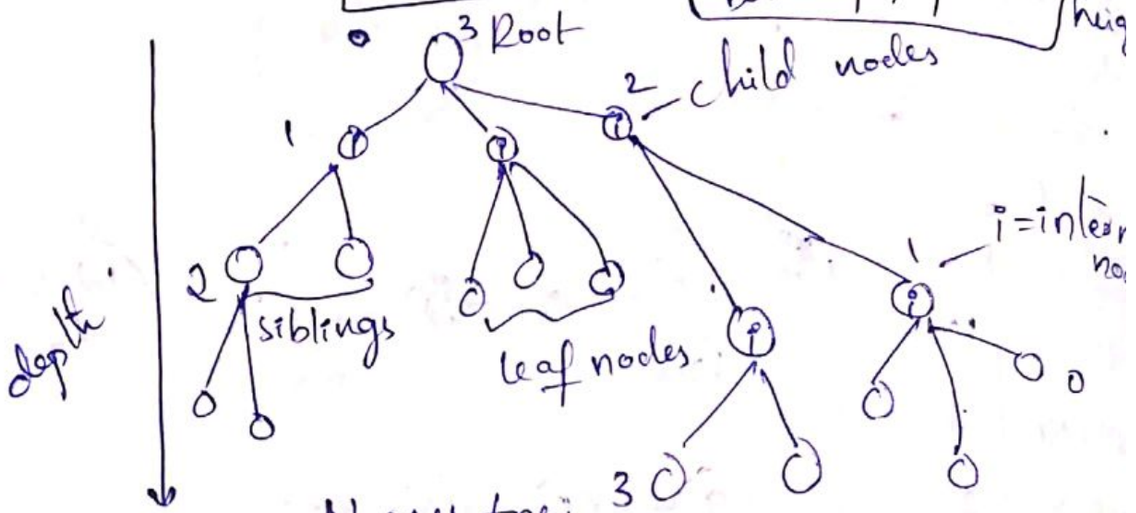
Trees

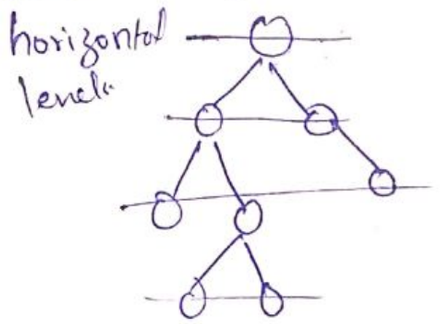


N-ary tree

* height of tree = height of root
* depth of root = 0
* depth of node = distance of the node from root
* internal node = neither Root / child / leaf
* Path length = no. of edges / links from root
* height of node = distance of the node from deepest leaf node
* Path = Root to leaf.

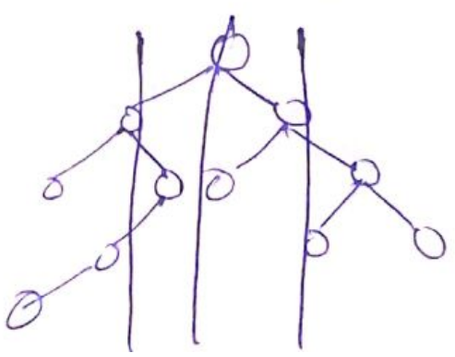
*** Longest path might not pass through Root**

* A tree with atmost two children



Binary tree

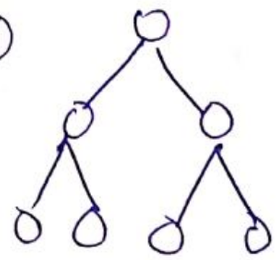
*



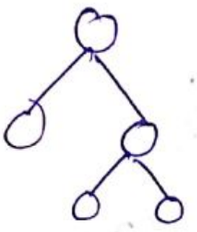
vertical level

* CBT : level by level — Complete Binary Tree

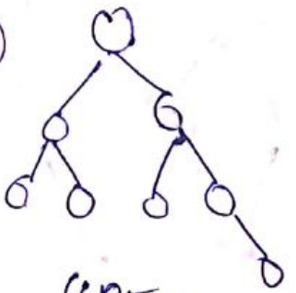
FBT : it will have either 0 or 2 — Full Binary Tree

① 

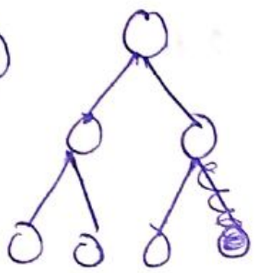
CBT, FBT

② 

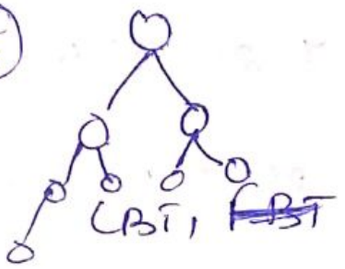
FBT

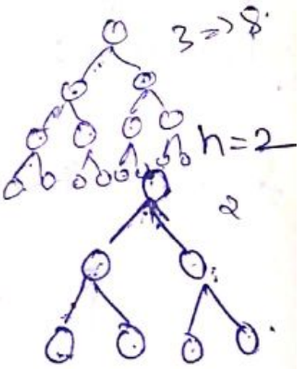
③ 

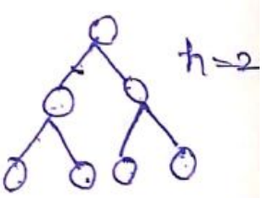
~~CBT~~ NO CBT, NO FBT

④ 

~~CBT~~, NO ~~FBT~~

⑤ 





5

	Min	Max
CBT	$h+2 = 4$	$h+5 = 7$
FBT	$h+3 = 5$	$h+5 = 7$

$\Rightarrow$ CBT → Min = 2^h

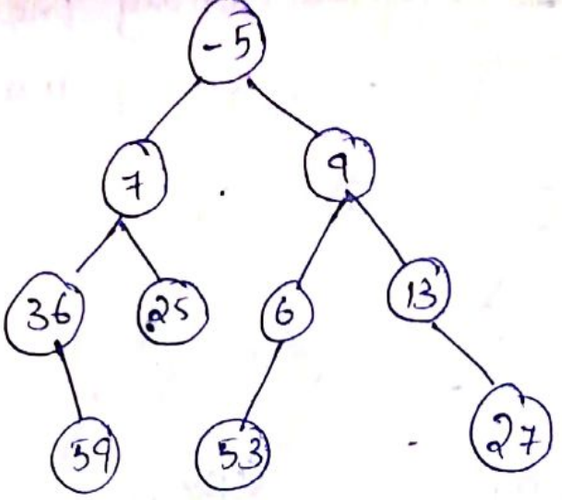
→ max = $2^{(h+1)} - 1$

$\Rightarrow$ FBT → min = $2(h)+1$

→ max = $2^{(h+1)} - 1$

```
* class Node
  {
    int data;
    node *left, *right;

  };
```

*



Root LR
Preorder:- -5 7 36 59 25 9 6 53 13 27

L Root R
Inorder:- 36 59 7 25 -5 53 6 9 13 27

L R Root
Postorder:- 59 36 25 7 53 6 27 13 9 -5

* Void Inorder (node *root)

```
{ Inorder( if root = NULL :
                return;
        Inorder (root → left);
        Inorder (root → data);
        Inorder (root → right);
}
```

* int no_of_Nodes (Node *root) => 10
* int max sum (node *root) =>
* int maxvalue (node *root) => 59
* int height (node * root) => 3
* Void filldepth ( )

1. int no of nodes (node * root)

```
{   if root = NULL;
        return 0;

    left = 1 + Inorder(*root)

    c = 1;
    left = no of nodes(root → left);
    right = no of nodes (root → right);
    return c + left + right;
}
```

2. int sum (node, *root)

```
{   if root == NULL:
        return 0;

    x = root → data

    left_data = sum(root → left);
    right_data = sum(root → right);

    return x + left_data + right_data;
}
```

3. int max value (node *root)

```
{       if root == NULL;
            return int_min

        x = root → data;
    left_data = max value (root → left)
    right_data = max value (root → right).
    return max (x, left_data, right_data);
}
```

* int height (node * root):

```
{ if root == NULL:
        return -1;
  return 1 + max (height(root->left),
                  height(root->right);
}
```

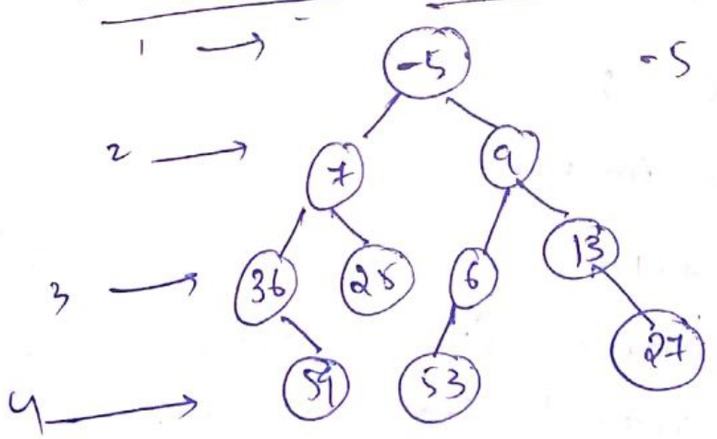
* Void fillDepth (Node *root, int depth)

```
{ if root==Null:
      return;
  root->depth = depth
  fillDepth (node * root, depth
  fillDepth (root->left, depth+1)
  fillDepth (root *right, depth+1).
}
```

```
node
{ int data;
  int depth;
node* left;
node * right;
}
```

---

* Level Order: Breadth first Traversal:



-5 7 9 36 25 6 13 59 53

void BST (node *root);

O(n), O(n)

{ if root = null:
    return

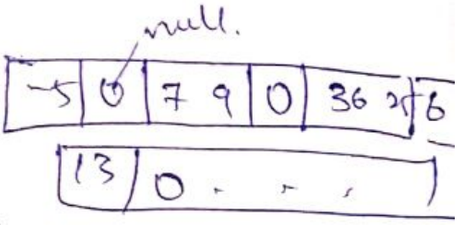
queue<node> q;
while (q != empty()) { k = q.pop-front()
print(q.pop().data); print(q.pop().data);

if k->left != null:
    q.push_back(k->left);
if k->right != null:
    q.push_back(k->right);
}

}

* If we want to print in like
-5
7 9
36 25 6 13
59 53 27

Sol:

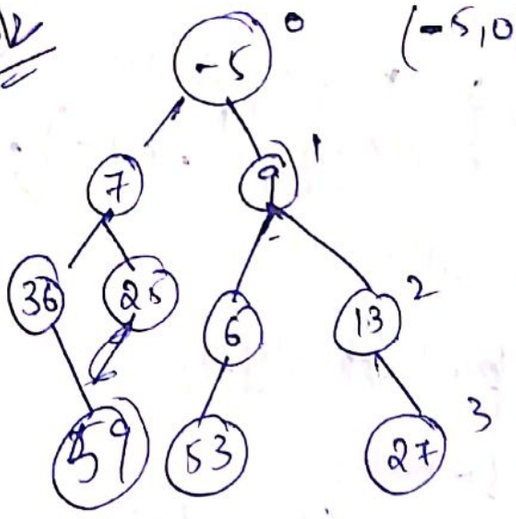


Void BST(node *root):
{ queue<node> q;
q.push(root); q.push(null);
while (!q.empty())
{ if k = q.pop-front();
if k == null && q == empty()
if k==Null { print("\n"); } q.push_back(null); }
{print("\n");} else { print(k.data, end=" "); }

if k->left != null:
    q.push(k->left);
if k->right != null:
    q.push(k->right);
}

* Depth along with data

Sol



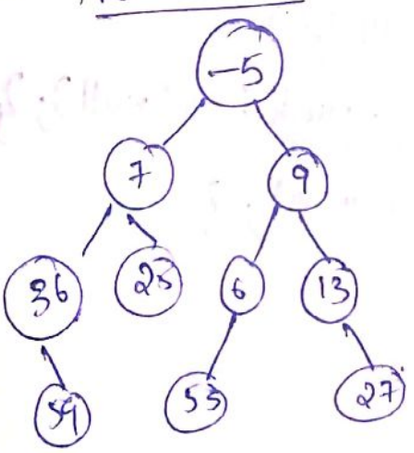
(-5,0) (7,1) (9,1), (36,2) (25,2)

Sol3

*

Sol4

* Two Queues:



Q1 [-5 36 25 6 13] . . .

Q2 [7 9 59 53 27] . . .

-5 \n
7 9 \n
36 25 6 13 \n
59 53 27 \n

Q1 & Print (\n when Q2 = empty

15:- Sorted HashMap using depth.

0 :- 5

1 : 7, 9

2 : 36, 25, 6, 13

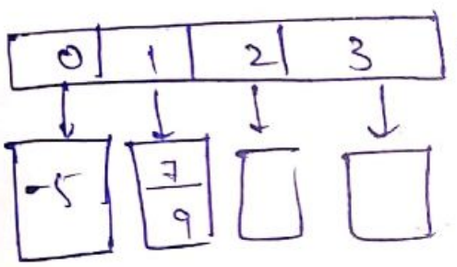
3 : 59, 53, 27

16 UnSorted HashMap:- using depth: (Search and print)

min_depth = 0

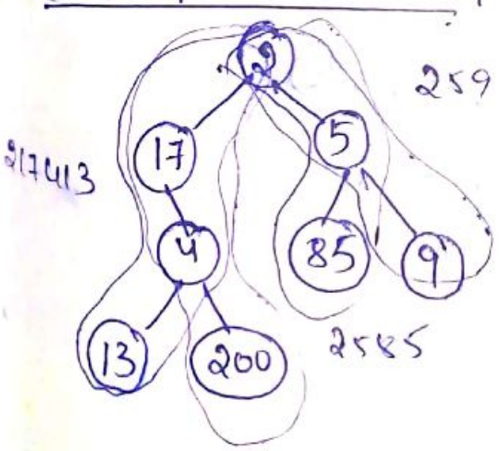
max_depth = 3

17: Lists of lists: with depth



---

* (Value of Root to leaf :)



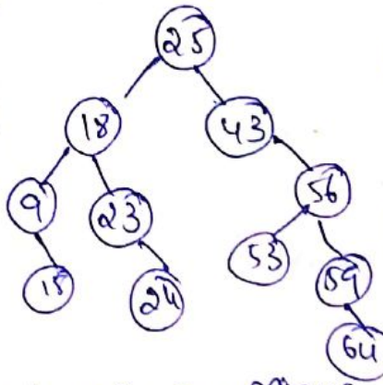
Sum = 2174200 + 217413 + 2585 + 259

```
int sum(node *root, int val)
{ if (root==Null): ret 0;
  val = val * pow(10, no of digits(root->data))
        + root->data;
  if (root->left==Null and root->right==Null).
  { ret val; }
  ret sum(root->left) + sum(root->right);
}
```

* BST: (No duplicates)

Inserting: Node *insert (Node* root, int ele)

O(Height) [O(n)]



Max(L) < D < Min(R)

```
{ if (ele < root->data)
        //insert
    if (root == NULL)
        return newnode(ele));
    if (ele< root:data )
    { root->left = insert(root->left, ele) }
    else { root->right = insert(root->right, ele)}
    return root;
}
```

Search:

```
bool search (node * root , int ele)
  { if (root == NULL) return false;
    if (ele == root->data) :
        return True
    if (ele < root->data)
        return search( root->left, ele)
    else
        return search (root->right, el);
```

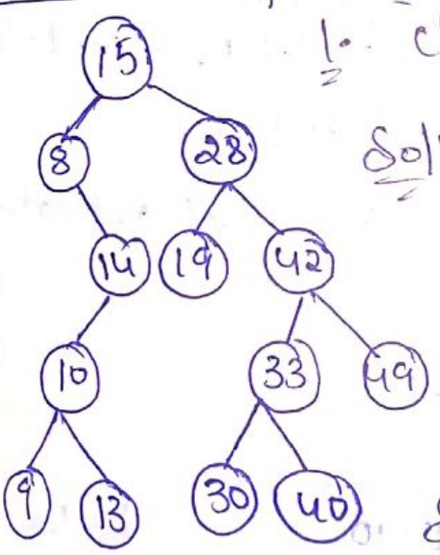
delete

```
delete (node* root , int ele)
{   if (root == Null)
        return null / root;
    if (ele < root->data)
root->left = delete(root->left, ele)
    else if (ele > root->data)
root->right = delete(root->right, ele)
```

```
else if (root.right == Null && root.left == Null):
        return Null;

    elif (root.left == Null)
        return root.right;

    elif (root.right = Null)
        return root.left;

    else { root.data = getMin(root.right)
root.right = delete(root.right, root.data);
        }
    return root;
}
```

BST (with Duplicates) :- All(L) < D < All(R)

1. check if given Binary Tree is BST



Sol: Inorder:

8 9 10 13 14 15 19 28 30 33 40 42 49

Time O(N + N) Space = O(N)

inorder check if sorted

Sol: Maintain a variable = INT_MIN

Time: O(N), Space O(1)

inorder

```
bool isBST(node* root, int prev)
{
    if root.data < prev:
        return false;
    return isBST(root->left,
```

bool BST (node * root, int prev)

{ if root == Null:

return True.

if (isBST(root.left, prev) == false)

return false;

if root.data < prev:

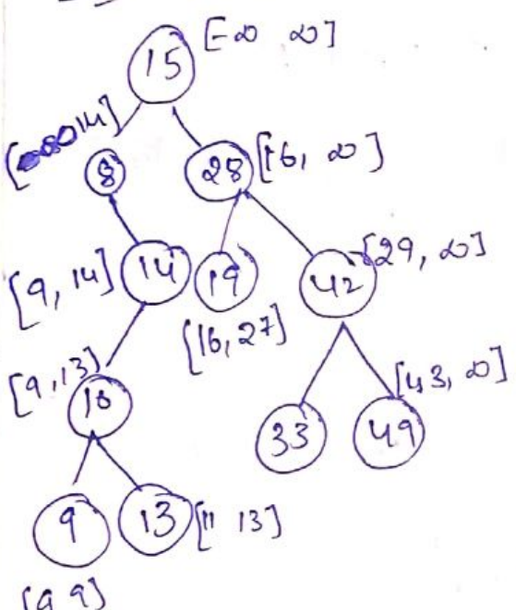
return false;

prev = root.data

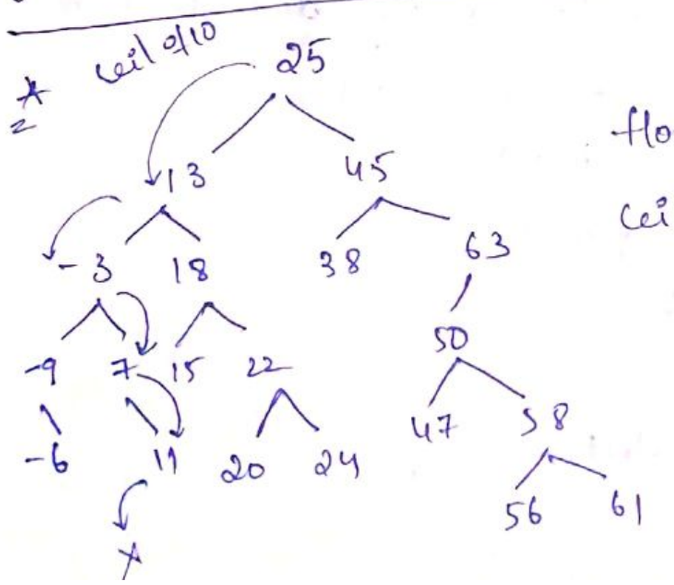
return isBST (root.right, prev);

}

Sol 8: Boundaries:




```
bool isBST(node * root, int a, int b)
{ if root == Null:
      return True
  if (root.data ≥ a and root.data ≤ b)
  { return isBST(root.left, a, root.data-1)
                and
           isBST(root.right, root.data+1, b);
  }
  else return false
}
```

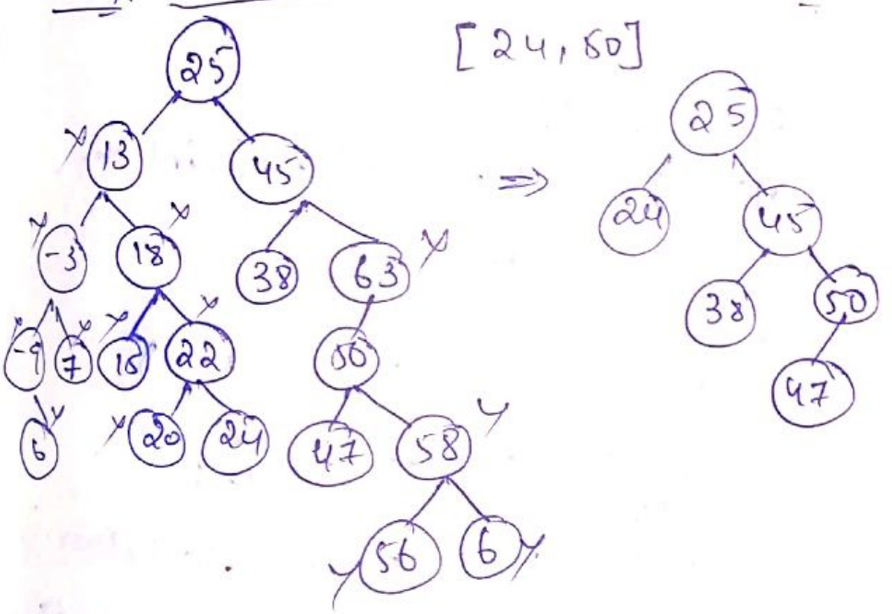


	K = 10	K = 52
floor	7	50
ceil	11	56

Sol:
```
int ceil (node * root, int key)
{ if root == Null : return INT_MAX;
  if (root.data == key) return key;
  if root.data < key:
        ceil(root.right, key)
  else: k = (root.data)
        l = ceil (root.left, key)
        return min (k, l)
}.
```

---

*TRIM. elements that are out of Range



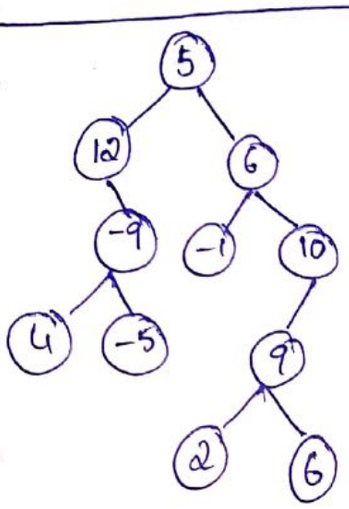
Sol 1:- Check each time from the root and if root is not in range, delete it

for N nodes, delete -> H => O(N×H), O(1)

Sol 2:
```
node * trim (node * root, int a, int b)
{ if (root == NULL)
      return NULL;
  if (root.data < a):
      return trim (root.right, a, b);
```

```
if (root->data > b):
    return trim(root->left, a, b);

root->left = trim(root->left, a, b);
root->right = trim(root->right, a, b);

return root;
}
```

Q.1.

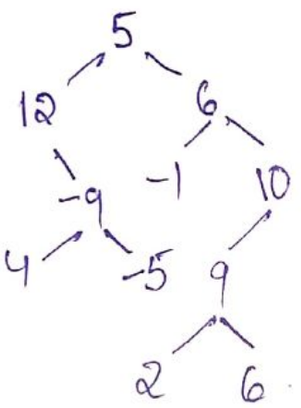


Preorder: 5 12 -9 4 -5 6 -1 10 9
Root LR

inorder: 12 4 -9 -5 (5) -1 6
L Root R start idx idx+1

Post order:

Sol:-




```
int pos=0;
node * f(int pre[],int in[],int st
         int end,
{ idx = search(in, pre[pos]);
  root = node(pre[pos]);
  pos++;
  root->left = f(pre,in, start, in
  root->right = f(pre,in, idx+1, e
  print(root->data);
  return root;
}
```

Sol: without creating tree:

```
void f(int pre[], int in[], int start, int end)
  if start >= end : return;
{ idx = search(in, pre[pos]);
  pos++;
  f(pre, in, start, idx-1);
  f(pre, in, idx+1, end);
```

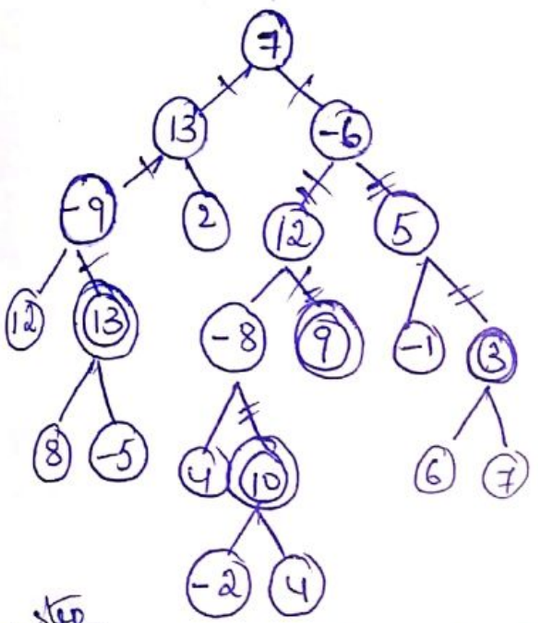
print(in[idx]);

}

Sol3: Maintain map <ele, idx> for efficient searching in inorder

---

2. Difference of path between 2 nodes.



[13,9] => 6. (3+3).

[10,3] => 5 (3+2)

10 -9 12 -6 7

3 5 -6 7

Sol: step 1. Make 2 lists of the elements that contribute the path

```
void path(node *root, node x, vector<nodes> l)
{   if root==None || x==None:
        return ;  (false)
    if (root==x) || (l.size>0)  (true)
    { l.insert(root); return; }
    if(path(root.left, x, l);)  (l.insert(root) ret true)
    if path(root.right, x, l);  (l.insert(root) ret true)
}
```

step 2. Remove common nodes from both lists and sum the remaining nodes from 2 lists.

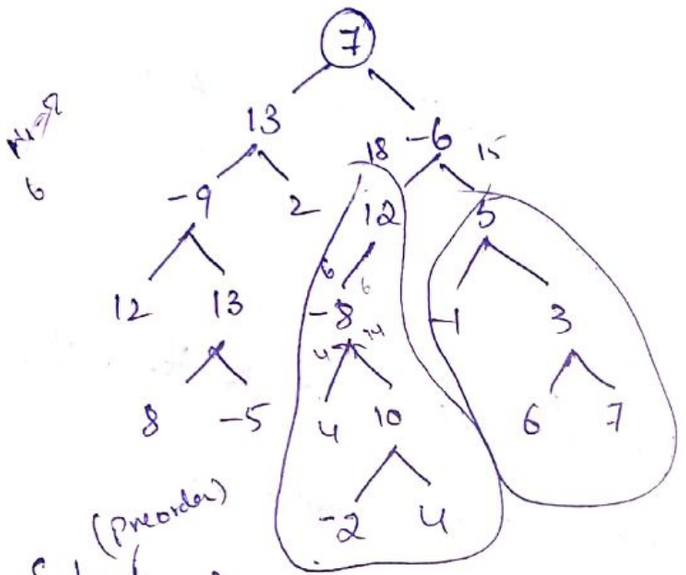
```
int f(node * root, node x, node y)
{ vector<node> l1, l2;
   path(root, x, d1);
   path(root, y, d2);
   return d1.size() + d2.size()
}
```

$\Rightarrow d_1 = 13 \; -9 \; 12$ (P1)

$d_2 = 9 \; 12$ (P2)

$P1 + P2 = 6$. (3 + 3)

Maximum Sum in tree:



(preorder)

Sol 1: $N + N^2$ paths $\times N$, N

Sol 2:

```
static int ans=0;                    O(n^2), O(1)
int solve(node *root)
{ int val = root.data + sum(root.left) + sum(root.right)
  ans = max(val, ans)
  solve(root.left)
  solve(root.right)
}
int sum(root left):
{ if root == null:
      return 0;
  return max(sum(root.left), sum(root.right))
```

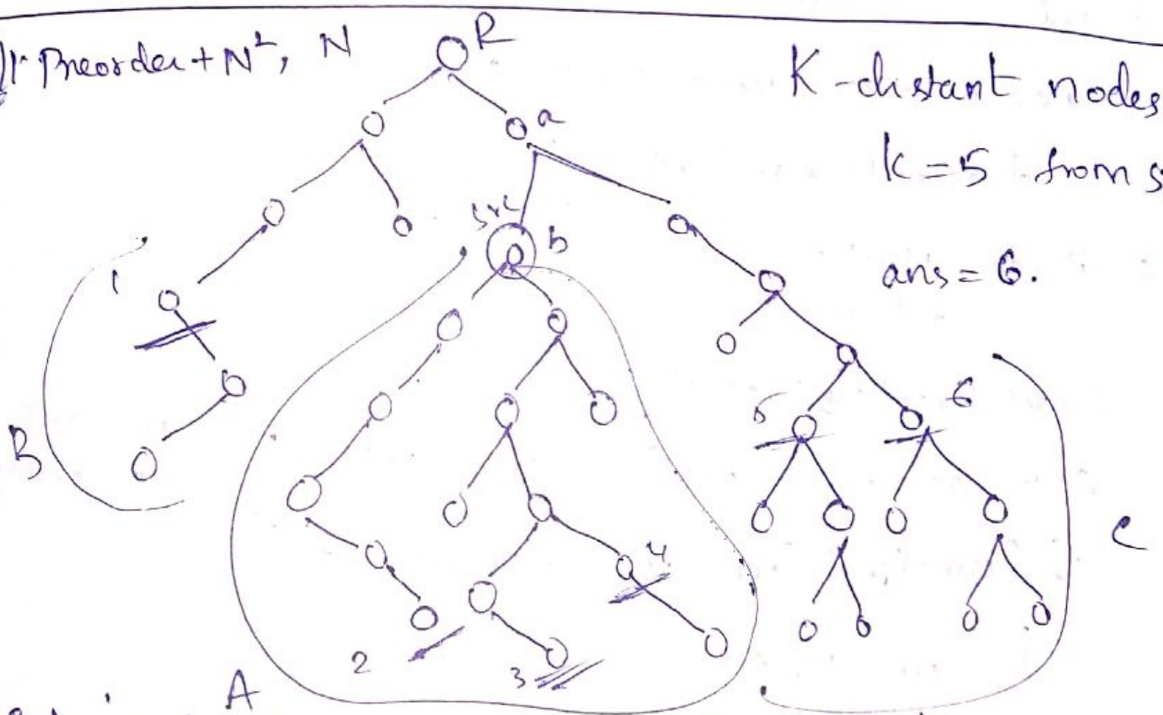
root.data);

Sol3: No need fore every element

```
int solve(node * root)
{ if root == Null:
      ret 0;
  l = solve(root.left);
  r = solve(root.right);
  ans = max(ans, root.data + l + r);
  ret max(0, max(l,r) + root.data);
}
```

---

* Sol: Preorder + N^2, N



K-distant nodes

k = 5 from src

ans = 6.

Sol2: int solve(node * root, node * src, int k)

```
{ L = find path(src);
  for i in range(len(L)):-
      if L[i] == (L[i+1].left):
          r = solve(L[i+1].right, k-i-1)
      if (L[i] == L[i+1].right):
          l = cnt(L[i+1].left, k-i-1)   ans += (l + r)
}
```

L: [b a R]
i 0 1 2

count = 0.

```
int cnt(node * src, int k)
{   if src != none and k == 0:
        cnt += 1
    if src == none:
        ret 0;
    if k == 0:
        ret 1;
    ret cnt(src.left, k-1) + cnt(src.right, k-1);
}
```

(For A part counting)

Print in Vertical order:

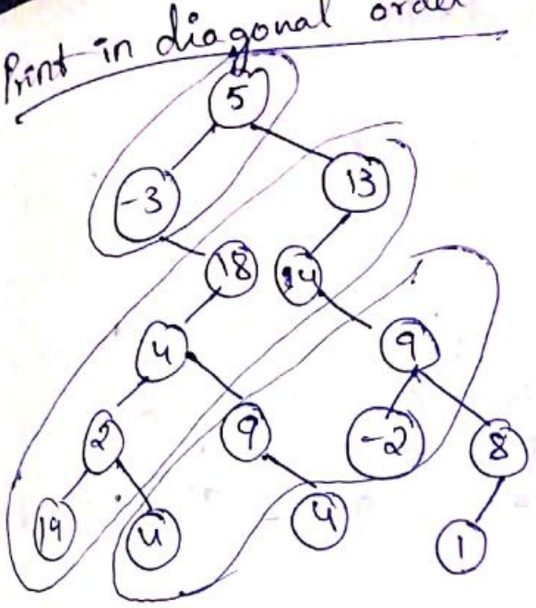
left levels = -ve.
Right levels = +ve

```
int min=      int max=
unordered_map<int, vector<int>> um;
void solve(node * root, int v)
{   if root = Null:
        return;
    Solve(root.left, v-1)
    Solve(root.right, v+1)
    if (um.find(um[v] == um.end()))
    {   um[v] = initialize vector;
        um[v].push_back( );
    }
    else
    {   um[v].push_back( );
    }
```

level.

min, max = min(min, v), max(max, v);

Print in diagonal order



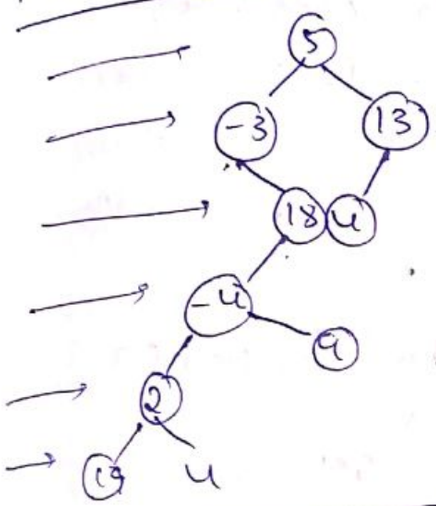
left levels ⇒ Right levels ⇒ +ve

Same as above code but

solve(root.left, V+1).

Print left view.

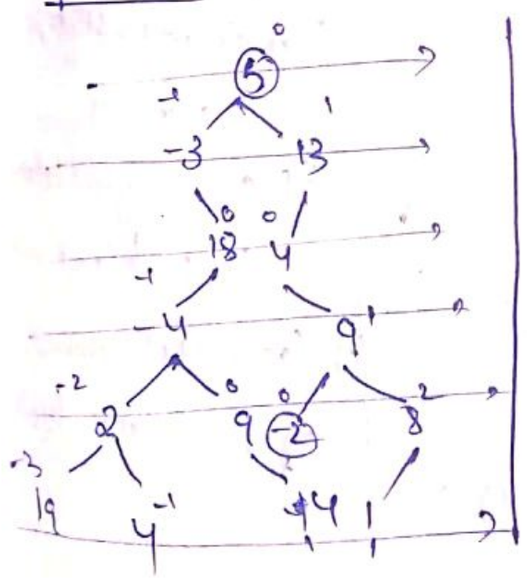
level order + first ele in level.



Right view.

level order + last node.

Top view. Vertical order + first ele



× Vertical order (But not getting crt ans)

-3 : 19

-2 : 2

-1 : -3, -4, 4, ~~-2~~ -2

0 : 5, 18, 9, 4, ~~-2~~

1 : ~~4~~, 1 4, 13, 9, 1

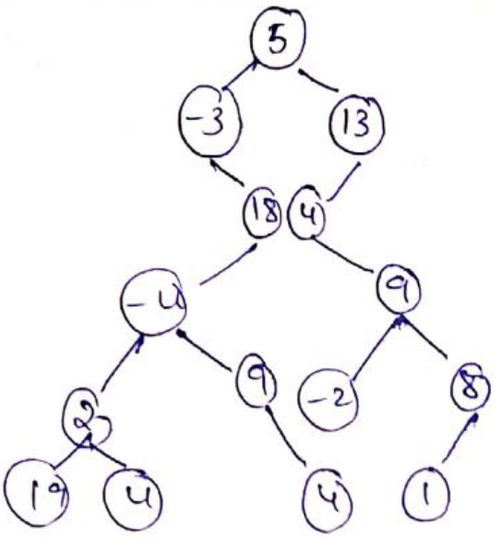
2 : 8

Ans = 19, 2, -3, 5, 13, 8

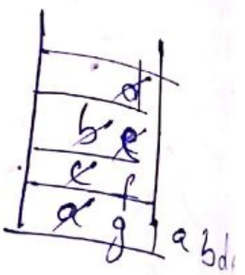
Horizontal order

int	int
-3	19
-2	2
-1	-3
0	5
1	13

* Iterative code for Preorder using stacks:



Preorder: 5 -3 18 -4 2 19

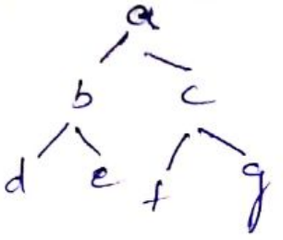



```
void preorder(root):
    Stack<nodes>
    if (root != Null):
        st.push(root);
    while(len(st)!=0)
    { printf(st
        k = st.pop();
        print(k.data);
        if k.right != None:
            st.push(k.right);
        if k.left != None:
            st.push(k.left);
    }
```

```
class preorder{
    stack<nodes> st;
    preorder(node root).
    { if (root == Null);
        st.push(root);
    }
    bool hasnext()
    {
        ret st.size!=0;
    }
    node getnext()
    { node root=st.top
        st.pop();
        if (root.right!=No
            st.push(root.ri
        if (root.left!=None)
        st.push(root.left)
```

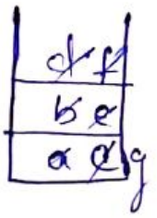
```
preorder obj = new Preorder(root)
while (obj.hasnext())
{ node x = obj.getnext()
} cout << x.data;
```

* Inorder



inorder: d b e a f c g

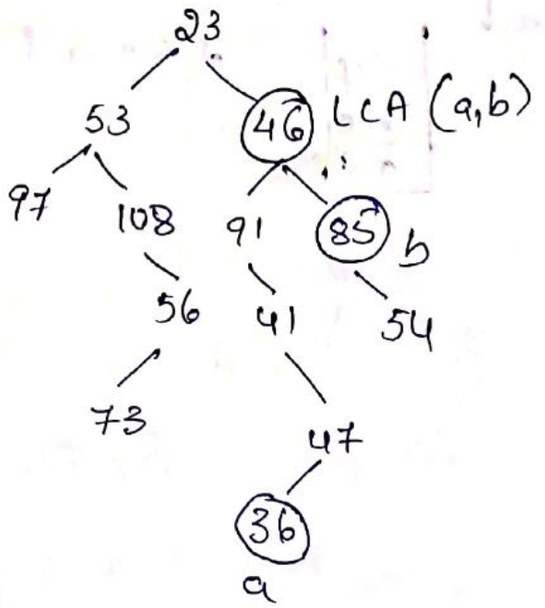
Stack



d b e a f c g

```
Void inorder(node *root)
{ Stack<node>
  if (root != Null)
     { st.push(root); }
  while( st!.empty() )
  { temp=root
    while (temp.left! =none):
      {St.push(temp left)
       temp = temp.left;
      }
    root = temp;
    Gem k = st.pop()
    print(k.data)
    while (k.right! =none)
    { st.push(k.right);
      k=k-
    if (k.right! = None)
      { k= k.right; }
    root = k;
  }
```

*LCA - Least Common Ancestor:



Solutions:-

1. find elements and store them in plists

l_1 | 85 | 46 | 23 |

l_2 | 36 | 47 | 41 | 91 | 46 | 23 |

2. Postorder

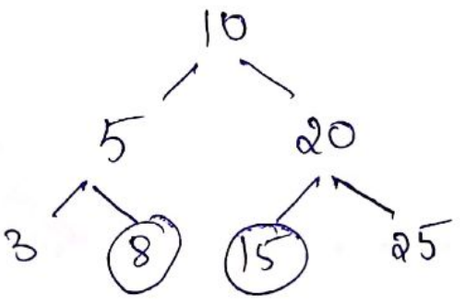
If left and right subtrees return true, return true

Postorder → O(n), O(1).

---

* LCA for BST:-

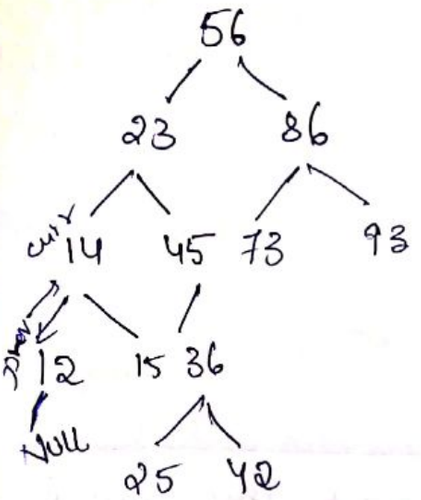
find target elements



if a,b = 8, 15

↳ LCA = 10.

* BST to Sorted Doubly linked list



BST

```
class Node
{ int data;
  node * left, right
}
```

SDLL

```
class Node
{ int data;
  node * prev, next;
}
```

→ Res: 12 ⟷ 14 ⟷ 15 ⟷ 23 ⟷ 25 ⟷ 36 ⟷ ⟷ ⟷

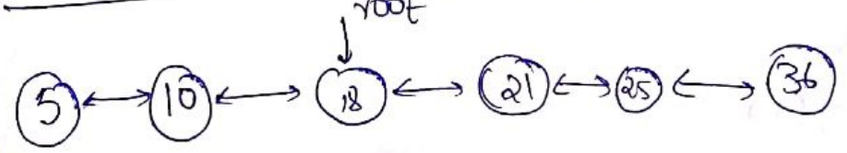
Inorder:- to get elements in sorted order.

Prev = None;

Sol:

```
Node f (node * root)                    O(n), O(1)
{ if root is None: return None          global prev
  f (root.left)
  root.prev = K
  root.left = prev;
  prev.right = root;
  prev = root;
  f (root.Right)
}
```

* Sorted DLL to Balanced BST:




```
node f (node * head)
{ root = findmid(head, false);
  root.prev.next = root.next.prev = nlong
```

root.prev = f(head)

root.next = f(root.next);

return root;

}

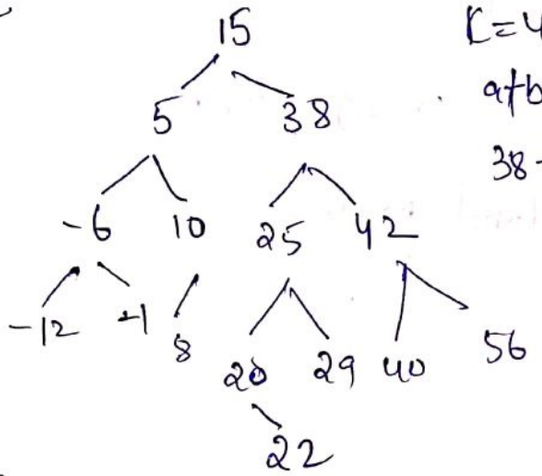
Comp for every node, constructing Tree

$O(N \times \log_2 N)$, $O(1)$

---

* Given a BST, find if ∃ 2 elements → a,b that sum to K

→



K=43

a+b = K

38+5=43 ✓ (return True)

Solutions:

1. Inorder → store in Array + 2ptr. | space O(N)

O(N) O(N)

↳ O(N), O(N)

2. Search for every element

take 15, 43-15 = 28

Search for 28

$O(N \times H)$, $O(1)$

3. Convert BST → SDLL, 1 + 2ptr

O(N) + O(N), O(1)

↳ O(N), O(1).

4. Sets

O(N), O(N).

5. p1 at 12] Two ptr technique
p2 at 56

p1 → Inorder [L D R]

p2 → Reverse Inorder [R D L]

```
Inorder O1 = new Inorder(root);
ReverseInorder O2 = new ReverseInorder(root);
Node p1 = O1.getnext();
Node p2 = O2.getnext();
while(p1 != p2)
{ if p1.data + p2.data == k
      return True
  elif (p1.data + p2.data < k)
      p1 = O1.getnext();
  else
      p2 = O2.getnext();
}
```

→ O(n), O(H)

TODO Threaded Binary Tree
Morris Traversal
AVL & Red-Black Tree

Table:-

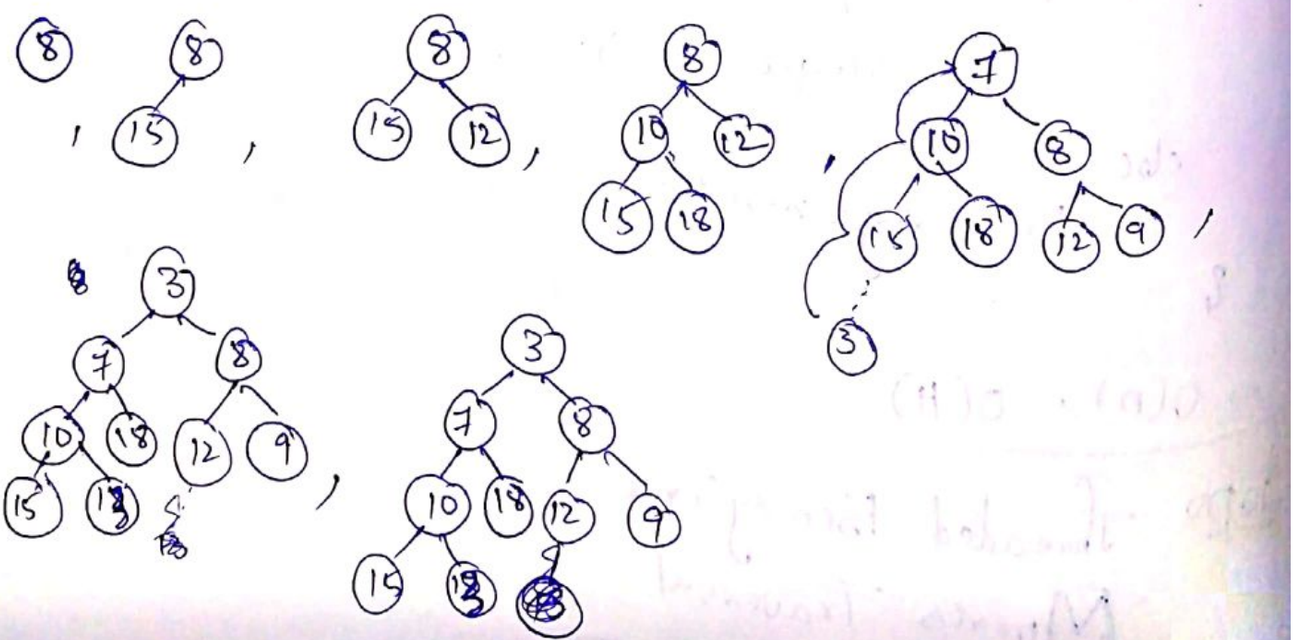
Solution	insert(x)	deleteMin( )	getMin( )
* Unsorted Array	1	N	N.
* Sorted Array	N	N	O(1)
* Linked lists singly	1	N	N
* Sorted list	N	1	1
* BST	N.	N	N.
* BBST	$\log_2 N$	$\log_2 N$	$\log_2 N$.
* Minheap	$\log_2 N$	$\log_2 N$	1.

HEAPS

Min heap: — [CBT, parent <= child.

Max Heap — [CBT, parent >= child

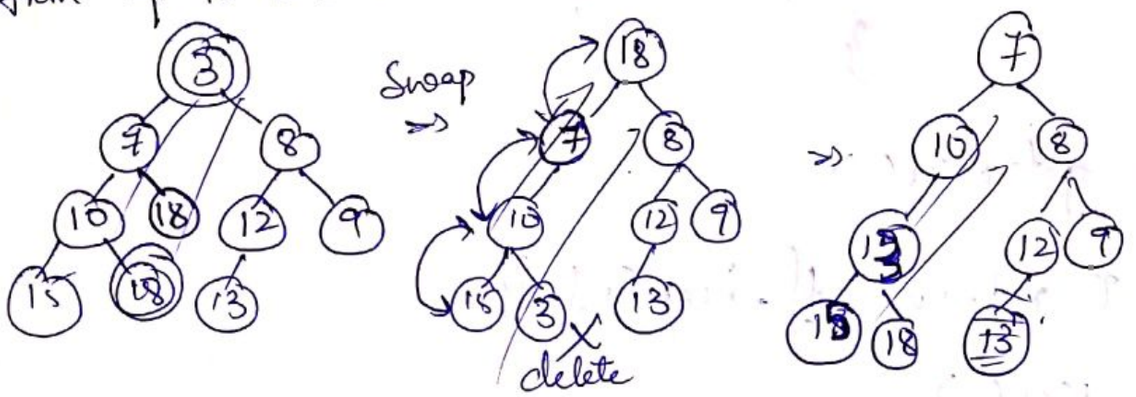
8, 15, 12, 10, 18, 7, 9, 3, 13.

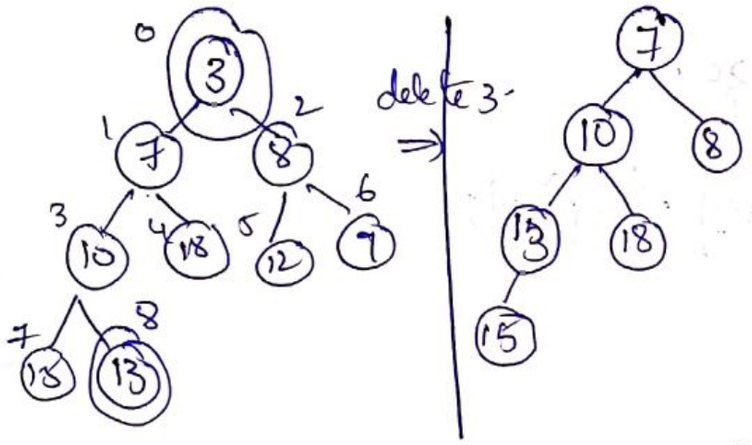


DeleteMin():

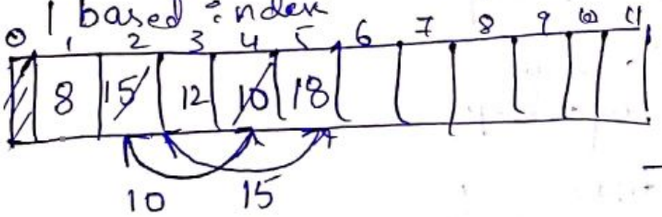
→ Replace the root with the last element

→ Adjust the Arrangement of tree to get min at the root from top to bottom





* Insertion using arrays:



index	Parent → Child
0	$i \rightarrow 2i+1,\ 2i+2$
0	$\lfloor \frac{i-1}{2} \rfloor \leftarrow i$
1	$i \rightarrow 2i,\ 2i+1$
1	$i/2 \leftarrow i$

```
class Minheap
{   def init__(self):
        self.size = 0;
        self.heap = [0];

    def insert(self, data):

        self.heap.append(data).
        self.size += 1
```

```
idx = self.size
while (idx>0 && self.heap[idx] < self.heap[idx//2]
        { swap (self.heap[idx], self.heap[idx//2]
            idx //= 2;
        }

def getMin()
  { return self.heap[1]; }
def sizeC)
  { return self.size;
        or
    return len(self.heap)-1;
  }
```

Inbuilt libraries:-

C++ → Priority_queue – (default minheap)

Java → PriorityQueue – (default maxheap)

Python → heapq : – [default maxheap].

---

* Ar:

0	1	2	3	4	5	6	7	8
10	3	15	8	5	18	-3	19	6

k=4. Print k smallest elements in any order

[-3 3 5 6]

Solutions:-

1. Sorting ⇒ $O(N\log_2 N)$, $O(1)$
   (quick sort).
2. Minheap ⇒ $O(N\log_2 N + k\log_2 N)$, $O(N)$ (insert, deleting)

3. Max heap

Printing

$K \log_2 K + N-K(\log_2 K + 1) + K\,O(K)$

↓ insert.

↓ compare & delete

overall $\longrightarrow O(N \log_2 K)$, K

4. Quick Select.

Best: $T(N) = T(K) + N$

Worst: $T(N) = T(N-1) + N \Rightarrow N^2$

Avg: $T(N) = T(\frac{9N}{10}) + N \Rightarrow O(N)$.

↳ $T((\frac{9}{10})^2 n) + \frac{9n}{10}$

↳ $T((\frac{9}{10})^3 n) + (\frac{9}{10})^2 n + \frac{9n}{10} + n$

↳ $\left[1 + \cdots + (\frac{9}{10})^k n + \frac{9}{10} n\right] + n$

$\Rightarrow n\left(\frac{1 - (9/10)^k}{1 - (9/10)}\right)$, $\frac{1}{1-(9/10)} < n$

$\Rightarrow \approx n$.

$\rightarrow O(n)$, $O(1)$.

Quicksort —

Best: $T(n/2) + T(n/2) + n$
$= n \log_2 n$.

Worst: $T(n-1) + T(1) + n$
$= n^2$

Avg: $T(\frac{9n}{10}) + T(\frac{n}{10}) + n$
$= n \log_2 n$