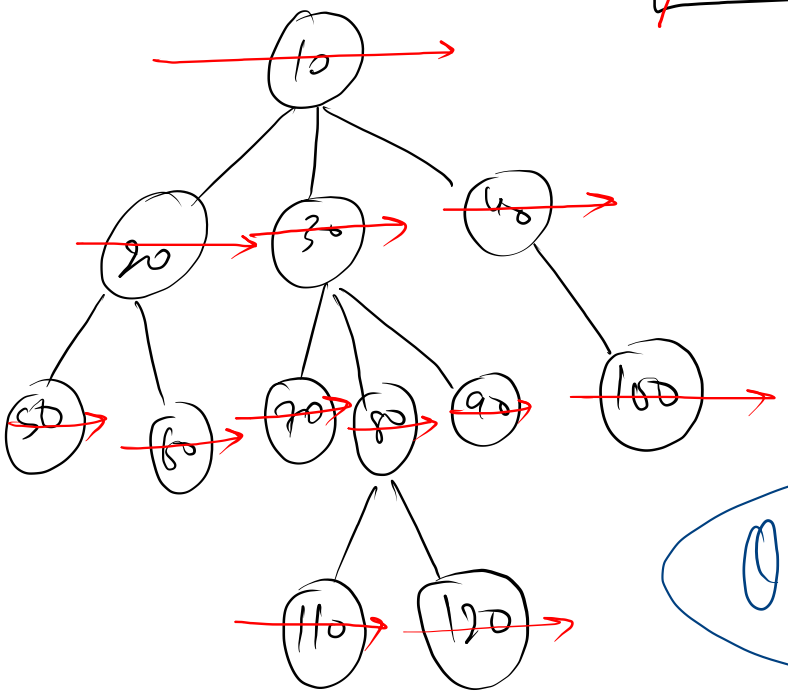
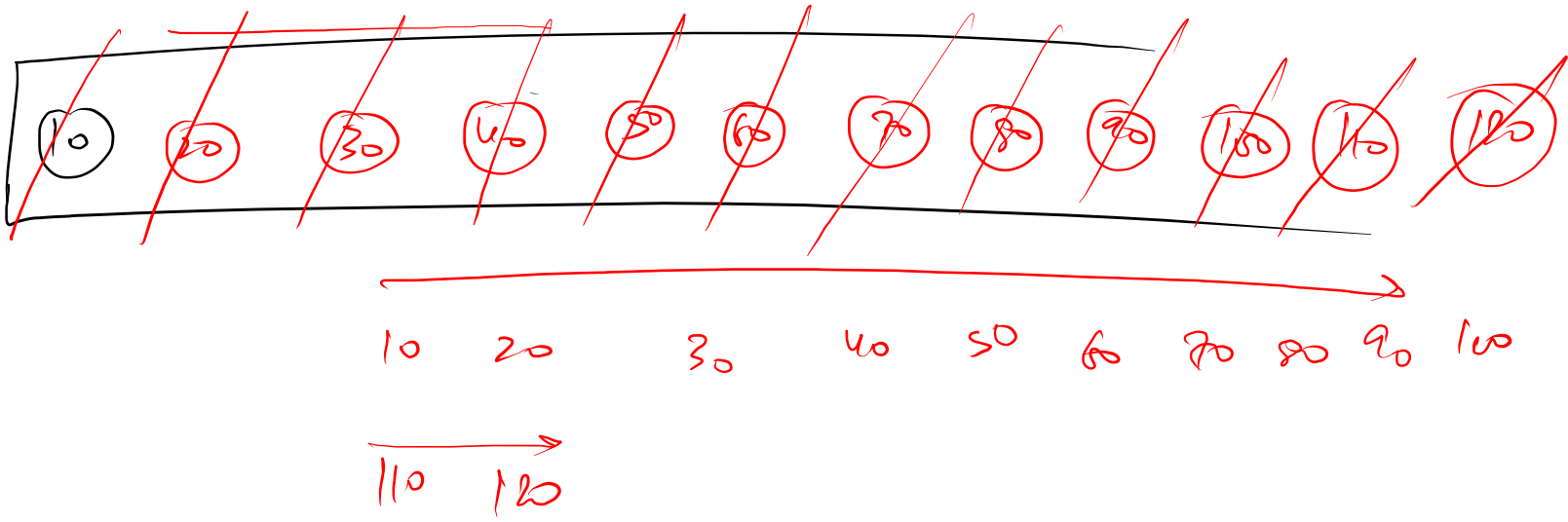
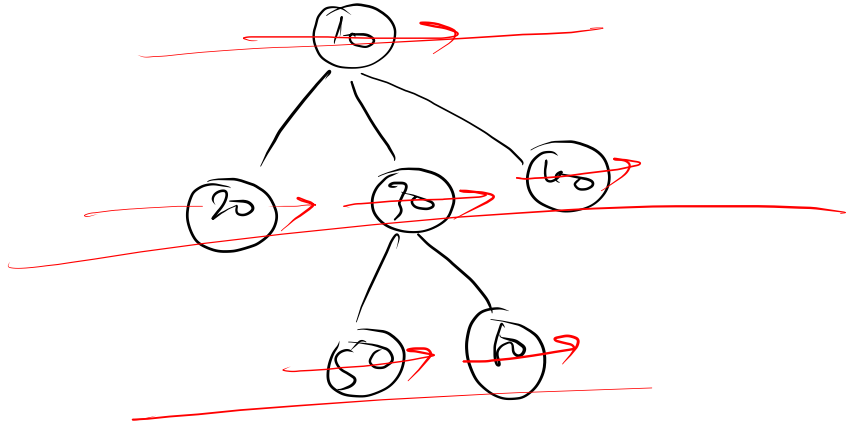
Level ORDER

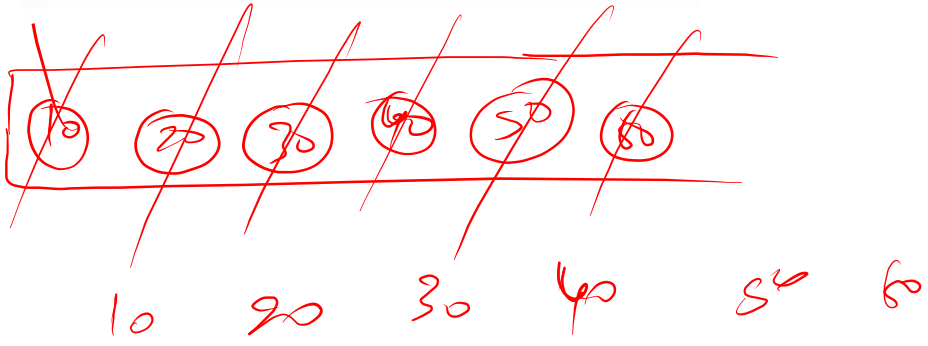




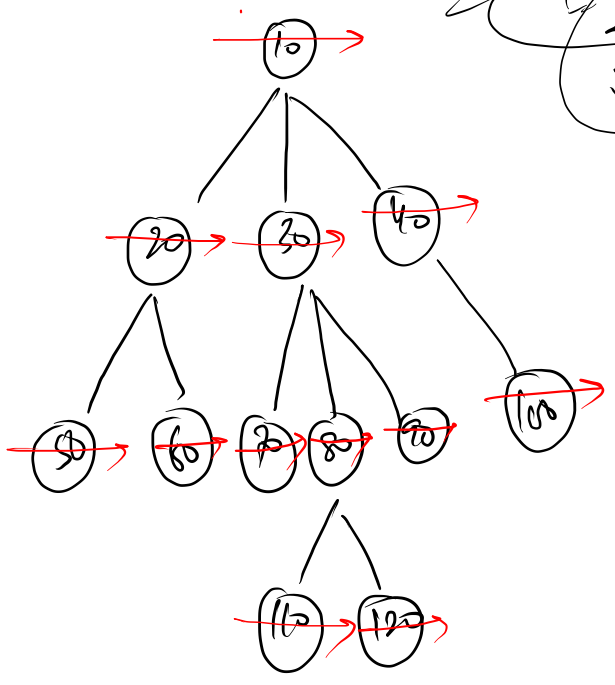
Queue<Node> queue = new ArrayDeque<>();

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

  queue.add(root);
  while(queue.size() > 0){
      Node tmp = queue.remove();
      System.out.print(tmp.data +" ");
      for(Node child : tmp.children){
          queue.add(child);
      }
  }
  System.out.println(".");
}
```

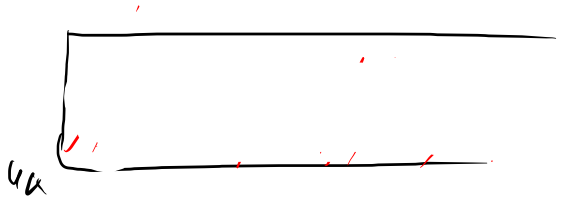




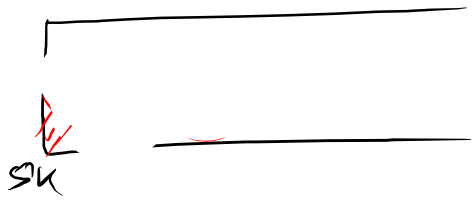
10 20 30 40 50 60



System.out.println()

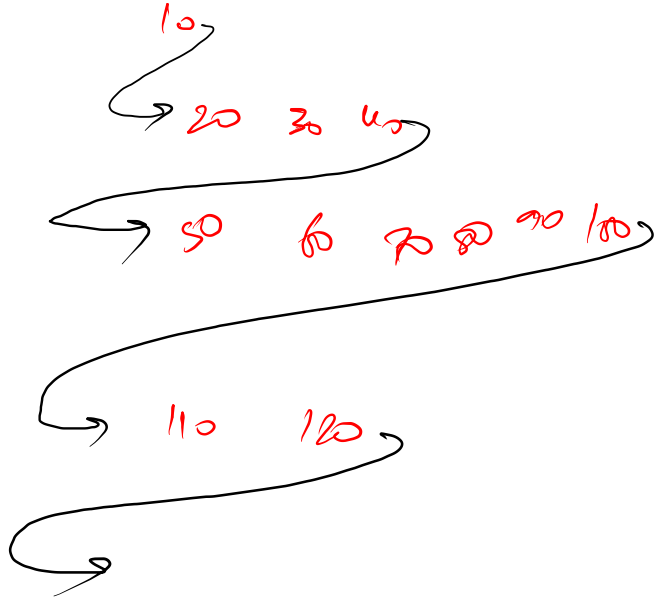


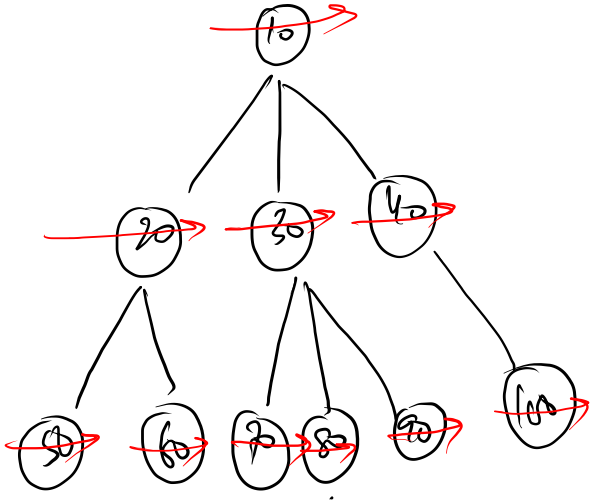
mainQ = 40

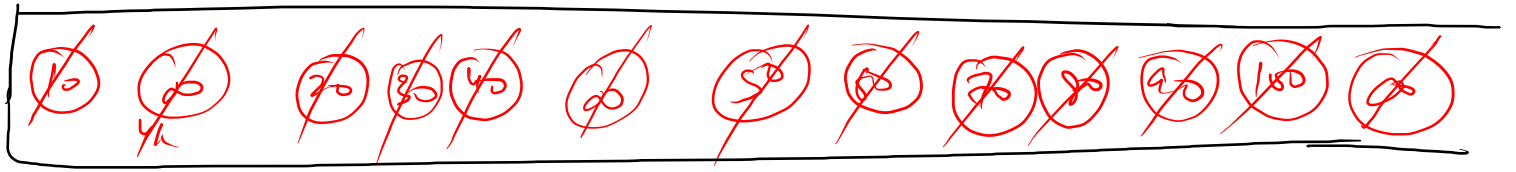


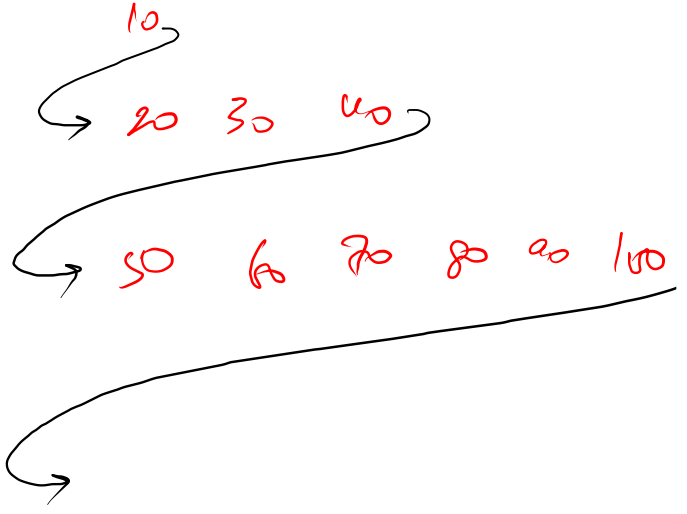
helperQ = 50

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

  mainQ.add(root);
  while(mainQ.size() > 0){
      Node curr = mainQ.remove();

      System.out.print(curr.data+" ");

      for(Node child : curr.children){
          helperQ.add(child);
      }

      if(mainQ.size() == 0){
          System.out.println();
          Queue<Node> tmp = mainQ;
          mainQ = helperQ;
          helperQ = tmp;
      }
  }
}
```





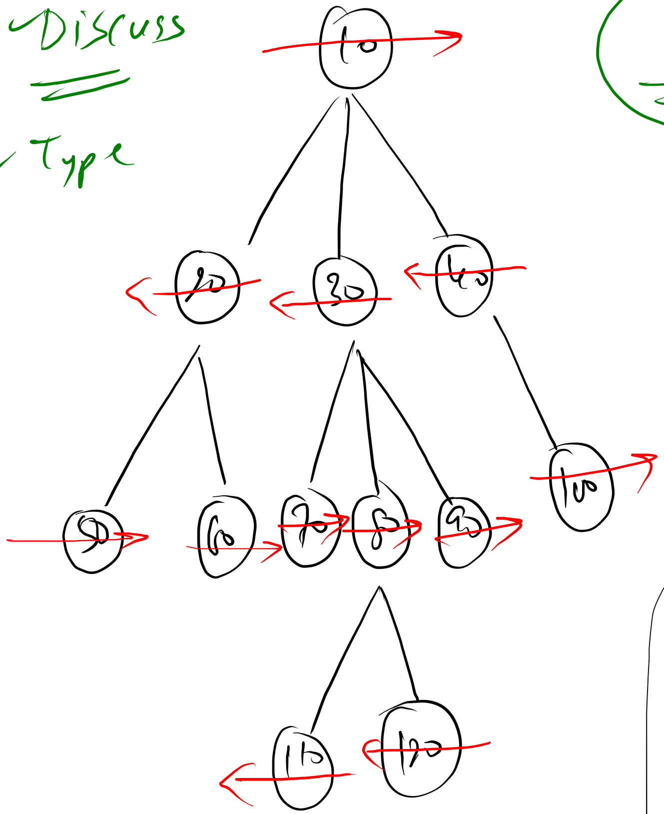





```java
public static void levelOrderLinewise(Node root) {
    Queue<Node> queue = new ArrayDeque<>();
    Node spe = new Node();
    spe.data = Integer.MAX_VALUE;

    queue.add(root);
    queue.add(spe);

    while(queue.size() > 0){
        Node curr = queue.remove();

        if(curr != spe){
            System.out.print(curr.data +" ");

            for(Node child : curr.children){
                queue.add(child);
            }
        }else{
            System.out.println();
            if(queue.size() != 0){
                queue.add(spe);
            }
        }
    }
}
```

Discuss

Type



H.W.

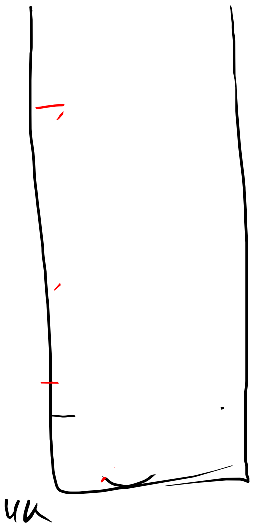
solution

Implementation

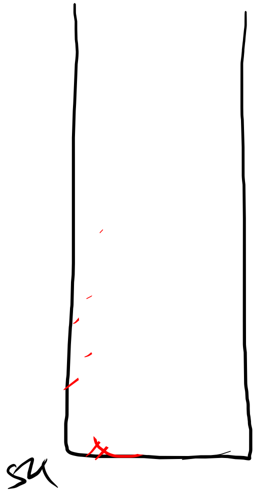
Algorithm

notes

Level: 1 2 3 4 5



mainS = 4K



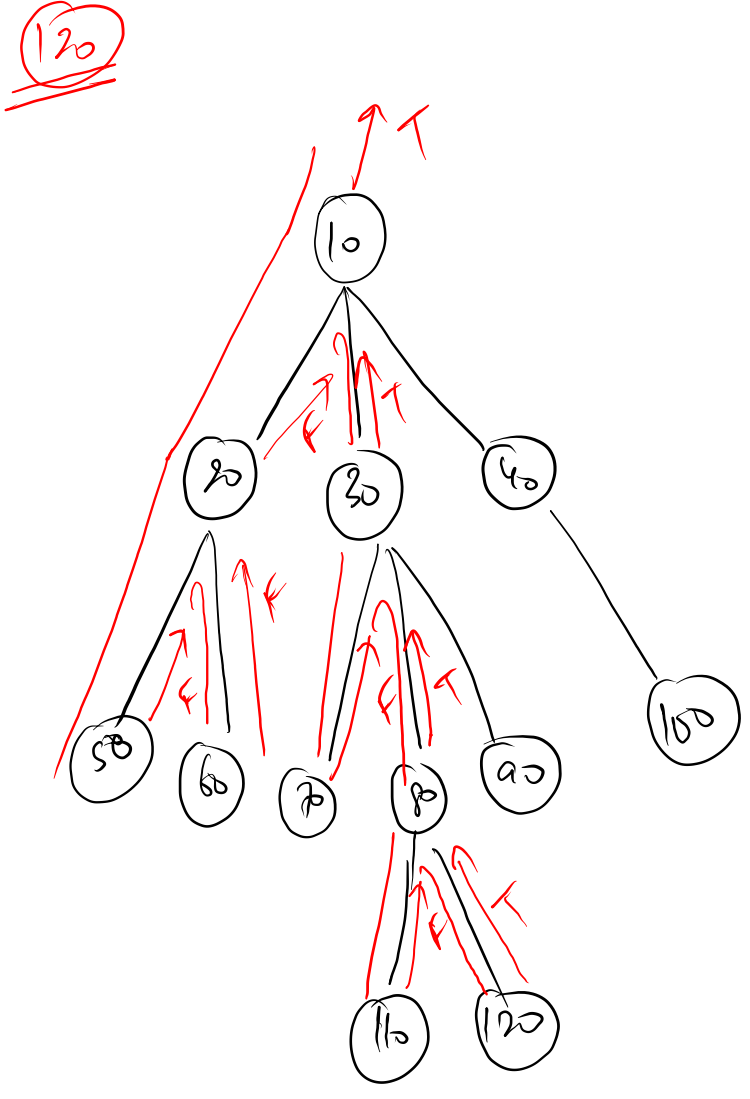
helperS = 5K

10

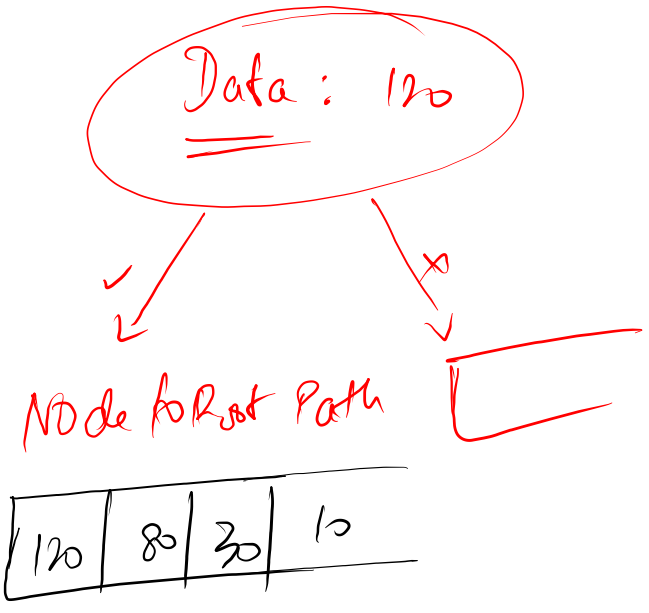
40 30 20

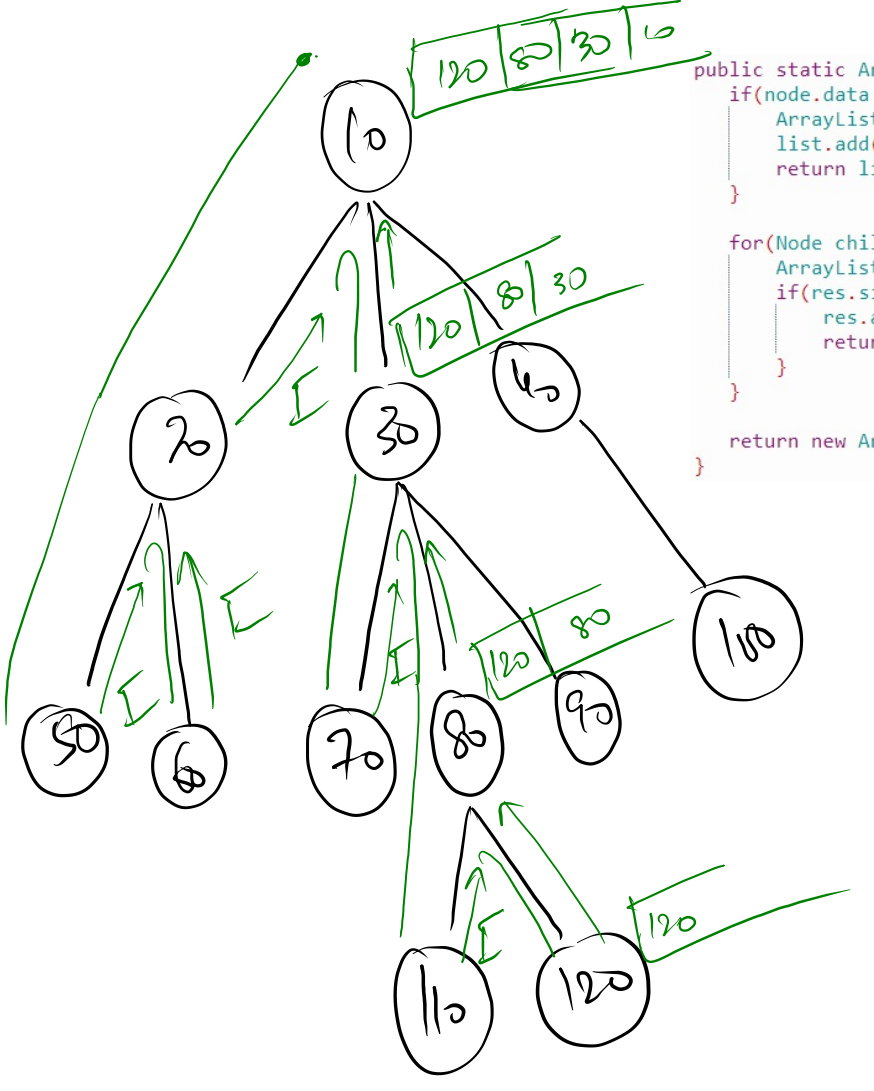
50 60 70 80 90 100

120 110

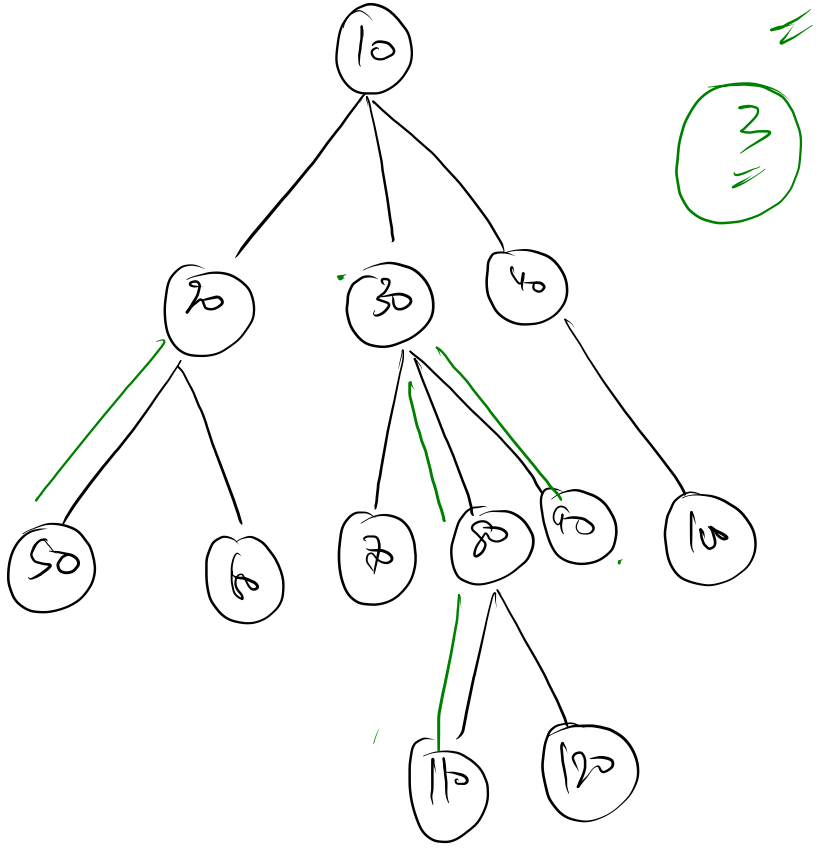



```java
public static boolean find(Node node, int data) {
  if(node.data == data){
      return true;
  }
  for(Node child : node.children){
      boolean res = find(child,data);
      if(res){
          return true;
      }
  }
  return false;
}
```



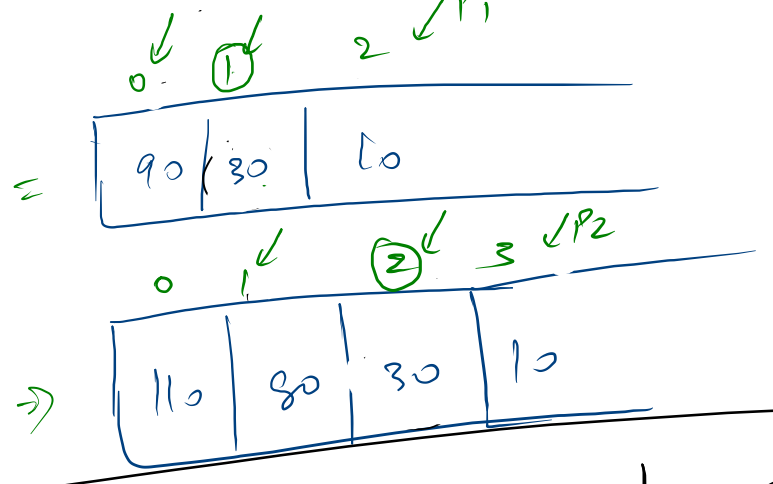



```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
    if(node.data == data){
        ArrayList<Integer> list = new ArrayList<>();
        list.add(data);
        return list;
    }

    for(Node child : node.children){
        ArrayList<Integer> res = nodeToRootPath(child,data);
        if(res.size() != 0){
            res.add(node.data);
            return res;
        }
    }

    return new ArrayList<>();
}
```

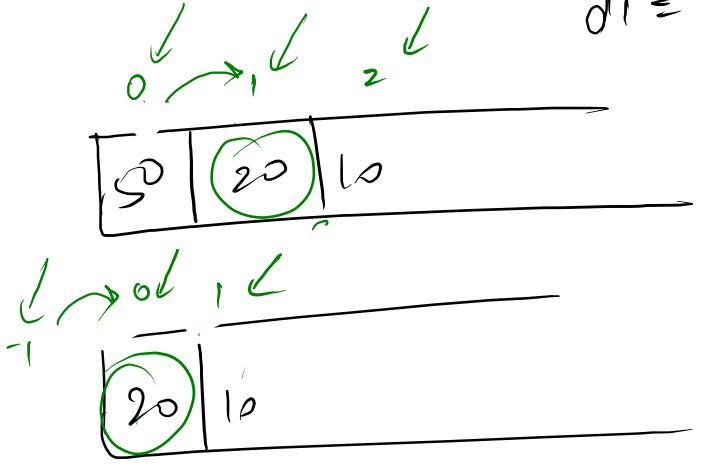


L.C.A.

$d_1 = 90$ $d_2 = 110$



$d_1 = 50$ $d_2 = 20$



```java
public static int lca(Node node, int d1, int d2) {
  ArrayList<Integer> p1 = nodeToRootPath(node, d1);
  ArrayList<Integer> p2 = nodeToRootPath(node, d2);

  int i = p1.size() - 1;
  int j = p2.size() - 1;

  while(i >= 0 && j >= 0 && p1.get(i) == p2.get(j)){
    i--;
    j--;
  }

  return p1.get(i + 1);
}

public static int distanceBetweenNodes(Node node, int d1, int d2){
  ArrayList<Integer> l1 = nodeToRootPath(node,d1);
  ArrayList<Integer> l2 = nodeToRootPath(node,d2);

  int p1 = l1.size()-1 , p2 = l2.size()-1;

  while( p1 >= 0 && p2 >= 0 && l1.get(p1) == l2.get(p2)){
      p1--;
      p2--;
  }
  p1++;
  p2++;
  return p1+p2;
}
```