height = max depth
= 3

root

depth

edge

edges

1

2

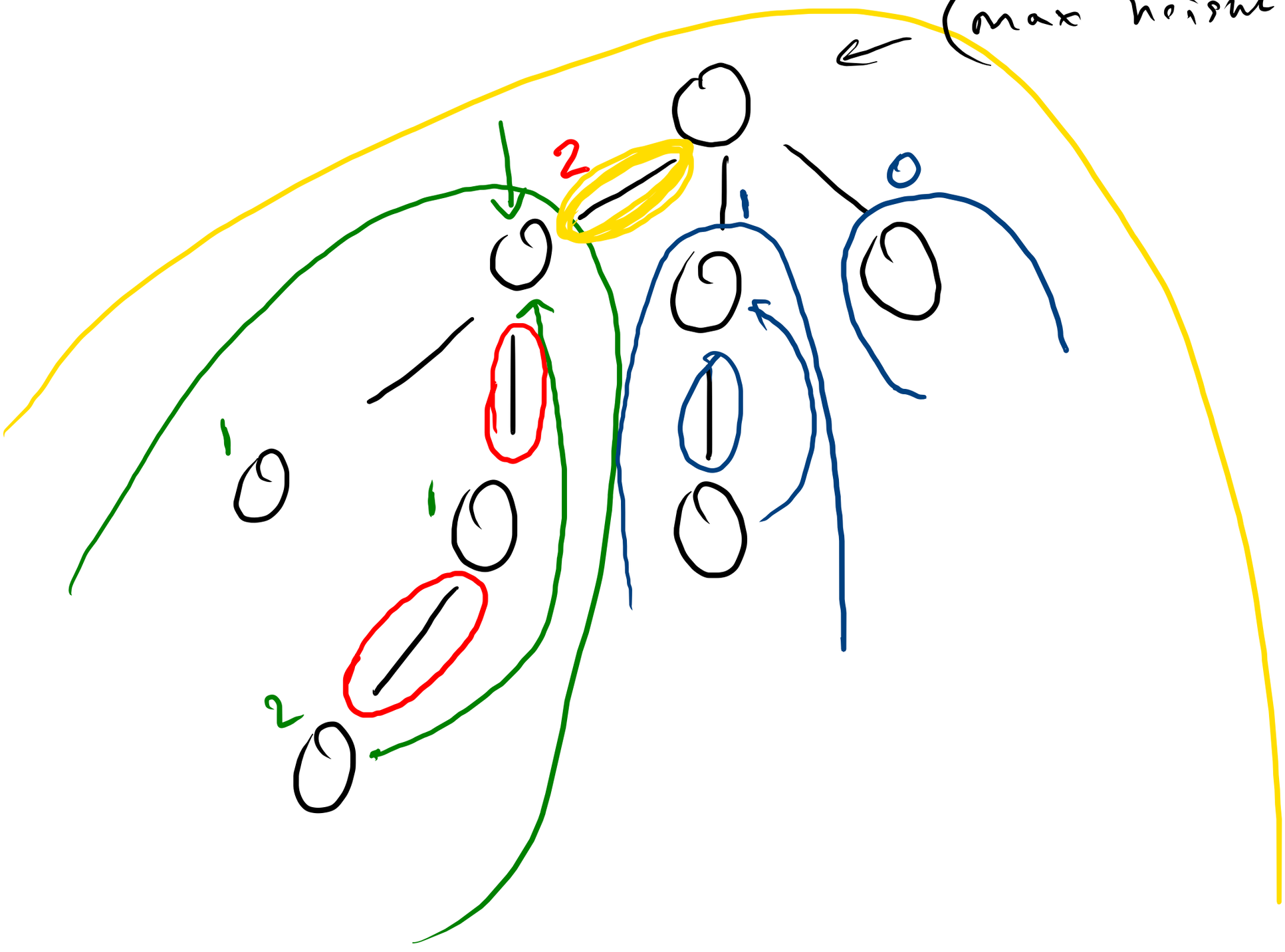
2

2

3

3

(max height of child) + 1



```java
public static int height(Node node) {

    int h= -1;

    for(Node child: node.children){
        h = Math.max(h, height(child));
    }
    return h+1;
}
```

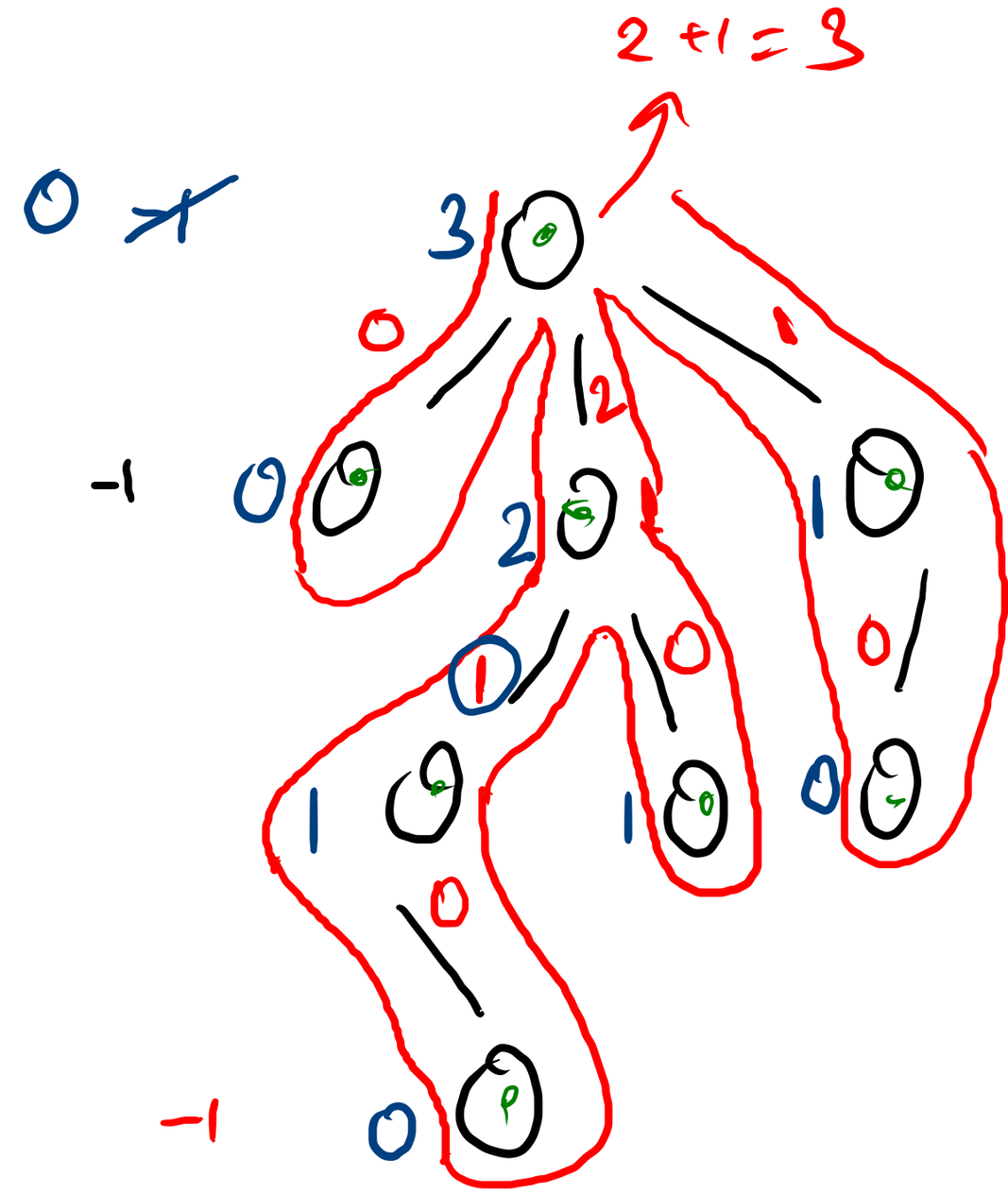
h

h+1 = 0

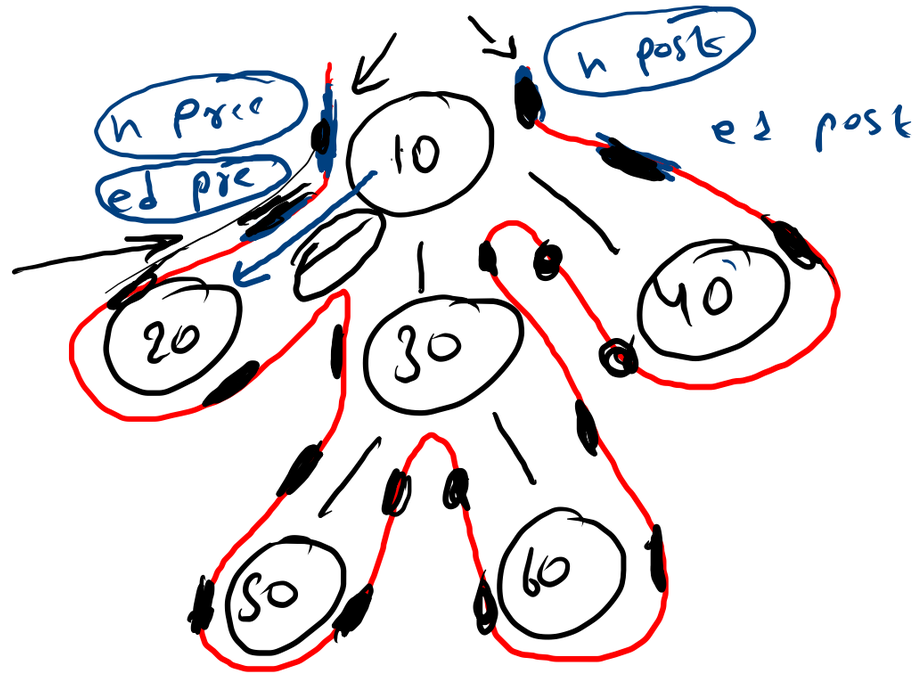
h = -1

Time complexity

O(n)



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



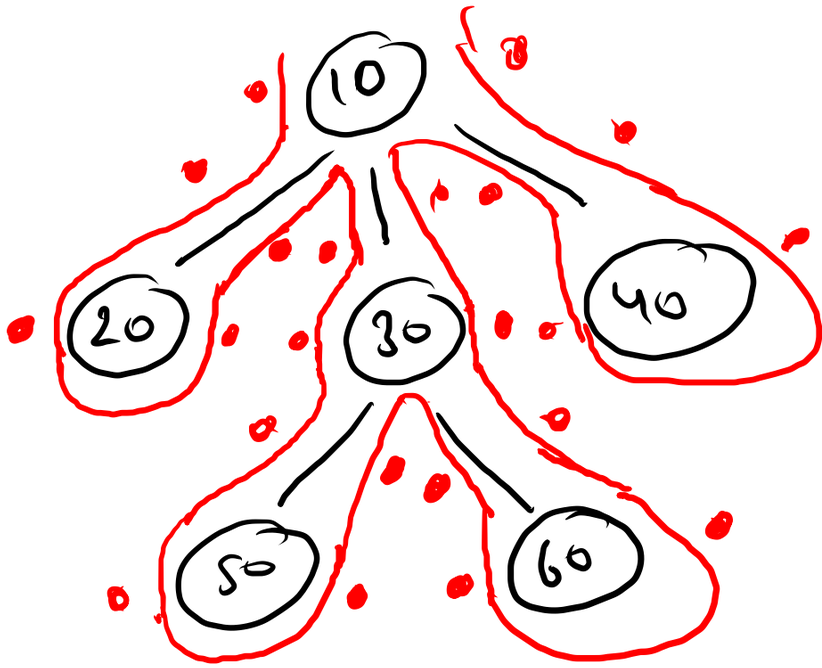
node pre 10
edge pre 10--20
node pre 20
node post 20
edge post 10--20
node pre 30
edge pre 30--50
node pre 50
node post 50
edge post 50--60
edge pre 50--60
node pre 60
node post 60
edge post 50--60

edge post 10--30
edge pre 10--40
node pre 40
node post 40
edge post 10--40
node post 10

node pre
node post

pre area
print()
post area

```java
public static void traversals(Node node){
  System.out.println("Node Pre "+ node.data);

  for (Node child : node.children) {
      System.out.println("Edge Pre "+ node.data+"--"+child.data);
      traversals(child);
      System.out.println("Edge Post "+ node.data+"--"+child.data);
  }

  System.out.println("Node Post "+ node.data);
}
```

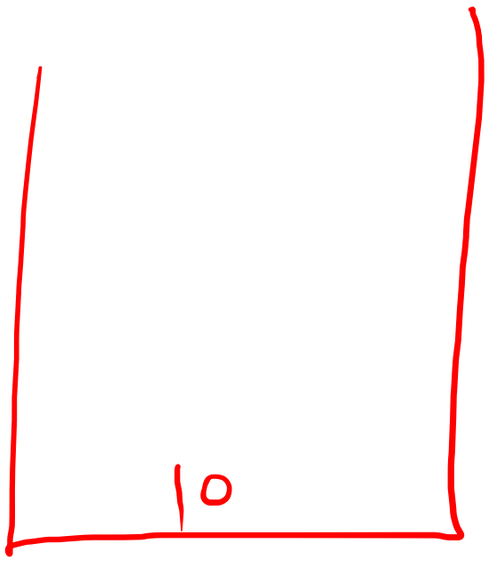


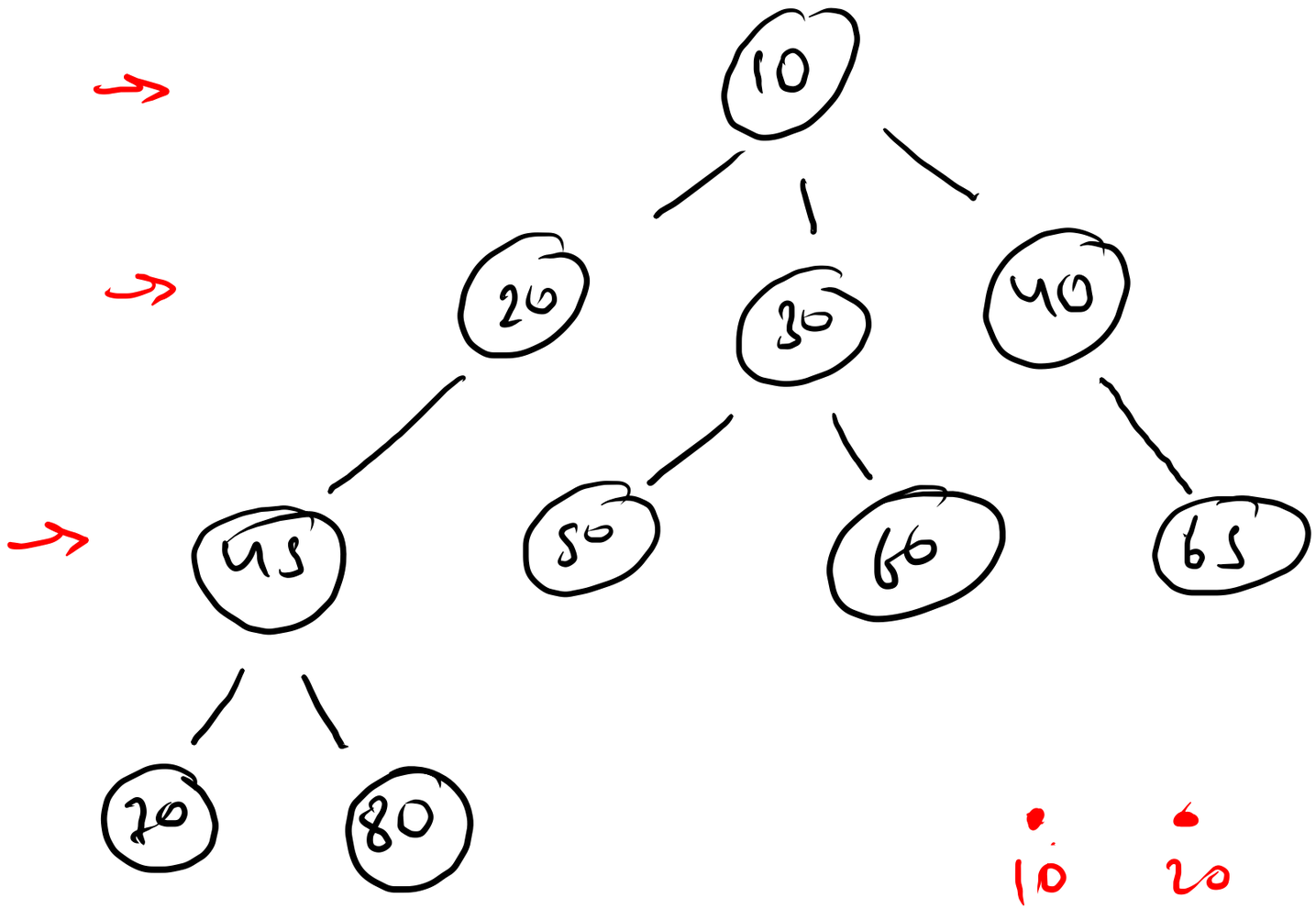
n p 10

e pre 10 -- 20

n Pre 20

n Post 20





10

10 | 20 | 40 |

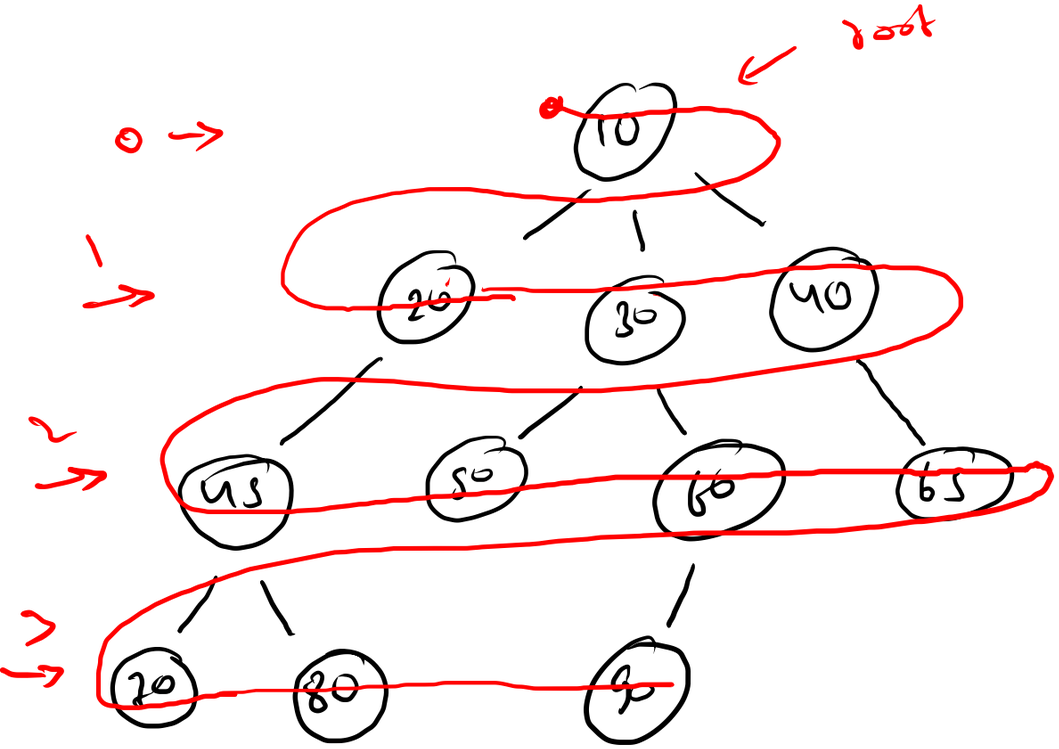
20 | 20 | 40 |

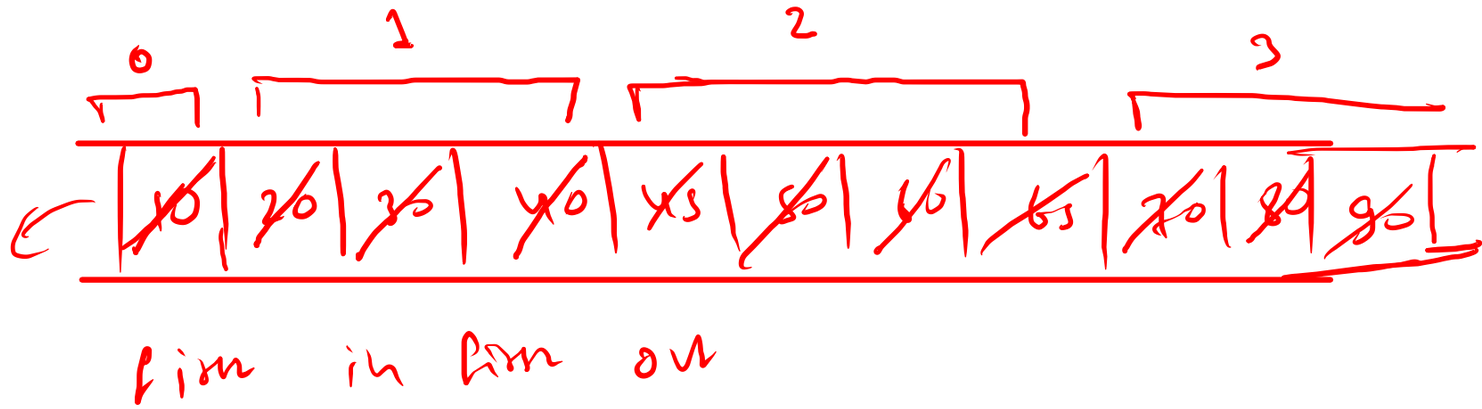
45 | 50 | 60 | 65 |

45 | 50

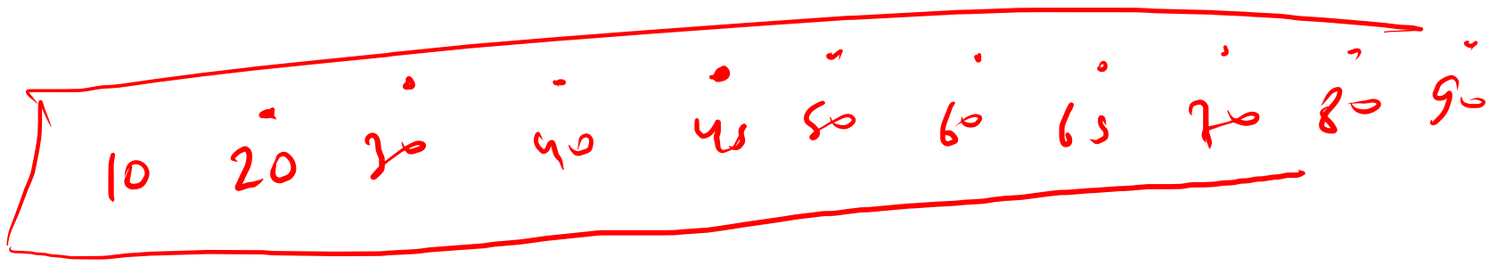
10 20 30 40 45 50 60 65

70 80





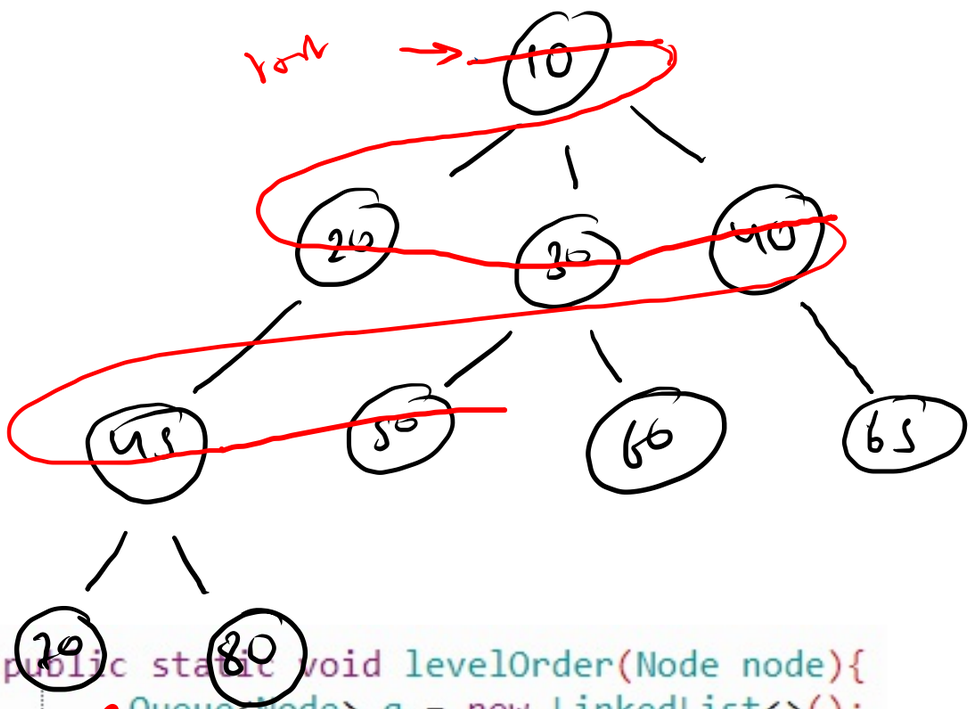
Fim in Fim out

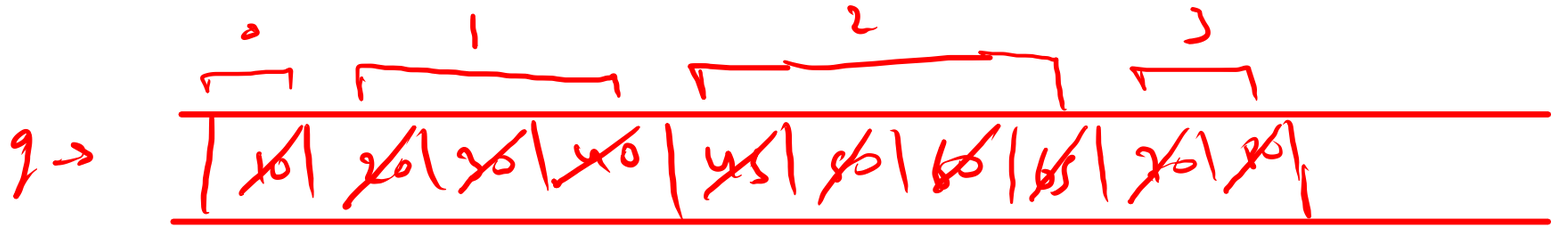


remove

print ←

child ←






```java
public static void levelOrder(Node node){
    Queue<Node> q = new LinkedList<>();

    q.add(node);

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

10 20 30 40 45 50

10

20 30 40

45 50 60 65

Queue<Node> q = new LinkedList / ArrayDeque

interface

new

class

class

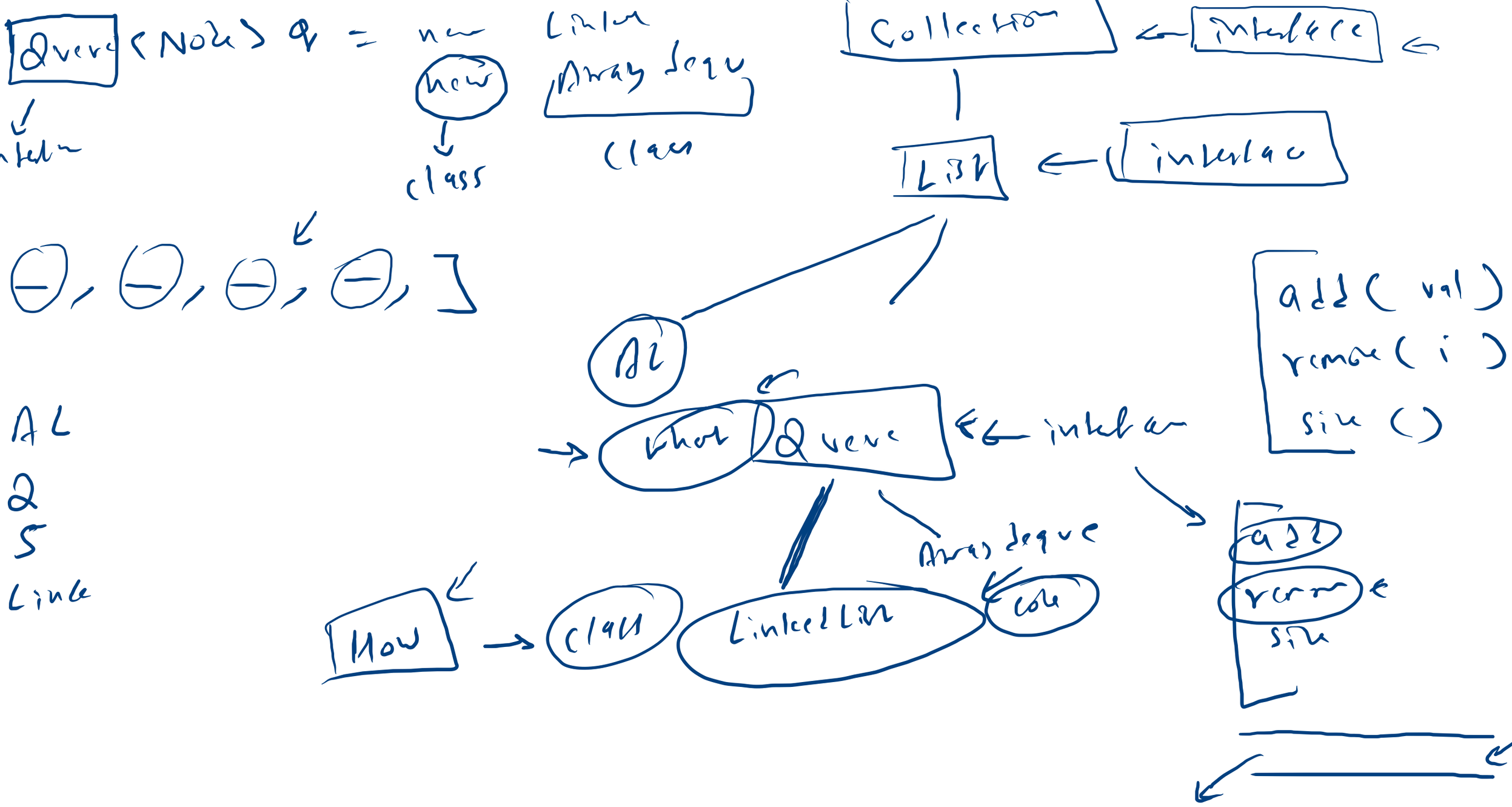
[0, 0, 0, 0,]

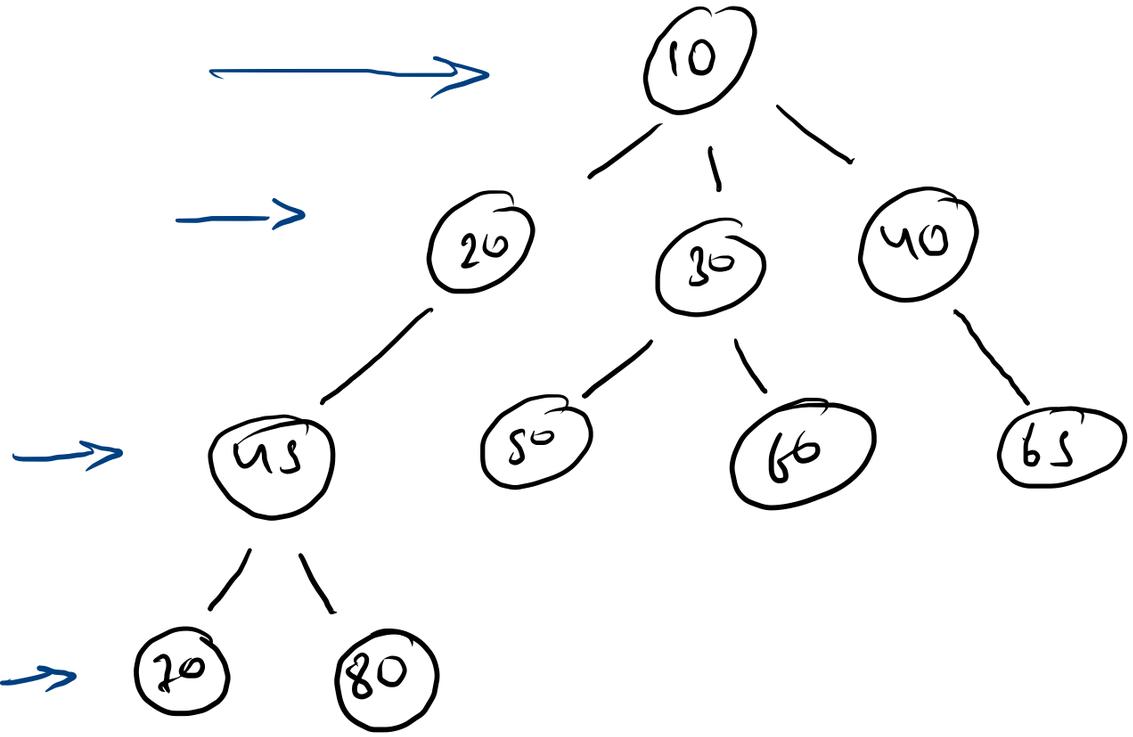
AL

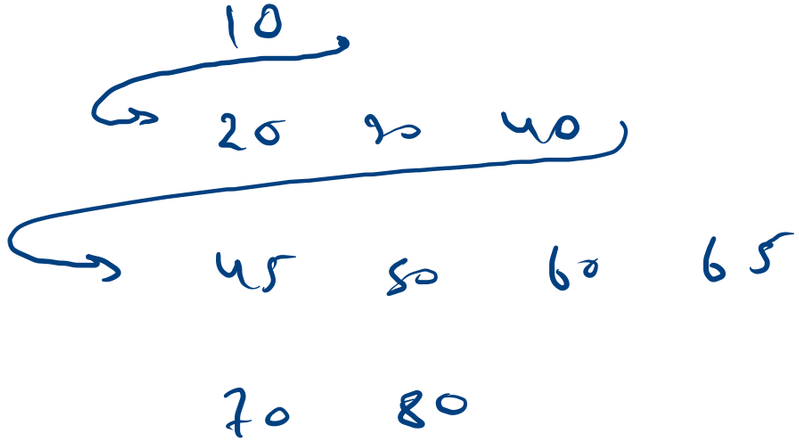
2

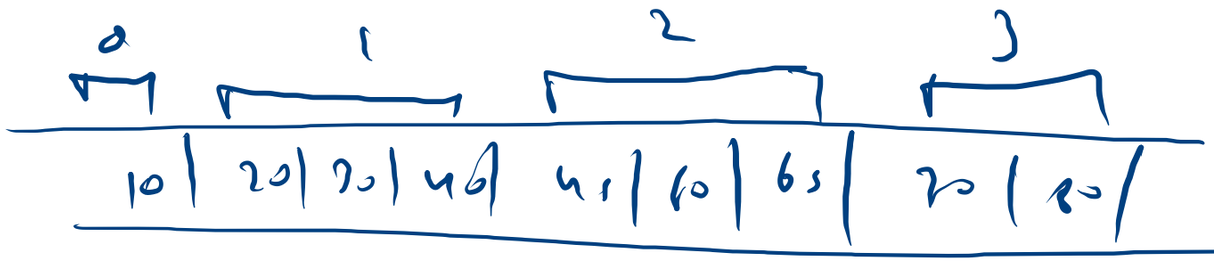
5

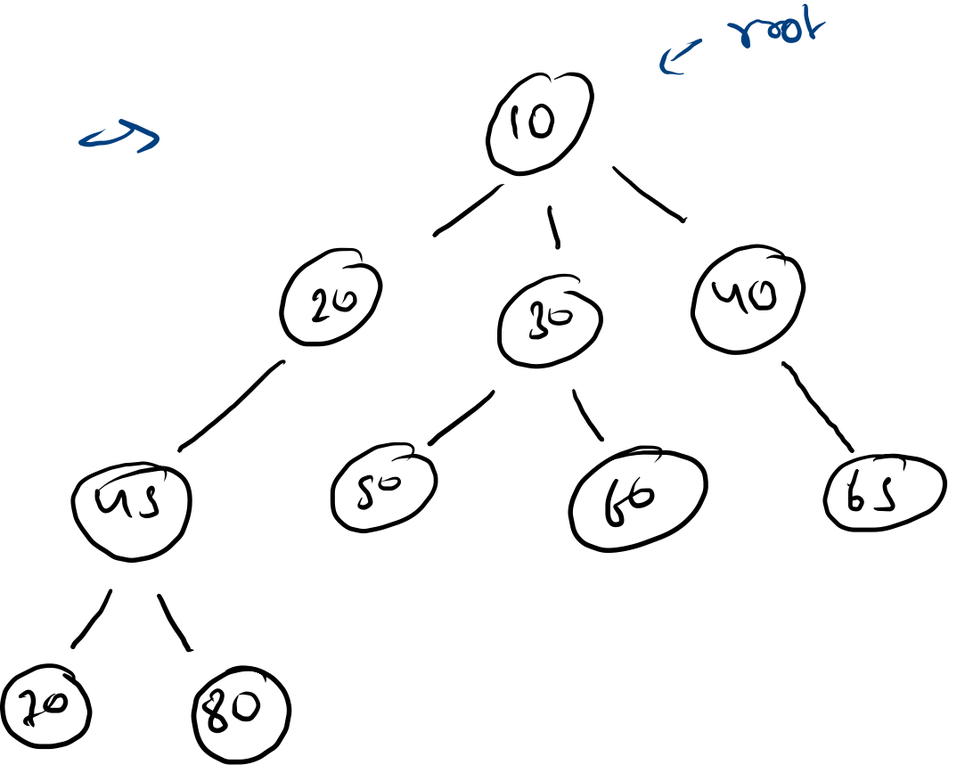
LinkedList




10
20
30
40
45
50
60
65
70
80


10
20
30
40
45
50
60
65
70
80


0
1
2
3
10
20
30
40
45
50
60
65
70
80



main
child

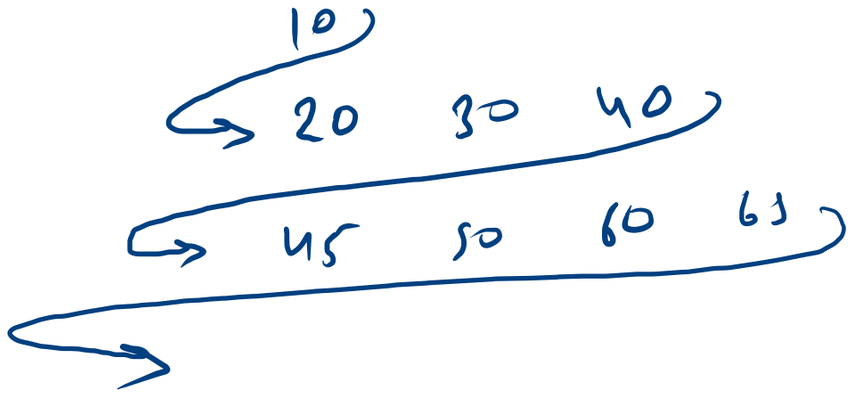
main

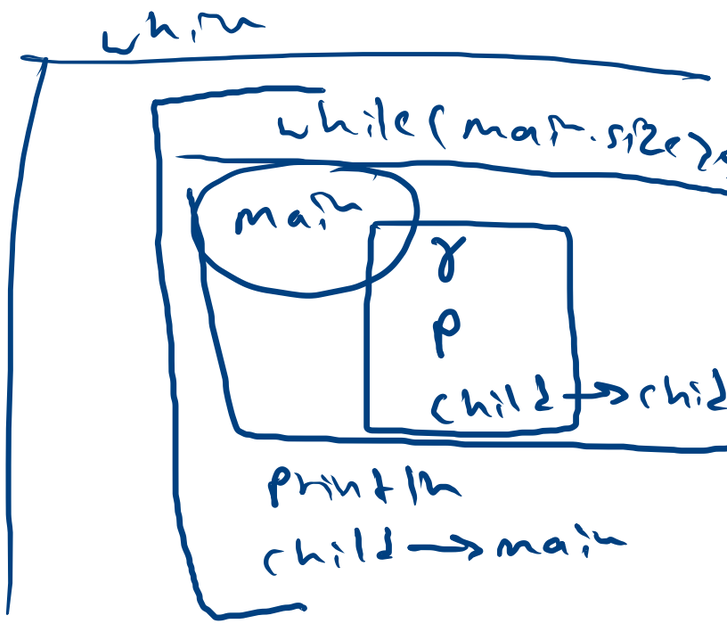
child

r

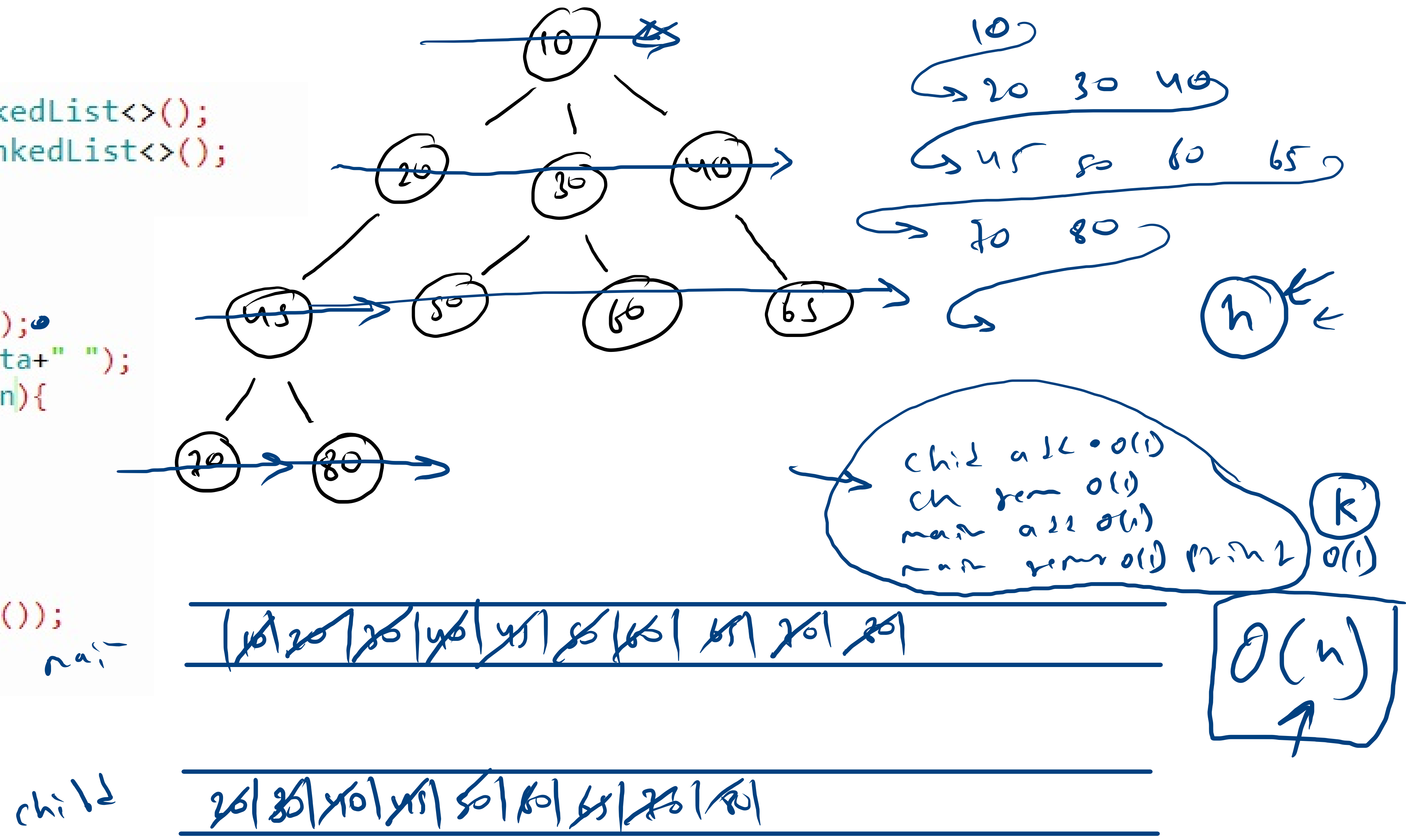
p

ch



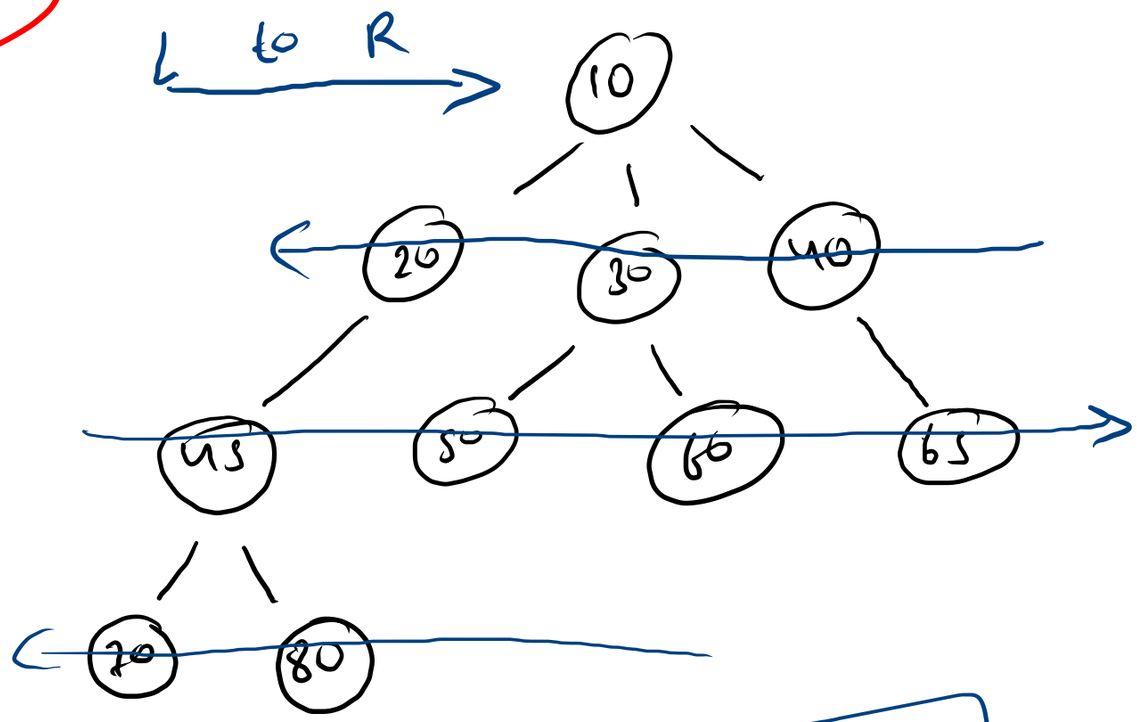


```java
Queue<Node> main = new LinkedList<>();
Queue<Node> child = new LinkedList<>();
main.add(node);

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



H.W.

2:3.293



10

40 30 20

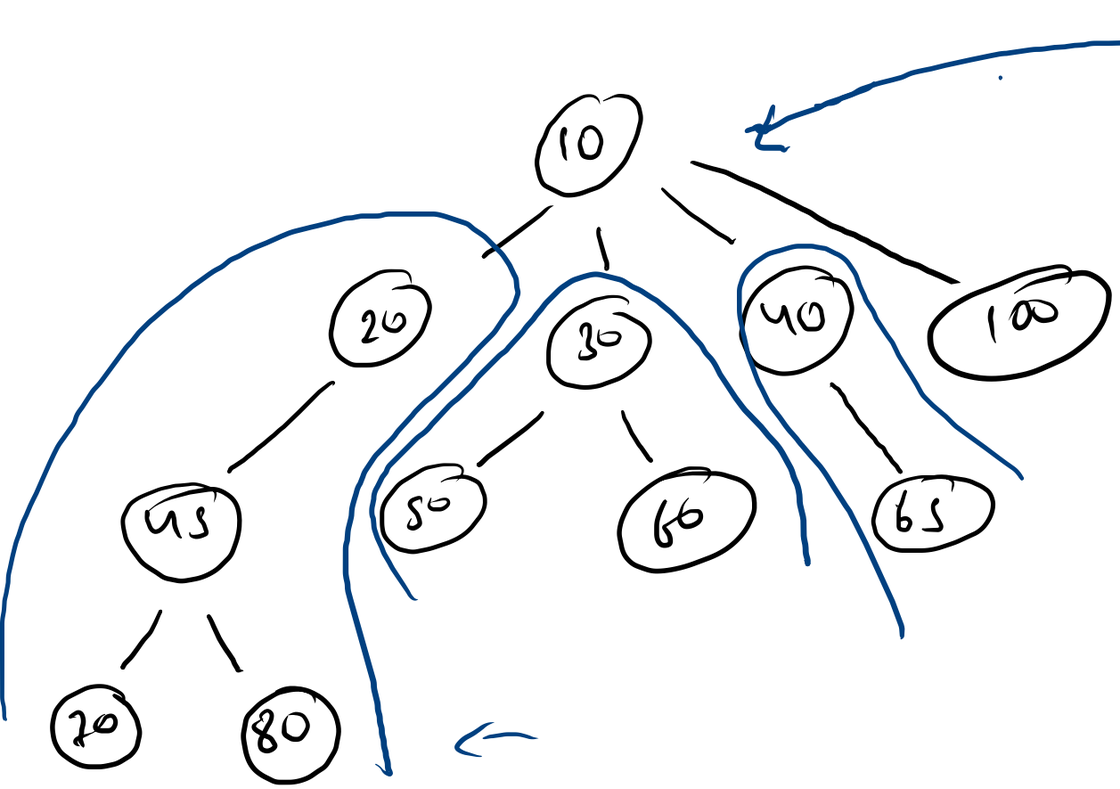
45 50 60 65

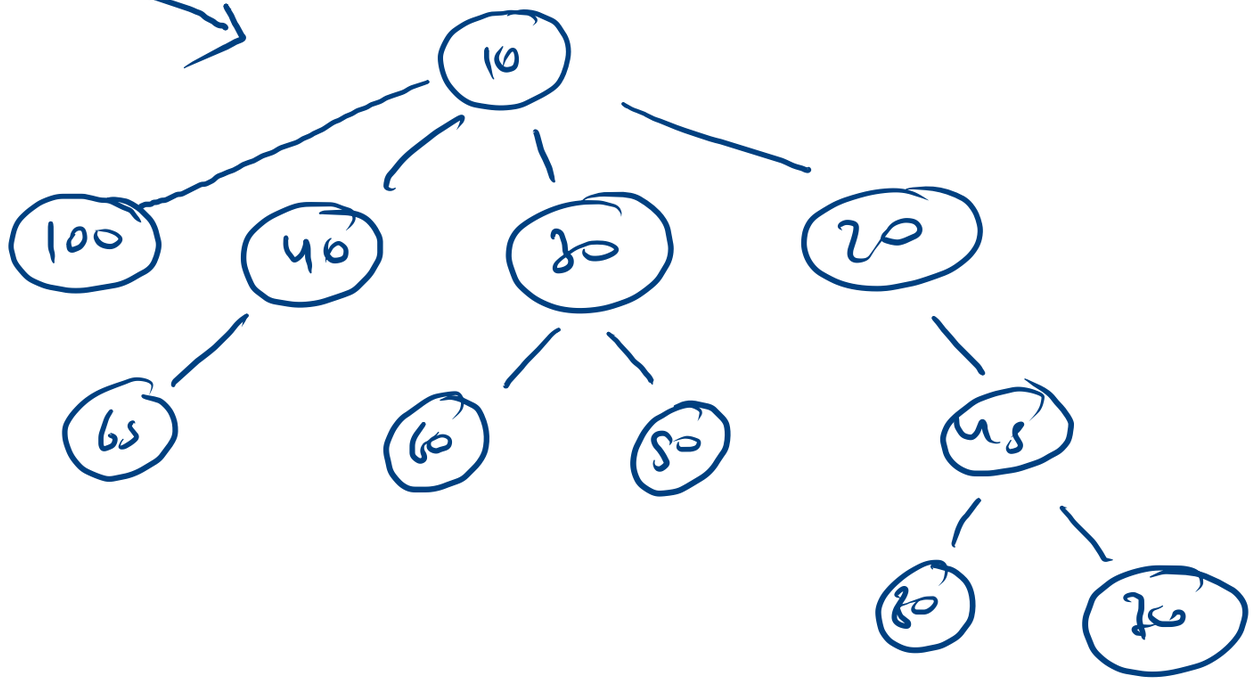
80 70

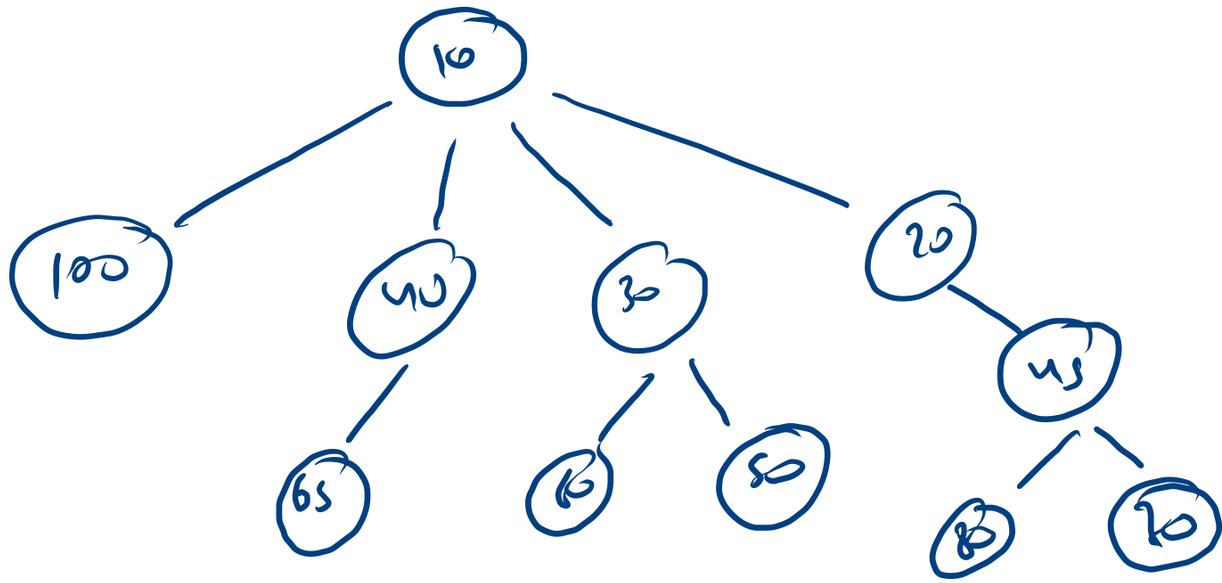
2 Queue

2 stacle

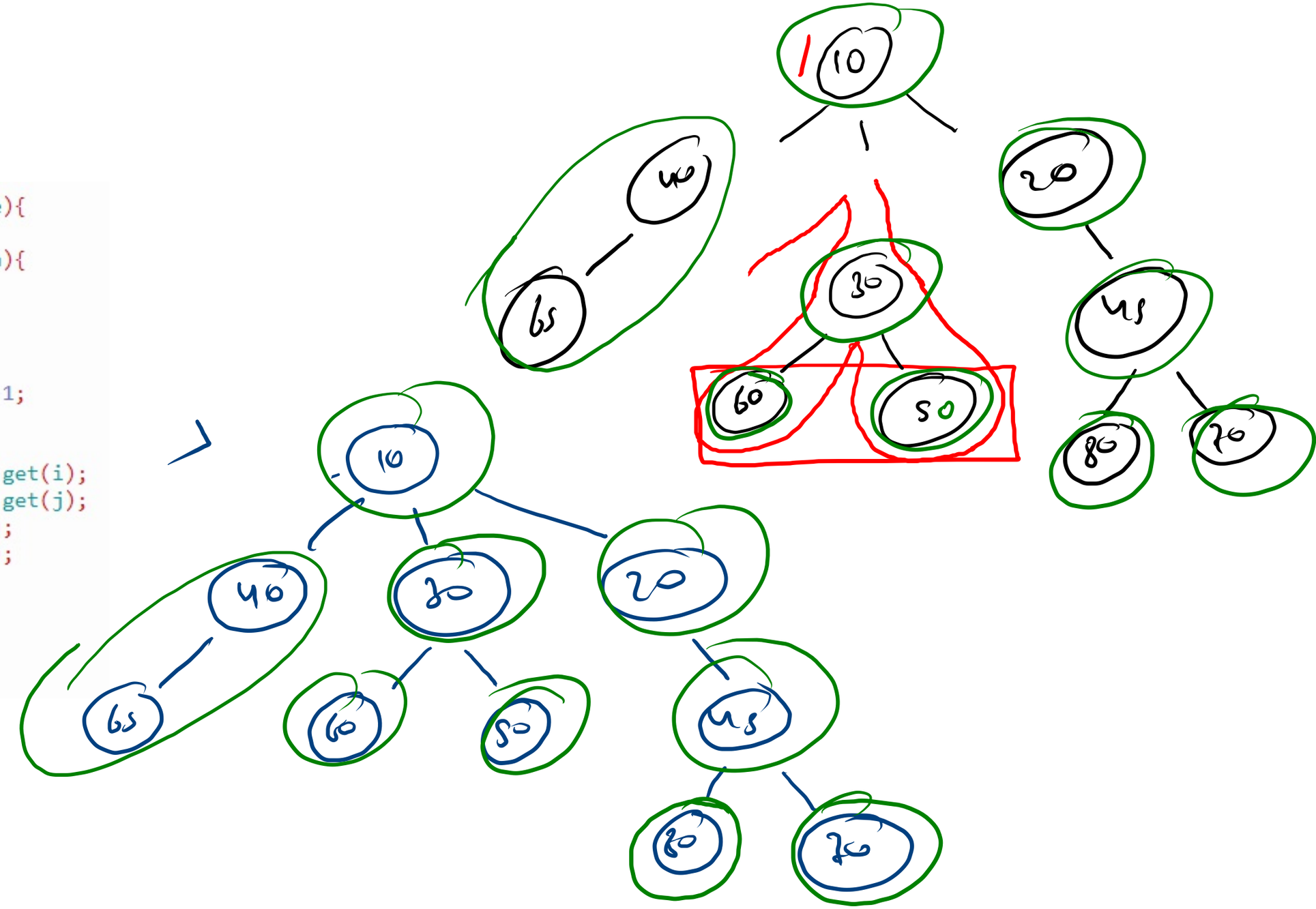
☆ child all order


10
20
30
40
100
45
50
60
65
20
80


10
100
40
30
20
65
60
50
45
80
20

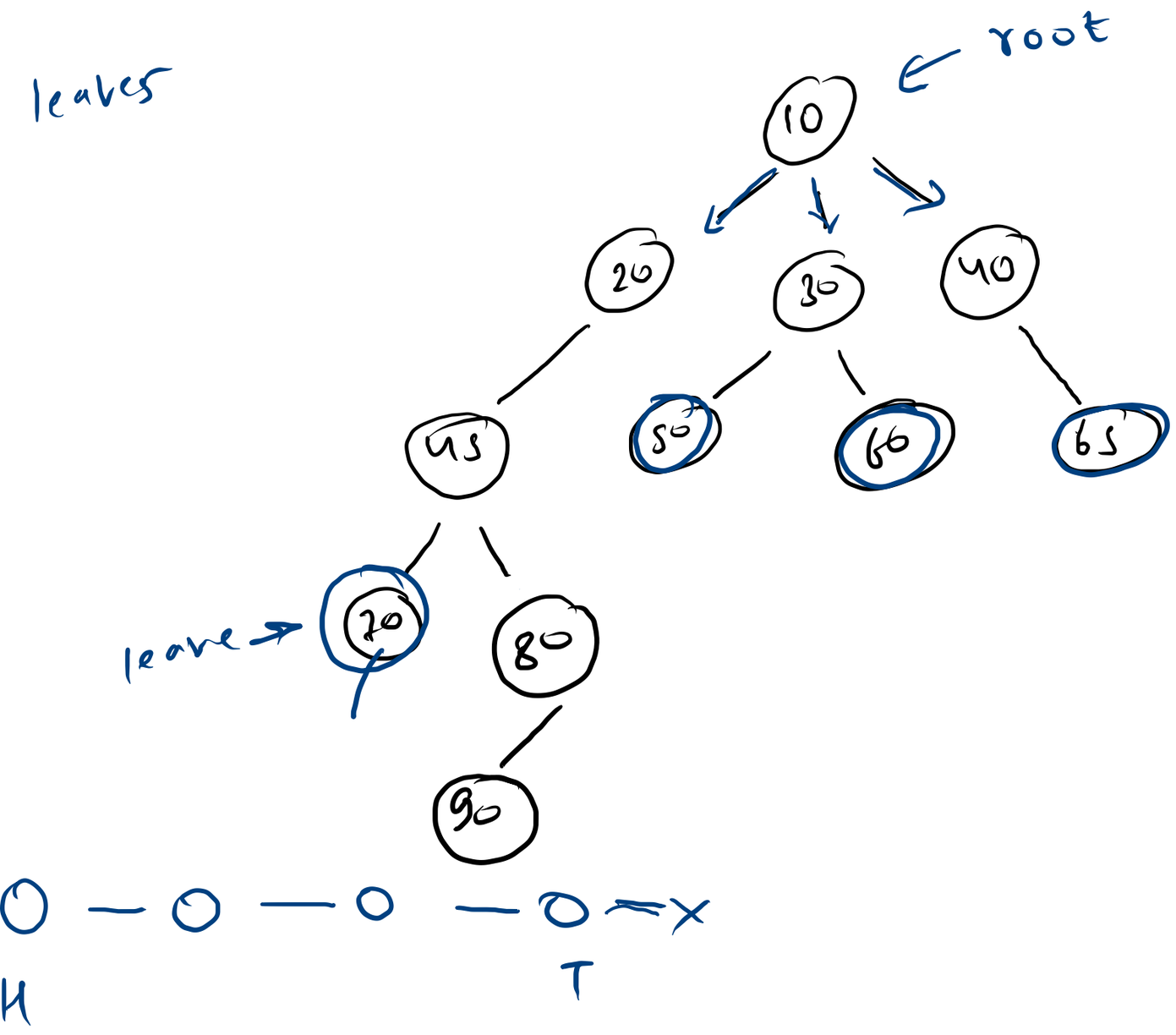

10
100
40
30
20
65
60
50
45
80
20

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

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

    int i=0;
    int j = node.children.size()-1;

    while(i<j){
        Node ith = node.children.get(i);
        Node jth = node.children.get(j);
        node.children.set(i, jth);
        node.children.set(j, ith);
        i++;
        j--;
    }

}
```

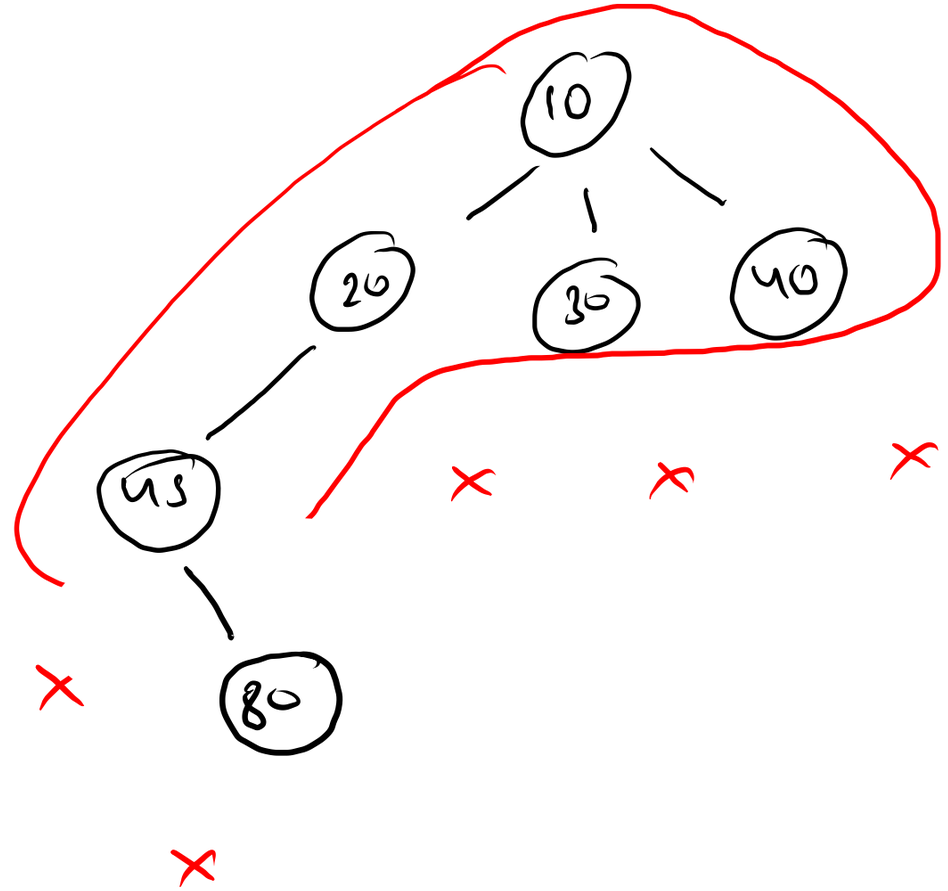


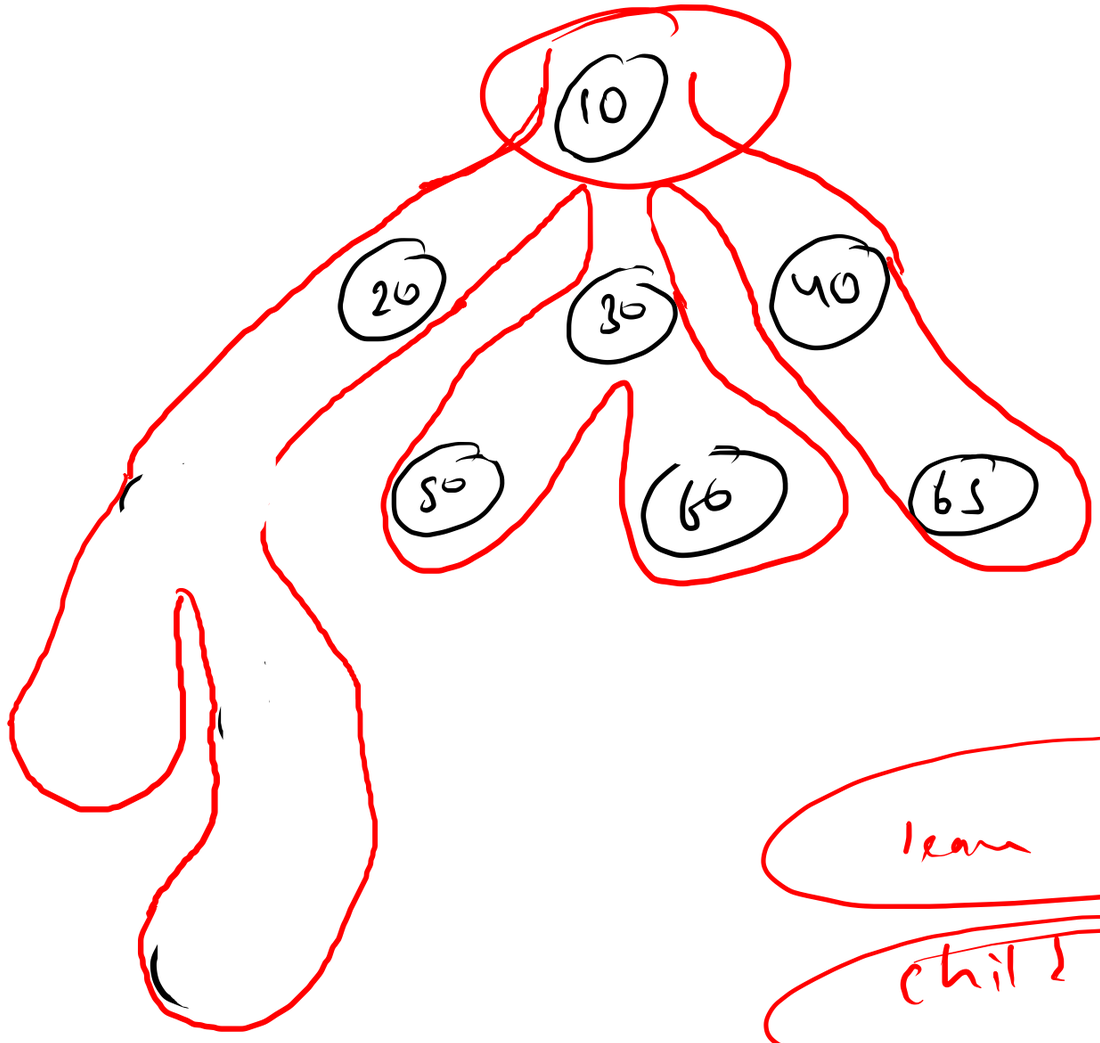
leaves

root



leave





Child

self work

learn & work

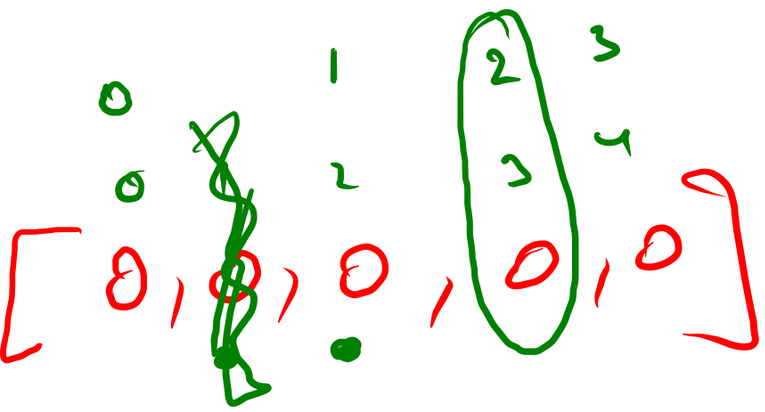
child faith

Perfect

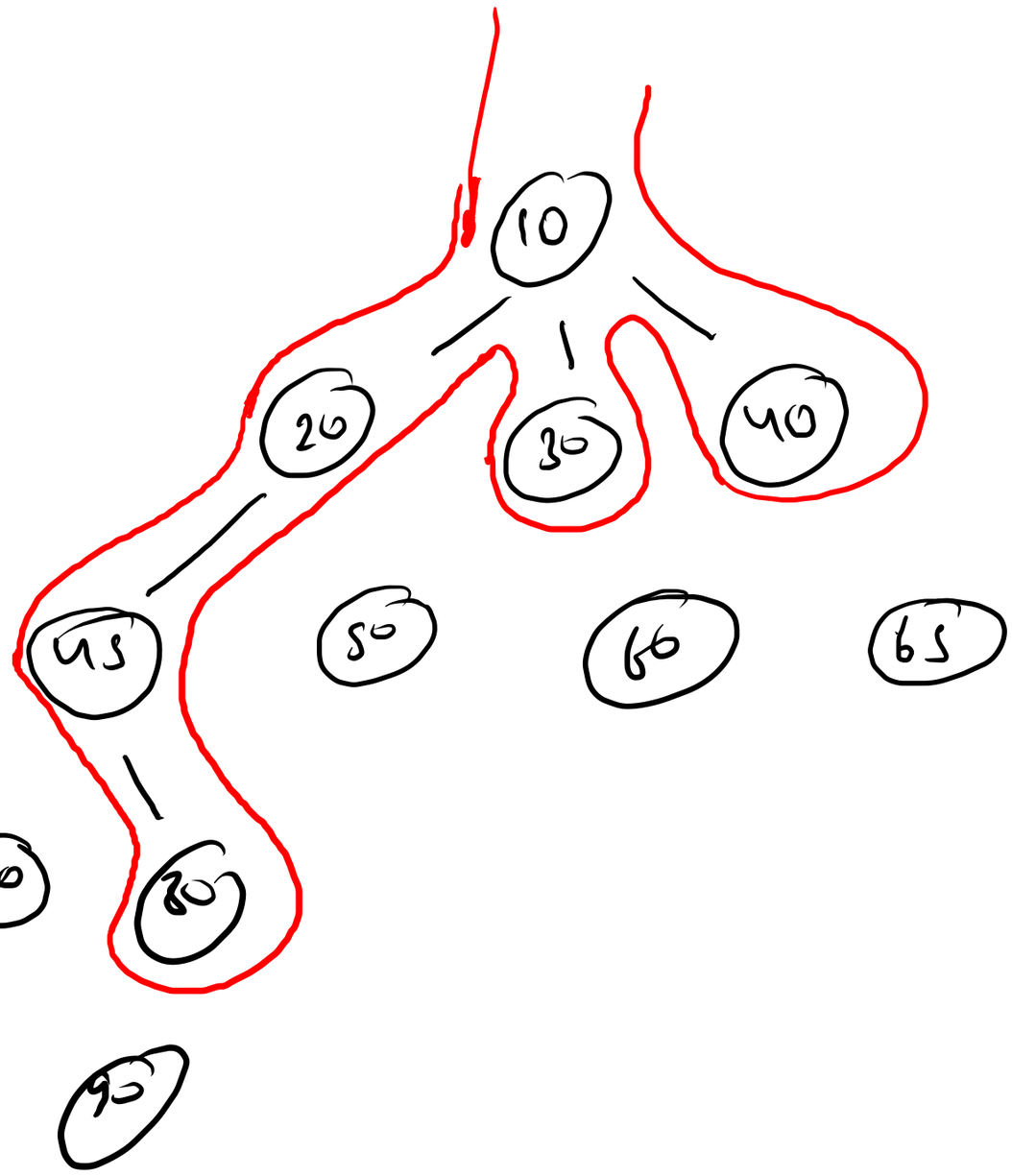
pre Arr

leaves sum

child sum



i 0 1 2



Original

root

10

20 30 40

45 50 60 65

70

root

10

20 30 40

45 50 60 65

70

10

20

30

40

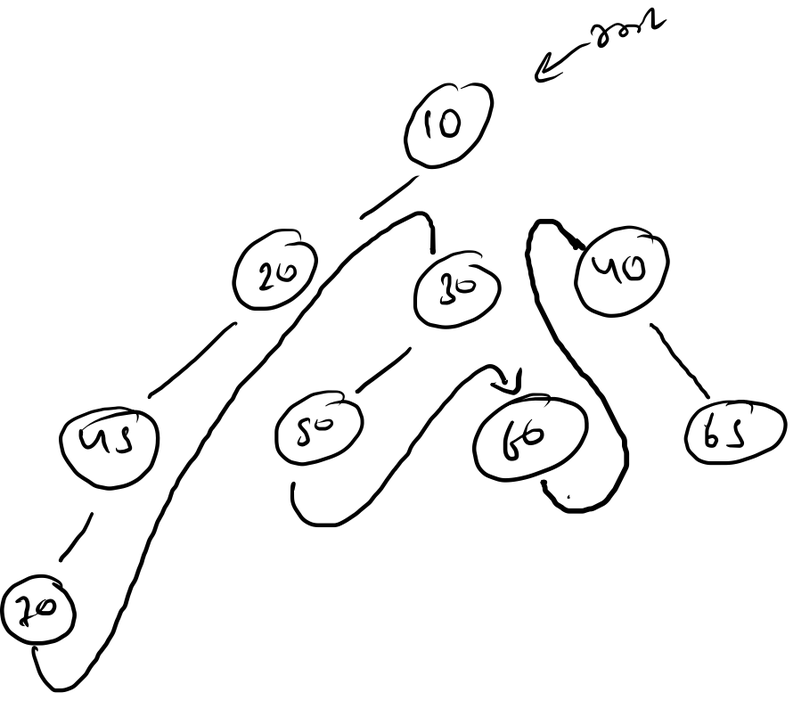
50

45

70

60

65




```
Node getLeave(...
```

faith in child

```
second last leaf
leaf.children.add(lastchild)
remove last child
```

[20 30]

```java
public static Node getLeaf(Node node){
    if(node.children.size() == 0)return node;
    return getLeaf(node.children.get(0));
}

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

    while(node.children.size() > 1){
        Node sLNode = node.children.get(node.children.size()-2);
        Node leaf = getLeaf(sLNode);
        Node last = node.children.get(node.children.size()-1);
        leaf.children.add(last);
        node.children.remove(node.children.size() - 1);
    }
}
```