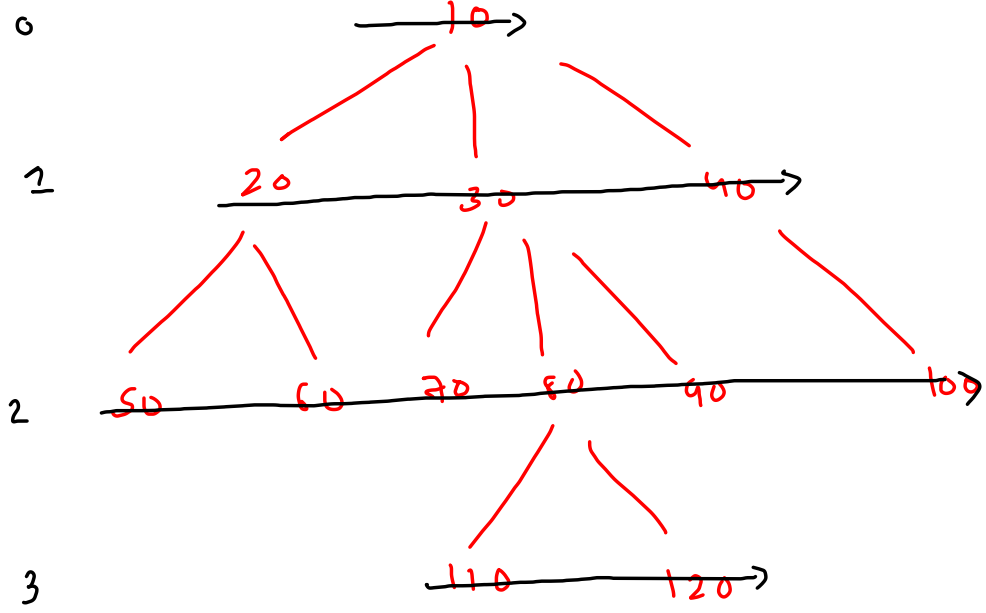
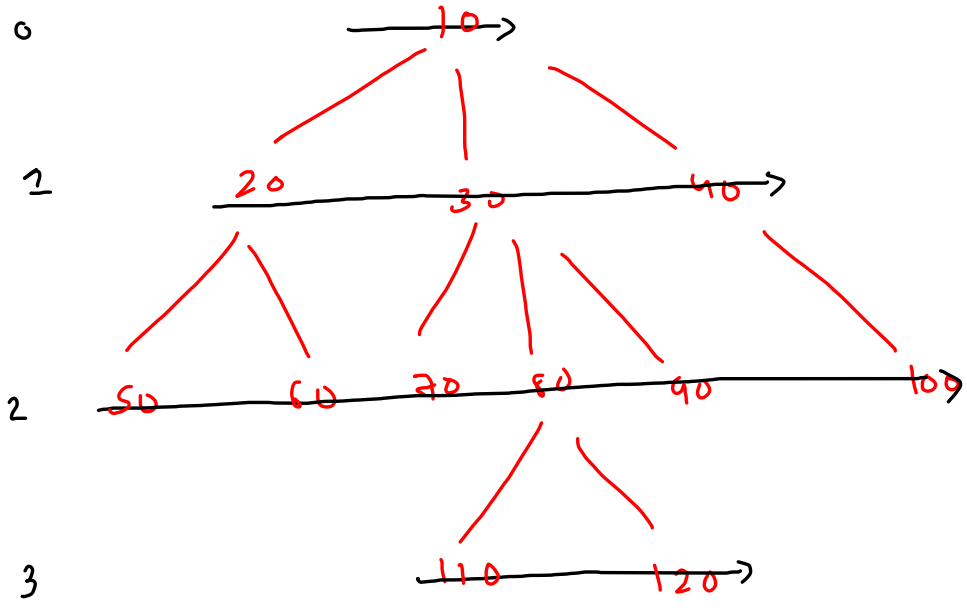
```java
public static void levelOrder(Node root){
    Queue<Node>q = new ArrayDeque<>();

    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



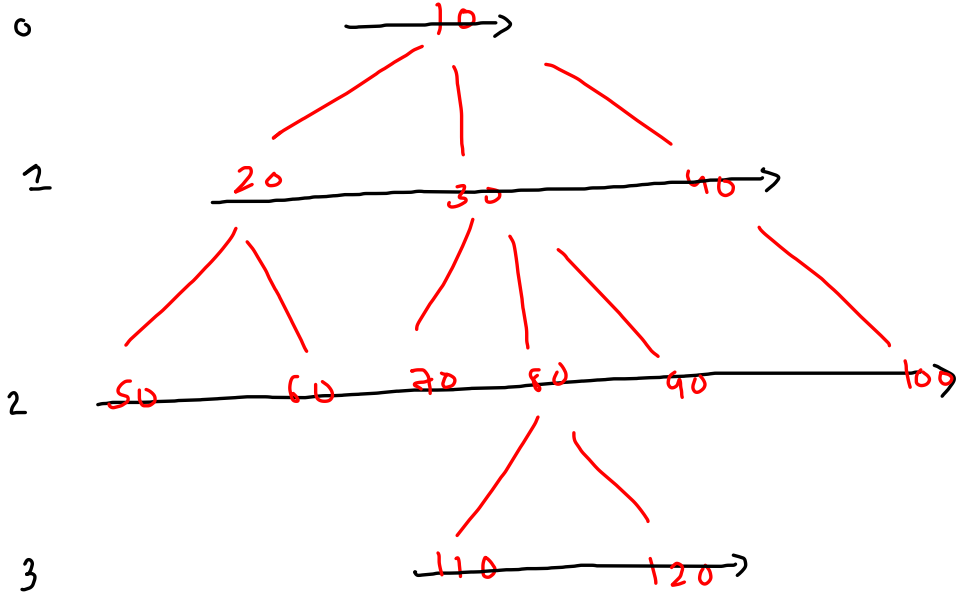
0 -> 10

1 -> 20 30 40

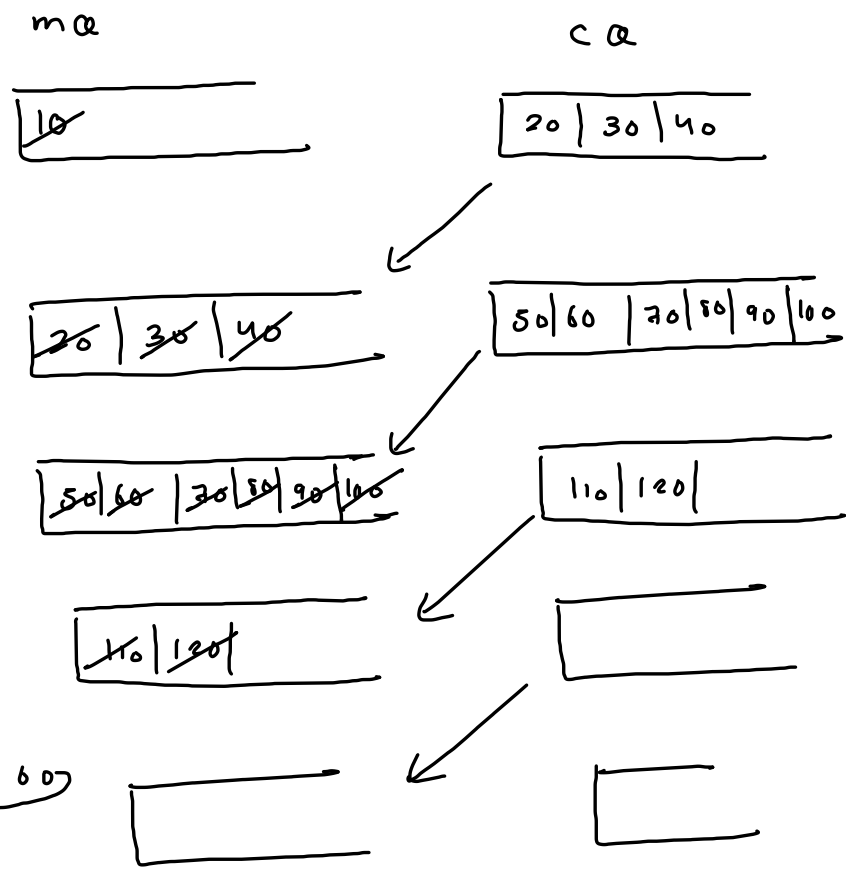
2 -> 50 60 70 80 90 100

3 -> 110 120

- 2 queues
- delimiter (-1)
- queue (count)
- class



① two queue



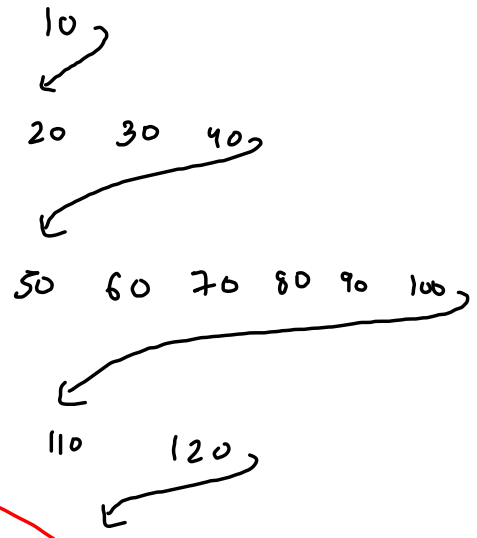
10

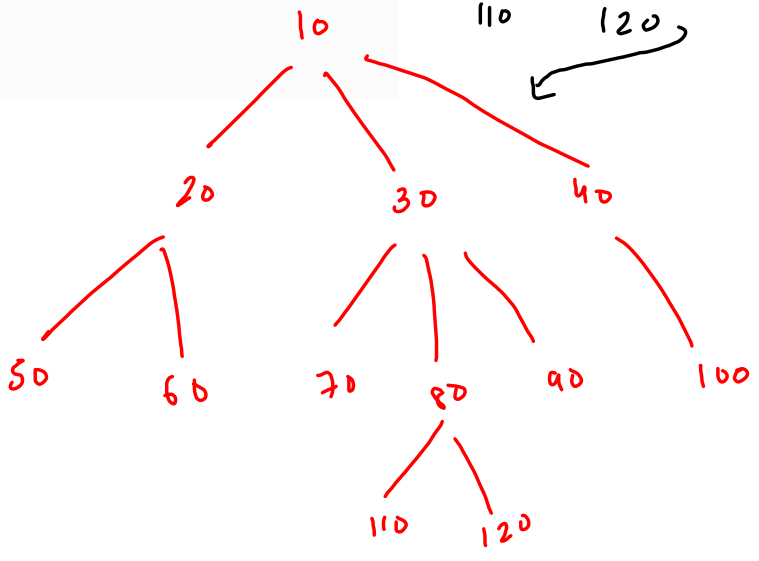
20 30 40

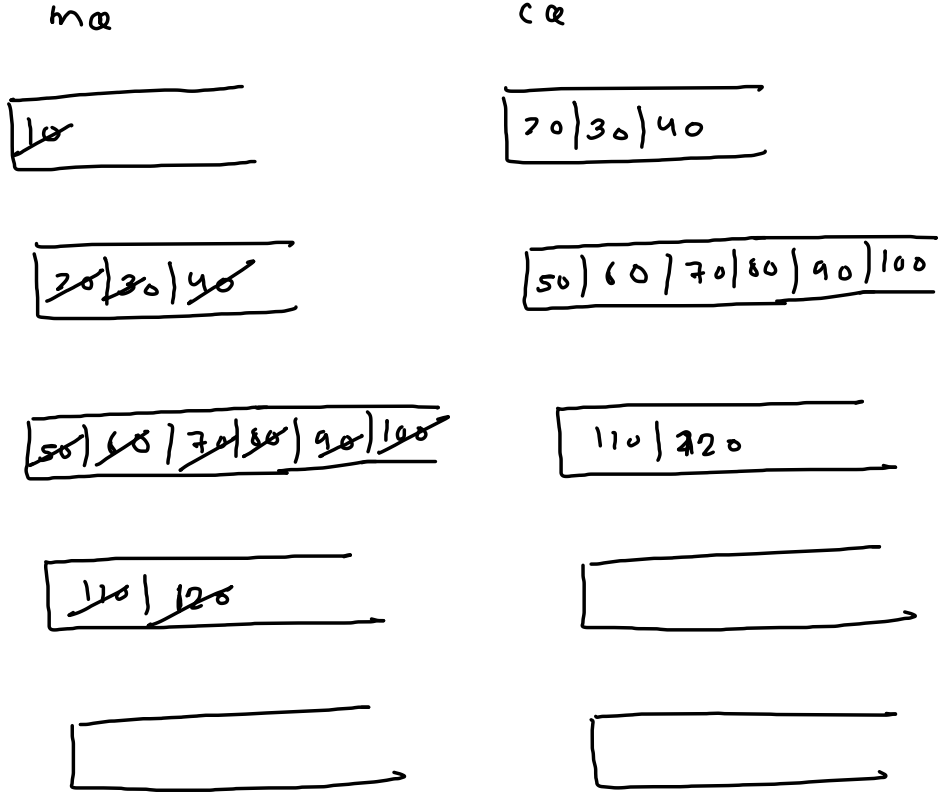
50 60 70 80 90 100

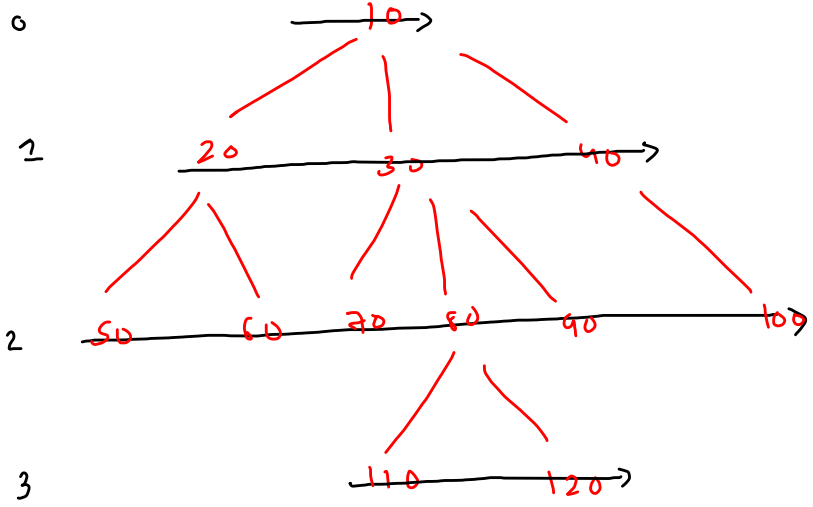
110 120

```java
while(mq.size() > 0) {
    //remove
    Node rem = mq.remove();

    //print
    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 rem was main queue's last element
    if(mq.size() == 0) {
        //1
        System.out.println();
        mq = cq;
        cq = new ArrayDeque<>();


    }
}
```

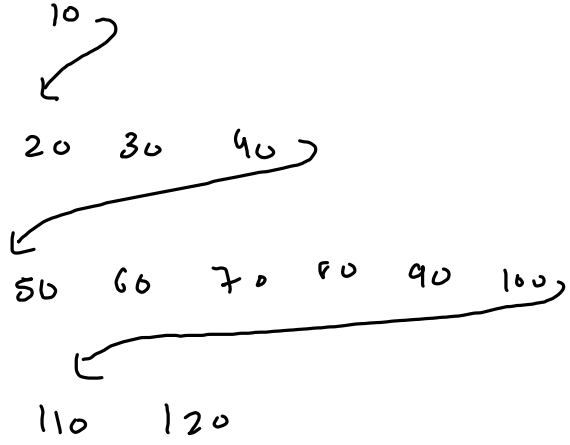






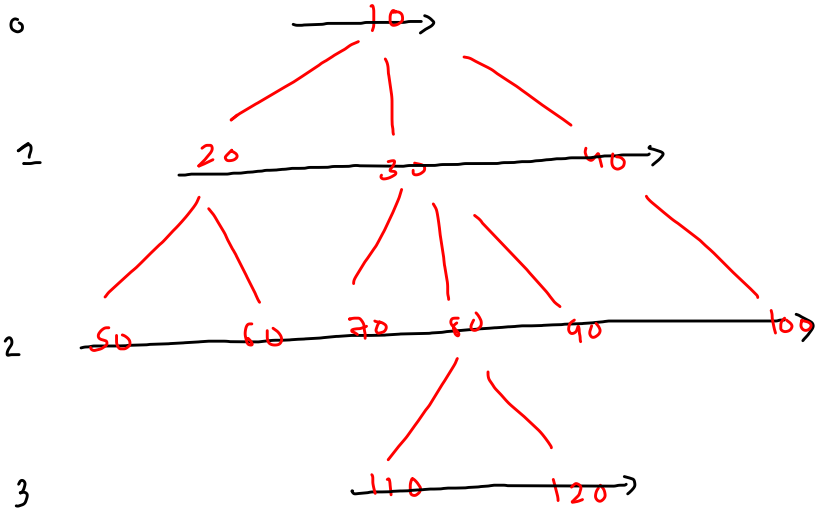


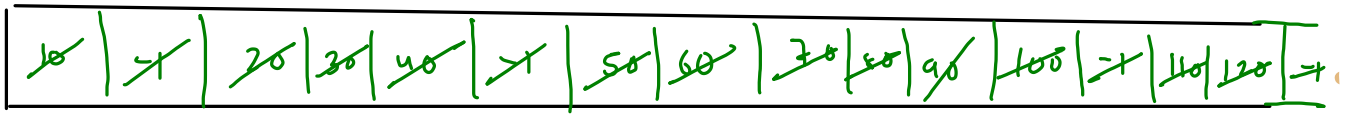
(2) - delimiter

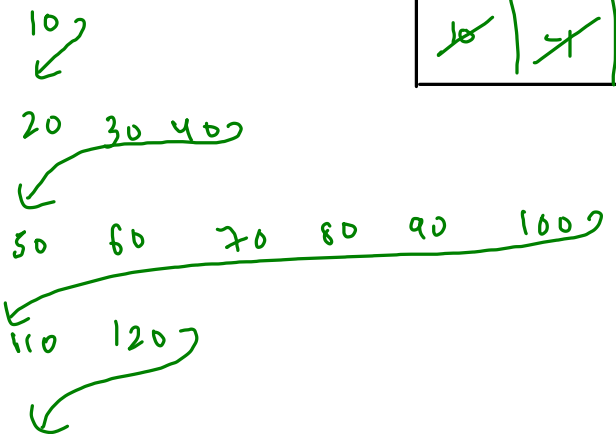


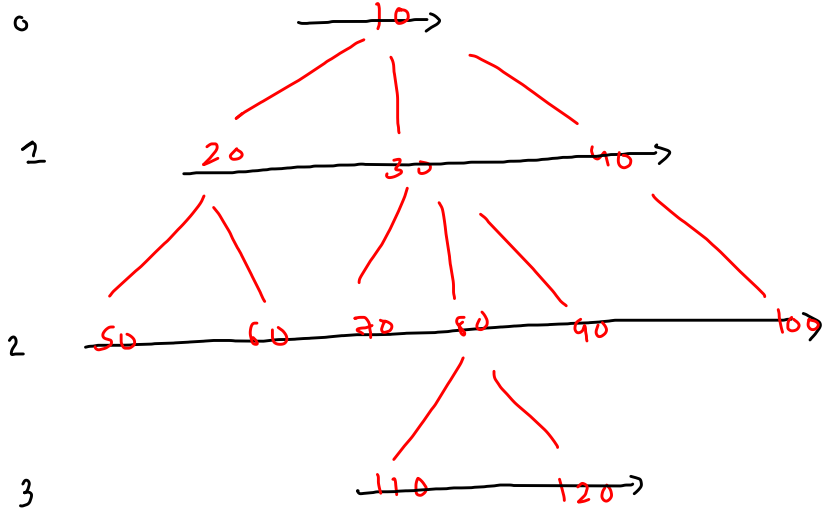
10	-1	20	30	40	-1	50	60	70	80	90	100	-1	110	120	-1

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

    q.add(root);
    q.add(new Node(-1));

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

        if(rem.data == -1) {
            System.out.println();

            if(q.size() == 0) {
                break;
            }

            q.add(rem);
            continue;
        }

        //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);
        }
    }
}
```









(**) single queue

(count)

c = 2

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

10

20 30 40

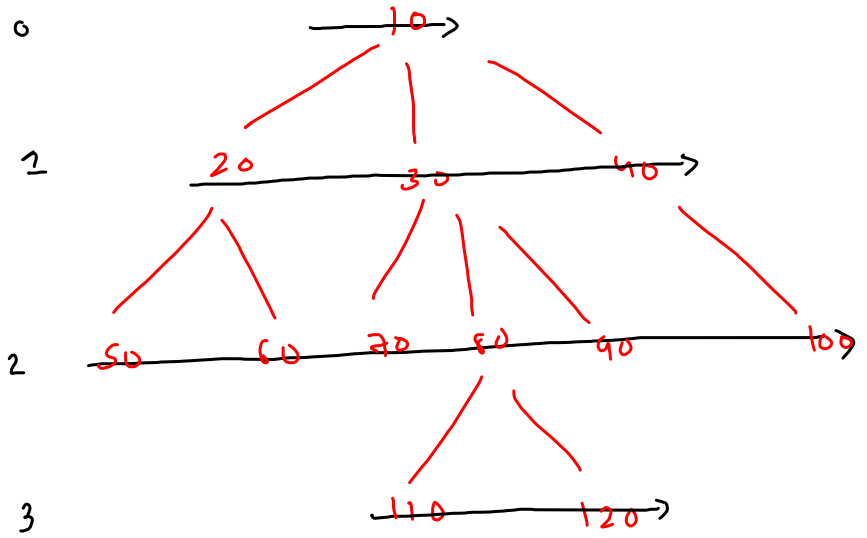
50 60 70 80 90 100

110 120

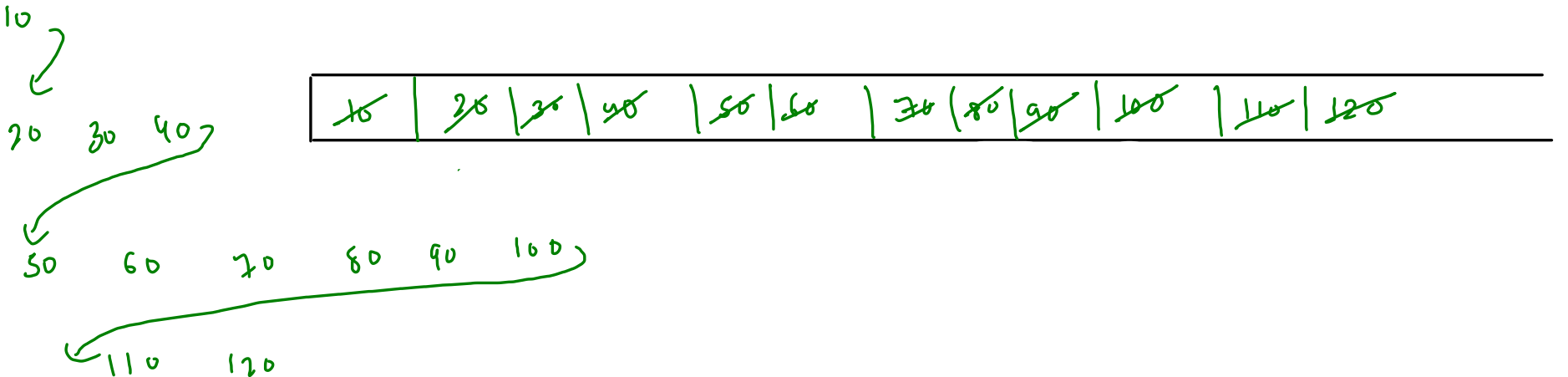
remove

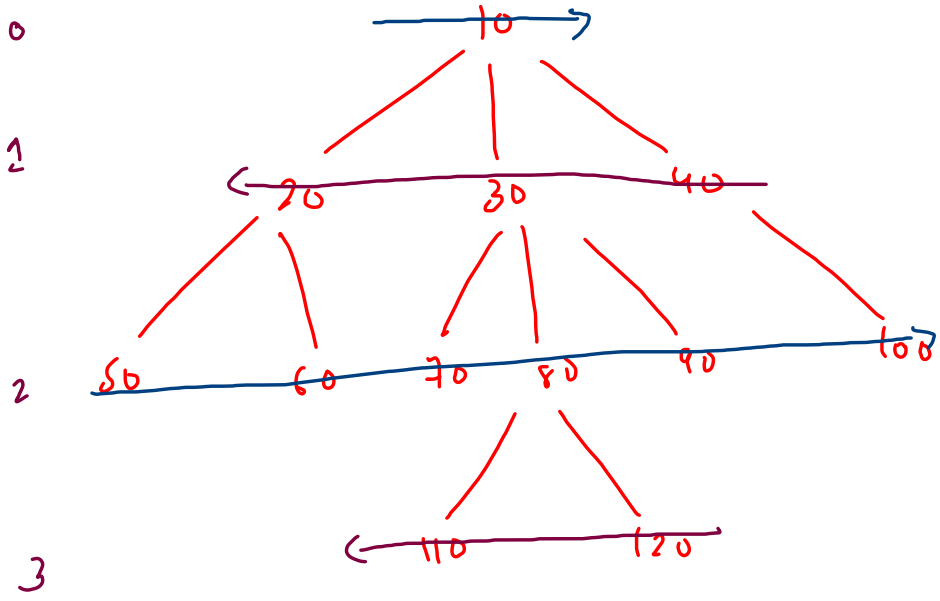
print

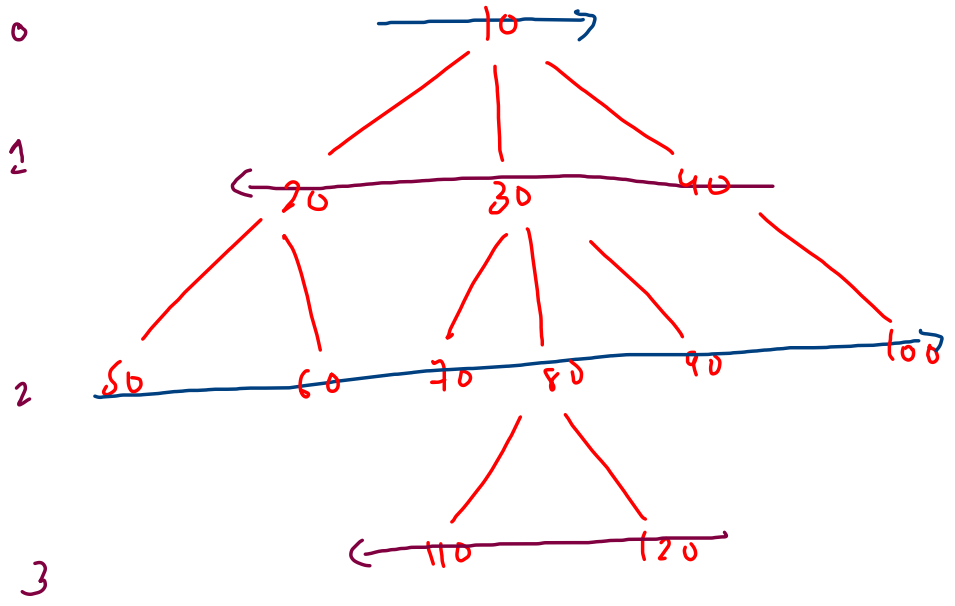
ac




```java
while(q.size() > 0) {
    int count = q.size();

    while(count-- > 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();
}
```




0
10
1
40
30
20
2
50
60
70
80
90
100
3
120
110



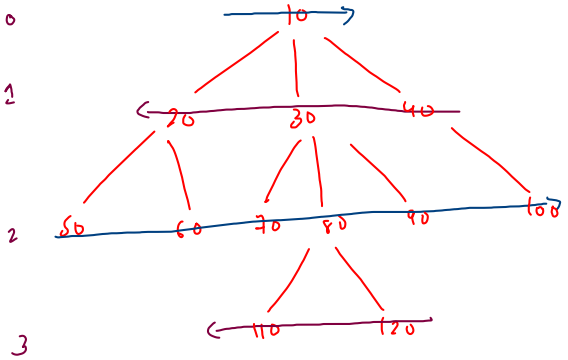
10

40 30 20

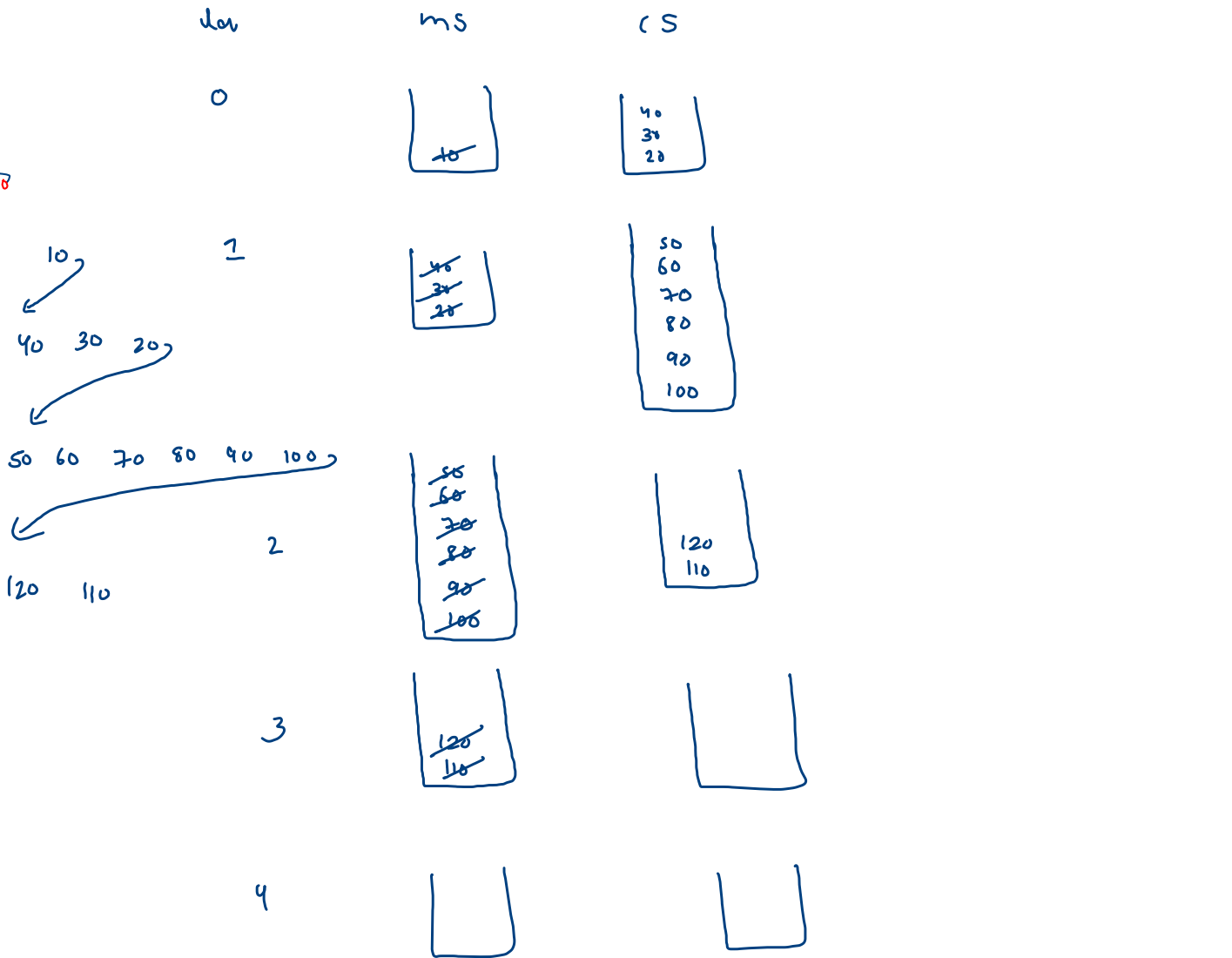
50 60 70 80 90 100

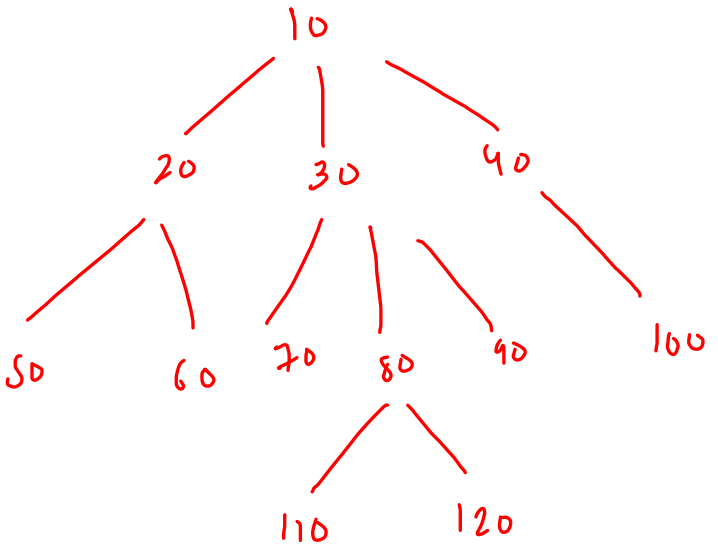
120 110

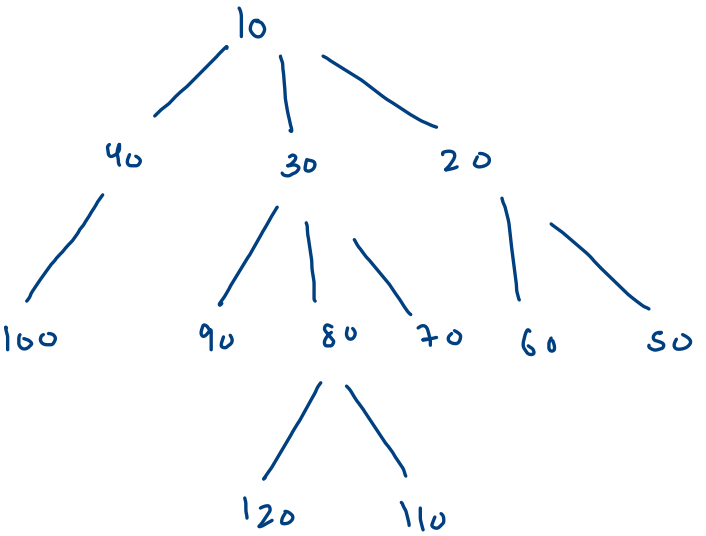
lev	ms	cs
0	10	40 30 20
1	~~40 30 20~~	50 60 70 80 90 100
2	~~50 60 70 80 90 100~~	120 110
3	~~120 110~~	

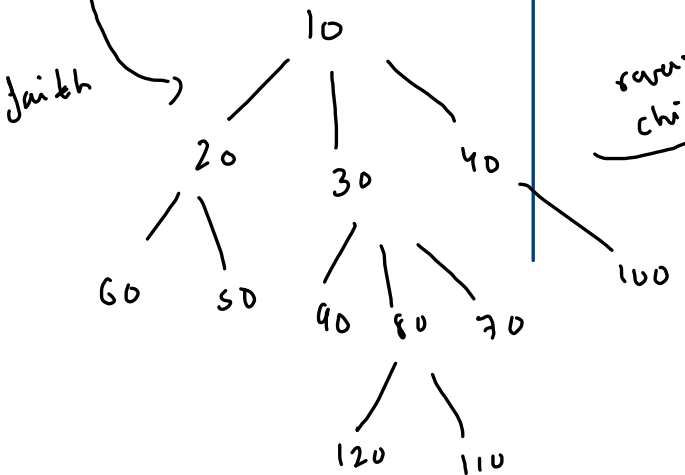


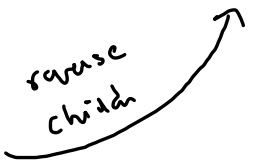

```
while(ms.size() > 0) {
    //remove
    Node rem = ms.pop();

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

    //add children
    if(lev % 2 == 0) {
        //push L->R
        for(int i=0; i < rem.children.size();i++) {
            Node child = rem.children.get(i);
            cs.push(child);
        }
    }
    else {
        //push R->L
        for(int i=rem.children.size()-1; i >= 0;i--) {
            Node child = rem.children.get(i);
            cs.push(child);
        }
    }

    if(ms.size() == 0) {
        lev++;
        System.out.println();
        ms = cs;
        cs = new Stack<>();
    }
}
```



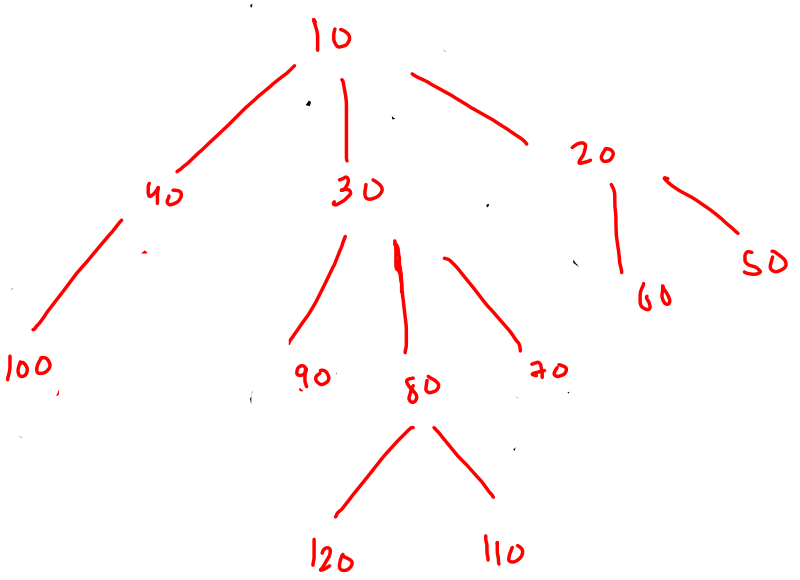

10
20
30
40
50
60
70
80
90
100
110
120

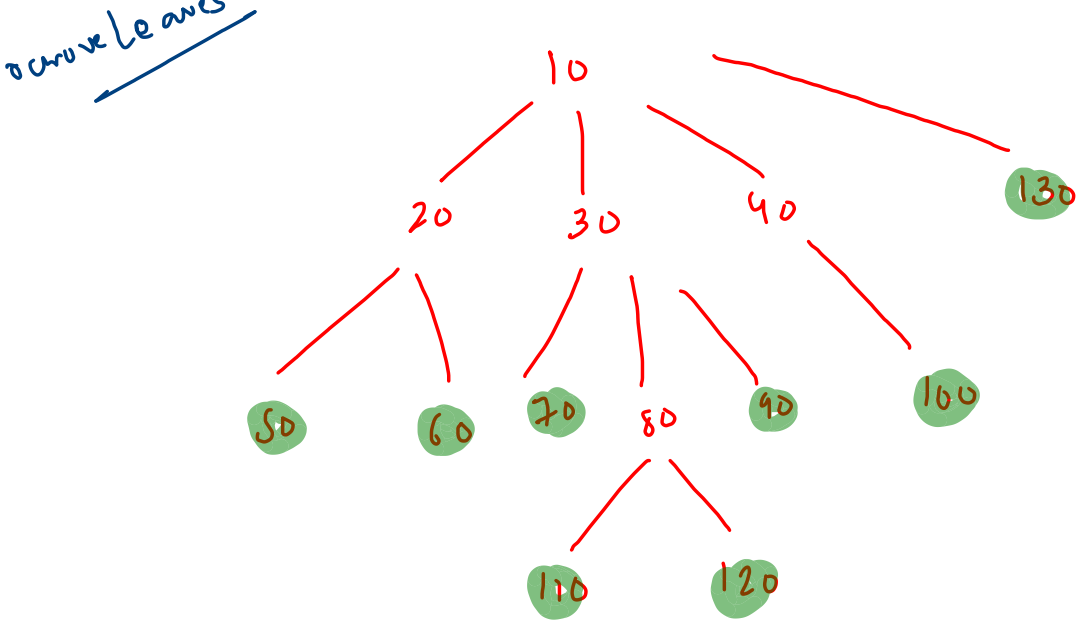

10
40
30
20
100
90
80
70
60
50
120
110

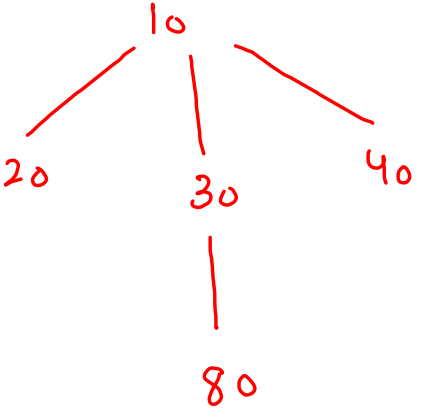

faith
10
20
30
40
60
50
90
80
70
120
110
100


reverse
child

```java
public static void mirror(Node node){
  //faith on each child's family
  for(int i=0; i < node.children.size();i++) {
      Node child = node.children.get(i);
      mirror(child);
  }

  //self work -> reverse node's children arraylist
  int lo = 0;
  int hi = node.children.size()-1;

  while(lo < hi) {
      Node lon = node.children.get(lo);
      Node hin = node.children.get(hi);

      node.children.set(lo,hin);
      node.children.set(hi,lon);

      lo++;
      hi--;
  }
}
```

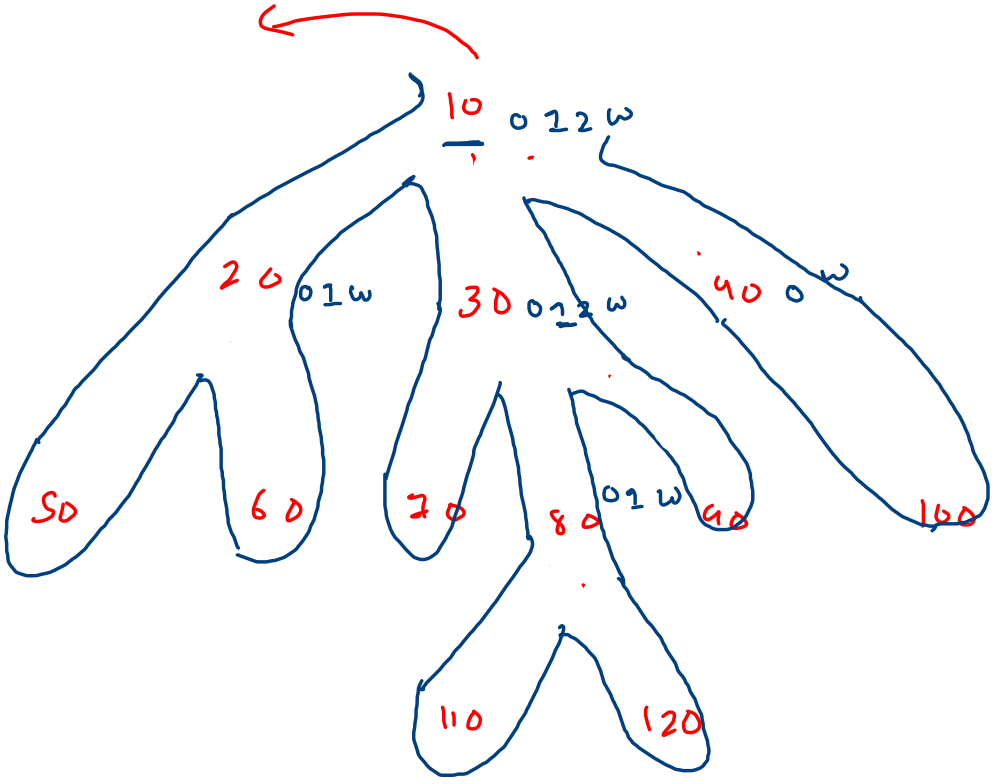



drive Leaves
10
20
30
40
130
50
60
70
80
90
100
110
120


10
20
30
40
80

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

    //faith on each child
    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        removeLeaves(child);
    }

    //self work
    for(int i=node.children.size()-1; i >= 0;i--) {
        Node child = node.children.get(i);
        if(child.children.size() == 0) {
            //child  is a leaf node
            node.children.remove(i);
        }
    }
}
```

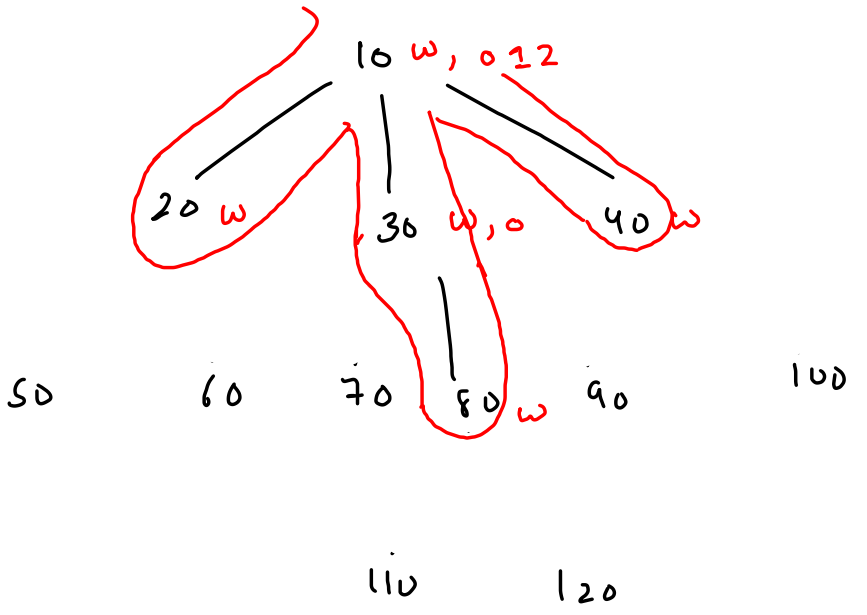
10



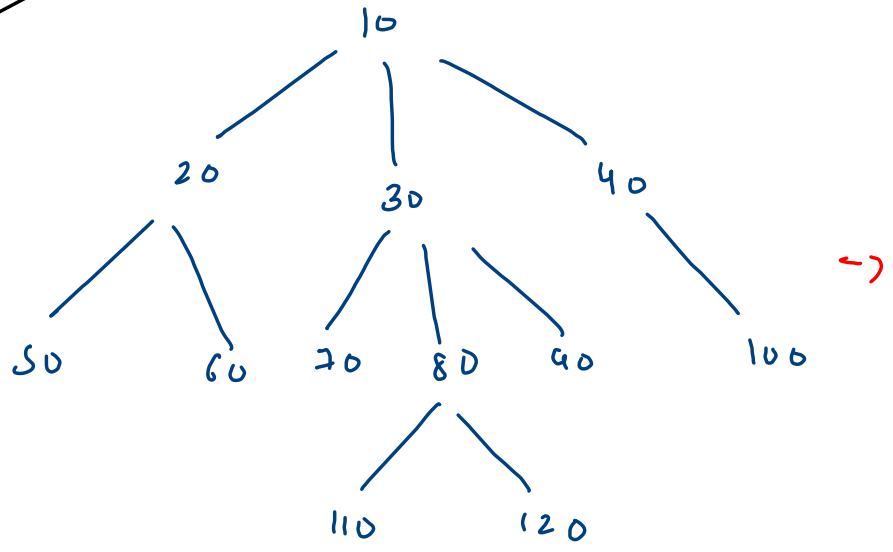
```java
public static void removeLeaves(Node node) {
    //self work
    for(int i=node.children.size()-1; i >= 0;i--) {
        Node child = node.children.get(i);
        if(child.children.size() == 0) {
            //child  is a leaf node
            node.children.remove(i);
        }
    }

    //faith on each child
    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);
        removeLeaves(child);
    }

}
```

w

calls

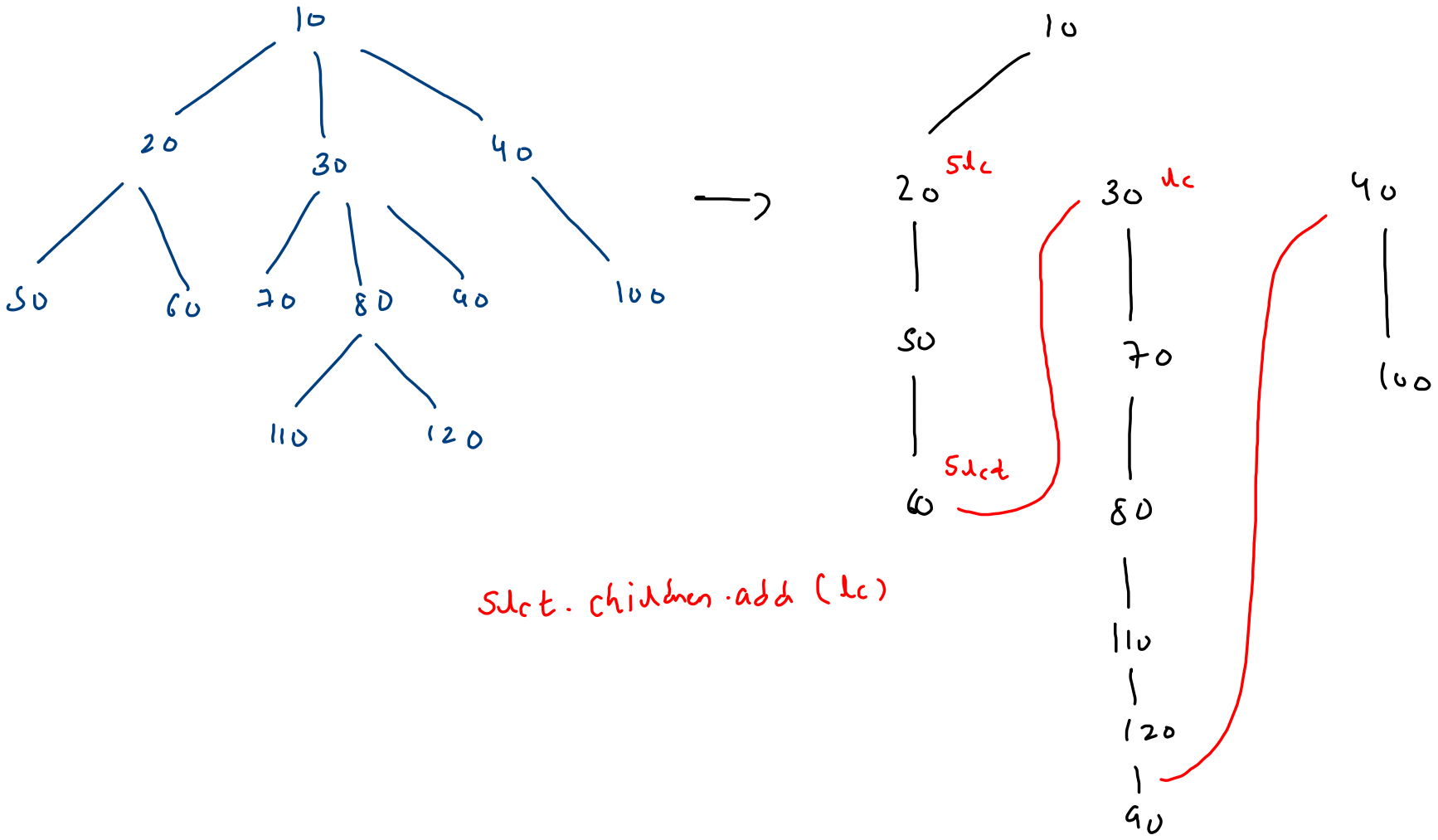


Linearise

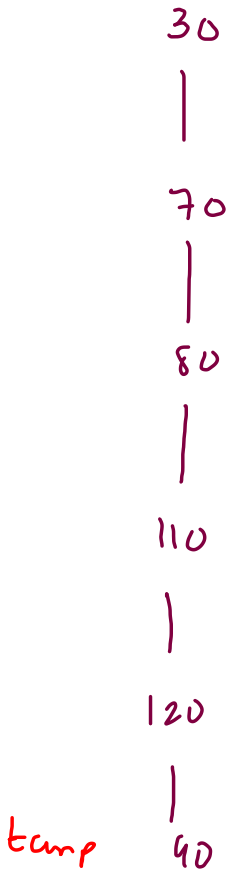


→

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


10
20
30
40
50
60
70
80
90
100
110
120
10
20
Slc
30
lc
40
50
70
100
Slct
60
80
110
120
90
Slct.children.add(lc)

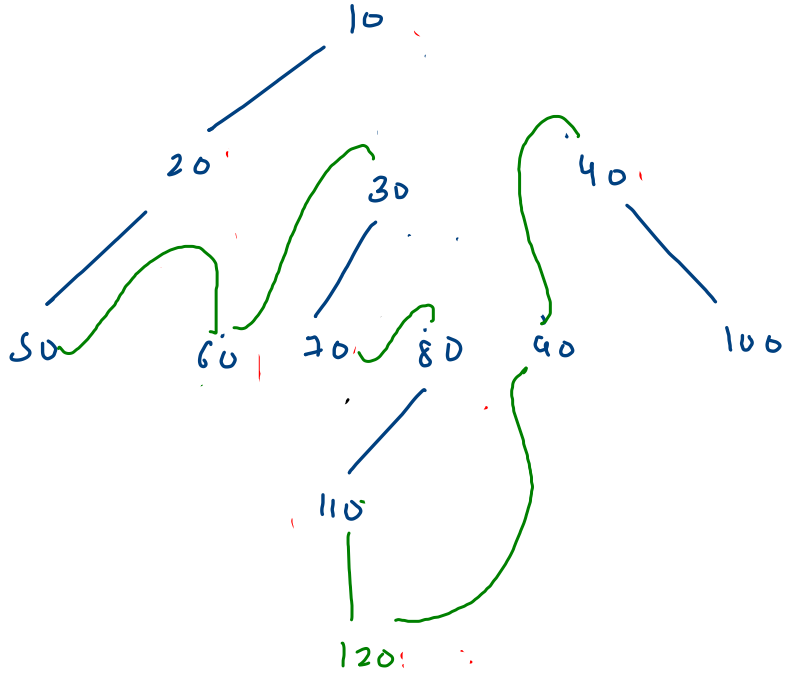
```java
public static Node getTail(Node node) {
    Node temp = node;

    while(temp.children.size() != 0) {
        temp = temp.children.get(0);
    }

    return temp;
}
```



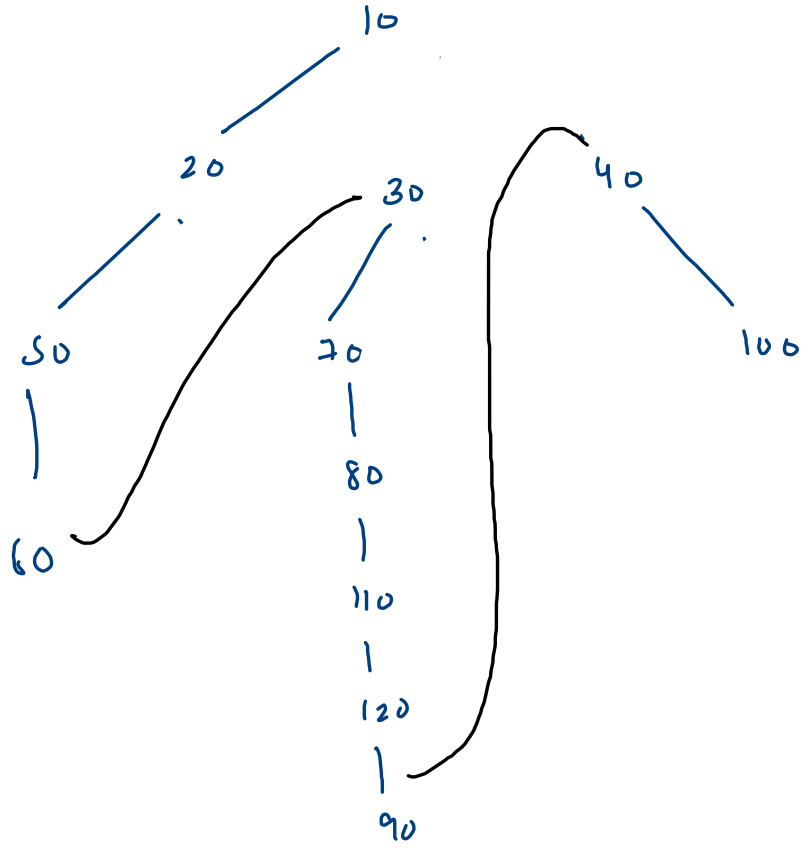
return 90.

```java
public static void linearize(Node node){
  //faith on each faith
  for(int i=0; i < node.children.size();i++) {
      Node child = node.children.get(i);
      linearize(child);
  }

  while(node.children.size() > 1) {
      int size = node.children.size();
      Node lc = node.children.get(size-1); //last child
      Node slc = node.children.get(size-2); //second last child

      //connections
      node.children.remove(size-1);
      Node slct = getTail(slc); //second last child tail
      slct.children.add(lc);
  }
}
```

```java
public static Node getTail(Node node) {
    Node temp = node;

    while(temp.children.size() != 0) {
        temp = temp.children.get(0);
    }

    return temp;
}
```



tail = linearise (dc);



return tail;

```java
public static Node linearize_helper(Node node) {
    if(node.children.size() == 0) {
        return node;
    }

    Node lc = node.children.get(node.children.size()-1);
    Node otail = linearize_helper(lc);  //overall tail

    while(node.children.size() > 1) {
        Node slc = node.children.get(node.children.size()-2);
        Node slct = linearize_helper(slc);

        //connections
        node.children.remove(node.children.size()-1);
        slct.children.add(lc);

        lc = slc;
    }

    return otail;
}
```