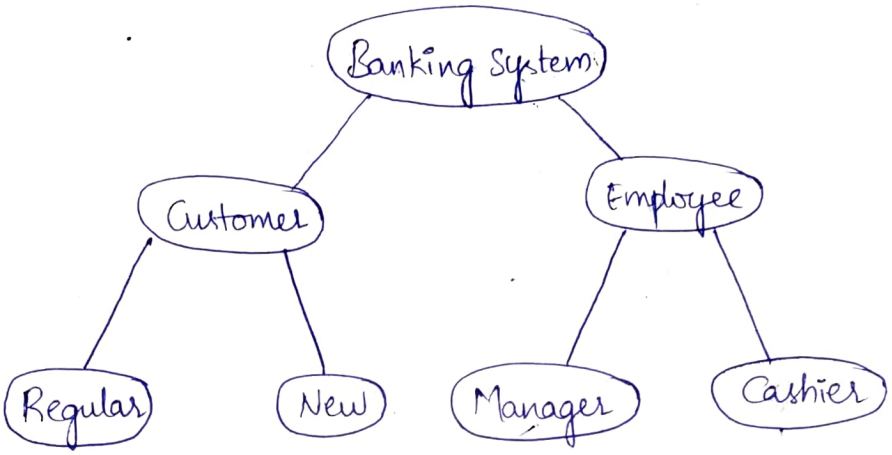
In previous, we have discussed Linear representation of the list data structures (Stack, Queue, Arrays, Data structures etc.,). Many Programming Languages uses this representation to solve the Complex problems, but there are some applications which involve the hierarchial information. for eg Consider an example of Banking System.



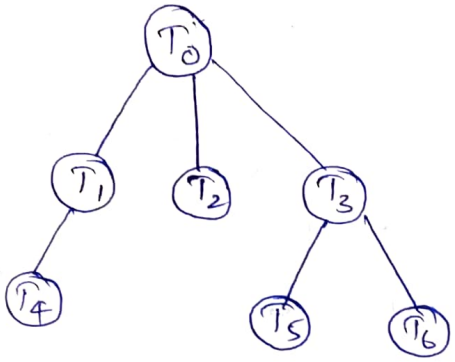
for any banking system, there are two important entities, Customer and employee. The customer can be regular customer (or) new customer. The employees may be manager, Cashier or any other employees in the bank. This information can be shown in efficient manner by hierarchial structure.

Tree is a data structure which is used to represent such type of hierarchial information. It represents some kind of relationship between the nodes of the tree. Every particular node may be a leaf node or parent node.

Definition of Tree:

A Tree is a finite set of one or more nodes such that
(i) There is a specially designed node called "Root".
(ii) The remaining nodes are partitioned into $n \geq 0$, disjoint sets

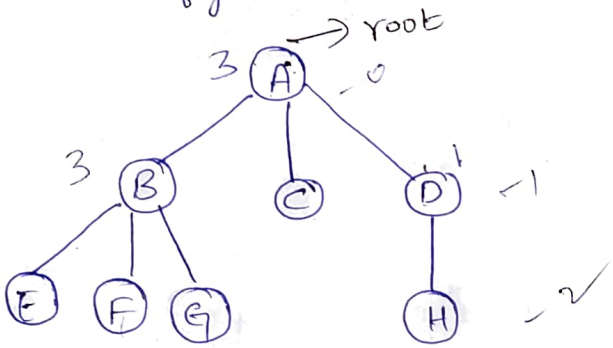
$T_1, T_2, T_3 \dots T_n$ where $T_1, T_2, T_3 \dots T_n$ are called subtrees



Tree Terminologies:

Root:

This is the unique node in the tree to which further subtrees are attached. In the above figure node 'A' is the root node.



Degree of the node:

The total no. of subtrees attached to the node is called the degree of the node. for node 'A', the degree is '3', for the node 'E', the degree is '0'.

Leaves:

These are the terminal nodes of the tree. The nodes with degree '0' are always the leaf nodes. In the above figure, E, F, G, C, H are the leaf nodes.

Internal nodes:

The nodes other than root node and leaf nodes are called internal nodes. In the above figure, B and D are the internal nodes.

Parent nodes:

The node which is having further subtree (branches) is

called the parent node of those subtrees. In the given figure, node 'B' is parent nodes of E, F, G.

Predecessor:

While displaying the tree, if some particular node occurs previous to some other node, then that node is called predecessor node of the other node. In the above figure, 'E' is the predecessor of node 'B'.

Successor:

The node which occurs next to some other node is a successor node. In the figure, node B is the successor of F and G.

Level of the tree:

The root node is always considered as level '0'. Then its adjacent children are supposed to be at level 'i' and so on... In the figure, the node 'A' is at level '0', the nodes B, C, D are at level '1', the nodes E, F, G, H are at level '2'.

Height of the tree:

The maximum level + 1 is the height of the tree. Here, height of the tree is "three". The height of the tree is also called depth of the tree.

Degree of the tree:

The maximum degree of the node in a tree is called the degree of the tree.

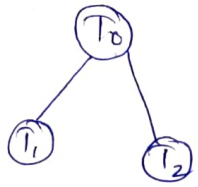
Siblings:-

child:-

Binary Tree:

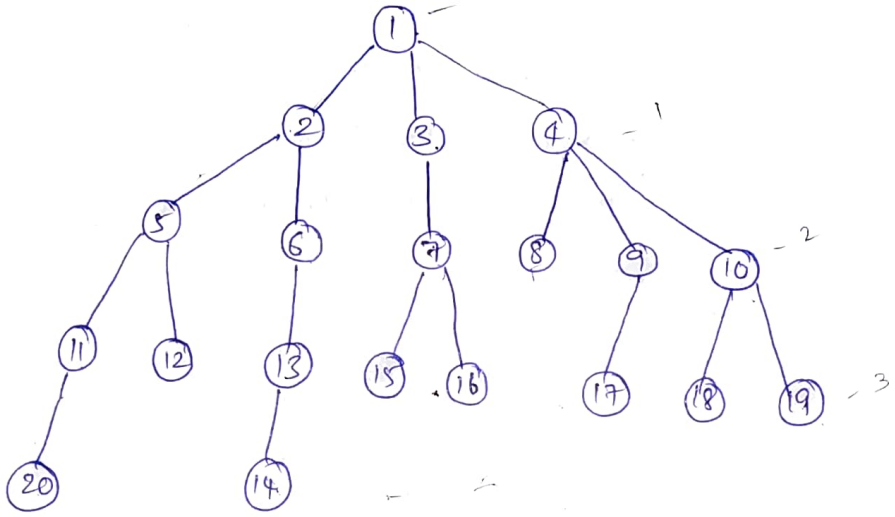
A binary tree is a finite set of nodes which is either empty (or) Consists of a root node and two disjoint trees called left subtree and right subtree.

Eg: 0 (or) T_0 (or)



In Binary tree, each node will have one datafield and two pointer fields for representing the sub branches. The degree of each node in the binary tree will be atmost 2.

Eg:



(i) Root node : 1

(ii) Degree of the node|tree : 3

(iii) Leaves : 20, 12, 14, 15, 16, 8, 17, 18, 19

(iv) Internal nodes : 11, 5, 6, 13, 7, 9, 10, 2, 3, 4

(v) Parent nodes : 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13

(vi) Predecessor: for 4, predecessor is 8

(vii) Successor: for 4, successor is 10.

Level of the tree = 4

Height of the tree = 5

Types of Binary Trees:

There are three types of Binary trees

1) Left Skewed Binary Tree:

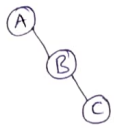
In this case, the right subtree is missing in every node of a tree

eg:

2) Right Skewed Binary Tree:

If the left subtree is missing in every node of a tree, then it is said to be a Right Skewed Binary Tree.

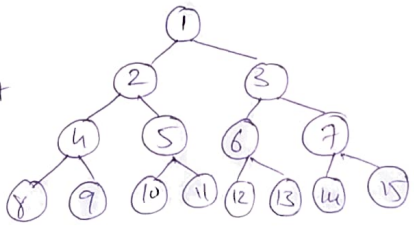
eg:



- Binary tree can be given with two types as full binary tree and complete binary tree.

- Full binary tree is a binary tree which contains the maximum possible number of nodes at all the levels.

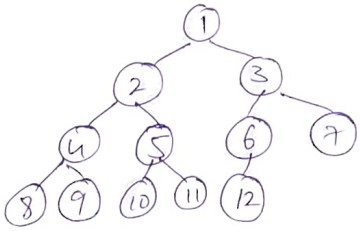
Height = 4



levels	nodes
0	1
1	2
2	4
3	8

- Complete binary tree is a binary tree, if all its levels, except possibly last level have maximum number of nodes, and all nodes at last level will be left empty.

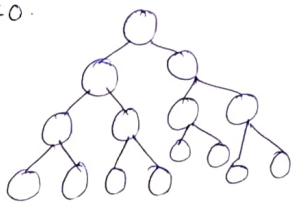
Height = 4



levels	nodes
0	1
1	2
2	4
3	5

Properties of a binary tree:-

- In any binary tree, maximum no. of nodes on level l is 2^l, where $l \geq 0$.



Level	Node
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
⋮	⋮
l	2^l

- Maximum no. of nodes possible in binary tree of height h is $2^{h+1}-1$.

- The minimum no. of nodes possible in binary tree of height h is h.

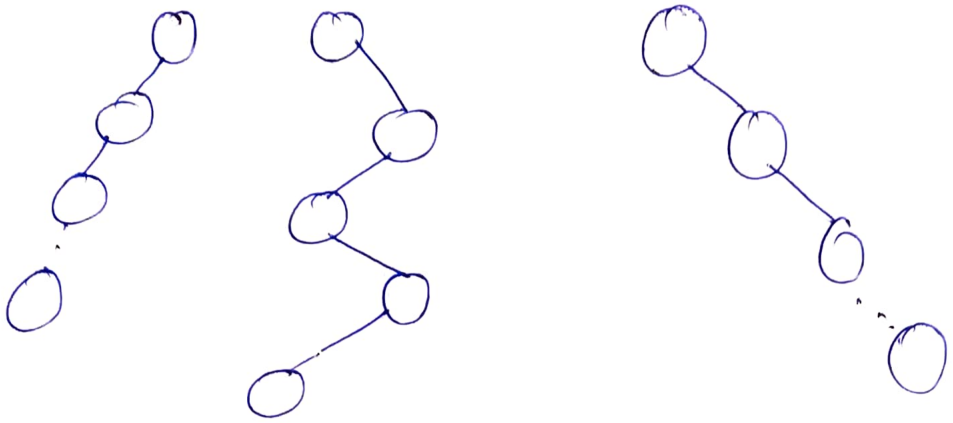


fig:- Skew binary trees with min. no. of nodes.

Binary Tree Representation:

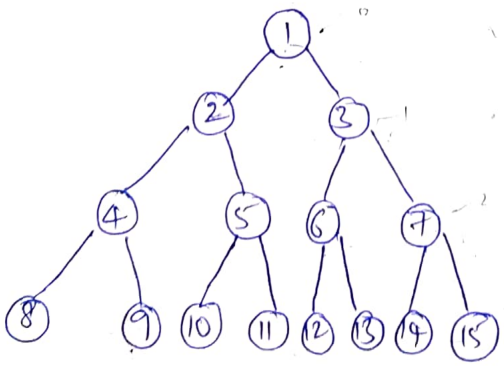
There are two ways of representing a binary tree.

(i) Sequential Representation (arrays)

(ii) Linked list Representation (pointers)

Sequential Representation:

Each node is sequentially arranged from top to bottom and from left to right. Let us Consider this concept by numbering each node, the numbering will start from root node and the remaining nodes will give increasing numbers in level wise direction. The node in the same level will number from left to right. The numbering will be as shown below.



Now, the binary tree of depth 'n' is having $2^n - 1$ nodes.

In the figure, the tree is having depth '4' $\Rightarrow 2^4 - 1 = 15$.

The total no. of nodes are $2^4 - 1 = 15$. If we know the max. depth of the tree, then we can represent binary tree using Array data structure.

We can use the Array size of 'n', the root will be at position '1' and its left child will be at position '2' and its right child will be at position '3' and so on... The another way of placing the elements in the array by applying the following formulae.

used to identify location

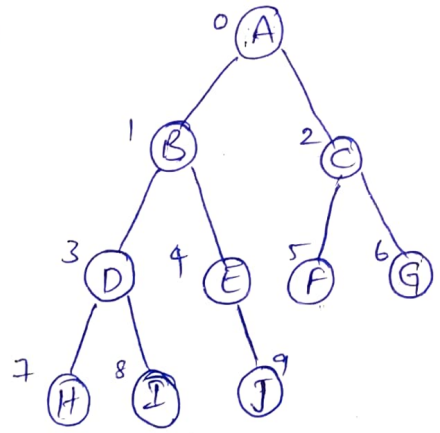
$$\text{Parent}[n] = \text{floor}[(n-1)/2]$$

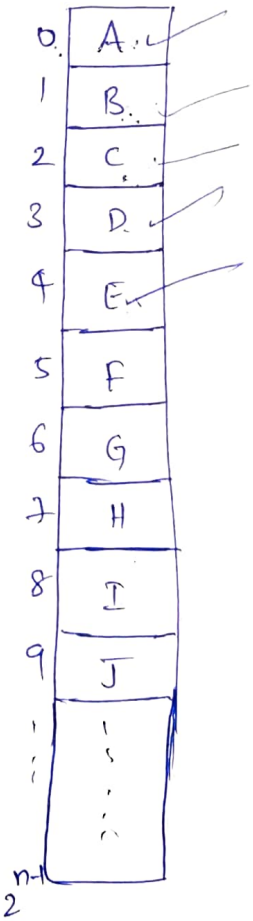
$$L[n] = 2n+1$$

$$R[n] = 2n+2 \quad \text{where } n \geq 0.$$

When $n = 0$, the root node will be placed at zeroeth location.

Eg:





$n = 0$, $P[0] = f(0) = 0$

$L[0] = 0+1 = 1$

$R[0] = 0+2 = 2$

$n = 1$, $P[1] = f(0/2) = 0$

$L[1] = 2+1 = 3$

$R[1] = 2+2 = 4$

$n = 2$, $P[2] = f[1/2] = 0$

$L[2] = 5$

$R[2] = 6$

$n = 3$, $P[3] = f[2/2] = 1$

$L[3] = 7$

$R[3] = 8$

$n = 4$, $P[4] = f[3/2] = 1$

$L[4] = 9$

$R[4] = 10$

Advantages of Sequential Representation:

* The only one advantage with this type of representation is that the direct access to any node can be possible by finding the parent or left or right children by using the formulae.

Disadvantages:

* The major disadvantage with this type of representation is wastage of memory.
* In this type of representation, the max. depth of the tree has to be fixed because we have to decide the array size. If we choose the array size larger than the depth of the tree then it will be the wastage of memory, if we choose array size lesser than the depth of the tree ; then we will be unable to represent some part of the tree.
* The insertion and deletion of any node in the tree will be complex as other nodes has to be adjusted at appropriate positions such that the meaning of the binary tree can't be changed.

As there are drawbacks with this type of representation, we will search for more flexible representation.

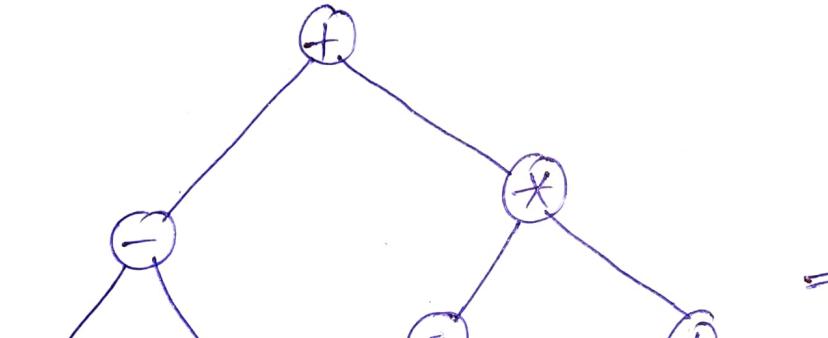
So, instead of array, we will make use of linked list to represent a tree.

Sequential representation / Linear representation :-

- This is static ie., block of memory for an array is allocated before storing actual tree in it and if once allocated then it restricts other memory storage.

Ex:-

Consider an Expression as $(A-B)+C*(D/E)$.



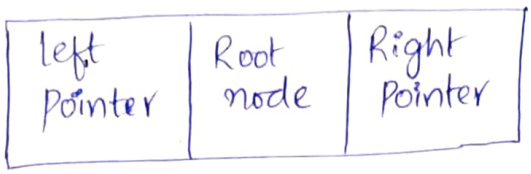
⇒ binary tree

array representation of binary tree is,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
+	-	*	A	B	C	/	.	.	.	.	.	.	D	E

Linked list Representation:

In linked list, each node will be as follows.



In Binary tree, each node will have left child, right child and data field. The left child is nothing but the left link which points to some address of left subtree where as right child is a right link which points to some address of right subtree and the data field which gives the information about the node.

let us see the structure of the node in a binary tree.

```
Struct tree
{
  int data;
  Struct tree *left, *right;
};
```

Advantages:

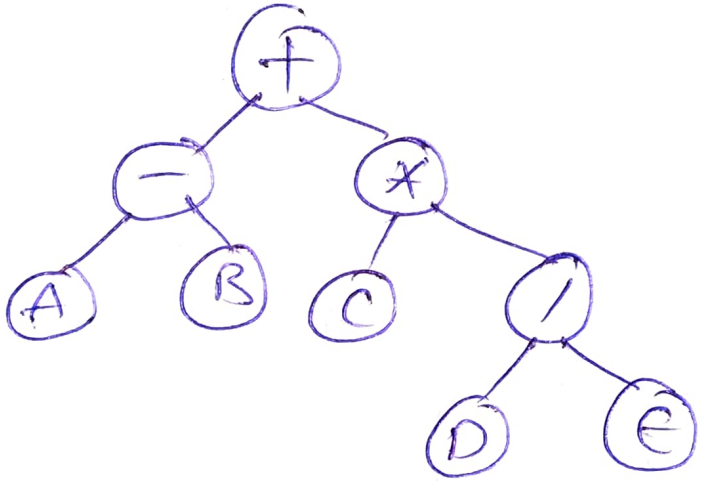
* This representation is very effective than the array representation.
* Insertion and deletion can be done without moving the other nodes.

Disadvantages:

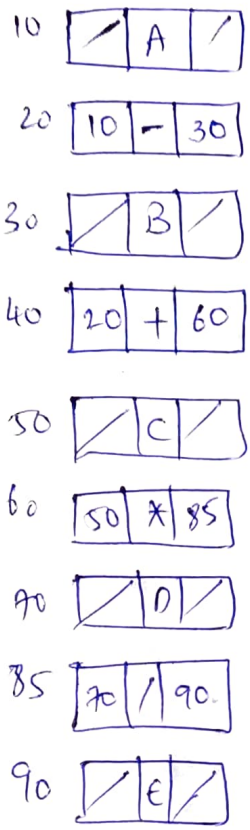
* This representation doesn't provide direct access to a node.
* This representation needs additional space in each node for storing left and right subtrees.

Linked List representation :-

- Now Consider the binary tree of above Expression as,

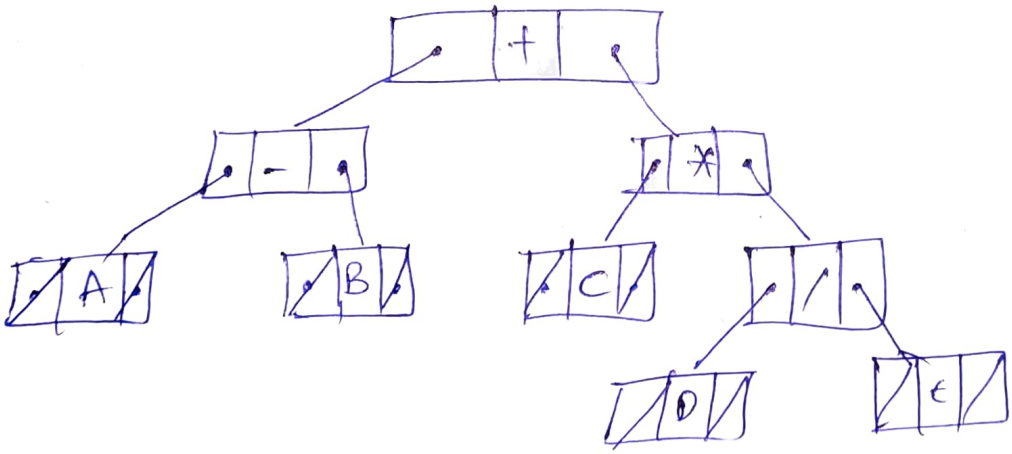


- The nodes physical view is given as,



$\Rightarrow (A-B) + C * (D/\epsilon)$

- linked list / logical view of a binary tree representation, is given as follows,



Note :-

- In linked representation of a binary tree, if there are 'n' no. of nodes then no. of null links is given as, $\boxed{\lambda = n+1}$.

⇒ Operations on a Binary tree :-

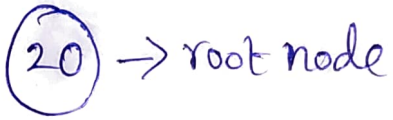
- The major operations on a binary tree can be listed as follows,

1. Insertion (To include a node into existing/empty binary tree)
2. Deletion (To delete a node from a non-empty binary tree)
3. Traversal (To visit all the nodes in a binary tree)
4. Merge (To merge two binary trees into a larger one)
5. Create (used to create the root node of a tree as null).

1. Create :-

- To avail/allocate space/the node need to be created.
- Whenever binary tree is empty, the resultant created node acts as a root node.

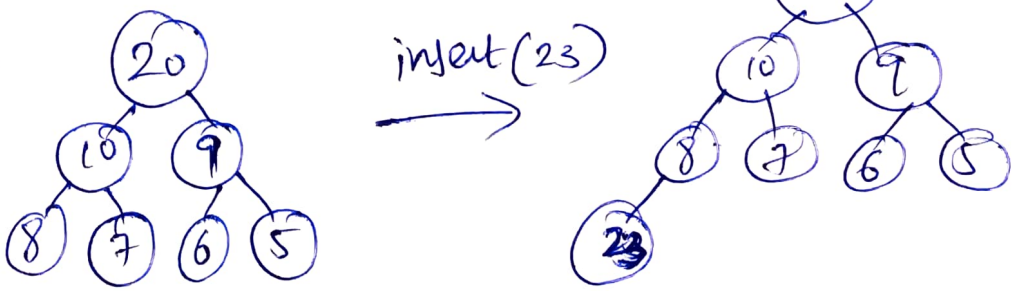
Ex :-



2. Insertion :-

- This is used to add a new element/node into existing binary tree (or) an empty binary tree.
- The elements/nodes added or inserted will be attached as leaf nodes.

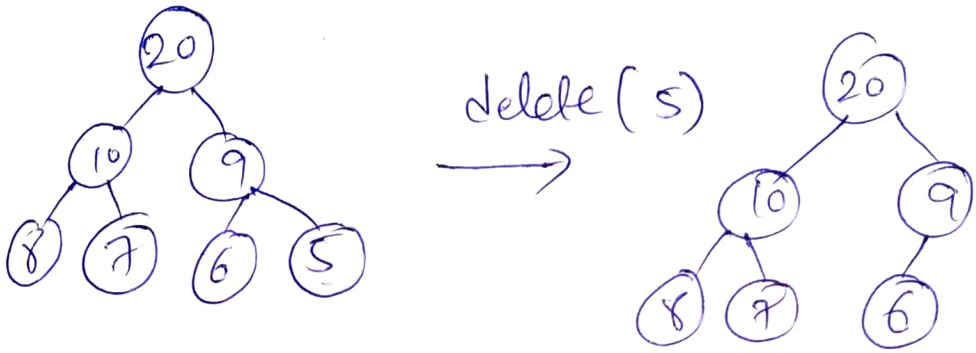
Ex :-



3, Deletion :-

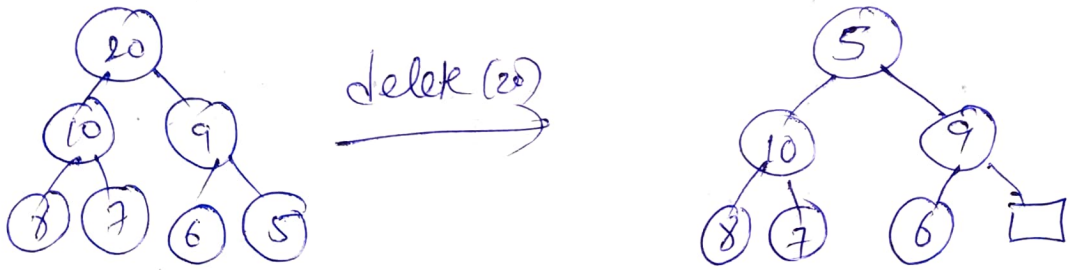
- This is used to delete the node from the tree.
- The deletion can happen in many ways as follows.

a) Deleting leaf node :-



b) Deleting node with left and Right child nodes :-

c) Deleting root :



4 Traversal :-

- Traversal means visiting each node exactly once.
- Basically there are 3 ways to traverse a tree.

1. Inorder - Left, Parent, Right (LPR)
2. Preorder - Parent, Left, Right (PLR)
3. Postorder - Left, Right, Parent (LRP)

Inorder

```
Ptr = root
if (Ptr ≠ NULL) then
Inorder (Ptr → L)
visit (Ptr)
Inorder (Ptr → R)
end if
stop
```

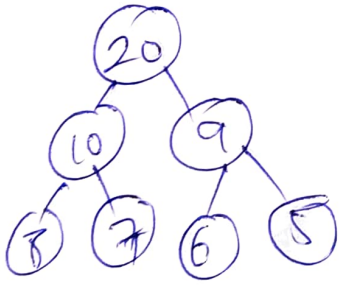
Preorder

```
ptr = root
if (Ptr ≠ NULL) then
visit (Ptr)
Preorder (Ptr → L)
Preorder (Ptr → R)
end if
stop
```

Postorder

```
Ptr = root
if (Ptr ≠ NULL) then
Postorder (Ptr → L)
Postorder (Ptr → R)
visit (Ptr)
end if
stop
```

Ex :-



Inorder - 8, 10, 7, 20, 6, 9, 5

Preorder - 20, 10, 8, 7, 9, 6, 5

Postorder - 8, 7, 10, 6, 5, 9, 20

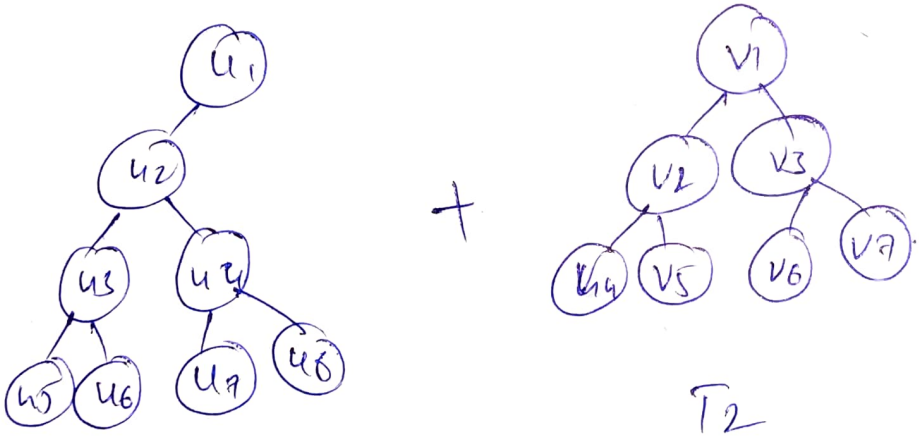
5 Merge :-

- whenever you need to merge two trees, T_1 & T_2 (binary trees), T_2 can be merged with T_1 if all nodes of T_2 are inserted one by one into T_1.
- other way is entire T_2 is included as subtree of T_1.
- for this, if both the trees have left & right sub-trees then merge will fail.

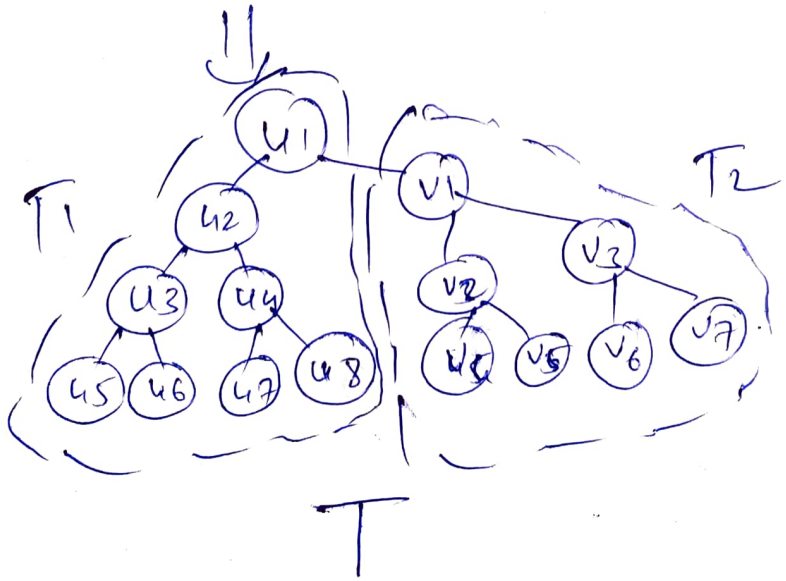
  Otherwise, if T_1 has left subtree empty then T_2 will be added as left sub-tree of T_1 and vice-versa.

$$T(n_1 + n_2) = T_1(n_1) + T_2(n_2)$$

where T is resultant tree after merging T_1 & T_2.



T_1 T_2



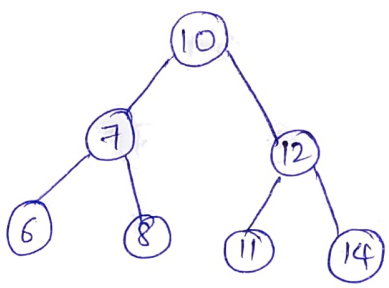
BINARY SEARCH TREES:

In the simple binary tree, the nodes are arranged in any fashion depending on the user's request. The new nodes can be attached as a left or right child of any node. In such case, finding any node is long procedure, because in that case we have to search the entire tree which increases time complexity of the algorithm. So, make the searching algorithm faster in a binary tree we will go for designing binary search tree. The Binary Search Tree is based on the binary search algorithm. While creating the binary search tree, the data is systematically arranged i.e., Values at left subtree is less than root node is less than right subtree values.

$$V(L) < V(P) < V(R)$$

ADT bin

Eg:



6-7-8-9-11-12-14

Operations on Binary Search Trees:

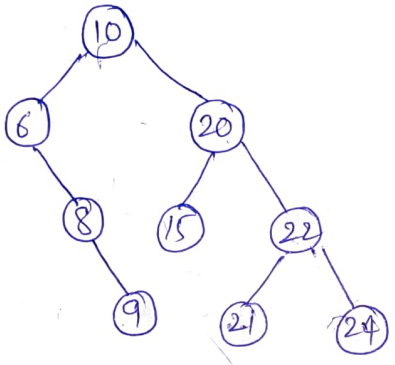
The basic operations which can be performed by Binary Search Trees are

1) Insertion of a new node into the Binary Search Tree.
2) Deletion of a node from Binary Search Tree.
3) Searching for a particular node in Binary Search Tree.

→ Insertion:

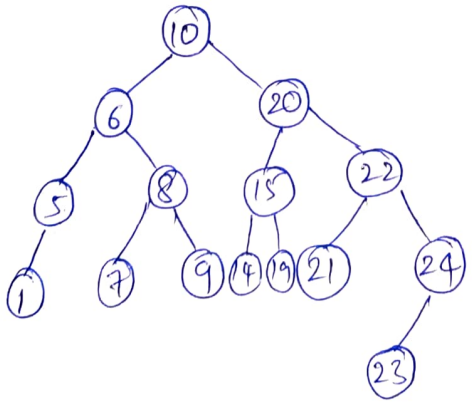
While inserting any node into the Binary search tree, first of all we have to look for its appropriate position in the binary search tree. We start Comparing the new node with each node of that tree, if the value of that node which is to be inserted is greater than the value of Current node, we will move on to the right subtree, otherwise move to the left subtree. As soon as appropriate position is found, we attach the new node as a left child or right child.

Eg:



In the above figure, if we want to insert 23, then we will start Comparing 23 with the value of root node i.e., as 23>10, we will move to the right subtree, now we will compare 23 with 20 and move to the right. Now, Compare 23 with 24, it is less than 24, we will move left subtree of 24. But, there is no node as a left child of 24, so we can attach 23 as left child of 24.

Eg:



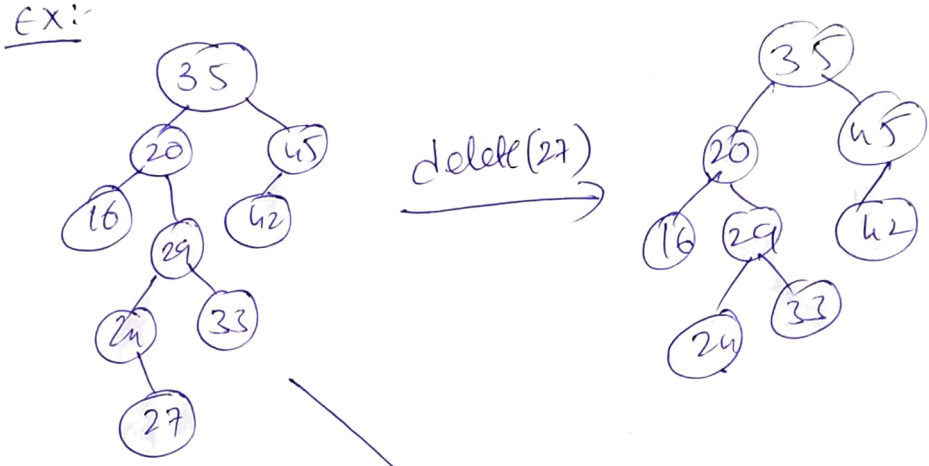
Deletion:

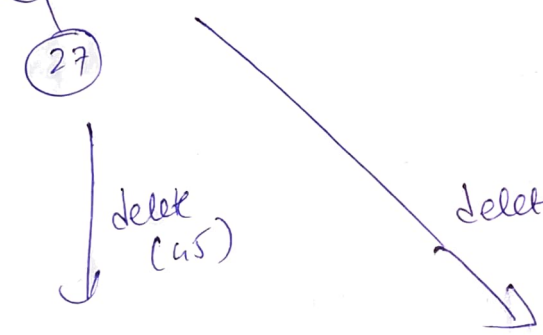
for deleting any node from binary search tree, there are 3 cases

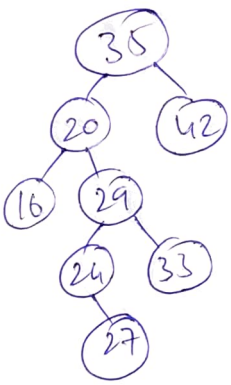
* Deletion of a leaf node
* Deletion of a node having 1 child
* Deletion of a node having 2 children

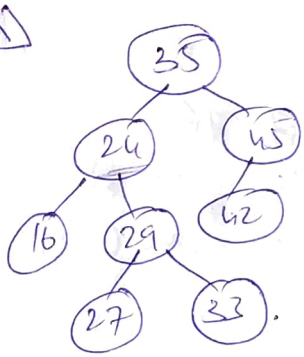
Deletion in BST:-

Ex:-

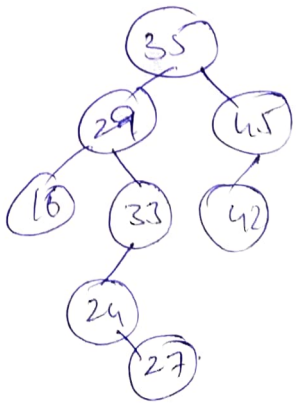






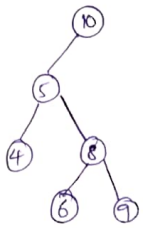


(d)



Searching for a node in the Binary search Tree:

In searching the node, the node which we want to search is called key node. The key node will be compared with each node from the start of the root node, if the value of the key node is greater than the current node, then we search for keynode on the right subtree otherwise on the left subtree. If we reached leaf node and we still we donot get the value of the key node, then we will declare that the "node is not present in the tree."

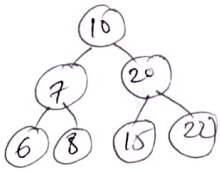


In the above tree, if we want to search for the node '9', then we will compare 9 with root node 10, as 9 is less than 10, we will reach on to the left subtree of 10. Now, Compare 9 with 5, but 9 is greater than 5, so we will move to the right subtree. Now, Compare 9 with '8', as 9 is greater than '8', we will move to right. Now, we will get the node '9' which says the search is successful.

→ Searching an element :-

→ To search an element in a BST, the same order as (left < Root < Right).

Ex:-



Search '8'. so let, key = '8'.

i) Compare key whether less/greater than root as,

a) 8 < 10
key < root (T), So move left

b) ~~8 < 7~~
~~key < root (F)~~
8 > 7
key > root (T), so move right.

c) 8 < 7 (F)
8 > 7 (F)
8 = 8 (T), Hence key is found.

⇒ Heap Trees :-

→ Heap is a tree datastructure denoted by either a maxheap (or) a minheap.

→ It is termed to be a heap tree in either of two following conditions as,

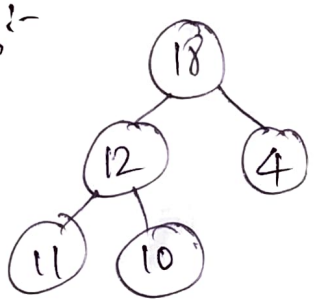
(a) For each node N in H, value of N is greater than/ Equal to value of Each of children of N.

(or)

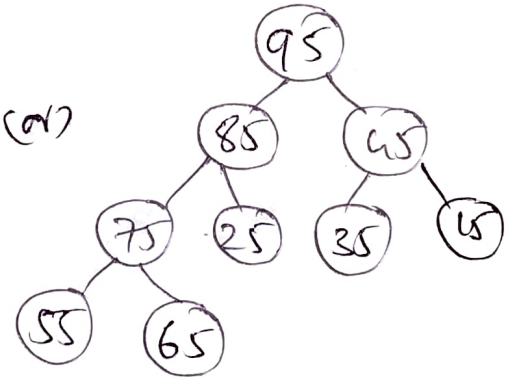
N has a value greater than/ Equal to value of every successor of N.

This is called as "Max heap"

Ex:-



(or)

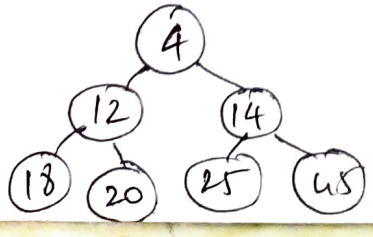


(b) For each node N in H, value of N has less/ Equal to value of each children of N,

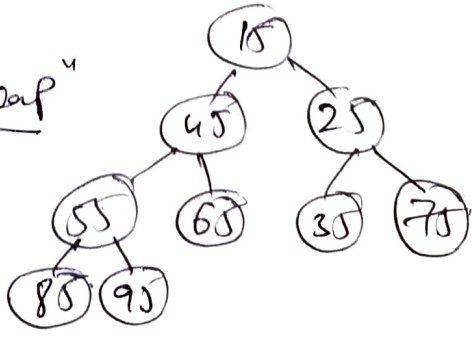
(or)

N has a value less than/ Equal to value of any of Successor of N.

This is called as "Min heap"



(or)



Representations of a heap tree :-

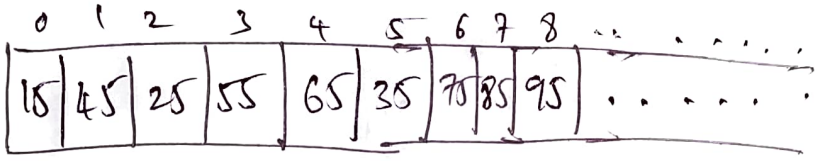
→ A heap tree is represented using Linked list.

→ But a single array representation has many advantages for a heap tree over the LL.

→ As a heap tree is a Complete binary tree, there will be no wastage of array space while entering data.

→ Also the links with descendants (child) are automatically implied.

→ So, here we use single array representation as follows.



Operations on a heap tree :-

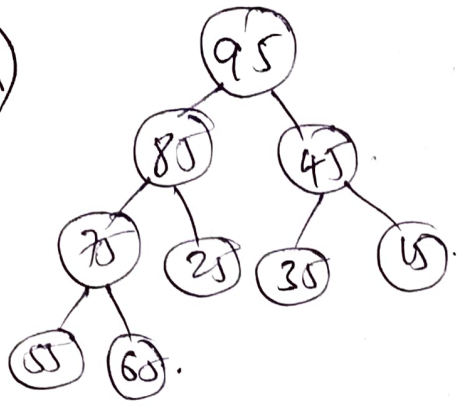
→ The major operations performed on heap tree are,

i) Insertion
ii) Deletion
iii) Merging

i) Insertion :-

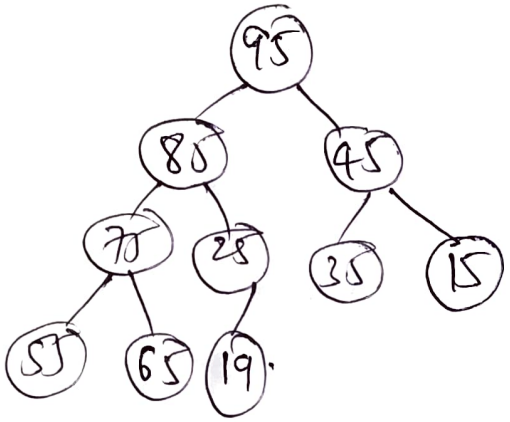
- This is used to insert a node into heap tree by satisfying properties of heap tree.
- Let us consider a max. heap tree and insert a new element as 19 to it as follows,

a)



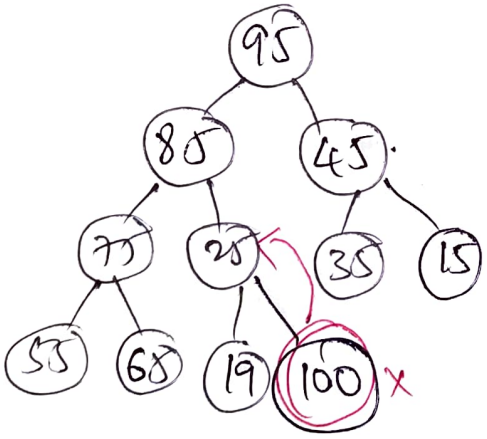
→ Max heap

Insert 19 as in the complete binary tree (ie, level wise)



(Here inserted position 19 follows property of heap)

b) Insert element 100 into above tree then,



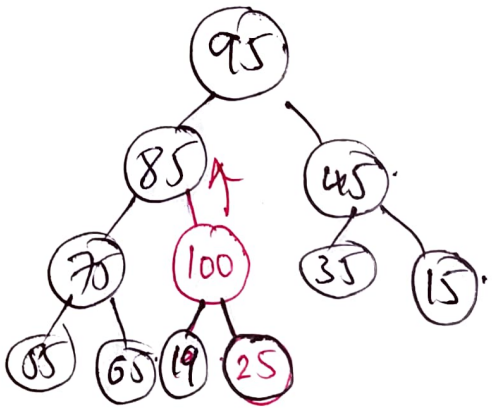
(But the 100 inserted here doesn't follow property of heap)

→ So, perform the comparison of inserted node & its root/parent node.

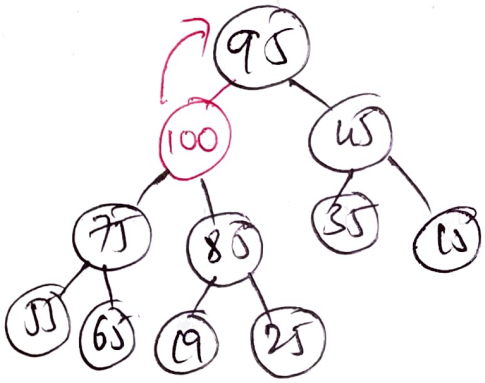
→ So, 100 is compared with 25, when 100 is compared it is greater one so interchange both.

→ Now continue same kind of comparision with 100 and 85 so again interchange 100.

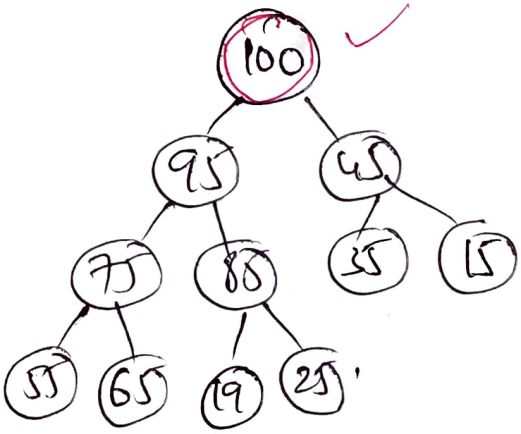
→ Finally 100 is compared with 95 then again interchange occurs. Now, root node becomes 100 & so we stop the process.



100 > 85 so, interchange.



100 > 95 so interchange.

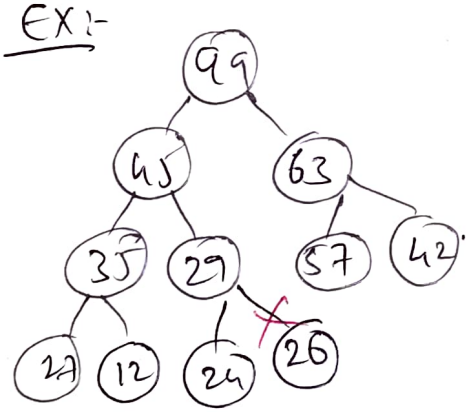


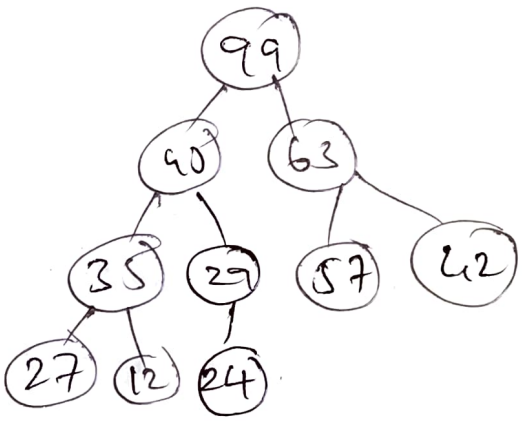
ii) Deletion :-

- Any node can be deleted from a heap tree.
- So, whenever you want to delete an element, there are different cases,
  a) If a child node ie, last nodes are deleted, then there exists no change in the tree.
  b) If a root node (current node) need to be deleted, then, replace it with the last node in heap tree.
     - If a node (root node) is the node then Compare its values with values of its two children. And replace the largest child node with the value of current node to be deleted.
     - Then make the newly updated node as current node and continue the reheap process again till it reaches empty node.

(Reheap process)

Ex :-



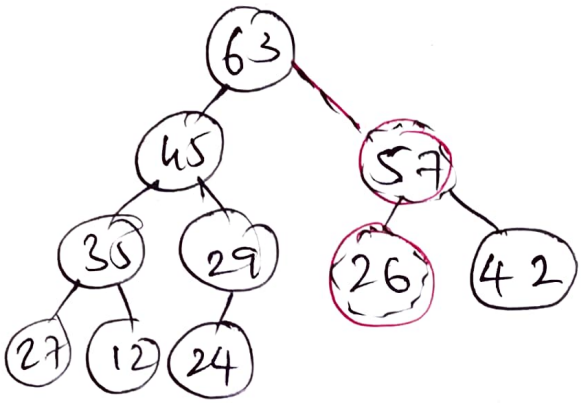


↓ delete 99

Replace last node (26) with root node, in order to delete '99' (root node). Then, Compare its child nodes 45 & 63. so 63, is largest value child node of 99 so. replace it with root node as follows)

(Similarly continue comparision with next nodes till it reaches end)

(Reheap Process)

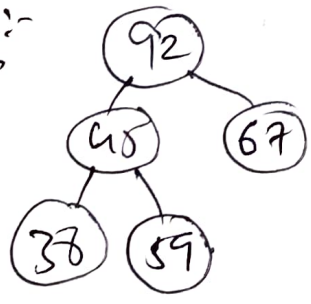


- First to delete 99, we replace last node in heap tree 26.
- Then start comparing 26 with two child nodes (of deleted node). And replace/interchange highest node with it. (i.e.,
- Continue process till end / till it reaches end node.

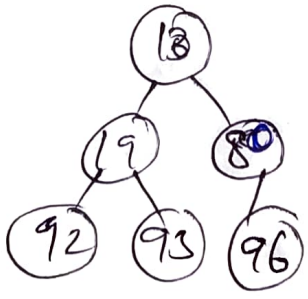
iii) Merging two heap trees :-

- Consider two heap trees H_1 & H_2. So, merging the trees includes all nodes from H_2 to H_1.
- H_2 can be either min heap / max heap.
- If H_1 is min / max heap tree then resultant tree is a min heap else max heap.
- Steps to be followed are,
  a) delete root node (deleted), from H_2.
  b) Insert node into H_1, satisfying property of H_1.

Ex :-



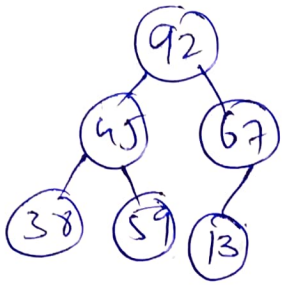
H_1 : Max heap



H_2 : Min heap.

i) Delete root node from H_2 & insert at H_1 (with property of maxheap)

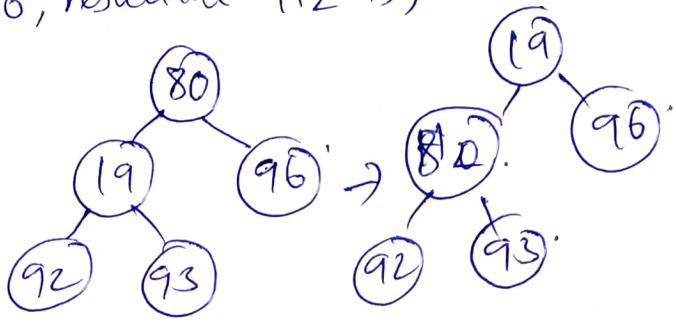
H_7



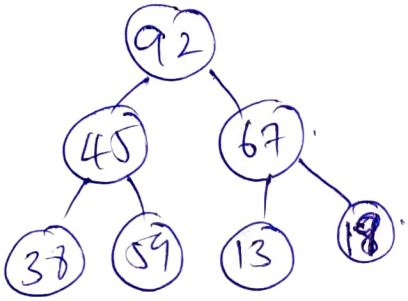
H_1

→ So, here after merging 13 at H_1 tree is in correct ordering heap.

→ So, resultant H_2 is,



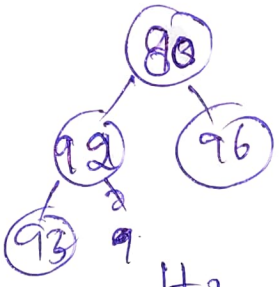
ii) Again consider root of H_2 (updated one) & insert at H_1.



H_1

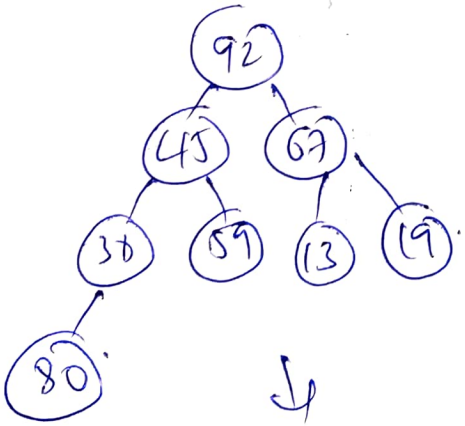
→ So, here after merging 8 also H_1 tree is perfect with heap

→ So resultant H_2 is.



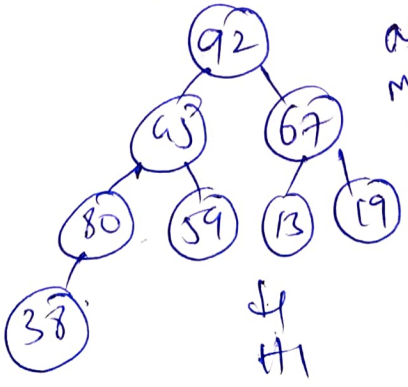
H_2

iii) Again Consider root of H_2 (updated) & insert at H_1



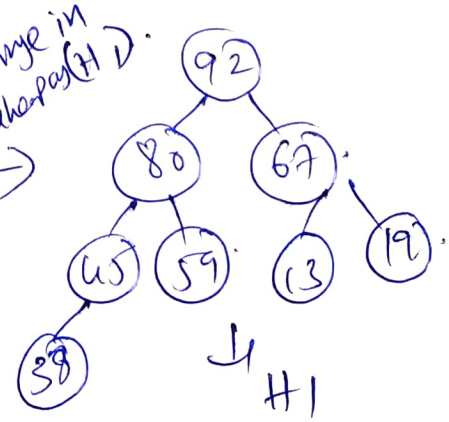
H_1

→ arrange it in maxheap :: H_1 is maxheap



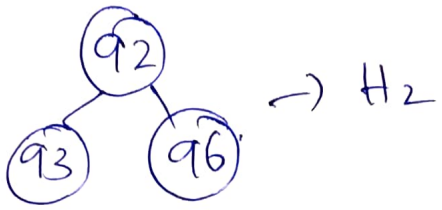
H_1

→ arrange in maxheap(H_1)

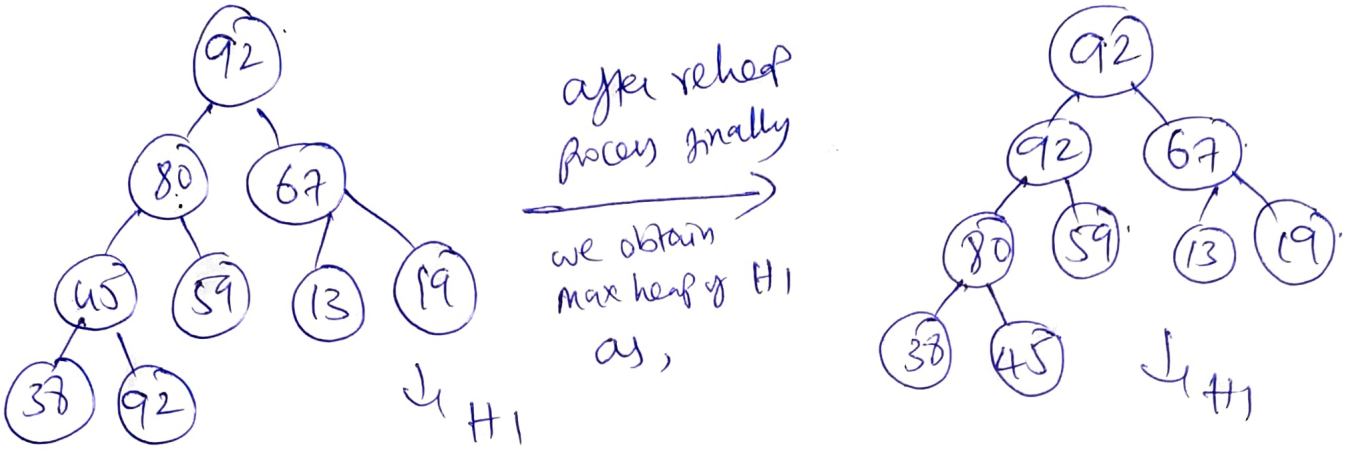


H_1

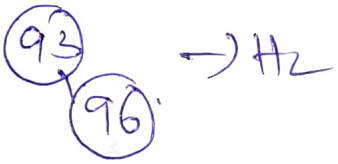
→ So, here after merging then H_2 is resulted as,



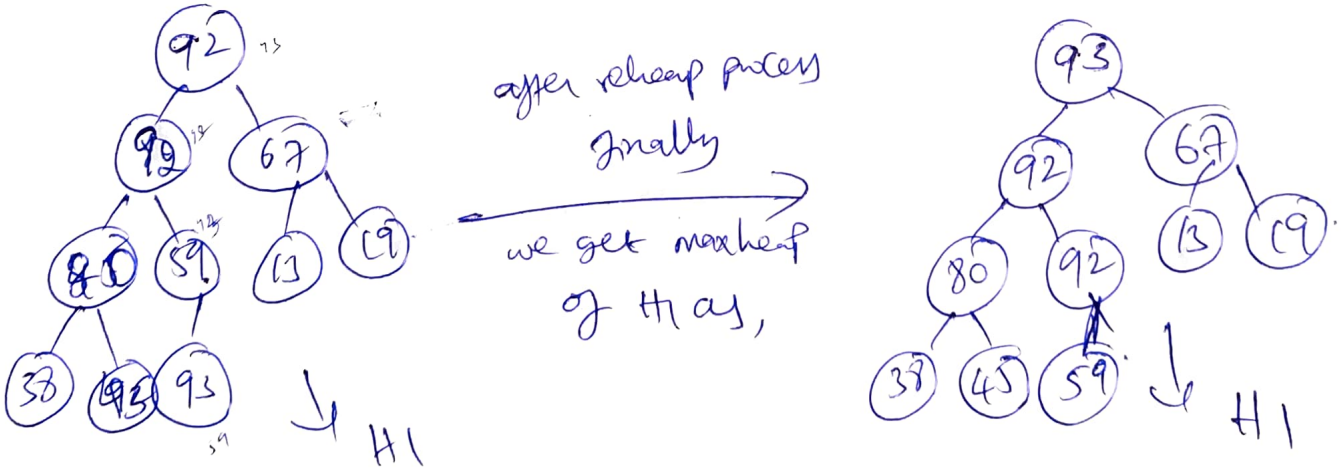
iv) Again Consider root of H_2 (updated) & insert at H_1



→ So here, H_2 is obtained with 2 nodes as,



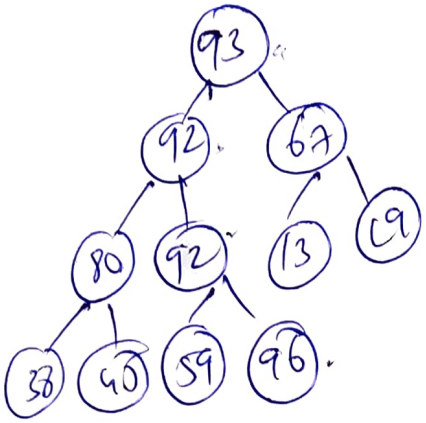
v) Again Consider root of H_2 (updated & insert at H_1



→ So here finally in H_1 only one node as,

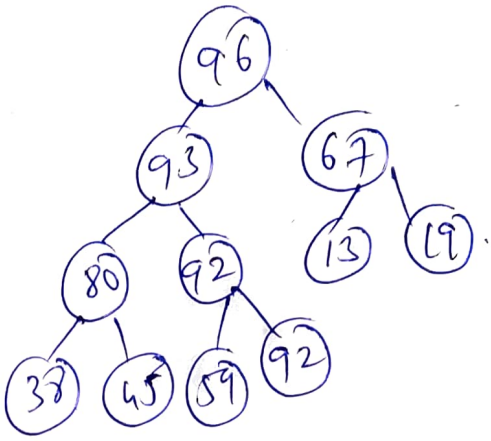
(96) exists.

vi) Again merge 96 from H_2 to H_1 & we get it as, (H8)



after reheap process finally

obtained maxheap of H_1 is,



So finally $H_1 \oplus H_2$ is, (ie, merging of H_1 & H_2)