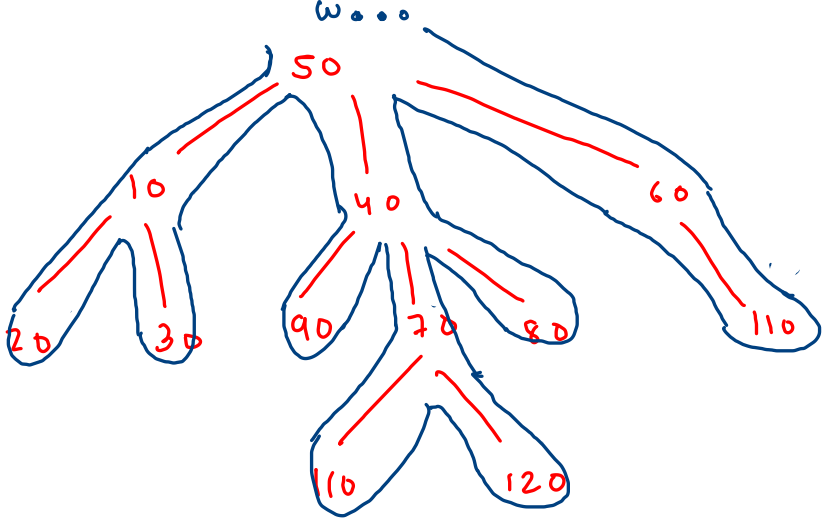


Size = ~~0 1 2 3 4 5 6 7 8 9 10 11~~ 12

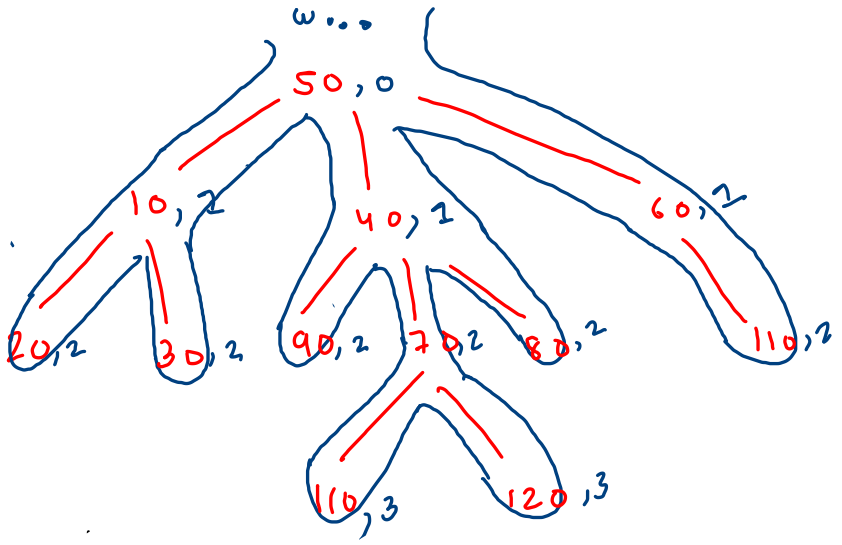
min : ~~∞~~ ~~50~~ 10

max : ~~-∞~~ ~~50~~ ~~90~~ ~~110~~ 120

node pre

```
size++;
min = math.min(min, node.val);
max = math.max(max, node.val);
for (child : node.children) {
    traverse(child);
}
```

height : ?




```java
public static void travel(Node node,int lev) {
    //node pre
    size++;
    min = Math.min(min,node.data);
    max = Math.max(max,node.data);
    height = Math.max(height,lev);

    //travel
    for(Node child : node.children) {
        travel(child,lev+1);
    }
}
```

12

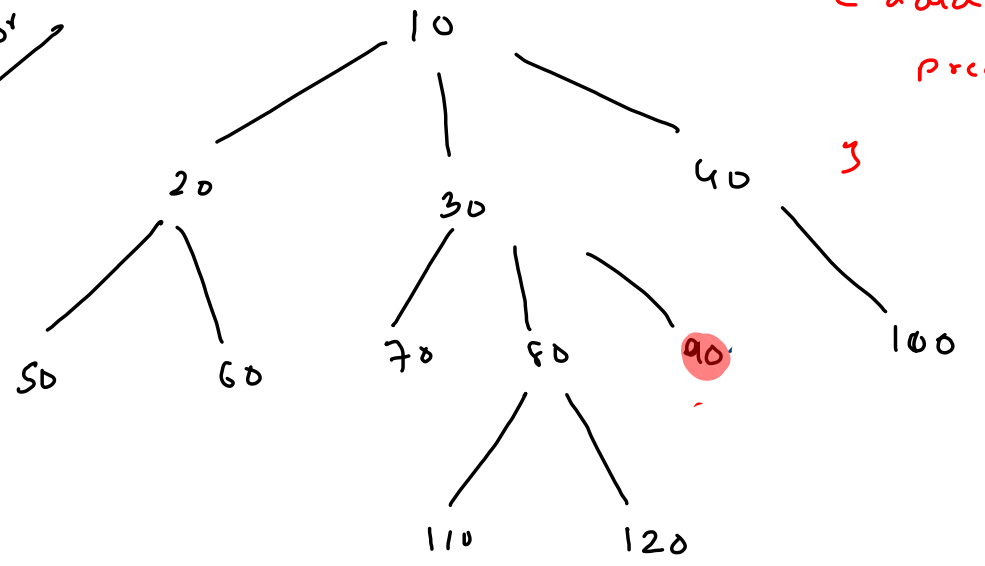
size : 0 1 2 3 4 5 6 7 8 9 10 11

min : ~~-∞~~ 50 10

max : ~~-∞~~ 50 90 ~~110~~ 120

ht : 0 1 2 3

predecessor
successor




```
c.data = data.{
    pred = p;
}
```

```
p.data = data{
    succ = c;
}
```

p -> prev

c -> cur

pre-order

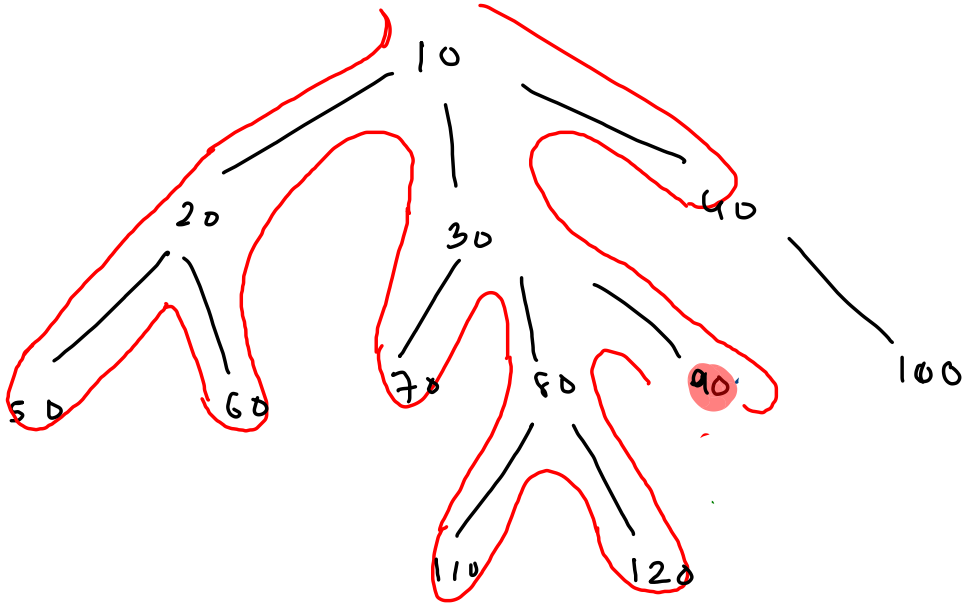
10 20 50 60 30 70

80 110 120 90 40 100

predecessor = 120

successor = 40

data = 90



node pre

* prev = cur;
  cur = node;

120 90
~~110~~ ~~80~~ ~~30~~ ~~70~~

P = ~~A~~ ~~10~~ ~~20~~ ~~50~~ ~~60~~

C = ~~10~~ ~~20~~ ~~50~~ ~~60~~

~~80~~ ~~30~~ ~~70~~

~~110~~

~~90~~ ~~120~~

40

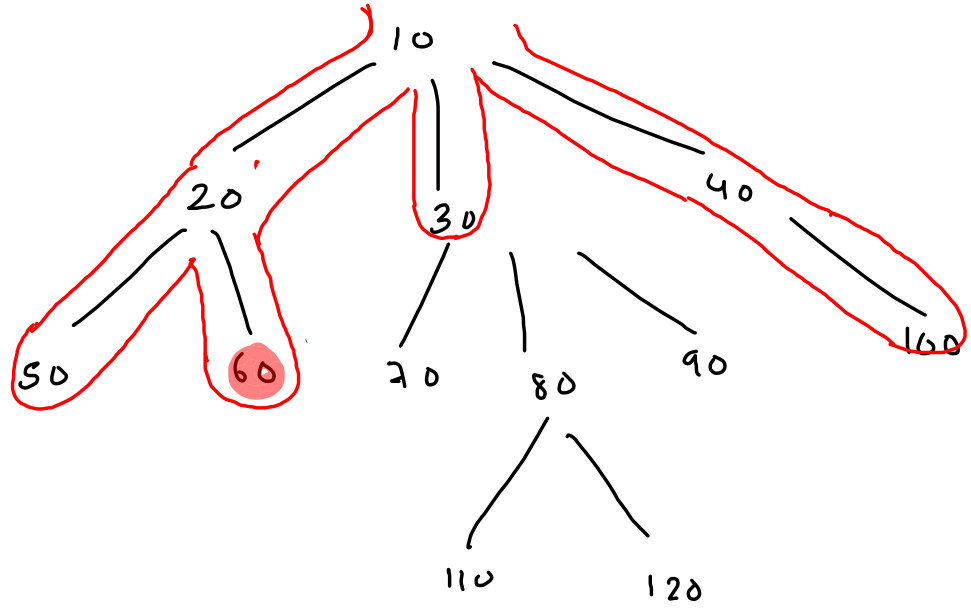
c.data = data {
  pred = p;
}

p.data = data {
  succ = c;
}

pred = ~~A~~ (120)

succ = ~~A~~ (40)

```java
public static void travel(Node node,int data) {
    prev = curr;
    curr = node;

    if(curr.data == data) {
        predecessor = prev;
    }

    if(prev != null && prev.data == data) {
        successor = curr;
        return;
    }

    for(Node child : node.children) {
        travel(child,data);
    }
}
```



pred → ~~n~~ 50

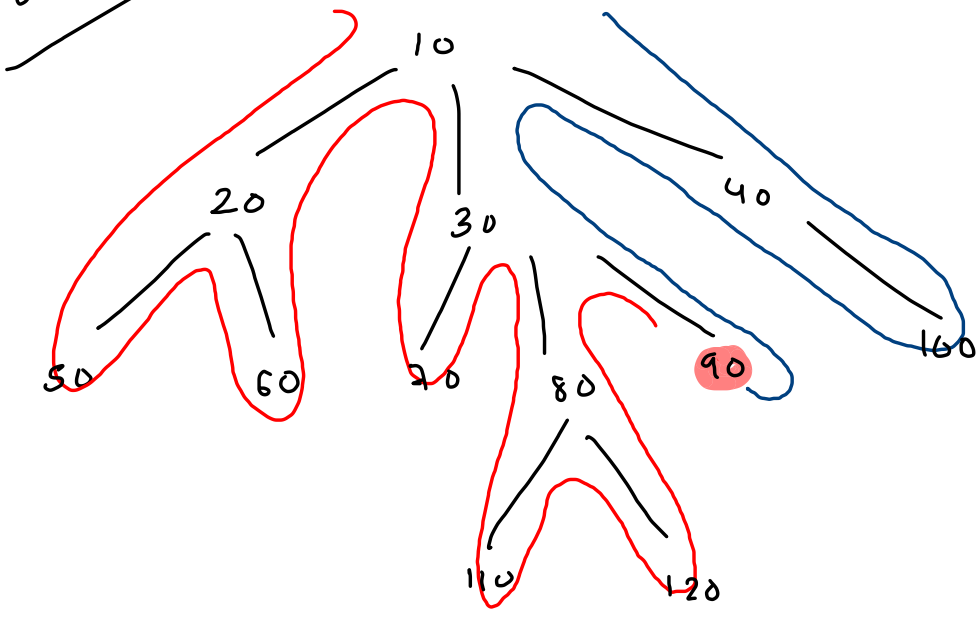
succ → ~~n~~ 30

prev → ~~n 10 20 50 60 50~~ 40

curr → ~~n 10 20 50 60 30~~ 40 100

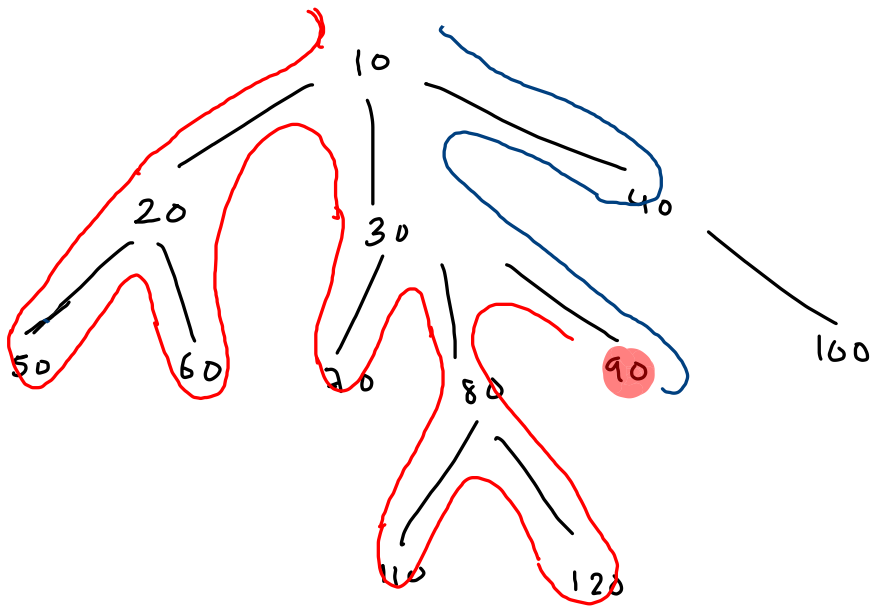
data = 60

predessor & successor



0 ——→ last node (pred)

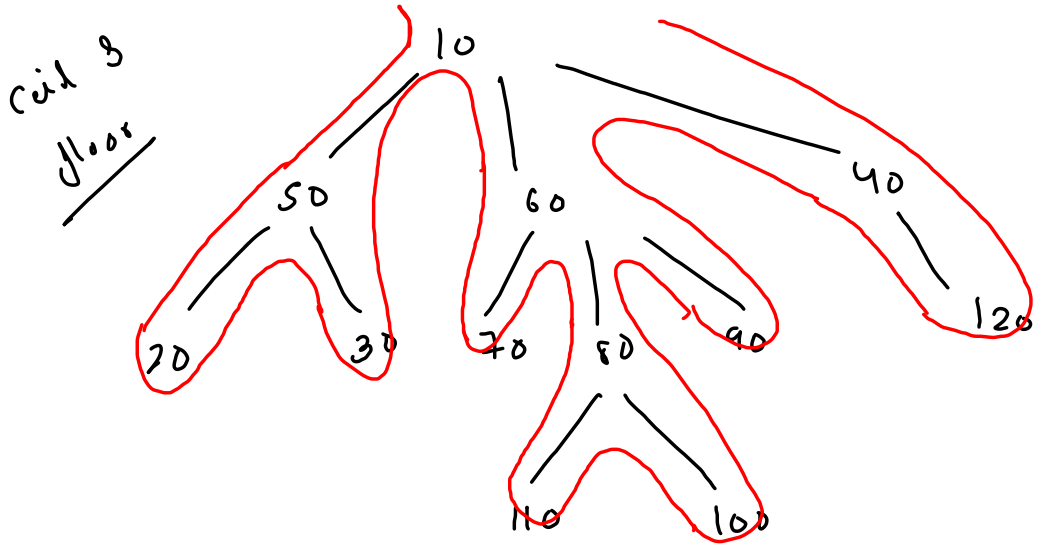
1 ——→ first node (succ)



State = ~~0~~ 1

pred = ~~10~~ ~~20~~ ~~30~~ ~~50~~ ~~60~~ ~~70~~ ~~80~~ ~~110~~ 120

succ = 40



ceil -> just larger / ∞ smallest among larger / qualified min

floor -> just smaller / -∞ largest among smaller / qualified max

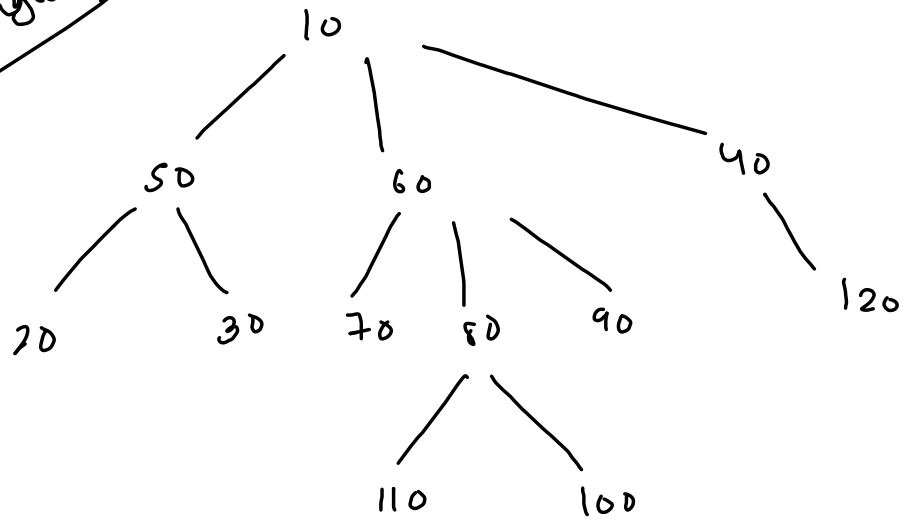
```java
public static void travel(Node node,int data) {
    //ceil -> qualified min
    if(node.data > data && ceil > node.data) {
        ceil = node.data;
    }

    //floor -> qualified max
    if(node.data < data && floor < node.data) {
        floor = node.data;
    }

    for(Node child : node.children) {
        travel(child,data);
    }
}
```

data = 37

ceil = ~~∞~~ ~~50~~ 40

floor = ~~-∞~~ ~~10~~ ~~20~~ 30

K^{th} largest



$\infty \xrightarrow{\text{floor}} \max \ (120)$

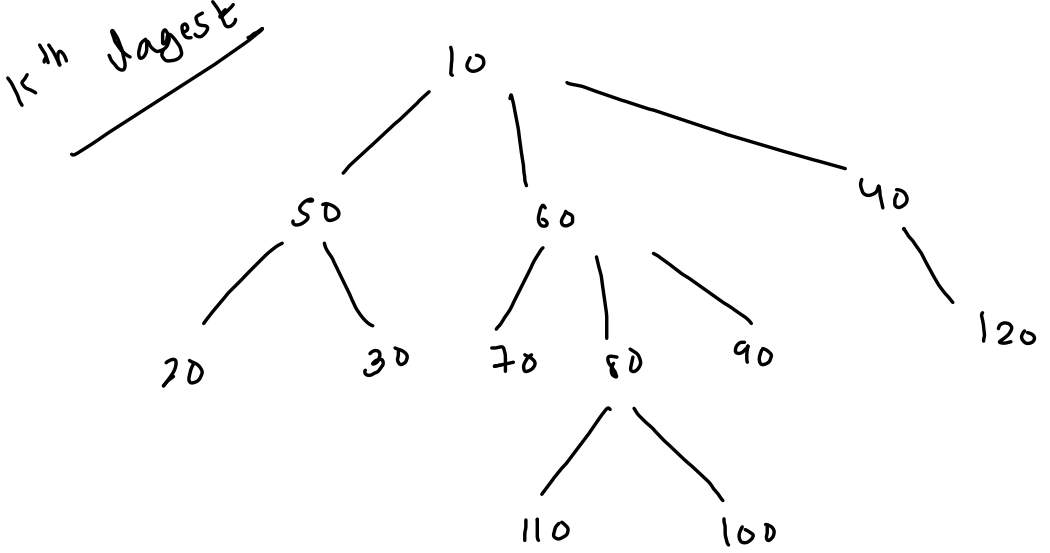
$\max \xrightarrow{\text{floor}} 2^{nd} \max \ (110)$

$2^{nd} \max \xrightarrow{\text{floor}} 3^{rd} \max \ (100)$

key = ∞ floor = 120

key = 120 floor = 110

key = 110 floor = 100



k = 4 , i = 0 → 1 → 2 → 3 → 4

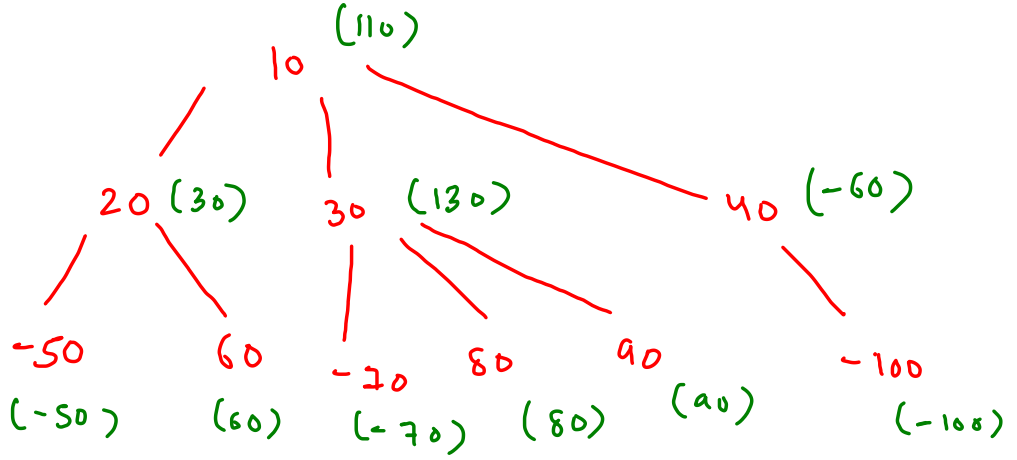
key = ∞ → 120 → 110 → 100

```java
public static int kthLargest(Node node, int k){
    int key = Integer.MAX_VALUE;

    for(int i=0; i < k;i++) {
        floor = Integer.MIN_VALUE;
        ceilAndFloor(node,key);
        key = floor;
    }

    return floor;
}
```

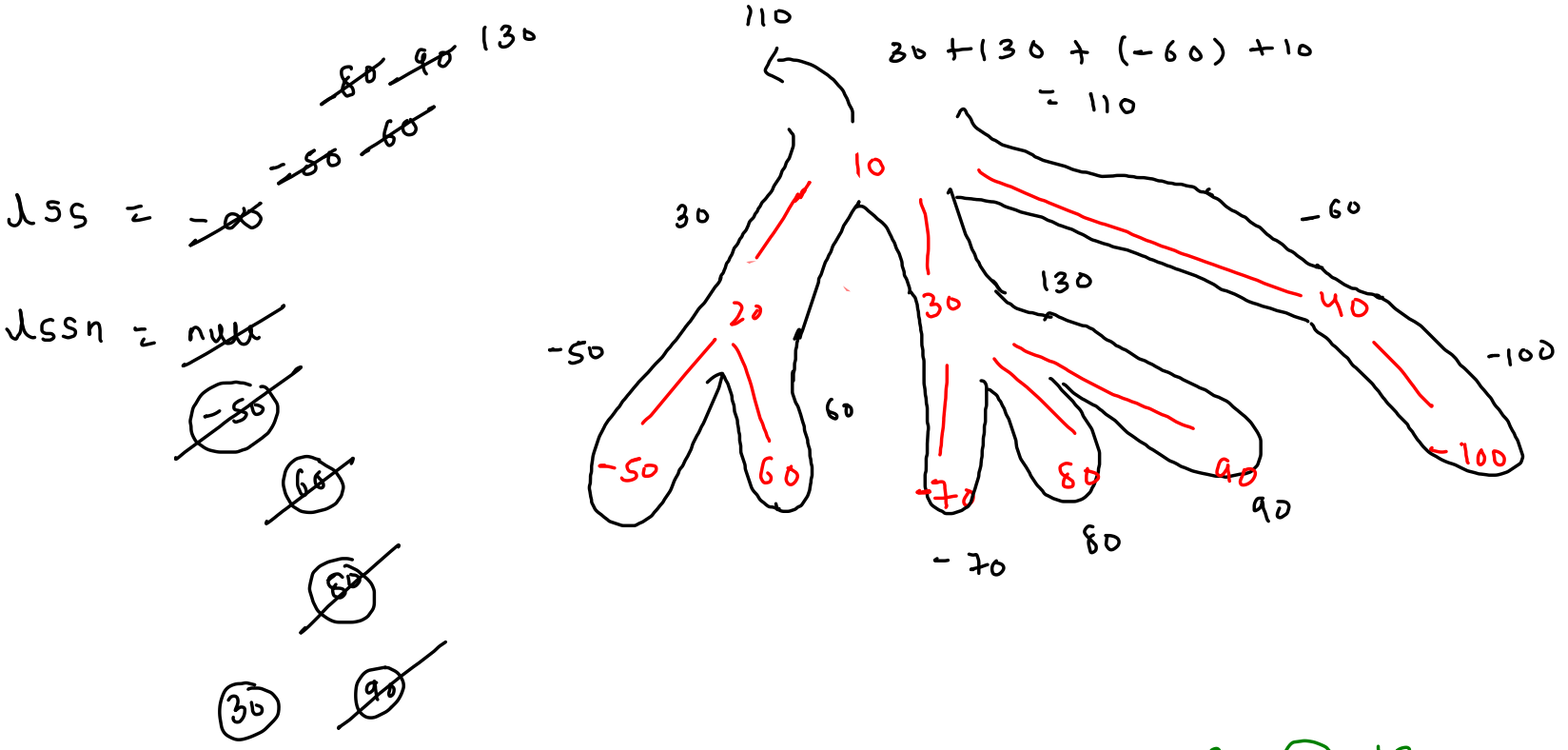
floor = −∞ → 120 → −∞ → 110 → −∞ → 100 → 90

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

node with largest subtree sum



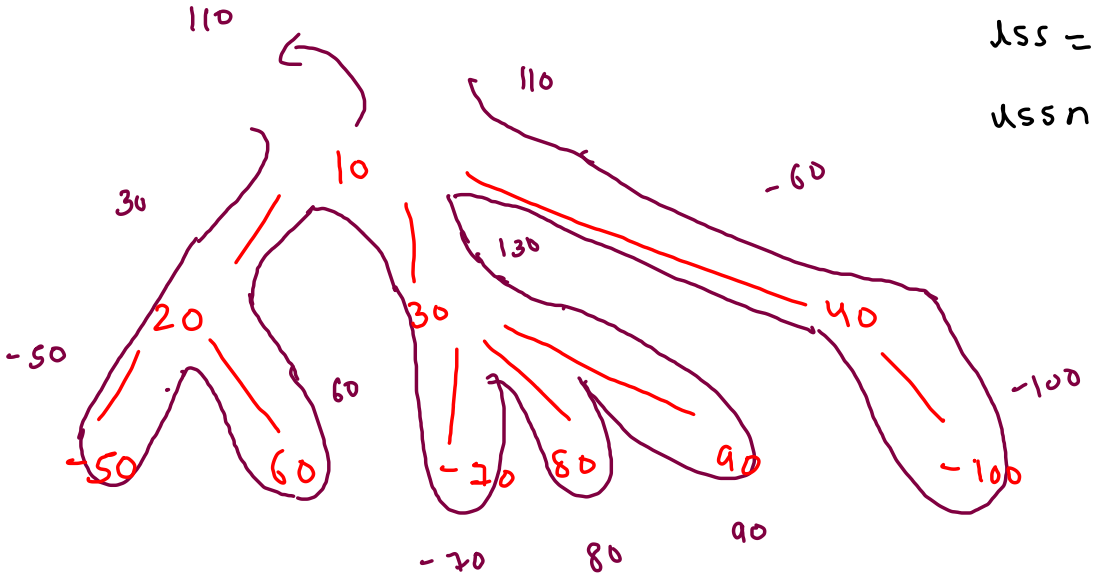
30 @ 130



lss = ~~80 90~~ 130 ~~-50 60~~ ~~-80~~

lssn = ~~null~~ ~~-50~~ ~~60~~ ~~80~~ 30 ~~90~~

30 @ 130



lss = ~~-∞~~ ~~-50~~ ~~60~~ ~~80~~ ~~90~~ 130

lssn = ~~null~~ ~~-50~~ ~~60~~ ~~60~~ ~~90~~ 30

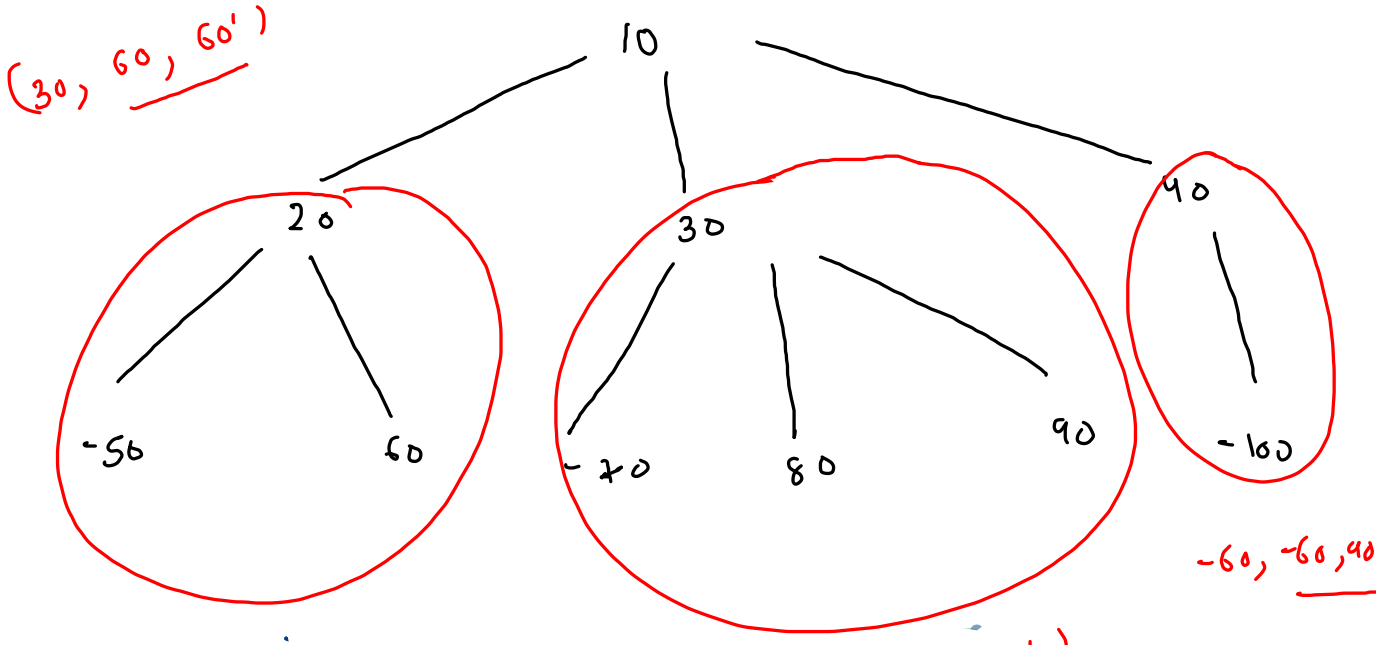
```
public static void largestSubtreeSum(Node node) {
    lss = Integer.MIN_VALUE;
    lssn = null;

    subtreeSum(node);

    System.out.println(lssn.data + "@" + lss);
}

public static int subtreeSum(Node node) {
    int ss = 0;

    for(Node child : node.children) {
        int css = subtreeSum(child);
        ss += css;
    }

    ss += node.data;

    if(ss > lss) {
        lss = ss;
        lssn = node;
    }

    return ss;
}
```

30 @ 130

```java
public static class Pair {
    int ss;
    int lss;
    Node lssn;

    Pair() {

    }

    Pair(int ss,int lss,Node lssn) {
        this.ss = ss;
        this.lss = lss;
        this.lssn = lssn;
    }
}
```

```java
public static Pair LSS(Node node) {

    int ss = 0;
    Pair bcp = new Pair(0,Integer.MIN_VALUE,null);

    for(Node child : node.children) {
        Pair cp = LSS(child);

        ss += cp.ss;

        if(cp.lss > bcp.lss) {
            bcp.lss = cp.lss;
            bcp.lssn = cp.lssn;
        }
    }

    ss += node.data;

    Pair np = new Pair();

    np.ss = ss;

    if(np.ss < bcp.lss) {
        np.lss = bcp.lss;
        np.lssn = bcp.lssn;
    }
    else {
        np.lss = ss;
        np.lssn = node;
    }

    return np;
}
```

ss = 30 + 130 + (-60) + 10



(30, 60, 60')

(130, 130, 30')

-60, -60, 40'

ss = 110

bcp.lss = 130

bcp.lssn = 30'

np.ss = 110

np.lss = 130

np.lssn = 30'