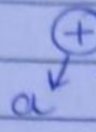
14-JAN-22

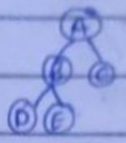
Date:
Page No.

BINARY TREE

→ A tree in which each node (parent) has at most two-child nodes (left and right) is called Binary Tree.

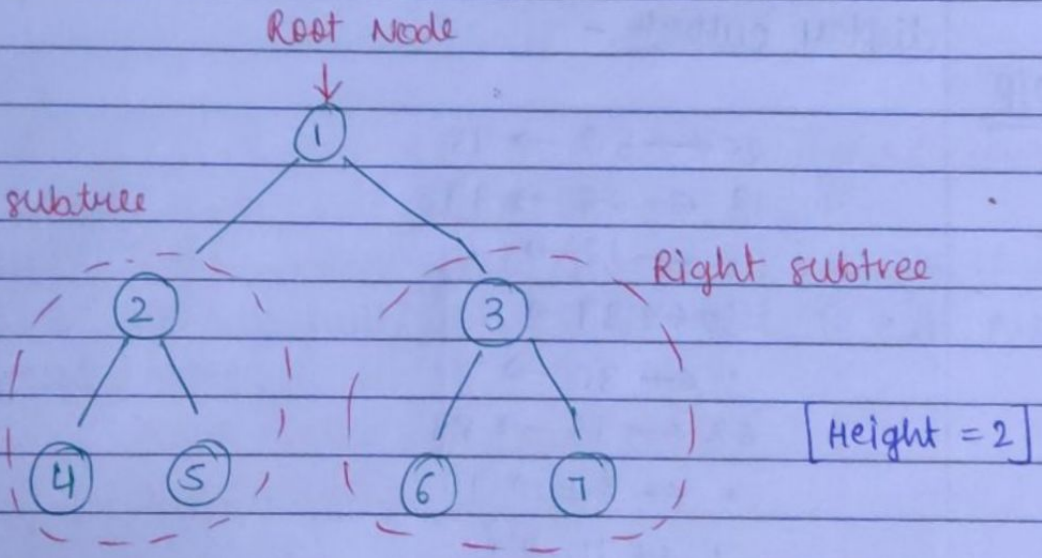
ex - a+b





→ The top most node is called root node.
In binary tree a node contains the data & the pointer (address) of the left and right child node.

If the tree is empty, the height is 0.

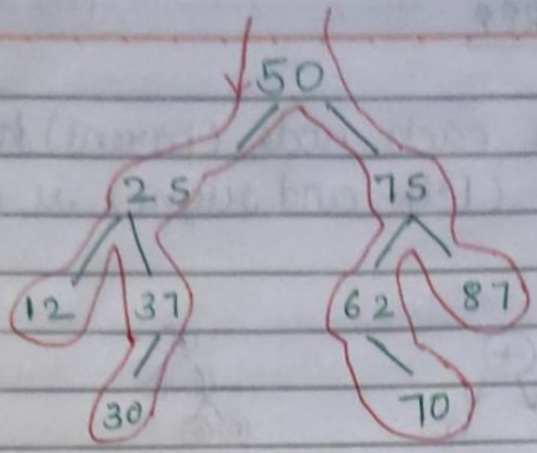


→ Binary trees are mainly used for searching and sorting as they provide a means to search data hierarchically.

→ Some common operations that can be conducted on binary trees include insertion, deletion and traversal.

1. BINARY TREE CONSTRUCTOR.

Date :
Page No.



I/P 19

50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n

↓

[Jb Integer[] arr (capital) bnahe n to null atta n]

display output -

O/P

25 ← 50 → 75
12 ← 25 → 37
. ← 12 → .
30 ← 37 → .
. ← 30 → .
62 ← 75 → 87
. ← 62 → 70
. ← 70 → .
. ← 87 → .

Date :
Page No.

```
import java.io.*; (Package for BufferReader)
import java.util.*;

public ~~static~~ class Main {
    public static class Node {
        int data;
        Node left;
        Node right;
    }

    public static Node construct(Integer[] arr) {

    }

    public static void display(Node node) {

    }

                                        throws Exception
    public static void main(String[] args) {
BufferReader    Integer[] arr = new Integer[] {50, 25, 12, null
    null, 37, 30, null, null, null, 75, 62, null, 70, null, null,
    87, null, null};
        Node root = construct(arr);
        display(root);
    }
}
BufferReader br = new BufferReader(new InputStreamReader(System.in));
int n = Integer.parseInt(br.readLine()); No. of integer pdhe (19)
Integer[] arr = new Integer[n]; utna bda array bnaliya
String[] values = br.readLine().split(" "); jo no. h space se split krliya
for(int i=0; i<n; i++) {
    if (values[i].equals("n") == false) {
        arr[i] = Integer.parseInt(values[i]);
    } else {
        arr[i] = null;  } } Node root =
```

Date :
Page No.

CONSTRUCT `public static Node construct(Integer[] arr) {`

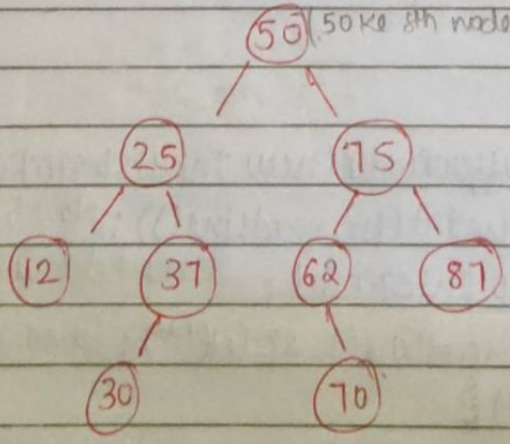
```
Stack<Pair> stack = new Stack<>();

Node root = new node(); //root bnaya uska data so set krdiya.
root.data = arr[0];     // arr[0] pe

Pair rootp = new Pair(); //rootpair bnaya.
rootp.node = root; // uska node root ke = kra
rootp.state = 1;

stack.push(rootp);
int idx = 1;

while (stack.size() > 0) {
```

3.

idx ↓ ↓ ↓↓ ↓ ↓ ↓↓↓ ↓ ↓ ↓ ↓ ↓↓ ↓ ↓ ↓

50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n.

1 → LEFT CHILD SET
2 → RIGHT CHILD SET
3 → POP.



Date :
Page No.

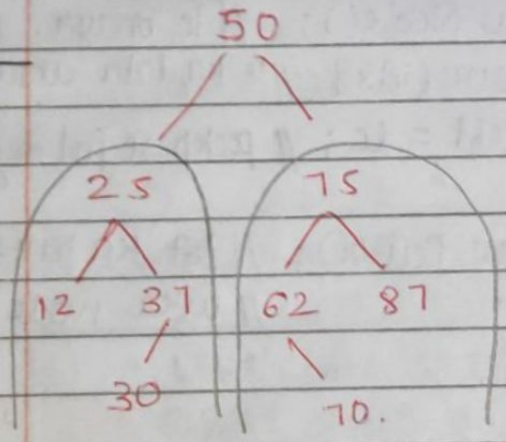
~~12 X3~~	pop.	~~30 X3~~	~~70 X3~~		
25 X2		37 X3	~~62 X3~~		87 X3
50 X2		~~25 X3~~	75 X2		75 X3
↑		50 2.	50 2.		50 X3

Pair bnaya state 1 ke sth or stack me push.

```
while (stack.size() > 0) {
    Pair peekp = stack.peek();

    if(peek.state == 1) {
    //left
        if(arr[idx] != null){
            Node lc = new Node();  // lc bnaya.
            lc.data = arr[idx]; // lc ka data array[idx] set kiya
            peekp.node.left = lc; // peekp se jod diya

            Pair lp = new Pair(); // left ka pair bnao.
            lp.node = lc;          // uska node lc
            lp.state = 1;

            stack.push(lp);        // (wko) lp push
        }
        peekp.state++;
        idx++;
    } else if (peek.state == 2) {
    // right.
        if(arr[idx] != null){
            Node rc = new Node();
            rc.data = arr[idx];
            peekp.node.right = rc;
```

```
Pair rp = new Pair();
rp.node = nc;
rp.state = 1;

stack.push(rp);
}
peek.state++;
idx++;
} else if(peek.state == 3) {
// pop.
stack.pop();
}
return root;
```

DISPLAY



25 ← 50 → 75.

25 & family
75 & family

(HIGH Level)

25 ← 50 → 75
12 ← 25 → 37
. ← 12 → .
30 ← 37 → .
. ← 30 →

left subtree

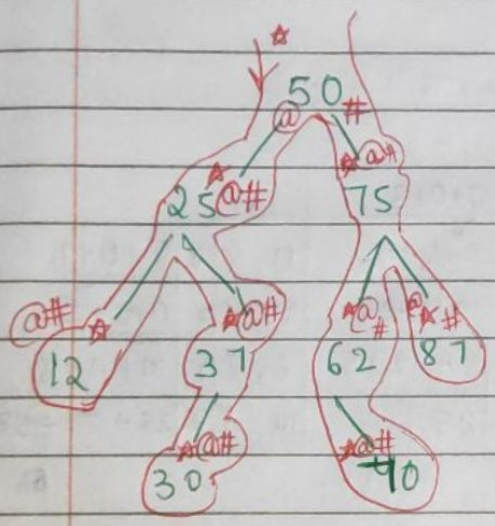
62 ← 75 → 87
. ← 62 → 70
. ← 70 → .
. ← 87 → .

right subtree

```
public static void display(Node node){
    if(node == null){
        return;
    }
    String str = "←" + node.data + "→";
    String lcstr = node.left == null?".":node.left.data+"";
    String rcstr = node.right == null?".":node.right.data+"";
    System.out.println(lcstr + str + rcstr);

    display(node.left); // will print the entire right tree
    display(node.right); // will print the entire left tree
}
```

node.left null h to . varna data print

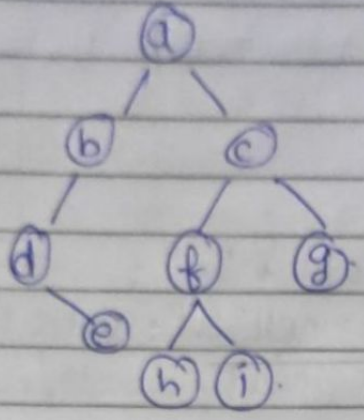
faith



2. SIZE, SUM, Maximum And Height Of A Binary Tree.

Date :
Page No.

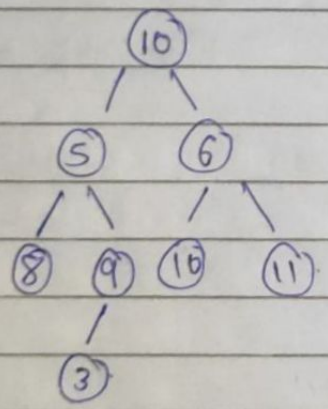
for size



e, l r	0+0+1		
d l r	0+1+1		
b l	2	b l r	2+0+1
a, l		a l	3.

h l.	0+0+1	i l r	0+0+1		
f l	1	f l r	1+1+1	g l r	0+0+1
c l		c l	3	c l r	3+1+1
a l r	3.	a r	3.	a r	3+5+1 = 9

for sum



3 l r	0+0+3		
9, l r	3+0+9	11, l r	0+0+11
8, l r	0+0+8	10, l r	0+0+10
5, l r	8+12+5	6, l r	10+11+6
10, l	25	10, l r	25+27+10
			= 62

int

SUM

```
public static sum (Node node) {
    if (node == null) return 0;

    int lsum = sum (node.left);
    int rsum = sum (node.right);

    return lsum + rsum + node.data;
}
// return (node == null ? 0 : sum(node.left) + sum(node.right) +
                                node.data);
```

Date :
Page No.

SIZE.

```
public static int size(Node node) {
    if(node == null) return 0;

    int ls = size(node.left);
    int rs = size(node.right);

    return ls + rs + 1;
}
```

// 1 line code.

// return (node == null ? 0 : size(node.left) + size(node.right) + 1);

3. act as base case.

MAX.

```
public static int max(Node node) {
    if(node == null) return Integer.MIN_VALUE;

    int lmax = max(node.left);
    int rmax = max(node.right);

    return Math.max(node.data, Math.max(lmax, rmax));
}
```

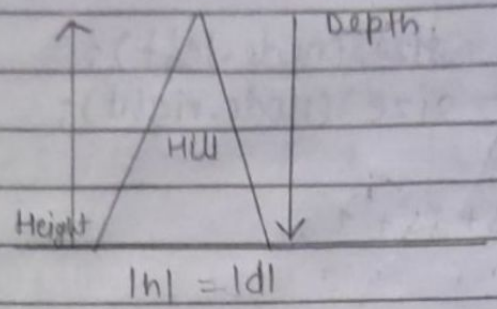
-∞ (se jo bdi value mil gai vhi dnu ↓ max.)

3. To compare root

// 1 line code

// return node == null ? Integer.MIN_VALUE : Math.max(node.data, Math.max(max(node.left), max(node.right)));

→ Height & Depth is same term magnitude wise only. (NOT DITRECTION WISE).



○ height|node = height|edge + 1.

0 = height|edge + 1

height|edge = -1 , (node == null).

HEIGHT
Edges term

```
public static int height (Node node) {.
    if (node == null) return -1;

    int lh = height (node.left);
    int rh = height (node.right);

    return Math.max (lh, rh) + 1;
}.
```

//1 line code.

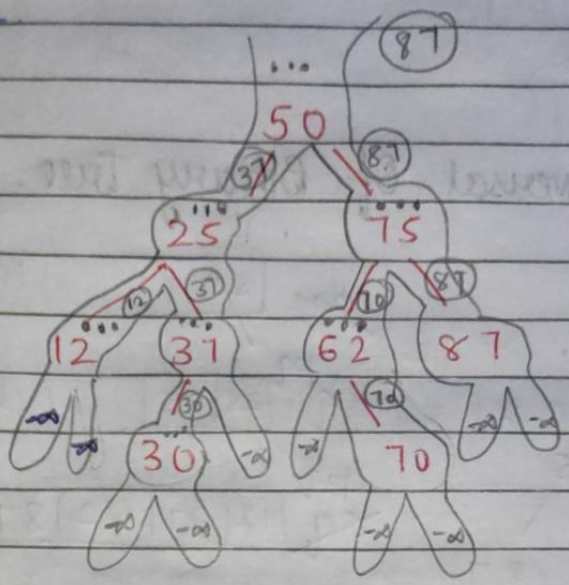
```
return node == null ? -1 : Math.max (height (node.
left), height (node.right)) + 1;
```

* Edges ki term me return -1
Node " " 0.

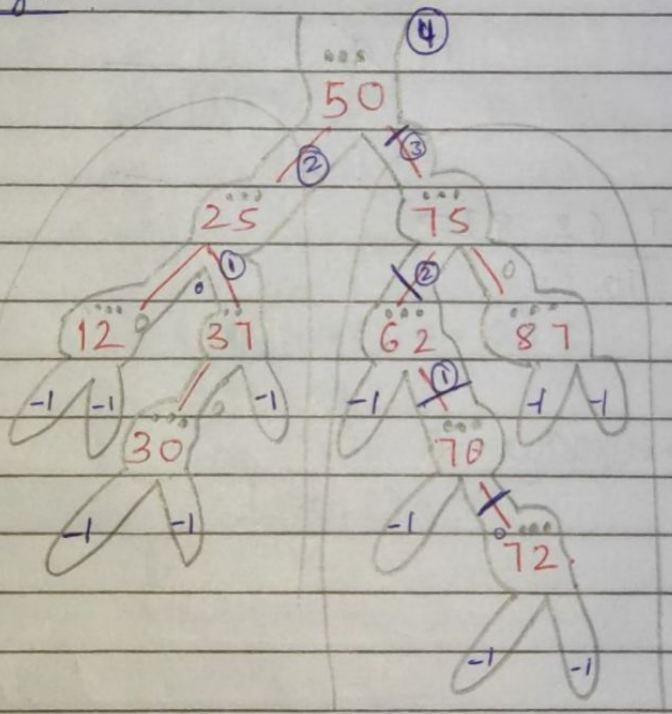
E = -1
N = 0

Date :
Page No.

max



height



(50) → 50 H in terms of Node 1
Edges 0.