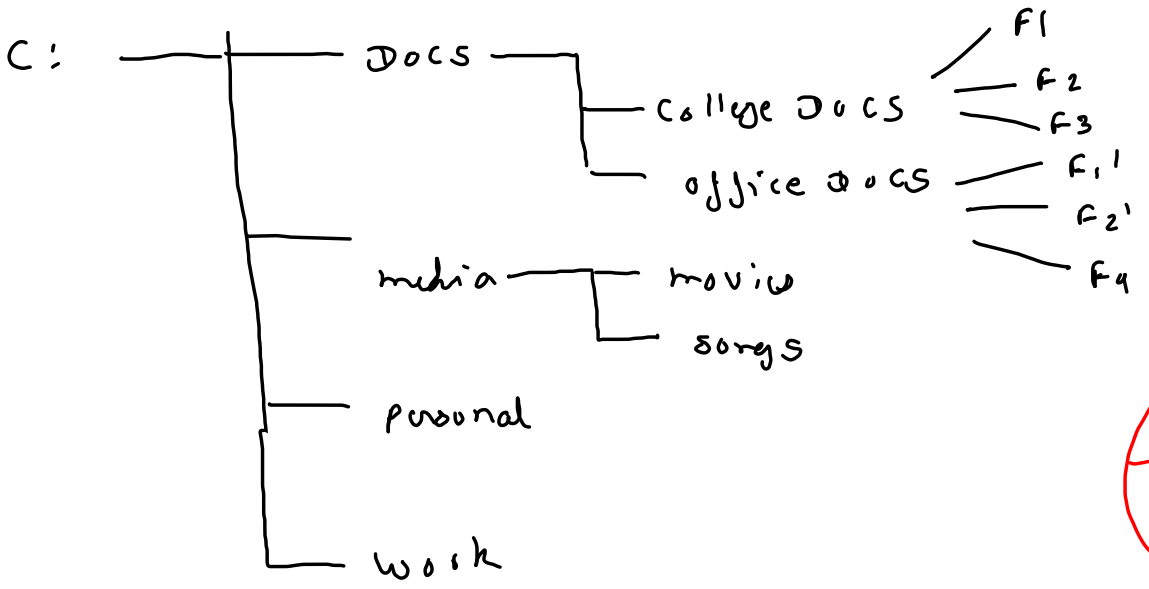
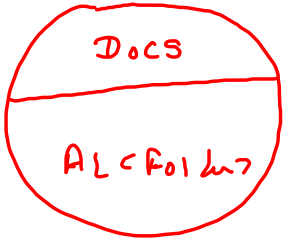
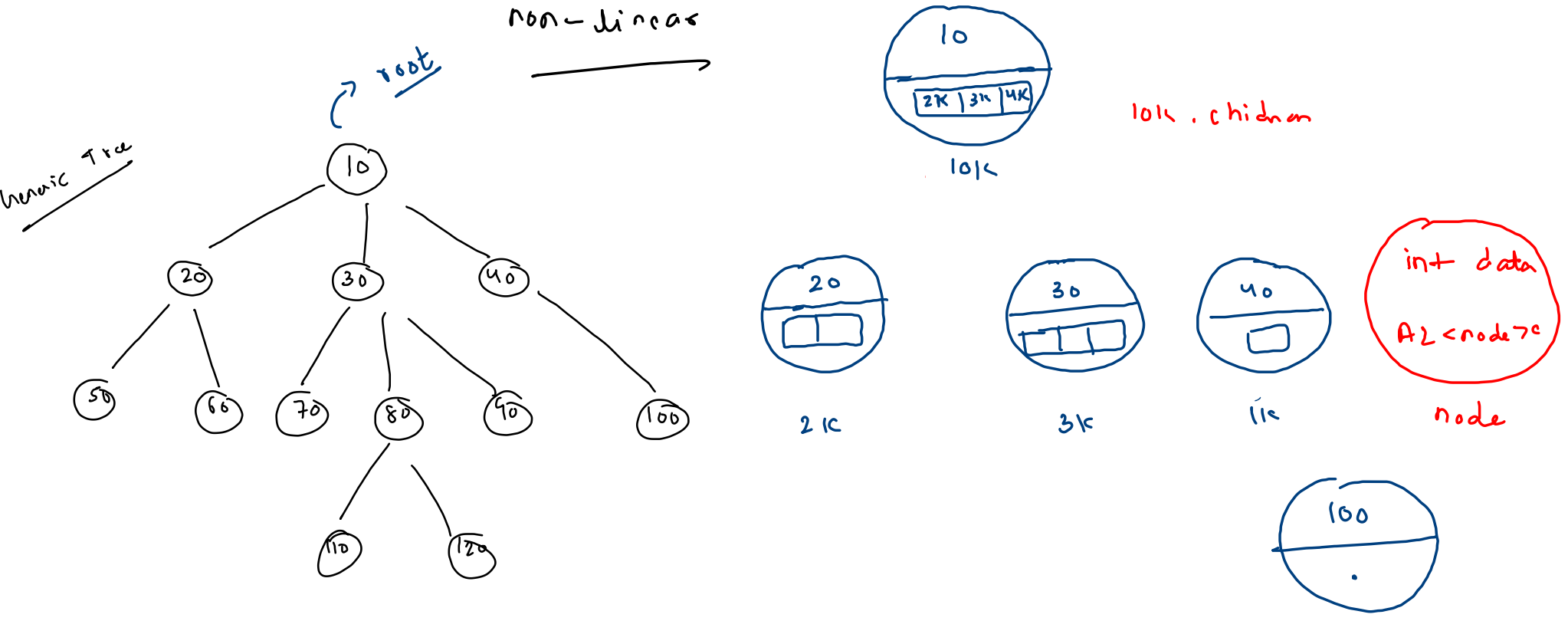




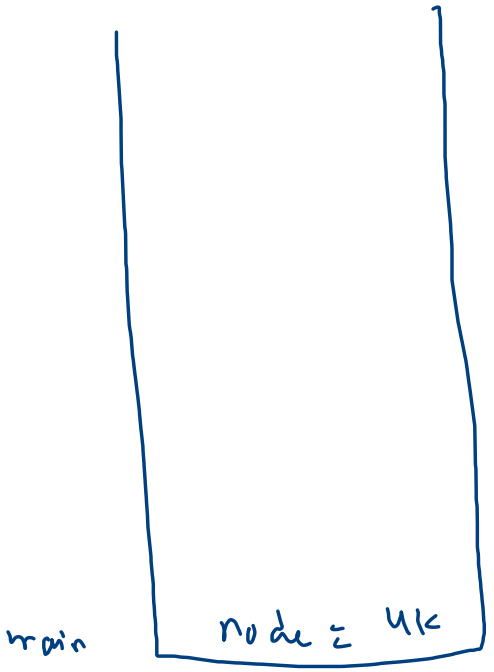
Fold

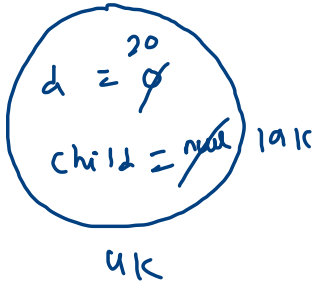
```
Fold {
    String name;
    AL <Fold> childFold;
}
```


non-linear
root
Generic Tree
10
20
30
40
50
60
70
80
90
100
110
120
10
2K
3K
4K
10K
10K . children
20
2K
30
3K
40
4K
int data
A2<node>c
node
100

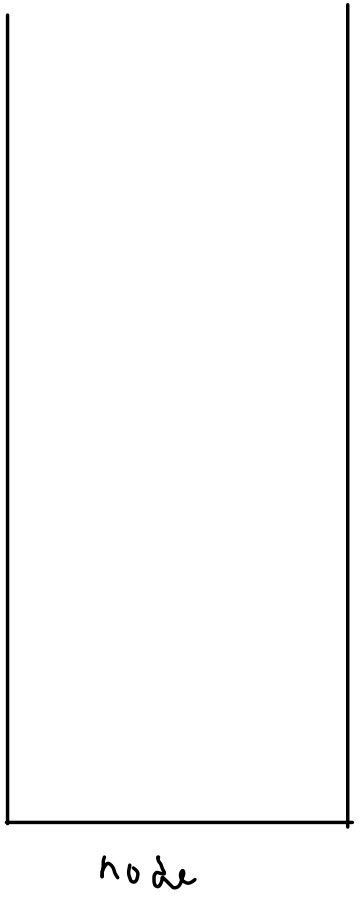
```java
public static class Node {
    int data;
    ArrayList<Node>children;

    Node() {

    }

    Node(int data) {
        this.data = data;
        children = new ArrayList<>();
    }
}
```

Node node = new Node(50);





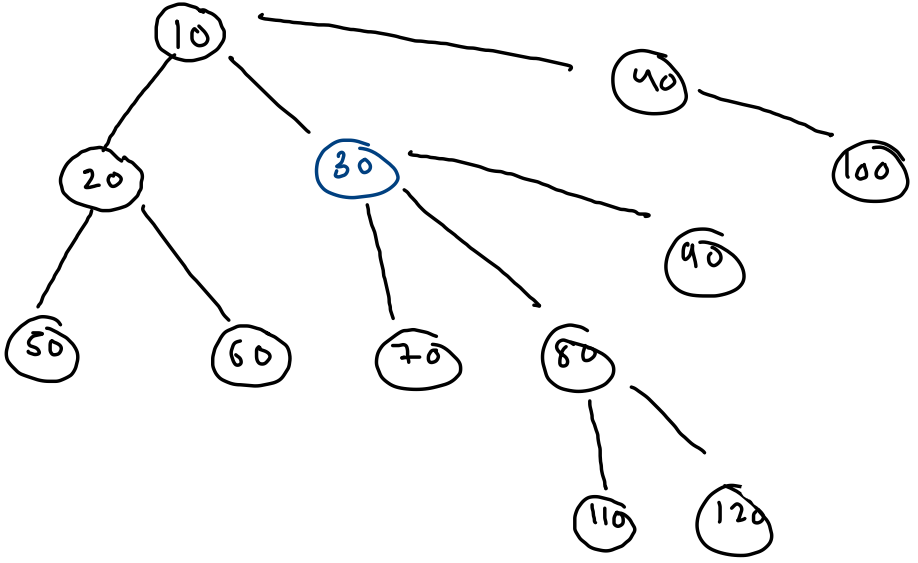
19k



dA ; 10 20 50 -1 60 -1 -1 30 70 -1

80 110 -1 120 -1 -1 90 -1 -1

40 100 -1 -1 -1 .



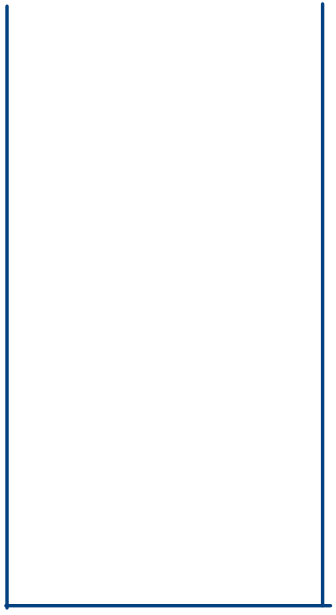
10 20 50 -1 60 -1 -1 30 70 -1

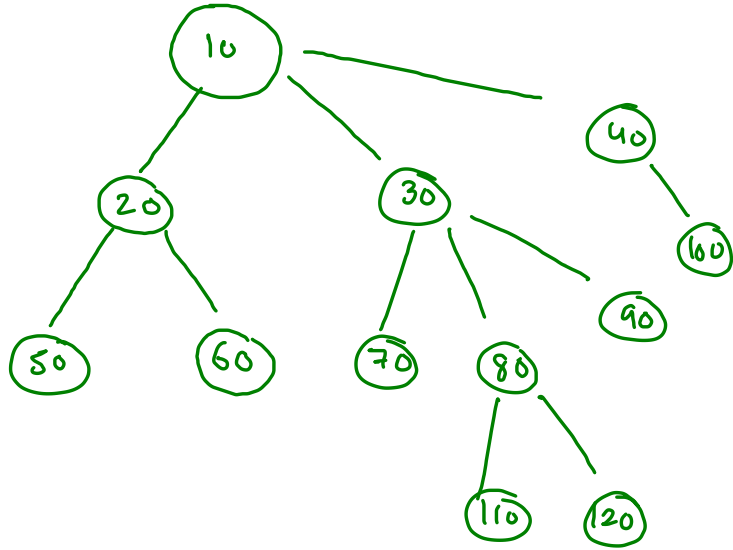
80 110 -1 120 -1 -1 90 -1 -1

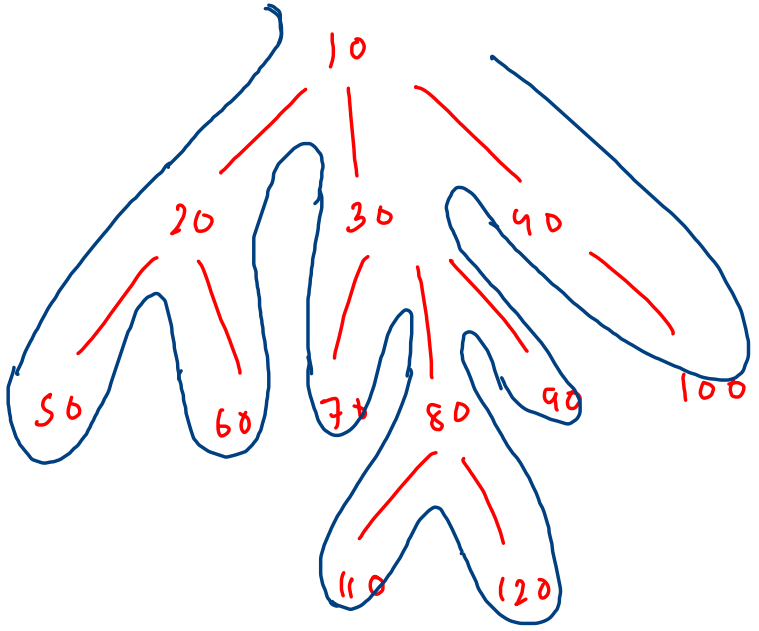
40 100 -1 -1 -1

root = ~~null~~ 10

```java
public static Node construct(int[]data) {
    Stack<Node>st = new Stack<>();
    Node root = null;

    for(int i=0; i < data.length;i++) {
        if(data[i] == -1) {
            st.pop();
        }
        else {
            Node nn = new Node(data[i]);

            if(st.size() > 0) {
                Node par = st.peek();
                par.children.add(nn);
            }
            else {
                root = nn;
            }

            st.push(nn);
        }
    }

    return root;
}
```







10 family : (i) 10 → 20 30 40

(ii) 20 family

(iii) 30 family

(iv) 40 family

10 → 20 30 40.

20 → 50 60.

50 → .

60 → .

30 → 70 80 90

70 → .

80 → 110 120

110 → .

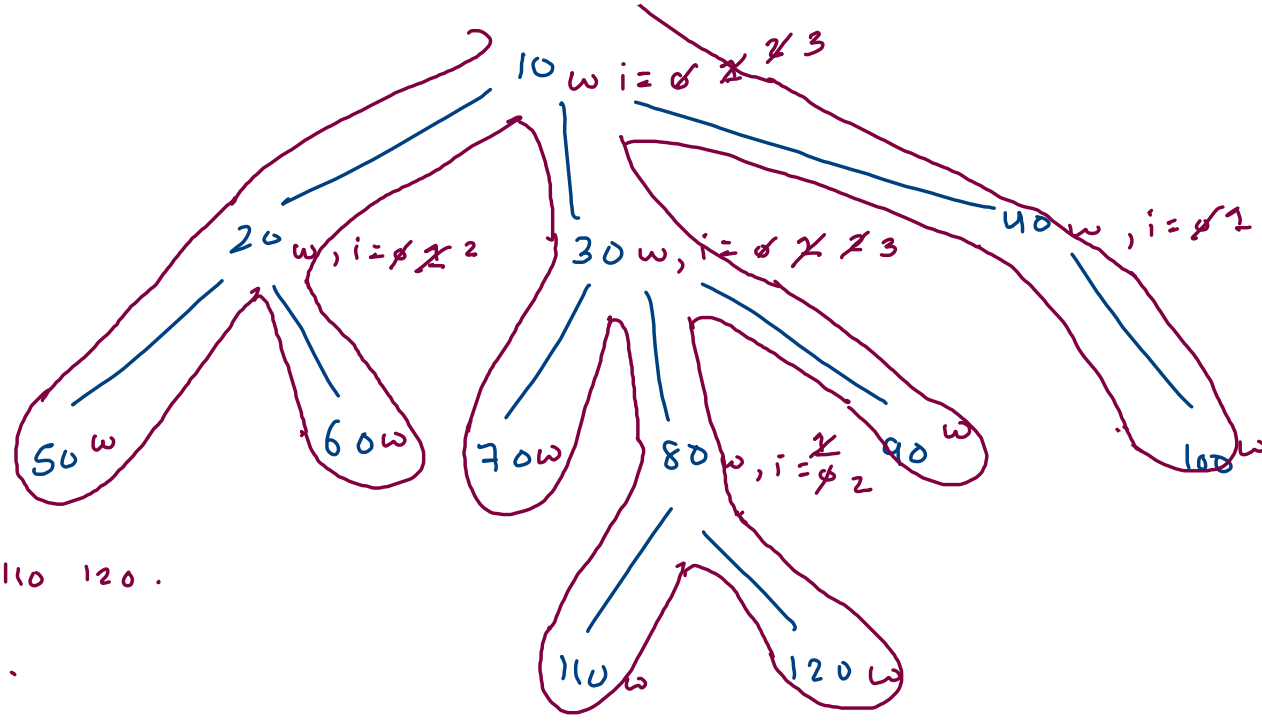
120 → .

90 → .

40 → 100

100 → .

```java
public static void display(Node node) {
    //node & its children
    System.out.print(node.data + " -> ");
    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        System.out.print(child.data + " ");
    }
    System.out.println(".");

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        display(child);
    }
}
```



10 -> 20 30 40.

20 -> 50 60.

50 -> .

60 -> .

30 -> 70 80 90.

70 -> .

80 -> 110 120.

110 -> .

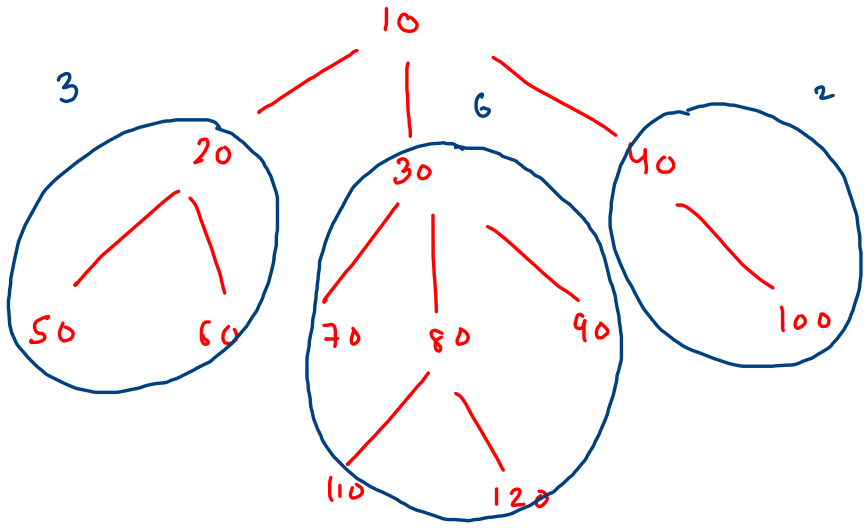
120 -> .

90 -> .

40 -> 100.

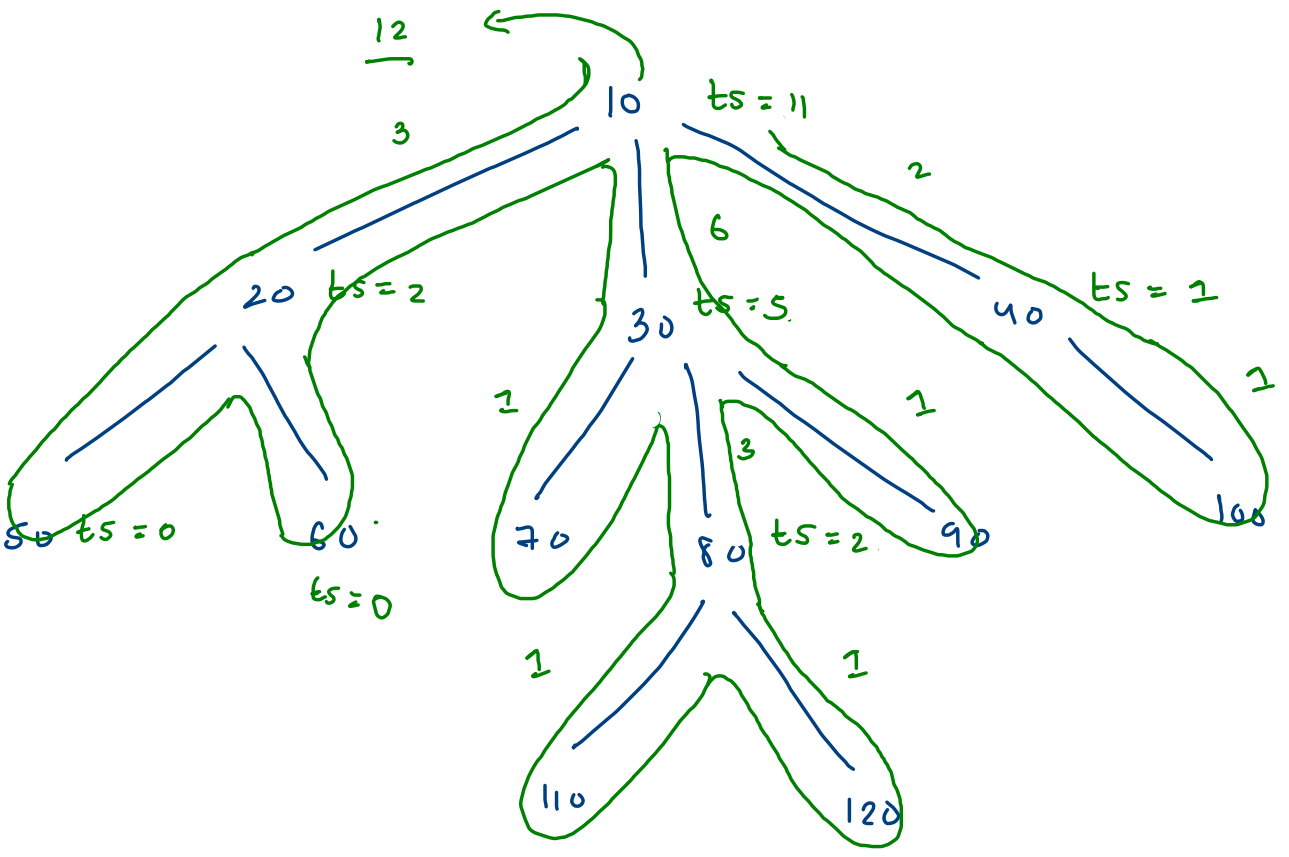
100 -> .

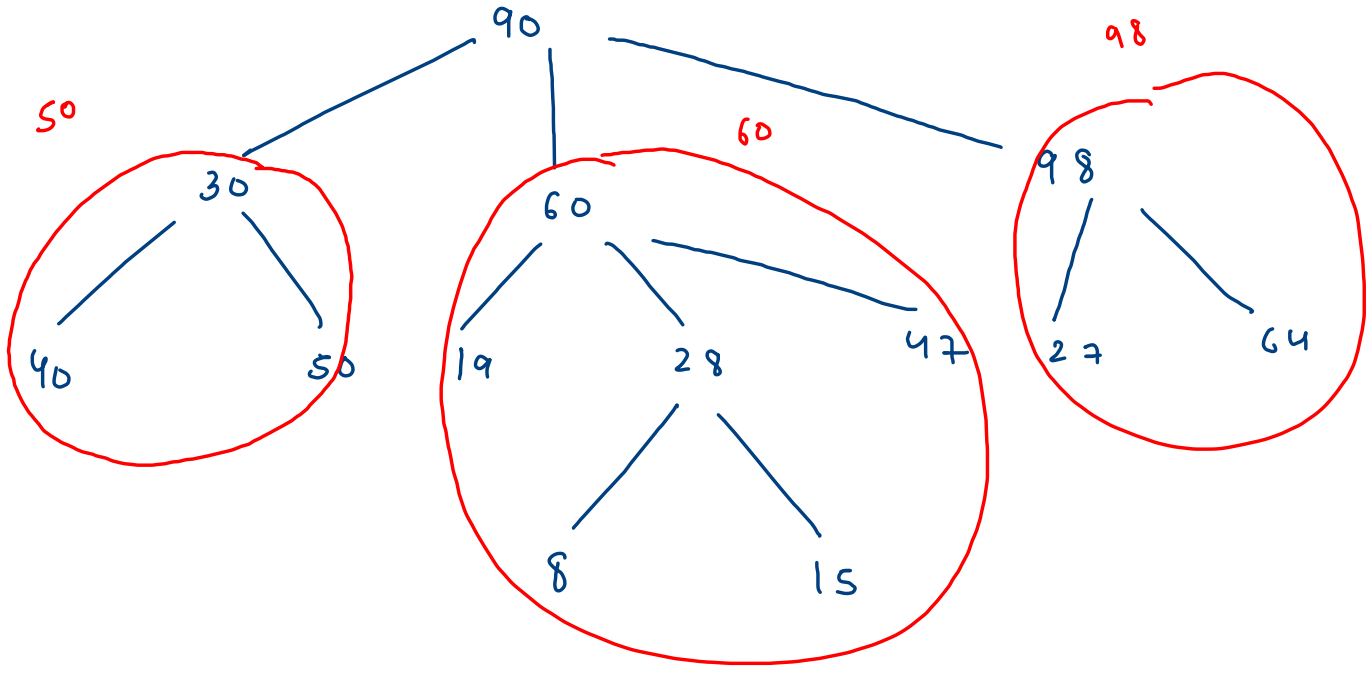
12 = 3 + 6 + 2 + 1



size(10) = size(20) + size(30) + size(40) + 1

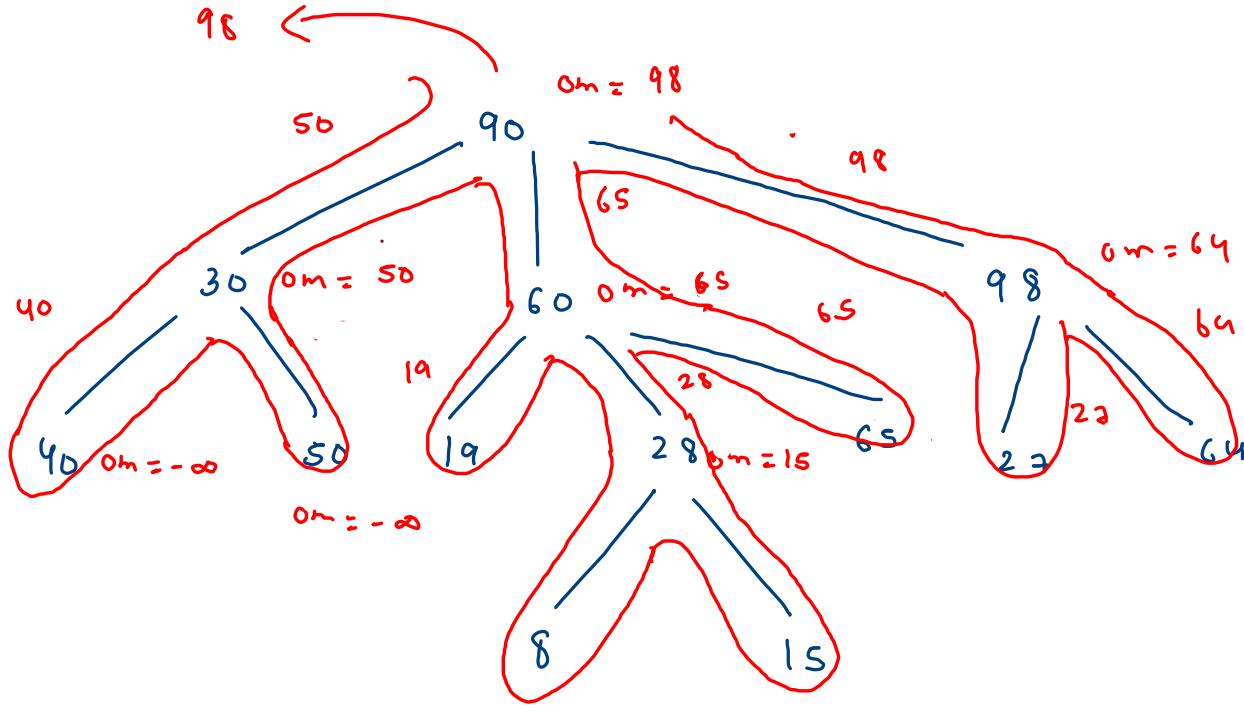
```java
public static int size(Node node){
    int ts = 0;

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        int cfs = size(child); //child family size

        ts += cfs;
    }

    return ts+1;
}
```



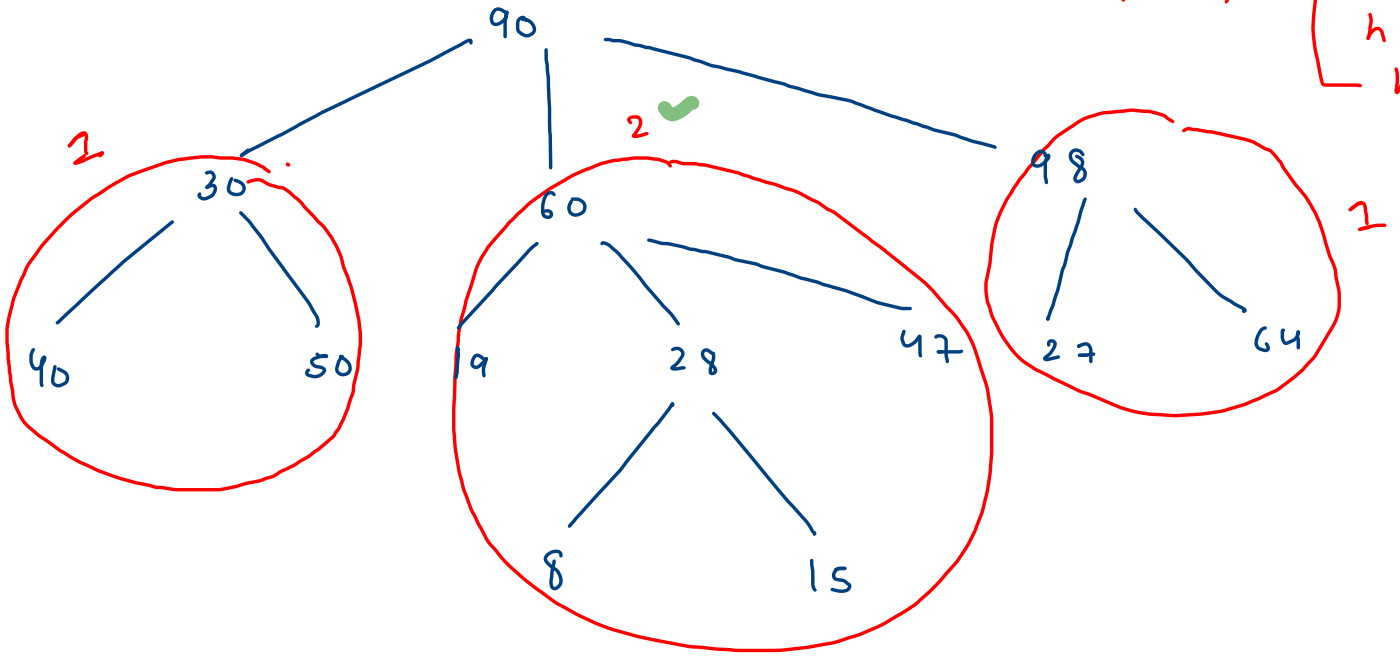


max(90) = Math.max [max(30), max(60), max(98), 90]

```java
public static int max(Node node) {
    int omax = Integer.MIN_VALUE;

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);

        int cfm = max(child); //child family max

        if(cfm > omax) {
            omax = cfm;
        }
    }

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

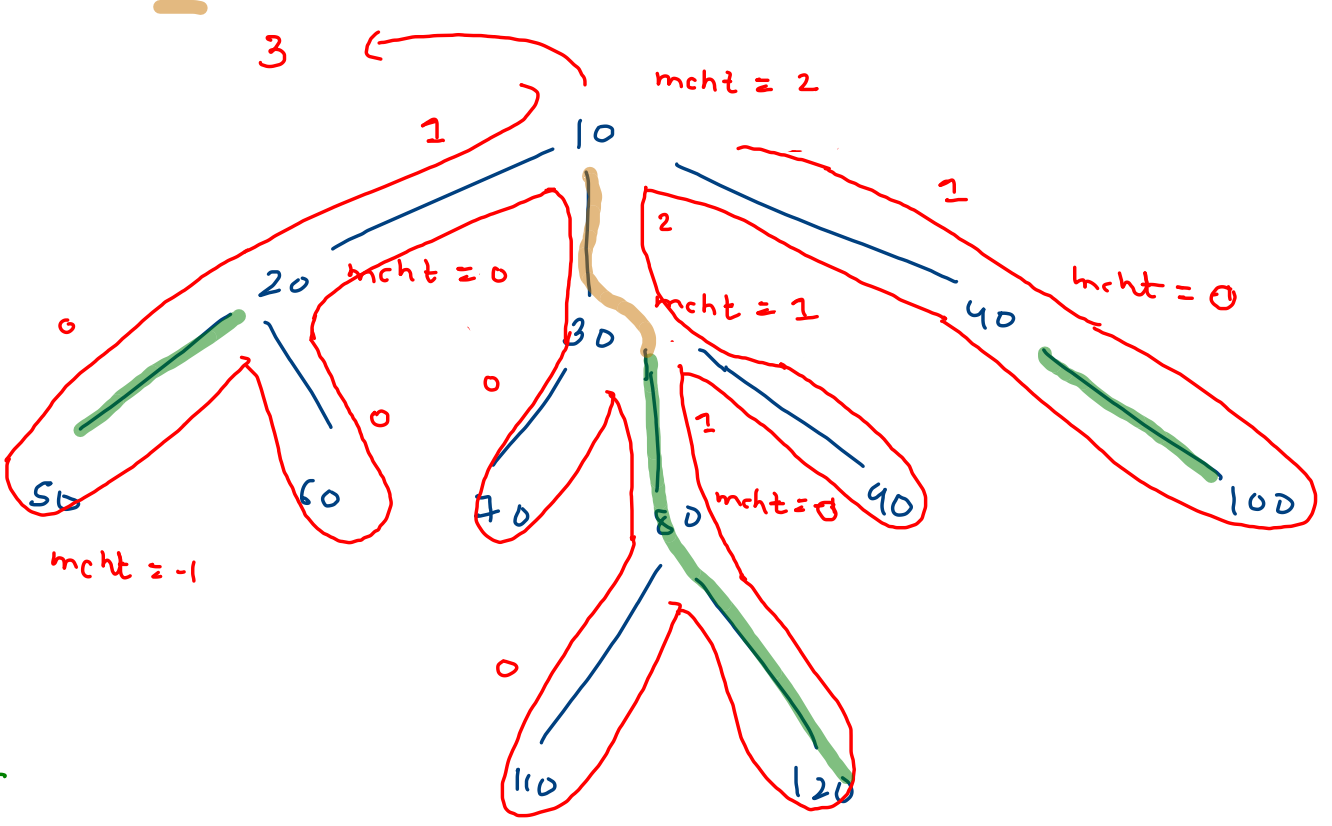


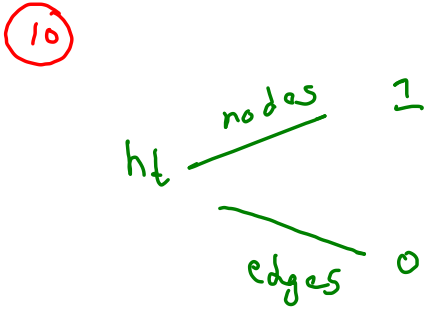
$ht(90) = 1 + \max \begin{cases} ht(30), \\ ht(60), \\ ht(98) \end{cases}$



height : depth of the deepest, distance b/w root & deepest distance

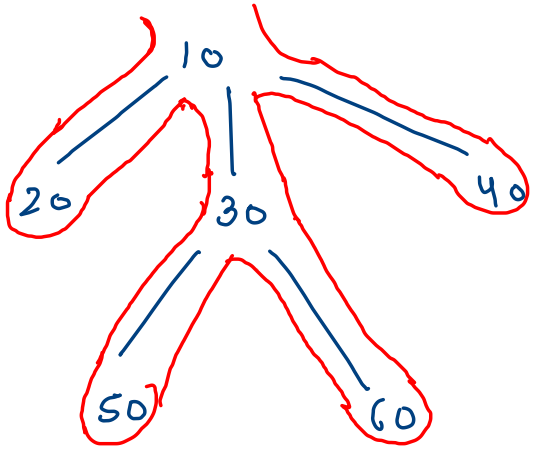
```java
public static int height(Node node) {
  int mcht = -1; //max child height

  for(int i=0; i < node.children.size();i++) {
      Node child = node.children.get(i);

      int htc = height(child); //height of child

      if(htc > mcht) {
          mcht = htc;
      }
  }

  return mcht + 1;

}
```

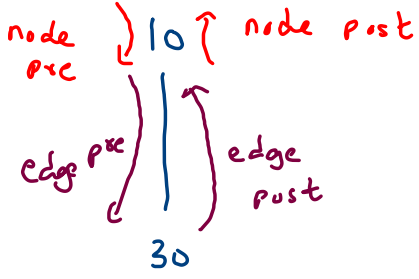




```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

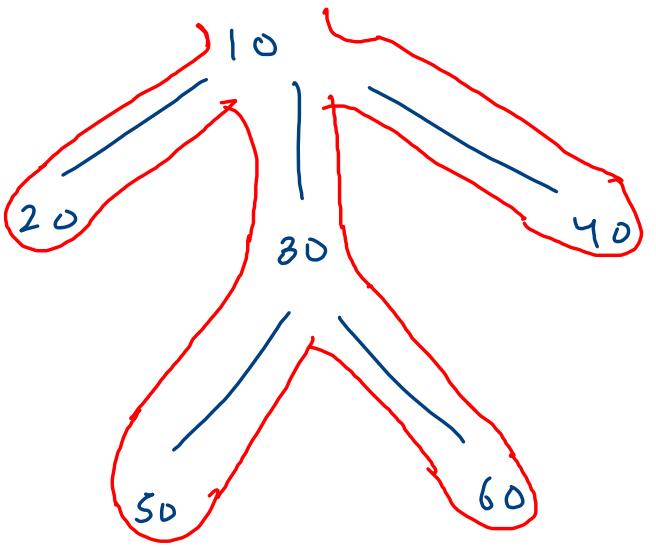
- Node Pre 10
- Edge Pre 10--20
- Node Pre 20
- Node Post 20
- Edge Post 10--20
- Edge Pre 10--30
- Node Pre 30
- Edge Pre 30--50
- Node Pre 50
- Node Post 50
- Edge Post 30--50
- Edge Pre 30--60
- Node Pre 60
- Node Post 60
- Edge Post 30--60
- Node Post 30
- Edge Post 10--30
- Edge Pre 10--40
- Node Pre 40
- Node Post 40
- Edge Post 10--40
- Node Post 10

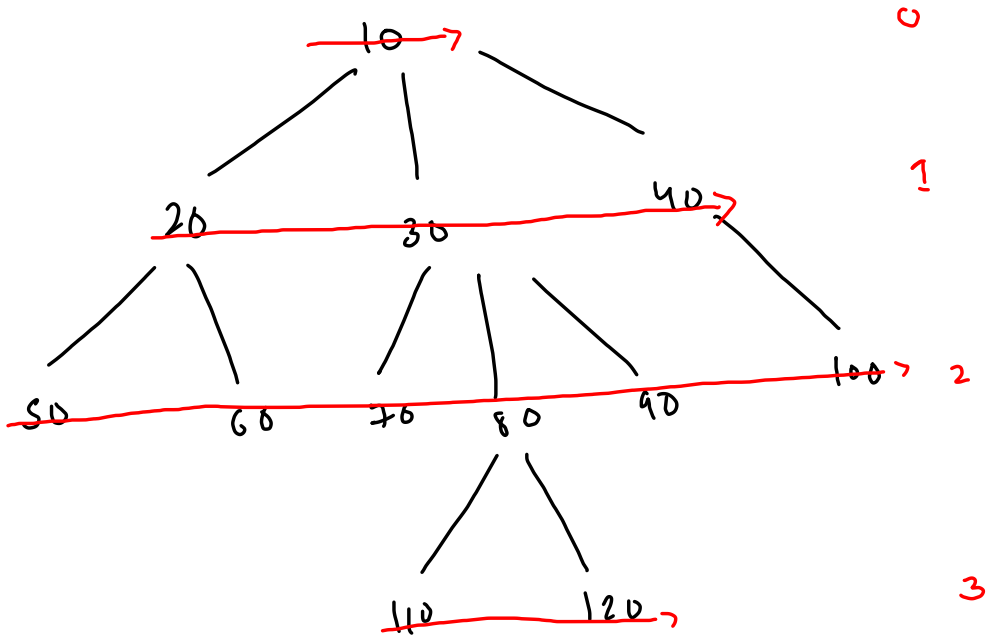




```java
public static void traversals(Node node){
    //node pre
    System.out.println("Node Pre " + node.data);

    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        //edge pre
        System.out.println("Edge Pre " + node.data+"--"+child.data);

        traversals(child); //edge

        //edge post
        System.out.println("Edge Post " + node.data+"--"+child.data)
    }

    //node post
    System.out.println("Node Post " + node.data);
}
```

Node Pre 10
Edge Pre 10--20
Node Pre 20
Node Post 20
Edge Post 10--20
Edge Pre 10--30
Node Pre 30
Edge Pre 30--50
Node Pre 50
Node Post 50
Edge Post 30--50
Edge Pre 30--60
Node Pre 60
Node Post 60
Edge Post 30--60
Node Post 30
Edge Post 10--30
Edge Pre 10--40
Node Pre 40
Node Post 40
Edge Post 10--40
Node Post 10

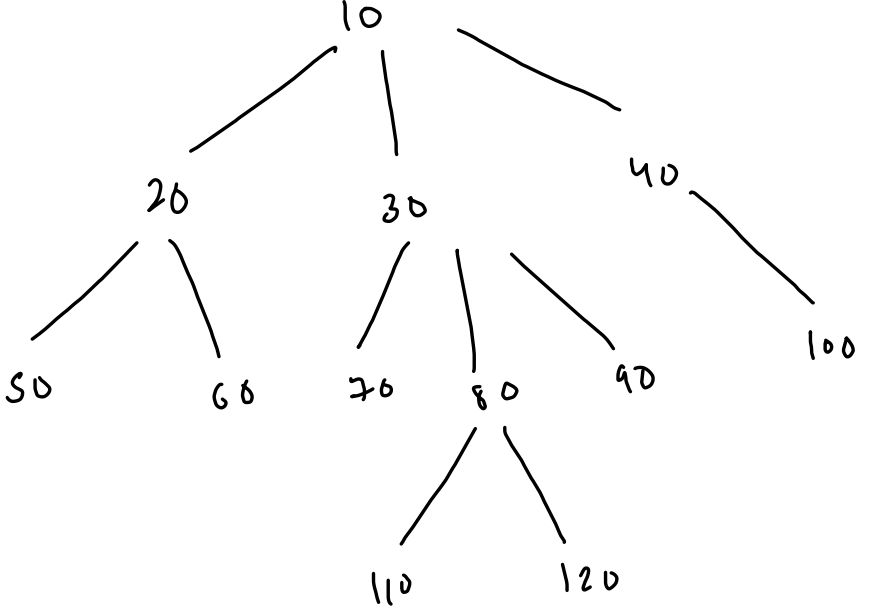




10 20 30 40

50 60 70 80 90 100

110 120



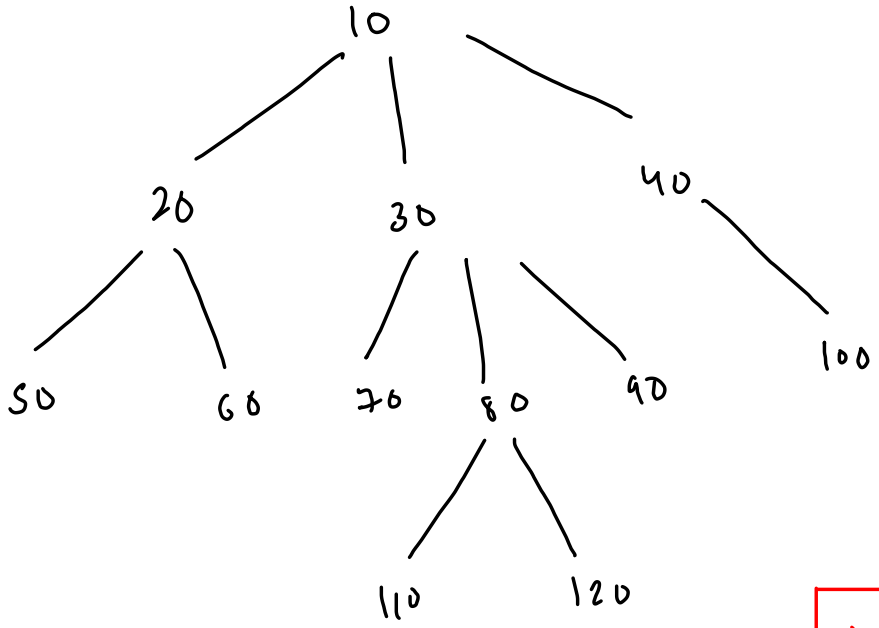

```
while(q.size() > 0) {
    remove();
    print i
    add children;
}
```

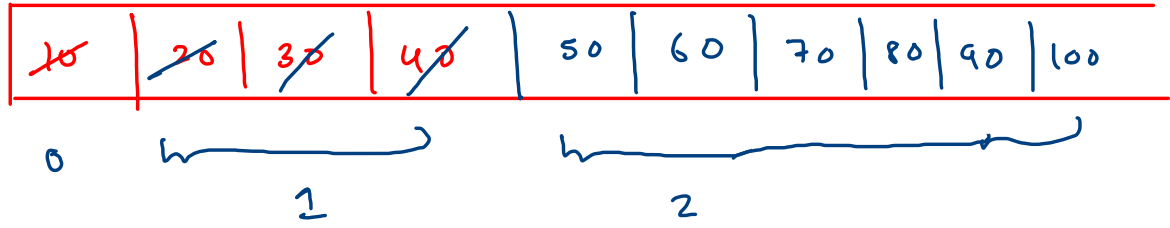
~~10~~	~~20~~	~~30~~	~~40~~	~~50~~	~~60~~	~~70~~	~~80~~	~~90~~	~~100~~	~~110~~	~~120~~

10 20 30 40 50 60

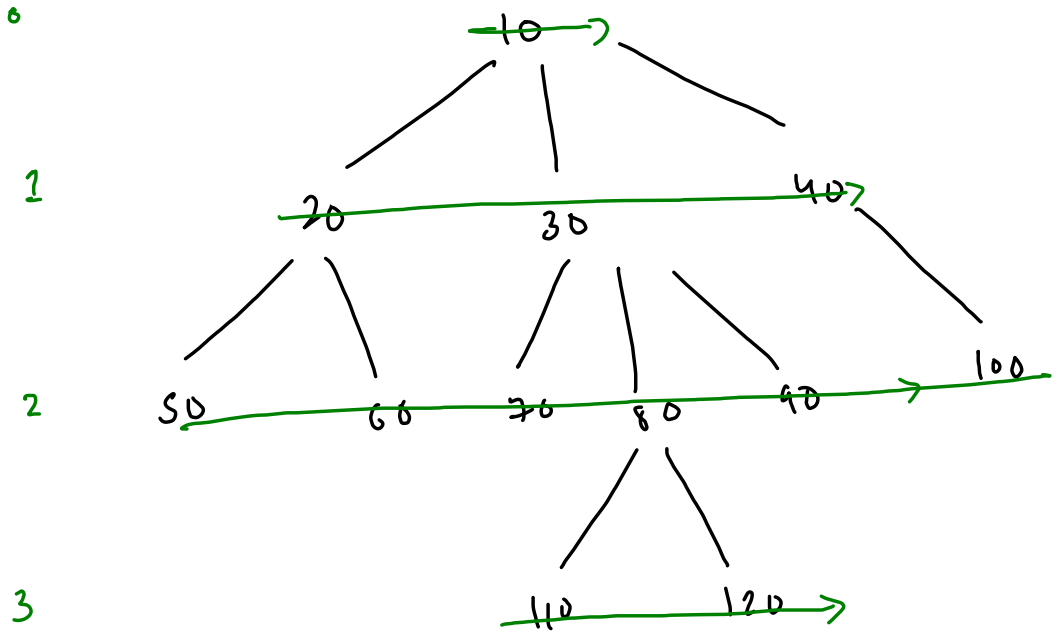
70 80 90 100 110

120


10
20
30
40
50
60
70
80
90
100
110
120


10
20
30
40
50
60
70
80
90
100
0
1
2

```java
public static void levelOrder(Node root){
    Queue<Node>q = new ArrayDeque<>();

    q.add(root);

    while(q.size() > 0) {
        //remove
        Node rem = q.remove();

        //print
        System.out.print(rem.data + " ");

        //add children
        for(int i=0; i < rem.children.size();i++) {
            Node child = rem.children.get(i);
            q.add(child);
        }
    }

    System.out.println(".");
}
```

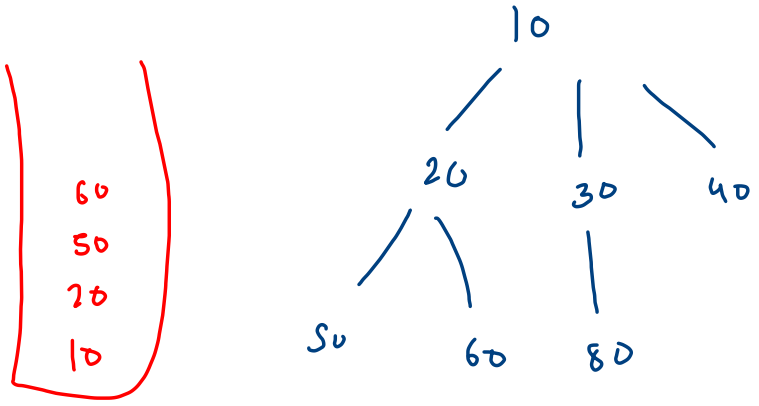


10 20 30 40 50 60 70 80 90 100

110 120.

Depth

(i) pre, post



(ii) child > sibling

Breadth

(i) level order

(ii) sibling > child