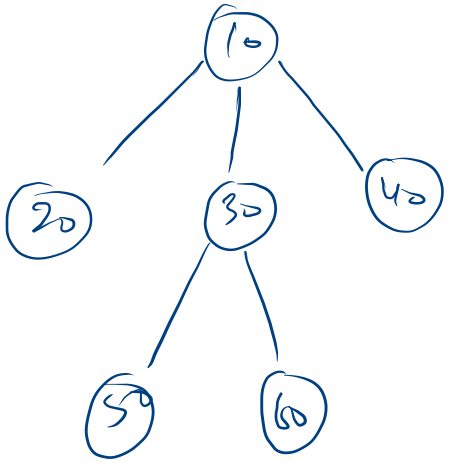
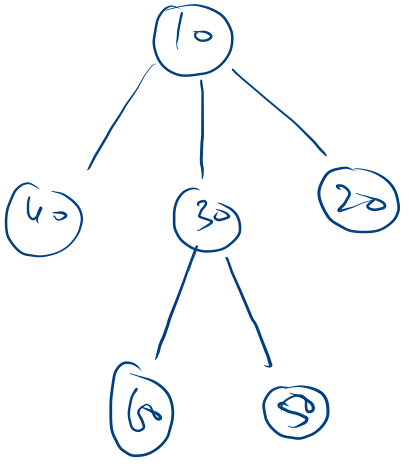
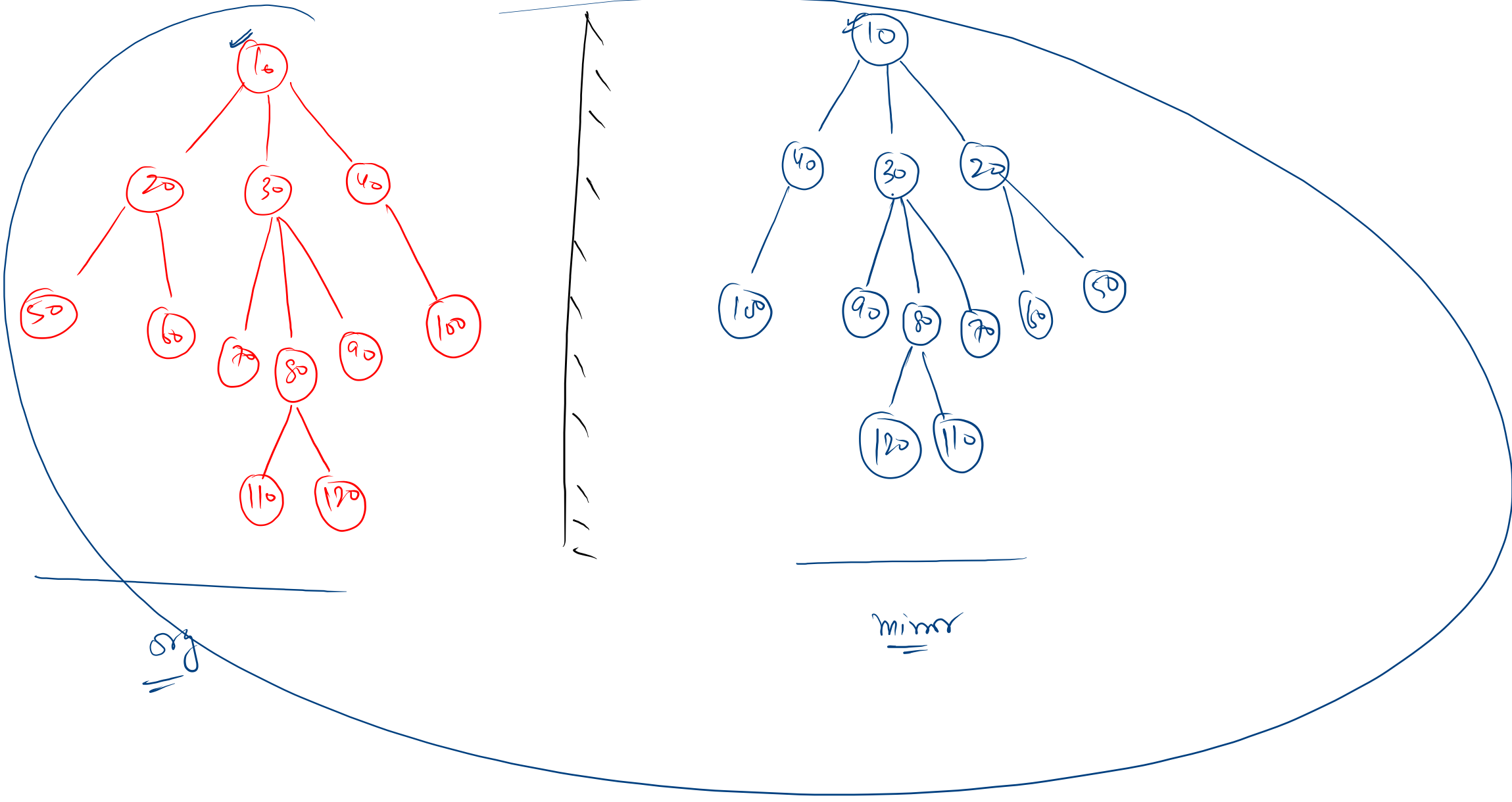
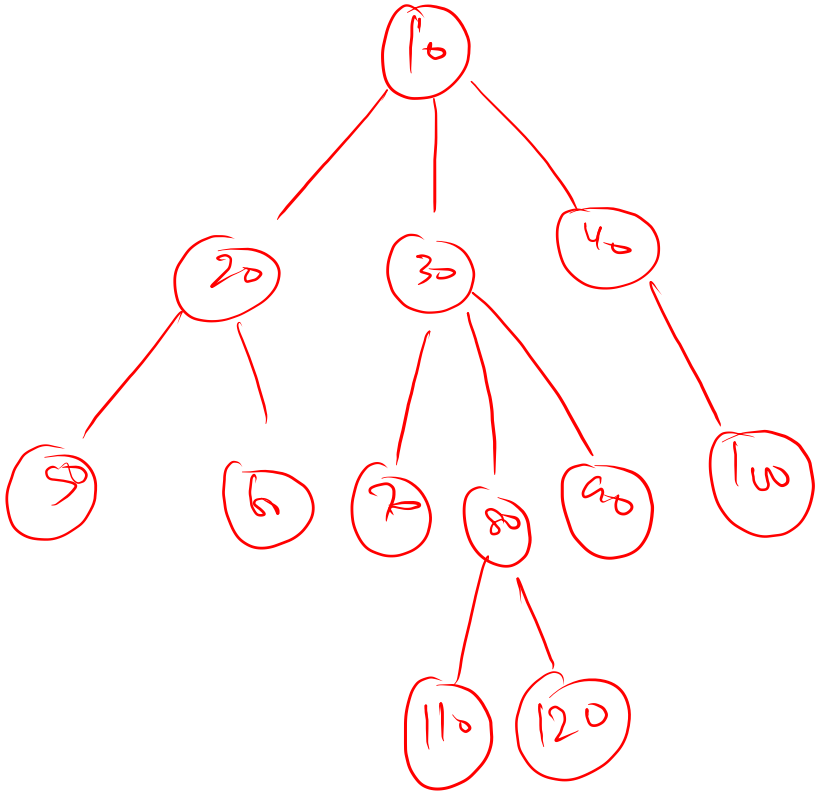
A B C D

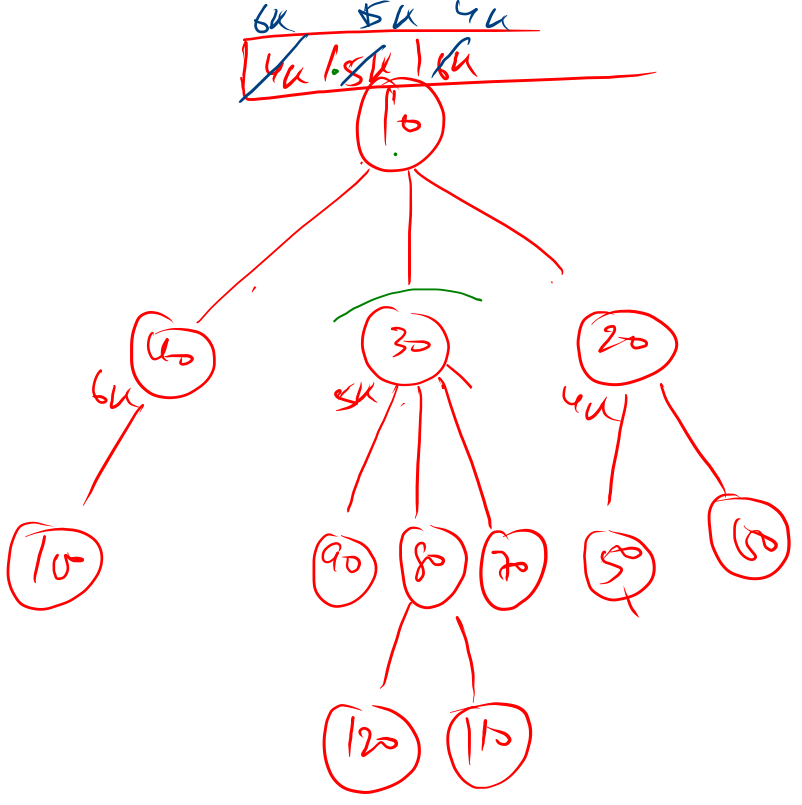
D C B A


10
20
30
40
50
60

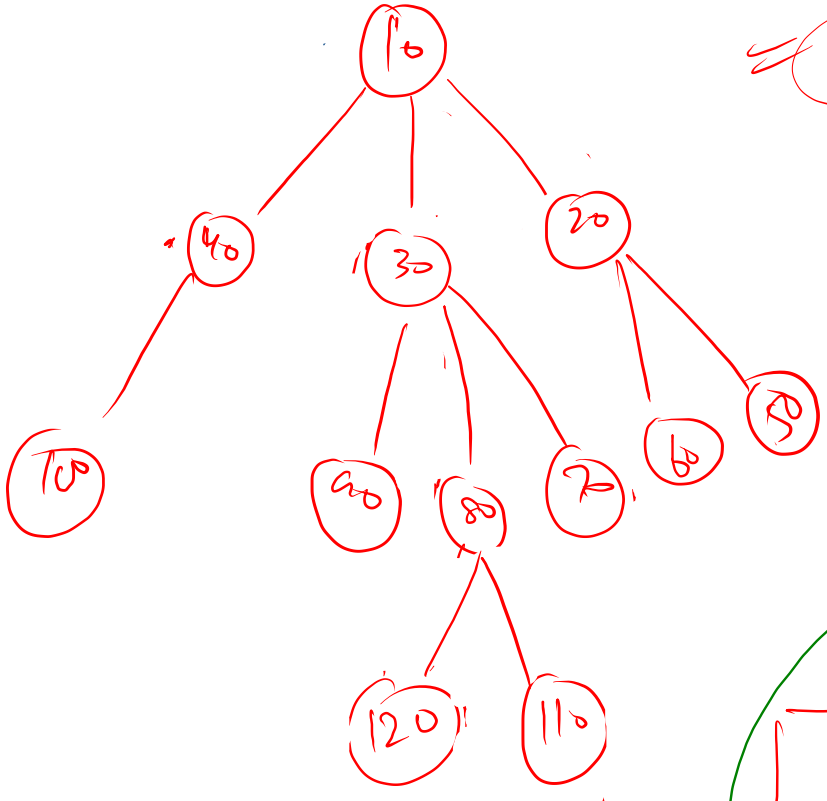

10
40
30
20
50
90


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



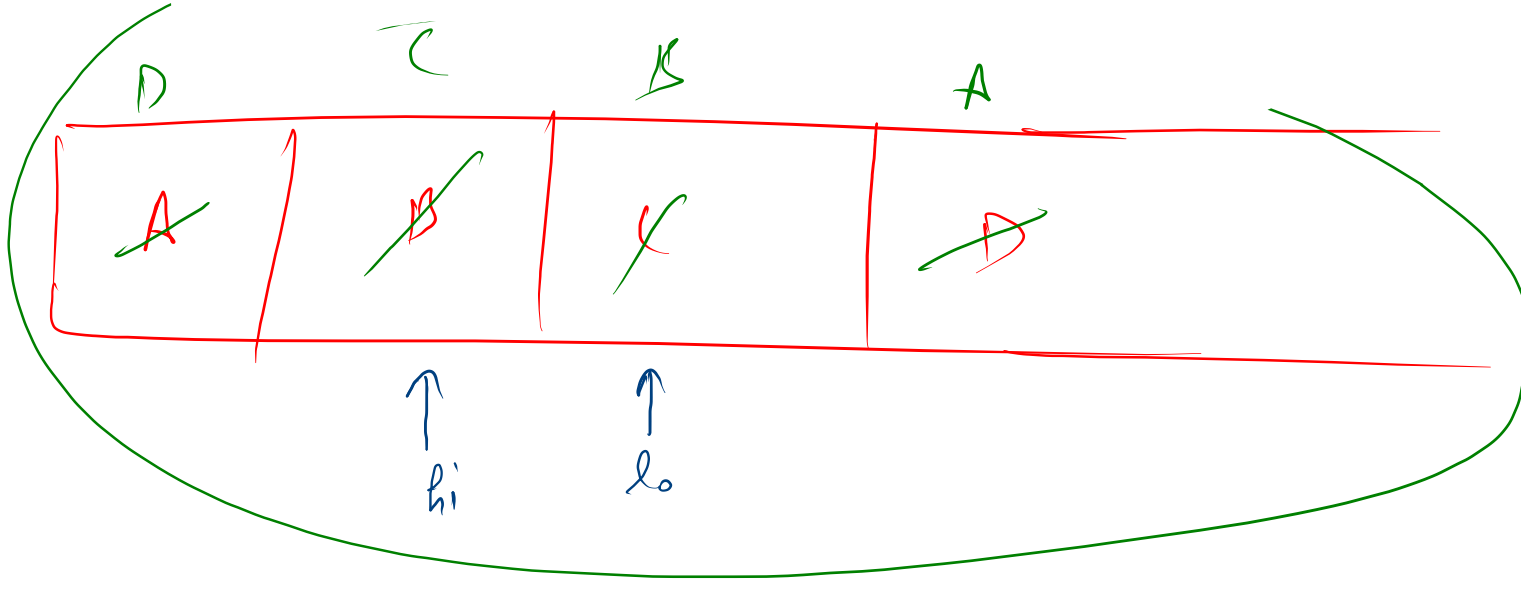


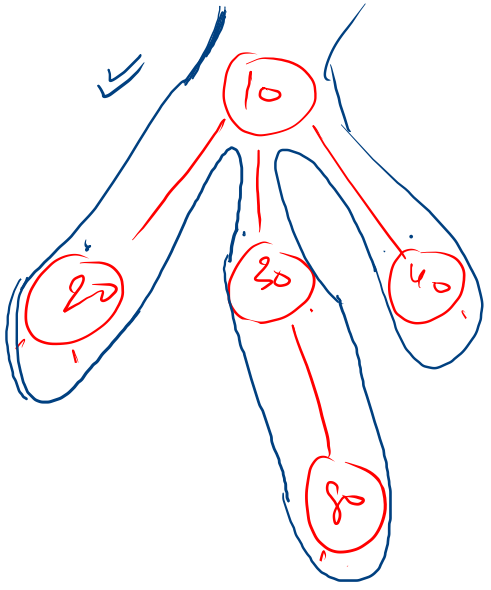

```java
public static void mirror(Node node){

  for(Node child : node.children){
      mirror(child);
  }

  Collections.reverse(node.children);
}
```

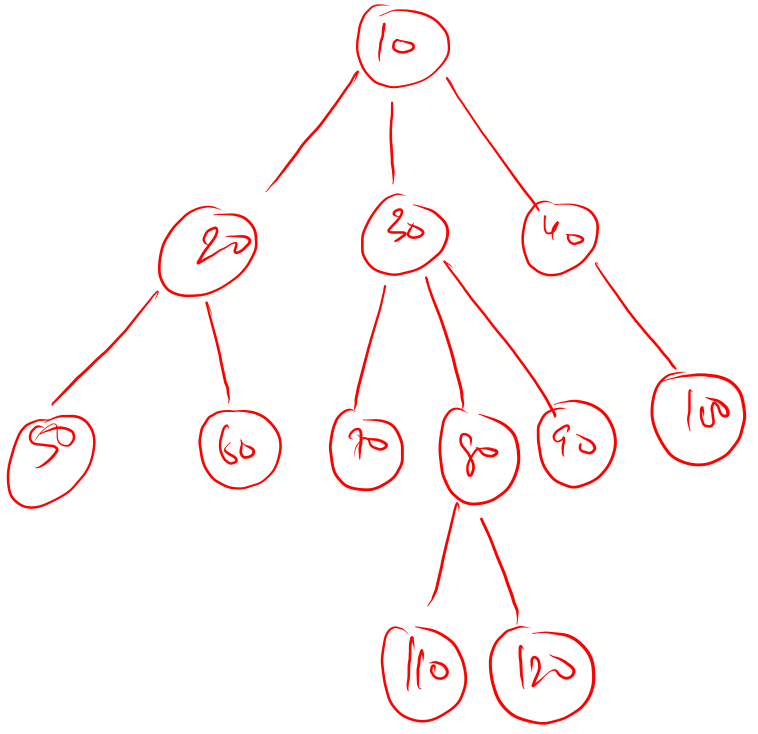


n^2

```
public static void mirror(Node node){

   for(Node child : node.children){
       mirror(child);
   }

   Collections.reverse(node.children);
}
```

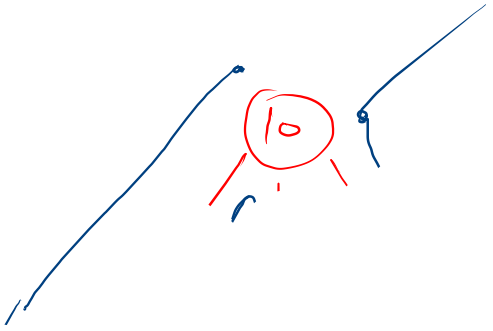




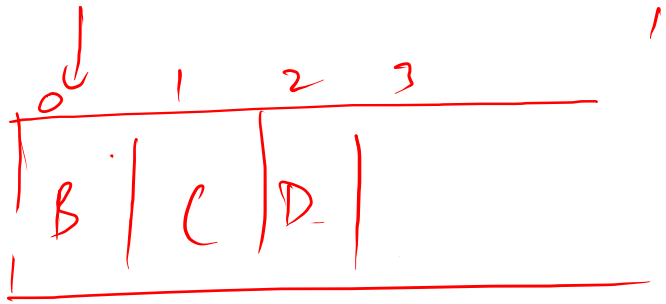


node.children.size() == 0

↳ leaf node

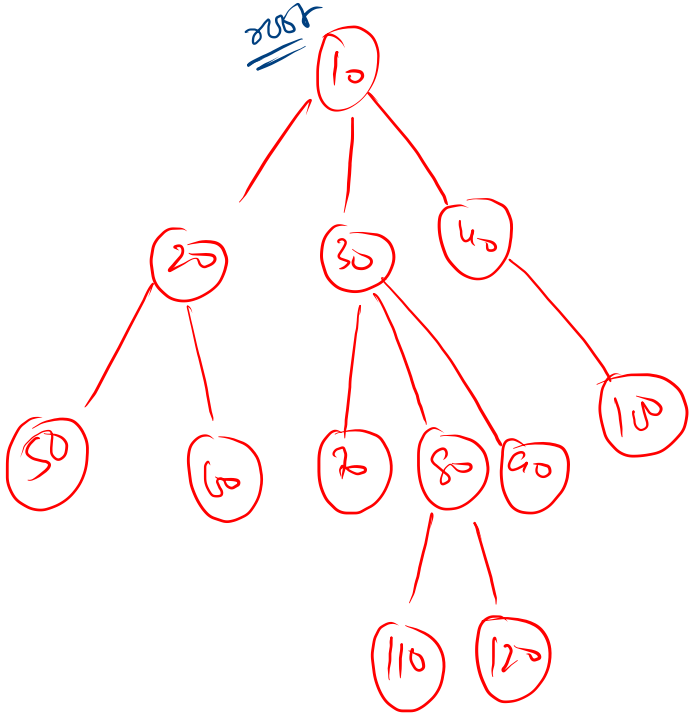


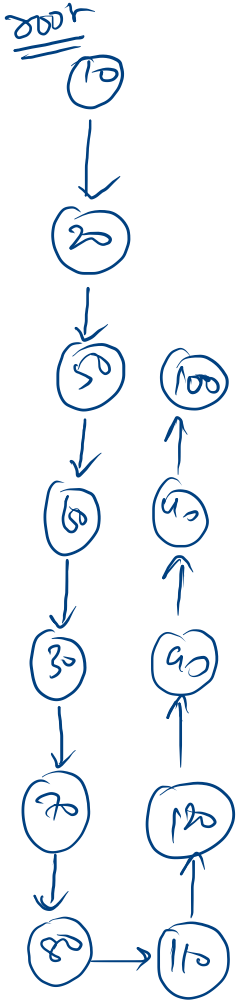
```java
public static void removeLeaves(Node node) {


  for(Node child : node.children){
      removeLeaves(child);
  }

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

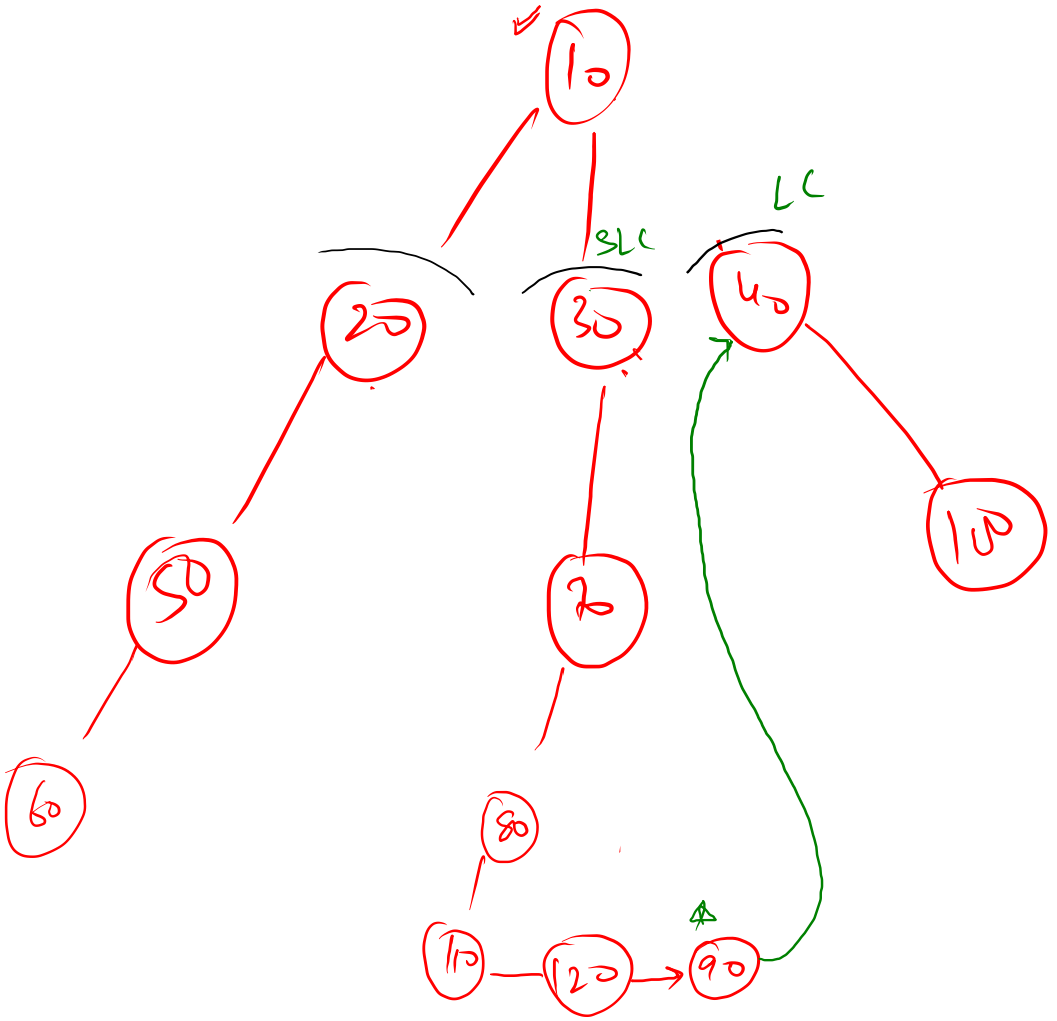


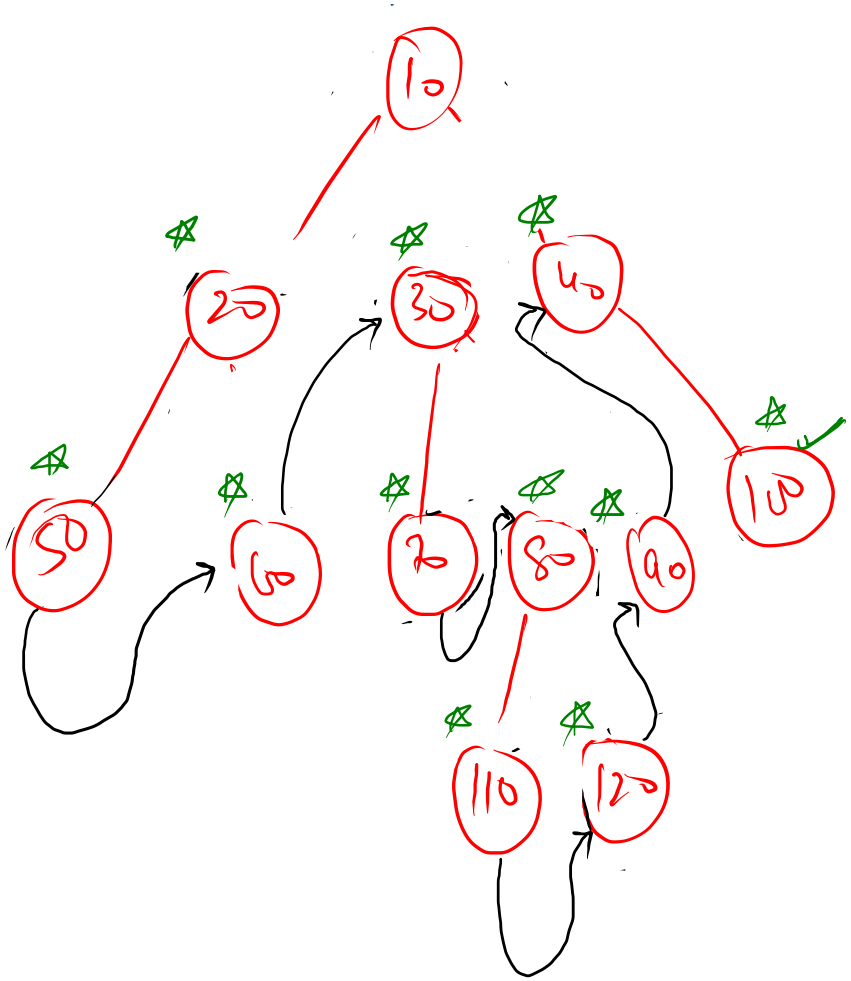
idx=0

```
for (int i=0 ; i< n; i++) {

}
```


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


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

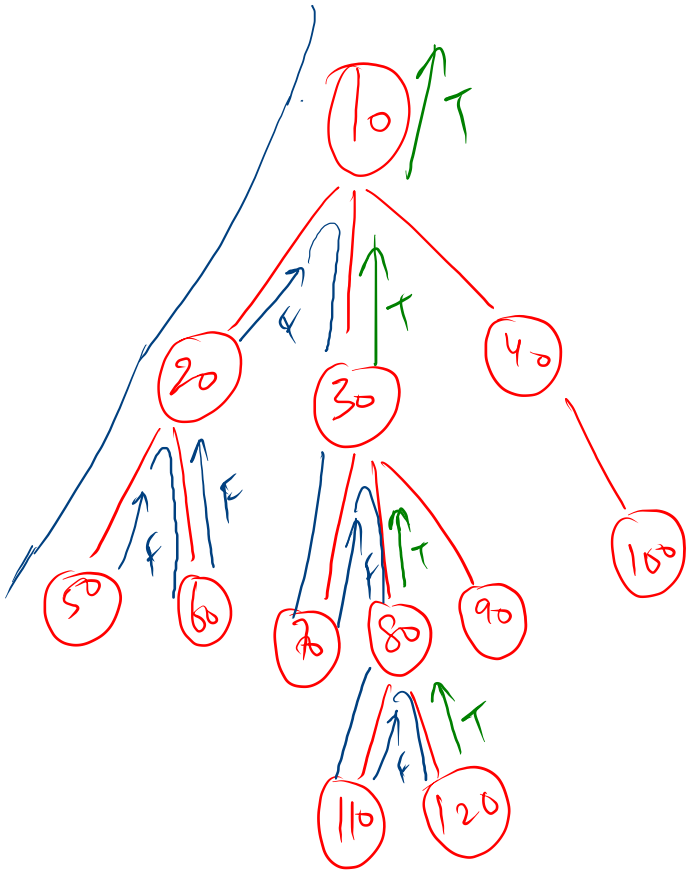

10
SLC
LC
25
30
40
50
20
15
60
80
110
120
90



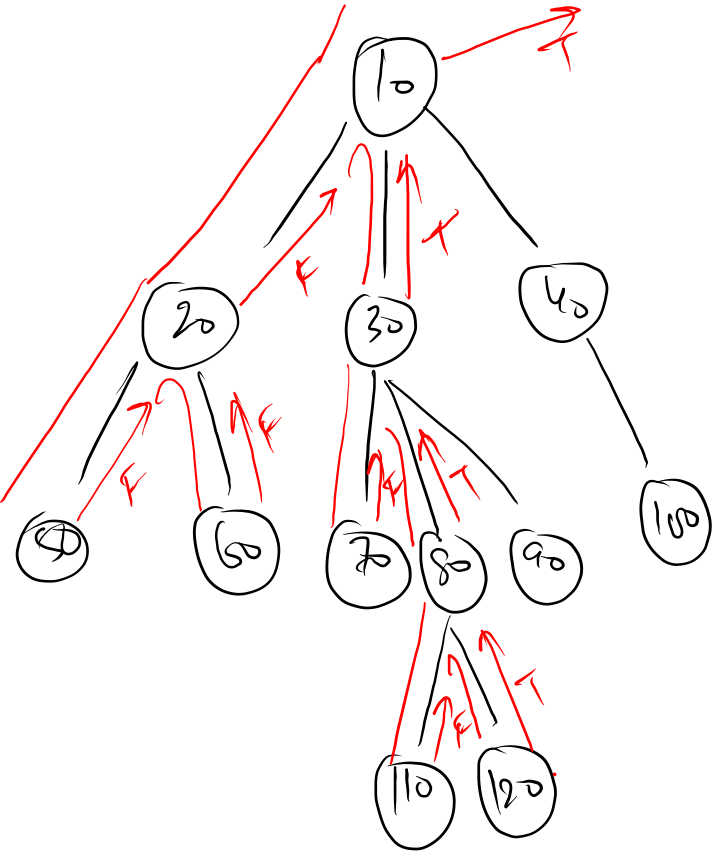

```java
public static Node getTail(Node node){
    while(node.children.size() == 1){
        node = node.children.get(0);
    }

    return node;
}
```

```java
public static void linearize(Node node){
  for(Node child : node.children){
      linearize(child);
  }

  while(node.children.size() > 1){
      Node lc = node.children.remove(node.children.size()-1);
      Node slc = node.children.get(node.children.size()-1);

      Node tail = getTail(slc);

      tail.children.add(lc);
  }
}
```

X = 120 ?

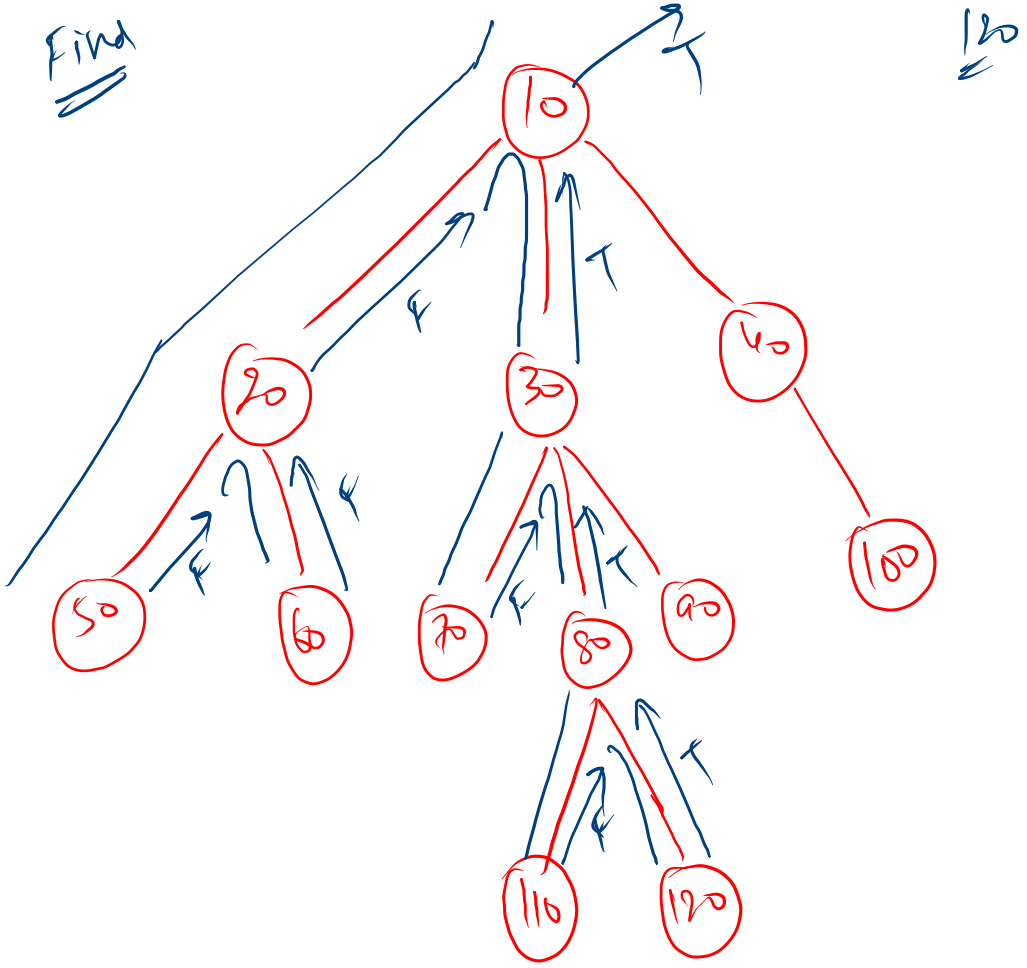


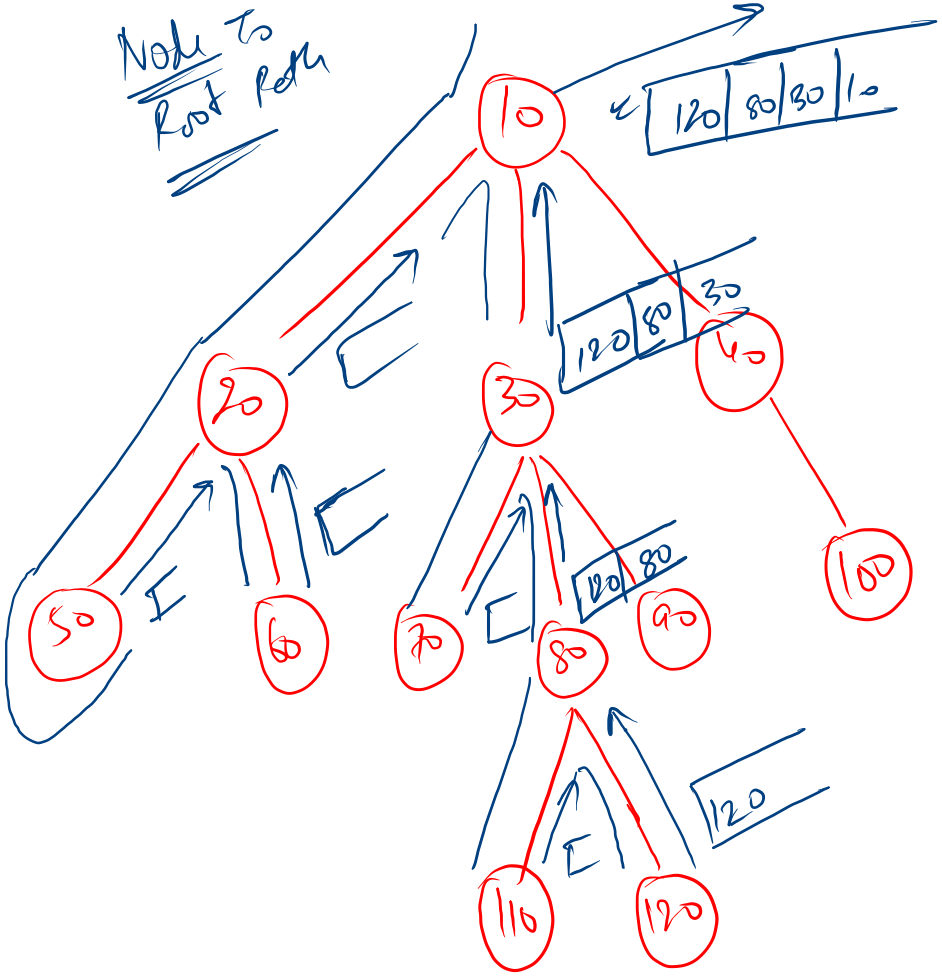
Edge Post

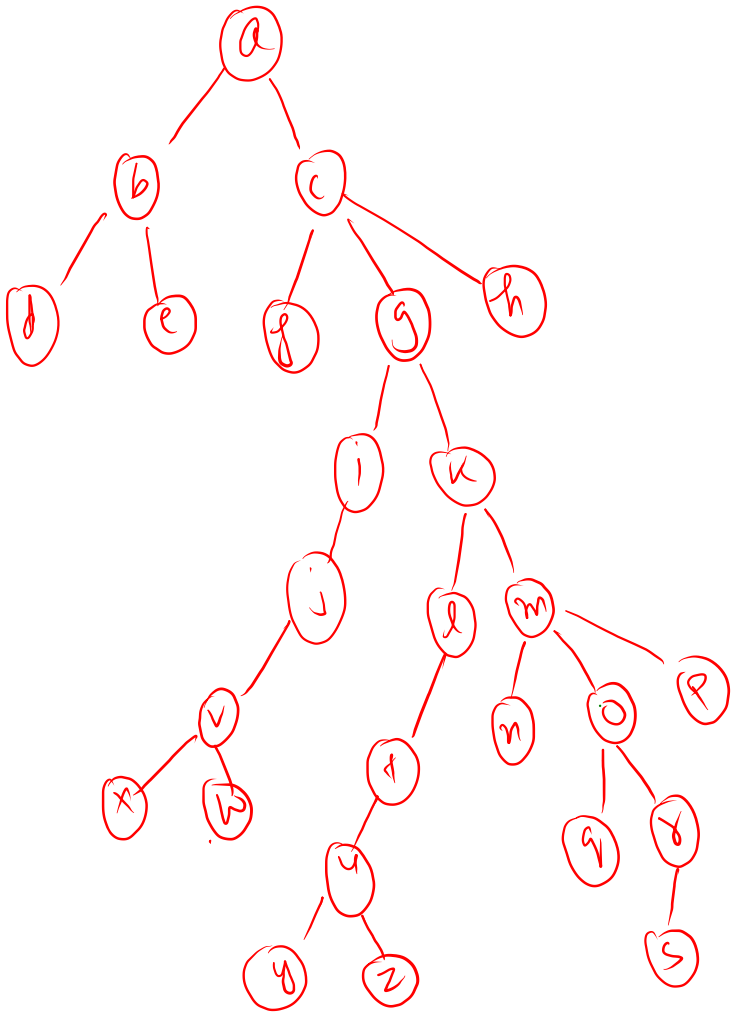


x: 120

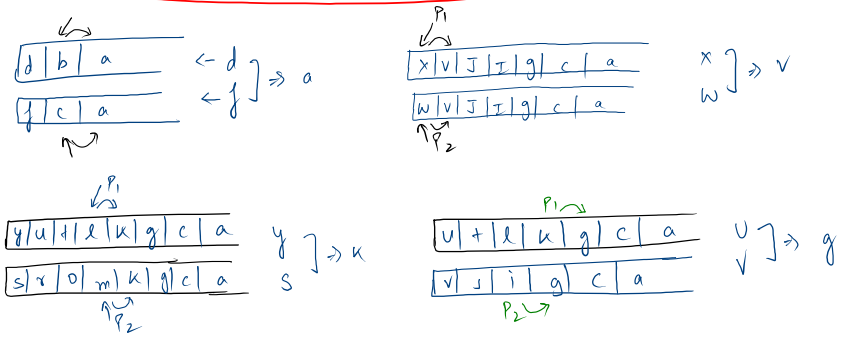
```java
public static boolean find(Node node, int data) {
    if(node.data == data){
        return true;
    }

    for(Node child : node.children){
        boolean res = find(child,data);
        if(res){
            return true;
        }
    }

    return false;
}
```


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

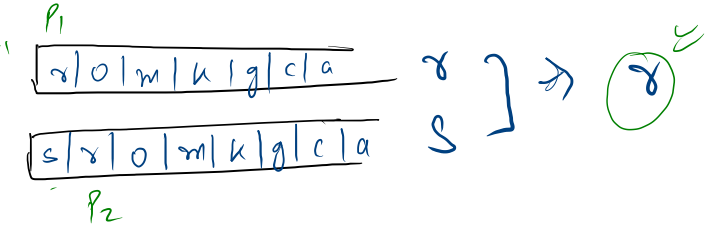

Node to Root path
10
20
30
40
50
60
70
80
90
100
110
120
120 80 30 10
120 80 30
120 80
120

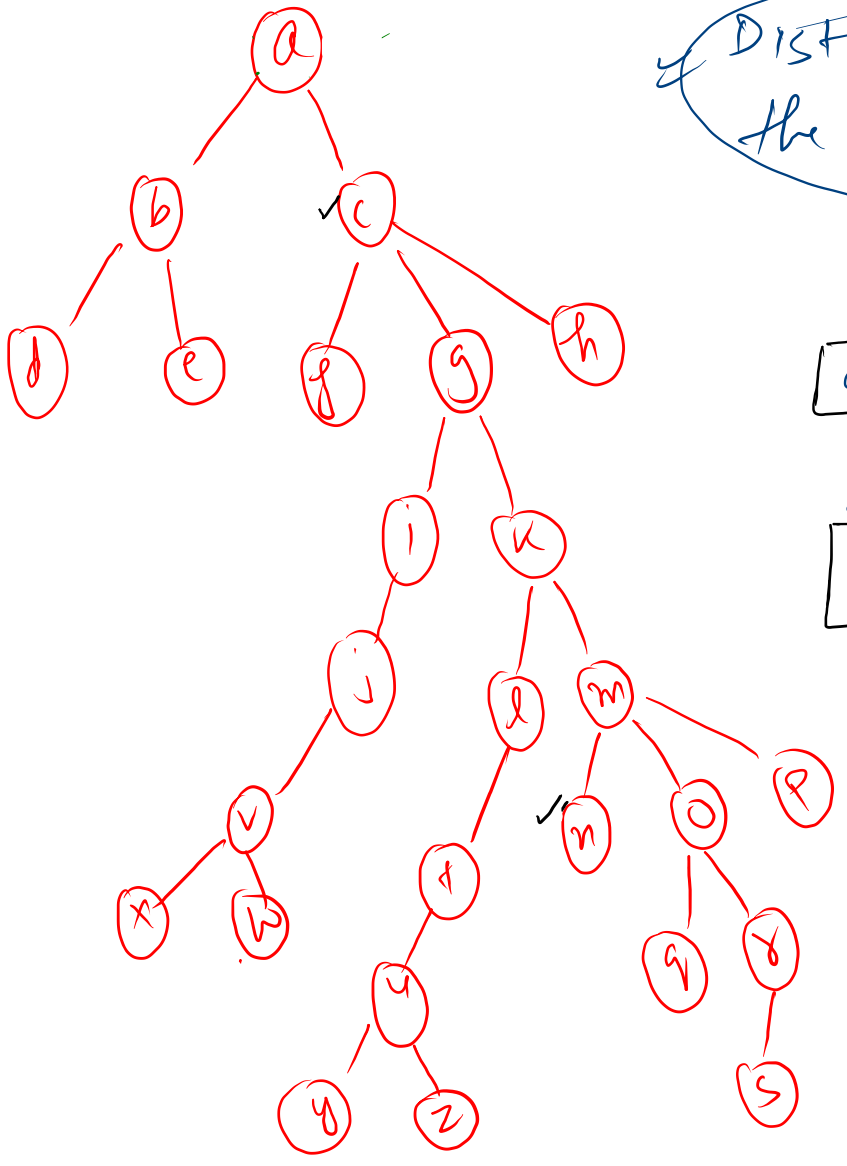


Lowest Common Ancestor




```java
public static int lca(Node node, int d1, int d2) {
    ArrayList<Integer> list1 = nodeToRootPath(node,d1);
    ArrayList<Integer> list2 = nodeToRootPath(node,d2);

    int p1 = list1.size()-1 , p2 = list2.size()-1;

    while(p1 >= 0 && p2 >= 0){
        if(list1.get(p1) == list2.get(p2)){
            p1--;
            p2--;
        }else{
            break;
        }
    }

    p1++;
    p2++;

    return list1.get(p1);
}
```





Distance b/w two nodes on the basis of edges

$p_1 \rightarrow$

0	1	2
d	b	a

← d $p_1 = 2$

0	1	2
f	c	a

$p_2 \rightarrow$

f $p_2 = 2$

distance = $p_1 + p_2$ →

$p_1 \rightarrow$ (-1)

0	1
c	a

← c $p_1 = 0$

0	1	2	3	4	5
n	m	k	g	c	a

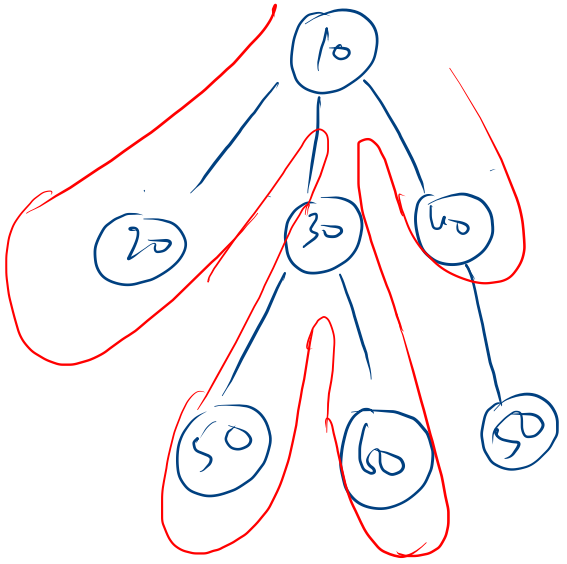
← n $p_2 = 4$

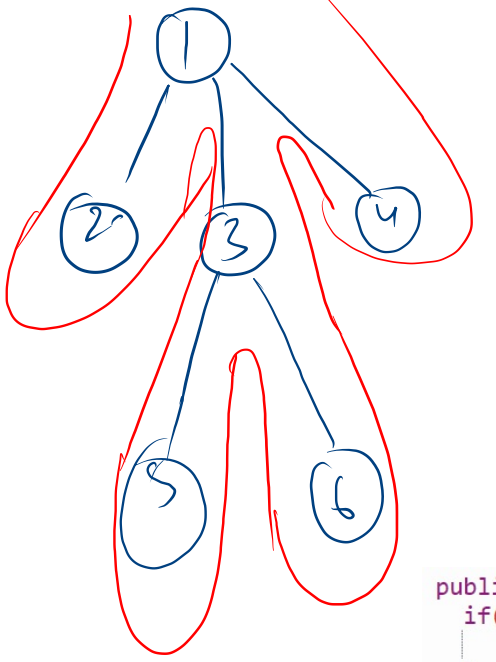
$p_2 \rightarrow$

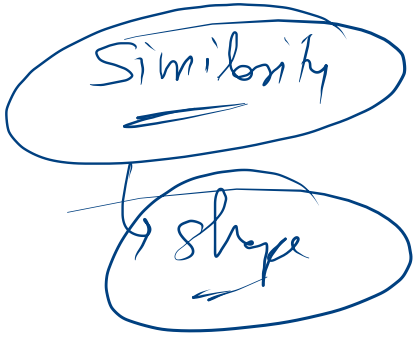
$p_1 + p_2 \rightarrow 0 + 4 =$

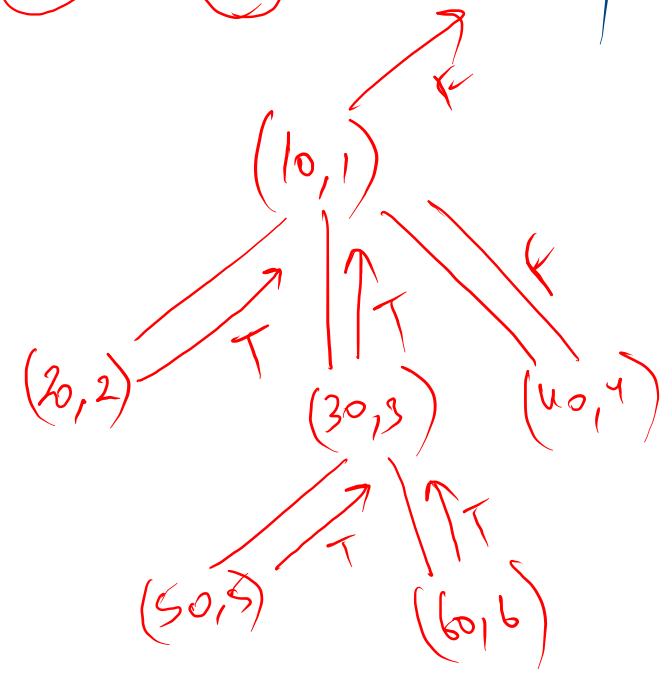
```java
public static int lca(Node node, int d1, int d2) {
  ArrayList<Integer> list1 = nodeToRootPath(node,d1);
  ArrayList<Integer> list2 = nodeToRootPath(node,d2);

  int p1 = list1.size()-1 , p2 = list2.size()-1;

  while(p1 >= 0 && p2 >= 0){
      if(list1.get(p1) == list2.get(p2)){
          p1--;
          p2--;
      }else{
          break;
      }
  }

  p1++;
  p2++;

  return list1.get(p1);
}
```

```java
public static int distanceBetweenNodes(Node node, int d1, int d2){
  ArrayList<Integer> list1 = nodeToRootPath(node,d1);
  ArrayList<Integer> list2 = nodeToRootPath(node,d2);

  int p1 = list1.size()-1 , p2 = list2.size()-1;

  while(p1 >= 0 && p2 >= 0){
      if(list1.get(p1) == list2.get(p2)){
          p1--;
          p2--;
      }else{
          break;
      }
  }

  p1++;
  p2++;

  return p1+p2;
}
```

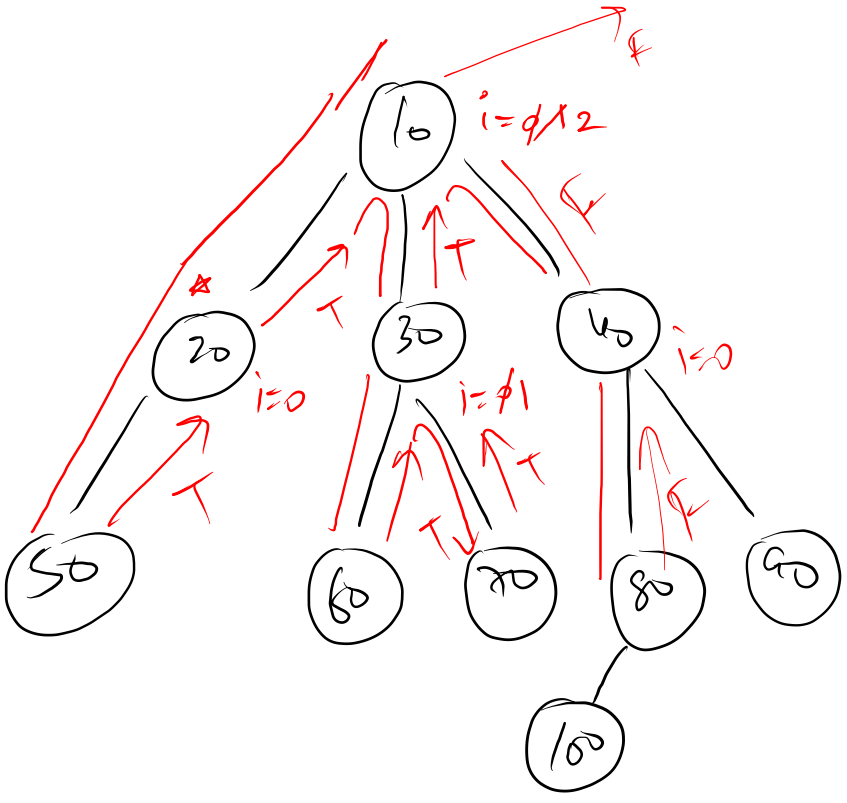


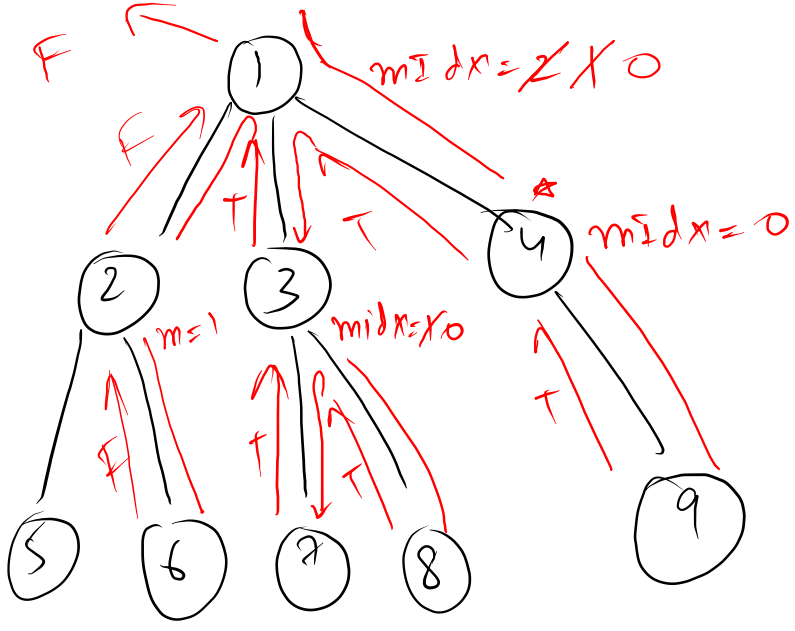




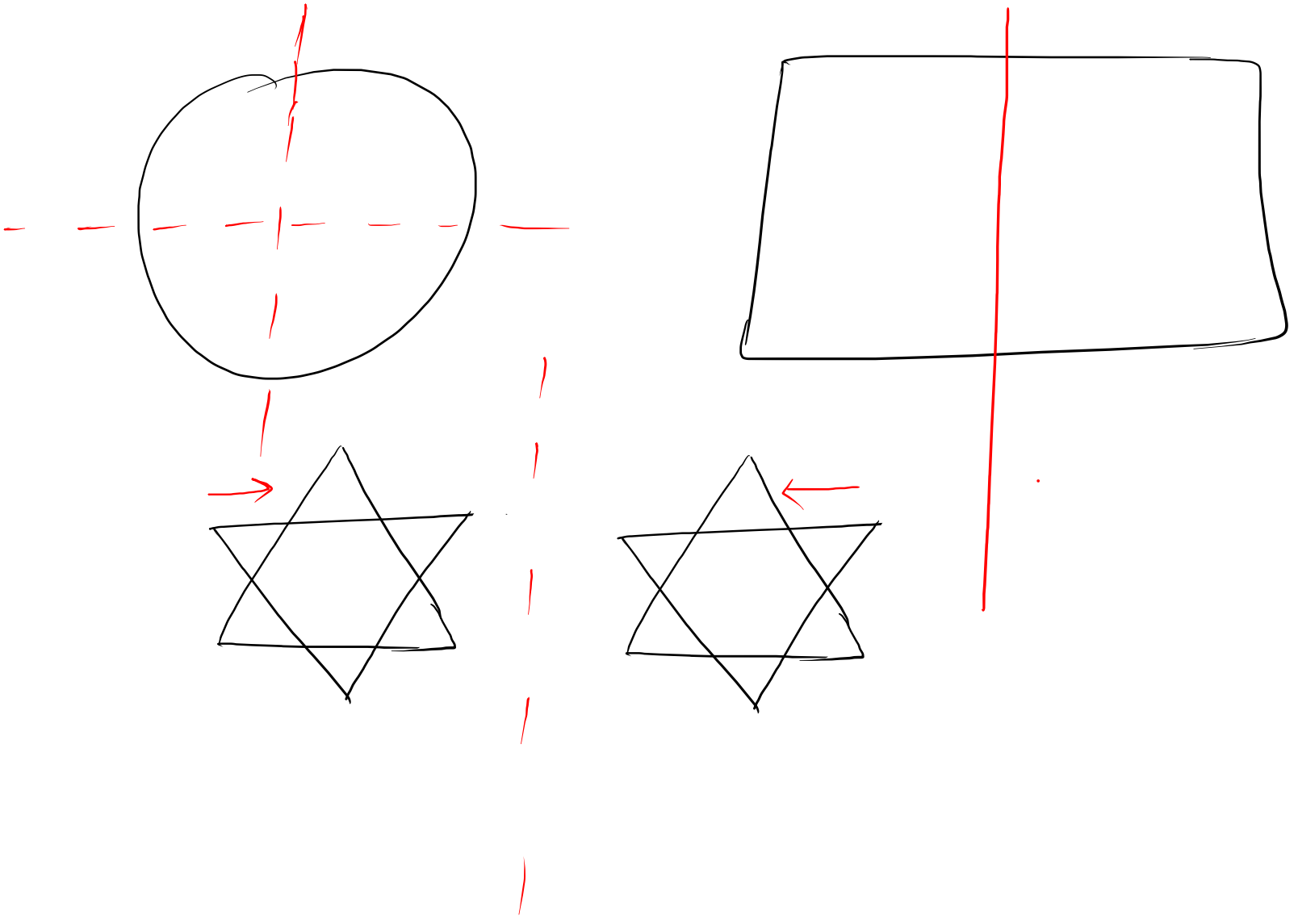


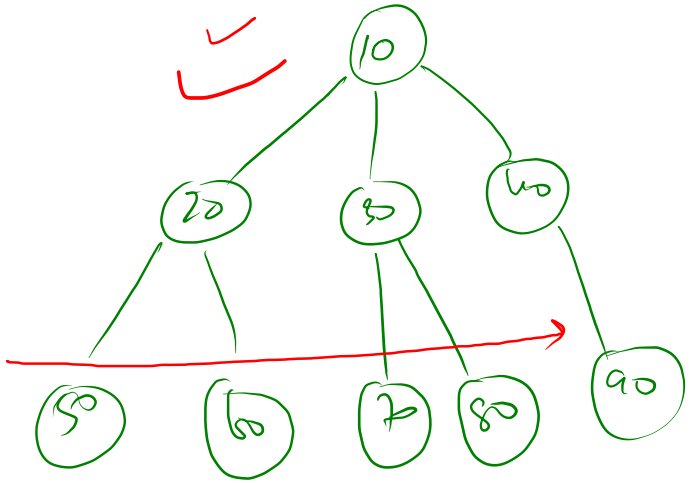

```java
public static boolean areSimilar(Node n1, Node n2) {
  if(n1.children.size() != n2.children.size()){
      return false;
  }

  for(int i = 0 ; i < n1.children.size() ; i++){
      Node child1 = n1.children.get(i);
      Node child2 = n2.children.get(i);

      boolean res = areSimilar(child1,child2);

      if(res == false){
          return false;
      }
  }

  return true;
}
```





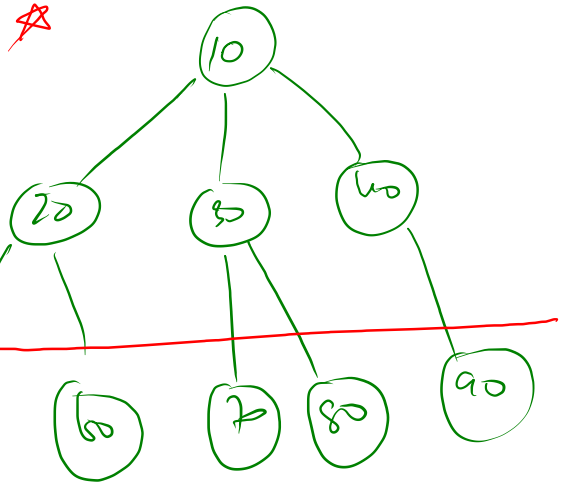

```java
public static boolean areMirror(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()){
        return false;
    }

    for(int i = 0 ; i < n1.children.size() ; i++){
        Node c1 = n1.children.get(i);
        int mIdx = n2.children.size()-1-i;
        Node c2 = n2.children.get(mIdx);

        boolean res = areMirror(c1,c2);
        if(res == false){
            return false;
        }
    }

    return true;
}
```

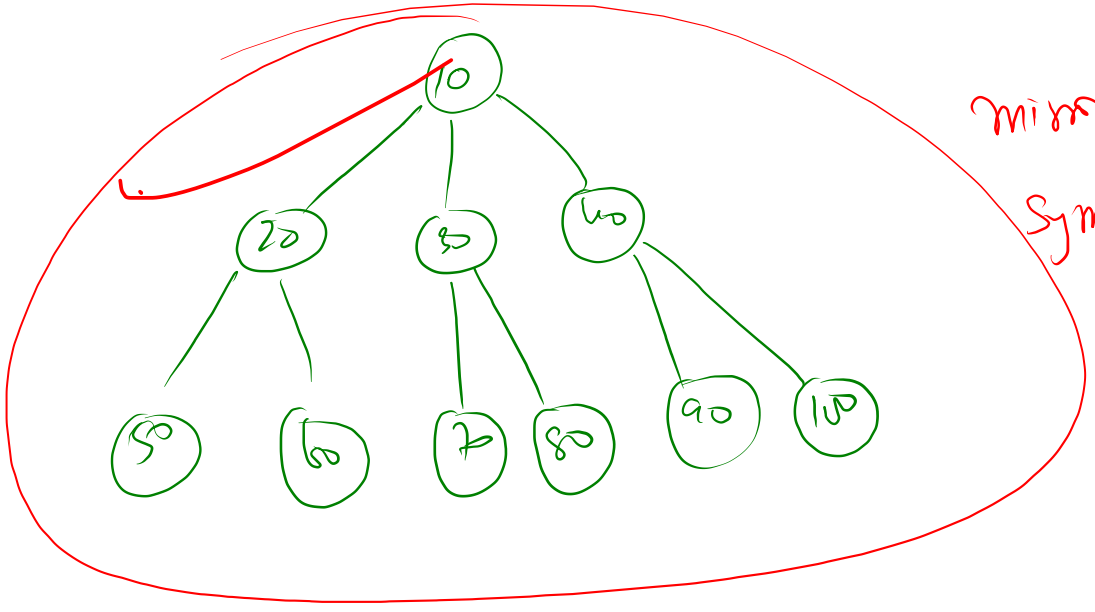




mirror → false

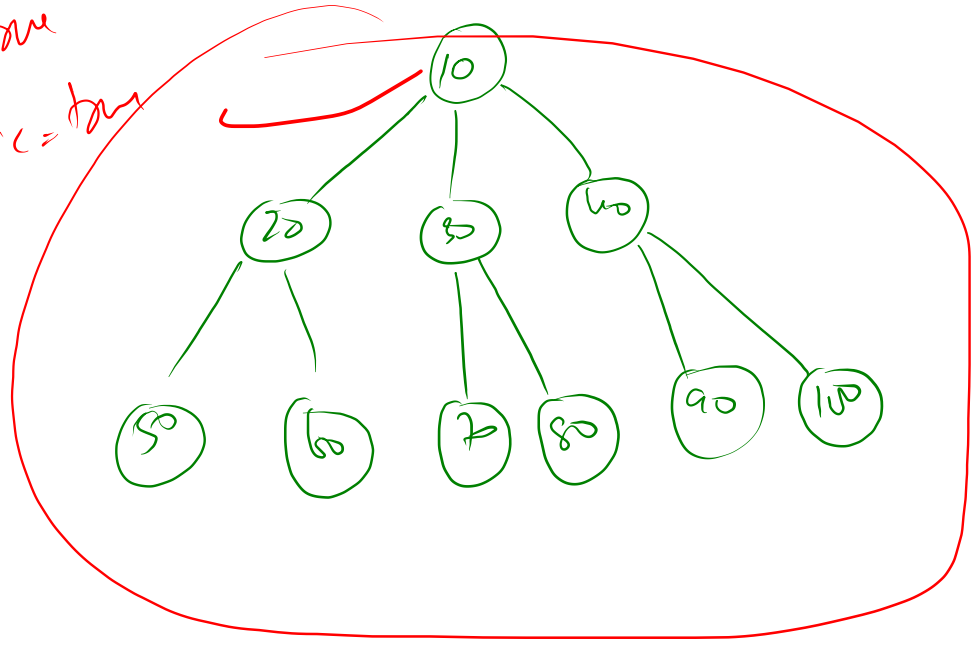
Symmetric → false





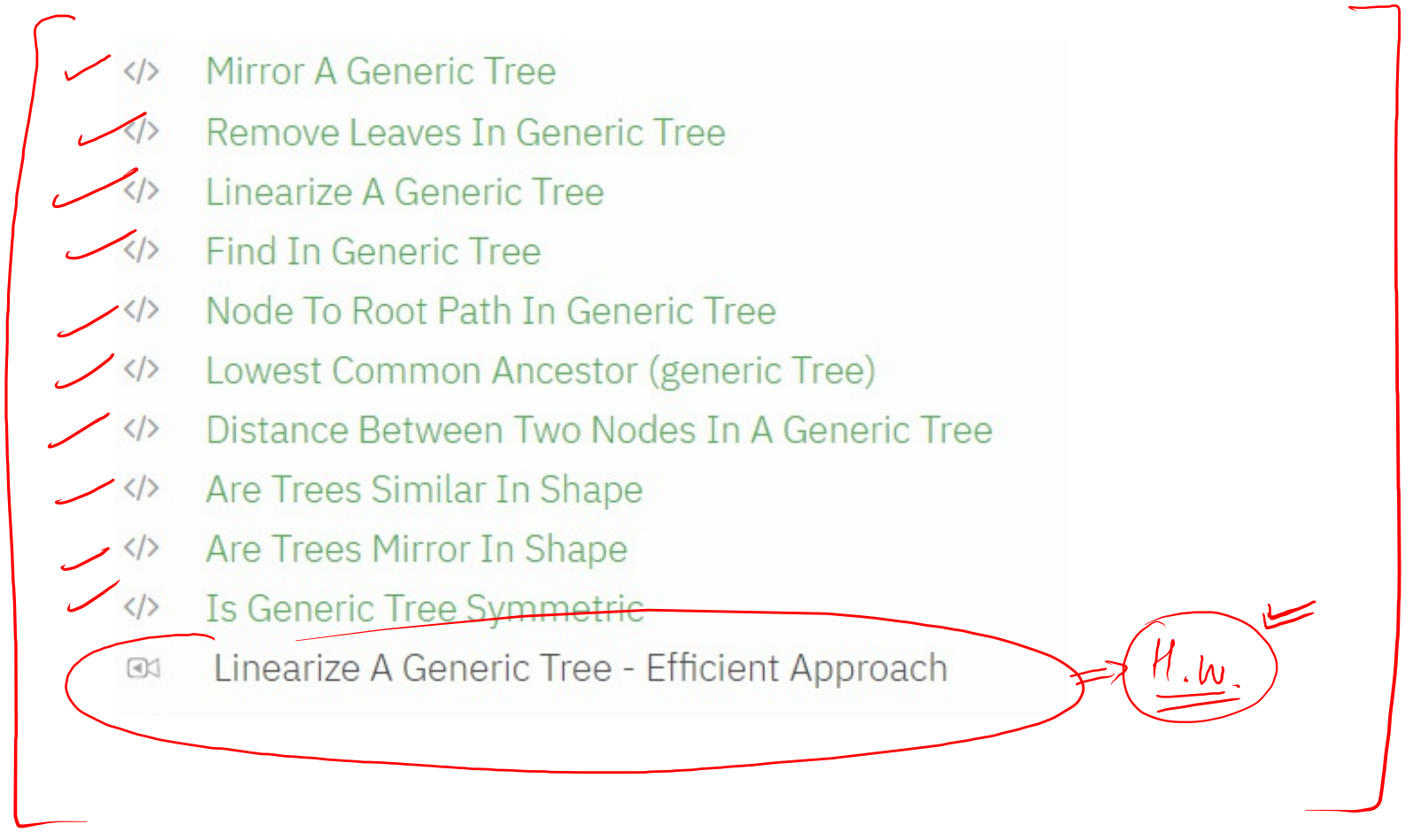
mirror → true

Symmetric = true



```java
public static boolean areSimilar(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()){
        return false;
    }

    for(int i = 0 ; i < n1.children.size() ; i++){
        Node child1 = n1.children.get(i);
        Node child2 = n2.children.get(i);

        boolean res = areSimilar(child1,child2);

        if(res == false){
            return false;
        }
    }

    return true;
}
```

```java
public static boolean areMirror(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()){
        return false;
    }

    for(int i = 0 ; i < n1.children.size() ; i++){
        Node c1 = n1.children.get(i);
        int mIdx = n2.children.size()-1-i;
        Node c2 = n2.children.get(mIdx);

        boolean res = areMirror(c1,c2);
        if(res == false){
            return false;
        }
    }

    return true;
}

public static boolean IsSymmetric(Node node) {
    return areMirror(node,node);
}
```


Mirror A Generic Tree
Remove Leaves In Generic Tree
Linearize A Generic Tree
Find In Generic Tree
Node To Root Path In Generic Tree
Lowest Common Ancestor (generic Tree)
Distance Between Two Nodes In A Generic Tree
Are Trees Similar In Shape
Are Trees Mirror In Shape
Is Generic Tree Symmetric
Linearize A Generic Tree - Efficient Approach
H.W.