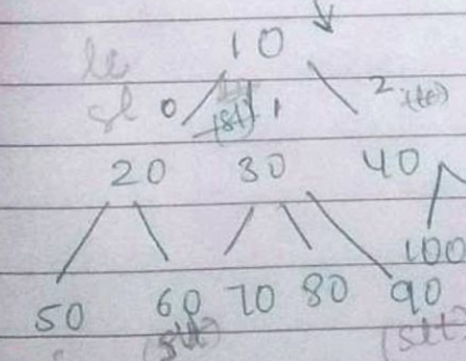
7-JAN-22

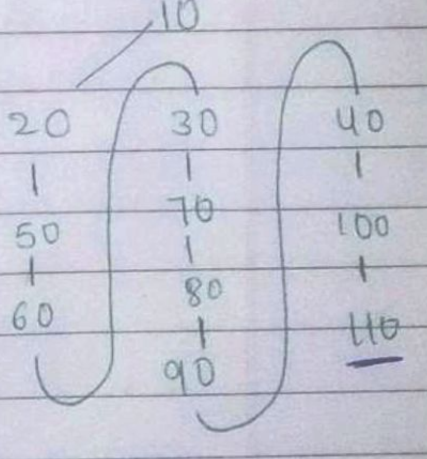
1. Linearize Efficient - O(n)

// It linearizes & returns the tail.

```
public static Node linearizeEfficient(Node node) {
    if (node.children.size()==0) {
        return node;                    → BASE CASE, Because of
    }                                     While loop

                                                     size
    Node lastChild = node.children.get(node.children() - 1);
    Node lastKiTail = linearizeEfficient(lastChild);

    while (node.children.size() > 1) {
                                                       size
        Node slastChild = node.children.get(node.children() - 2);
        Node slastKiTail = linearizeEfficient(slastChild);
        slastKiTail.children.add(lastChild);

        node.children.remove(node.children.size() - 1);
        lastChild = slastChild;
    }
    return lastKiTail;
}
```

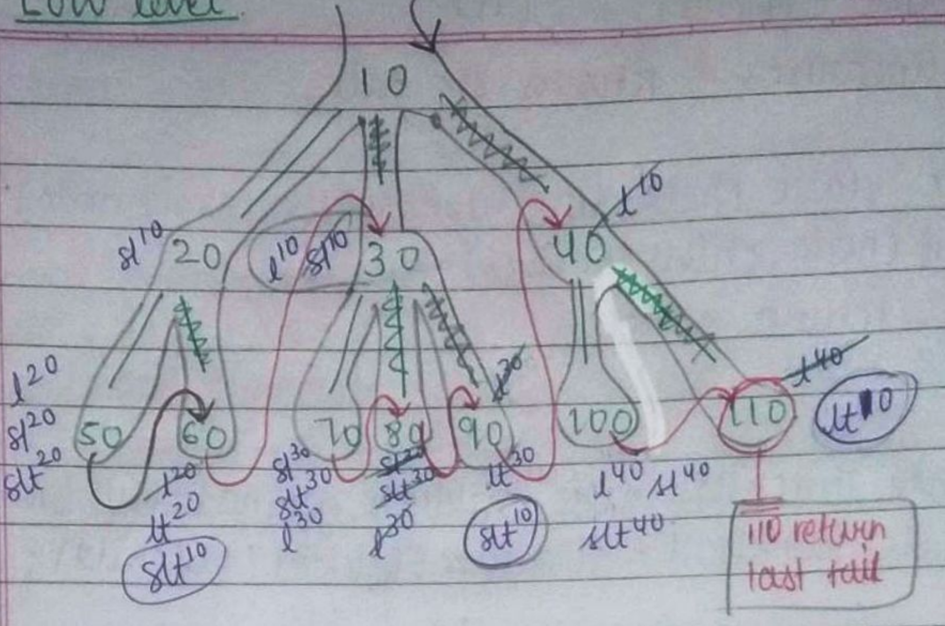
HIGH LEVEL



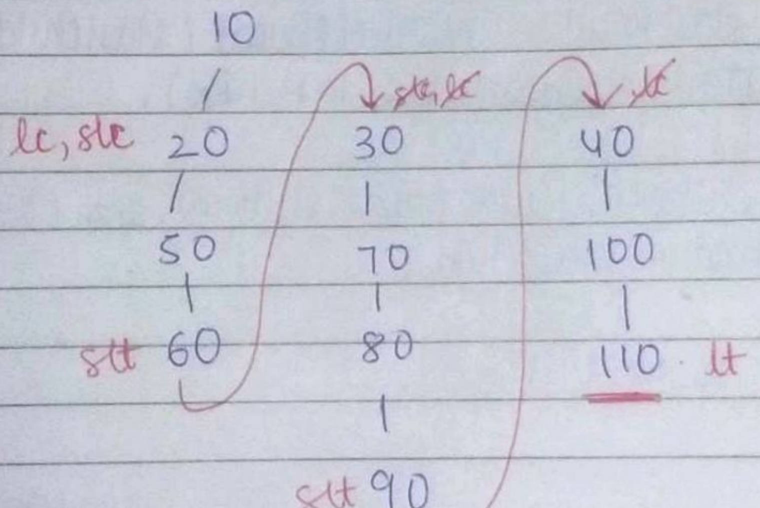


100 return

Low level.



10
20
30
40
50
60
70
80
90
100
110
sl 10
l 10 sl 10
l 10
l 20
sl 20
slt 20
lt 20
slt 10
sl 30
slt 30
l 30
l 30
lt 30
slt 10
l 40
sl 40
slt 40
l 40
lt 10
110 return last tail



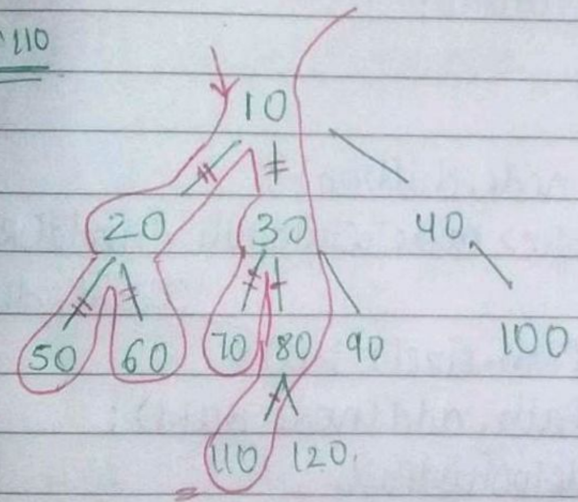
10
lc, stc
20
50
slt 60
st, lc
30
70
80
slt 90
lc
40
100
110
lt

2. Find In Generic Tree -

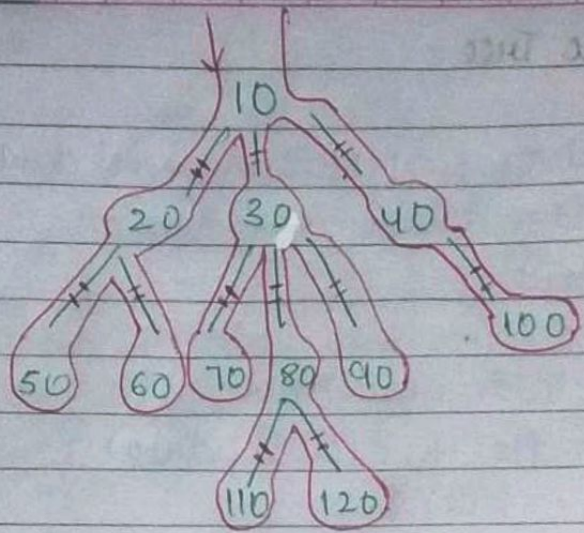
```
public static boolean find(Node node, int data) {
pre-   ( if(node.data == data){
order  (    return true;
       ( }
        for(Node child : node.children){
            boolean fic = find(child, data) ; (found in child)
            if(fic == true){
                return true; → (Bich me return to partial euler)
            }
        }

post-  ( return false;
order
}.
```

(partial euler - Bichse true return honge.)

for 110



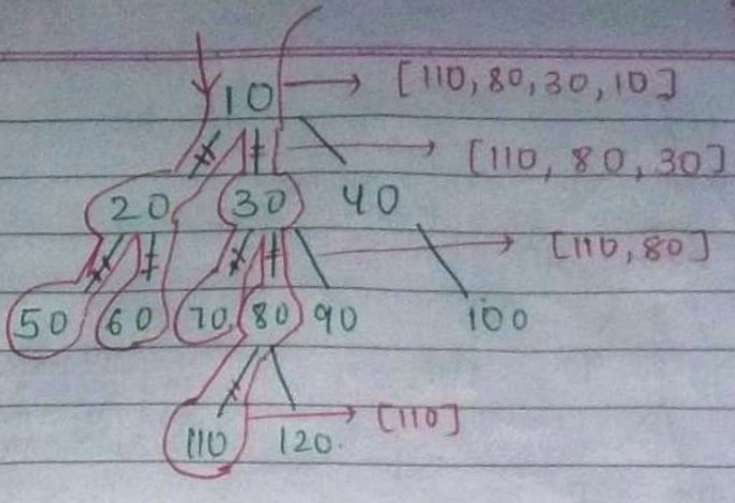
for 150



return false

3. Node To Root Path in Generic Tree :-

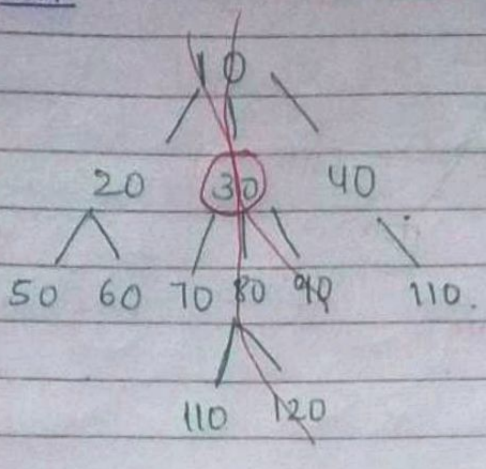
```
public static ArrayList<Integer> nodeToRootPath
                              (Node node, int data) {
    if(node.data ==data) {
        ArrayList<Integer> bres = new ArrayList<>();
        bres.add(node.data);
        return bres;
    }
    for(Node child: node.children) {
        ArrayList<Integer> nodeToChildPath = nodeToRootPath
                                              (child, data);
        if(nodeToChildPath.size() >0) {
            nodeToChildPath.add(node.data);
            return nodeToChildPath;
        }
    }
    return new ArrayList<>();
}
```



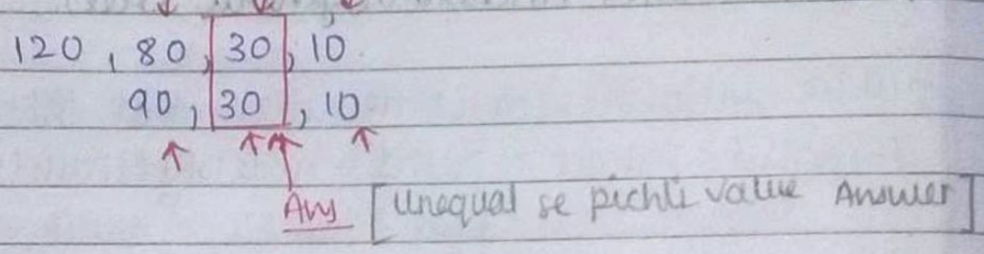
4. Lowest Common Ancestor (generic Tree).

```
public static int lca (Node node, int d1, int d2) {
 ArrayList<Integer> path1 = nodeToRootPath (node, d1);
    "          "          path 2 =         "              d2);
 int i = path1.size() - 1;
 int j = path2.size() - 1;
 while ( i>=0 && j >=0) {
     if( path1.get(i) == path2.get(j)) {
          i--;
          j--;
     } else {
         break;
     }
 }
 int lca = path.get (i+1);
 return lca;
}
```

LCA



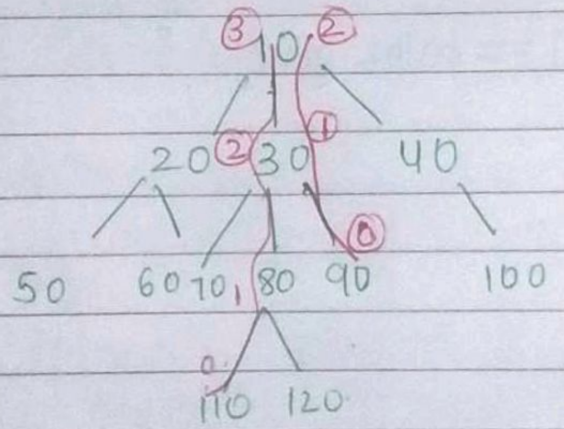
LCA -
120 & 90 = 30
70 & 110 = 30
50 & 110 = 10
110 & 120 = 80



120, 80, 30, 10
90, 30, 10

Ans [Unequal se pichli value Answer]

5. Distance b/w Two Nodes In a Generic Tree -



110 = 110^0, 80^1, 30^2, 10^3

90 = 90^0, 30^1, 10^2

3 return.

```
public static int distanceBetweenNodes(Node node,int d1,
                                     int d2){
 ArrayList<Integer> path1 = nodeToRootPath(node, d1);
ArrayList<Integer> path2 = nodeToRootPath(node, d2);

 int i = path.size() - 1;
 int j = path.size() - 1;

 while (i>=0 && j>=0 && path.get(i)==path2.get(j)){
         i--;
         i--;
         }
         i++;
         j++;

      return i+j;
}.
```