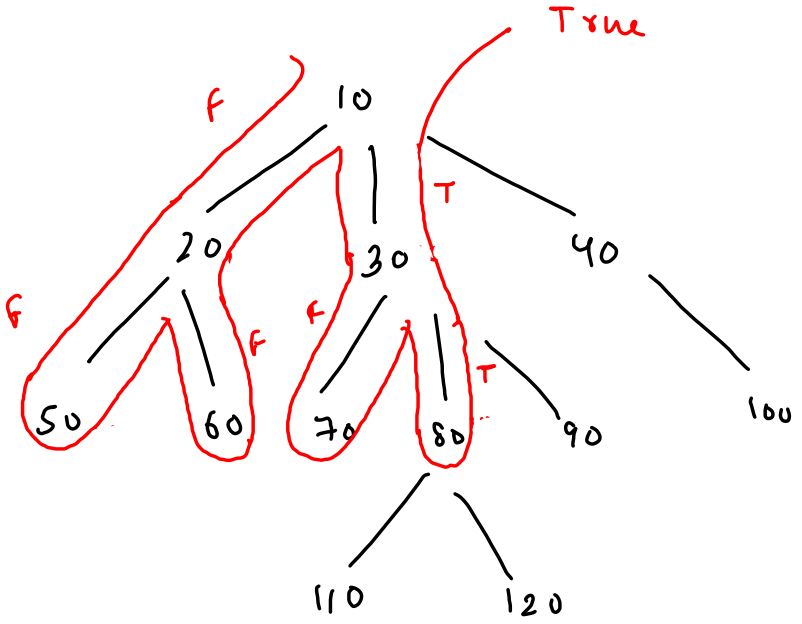
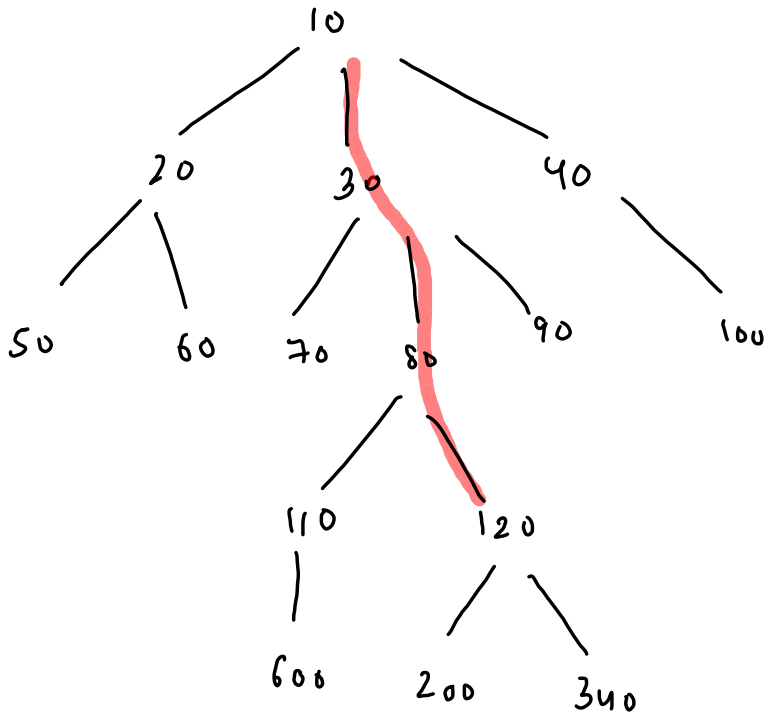
```java
public static boolean find(Node node, int data) {
    //self check
    if(node.data == data) {
        return true;
    }

    //faith on each child
    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);

        boolean ificf = find(child,data); //is found in child family

        if(ificf == true) {
            return true;
        }
    }


    return false;
}
```



data = 80

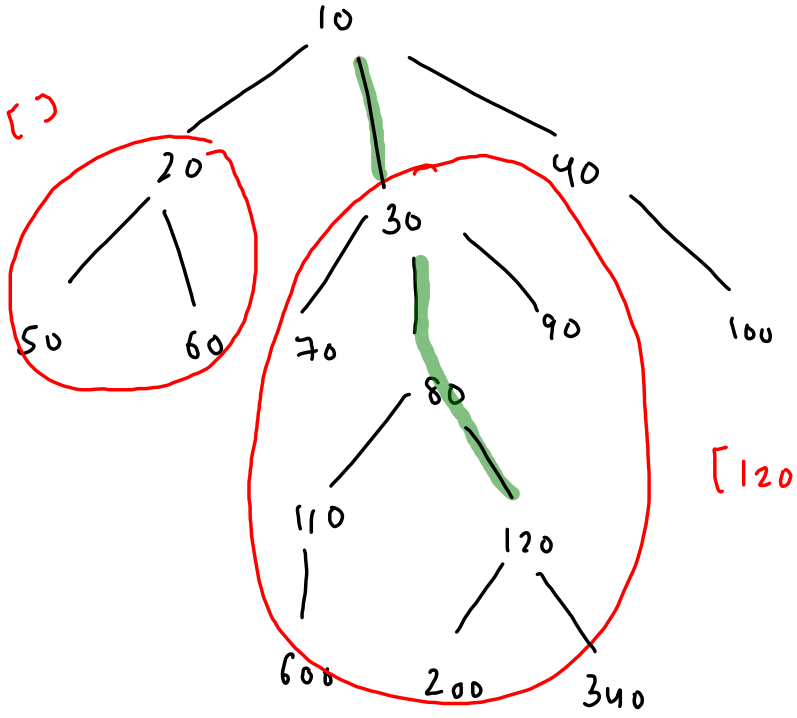


node to root path

data = 120

[120, 80, 30, 10]

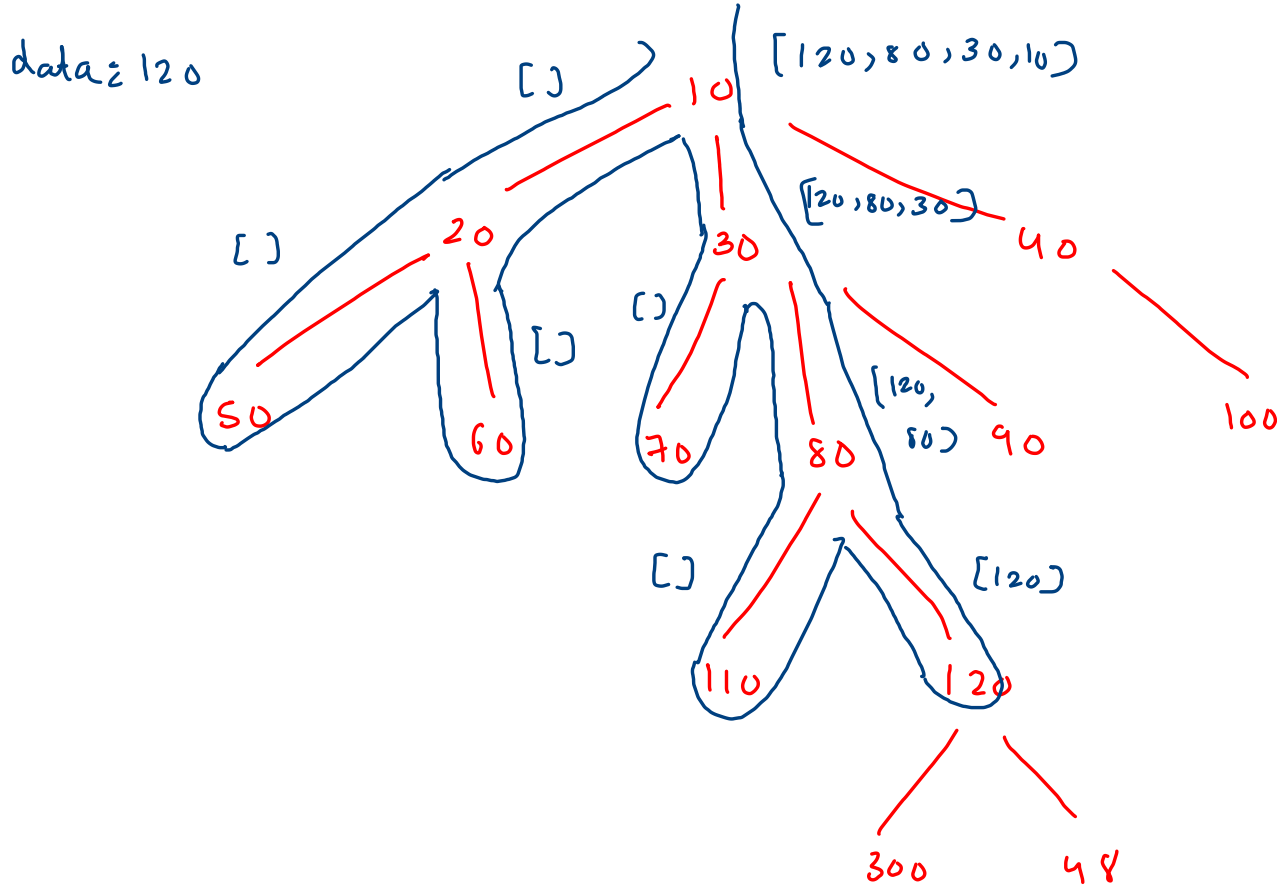
[120, 80, 30, 10] -> n2rp



data = 120

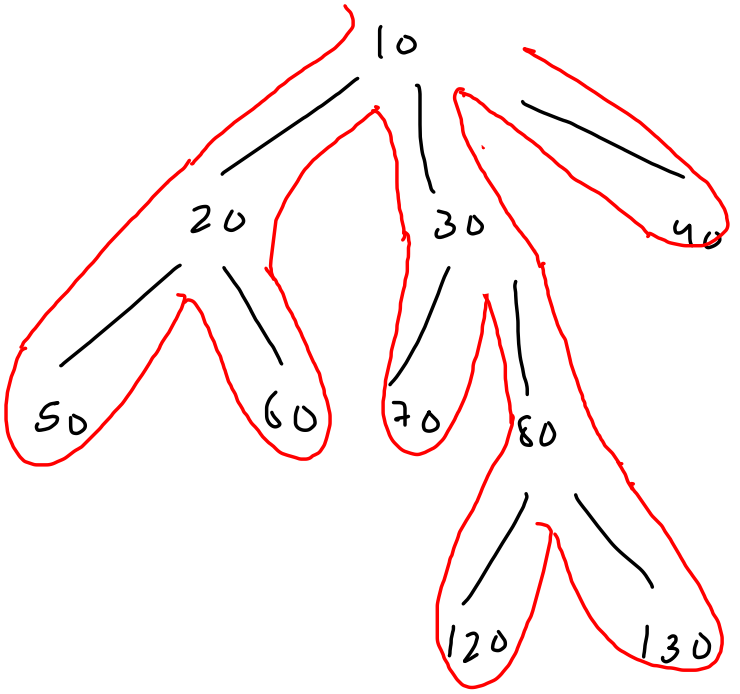
[120, 80, 30) = n2cp

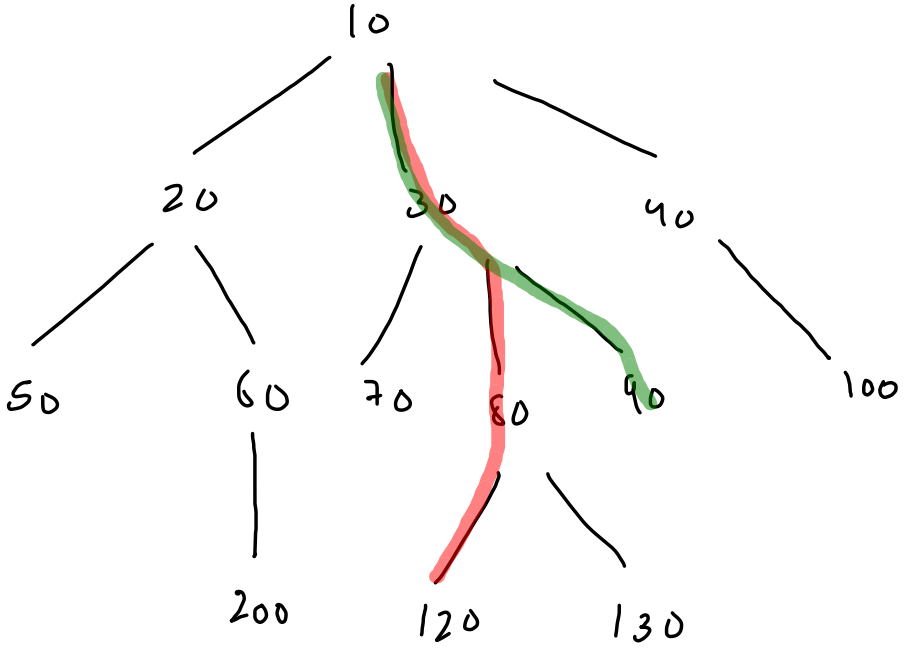
```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
    //self check
    if(node.data == data) {
        ArrayList<Integer>list = new ArrayList<>();
        list.add(node.data);
        return list;
    }

    //faith on each child
    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);

        ArrayList<Integer>n2cp = nodeToRootPath(child,data); //node to child path

        if(n2cp.size() > 0) {
            //data exists in child family
            n2cp.add(node.data); //node to child path -> node to root path
            return n2cp;
        }
    }

    return new ArrayList<>();

}
```



data = 100 [ ]

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

    //faith on each child
    for(int i=0; i < node.children.size();i++) {
        Node child = node.children.get(i);

        ArrayList<Integer>n2cp = nodeToRootPath(child,data); //node to child path

        if(n2cp.size() > 0) {
            //data exists in child family
            n2cp.add(node.data); //node to child path -> node to root path
            return n2cp;
        }
    }

    return new ArrayList<>();

}
```





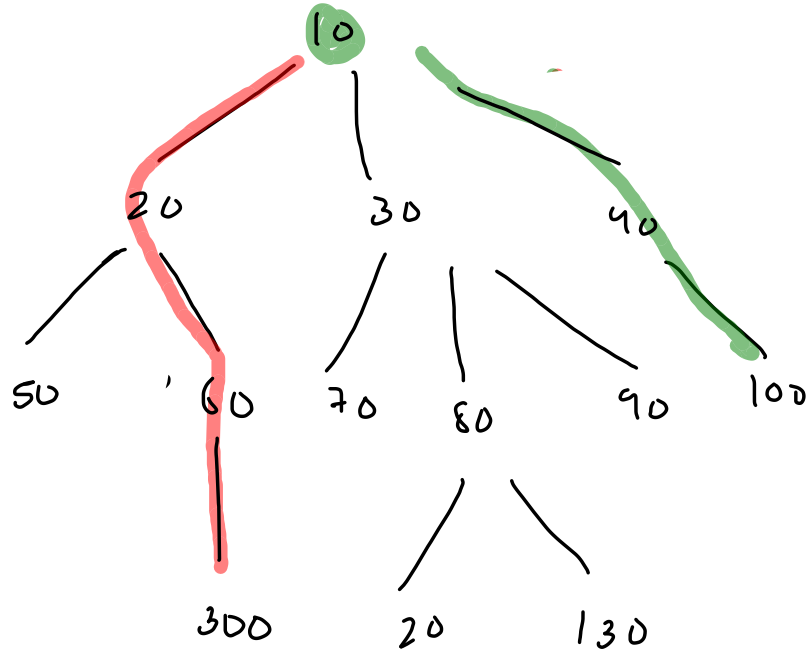
lowest common ancestor

90 -> [90, 30, 10]

i

120 -> [120, 80, 30, 10]

j



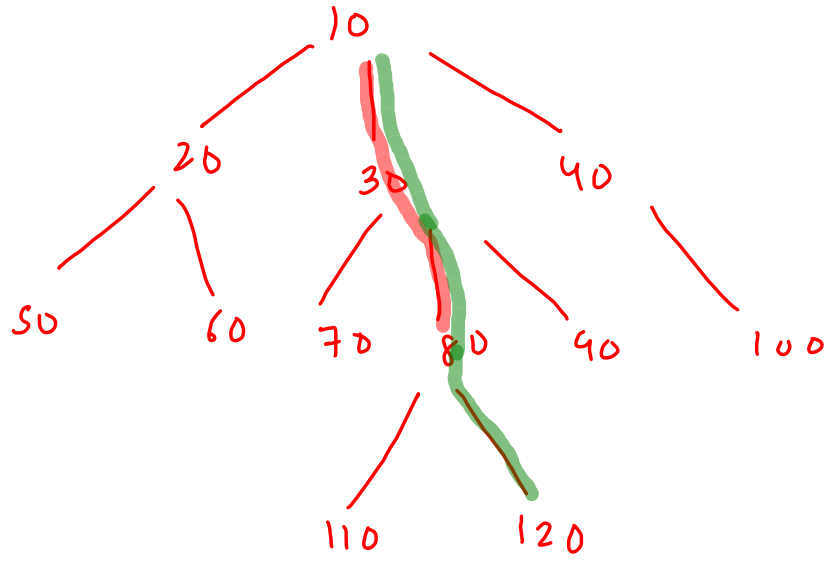
d1 -> 100

d2 -> 300

n2p1 = [100, 40, 10]
i ✗

n2p2 = [300, 60, 20, 10]
j ✗

10



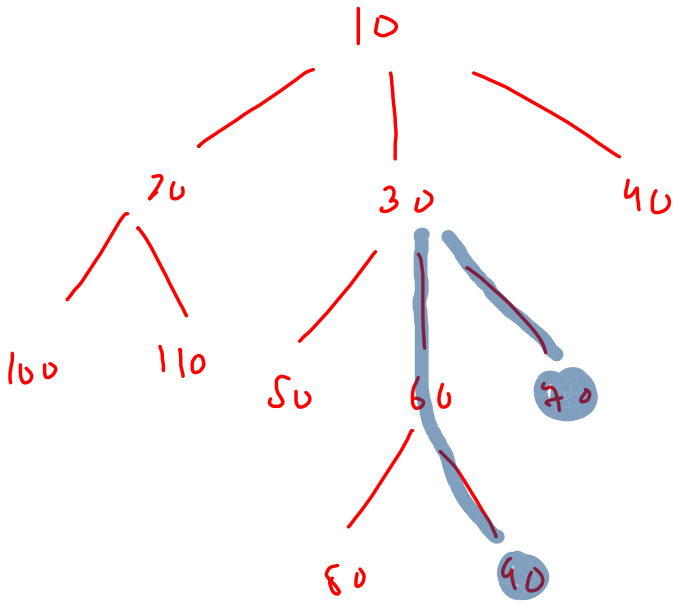
$d_1 = 80$

$d_2 = 120$

n2xp1 = [80, 30, 10]

n2xp2 = [120, 80, 30, 10]

draw, 80




```java
public static int lca(Node node, int d1, int d2) {
  // write your code here
  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);
}
```

d1 = 70

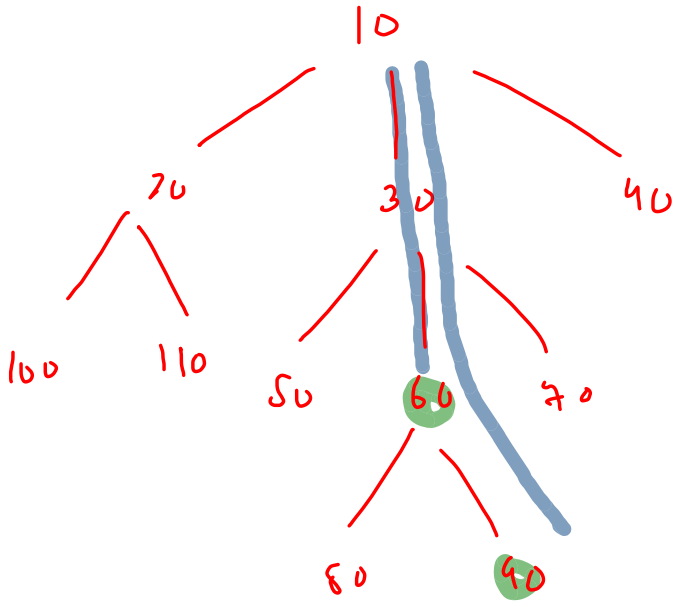
d2 = 90

30

p1 = [70, 30, 10]

p2 = [90, 60, 30, 10]

i i i



60

lca

```java
public static int lca(Node node, int d1, int d2) {
  // write your code here
  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);
}
```

d1 = 60

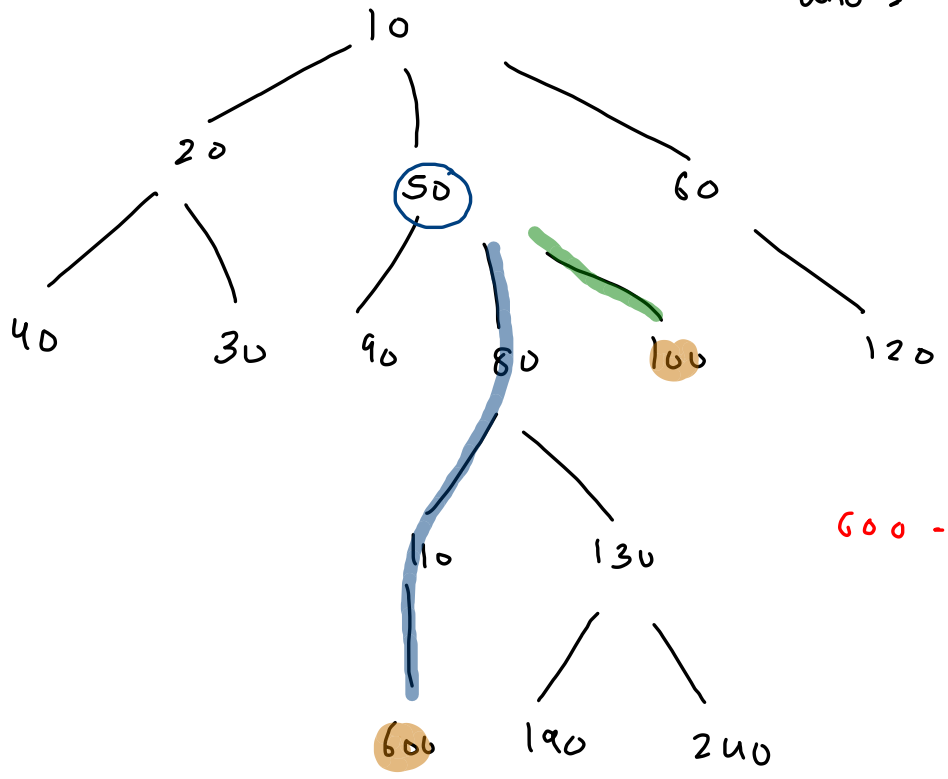
d2 = 90

p1 = [60, 30, 10]

i

p2 = [90, 60, 30, 10]

j



ans = (i+1) + (j+1)

(i+1) -> d1 to lca

(j+1) -> d2 to lca

d1 = 600

d2 = 100

d1 lca

600 -> [600, 110, 80, 50, 10)

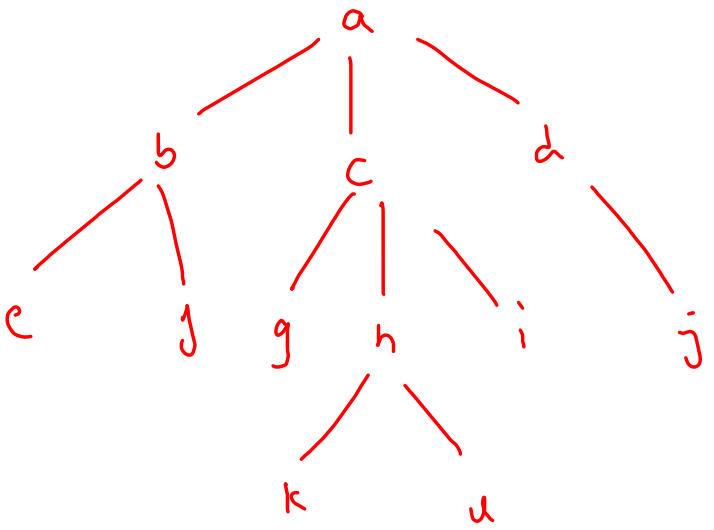
i i i

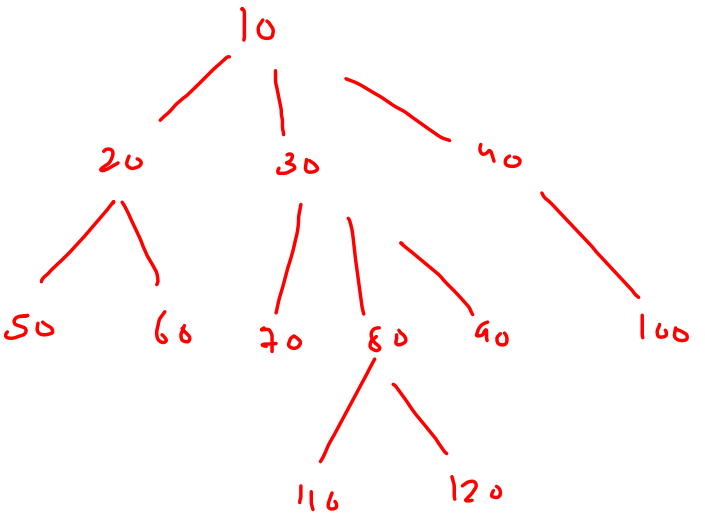
0 i+1 lca -> i+1

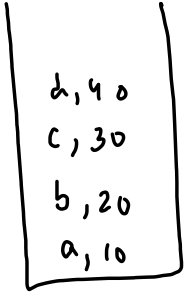
100 -> [100, 50, 10)

j j j

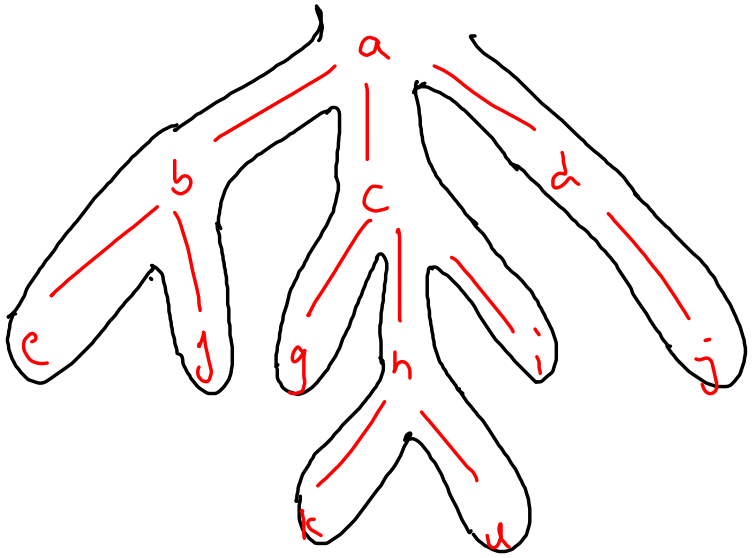
d2 lca

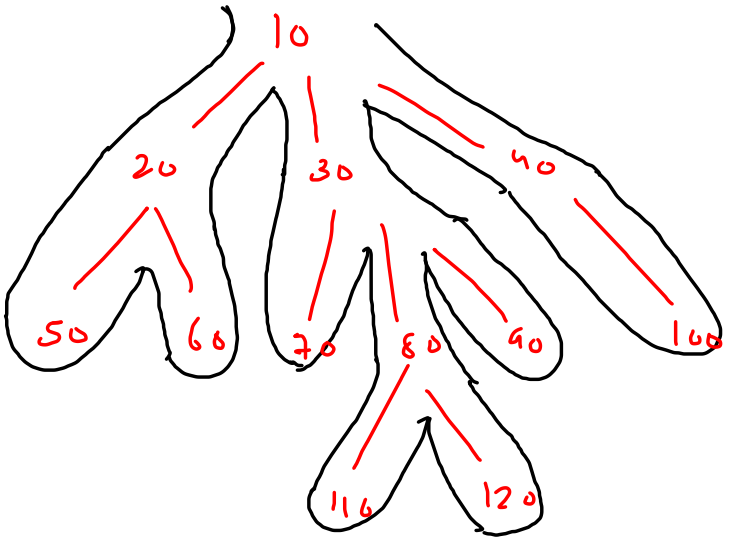




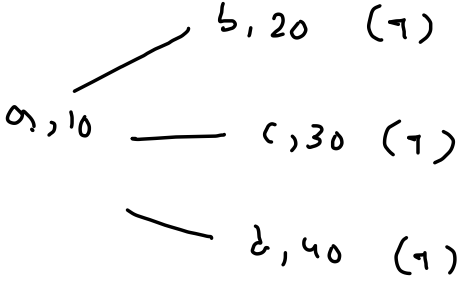


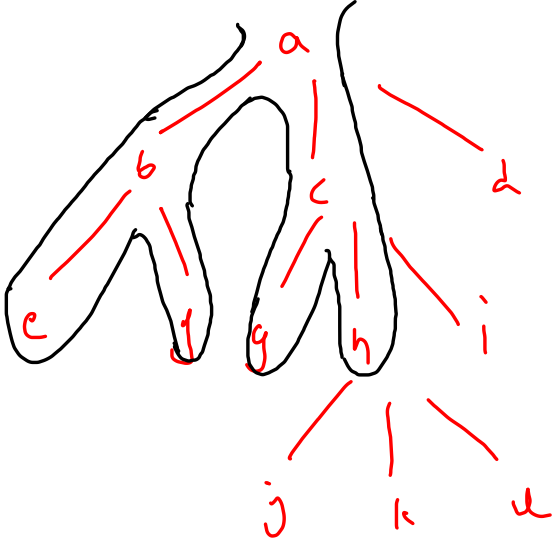
Similion : structure same

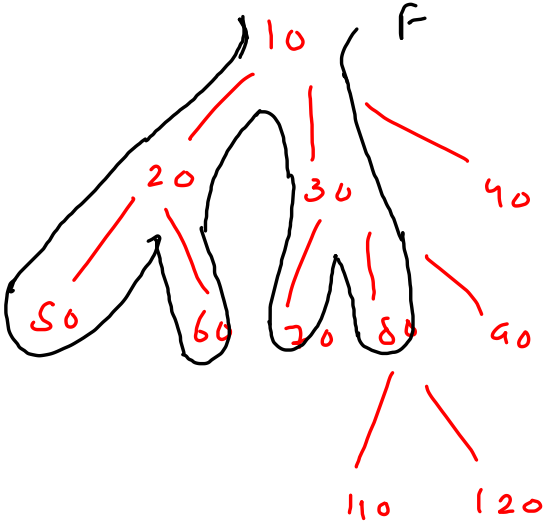




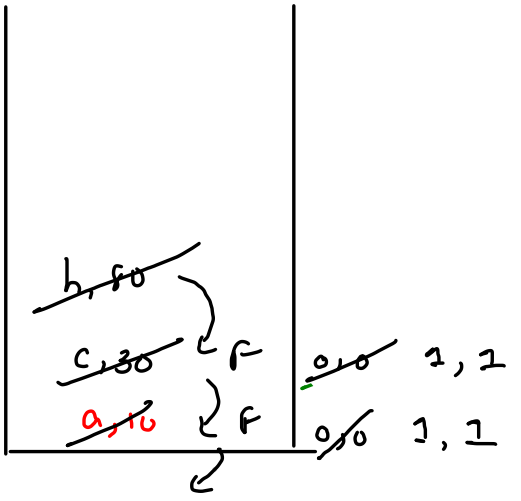

```java
public static boolean areSimilar(Node n1, Node n2) {
    //self check
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

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

        boolean areSimilarChild = areSimilar(c1,c2);

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

    return true;
}
```

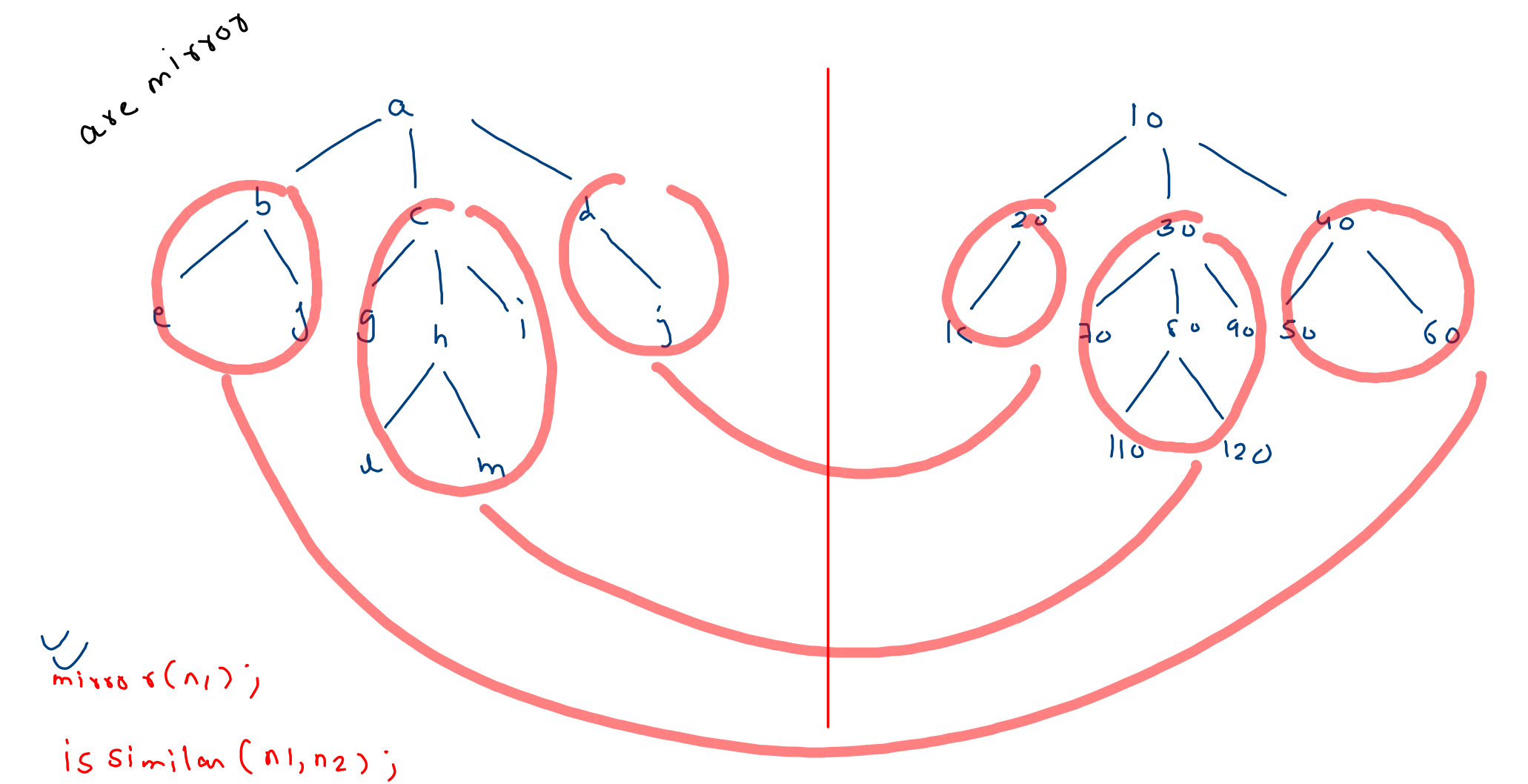






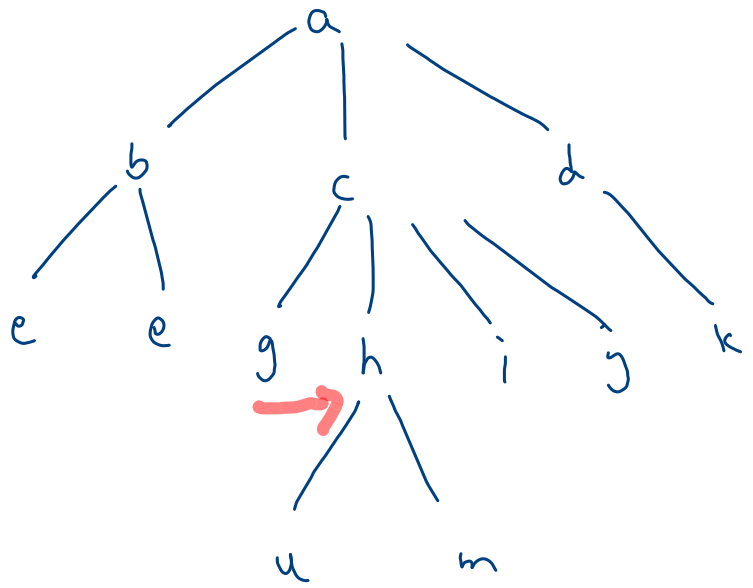

```java
public static boolean areSimilar(Node n1, Node n2) {
    //self check
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

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

        boolean areSimilarChild = areSimilar(c1,c2);

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

    return true;
}
```

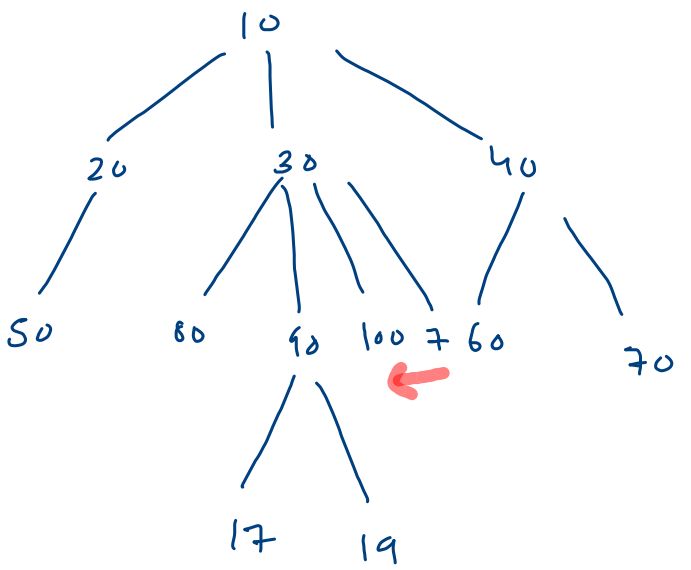


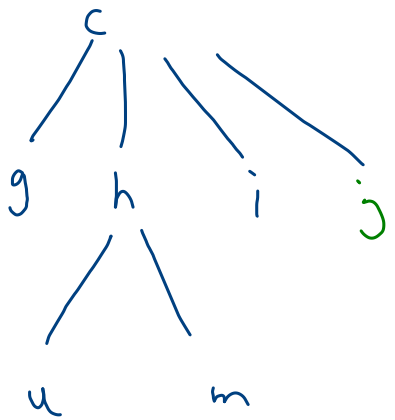


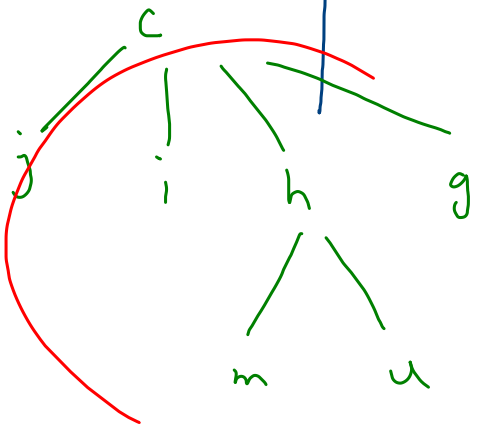
mirror(n1);

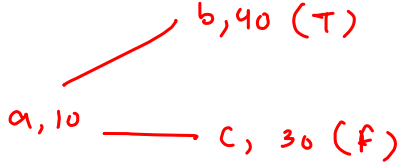
isSimilar(n1, n2);

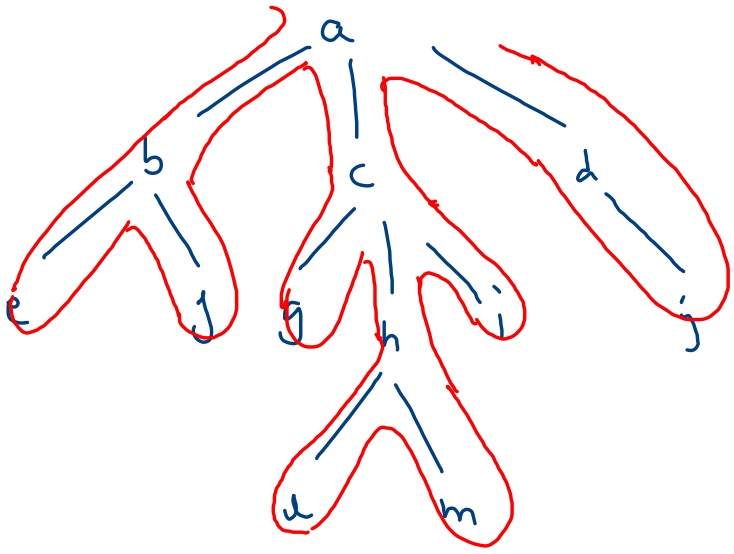

a
b
c
d
e
e
g
h
i
j
k
u
m

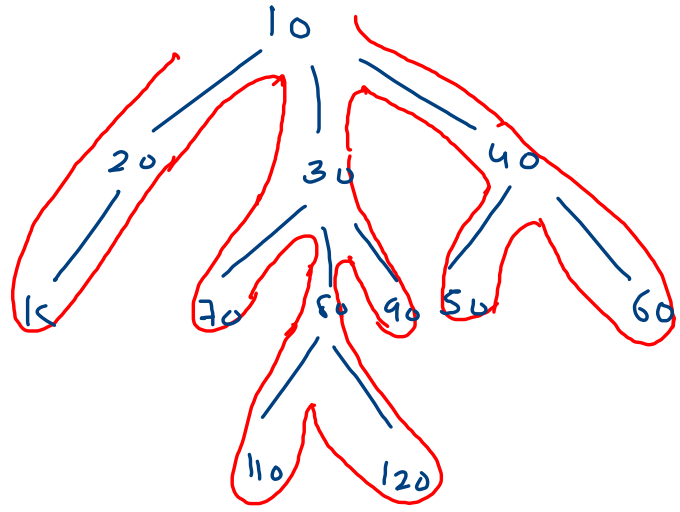

10
20
30
40
50
60
90
100
60
70
17
19


c
g
h
i
j
u
m

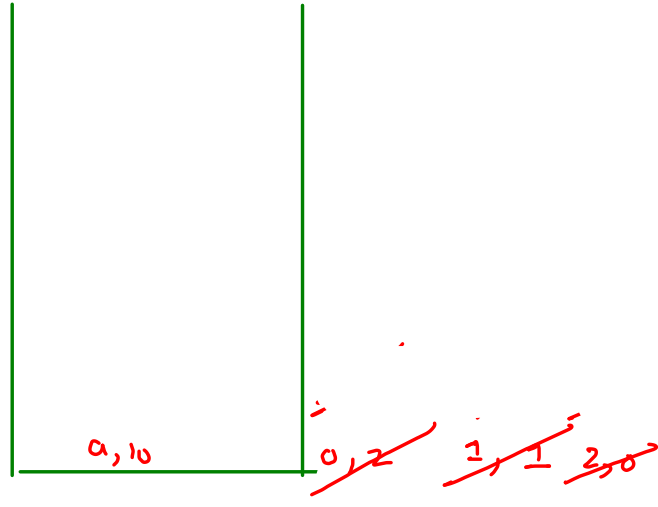

c
j
i
h
g
m
u

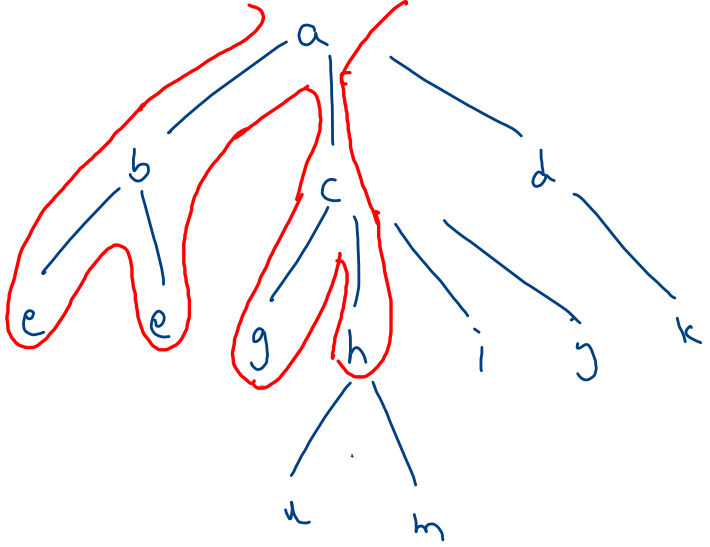

b, 40 (T)
a, 10
c, 30 (F)

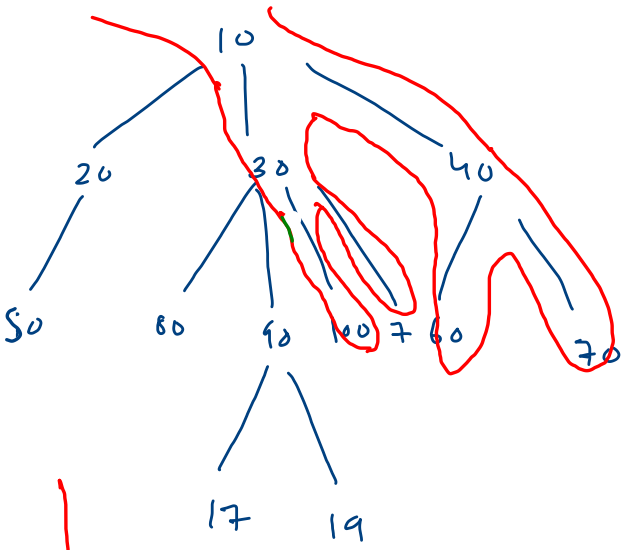




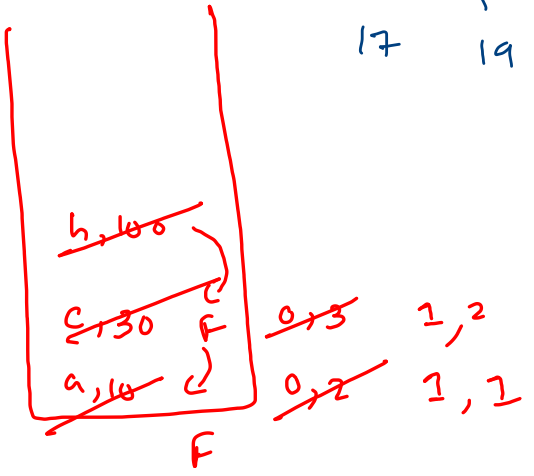

```java
public static boolean areMirror(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

    for(int i=0; i < n1.children.size();i++) {
        Node c1 = n1.children.get(i);
        int s2 = n2.children.size();
        Node c2 = n2.children.get(s2-i-1);

        boolean areMirrorChild = areMirror(c1,c2);

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

    return true;

}
```

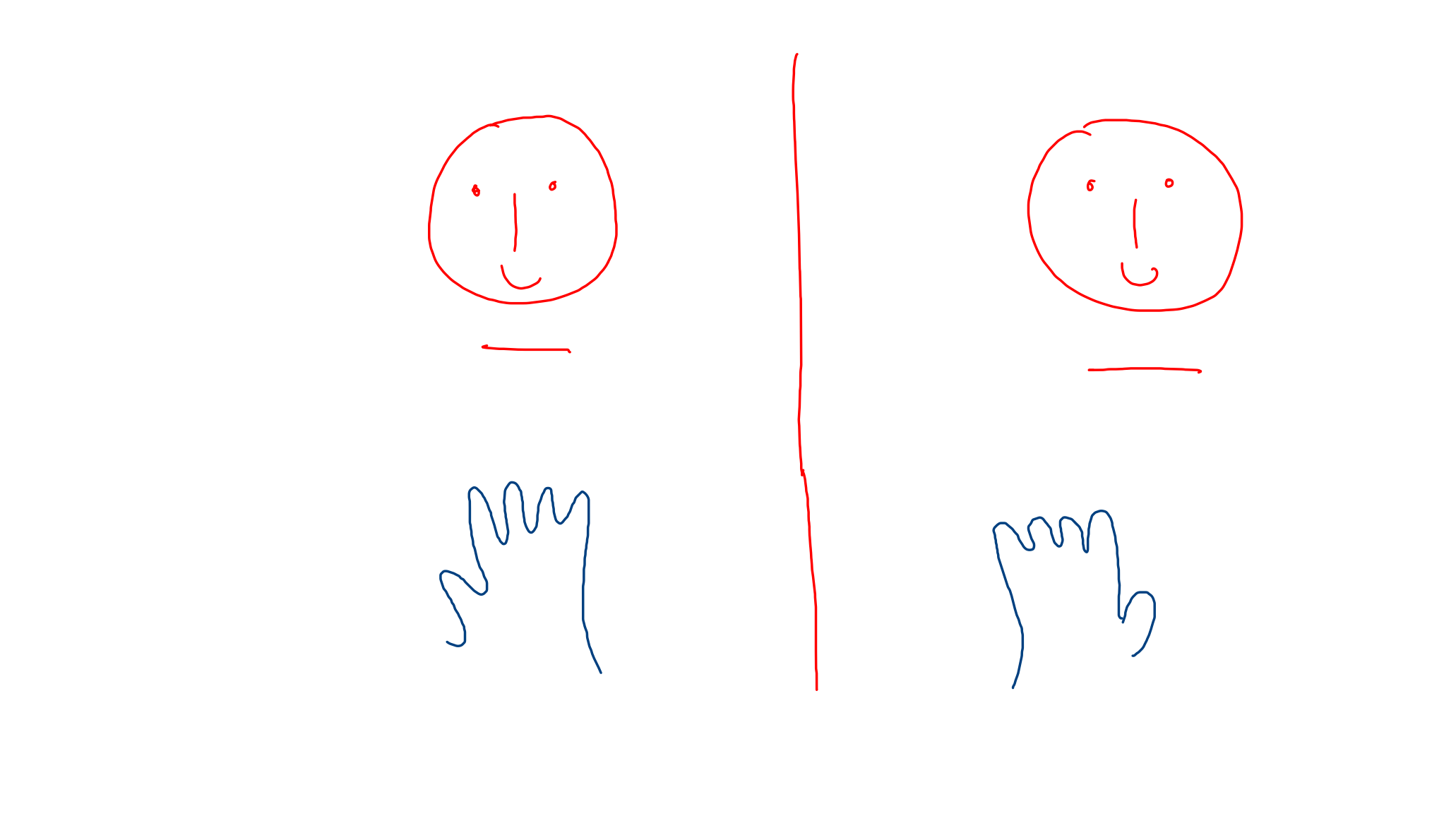




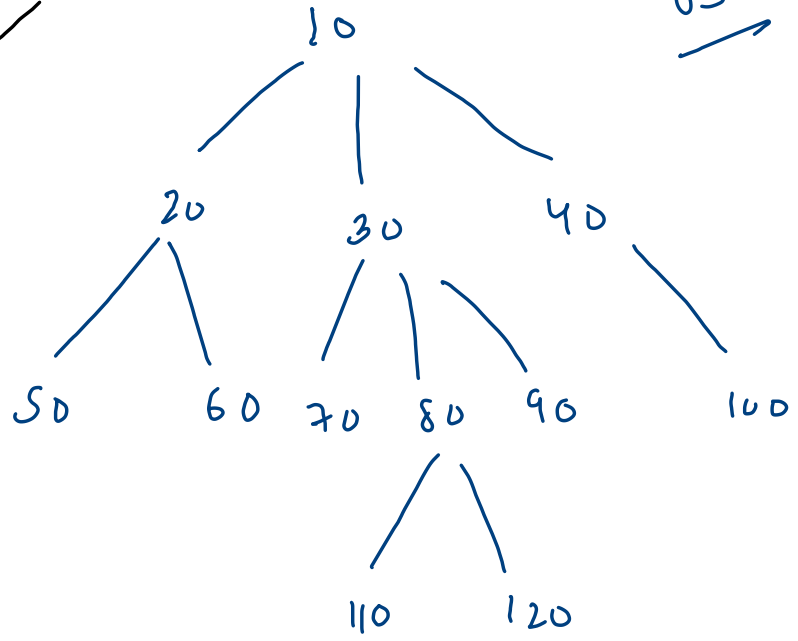



```java
public static boolean areMirror(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size()) {
        return false;
    }

    for(int i=0; i < n1.children.size();i++) {
        Node c1 = n1.children.get(i);
        int s2 = n2.children.size();
        Node c2 = n2.children.get(s2-i-1);

        boolean areMirrorChild = areMirror(c1,c2);

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

    return true;

}
```





is symmetric

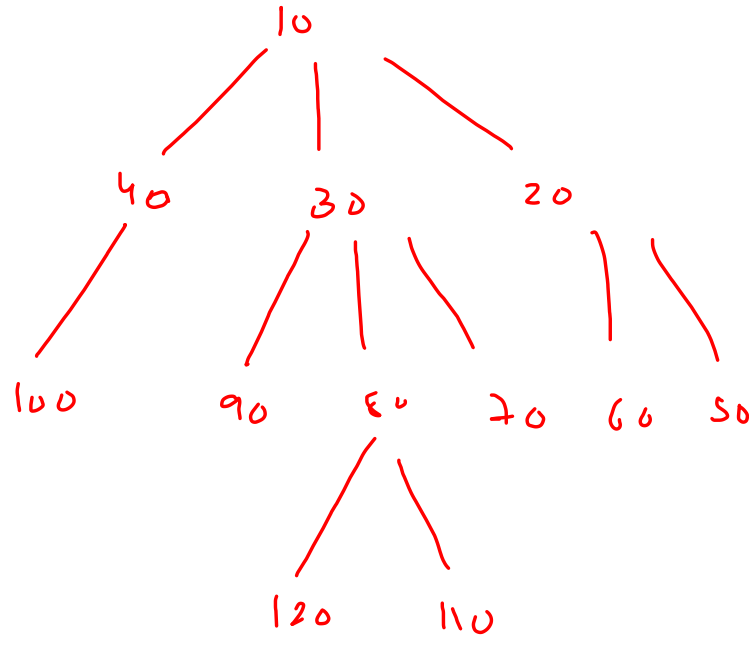


OS

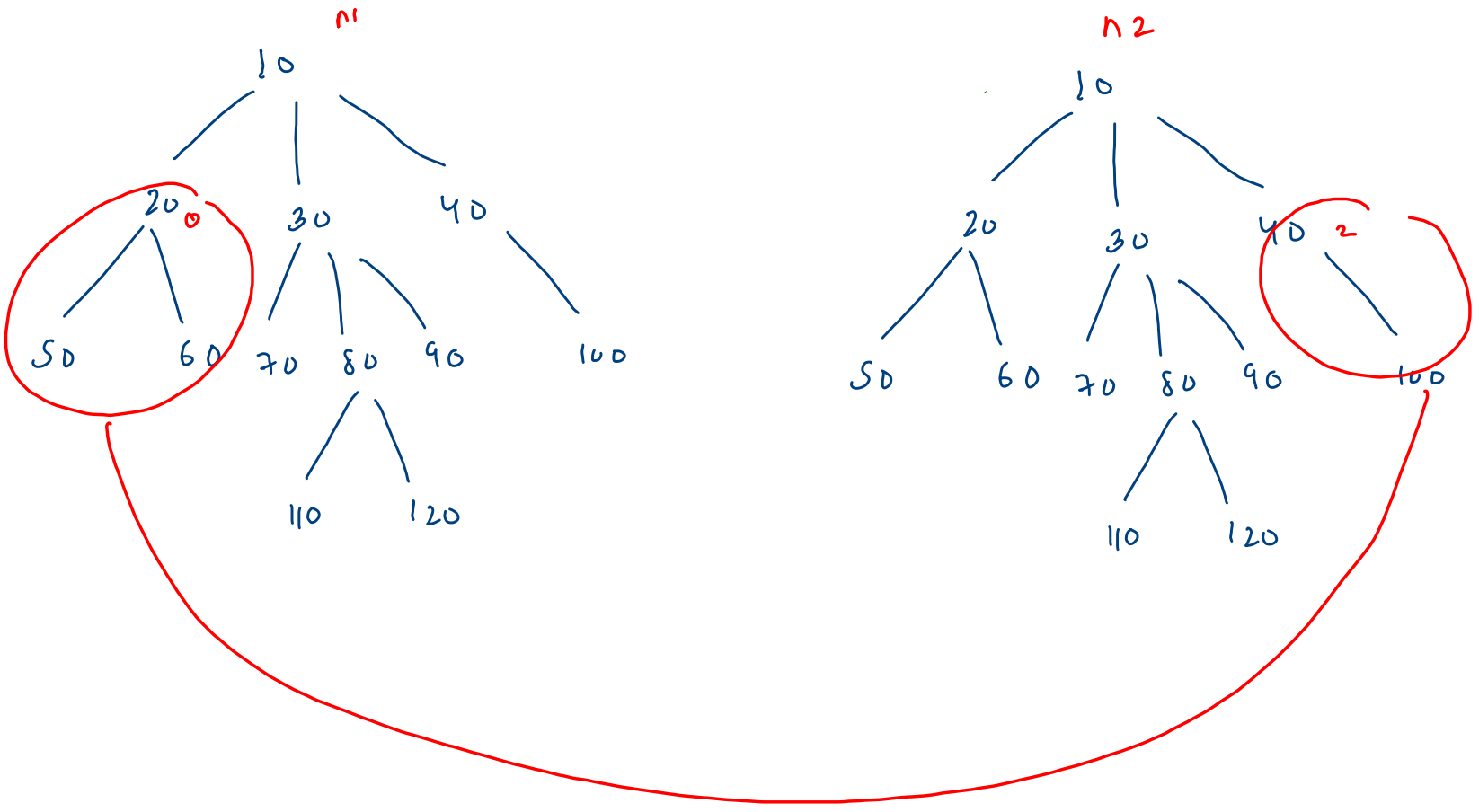
OS != mirror

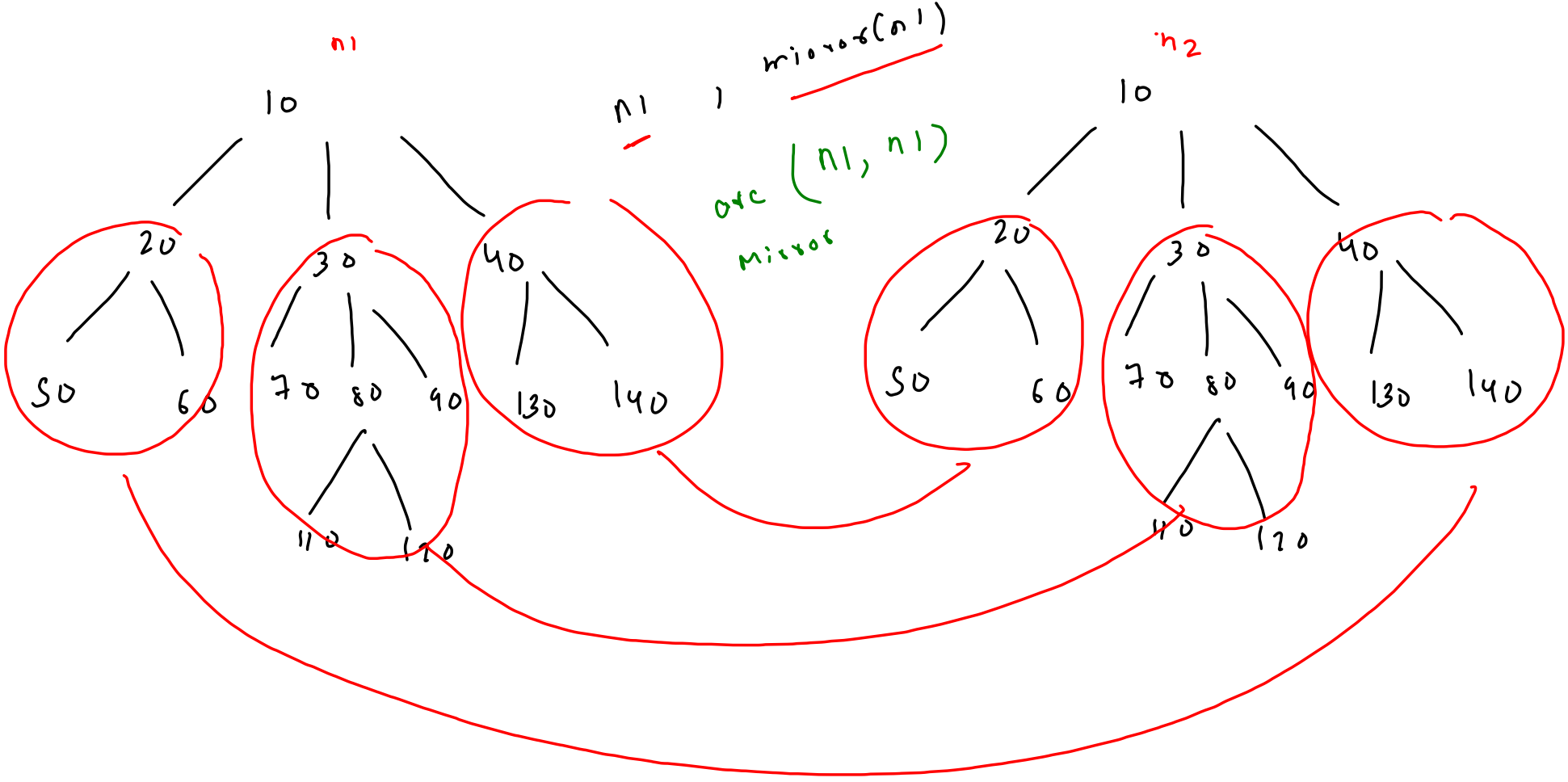
-) asymmetric

mirror



isMirror(n1, n1)


n1
10
20
30
40
50
60
70
80
90
100
110
120
n2
10
20
30
40
2
50
60
70
80
90
100
110
120


n1
10
20
30
40
50
60
70
80
90
110
120
130
140
n1
mirror(n1)
or (n1, n1)
Mirror
n2
10
20
30
40
50
60
70
80
90
110
120
130
140