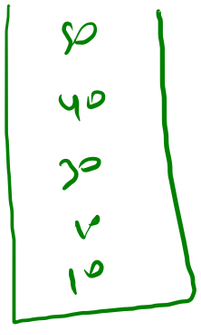
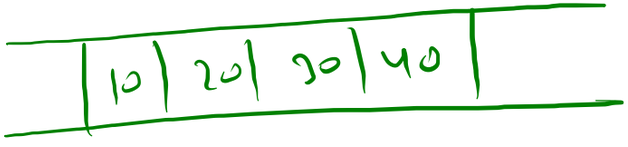
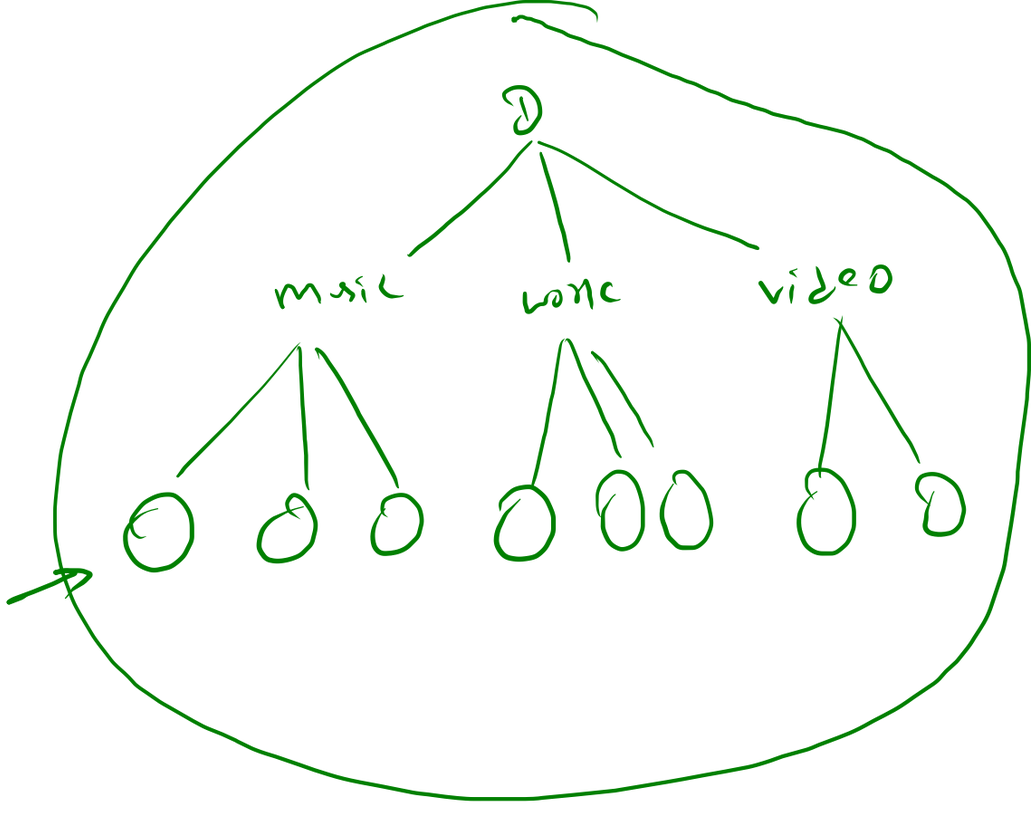
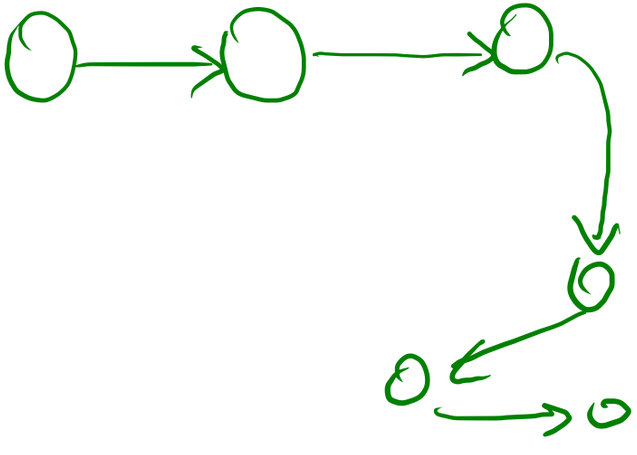
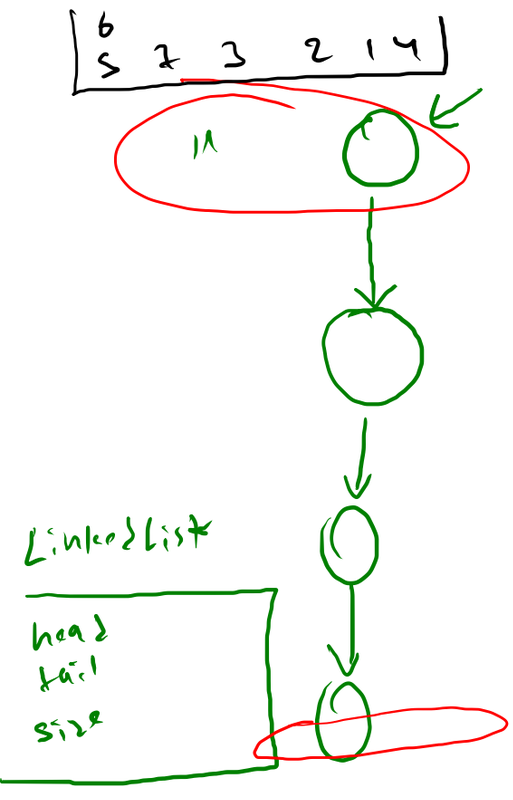






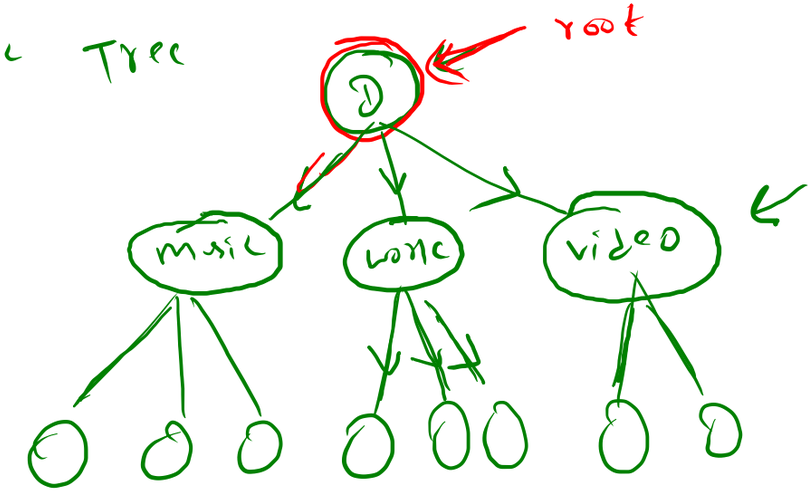
Tree






```
class Node {
    int data
    Node next;
}
```

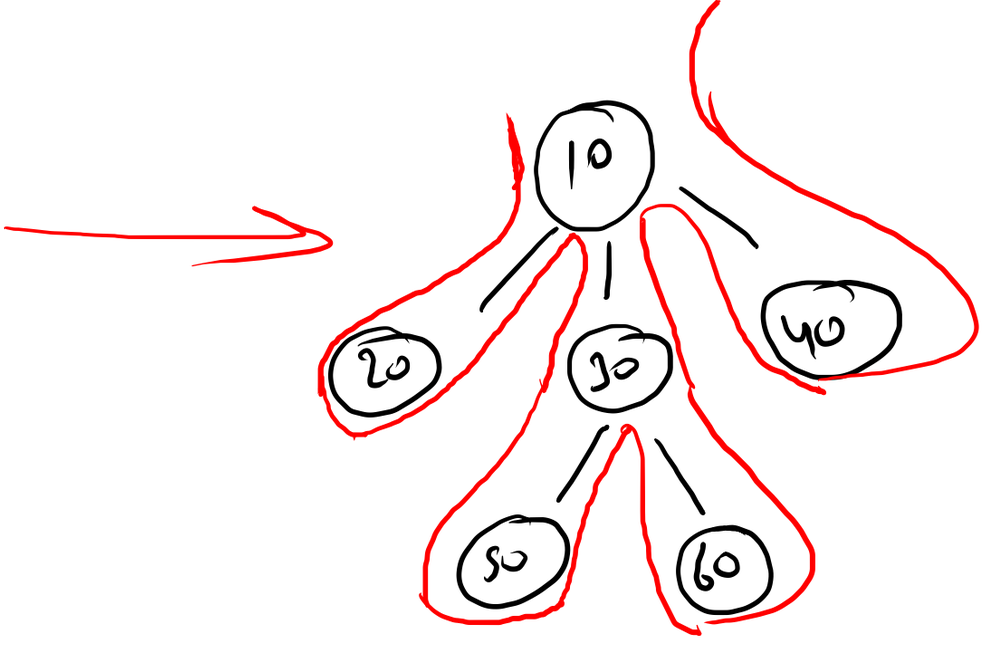
Generic Tree



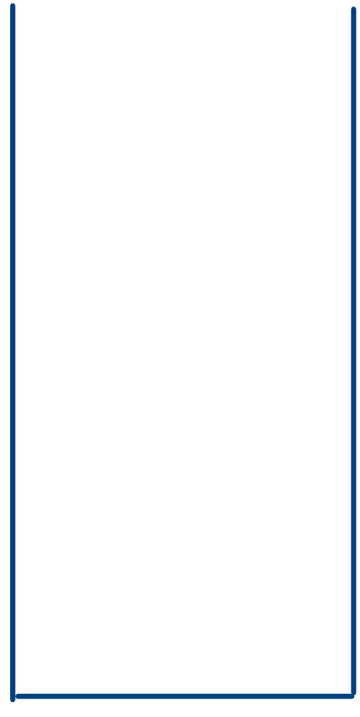

```
Node {
    int data;

    ArrayList<Node> child;
}
```

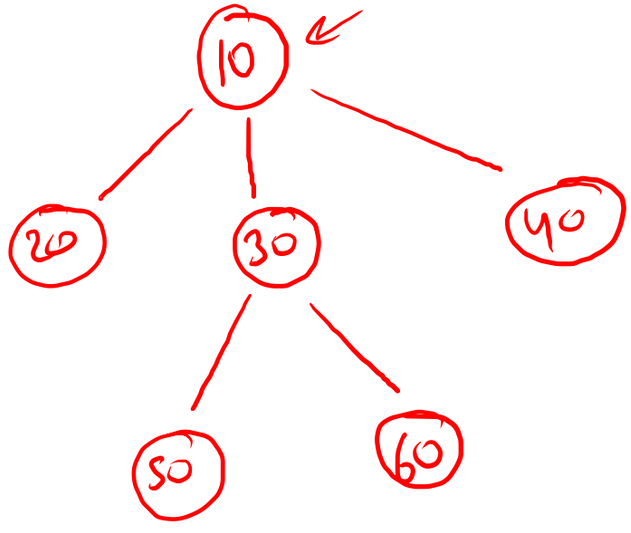
GT

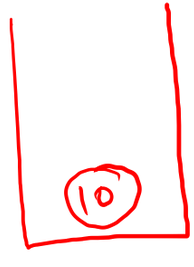
10 20 -1 30 50 -1
60 -1 -1 40 -1 -1



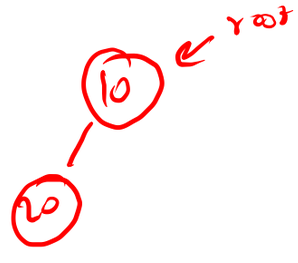
10 20 -1 30 50 -1
60 -1 -1 40 -1 -1







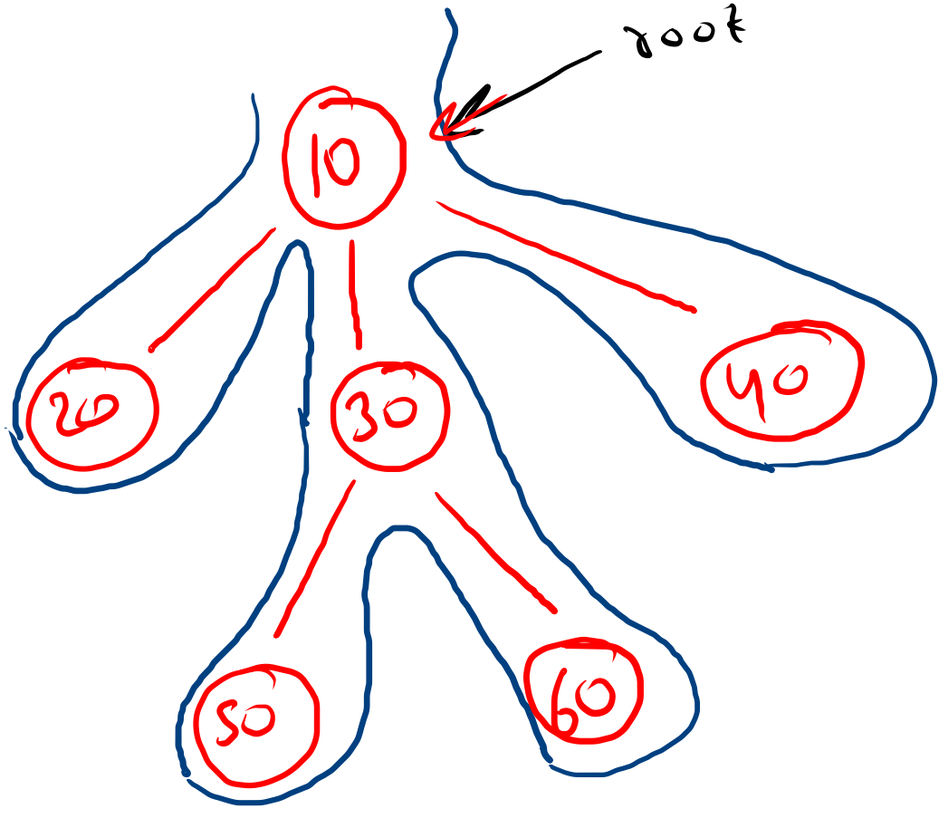
data



how will

Link with TOS

push to stack



[10 → 20, 30, 40.

[20 → .

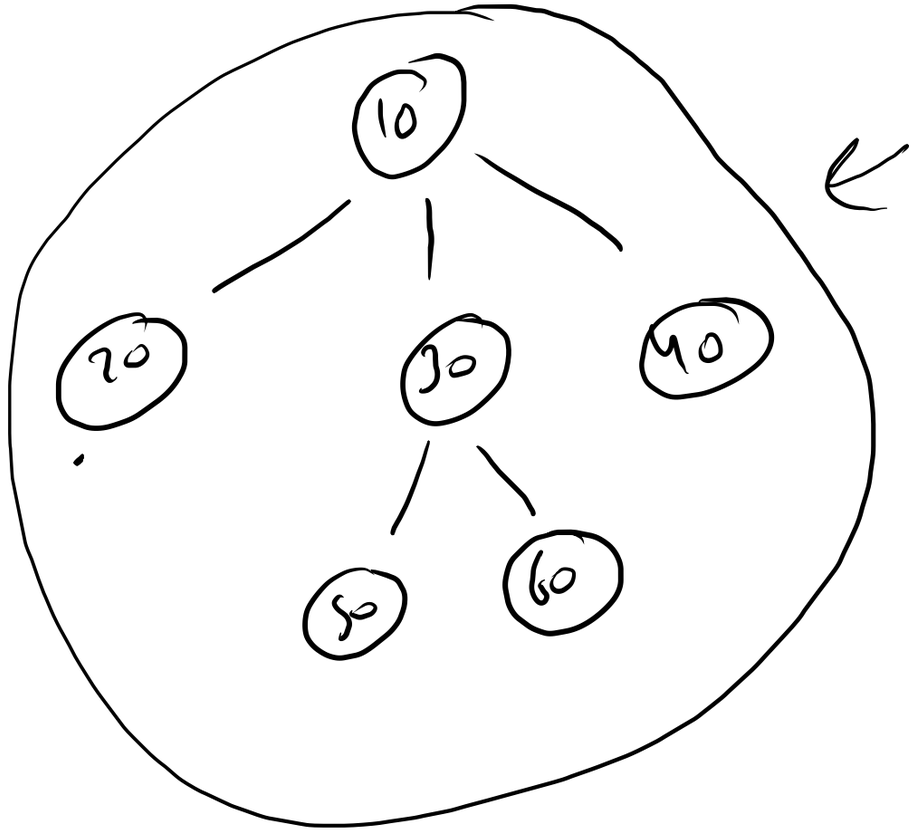
[30 → 50, 60.

50 → .

60 → .

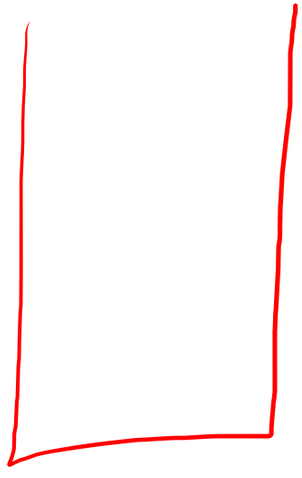
[40 → .

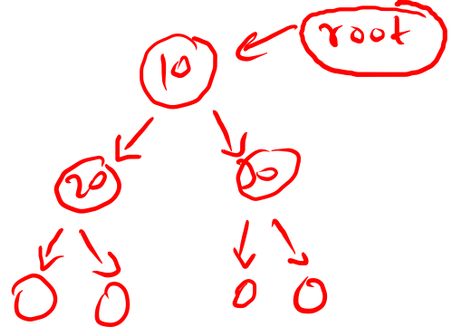
10 -> 20 30 40 .
20 -> .
30 -> 50 60 .
50 -> .
60 -> .
40 -> .

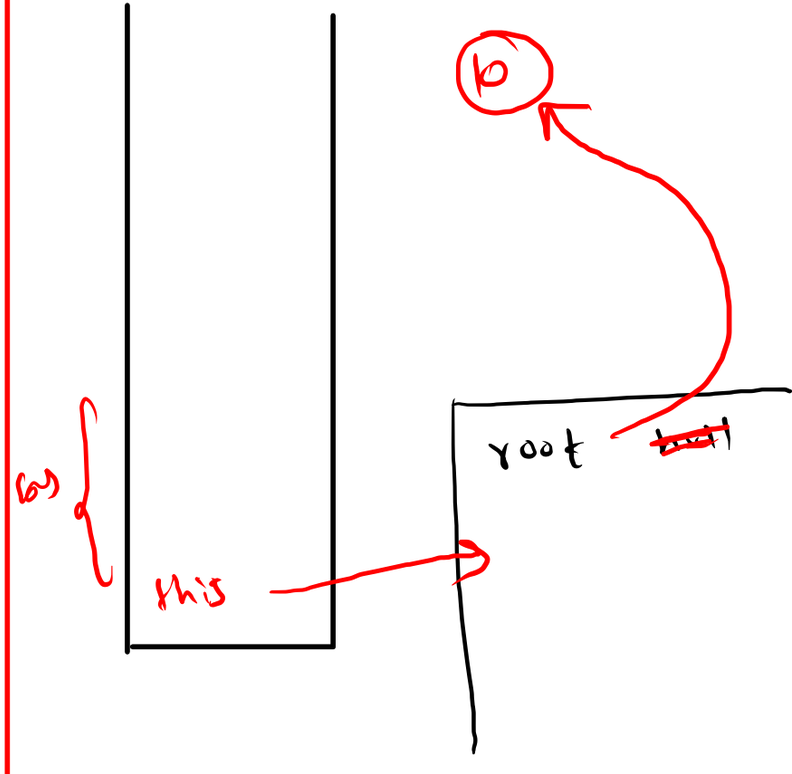


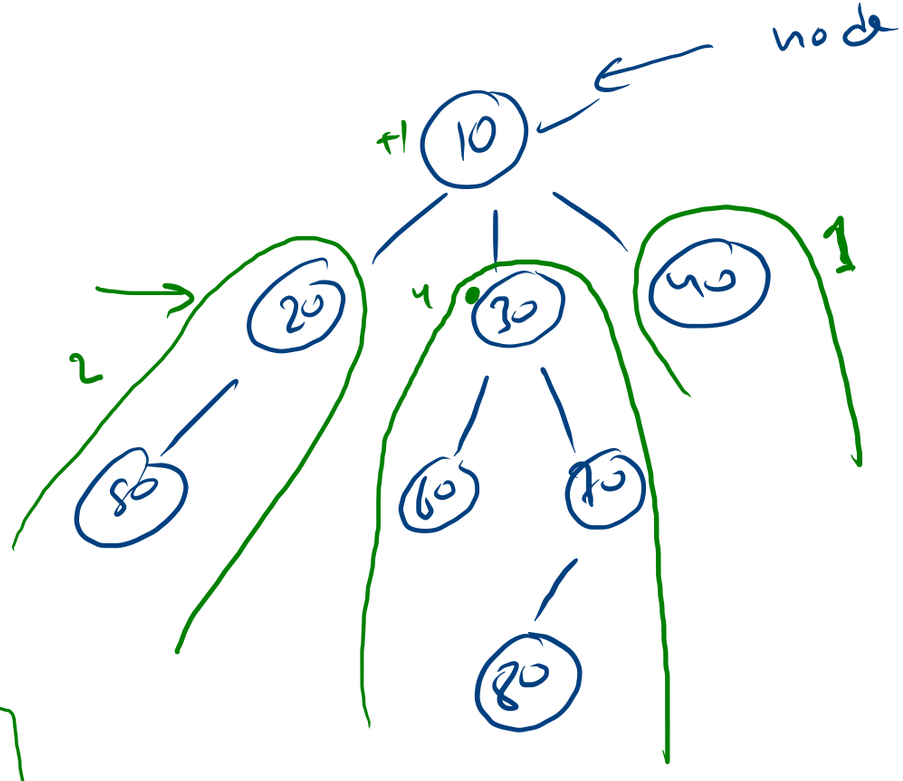
int [10 20 -1 30 ...

```
Node root = null;

public void construct(int arr[]){
    Stack<Node> st = new Stack<>();

    for(int val : arr){

        if(val == -1){
            st.pop();
        }else{
            Node n = new Node(val);
            if(st.size() == 0){
                root = n;
            }else{
                Node parent = st.peek();
                parent.childs.add(n);
            }
            st.push(n);
        }
    }
}
```









size = 8

[20, 30, 40]

size

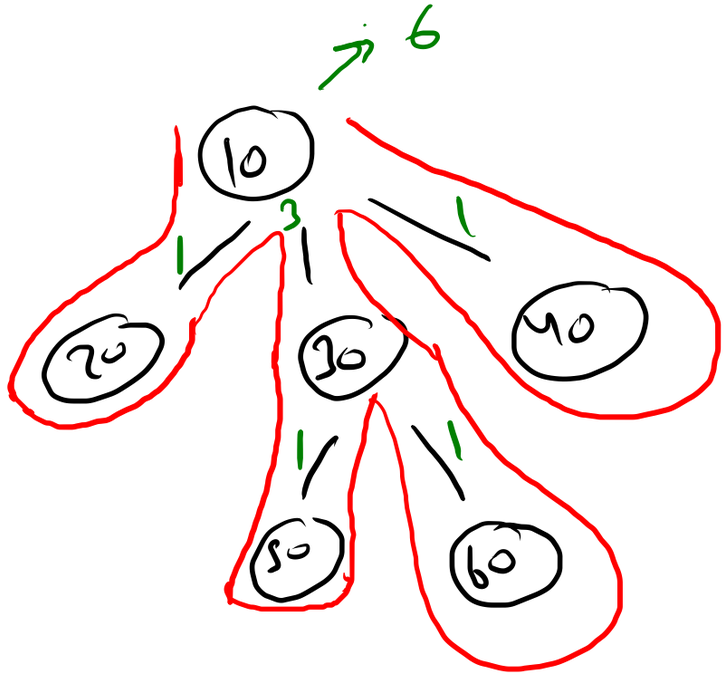
int sum = 0

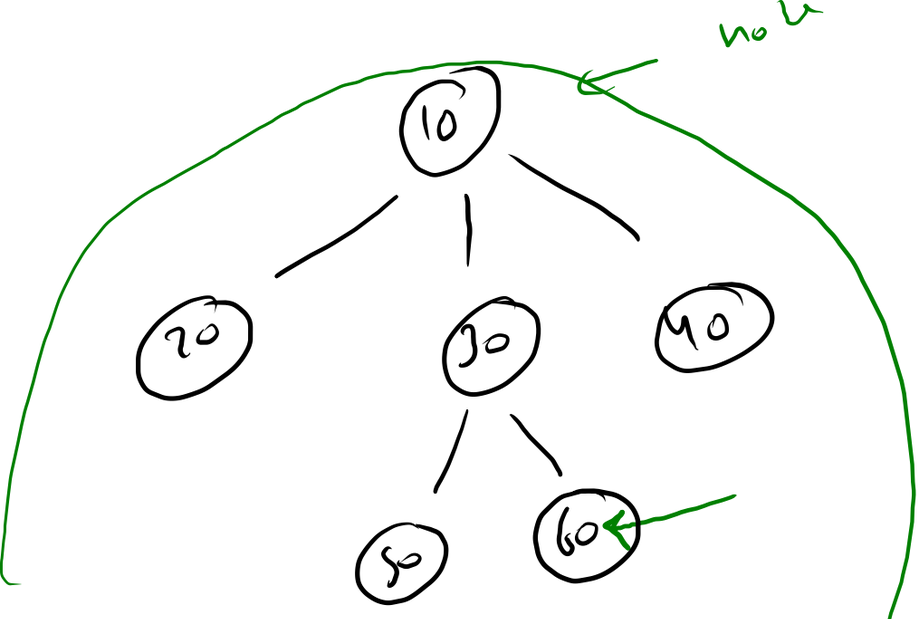
call childs

sum += size(child)

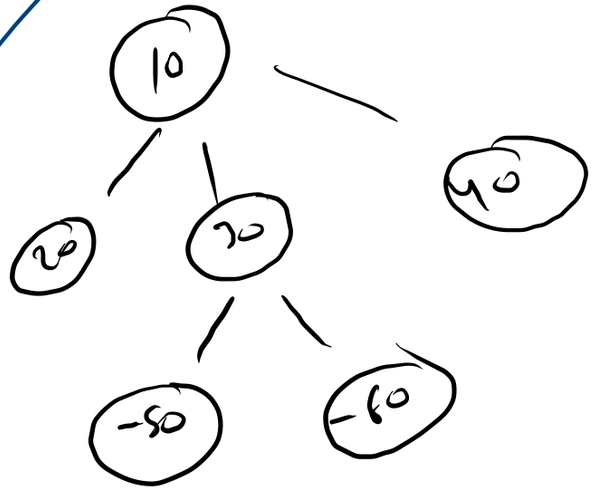
sum ++;

```java
public static int size(Node node){

    int sum=1;

    for(Node child: node.children){
        sum += size(child);
    }
    // sum++;

    return sum;
}
```

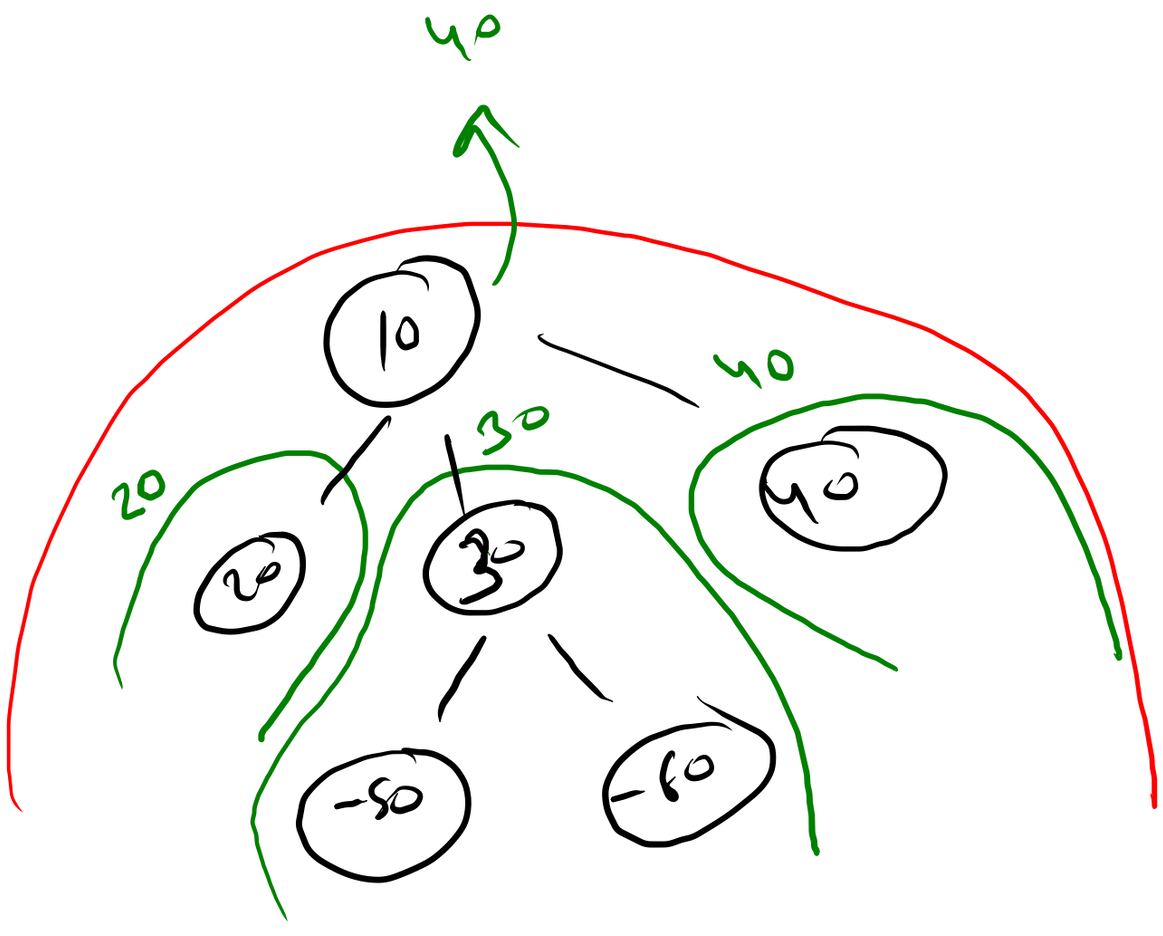




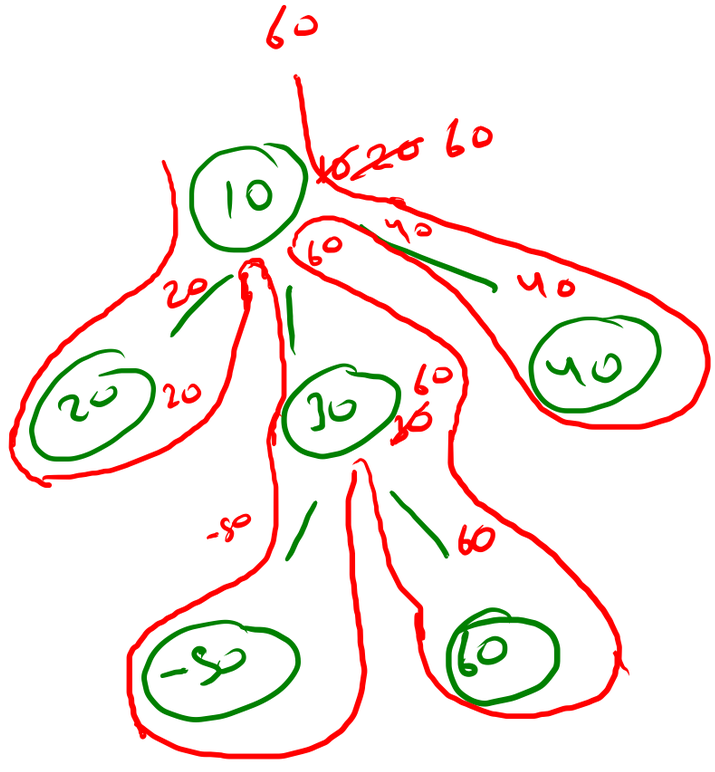
60 max



40 max


40
10
40
20
30
20
30
40
-50
-10

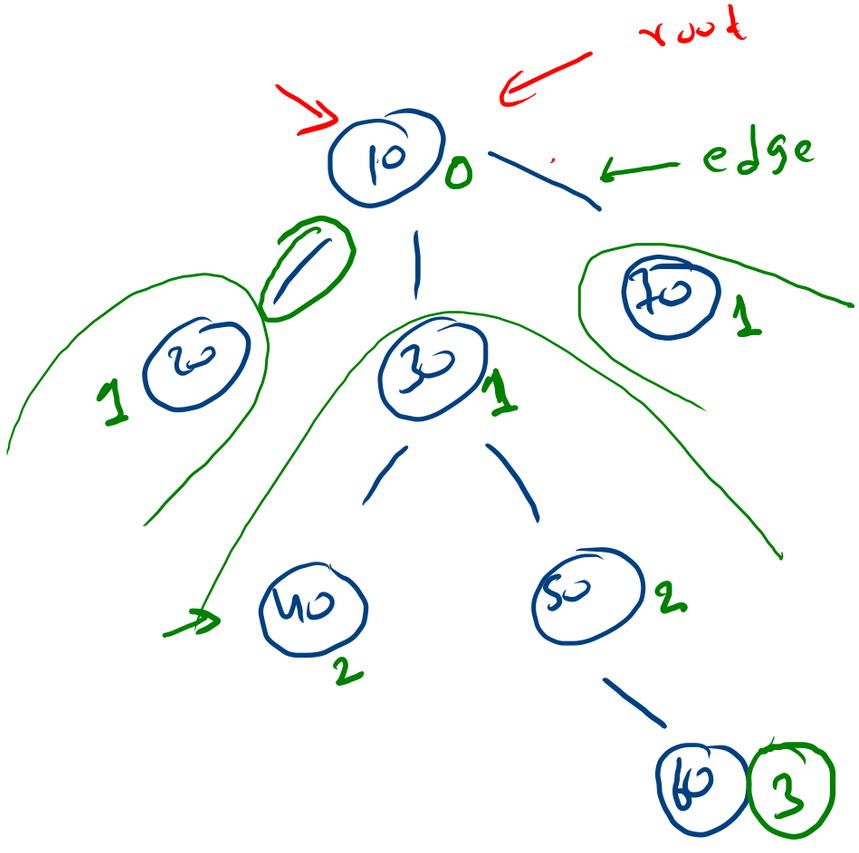
```java
public static int max(Node node) {

    int max = node.data;

    for(Node child : node.children){
        int fmax = max(child);
        max = Math.max(max, fmax);
    }
    return max;
}
```



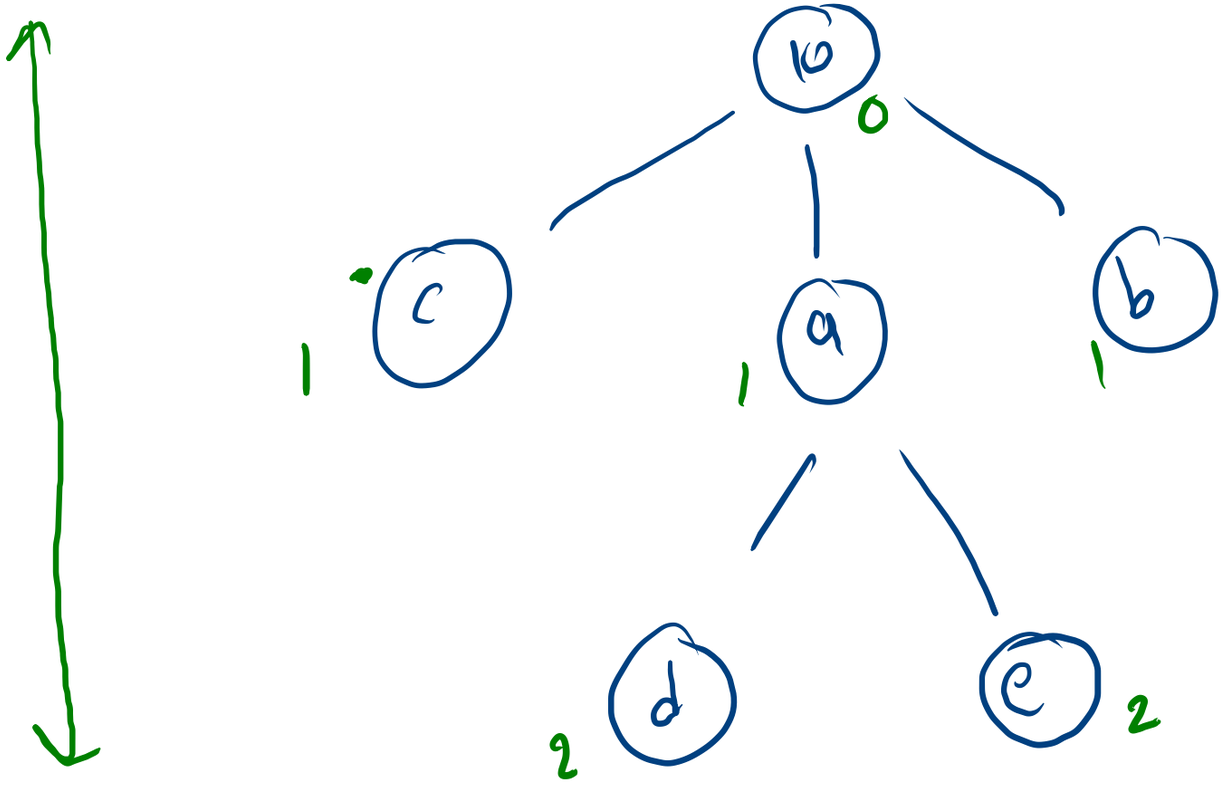
Tree

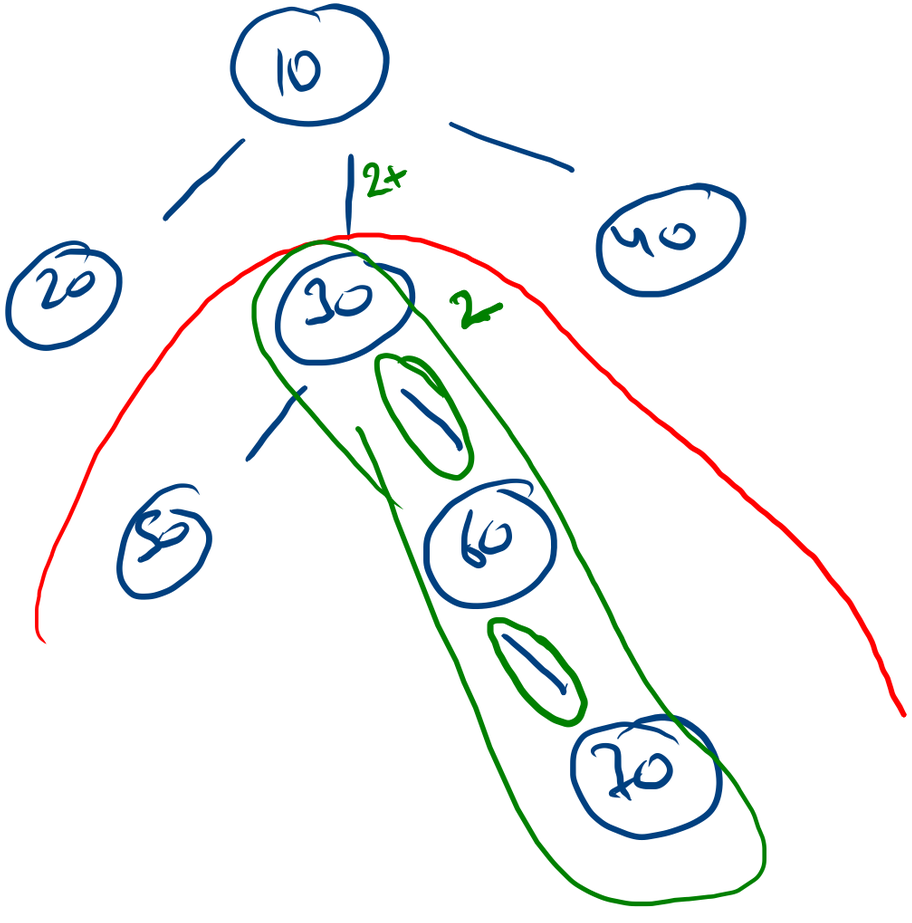
height → max depth

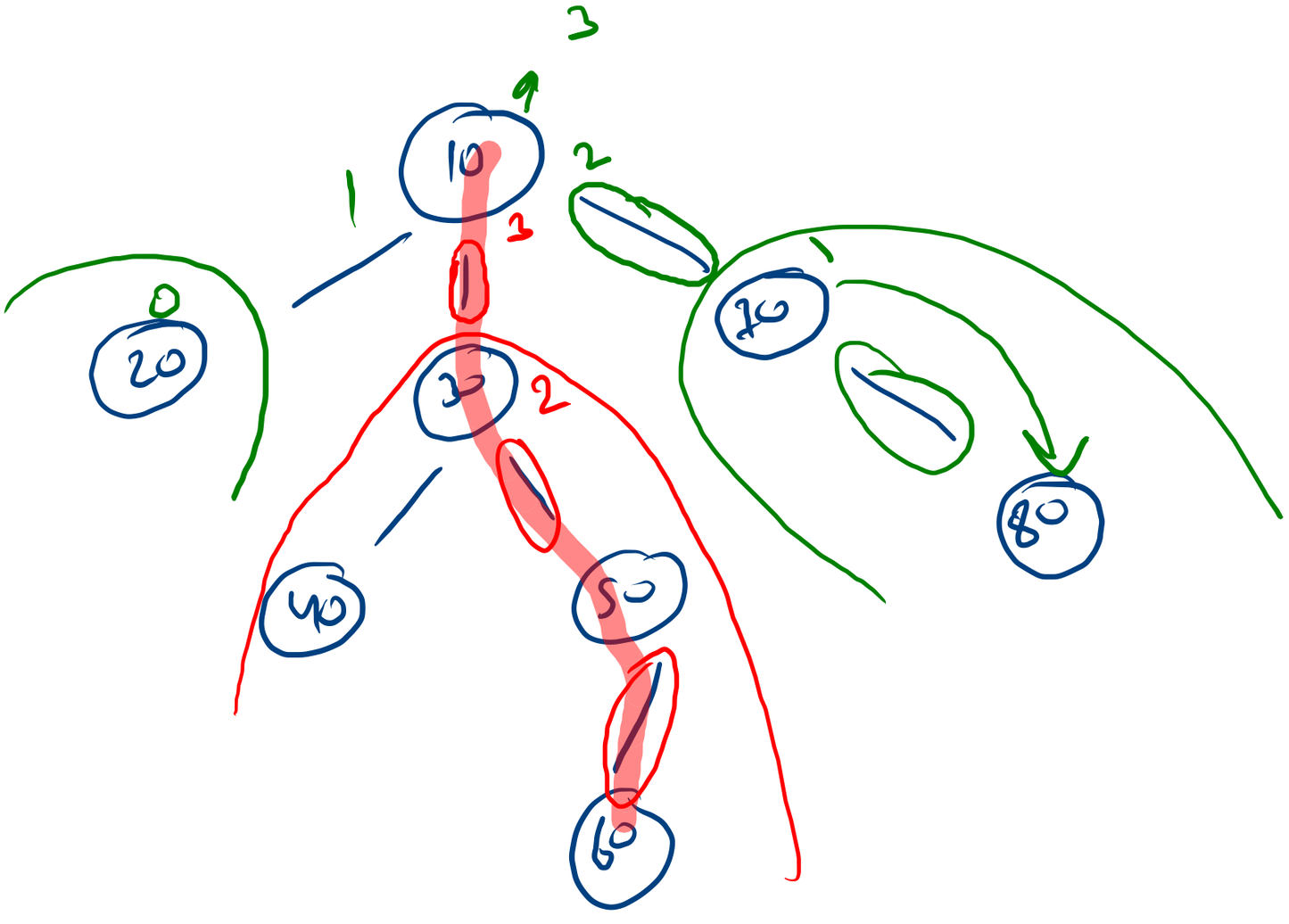
3



height → 2




10
20
30
40
50
60
70
2+
2
1
1


3
10
2
1
1
0
20
30
2
70
1
40
50
80
60

```java
public static int height(Node node) {

    int h = 0;
    for (Node child : node.children){
        int ch = height(child);
        h = Math.max(h, ch+1);
    }
    return h;
}
```

Steps

Dry

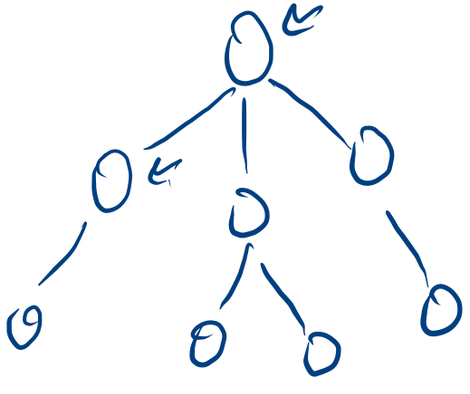
Recursion

Relation

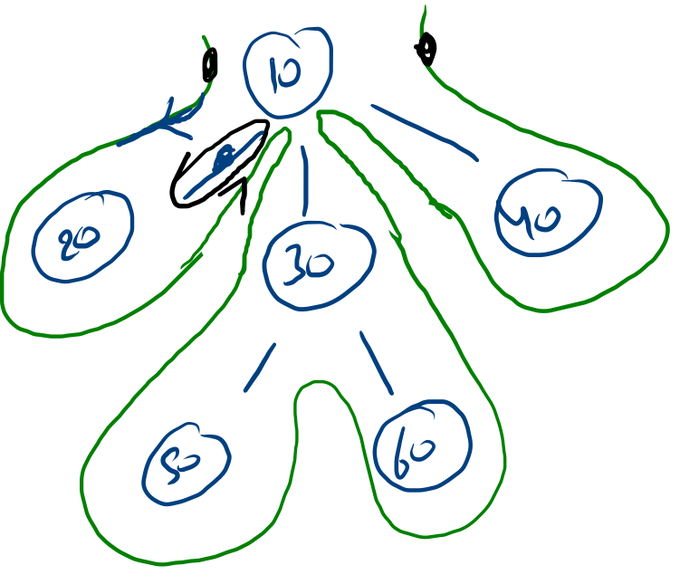
formula

complexity

1 node → k



n → nk → O(n)



Node pre

edge pre

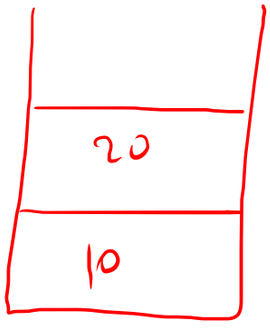
Node post

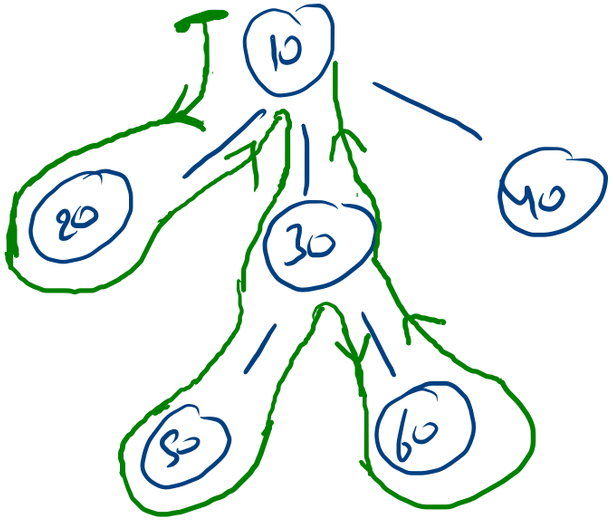
edge post

pre → left

post → right

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





Node pre 10
edge pre 10-20
Node pre 20
Node post 20
edge post 10-20
e pre 10-30
N pre 30
e pre 30-50
N pre 50
N post 50
e post 30-50

e pre 30-60
N pre 60
N post 60
e post 30-60
N post 30
e post 10-30

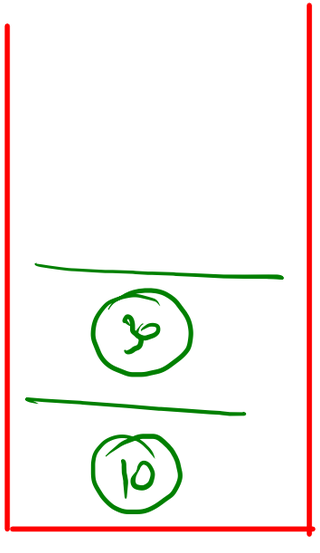
```
public static void traversals(Node node){
    Node pre
    for (Node child : node.children){
        pre
        traversals(child);
        post
    }
    Node post
}
```

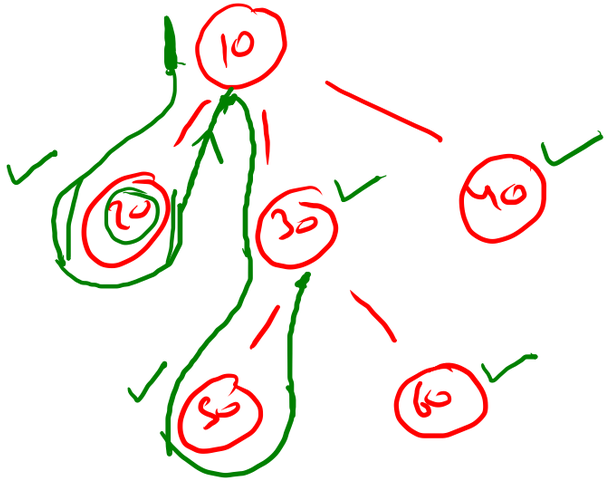
loop

Node pre node.data
post

Edge pre d1--d2
post

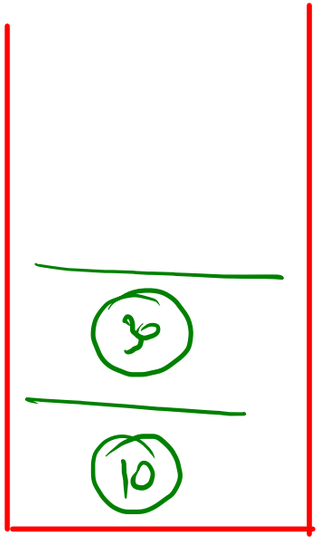
```java
public static void traversals(Node node){
    System.out.println("Node Pre "+node.data);
    for (Node child : node.children){
        System.out.println("Edge Pre "+node.data+"--"+child.data);

        traversals(child);

        System.out.println("Edge Post "+node.data+"--"+child.data);
    }
    System.out.println("Node Post "+node.data);
}
```

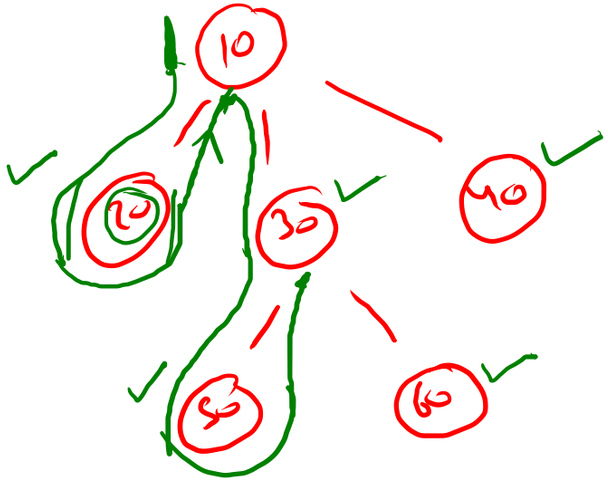




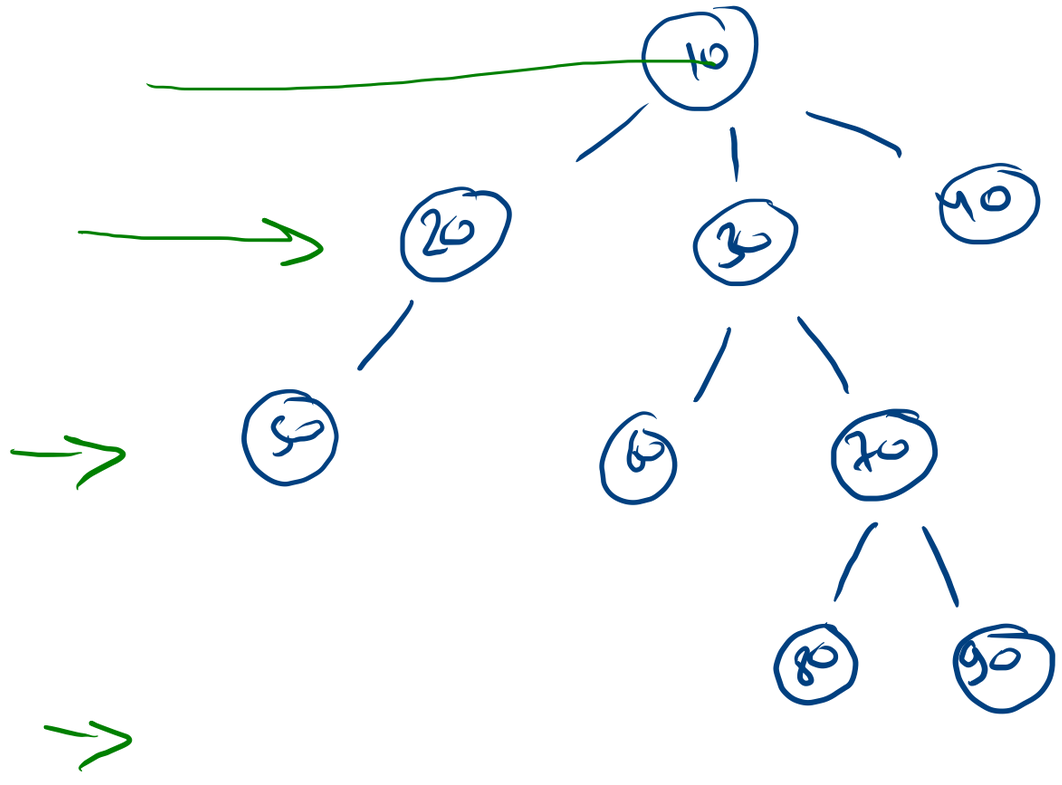
Node pre 10
Edge pre 10 -- 20
N pre 20
N post 20
E post 10 -- 20
E pre 10 -- 30
N pre 30
E pre 30 -- 50
N pre 50
N post 50
E post 30 -- 50

```java
public static void traversals(Node node){
    System.out.println("Node Pre "+node.data);
    for (Node child : node.children){
        System.out.println("Edge Pre "+node.data+"--"+child.data);

        traversals(child);

        System.out.println("Edge Post "+node.data+"--"+child.data);
    }
    System.out.println("Node Post "+node.data);
}
```



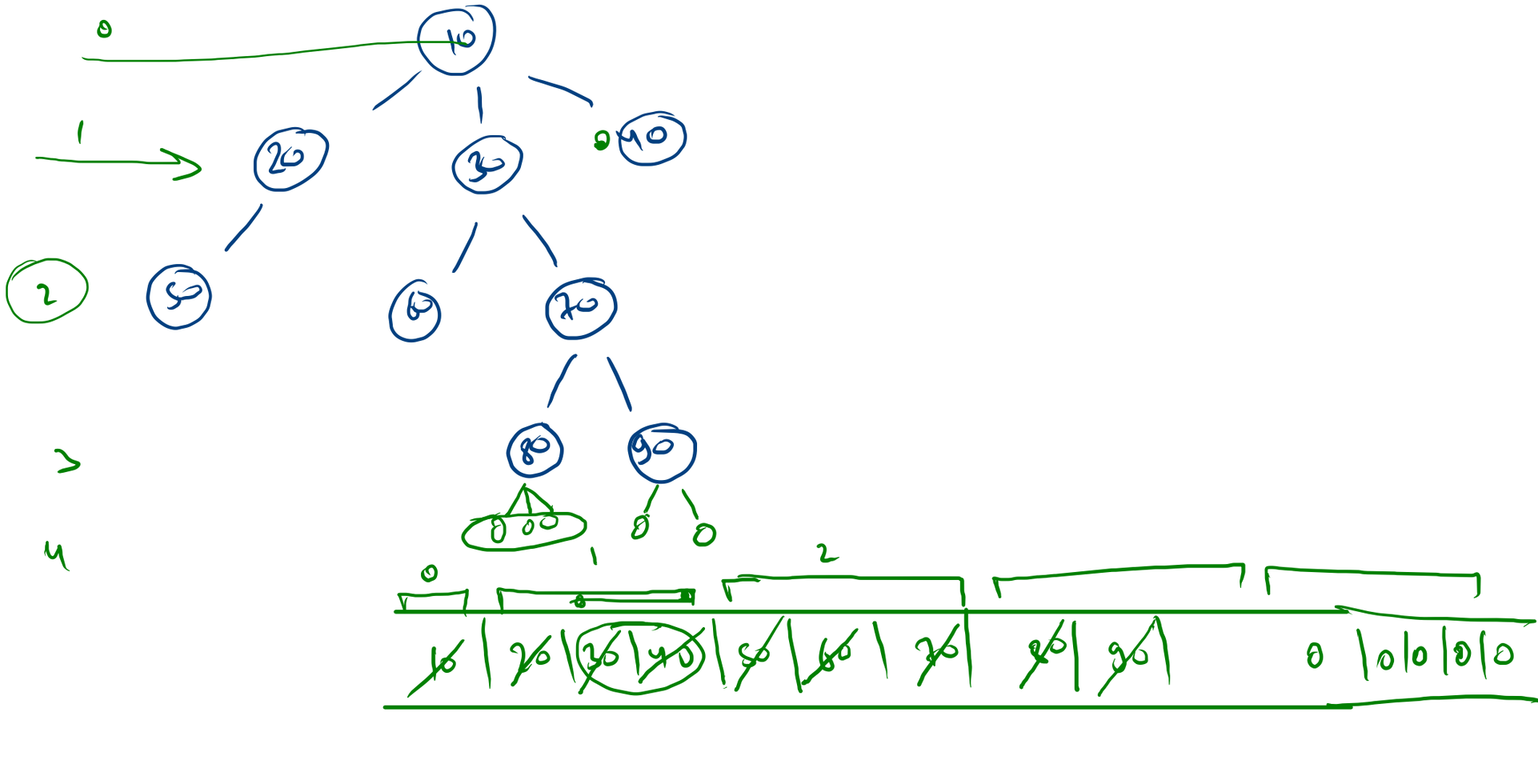


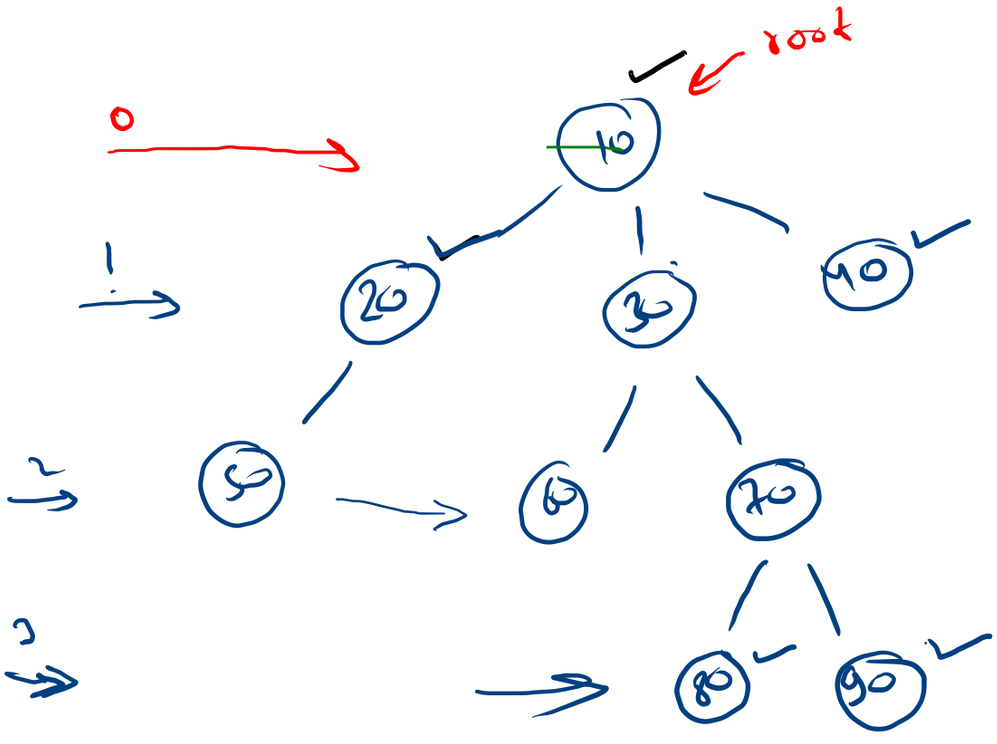
Node pre 10
Edge pre 10 -- 20
N pre 20
N post 20
E post 10 -- 20
E pre 10 -- 30
N pre 30
E pre 30 -- 50
N pre 50
N post 50
E post 30 -- 50



~~10~~	~~20~~	30	40	50

10 20 30 40 50 60 70 80 90.


10
20
30
40
50
60
0
1
2
3
4
0
1
2

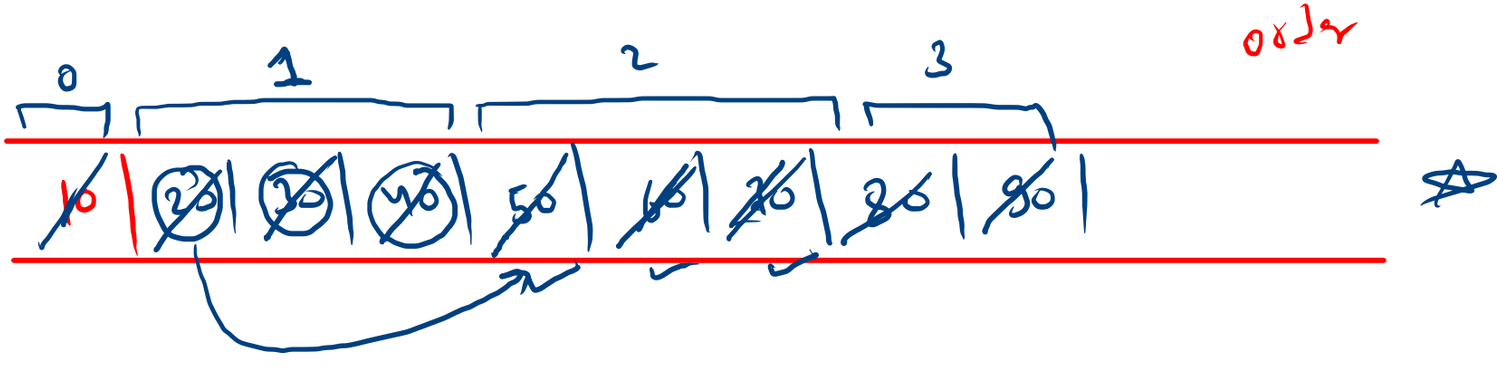



```
q.siz != 0
    n = q.remove
    prin n
    adz all chilg
```

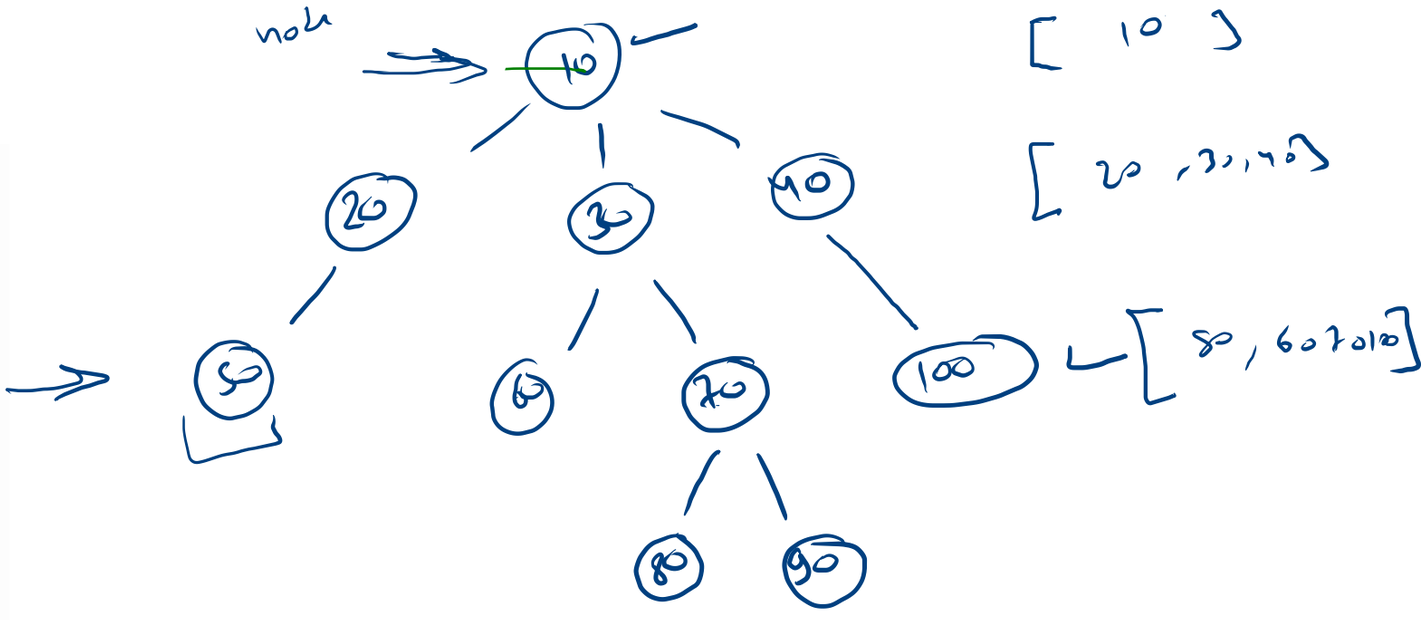
10 20 30 40

50 60 70 80 90

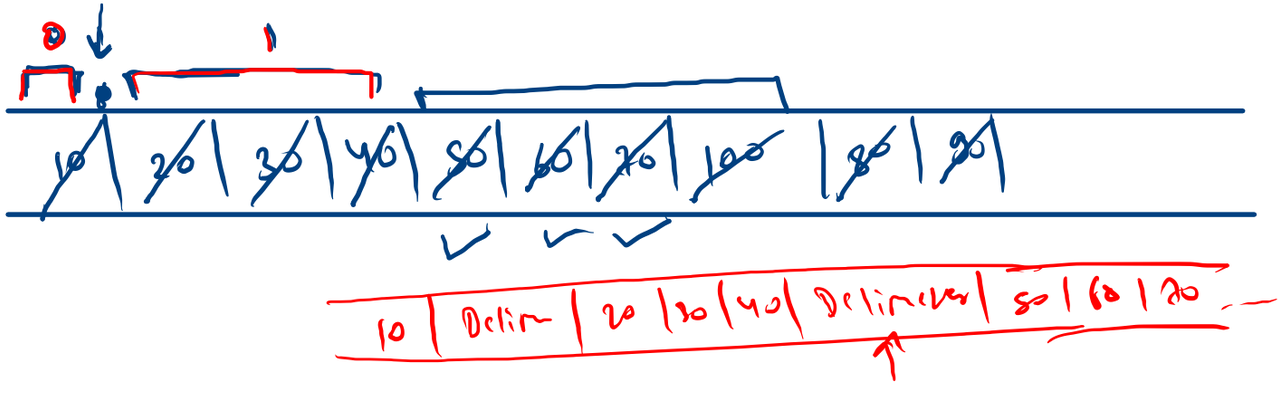
order

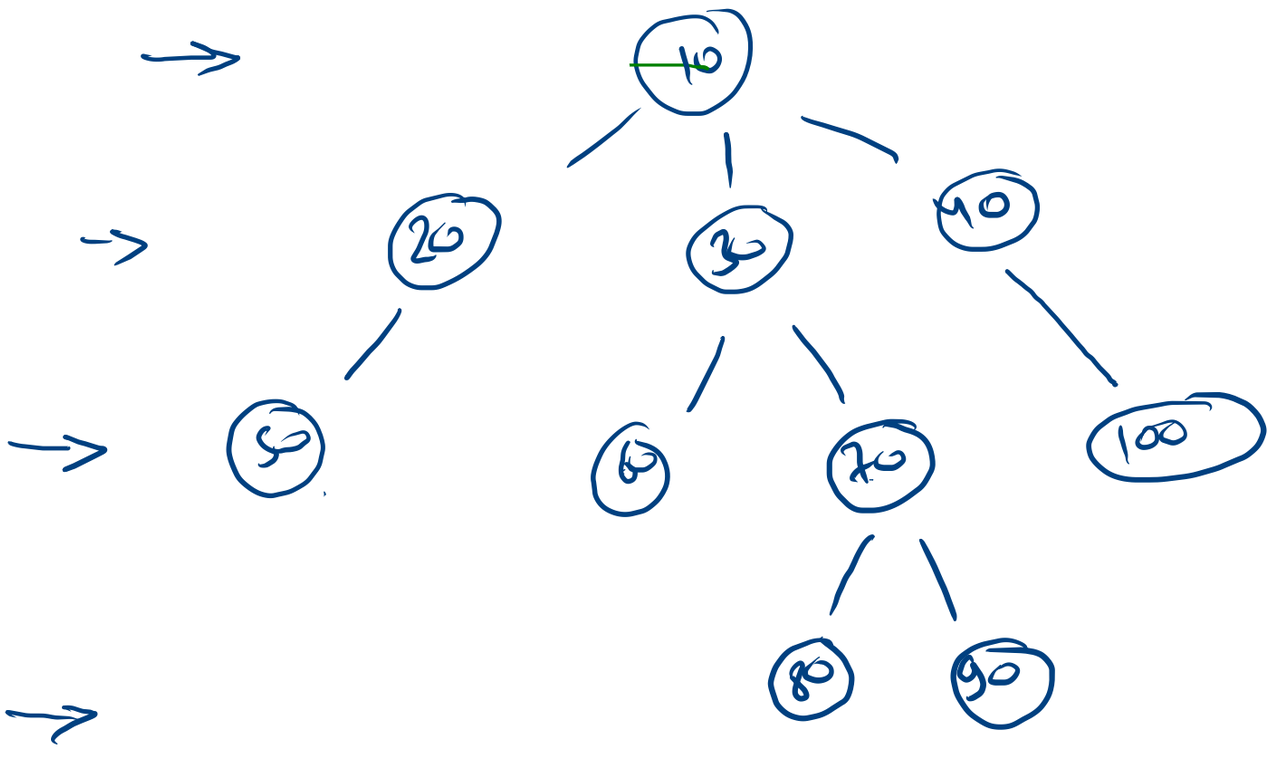


```java
public static void levelOrder(Node node){
    Queue<Node> q = new LinkedList<>();

    q.add(node);

    while(q.size() != 0){
        Node n = q.remove();
        System.out.print(n.data+" ");
        for(Node child: n.children){
            q.add(child);
        }
    }

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



10 20 30 40 50
60 70 100 80 90 .





10

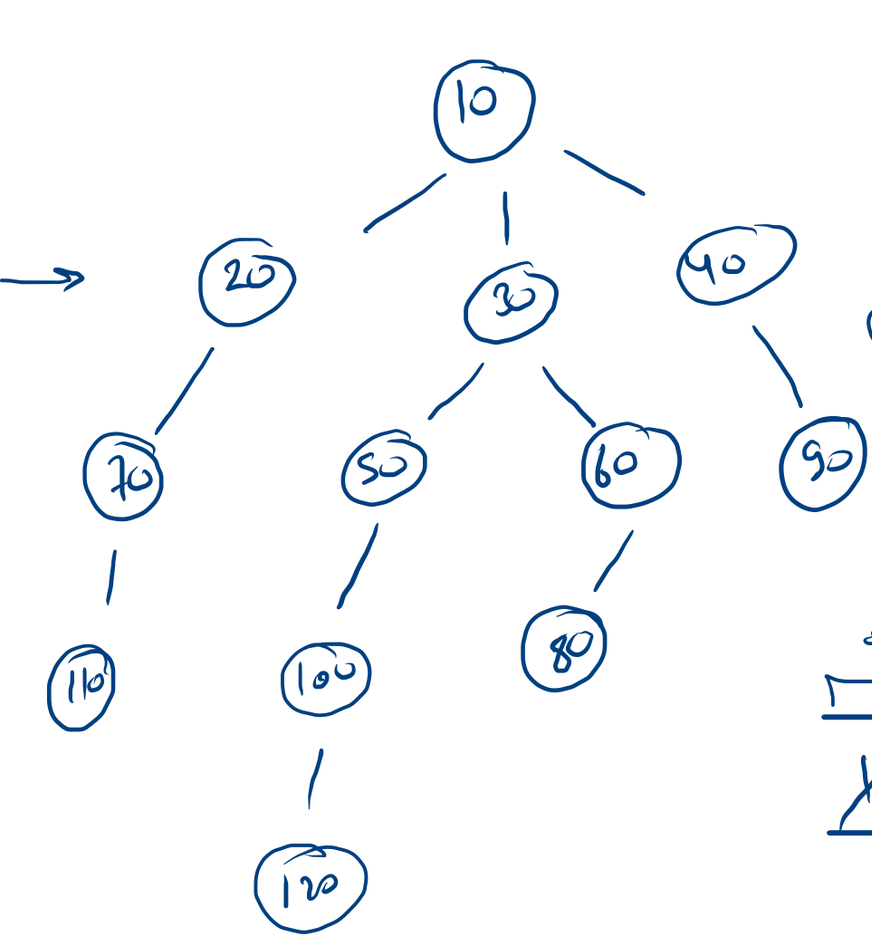
20 30 40

50 60 70 100

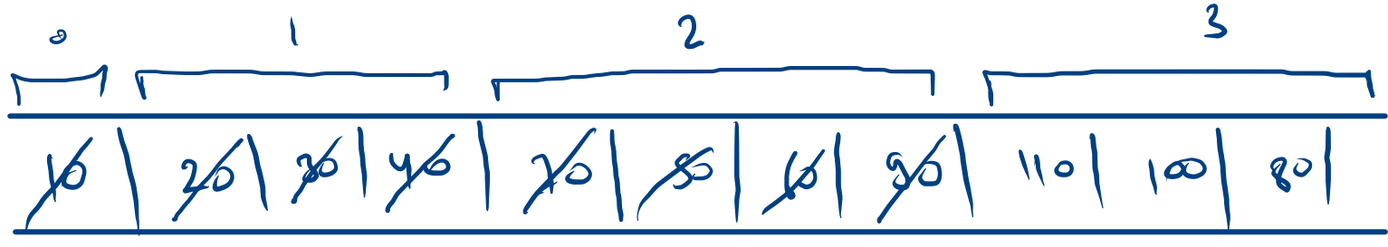
80 90

within	delimiter
	2 2

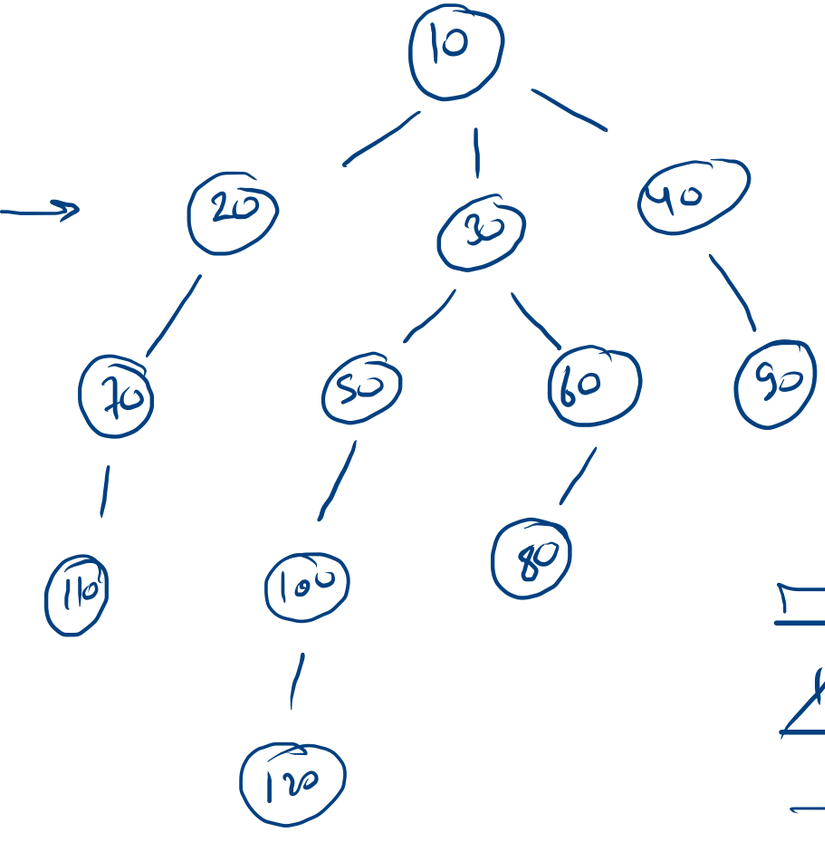
delimiter



10 20 30 40 70 50 60 90



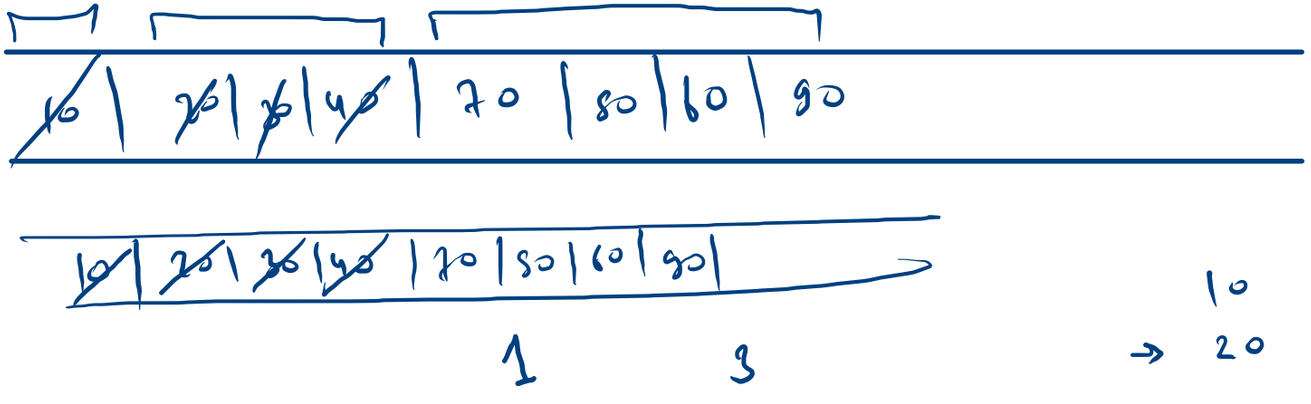
3



10
20 30 40

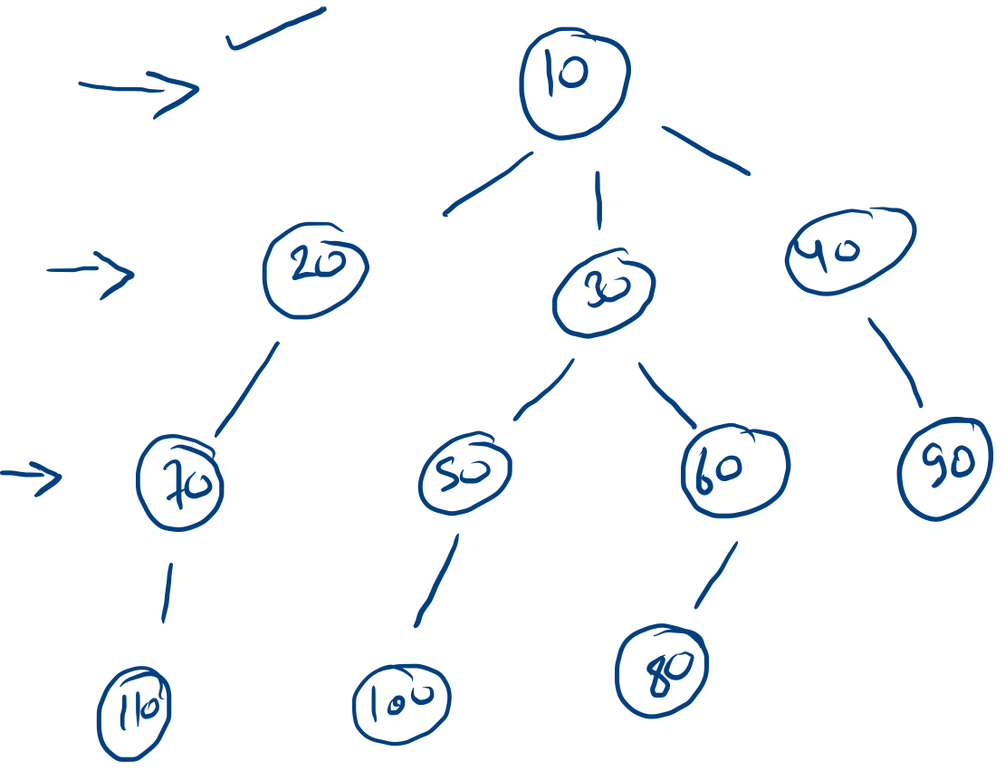
for(count times)
remove
print(
add(child

println
count= q.size();



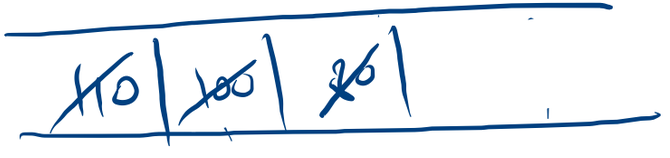
1 3

10
→ 20 30 40

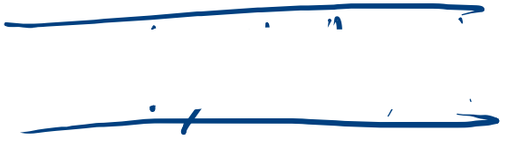


10
20 30 40
70 50 60 90
110 100 80

main q

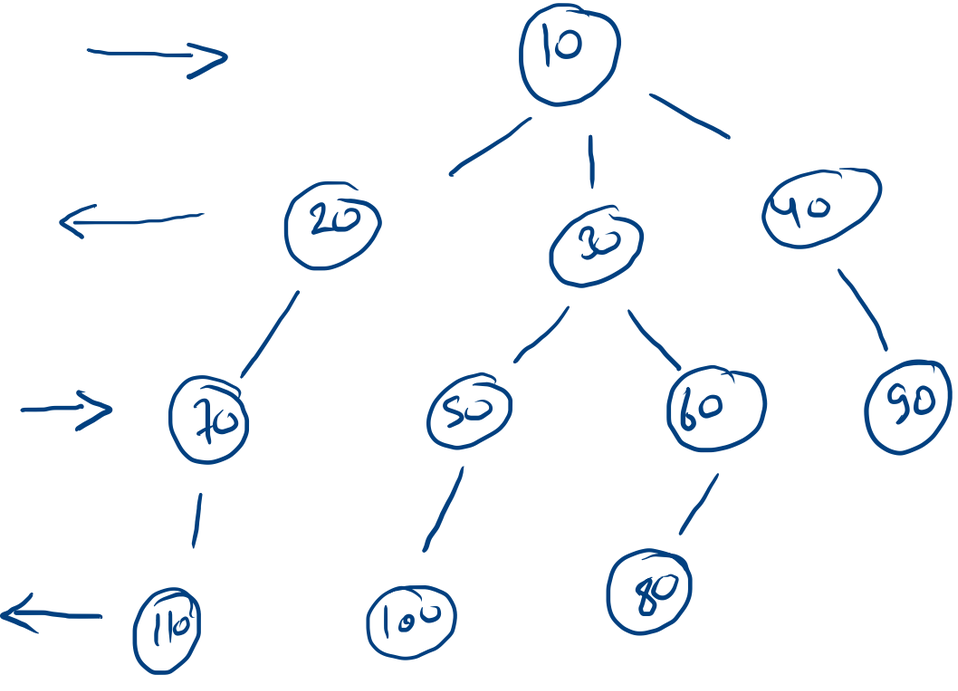


child q



H.W.

Levelorder
Linewise Zig Zag



10

40 30 20

70 50 60 90

80 100 110

public class {

static class {

}

public static {
new A();

}

}

public class {

class {

mylis

}

public static {
new A();

}

}