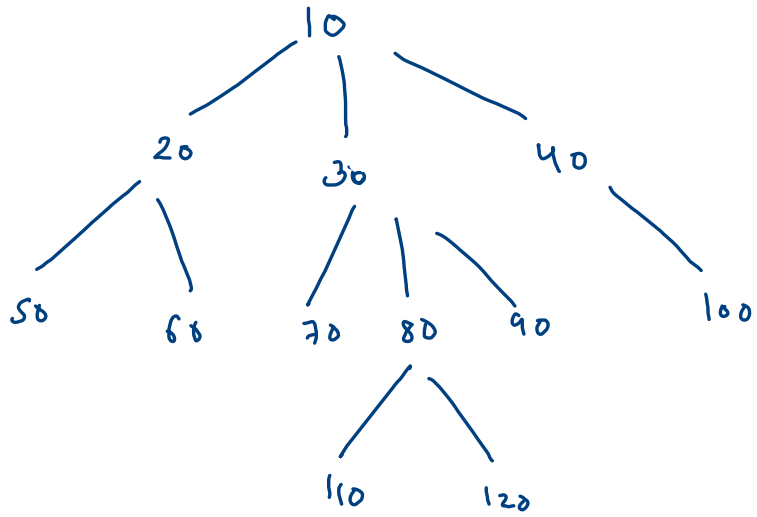
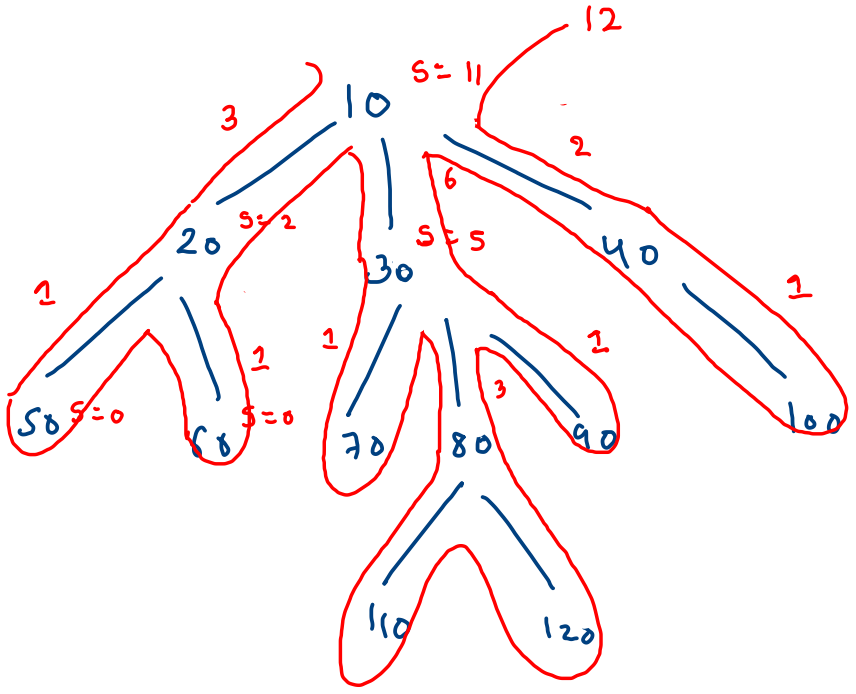
Size Of Generic Tree

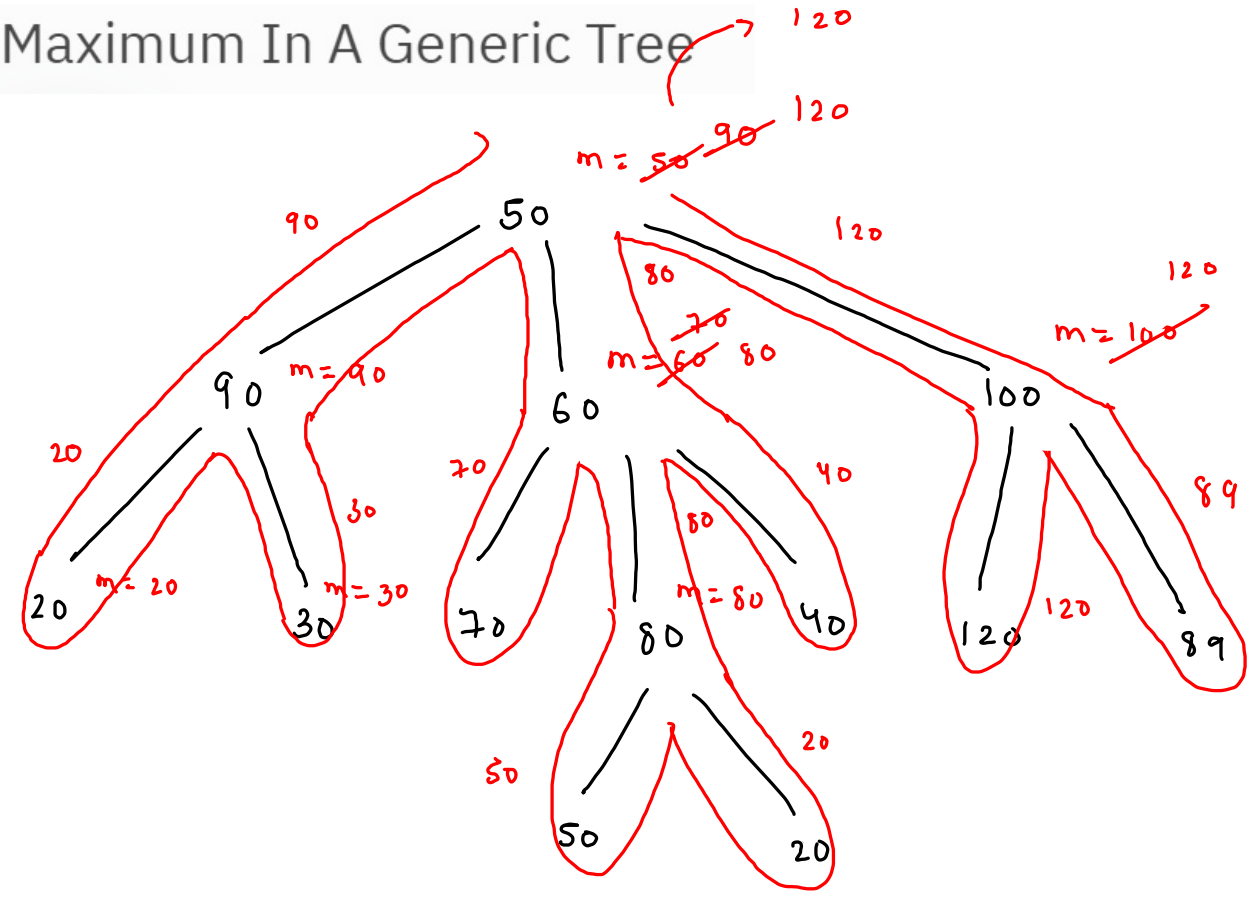


size (10) = size (20) + size(30)
+ size(40) + 1




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

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

        int cfs = size(child);

        size += cfs;
    }

    return size + 1;
}
```

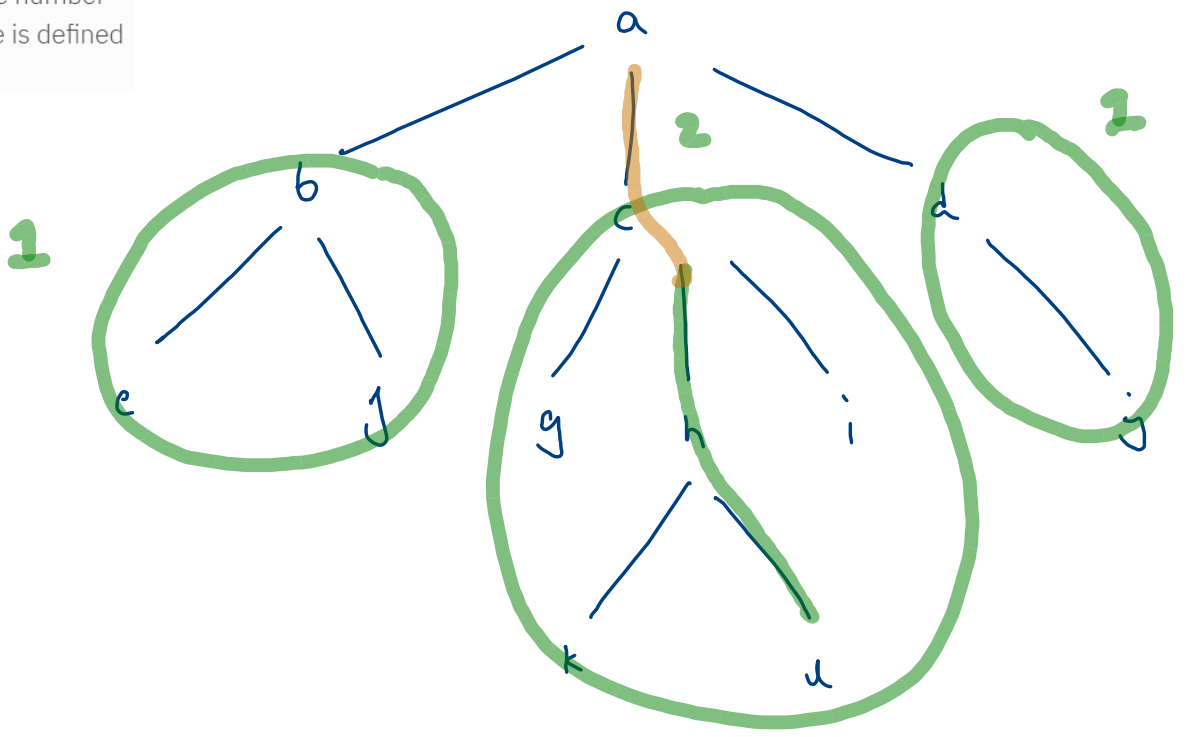
Maximum In A Generic Tree



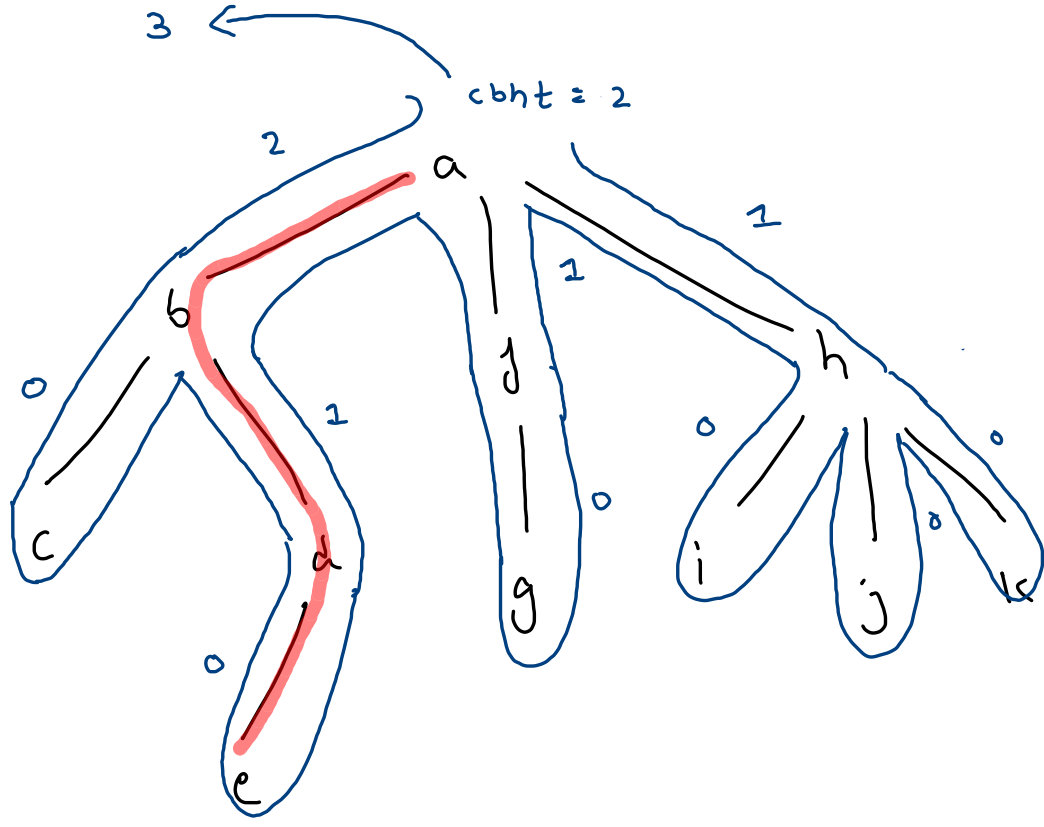

```java
public static int max(Node node) {
    int max = node.data;

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

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

    return max;
}
```

Height Of A Generic Tree

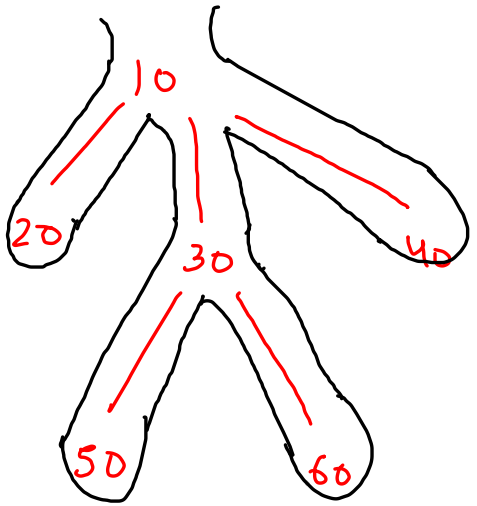
expected to find the height of tree. Depth of a node is defined as the number of edges it is away from the root (depth of root is 0). Height of a tree is defined as depth of deepest node.

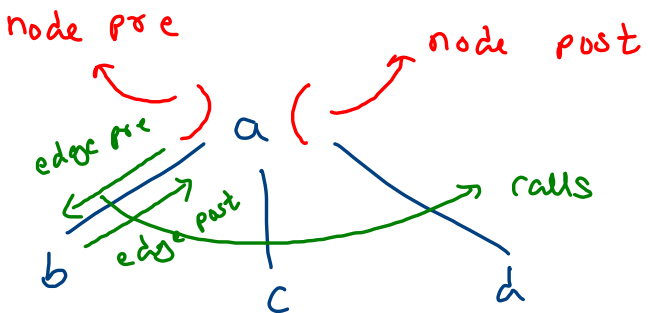


```java
public static int height(Node node) {

    int cbht = -1; //height is required in terms of edges

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

        if(ht > cbht) {
            cbht = ht;
        }
    }

    return cbht + 1;

}
```



Generic Tree - Traversals (pre-order, Post-order)

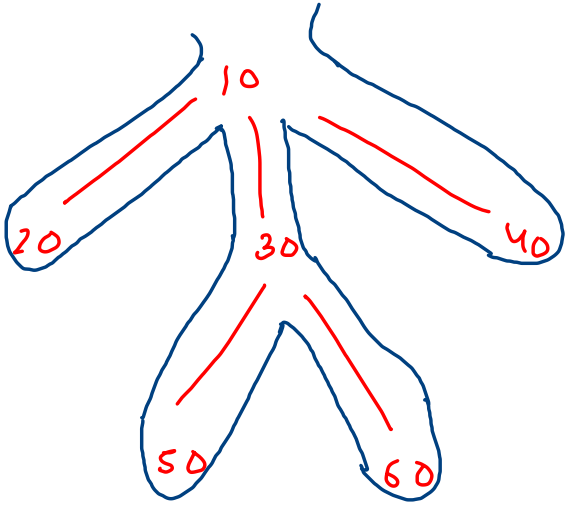




node pre 10
edge pre 10 - 20
node pre 20
node post 20
edge post 10 - 20
edge pre 10 - 30
node pre 30
edge pre 30 - 50
node pre 50
node post 50
edge post 30 - 50

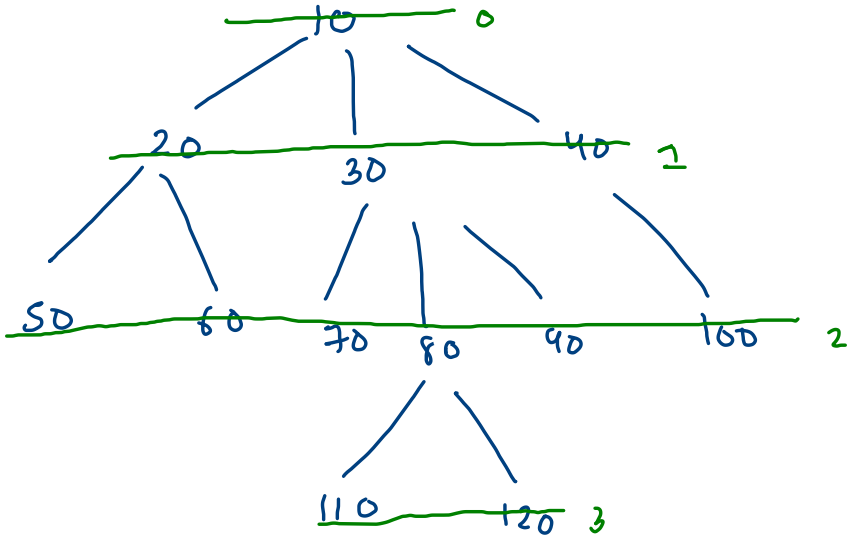
edge pre 30-60
node pre 60
node post 60
edge post 30-60
node post 30
edge post 10-30
edge pre 10-40
node pre 40
node post 40
edge post 10-40
node post 10

```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);

        //edge (call)
        traversals(child);

        //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
```

Level-order Of Generic Tree

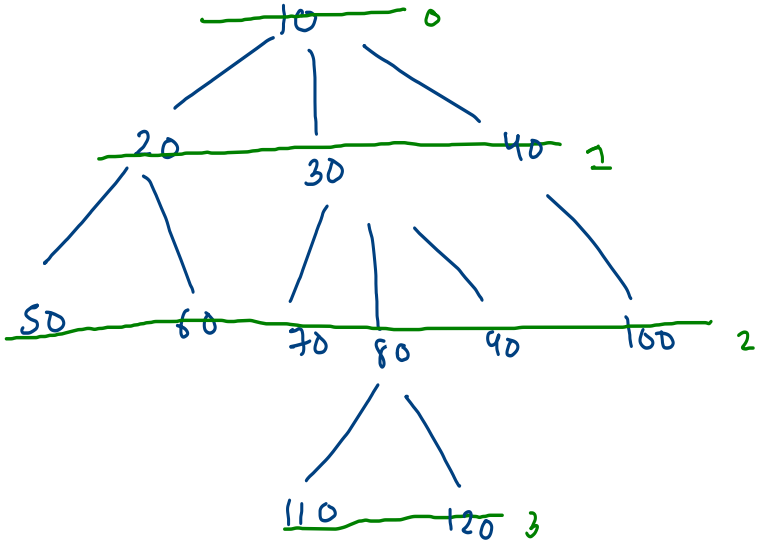


do : 10 20 30 40 50 60

70 80 90 100 110 120

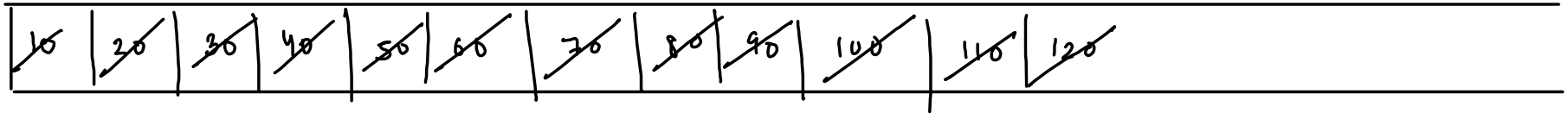
queue FIFO

(Sibling > child)

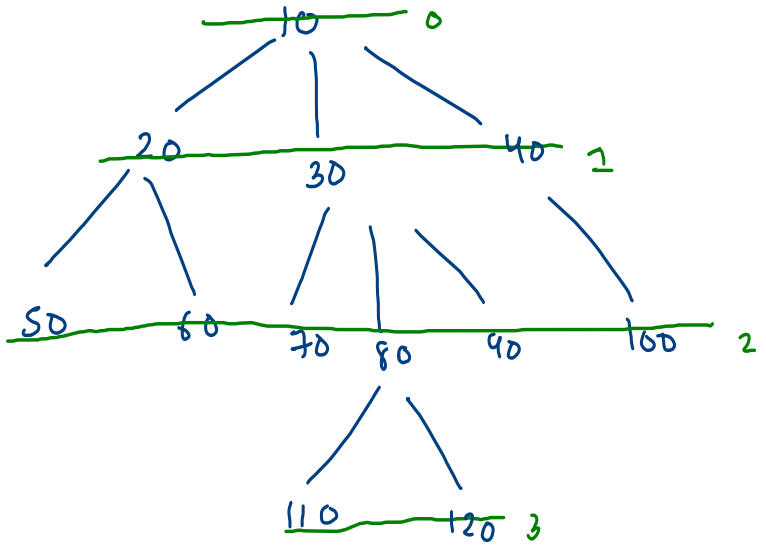


normal level order :

algo
- remove
- work
- add children



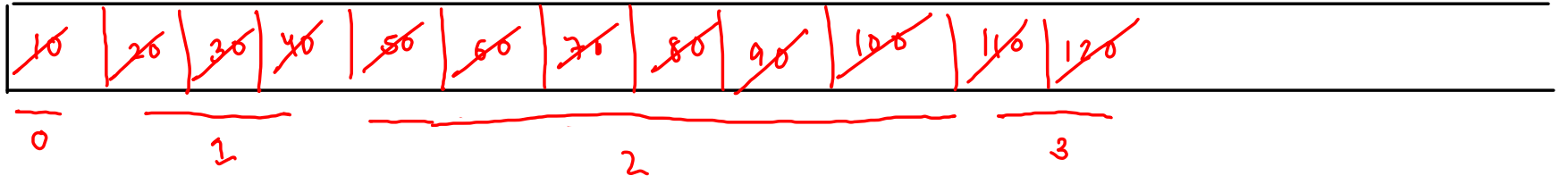
10 20 30 40 50 60 70 80 90 100 110 120



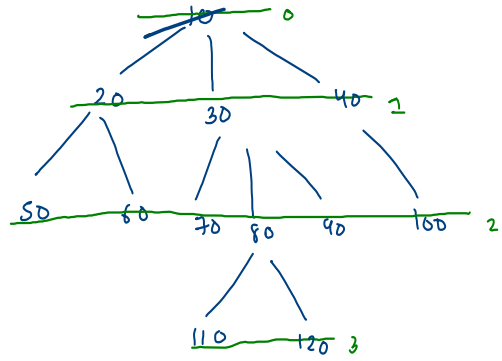
normal lo

10 20 30 40 50 60 70

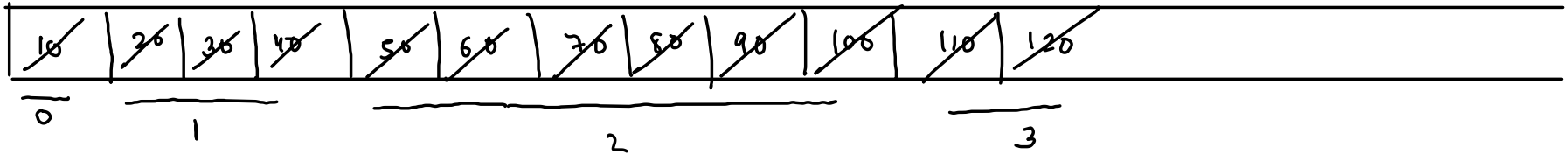
80 90 100 110 120

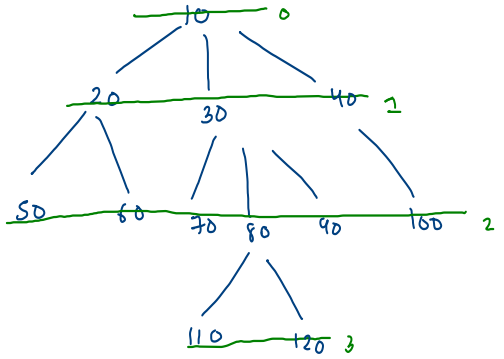


```java
public static void levelOrder(Node node){
    ArrayDeque<Node>q = new ArrayDeque<>();

    q.add(node);

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

        //work
        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 .





level order linewise

10

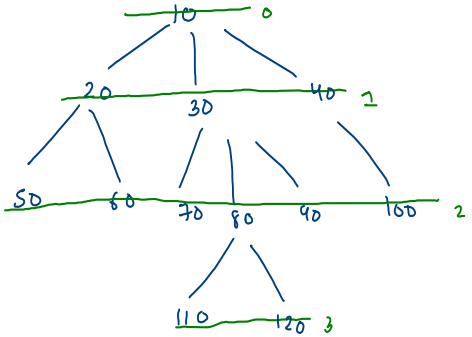
20 30 40

50 60 70 80 90 100

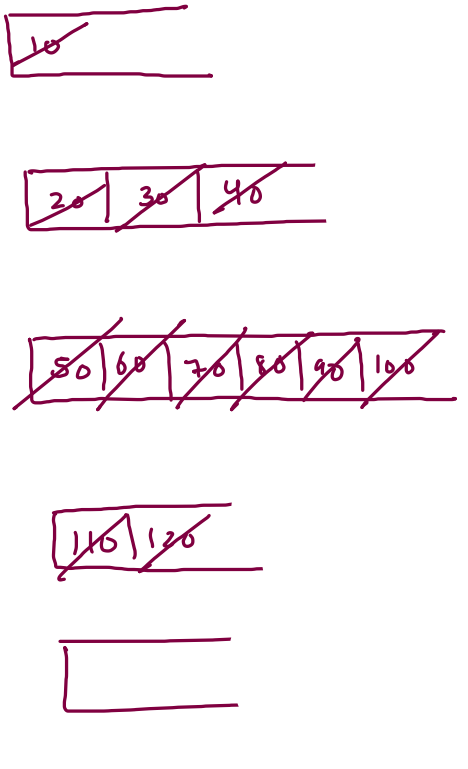
110 120

Levelorder Linewise (generic Tree)

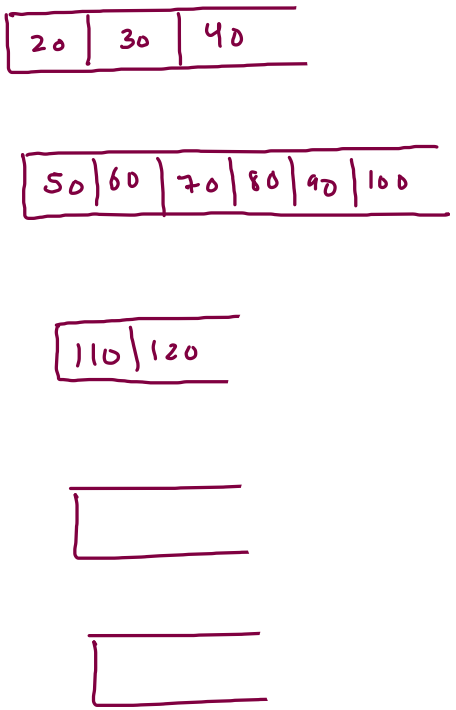
① two queue



mQ



cQ

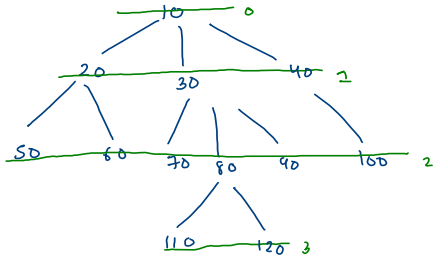


10

20 30 40

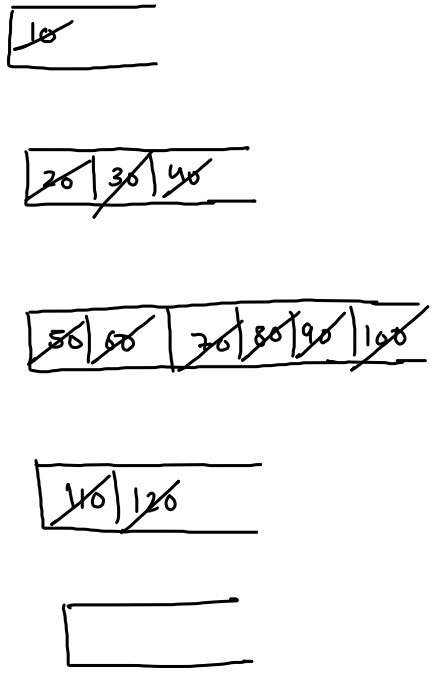
50 60 70 80 90 100

110 120

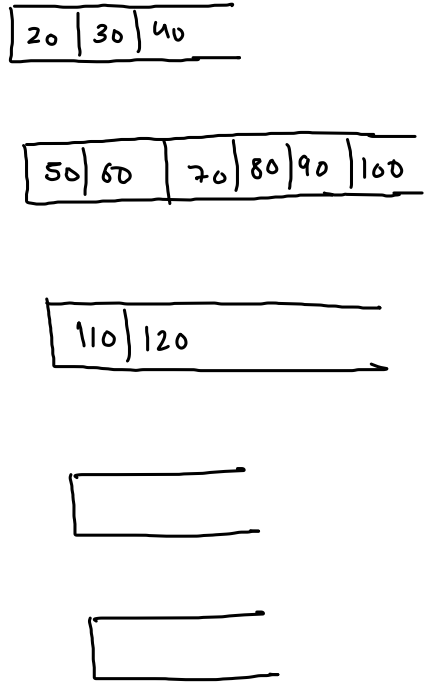


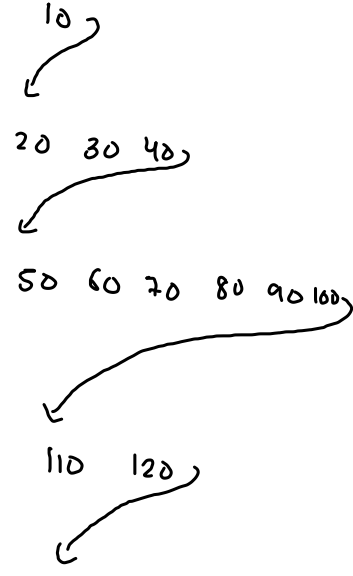

```java
public static void levelOrderLinewise(Node node){

    ArrayDeque<Node>mQ = new ArrayDeque<>();
    ArrayDeque<Node>cQ = new ArrayDeque<>();

    mQ.add(node);

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

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

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


        if(mQ.size() == 0) {
            System.out.println();
            mQ = cQ;
            cQ = new ArrayDeque<>();
        }
    }
}
```

mQ



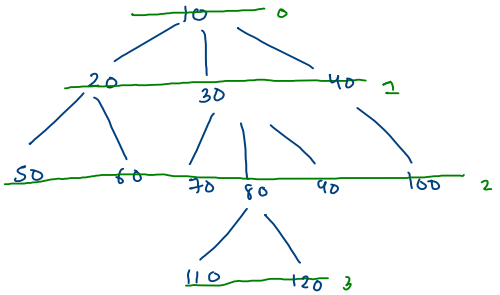
cQ





Levelorder Linewise (generic Tree)

** (2) only one queue (count method)

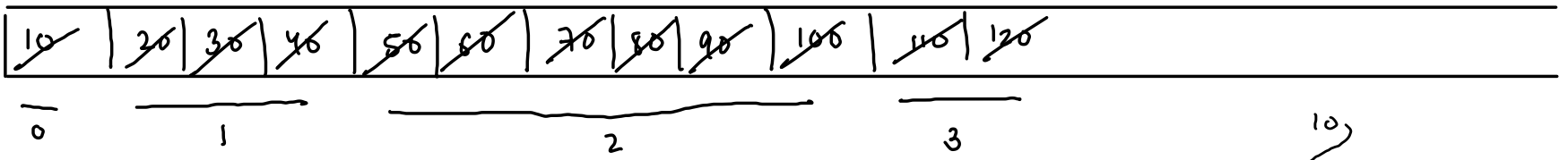


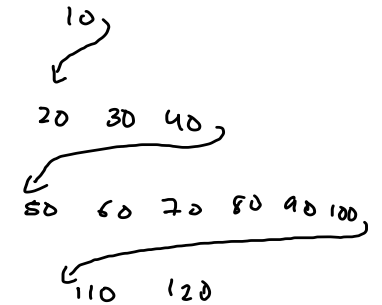
c = 2

→ x

count times [remove
work
add children

→ x+1





```java
while(q.size() > 0) {
    int count = q.size();
    for(int k=0; k < count;k++) {
        //remove
        Node rem = q.remove();

        //work
        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();
}
```