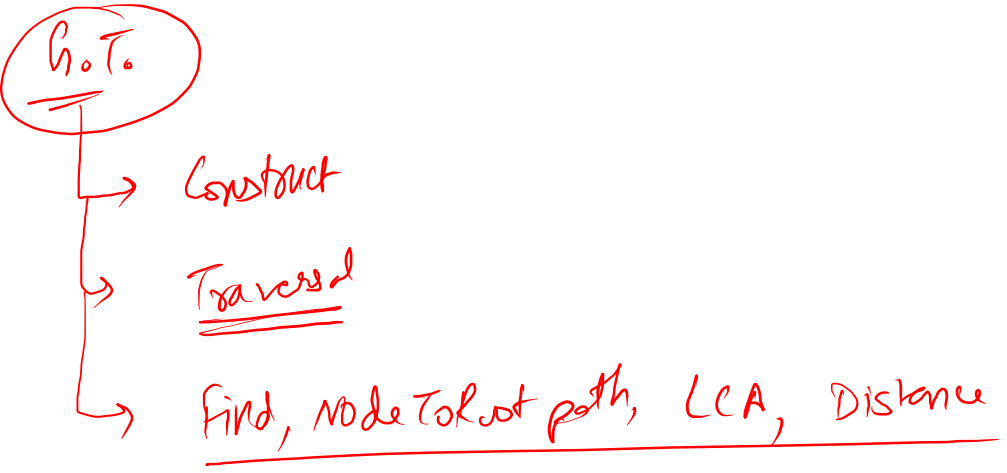
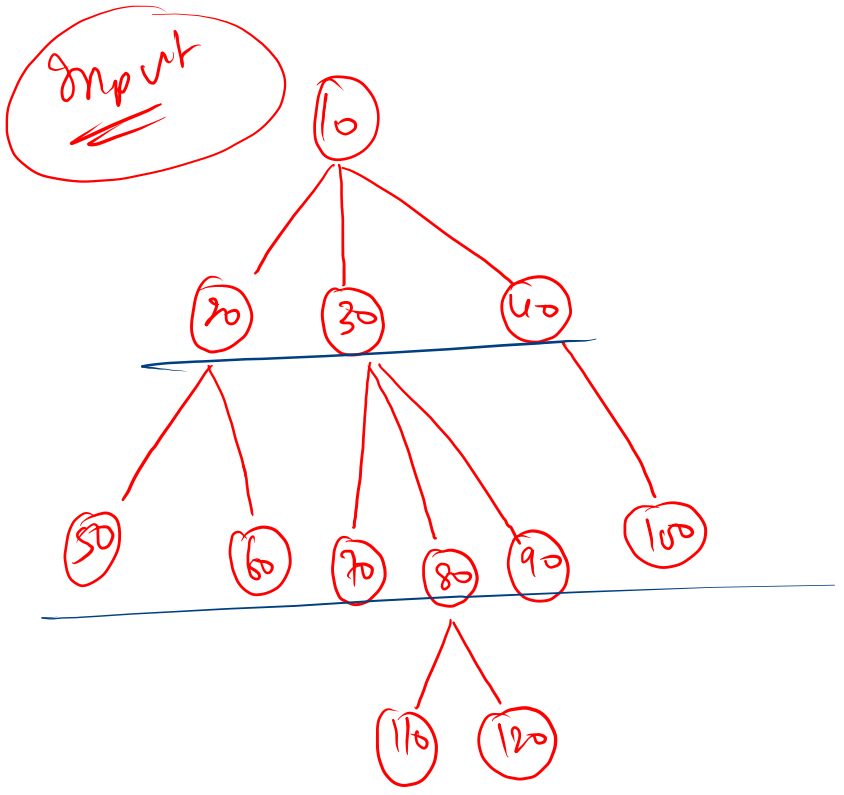
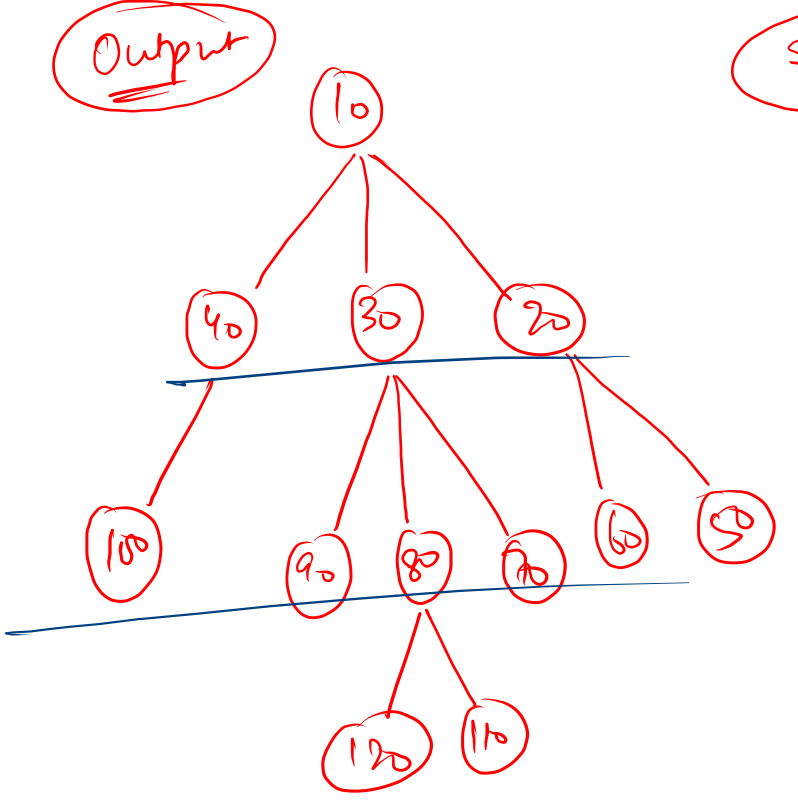


- ✓ Mirror A Generic Tree
- ✓ Remove Leaves In Generic Tree
- ✓ Linearize A Generic Tree
- ✓ Are Trees Similar In Shape
- ✓ Are Trees Mirror In Shape
- ✓ Is Generic Tree Symmetric
- Predecessor And Successor Of An Element
- Ceil And Floor In Generic Tree
- Kth Largest Element In Tree
- Node With Maximum Subtree Sum
- Diameter Of Generic Tree
- ✓ Iterative Preorder And Postorder Of Generic Tree

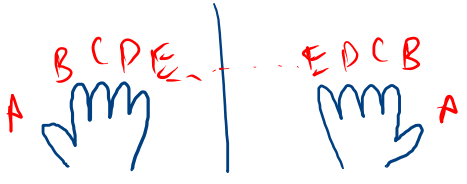


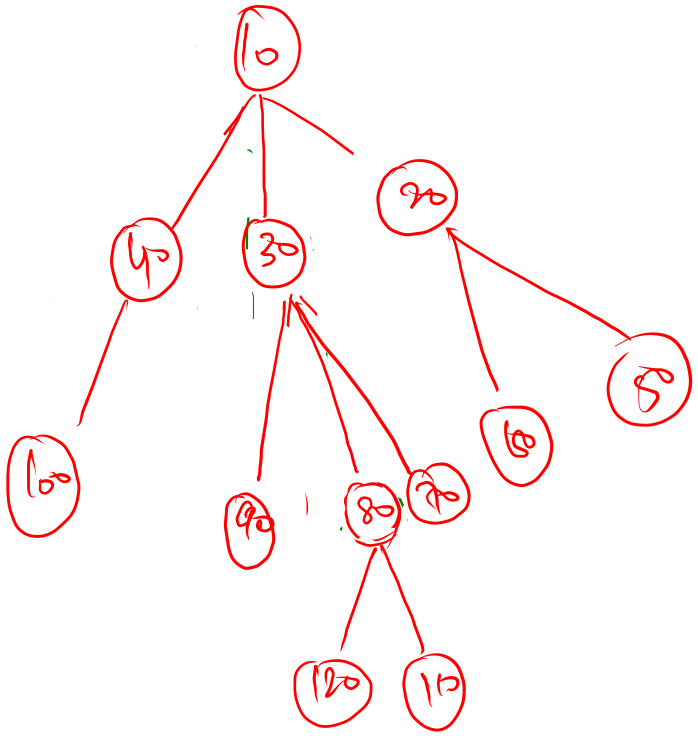


S.C. → O(1)

10

30 ~~10~~ 20 ~~30~~

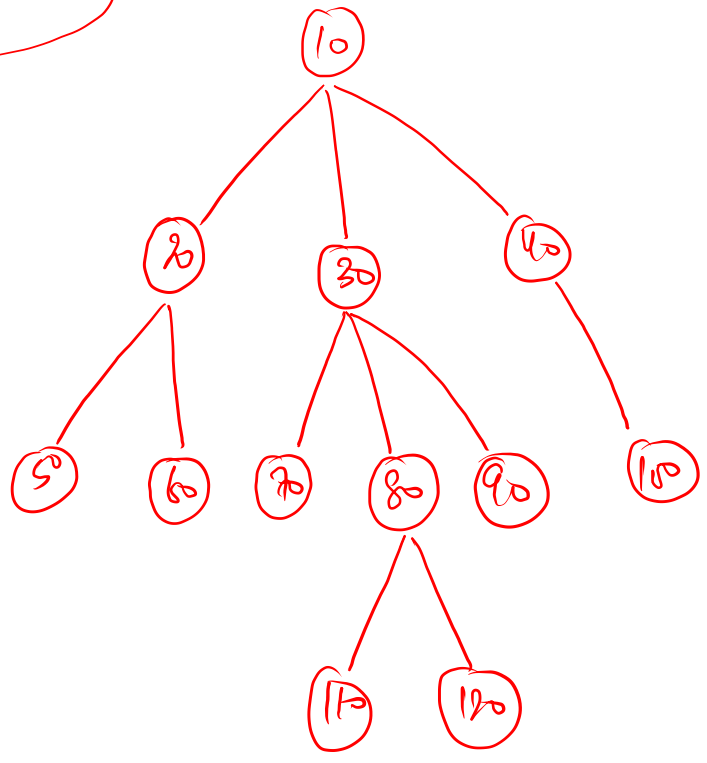




POST ORDER

```
public static void mirror(Node node){
    for(Node child : node.children){
        mirror(child);
    }
    Collections.reverse(node.children);
}
```

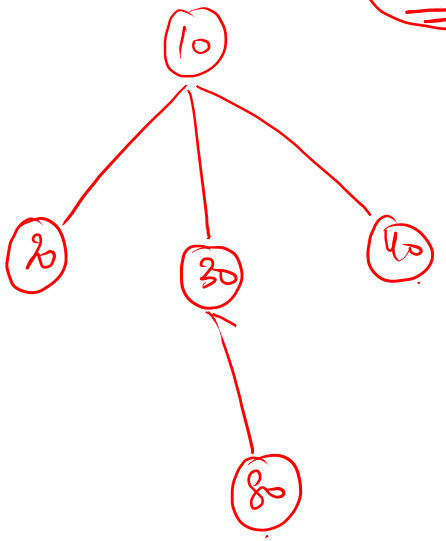
Input



Leaf Nodes

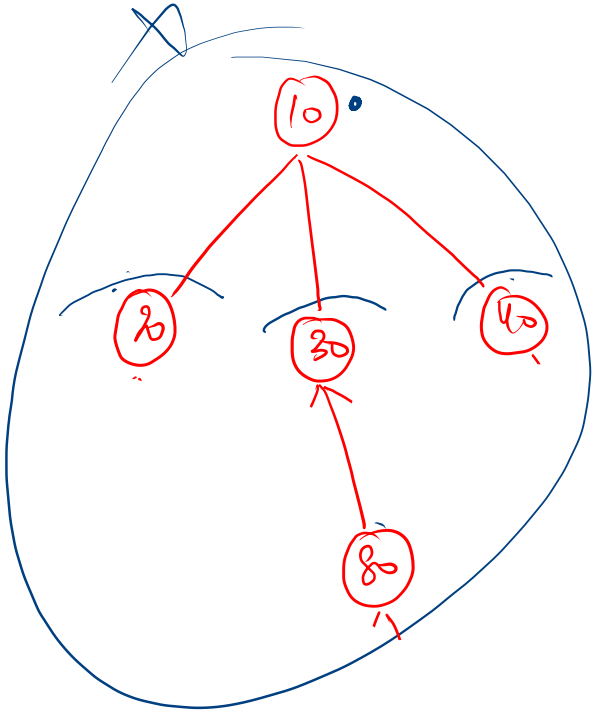
children == 0

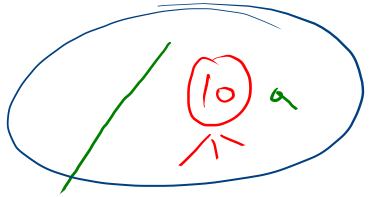
Output



ArrayList ⇒ Remove ⇒ last / previous

Post Order → Remove Leaves



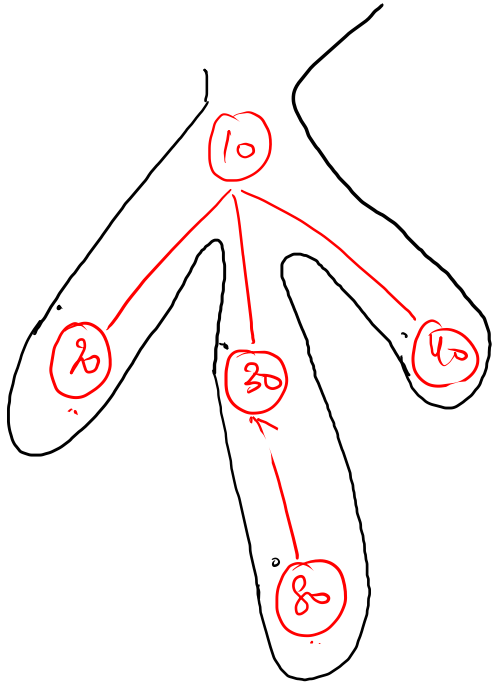


```java
public static void removeLeaves(Node node) {
  for(Node child : node.children){
      removeLeaves(child);
  }

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

Preorder

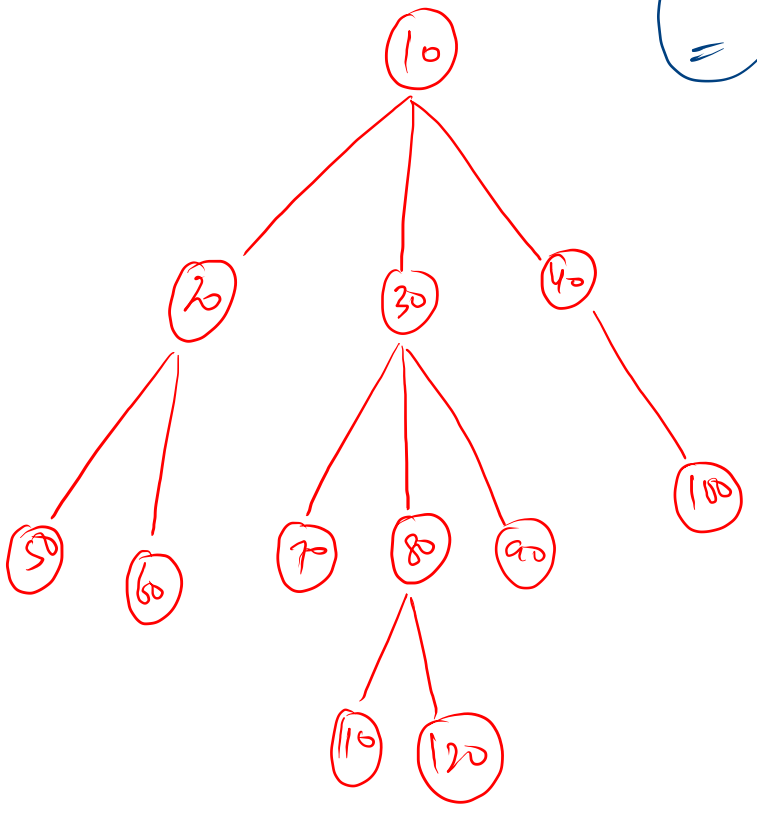
leaf remove

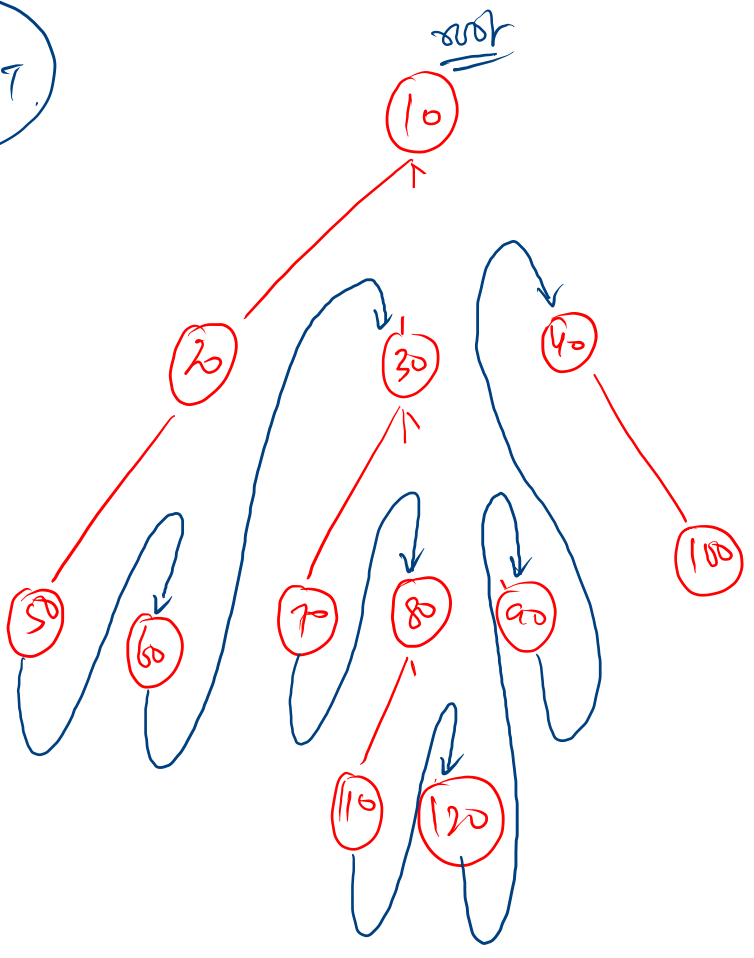
```java
public static void removeLeaves(Node node) {
  for(int idx = node.children.size()-1 ; idx >= 0 ; idx--){
      Node child = node.children.get(idx);
      if(child.children.size() == 0){
          node.children.remove(idx);
      }
  }

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

}
```



Input

GT → Linearize GT

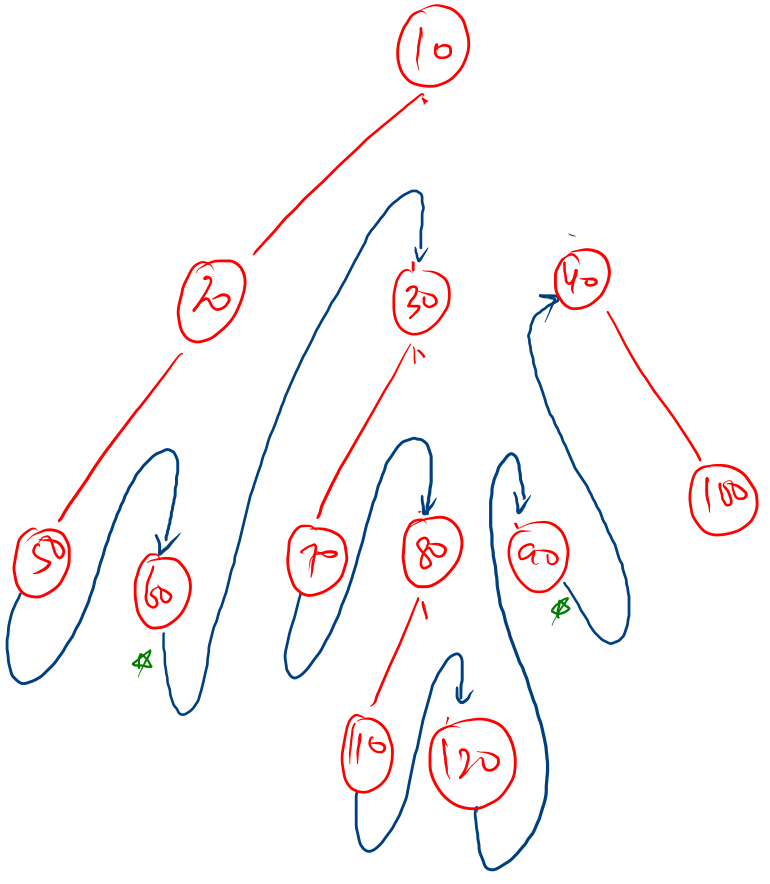


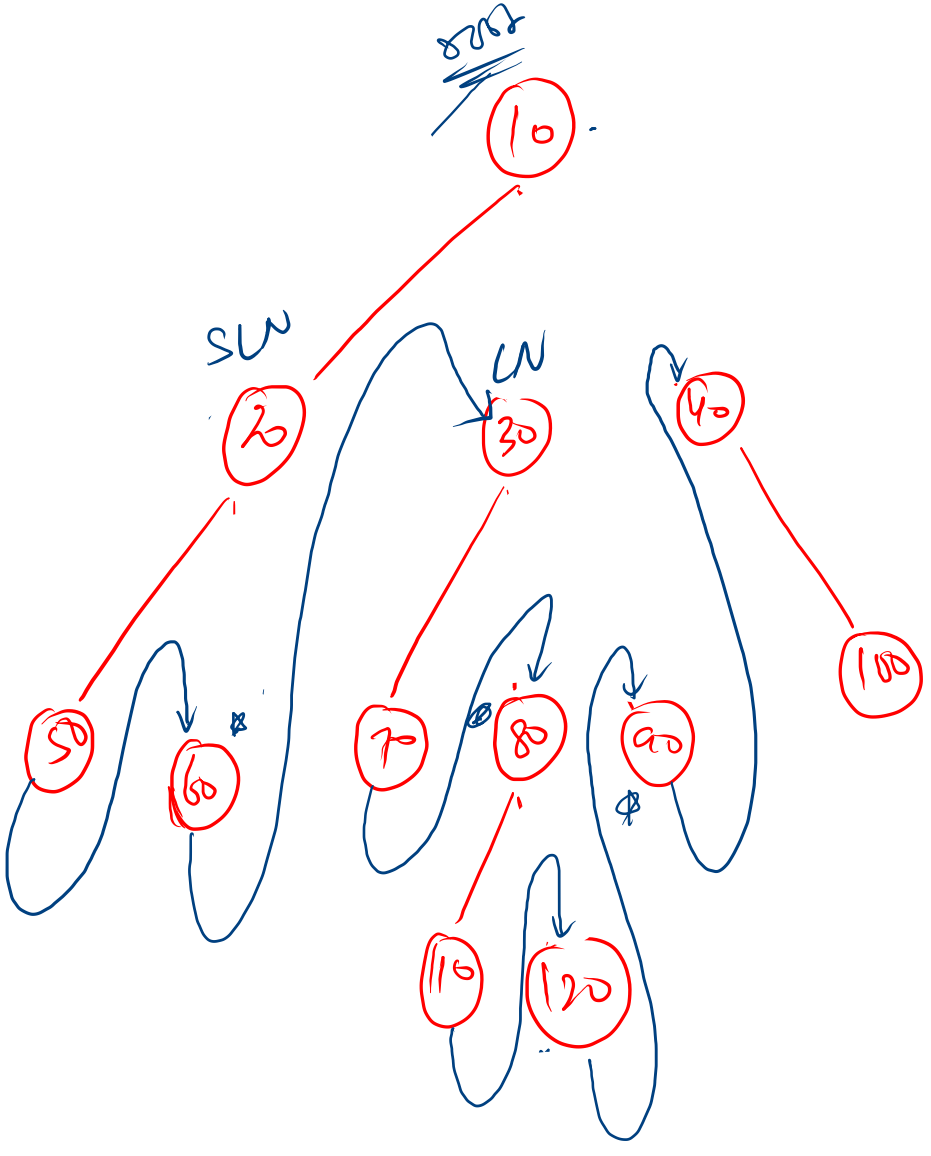


Output

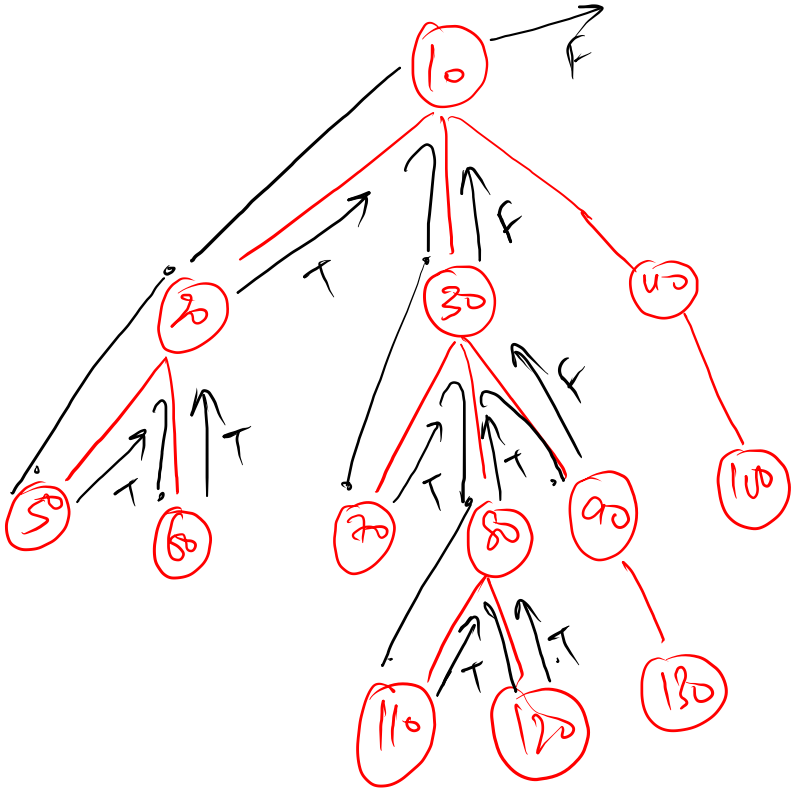
```
24
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1
```

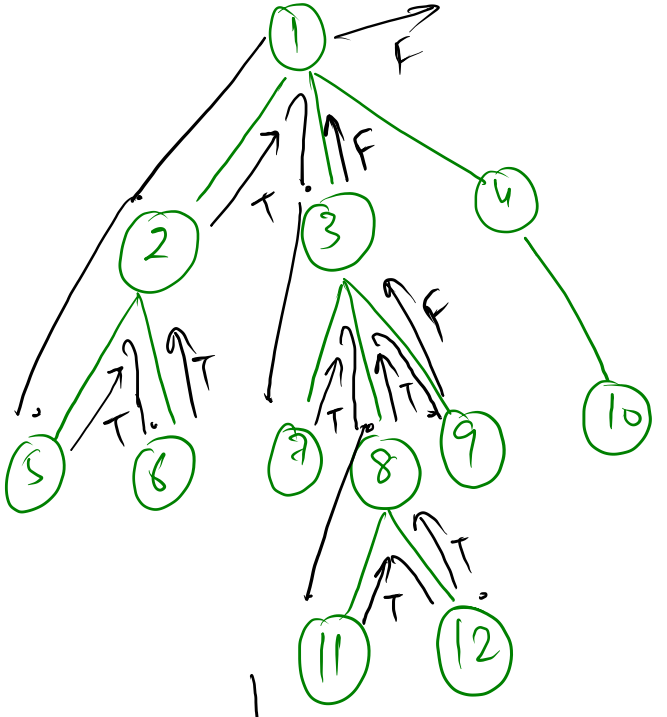
getTail





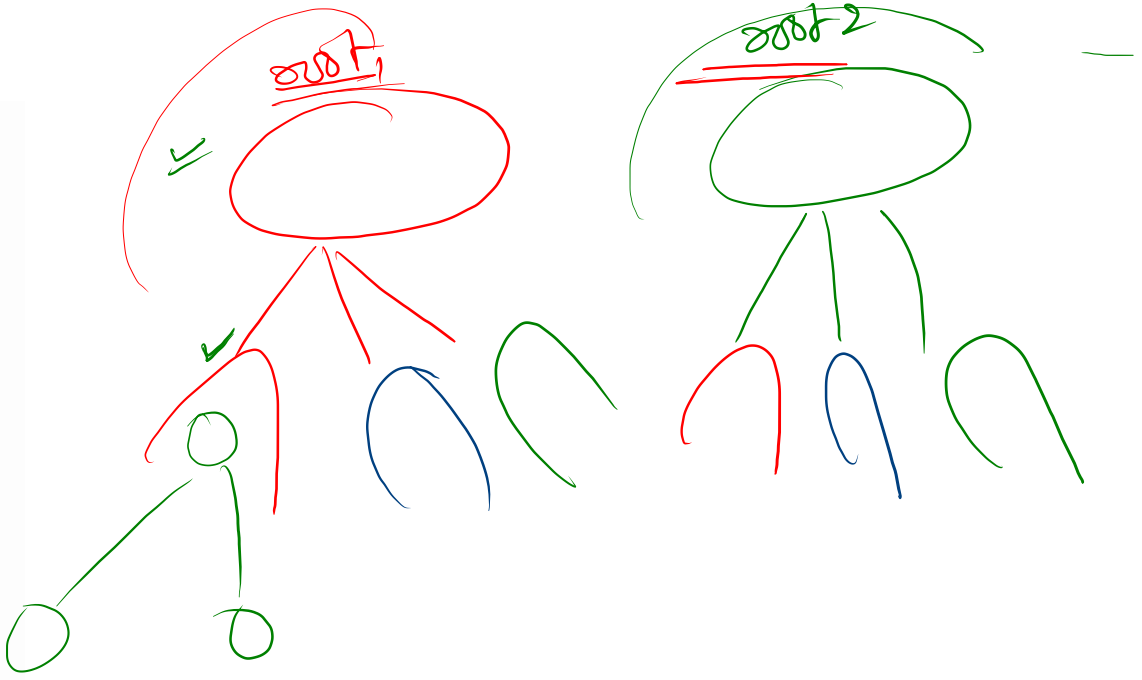

```java
public static Node getTail(Node node){
    while(node.children.size() != 0){
        node = node.children.get(0);
    }
    return node;
}
public static void linearize(Node node){
  for(Node child : node.children){
      linearize(child);
  }

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

      Node tail = getTail(sln);
      tail.children.add(ln);
  }
}
```

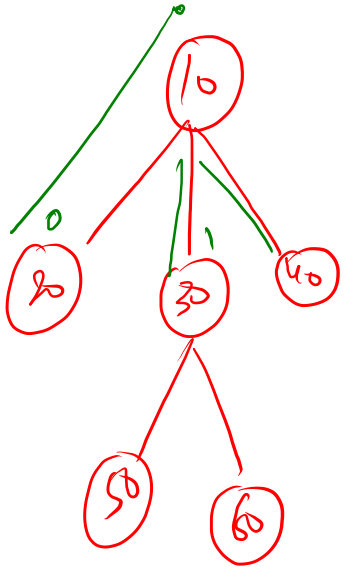


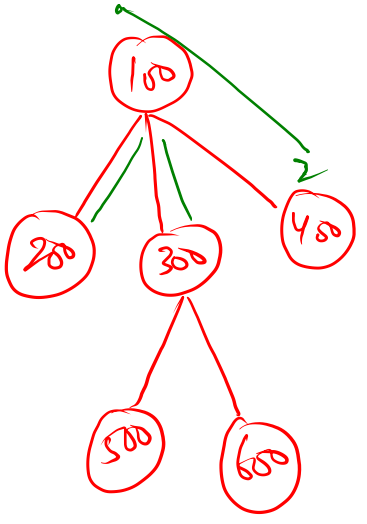



```java
public static boolean areSimilar(Node n1, Node n2) {
  if(n1.children.size() != n2.children.size()){
      return false;
  }

  for(int idx = 0 ; idx <n1.children.size() ; idx++){
      Node c1 = n1.children.get(idx);
      Node c2 = n2.children.get(idx);
      if(areSimilar(c1,c2) == false){
          return false;
      }
  }

  return true;
}
```

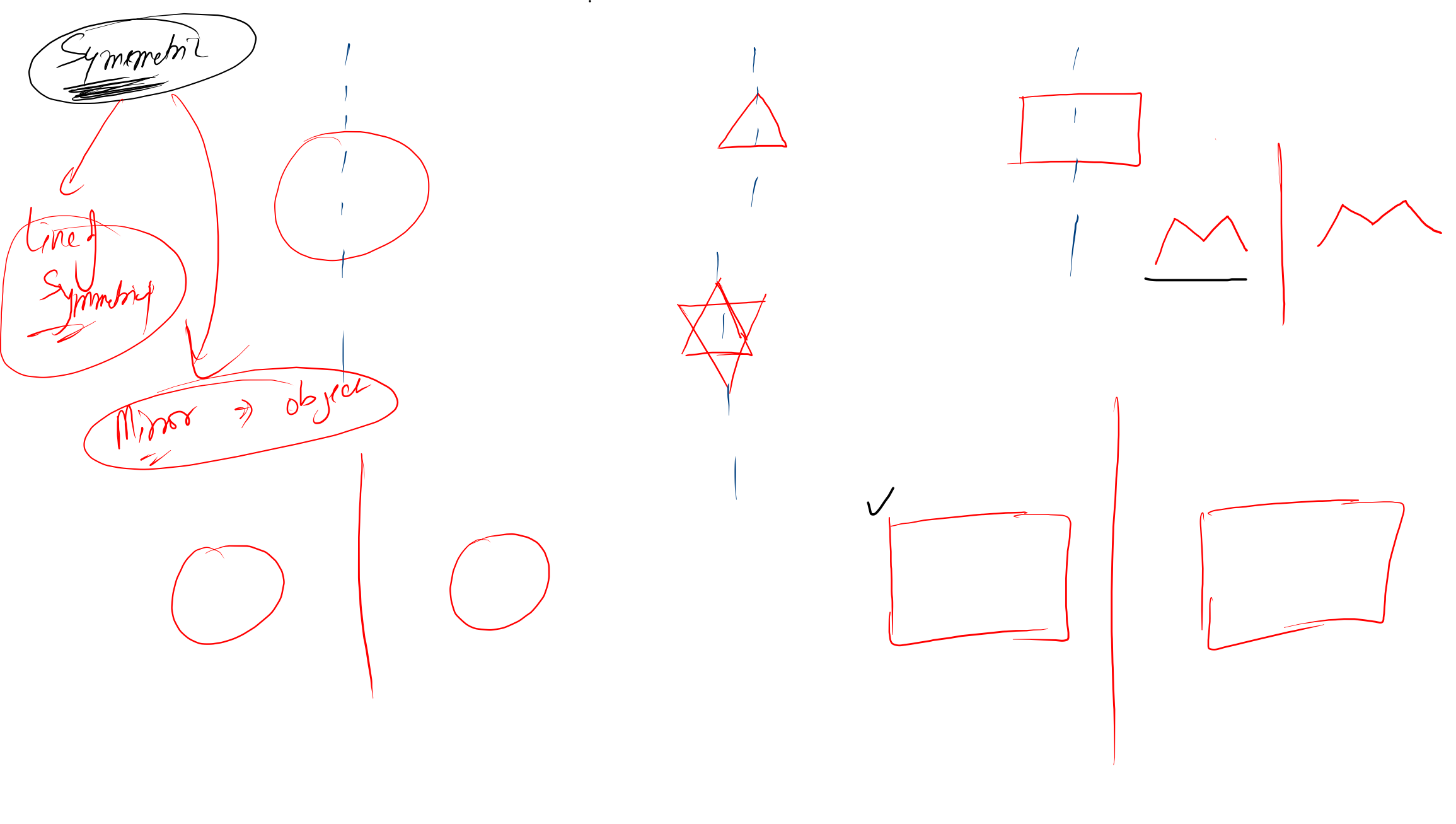
```java
public static boolean areSimilar(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()){
        return false;
    }

    for(int idx = 0 ; idx <n1.children.size() ; idx++){
        Node c1 = n1.children.get(idx);
        Node c2 = n2.children.get(idx);
        if(areSimilar(c1,c2) == false){
            return false;
        }
    }

    return true;
}
```

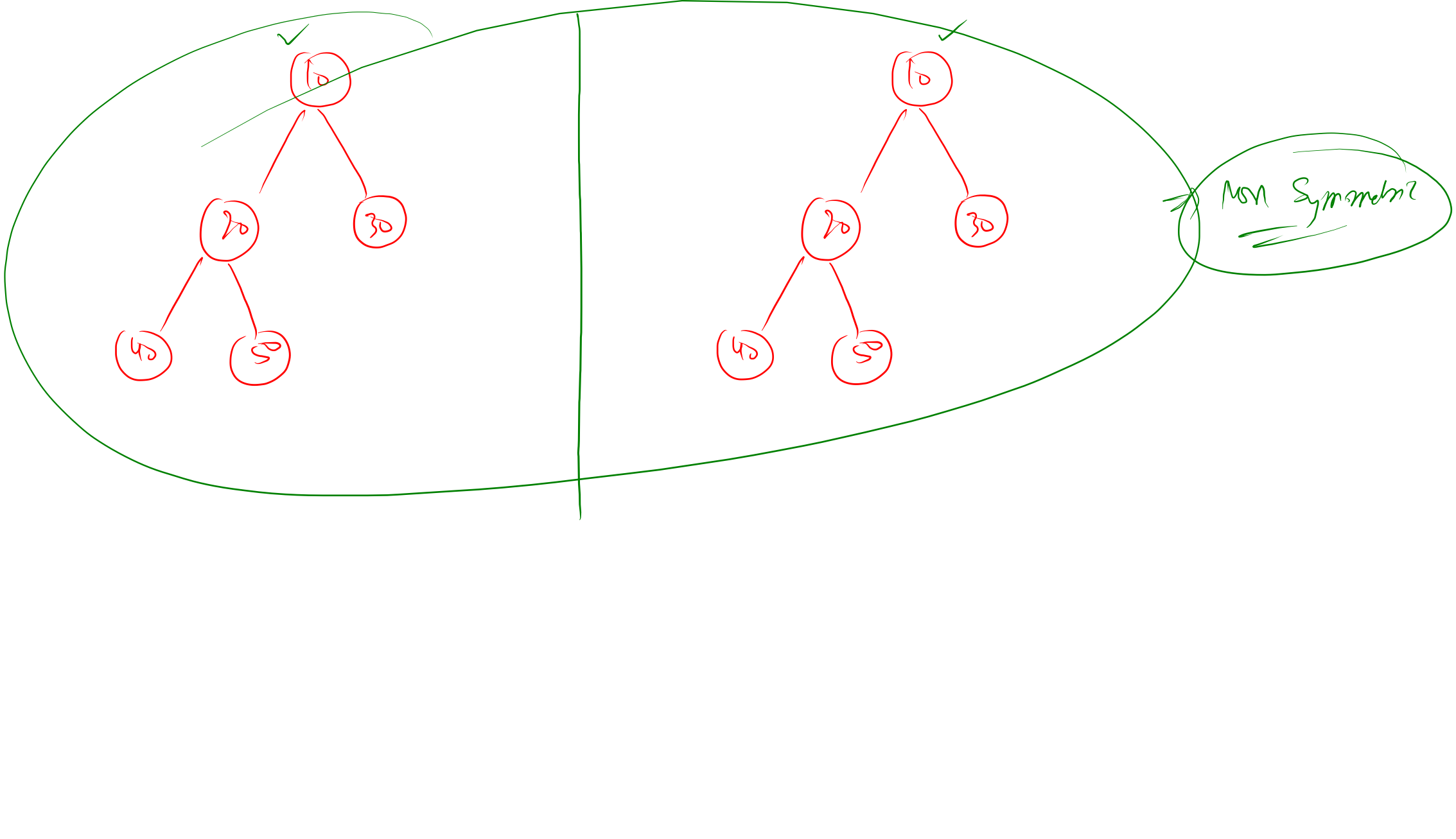


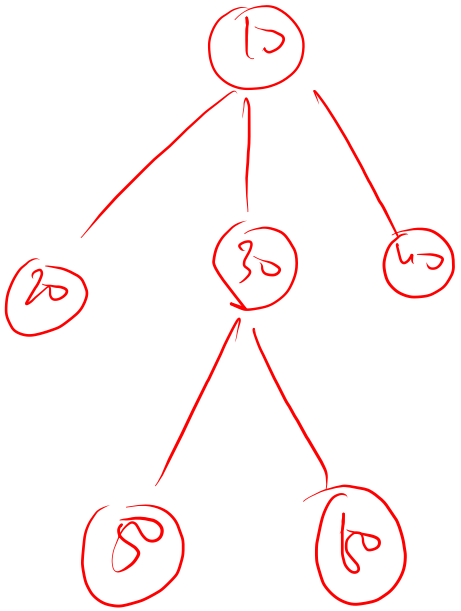


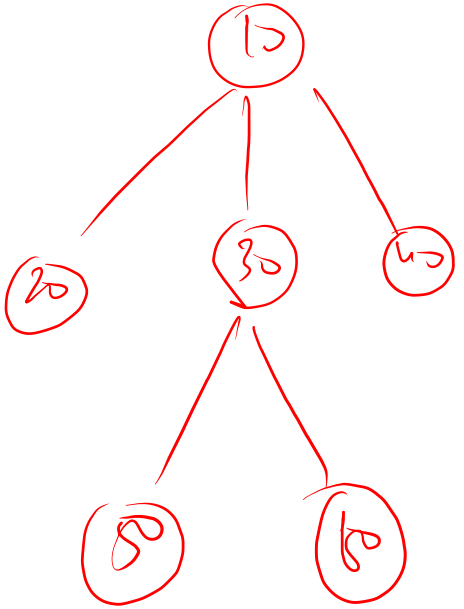



```java
public static boolean areMirror(Node n1, Node n2) {
  if(n1.children.size() != n2.children.size()){
      return false;
  }

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

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

  return true;
}
```


Symmetry
Line Symmetry
Mirror ⇒ object


10
20
30
40
5
10
20
30
40
5
Non Symmetric





mirror →

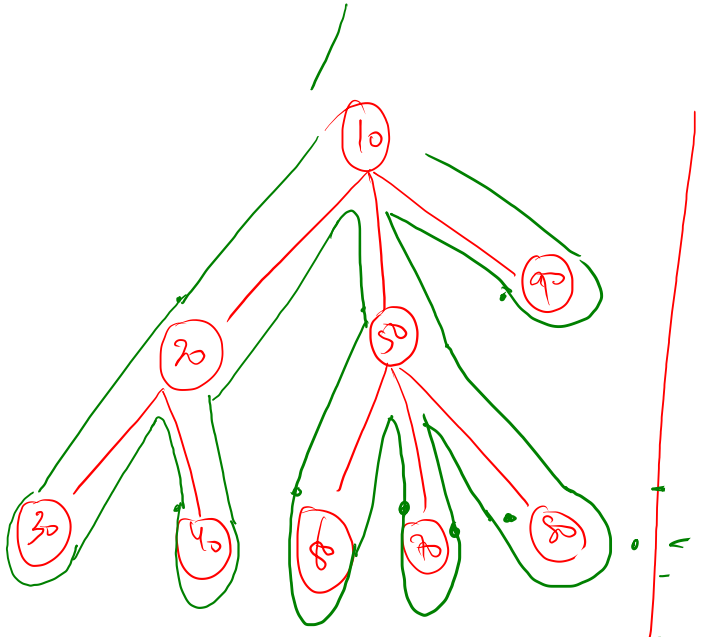
Symmetric

recursion ⇒ Iteration

-1 ⇒ pre

state == numOfChild ⇒ post

state ⇒ child idx




```java
public static void IterativePreandPostOrder(Node node) {
  Stack<Pair> st = new Stack<>();
  String pre = "" , post = "";

  st.push(new Pair(node,-1));
  while(st.size() > 0){
      Pair top = st.peek();

      if(top.state == -1){
          // pre
          pre += top.node.data+" ";
          top.state++;
      }else if(top.state == top.node.children.size()){
          // post
          post += top.node.data+" ";
          st.pop();
      }else{
          int idx =top.state;
          Node child = top.node.children.get(idx);
          top.state++;
          st.push(new Pair(child,-1));
      }
  }

  System.out.println(pre+"\n"+post);
}
```

{ node, state }

Pre ⇒ 10 20 30 40 50 60 70 80 90

Post ⇒ 30 40 20 60 70 80 50 90 10

```java
public static class Pair{
    int state;
    Node node;
    Pair(Node node,int state){
        this.node = node;
        this.state = state;
    }
}
```