Generic Trees

Characteristic

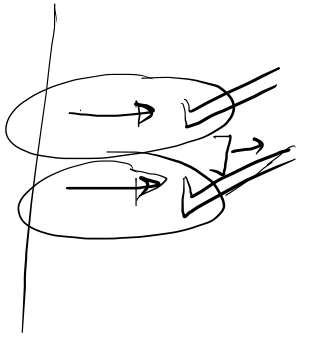
n children

Node

- int
- ArrayList<Node>

1. Input
2. Construct → returns root (Node)
3. Display → stack

ht == -1 (Base case)

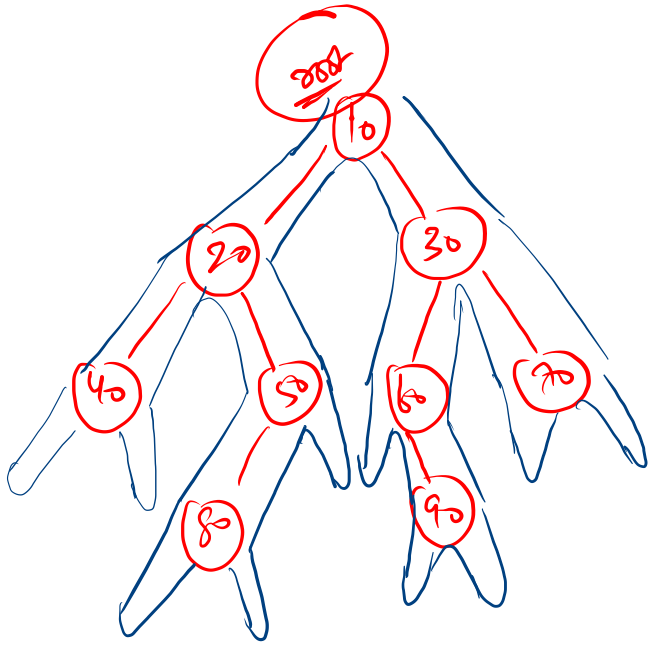


BINARY TREES :→ Concept is similar to ~~Binary~~ General Tree

+

Structural limit :- A node can have atmost 2 children

↳ 0, 1, 2



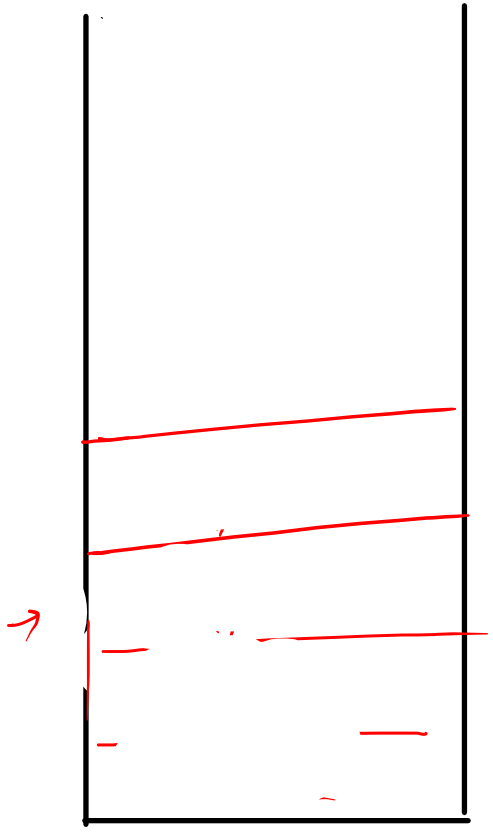
Node
- Data (int)
- left (Node)] ⇒ Default: null
- right (Node)

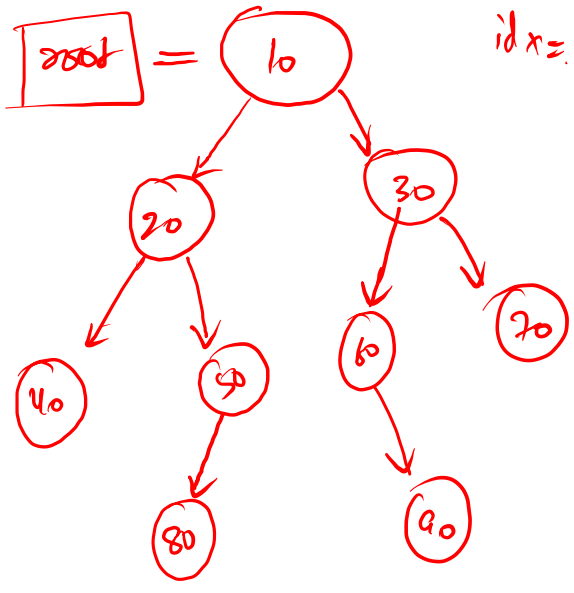
10 20 40 null null 50 80 null null null 30 60 null 90 null null 70 null null

{10,20,40,null,null,50,80,null,null,null,30,60,null,90,null,null,70,null,null};

Pair
↳ Node
↳ state

state
0 → left
1 → right
2 → pop

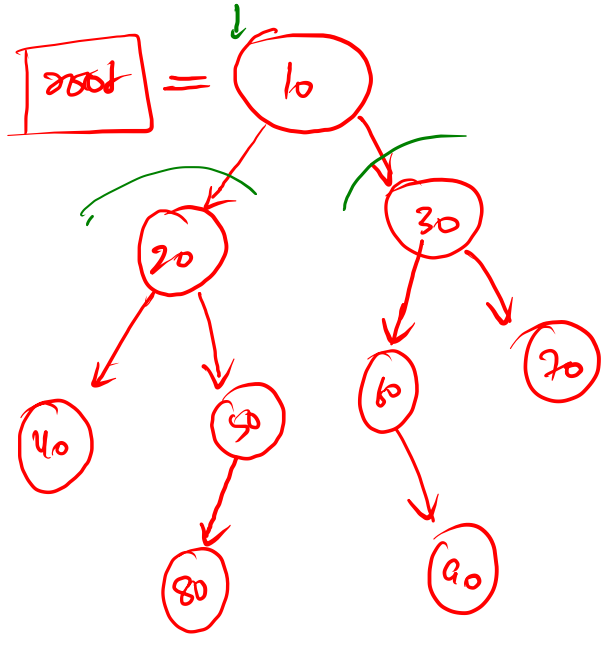


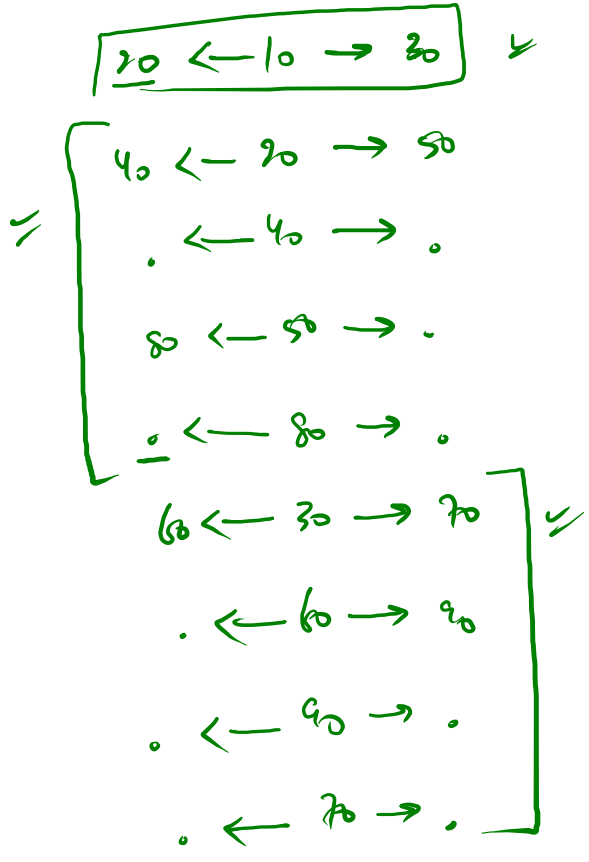


idx = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

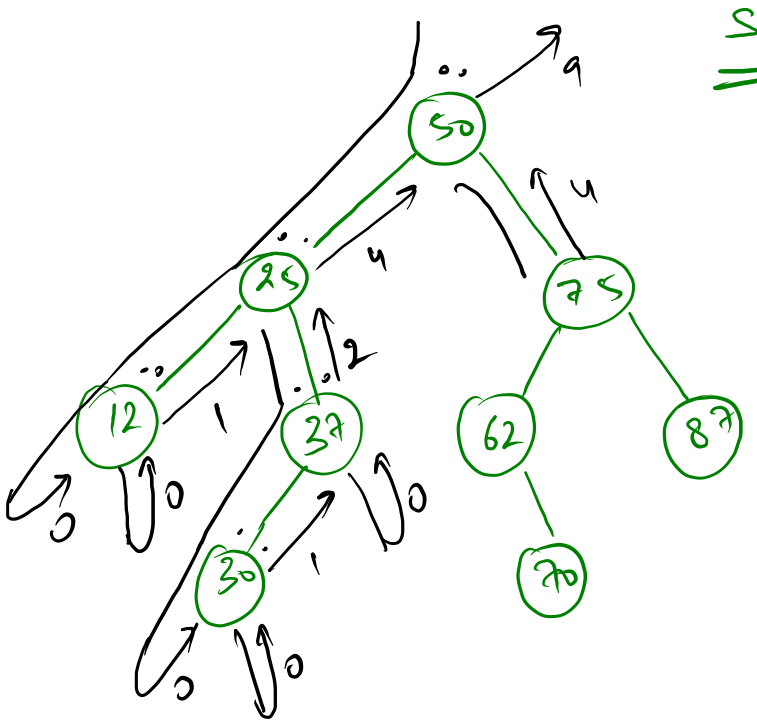
```
while(st.size() > 0){
    Pair top = st.peek();
    if(top.state == 0){
        Integer ele = inp[idx]; // null
        if(ele != null){
            Node newNode = new Node(ele,null,null);
            top.node.left = newNode;
            st.push(new Pair(newNode,0));
        }
        idx++;
        top.state++;
    }else if(top.state == 1){
        Integer ele = inp[idx]; // null
        if(ele != null){
            Node newNode = new Node(ele,null,null);
            top.node.right = newNode;
            st.push(new Pair(newNode,0));
        }
        idx++;
        top.state++;
    }else{
        st.pop();
    }
}
```

```
Integer inp[] = {10,20,40,null,null,50,80,null,null,null,30,60,null,90,null,null,70,null,null};
                0  1  2  3    4    5  6  7    8    9    10 11 12   13 14   15   16 17   18
```






```
public static void display (Node node){
    String str = "";
    str += node.left != null ?node.left.data : ".";
    str += " <- " + node.data + " -> ";
    str += node.right != null;node.right.data :".";
    print (str);
    display(node.left);
    display(node.right);
}
```

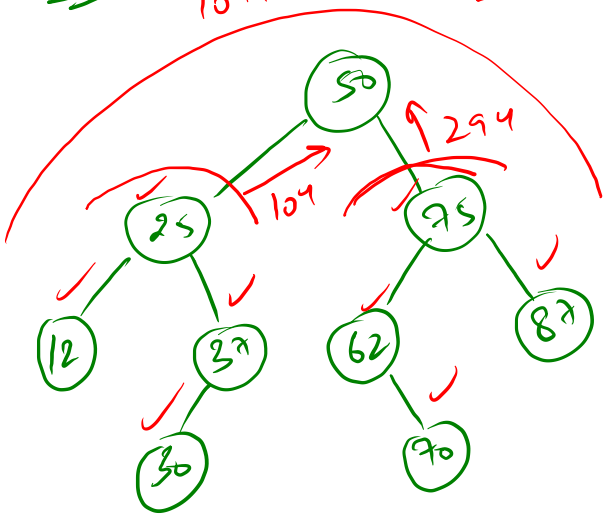



```
if(node == null){
    return 0;
}
```

```java
public static int size(Node node) {
  int sizeOfLTree = size(node.left); -1
  int sizeOfRTree = size(node.right); -2

  return sizeOfLTree+sizeOfRTree+1;
}
```

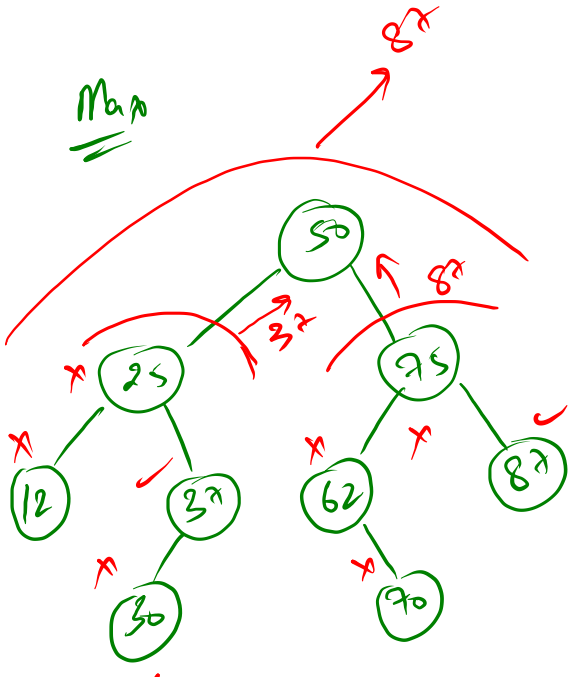
Sum

104+294+50 = 448



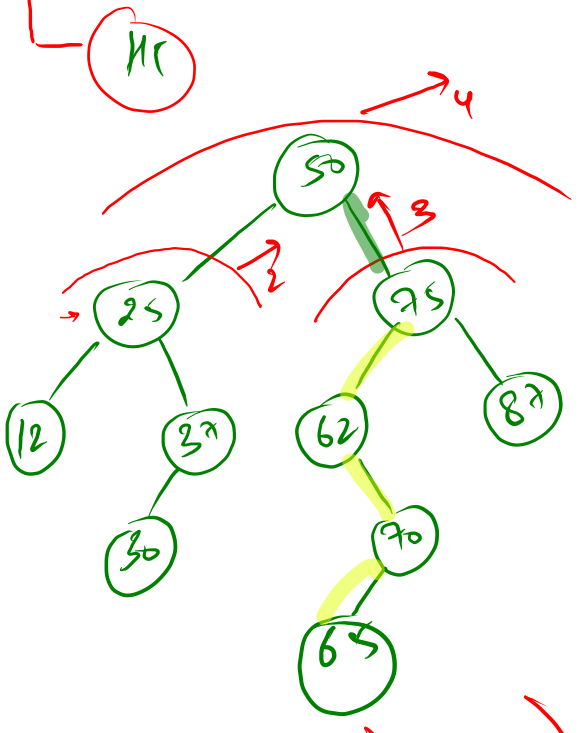
LSum + RSum + node.data

Max



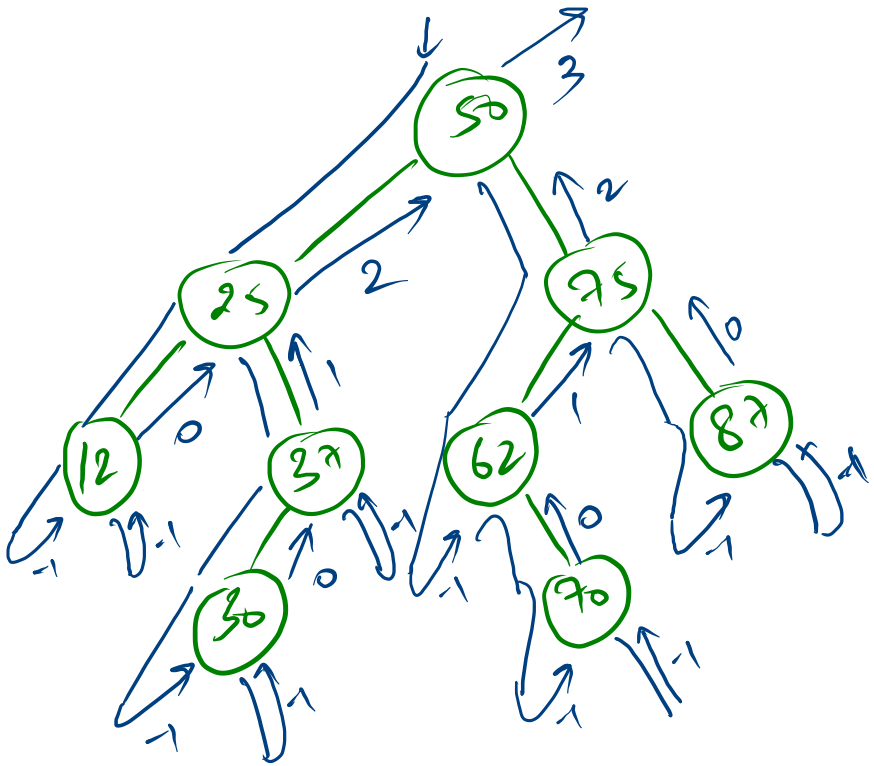
Max(LMax, RMax, node.data)

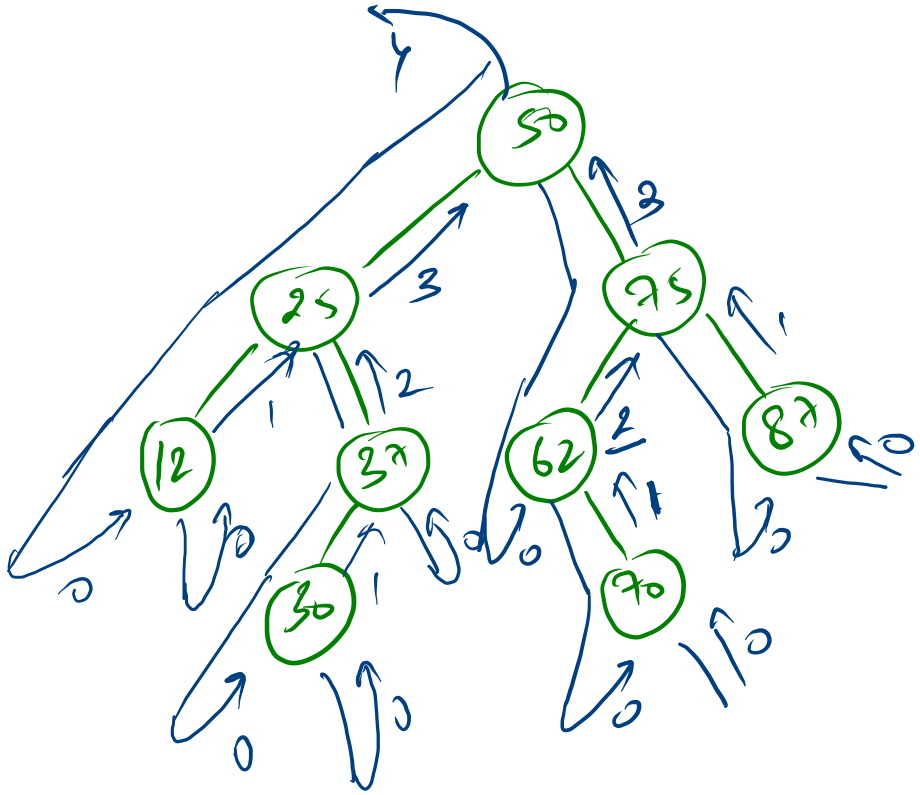
Distance b/w root & it's deepest node



(Max(LHt, RHt) + 1)

Height => Distance b/w root & its deepest node.

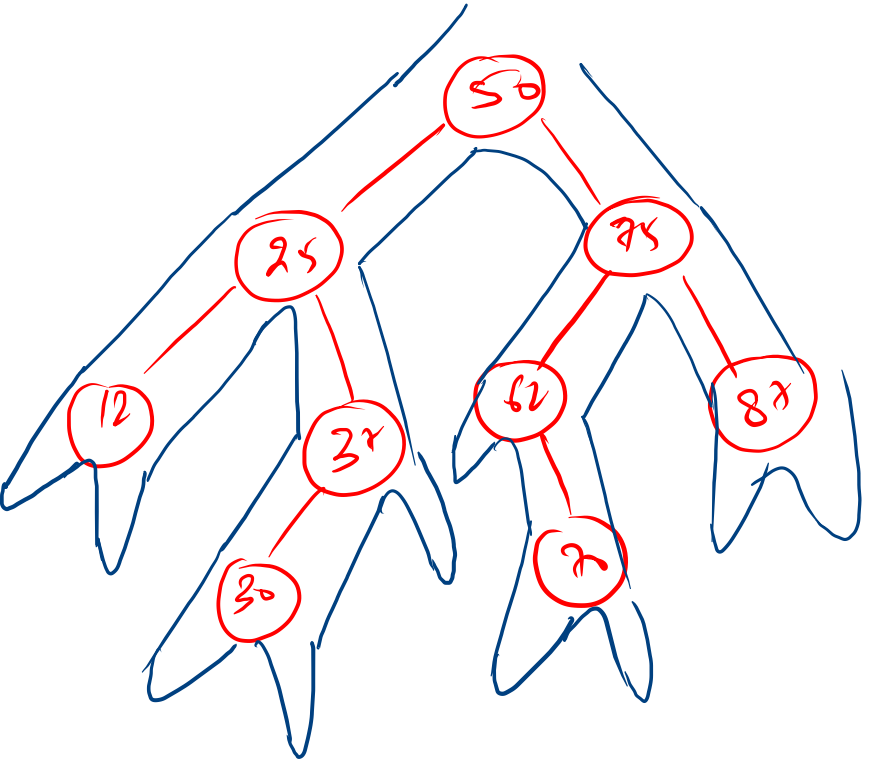





```
public static int height(Node node) {
  if(node == null){

  }
  int lht = height(node.left);
  int rht = height(node.right);
  return Math.max(lht,rht)+1;
}
```

-1 => Height
↳ on the basis of edges

0 => Height
↳ on the basis of nodes

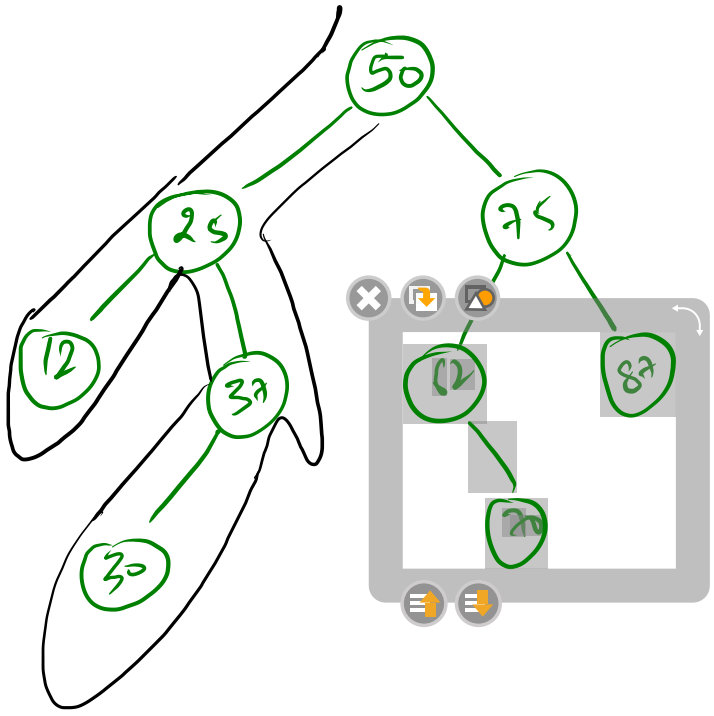


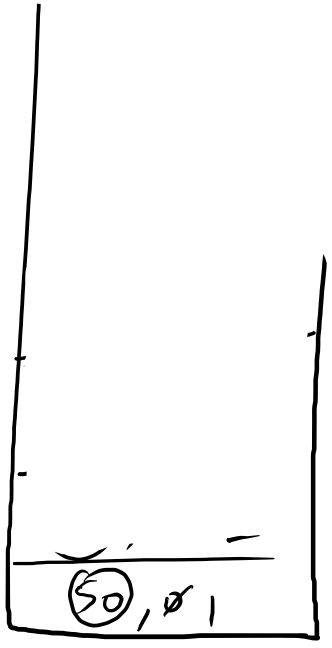
- Pre → first [N L R]
- In → second [L N R]
- Post → last [L R N]

Pre = 50 → 25 → 12 → 37 → 30 → 75 → 62 → 70 → 87 →.

In = 12 → 25 → 30 → 37 → 50 → 62 → 70 → 75 → 87 →.

Post = 12 → 30 → 37 → 25 → 70 → 62 → 87 → 75 → 50 →.





Stack

0 → px, left add

1 → in, right add

2 → post, pop

px 50 25 12 37 30

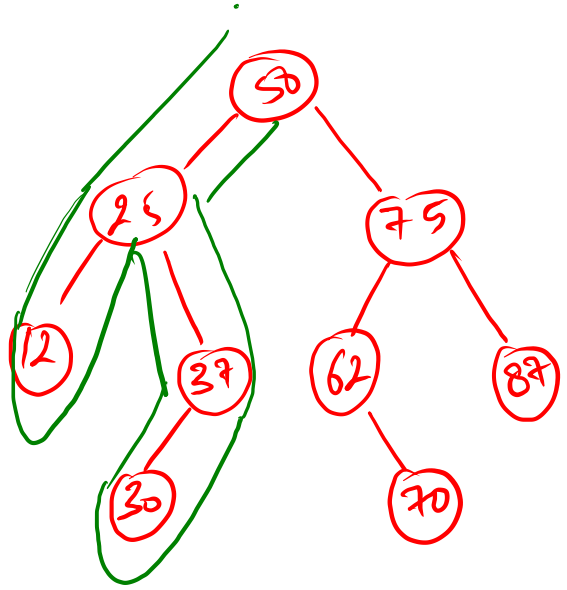
in 12 25 30 37

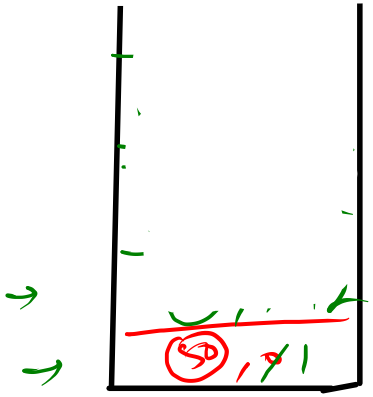
post 12 30 37 25

pre = 50 25 12 37 30

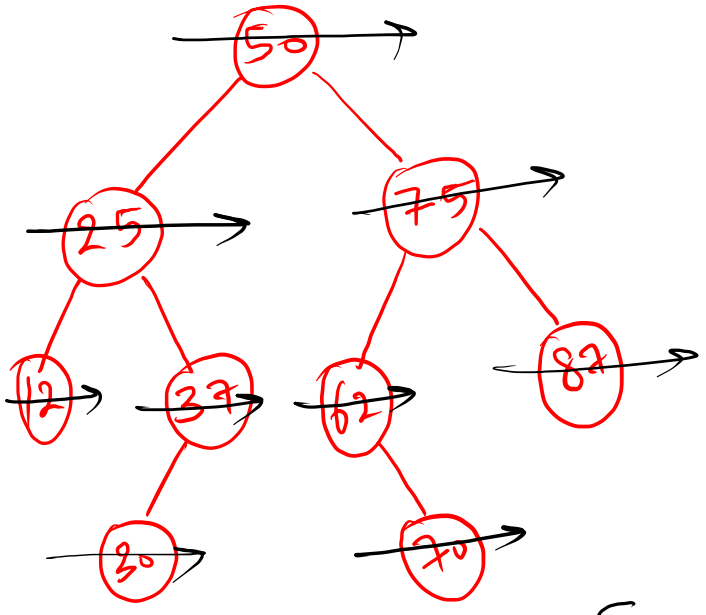
in = "12 25 30 37"

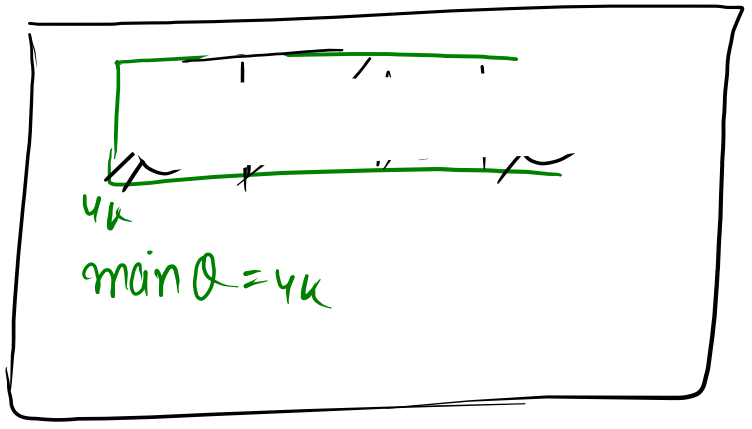
post = "12 30 37 25"

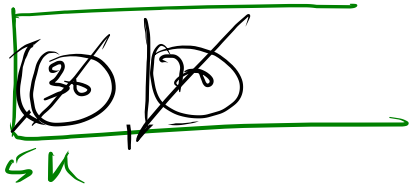


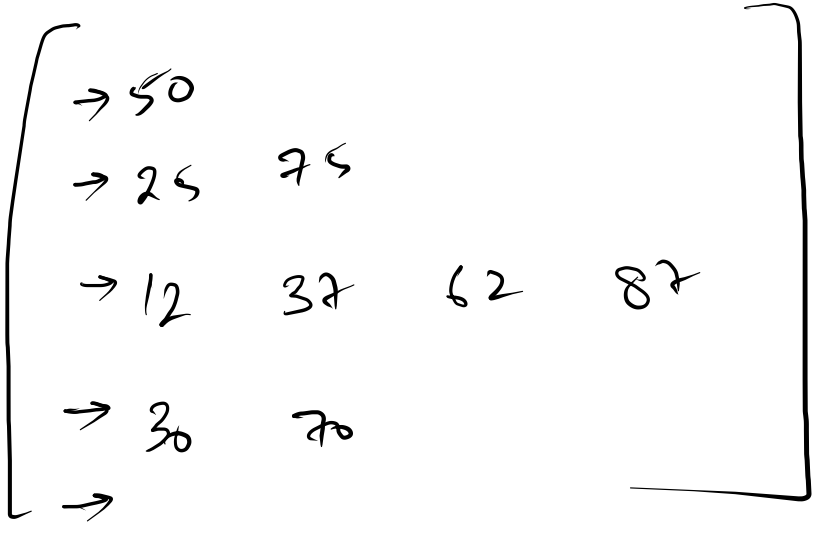



```java
Stack<Pair> st = new Stack<>();
st.push(new Pair(node,0));
String pre = "" , post = "" , in = "";
while(st.size() > 0){
    Pair top = st.peek();

    if(top.state == 0){
        pre += top.node.data+" ";
        if(top.node.left != null){
            st.push(new Pair(top.node.left,0));
        }
        top.state++;
    }else if(top.state == 1){
        in += top.node.data+" ";
        if(top.node.right != null){
            st.push(new Pair(top.node.right,0));
        }
        top.state++;
    }else if(top.state == 2){
        post += top.node.data+" ";
        st.pop();
    }
}
System.out.println(pre+"\n"+in+"\n"+post);
```

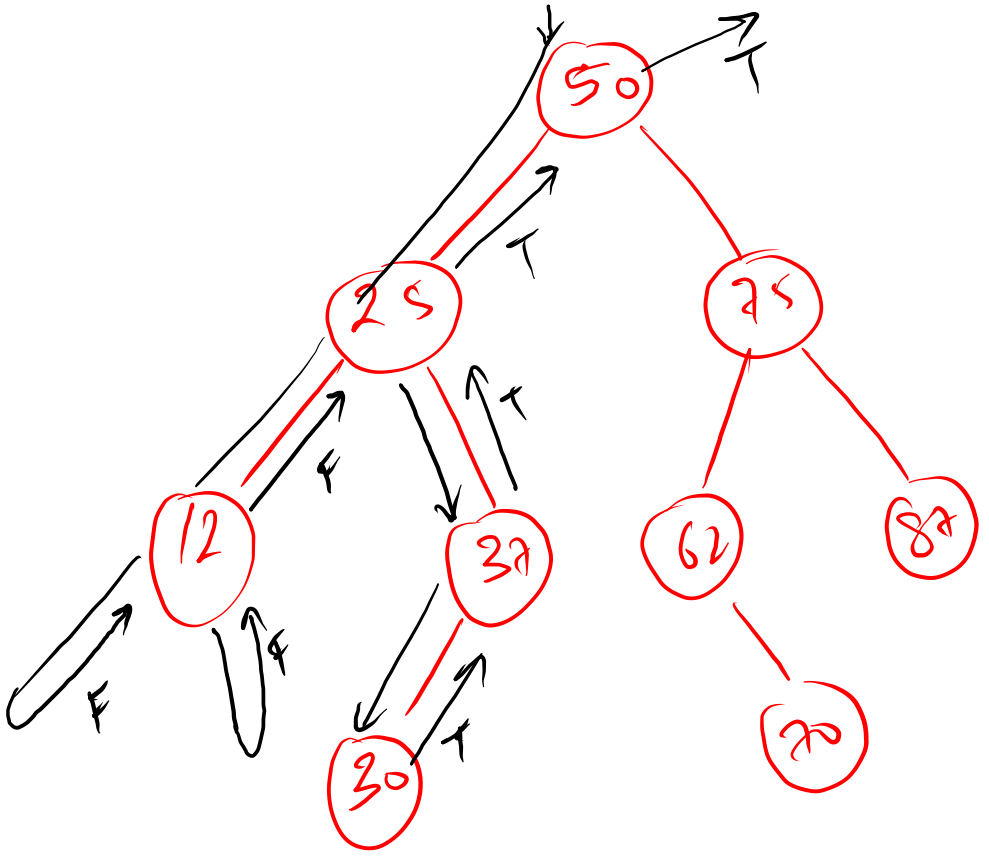

50
25
75
12
37
62
87
30
70


4K
main() = 4K


30
70
5K
helper() = 5K


50
25
75
12
37
62
87
30
70

data = 30

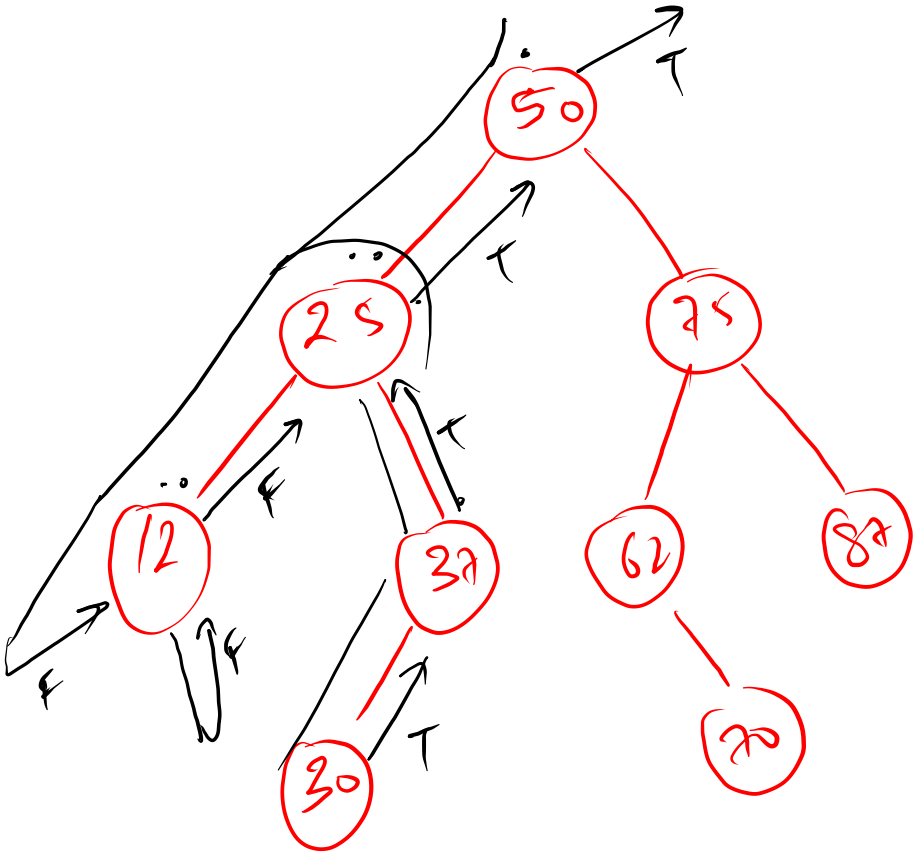


19

50 25 12 n n 37 30 n n n 75 62 n 70 n n 87 n n

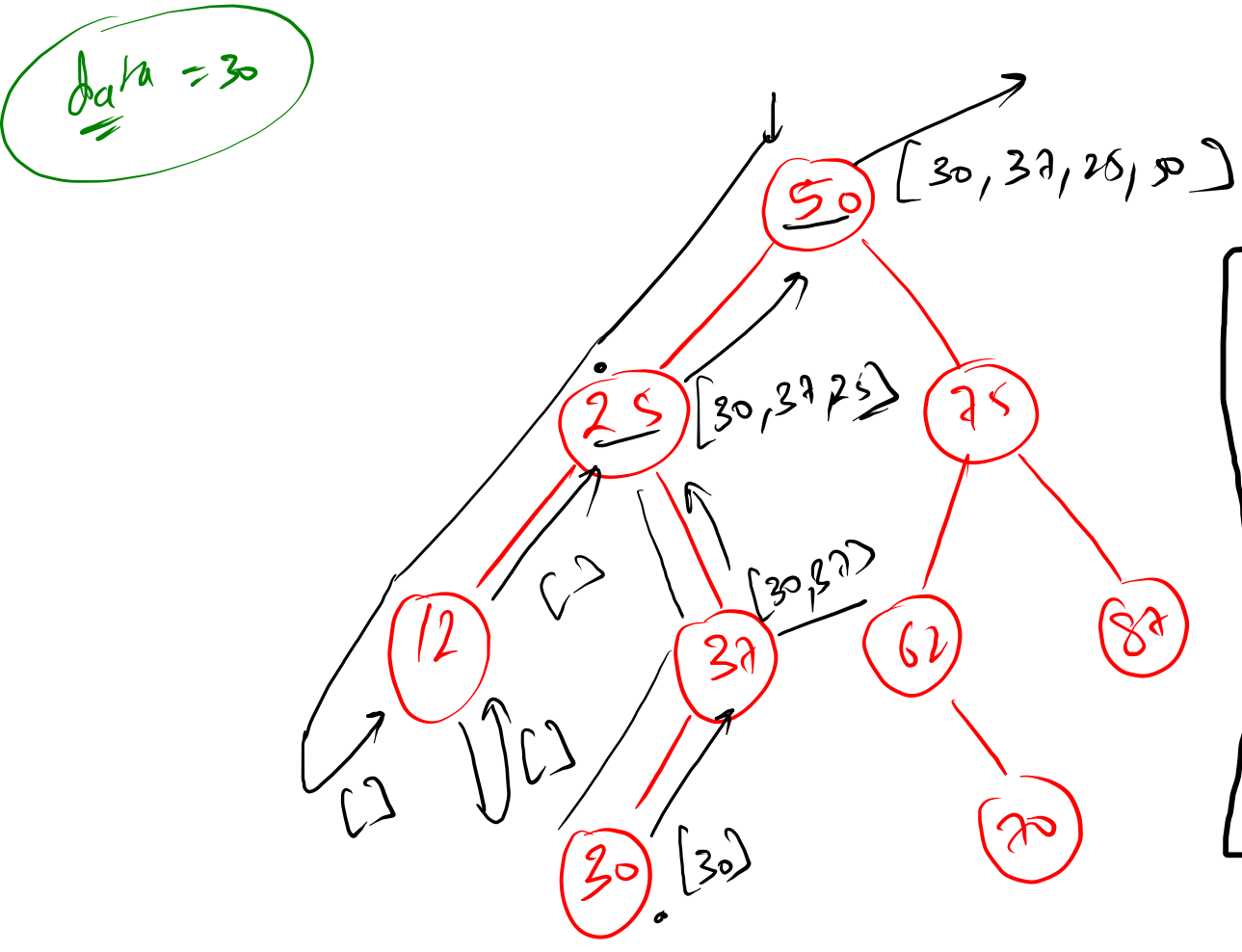
30

data = 30



30

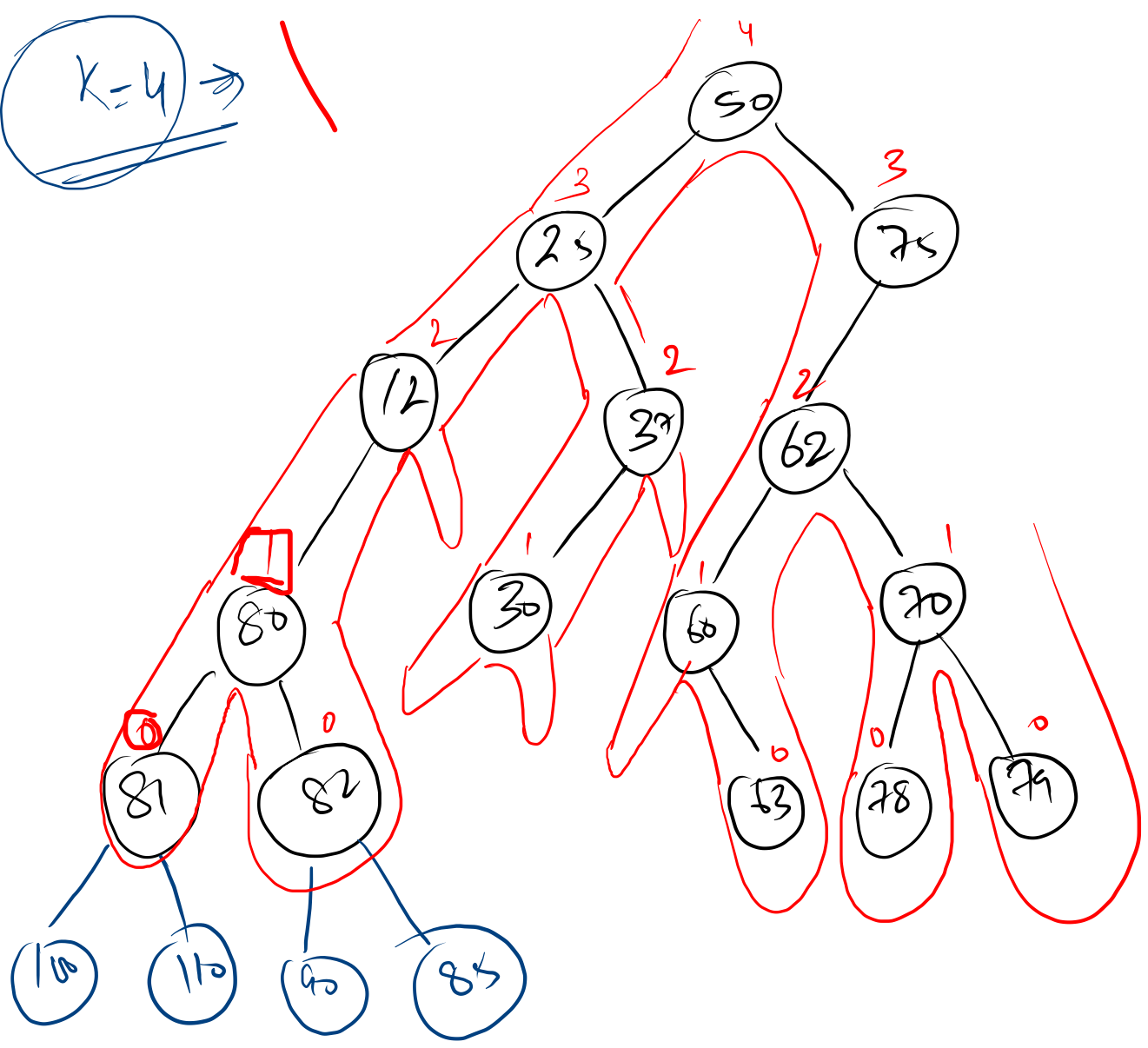
```java
public static boolean find(Node node, int data){
  if(node == null){
      return false;
  }
  if(node.data == data){
      return true;
  }

  boolean lres = find(node.left,data);
  if(lres){
      return true;
  }
  boolean rres = find(node.right,data);
  if(rres){
      return true;
  }
  return false;
}
```




```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
    if(node == null){
        return new ArrayList<>();
    }
    if(node.data == data){
        ArrayList<Integer> al = new ArrayList<>();
        al.add(data);
        return al;
    }

    ArrayList<Integer> lres = nodeToRootPath(node.left,data);
    if(lres.size() > 0){
        lres.add(node.data);
        return lres;
    }
    ArrayList<Integer> rres = nodeToRootPath(node.right,data);
    if(rres.size() > 0){
        rres.add(node.data);
        return rres;
    }
    return new ArrayList<>();
}
```

```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
  if(node == null){
      return new ArrayList<>();
  }
  if(node.data == data){
      ArrayList<Integer> al = new ArrayList<>();
      al.add(data);
      return al;
  }

  ArrayList<Integer> lres = nodeToRootPath(node.left,data);
  if(lres.size() > 0){
      lres.add(node.data);
      return lres;
  }
  ArrayList<Integer> rres = nodeToRootPath(node.right,data);
  if(rres.size() > 0){
      rres.add(node.data);
      return rres;
  }
  return new ArrayList<>();
}
```

```java
public static boolean find(Node node, int data){
  if(node == null){
      return false;
  }
  if(node.data == data){
      return true;
  }

  boolean lres = find(node.left,data);
  if(lres){
      return true;
  }
  boolean rres = find(node.right,data);
  if(rres){
      return true;
  }
  return false;
}
```



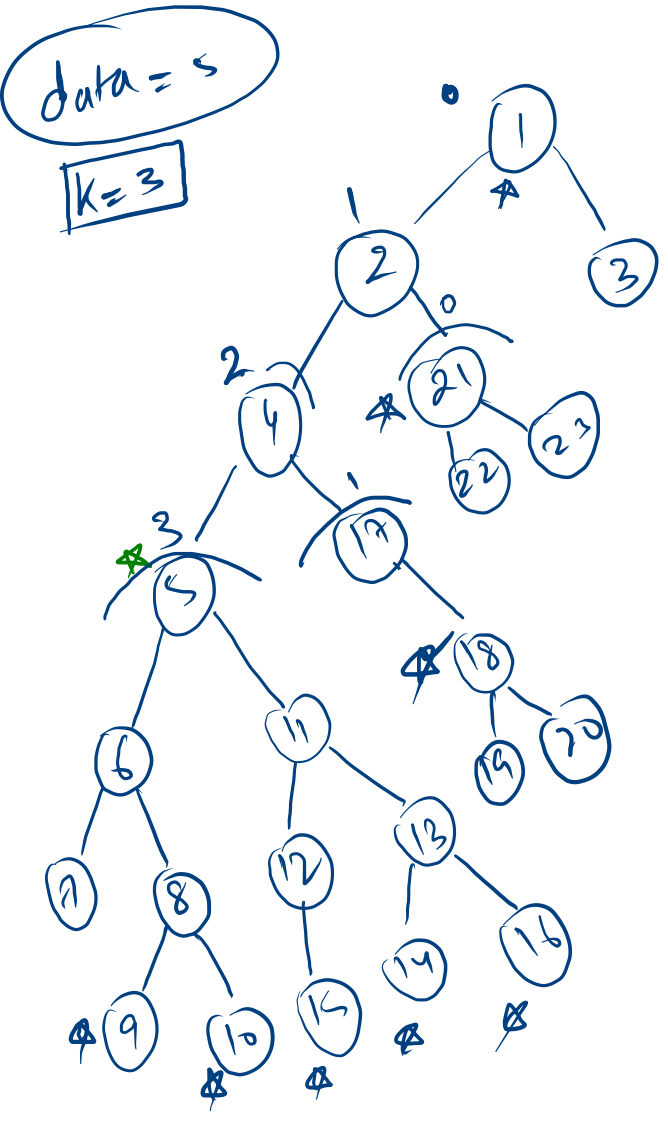
81

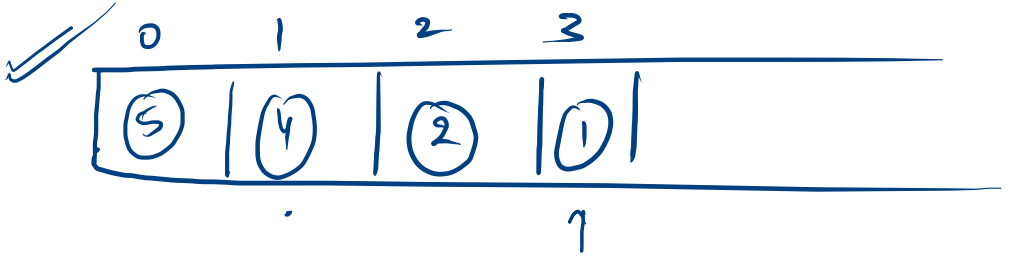
82

63

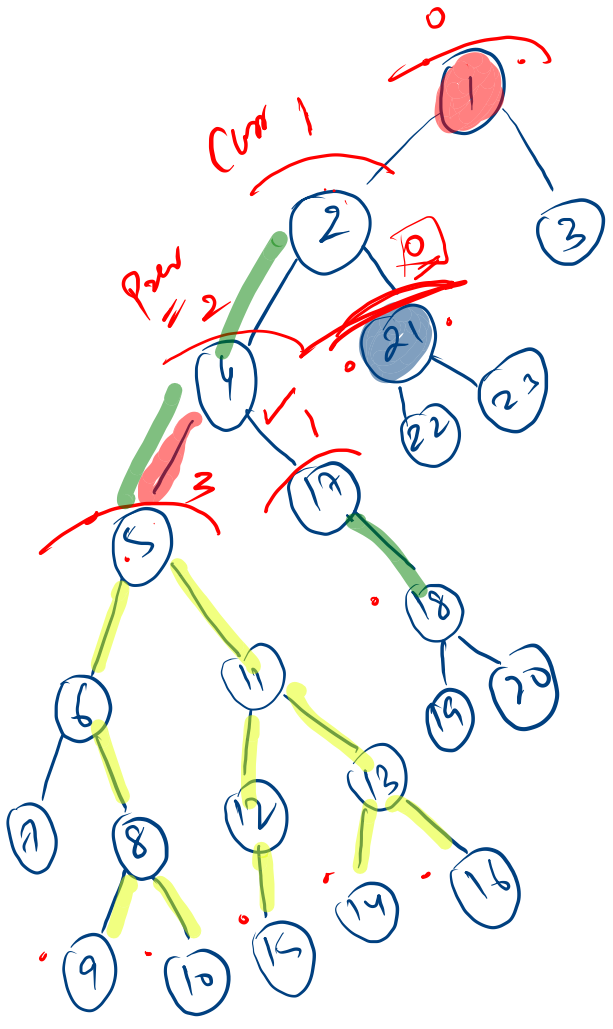
78

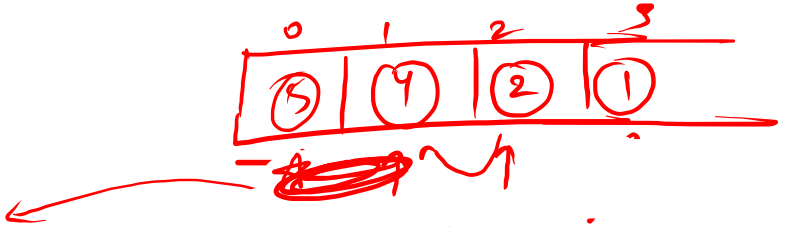
79





remDist ⇒ K - idx





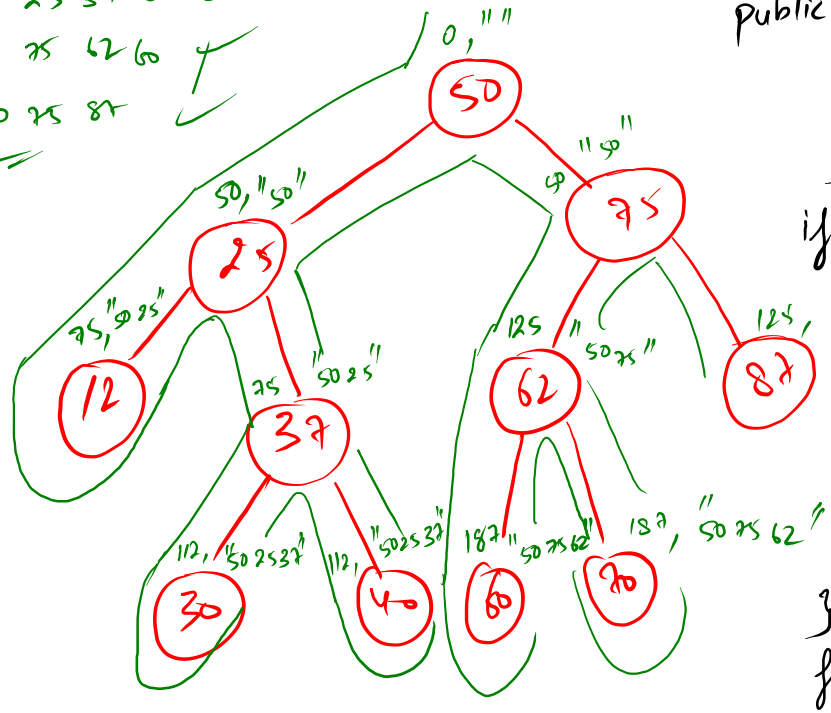
remDist = 0 d=5 , k=3

9
10
15
14
16
18
21
1

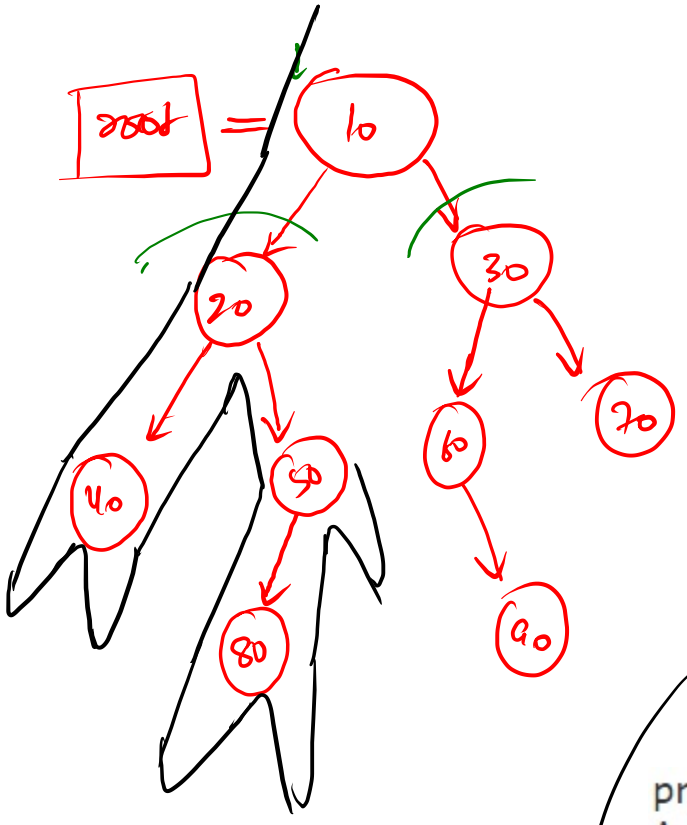
```
ArrayList<Node> list = nodeToRootPath(node,data);

printKLevelsDown(list.get(0),k);

for(int idx = 1 ; idx < list.size() ; idx++){
    int remDist = k - idx;
    Node curr = list.get(idx);
    Node prev = list.get(idx-1);
    if(remDist == 0){
        System.out.println(curr.data);
        break;
    }else{
        if(curr.left == prev){
            printKLevelsDown(curr.right , remDist-1);
        }else if(curr.right == prev){
            printKLevelsDown(curr.left , remDist-1);
        }
    }
}
```

lo = 150
hi = 250

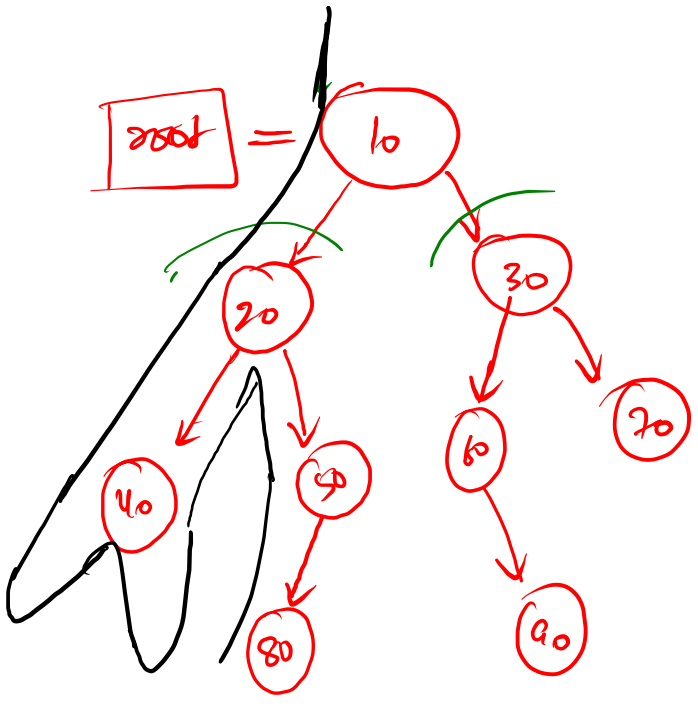
50 25 37 40
50 75 62 60
50 75 87




```
public static void fun(Node node, int sum, String path, int lo, int hi ){
    if(node == null){
      return;
    }
  if(node.left == null && node.right == null){
     sum += node.data;
     path += node.data;
     if(sum >= lo && sum <= hi){
        print(path);
     }
     return;
  }
  fun(node.left, sum + node.data, path + node.data + " ", lo, hi);
  fun(node.right, sum + node.data, path + node.data + " ", lo, hi);
}
```



pre : 10 -> 20 -> 40 -> 50 -> 80 -> 30 -> 60 -> 90 -> 70 -> .
in :40 -> 20 -> 80 -> 50 -> 10 -> 60 -> 90 -> 30 -> 70 -> .
post :40 -> 80 -> 50 -> 20 -> 90 -> 60 -> 70 -> 30 -> 10 -> .




```
if (node == null) {
    return;
}
```

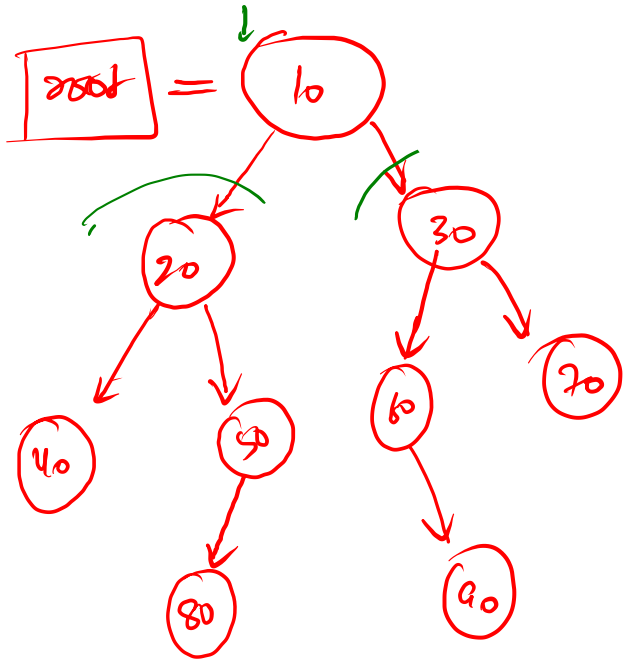
```java
public static void display(Node node){
    String str = "";
    str += node.left != null ? node.left.data : ".";
    str += " <- "+node.data+" -> ";
    str += node.right != null ? node.right.data : ".";
    System.out.println(str);
    display(node.left);
    display(node.right);
}
```

20 ← 10 → 30

40 ← 20 → 50

. ← 40 → .

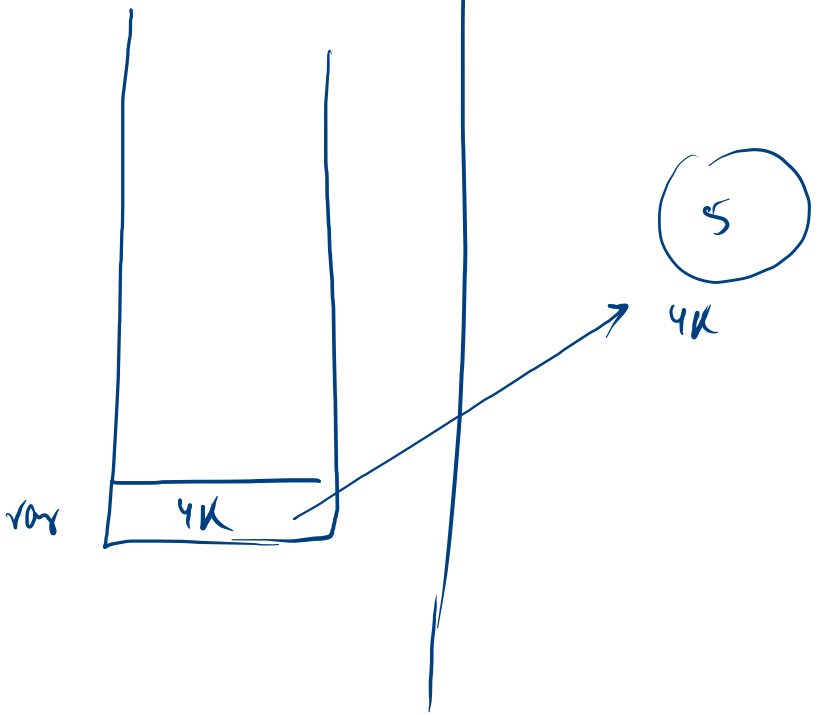
80 ← 50 → .




```
20 <- 10 -> 30
40 <- 20 -> 50
. <- 40 -> .
80 <- 50 -> .
. <- 80 -> .
60 <- 30 -> 70
. <- 60 -> 90
. <- 90 -> .
. <- 70 -> .
```

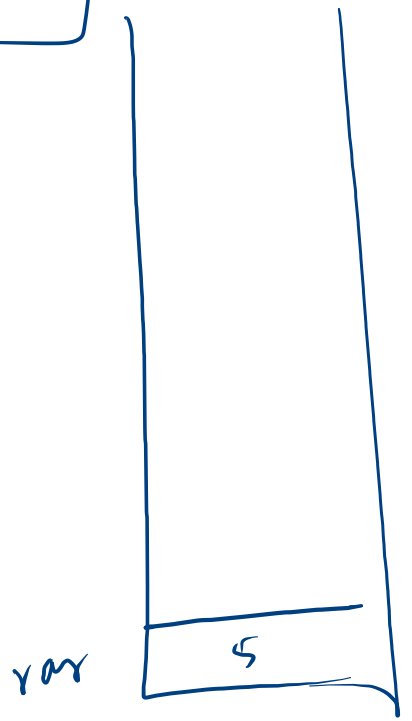
Integer
↳ wrapper Class of int

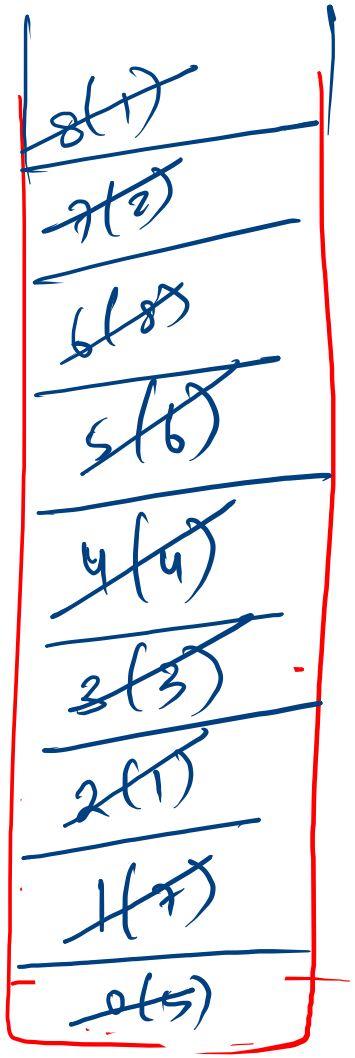
Integer var = 5;

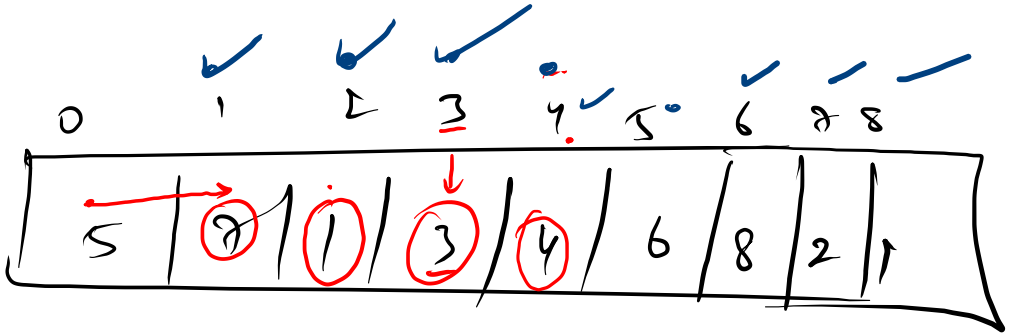


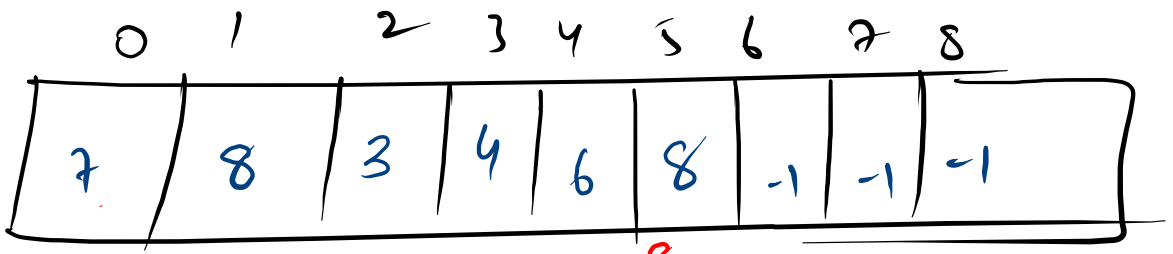
int (primitive)

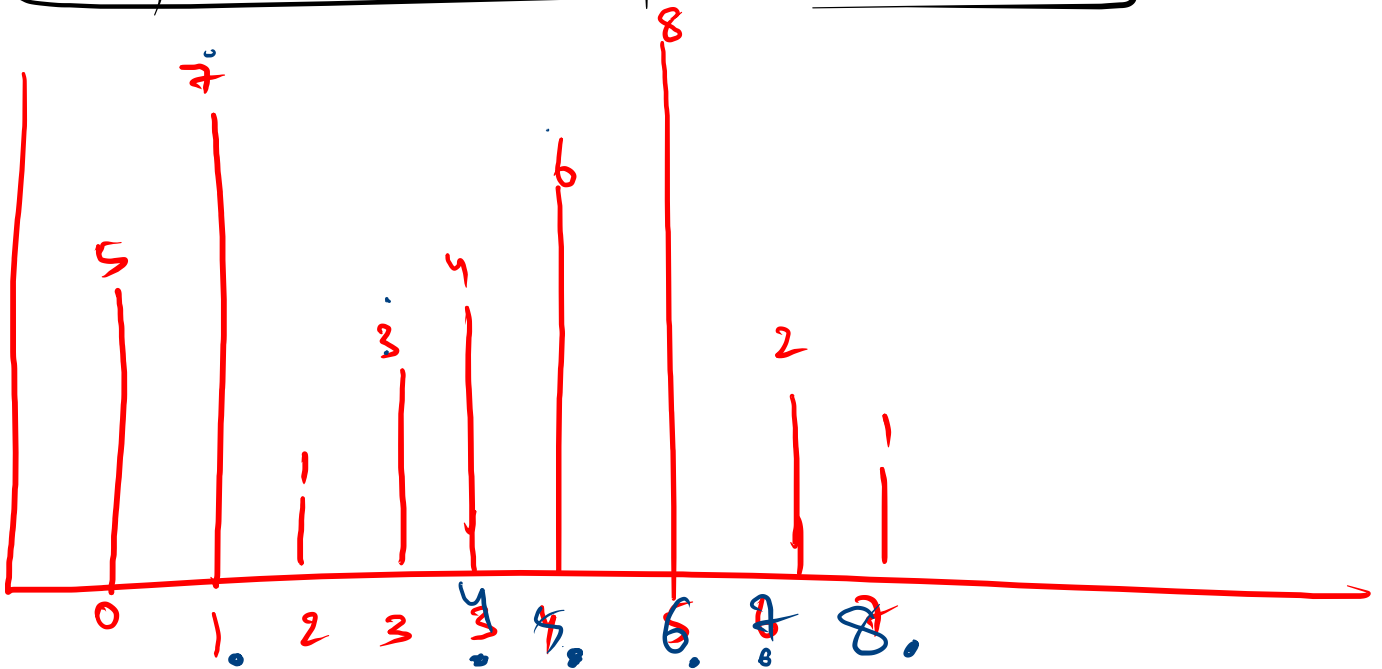
int var = 5






0 1 2 3 4 5 6 7 8
5 7 1 3 4 6 8 2 1


0 1 2 3 4 5 6 7 8
7 8 3 4 6 8 -1 -1 -1


0 1 2 3 4 5 6 7 8