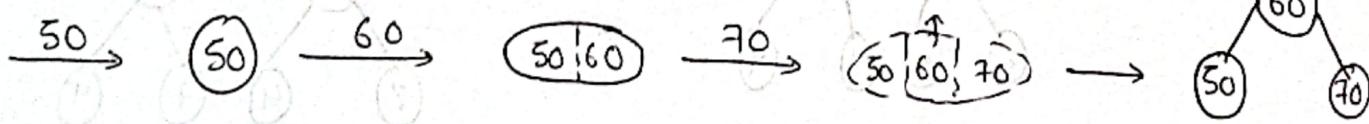
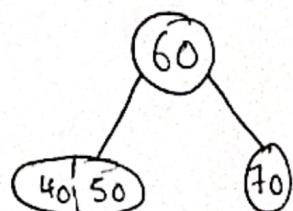
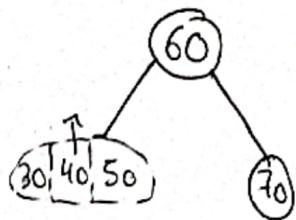
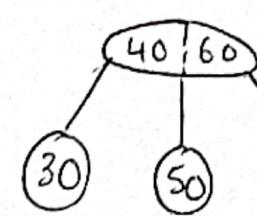


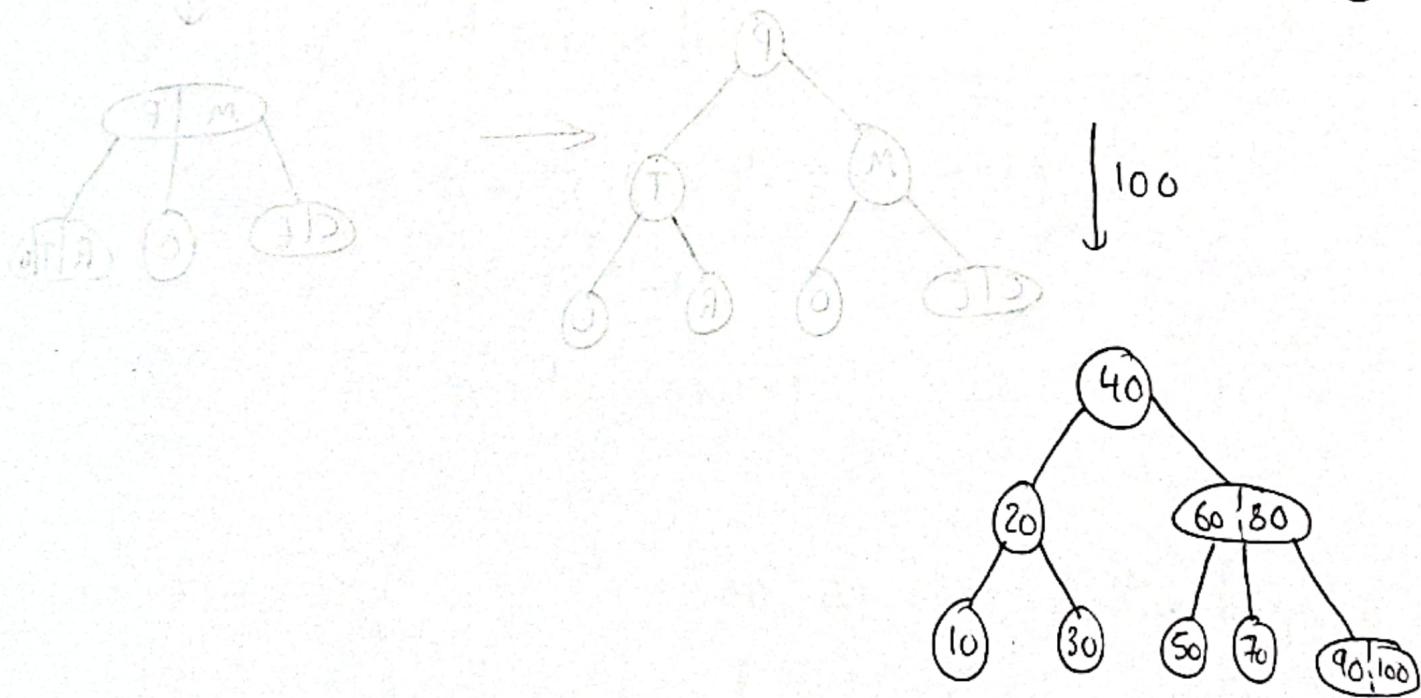
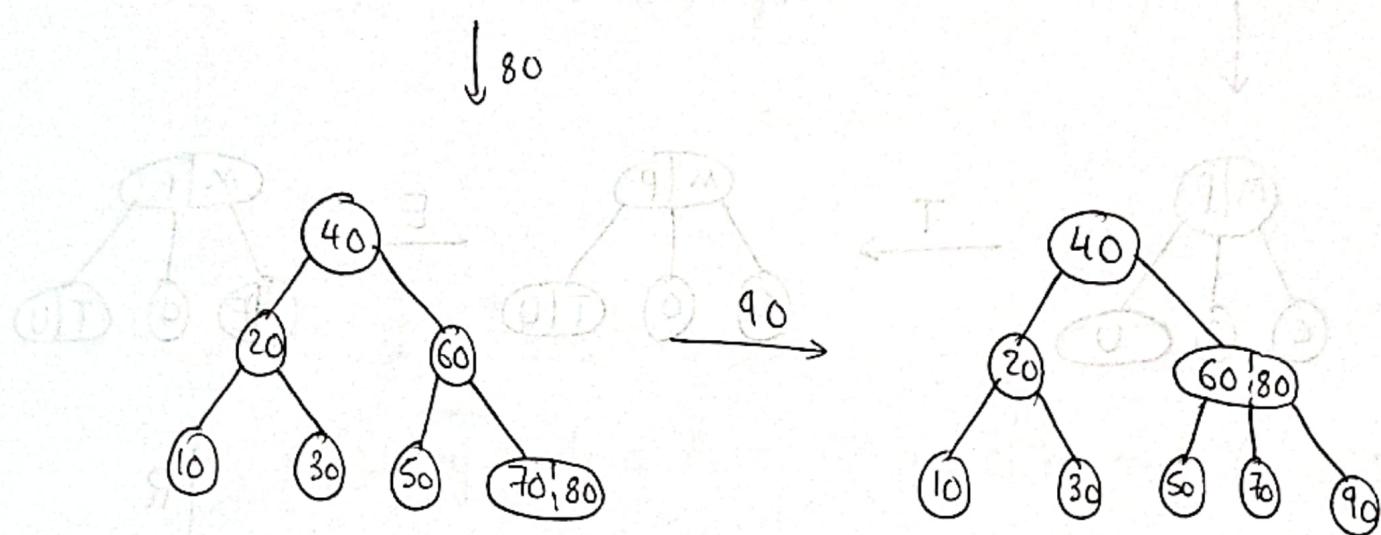
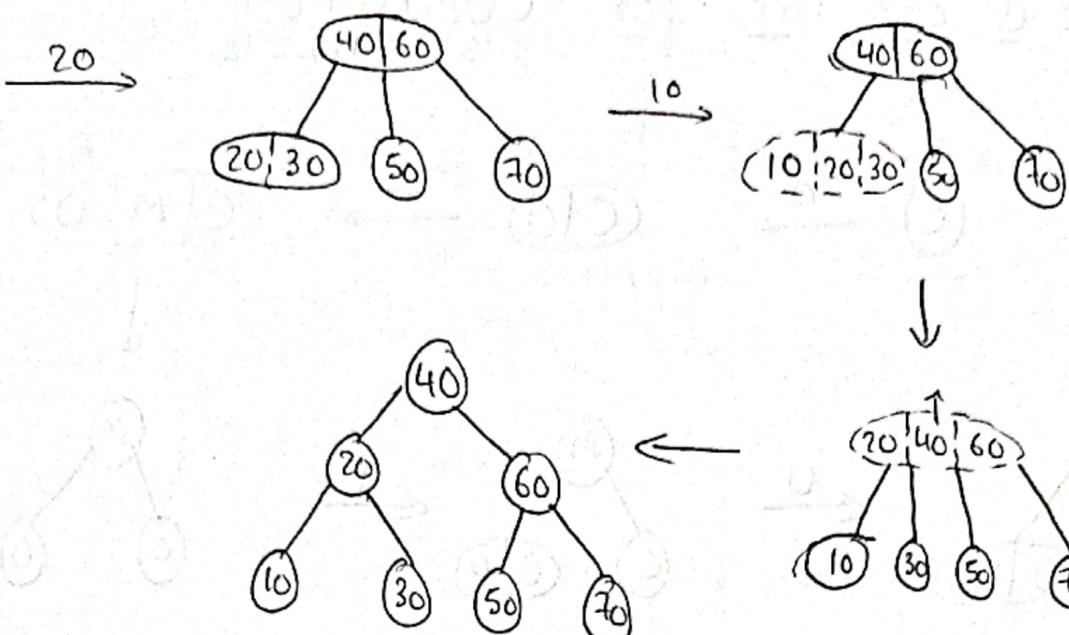
→ Construct a 2-3 Tree:-

(ii) 55, 30, 75, 65, 45, 5, 15



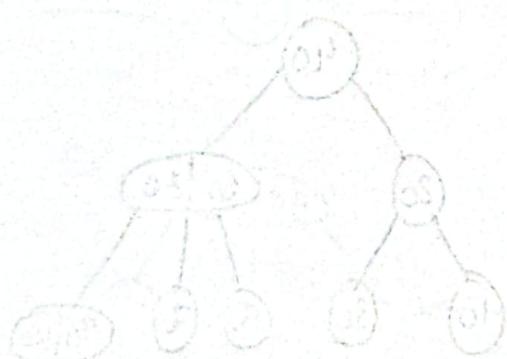
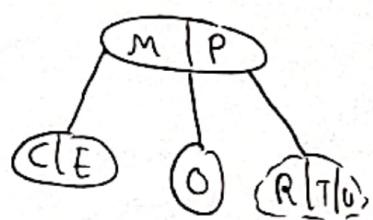
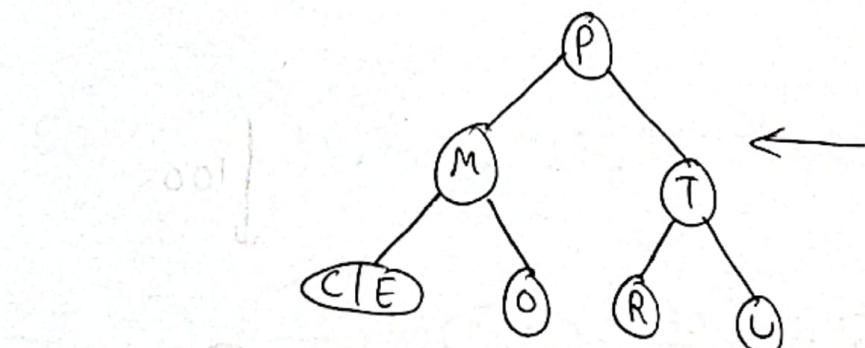
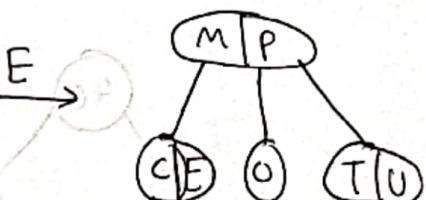
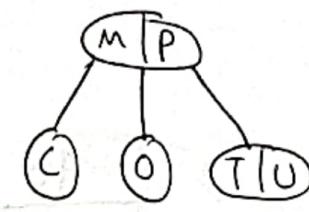
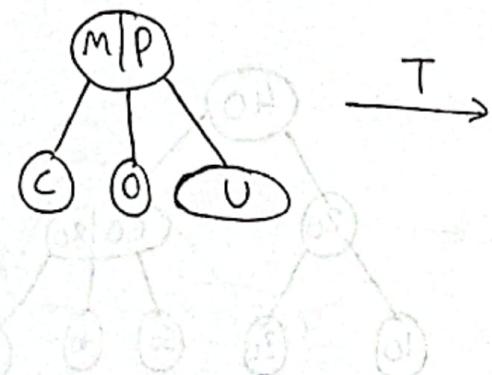
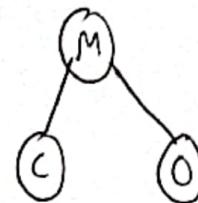
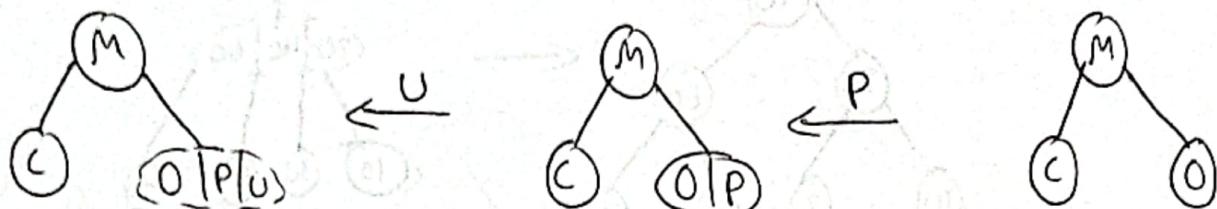
(ii) 50, 60, 70, 40, 30, 20, 10, 80, 90, 100





→ Construct a 2-3 Tree for COMPUTER

3 15 13 16 20 20 5 18



## Worst Case in 2-3 Trees:-

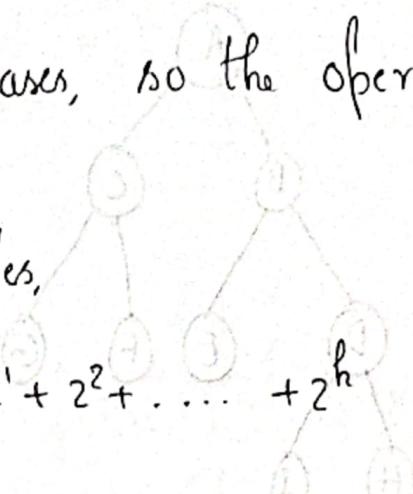
When every node in the tree is a two node then the height of tree increases, so the operations require more time.

Total number of nodes,

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h$$

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

$$h = \log_2(n+1) - 1$$



## Best Case in 2-3 Trees:-

When every node in tree is a three node (2-key & 3 child) then height of tree is less.

$$n = 2(3^0 + 3^1 + 3^2 + \dots + 3^h)$$

$$n = 3^{h+1} - 1$$

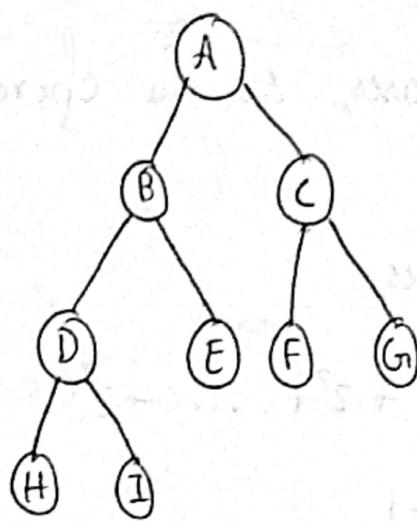
$$h = \log_3(n+1) - 1$$

∴ For any 2-3 Tree on an average, the height is

$$\log_2(n+1) - 1 \leq h \leq \log_3(n+1) - 1$$

∴ Efficiency class  $\in \Theta(\log n)$

→ Array Representation of Trees :-



1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I

→ Some properties :-

① First  $\frac{n}{2}$  location → Parent nodes

② For any parent,  $i \Rightarrow$  Left Child =  $2i$

③ For any parent,  $i \Rightarrow$  Right Child =  $2i+1$

④ For any child  $i \Rightarrow$  Parent =  $i/2$

⇒ Heap :-

It is a Data Structure i.e a Tree

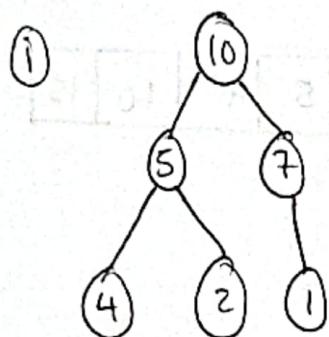
→ Two properties of Heap :-

① Trees shape requirement :- It is binary tree i.e complete i.e all level are full except possibly the last level, the rightmost leafs may be missing

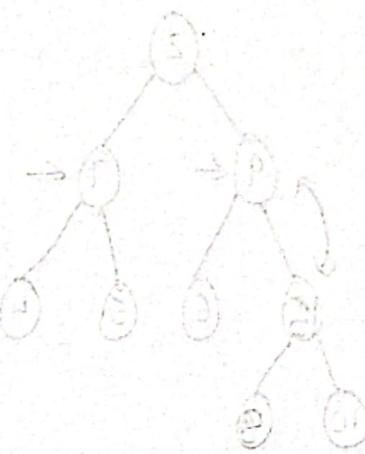
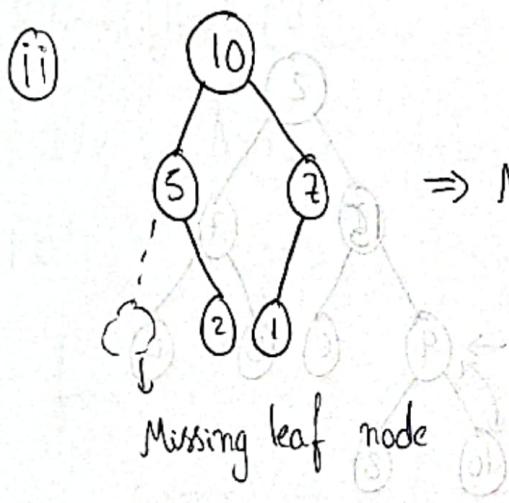
ii) It should follow Parental Dominance:-

The key at each node is greater than or equal to the keys at its children.

→ Example :-



The path from root to any leaf node gives elements in sorted order



Note :- In a heap,

For any 'i' such that  $1 \leq i \leq \lfloor n/2 \rfloor$ ,

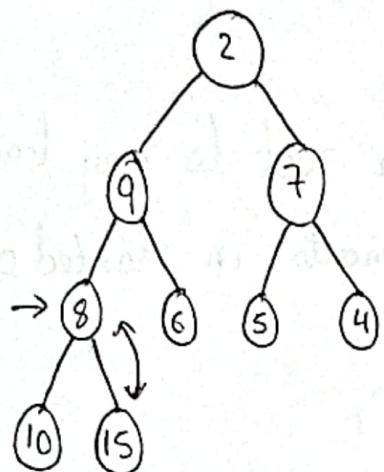
$$H[i] \geq \max \{ H[2i], H[2i+1] \}$$

→ Construct a heap for the values :-

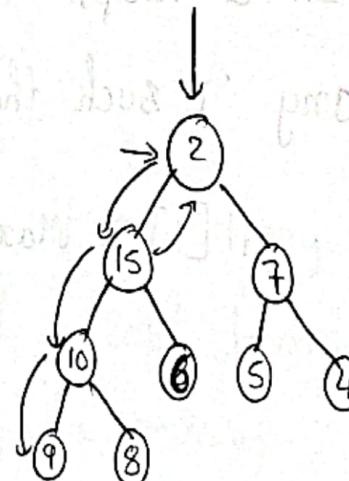
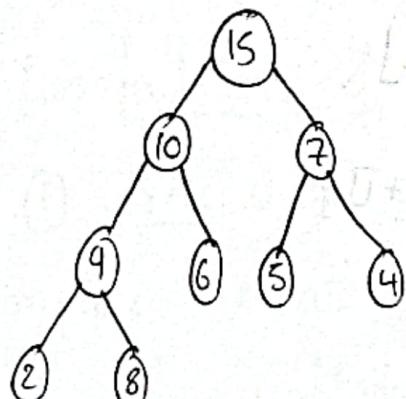
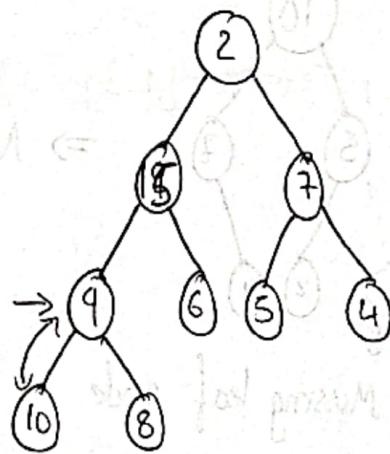
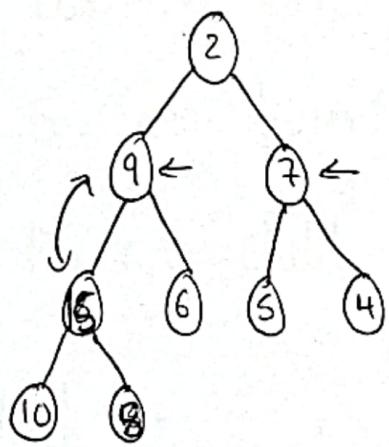
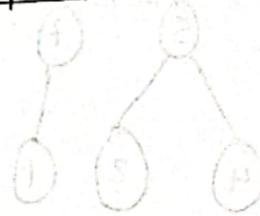
2, 9, 7, 8, 6, 5, 4, 10, 15

→ Bottom Up Approach :- (Phase - I)

Step 1:- Construct a Complete BT



2	9	7	8	6	5	4	10	15
---	---	---	---	---	---	---	----	----



Array :-

2	9	7	8	6	5	4	10	15
2	9	7	15	6	5	4	10	8
2	9	7	15	6	5	4	10	8
2	15	7	9	6	5	4	10	8
2	15	7	10	6	5	4	9	8
15	2	7	10	6	5	4	9	8
15	20	7	2	6	5	4	9	8
15	10	7	9	6	5	4	9	8

Algorithm :-

BottomUpHeap ( $H[1 \dots n]$ )

|| I/p : Array  $H[1 \dots n]$

|| O/p : Heap  $H[1 \dots n]$

for  $i \leftarrow n/2$  down to 1 do

    parent  $\leftarrow i$

    P  $\leftarrow H[\text{parent}]$

    heap  $\leftarrow \text{false}$

    while not heap and  $2 * \text{parent} \leq n$  do

        child  $\leftarrow 2 * \text{parent}$

```

    if child < n
        if H[child+1] > H[child]
            child ← child + 1
        end if
    end if

```

```

if P > H[child]

```

```

    heap ← true

```

```

else

```

```

    H[parent] ← H[child]

```

```

    parent ← child

```

```

end while

```

```

H[Parent] ← P

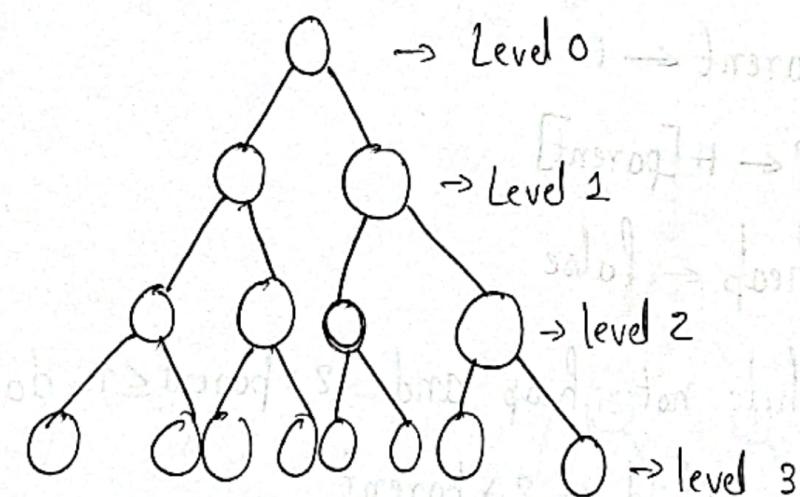
```

```

end for

```

$\Rightarrow$  Efficiency Analysis :-



To Full heap Tree.

For a full Tree of level 'k', the number of tree nodes in the tree is

$$n = 2^{k+1} - 1$$

The height of tree is,

$$h = \lfloor \log_2 n \rfloor$$

$$h =$$

For ex:-

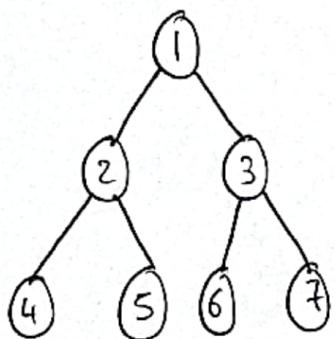
$$n = 15$$

$$h = \lfloor \log_2 15 \rfloor = \lfloor 3.9 \rfloor = 3$$

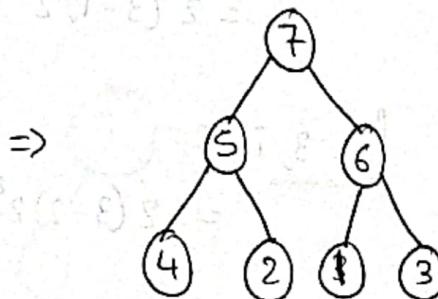
④ else

$$h = \lceil \log_2(n+1) \rceil - 1$$

$$h = \lceil \log_2 (16) - 1 \rceil = 3$$



Min Heap



Max heap.

- ⑤ Worst Case in when we have to convert a min heap into max heap which takes maximum comparisons each time.

For moving any key in level 'i' requires  
 $2(h-i)$  comparisons

$$C_{\text{worst}}(n) = \sum_{i=0}^{h-1} \sum_{\text{level } i \text{ Keys}} 2(h-i)$$

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

$$\vdots$$

$$2^1 = 2$$

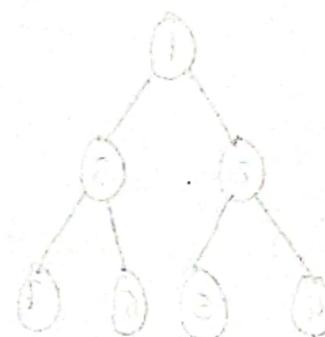
$$= 2(n - \log_2(n+1))$$

For  $h=3, i=0$   
 $= 2(3-0)2^0 = 6$

For  $h=3, i=1$   
 $= 2(3-1)2^1 = 8$

For  $h=3, i=2$   
 $= 2(3-2)2^2 = 8$

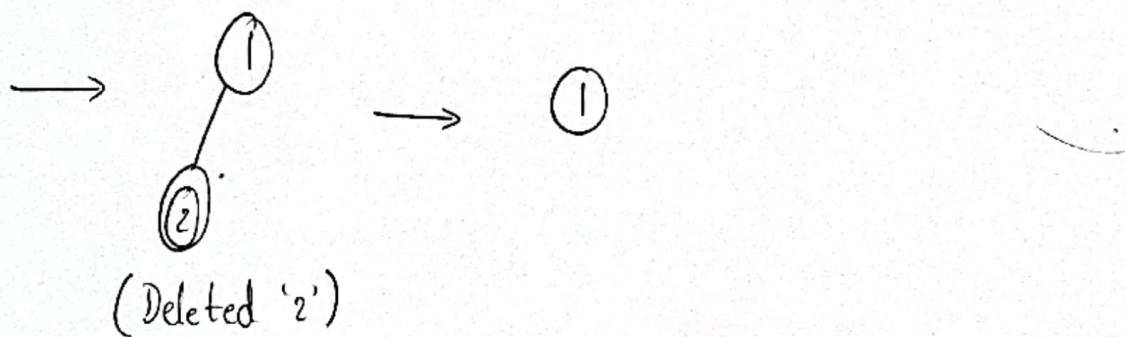
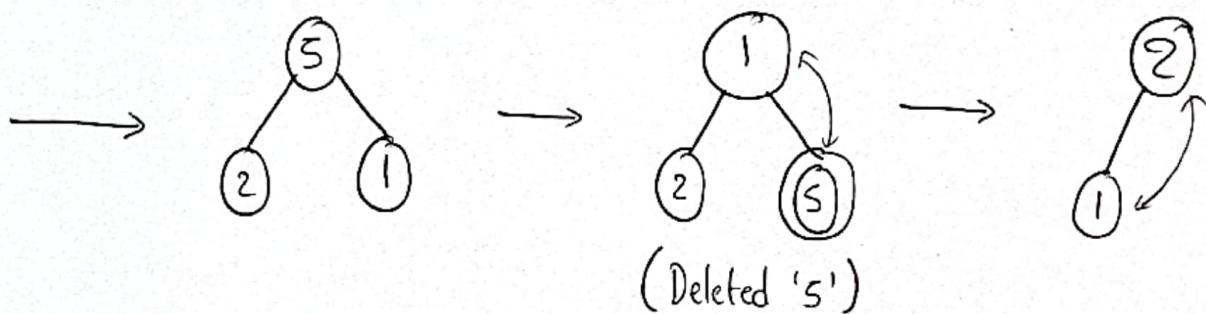
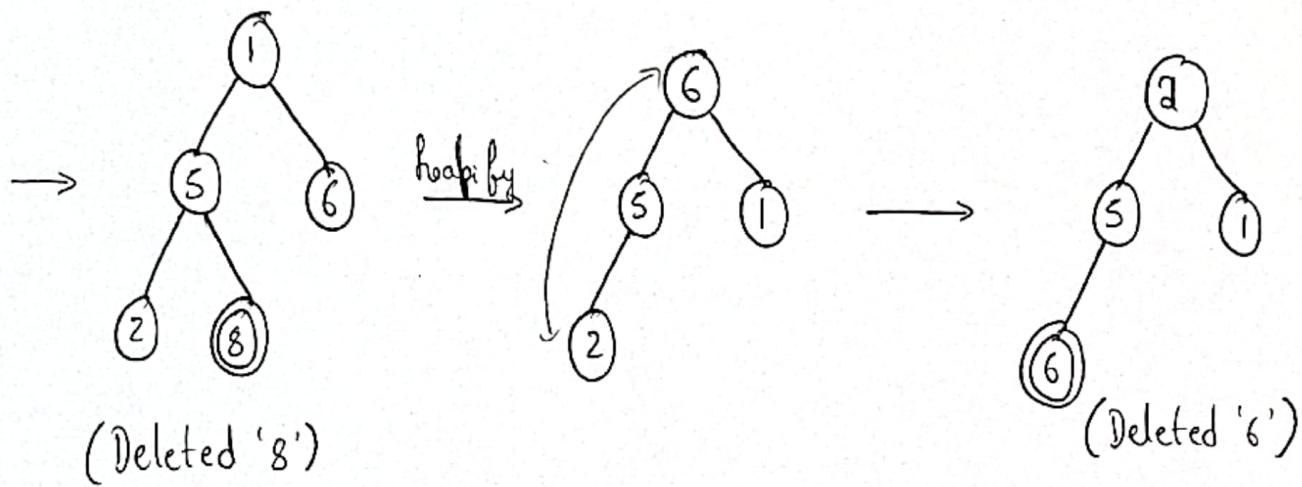
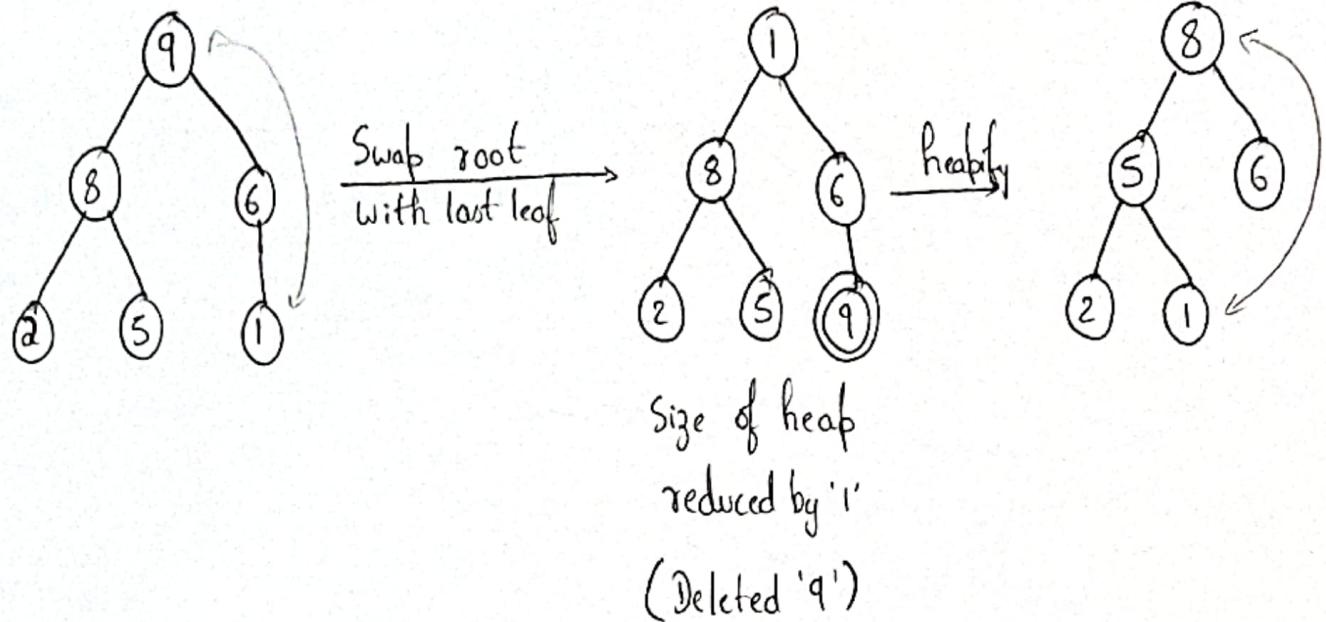
For  $h=3, i=3$   
 $= 2(3-3)2^3 = 0$



∴ Total number of comparison =  $6 + 8 + 8 + 0 = 22$

From Formula,  
 $\text{Total no. of comparison} = 2(15 - \log_2(6)) = 2(11) = 22$

$\Rightarrow$  Heap Deletion :- (Phase - II)



$\Rightarrow$  Sort the given numbers using Heap Sort.

① 2 9 7 6 5 8

$\rightarrow$  Construction :-

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 9 & 7 & 6 & 5 & 8 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 9 & 8 & 6 & 5 & 7 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 2 & 8 & 6 & 5 & 7 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 9 & 6 & 8 & 2 & 5 & 7 \end{matrix}$

$\rightarrow$  Deletion

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ P & S & P & S & P & S & P & S \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 6 & 8 & 2 & 5 & 5 & | & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 6 & 7 & 2 & 5 & 5 & | & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 5 & 6 & 7 & 2 & | & 8 & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 6 & 5 & 2 & | & 8 & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 6 & 5 & 1 & 7 & 8 & | & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 2 & 5 & | & 7 & 8 & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 5 & 2 & | & 6 & 7 & 8 & 9 \end{matrix}$

$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & | & 5 & 6 & 7 & 8 & 9 \end{matrix}$

Sorted Array :- 2 5 6 7 8 9

Analysis:-

Order of growth of Construction :-  $\Theta(n)$

Order of growth of Deletion :-  $\Theta(n \log n)$

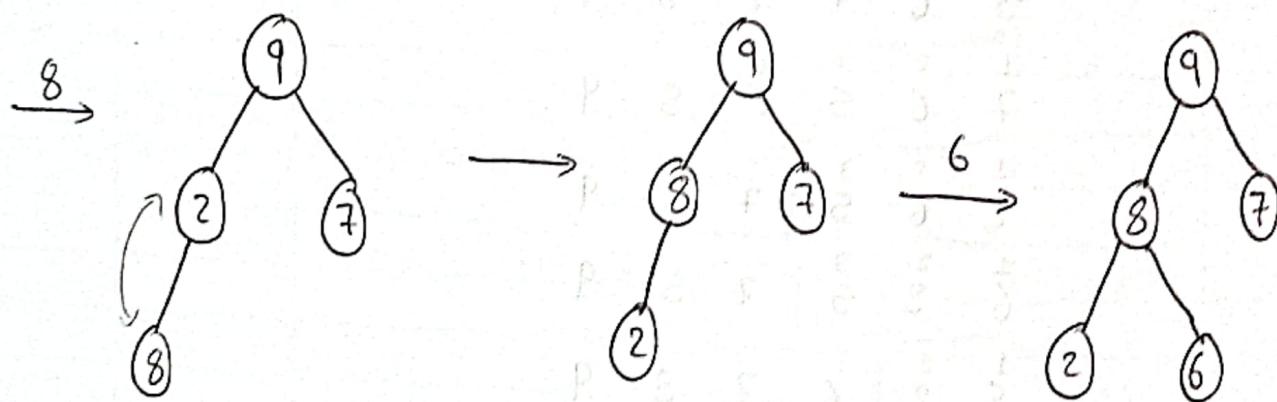
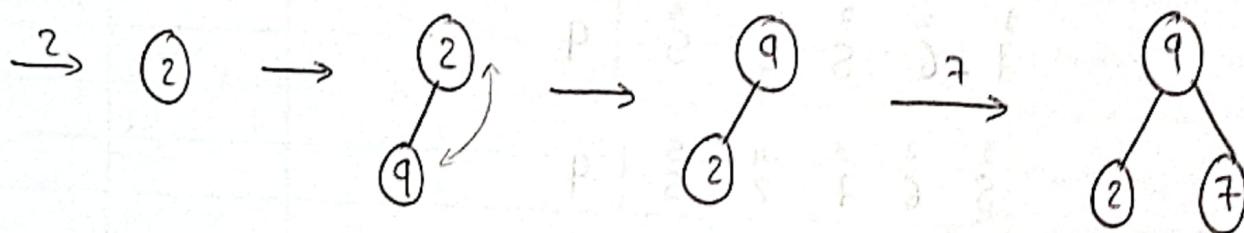
∴ Order of growth of Heapsort

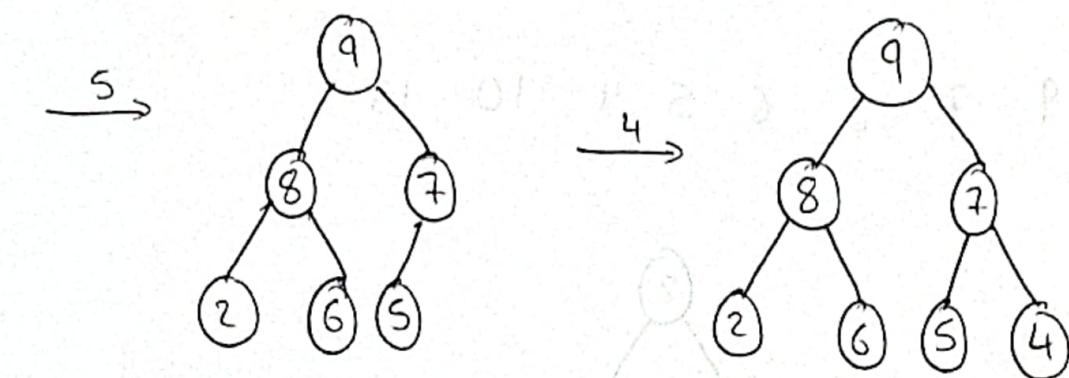
$$= \Theta(n) + \Theta(n \log n)$$

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

$\Rightarrow$  Top - Down Approach of Heap Construction :-

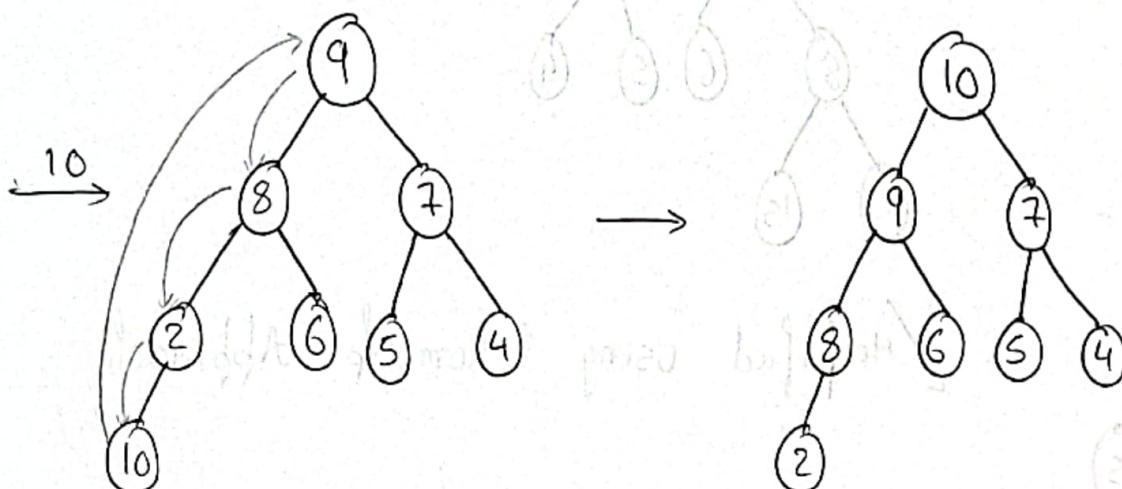
2 9 7 8 6 5 4 10 15





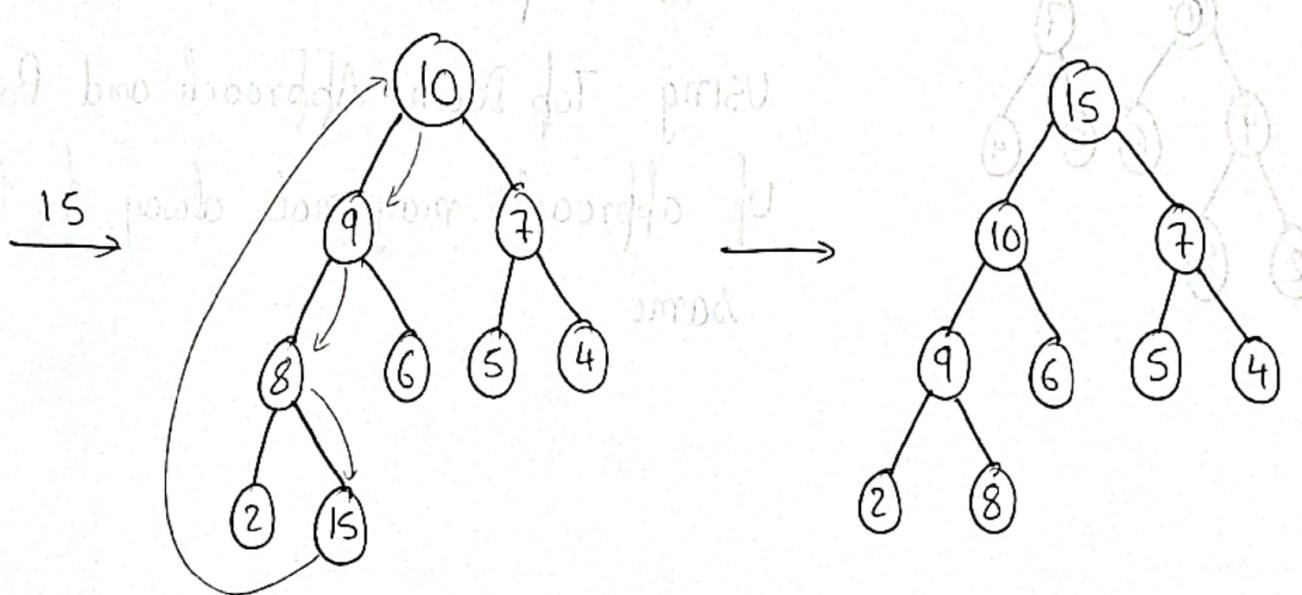
5 →

4 →



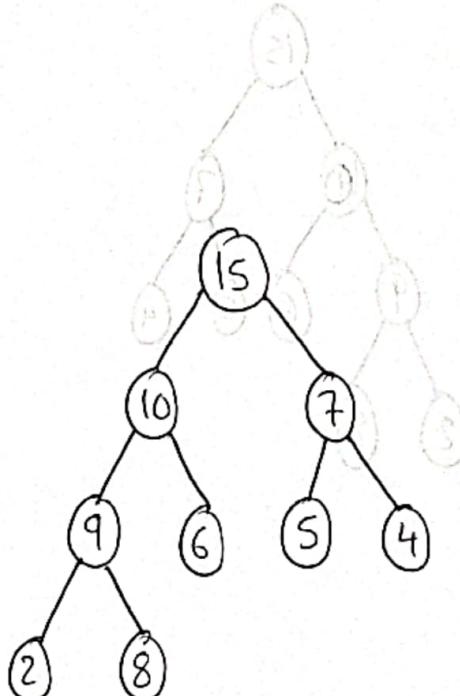
10 →

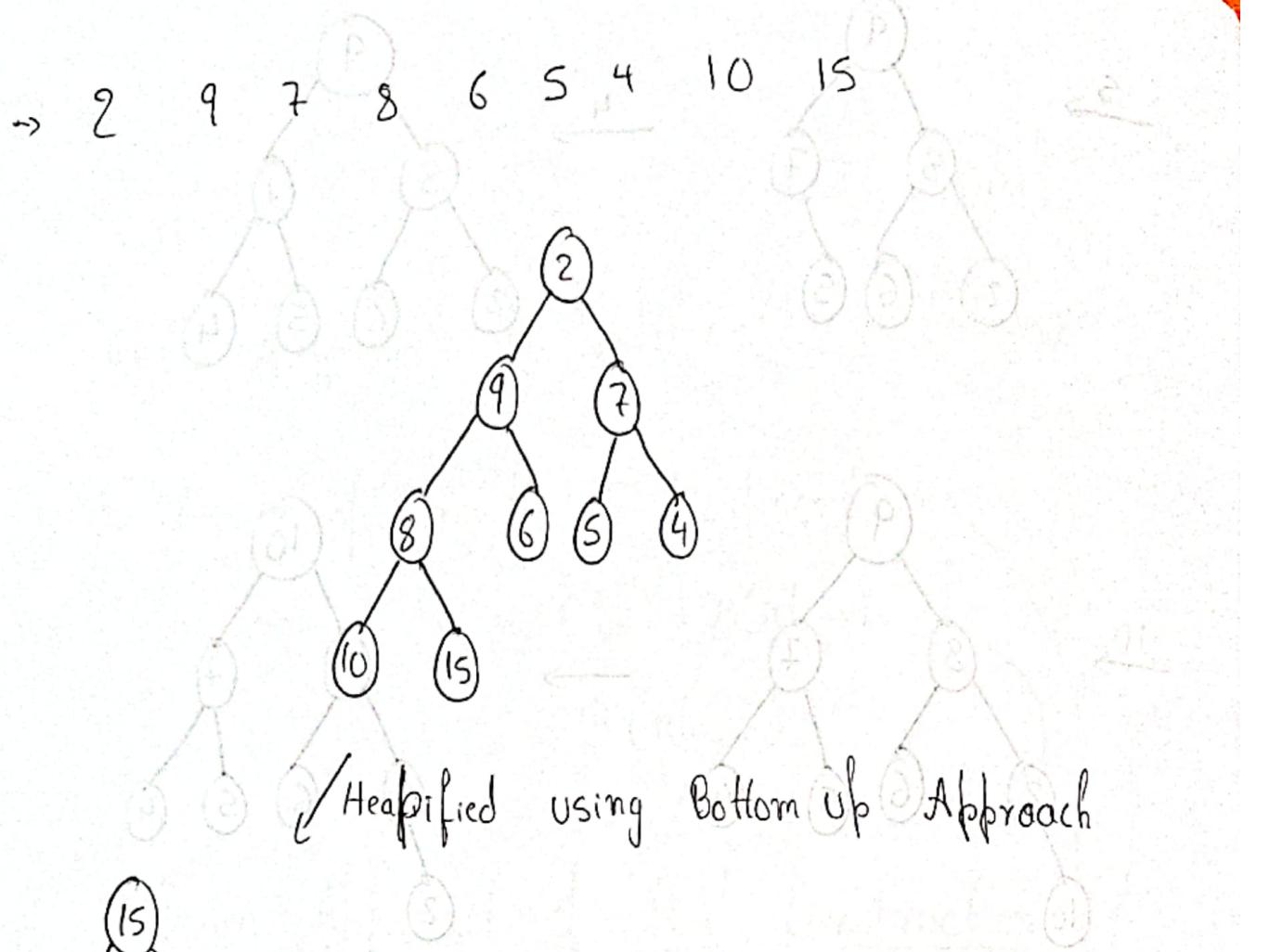
→



15 →

→

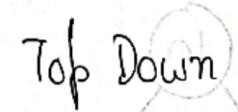
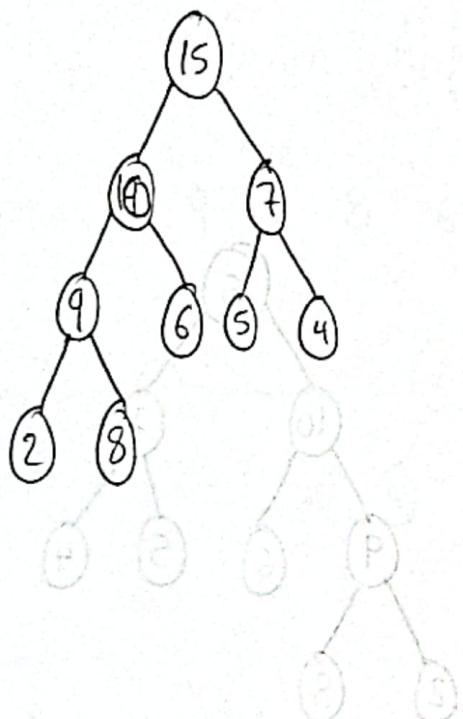




The heap trees constructed

using Top Down Approach and Bottom

Up approach may not always be the  
same



## $\Rightarrow$ Space to Time Tradeoff :-

Comprising on space to increase the efficiency of  
in Algorithm

$\rightarrow$  Two approaches :-

(i) Input Enhancement :-

$\rightarrow$  Counting - Sorting (Comparing Counting Sorting)

$\rightarrow$  Horspool's String Matching Algorithm

(ii) Pre Structuring :-

$\rightarrow$  Hashing

### Counting Sorting Algorithm

Array : 3, 2, 15, 18, 20, 13, 6

The number of elements less than the element in the array

3  $\rightarrow$  0      2  $\rightarrow$  0      15  $\rightarrow$  4      18  $\rightarrow$  5      20  $\rightarrow$  6      13  $\rightarrow$  3      6  $\rightarrow$  2

Sorted Array :-      0      1      2      3      4      5      6  
                        2      3      6      13     15     18     20

A :-

15	7	6	8	12	1
----	---	---	---	----	---

Pass - 0	0	0	0	0	0	0
Pass - 1	5	0	0	0	0	0
Pass - 2	-	2	0	1	1	0
Pass - 3	-	-	1	2	2	0
Pass - 4	-	-	-	3	3	0
Pass - 5	-	-	-	-	4	0
	-	-	-	-	-	0

Final Count array

0	1	2	3	4	5
5	2	1	3	4	0

Sorted array :-

0	1	2	3	4	5
1	6	7	8	12	15

Algorithm :-

Comparing Counting ( $A[0 \dots n-1]$ )

|| I/P :- An unsorted array  $A[0 \dots n-1]$

|| O/P :- A sorted array

[ for  $i \leftarrow 0$  to  $n-1$  do

    count [ $i$ ]  $\leftarrow 0$

end for.

[ for  $i \leftarrow 0$  to  $n-2$  do

    for  $j \leftarrow i+1$  to  $n-1$  do

        if  $A[i] < A[j]$

            count [ $j$ ]  $\leftarrow$  count [ $j$ ] + 1

        Else

            count [ $i$ ]  $\leftarrow$  count [ $i$ ] + 1

    end for

end for

[ for  $i \leftarrow 0$  to  $n-1$  do

$s[\text{count}[i]] \leftarrow A[i]$

end for

return  $s$

Analysis :-

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

$$= \sum_{i=0}^{n-2} n-1-i-x+x$$

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

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

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

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

$$= \frac{(n-1)n}{2} \approx \frac{n^2}{2}$$

$$C(n) \in \Theta(n^2)$$

④ Although we are compromising on space, the time efficiency of algorithm has not improved.

⑤ It is very similar to Selection Sort but inferior to it since Selection Sort is inplace.

$\Rightarrow$  Distribution-Sorting Algorithms -<sup>Counting</sup>

A : 13 11 12 13 12 12

{11, 12, 13}

Array	11	12	13
Frequency	1	3	2
Distribution	1	4	6

Sorted Array

	0	1	2
A[5]	1	4	6
A[4]	1	3	6
A[3]	1	2	6
A[2]	1	2	5
A[1]	1	1	5
A[0]	0	1	5

	0	1	2	3	4	5
				12		
					13	
			12			
		11				
				13	13	

Final Sorted array

11	12	12	12	13	13
----	----	----	----	----	----

## $\Rightarrow$ Distribution Counting ( $A[0 \dots n-1], l, u$ )

[ for  $i \leftarrow 0$  to  $u-1$

$D[i] \leftarrow 0$

] end for

[ for  $i \leftarrow 0$  to  $n-1$  do

$D[A[i]-l] \leftarrow D[A[i]-l] + 1$

] end for

[ for  $i \leftarrow 1$  to  $u-1$  do

$D[i] \leftarrow D[i] + D[i-l]$

] end for

[ for  $i \leftarrow n-1$  downto  $0$  do

$j \leftarrow A[i]-l$

$S[D[j]-l] \leftarrow A[i]$

$D[j] \leftarrow D[j] - 1$

] end for

return  $S$

## Input Enhancement :-

⇒ Horspool's String Matching :-

Preprocess the Pattern and prepare a Shift Table.

Pattern :- BARBER

String :-  $S_0 \dots S_{n-1}$   
BARBER

The comparison is done from right to left.

Case-i):-

$S_0 \dots S_{n-1}$   
BARBER

The character of text 'S' aligned with the last character of the Pattern does not exist in Pattern at all.

∴ shift 'm' characters.

Case-ii):-

$S_0 \dots B \dots S_{n-1}$   
→ BARBER  
→ BARB ER

The character of text aligned with last character of Pattern does not match last but exists in the pattern.

case (iii) :-

The last character of the Pattern is matching with a character 'c' in the String text but there is a mismatch in between and the character 'c' does not exist again in Pattern.

∴ Shift :- m characters.

..... MER .. .  
LEADER  
→ LEADER

case (iv) :-

The last character of Pattern is matching with a character 'c' in the String text but there is a mismatch in between and the character 'c' exists again in Pattern.

..... MER .. .  
REOR D E R  
→ REORDER

$\Rightarrow$  Creating Shift Table:-

Formula,

$$f(c) = \begin{cases} m, & c \notin \text{Pattern} \\ m - 1 - i, & c \in \text{Pattern}, i \rightarrow \text{index of } 'c' \\ & \text{in Pattern.} \end{cases}$$

Shift Table:-

A	B	C	D	E	.....	R	.....	-
4	2	6	6	1	.....	3	.....	6

Example:-

Text:- JIM\_SAW\_ME\_IN\_A\_BARBER\_SHOP  
(shift - 4)

Pattern:- BARBER  
BARBER  
BARBER  
(shift - 1)  
(shift - 6)

BARBER  
BARBER  
BARBER  
BARBER  
(shift - 2)  
BARBER  
(shift - 3)

Total  $\rightarrow 16$

$\therefore$  The algorithm returns 16

Ex ② :-

String :- BESS\_KNEW\_ABOUT\_BAOBABs

Pattern :- BAOBAB

Shift Table

A	B	.	.	O	.	.	-
1	2			3			6

BESS\_KNEW\_ABOUT\_BAOBABs

BAOBAB

Shift - 6

(E-1) BAOBAB

Shift - 2

BAOBAB

Shift - 6

BAO**BAB**

Shift - 2

BAOBAB

Total Shift  $\rightarrow$  16

BAOBAB

Index of String Matching Pattern :- 16

$\Rightarrow$  Algorithm :-

$\Rightarrow$  Shift Table ( $P[0 \dots m-1]$ )

Initialize all the Table entries with 'm'

[ for  $j \leftarrow 0$  to  $m-2$

    Table [ $P[j]$ ]  $\leftarrow m-1-j$

] end for

return Table

$\Rightarrow$  Horspool Matching ( $P[0 \dots m-1], T[0 \dots n-1]$ )

Shift Table ( $P[0 \dots m-1]$ )

$i \leftarrow m-1$

[ while  $i \leq n-1$  do

$K \leftarrow 0$

    [ while  $K \leq m-1$  and  $P[m-1-K] = T[i-K]$  do

$K \leftarrow K + 1$

    ] end while

    if  $K = m$

        return  $i - m + 1$

    Else

$i \leftarrow i + \text{Table}[T[i]]$

] end while

return -1

## ⇒ Hashing :-

① Distribution of keys over an 1-D array called as a Hash Table.

② Hashing is done using an Hash function

$$h(k) = k \bmod m \quad (\text{Remainder hash function})$$

generally prime

③ Hash function returns the hash address

Ex:-

Values:- 13, 15, 18, 19, 31, 91, 56

$$m = 7$$

$$h(13) = 13 \bmod 7 = 6$$

$$h(15) = 15 \bmod 7 = 1$$

$$h(18) = 18 \bmod 7 = 4$$

$$h(19) = 19 \bmod 7 = 5$$

$$h(31) = 31 \bmod 7 = 3$$

$$h(91) = 91 \bmod 7 = 0$$

$$h(56) = 56 \bmod 7 = 0$$

Collision:- Two keys having same hash address.

Usually occurs when  
① Small size of hash Table

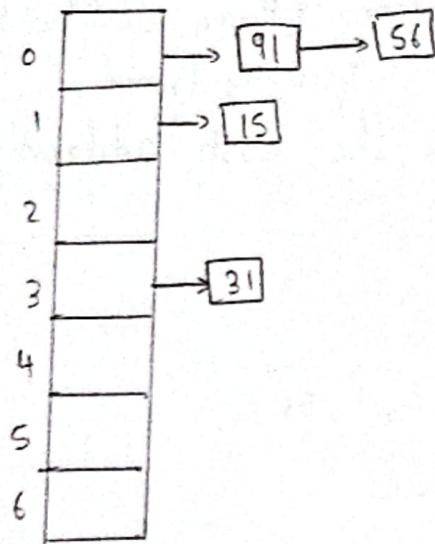
② Poor choice of Hash function

0	1	2	3	4	5	6
91	15		31	18	19	13

$\Rightarrow$  Methods of Hashing :-

- (i) Open Hashing (Separate Chaining) :-
- (ii) Closed Hashing (Open Addressing)

(i)



④ Keys are stored in linked list attached to cells of the Hash Table.

Ex:- A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED  
 $m = 13$

$$h(A) = 1 \bmod 13 = 1$$

$$h(FOOL) = 48 \bmod 13 = 9$$

$$h(AND) = 19 \bmod 13 = 6$$

$$h(HIS) = 36 \bmod 13 = 10$$

$$h(MONEY) = 72 \bmod 13 = 17$$

$$h(ARE) = 24 \bmod 13 = 11$$

$$h(SOON) = 63 \bmod 13 = 11$$

$$h(PARTED) = 64 \bmod 13 = 12$$

If the keys are distributed evenly then each list will be of length  $n/m$  called load factor. ( $\alpha$ )

$\alpha = 0$  (Empty locations present in Hash Table)

$\alpha = 1$  (Hash Table is Full)

=> Closed Hashing:-

All keys are stored in Hash Table itself.

=> Mechanisms for handling Collision:-

① Linear Probing:-

0	A
1	A PARTED
2	PARTED *
3	
4	
5	MONEY
6	AND
7	
8	
9	FOOL
10	HIS
11	ARE
12	SOON *

④ The collided value is added to the next available address

⑤ While searching a Key if the  $h(\text{Key})$  is filled and does not match, then the matching is done until an empty spot is encountered ⑥ match is found

④ While deleting a key, fill the cell in Hash Table with some value (This is called Lazy Deletion).

Problems:-

→ Clustering :- Grouping of Keys

② Double Hashing:-

$$h_i(k) = (\text{hash}_1(k) + f(i)) \text{ Mod } m$$

where,  $\text{hash}_1(k) = k \text{ mod Table Size}$

$$f(i) = i * \text{hash}_2(k)$$

$\text{hash}_2(k) = R - k \text{ Mod } R$  ( $R$  is a Prime number less than Table Size)

Ex:- 89, 18, 49, 58, 69, 48

0	1	2	3	4	5	6	7	8	9
69	48		58			49		18	89

$$h(89) = ((89 \times 10) + f(0)) \text{ Mod } 10 = 9 \times 10 = 9$$

$$h(18) = ((18 \times 10) + f(0)) \text{ Mod } 10 = 8 \times 10 = 8$$

$$h(49) = ((49 \times 10) + f(0)) \text{ Mod } 10 = 9 \times 10 = 9 \quad (\text{Collision})$$

$$= ((49 \times 10) + f(1)) \text{ Mod } 10 = (9 + f(1)) \text{ Mod } 10$$

$$f(1) = 1 \times \text{hash}_2(49) = 1 \times (7 - 49 \text{ Mod } 7) = 7$$

$$h(49) = (9 + 7) \times 10 = 16 \times 10 \Rightarrow h(49) = 6$$

$$h(58) = ((58 \times 10) + f(0)) \times 10 = 8 \text{ (Collision)}$$

$$h(58) = ((58 \times 10) + f(1)) \times 10$$

$$f(1) = 1 \times (7 - 58 \bmod 7) = 5$$

$$= (8 + 5) \times 10 \Rightarrow h(58) = 3$$

$$h(69) = ((69 \times 10) + f(0)) \times 10 = 9 \text{ (Collision)}$$

$$h(69) = ((69 \times 10) + f(1)) \times 10 = (9 + f(1)) \times 10$$

$$f(1) = 1 \times (7 - 69 \bmod 7) = 1$$

$$h(69) = (9 + 1) \times 10 = 0$$

$$h(48) = ((48 \times 10) + f(0)) \times 10 = 8 \text{ (Collision)}$$

$$h(48) = ((48 \times 10) + f(1)) \times 10 = (8 + f(1)) \times 10$$

$$f(1) = 1 \times (7 - 48 \bmod 7) = 1$$

$$= 9 \times 10 = 9 \text{ (Collision)}$$

$$h(48) = ((48 \times 10) + f(2)) \times 10 = (8 + f(2)) \times 10$$

$$f(2) = 2 \times 1 = 2$$

$$= (8 + 2) \times 10 = 0 \text{ (Collision)}$$

$$h(48) = ((48 \times 10) + f(3)) \times 10 = (8 + f(3)) \times 10$$

$$f(3) = 3 \times 1 = 3$$

$$= (8 + 3) \times 10 = 1$$