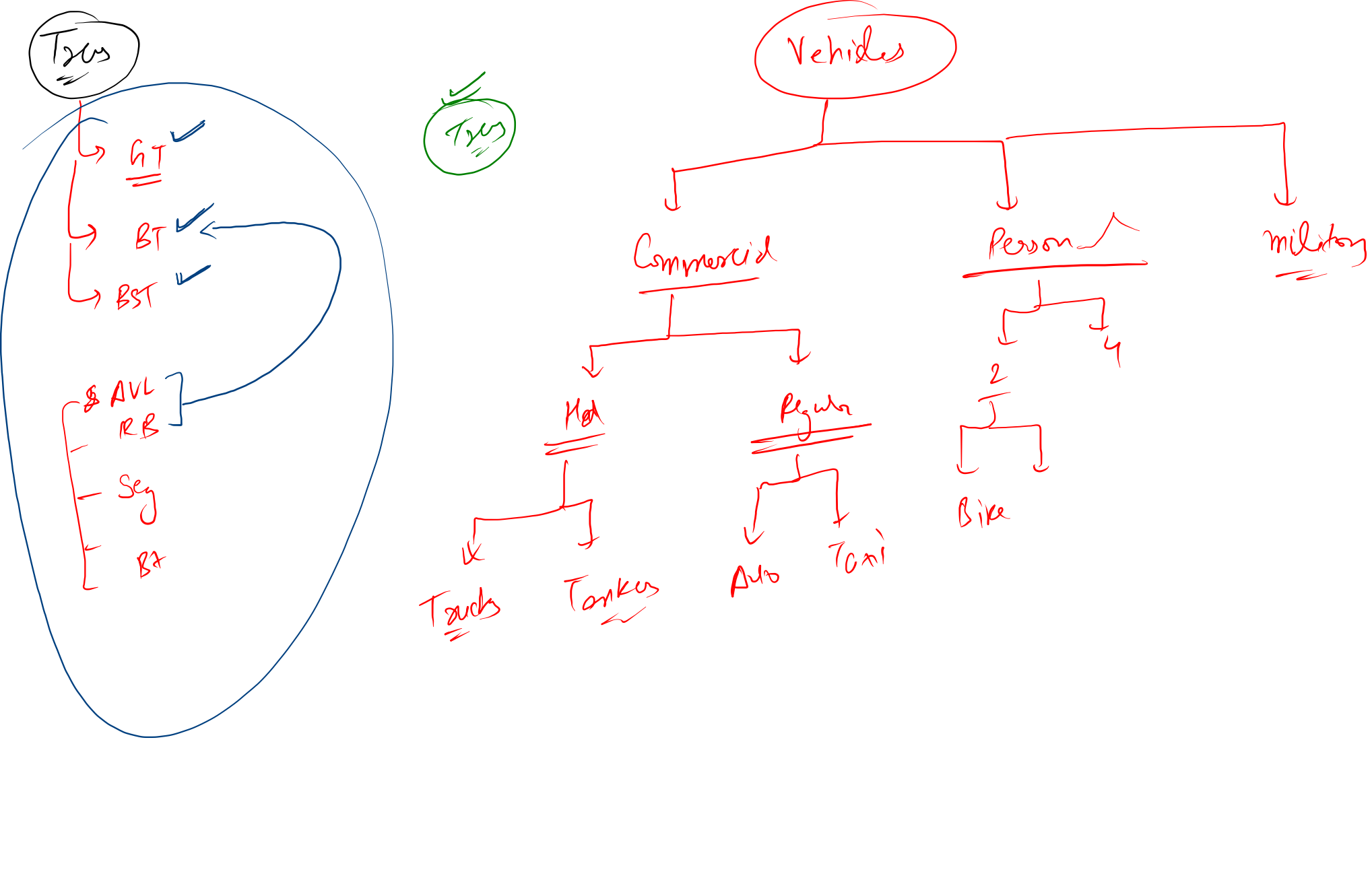
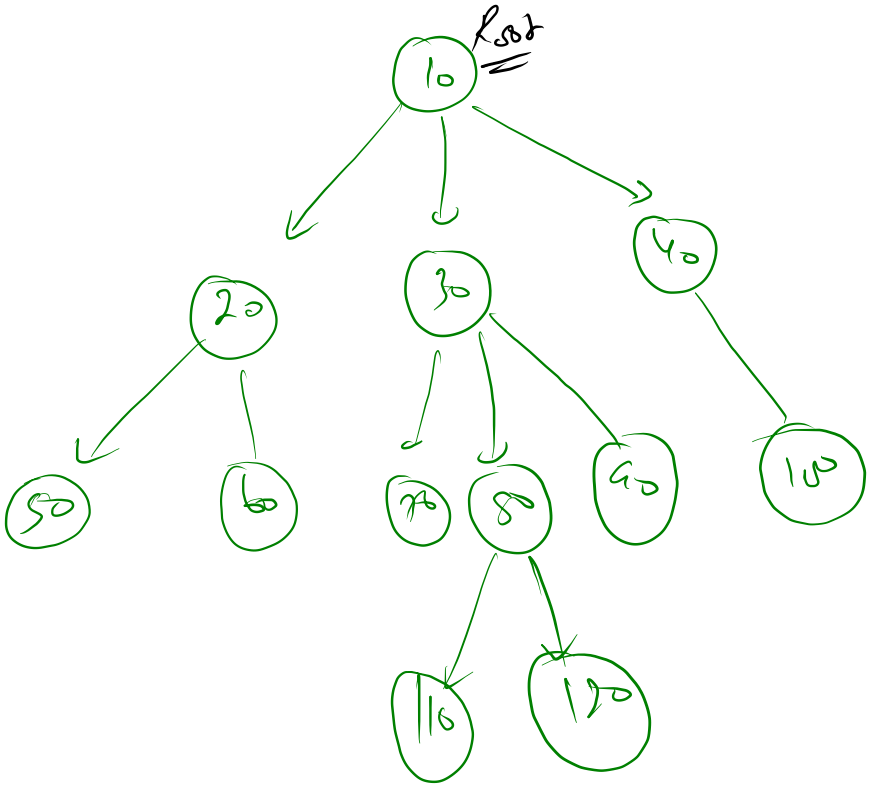

Trees
GT
BT
BST
AVL
RB
Seg
BT
Trees
Vehicles
Commercial
Person
military
Heavy
Regular
2
4
LV
Trucks
Tankers
Auto
Taxi
Bike

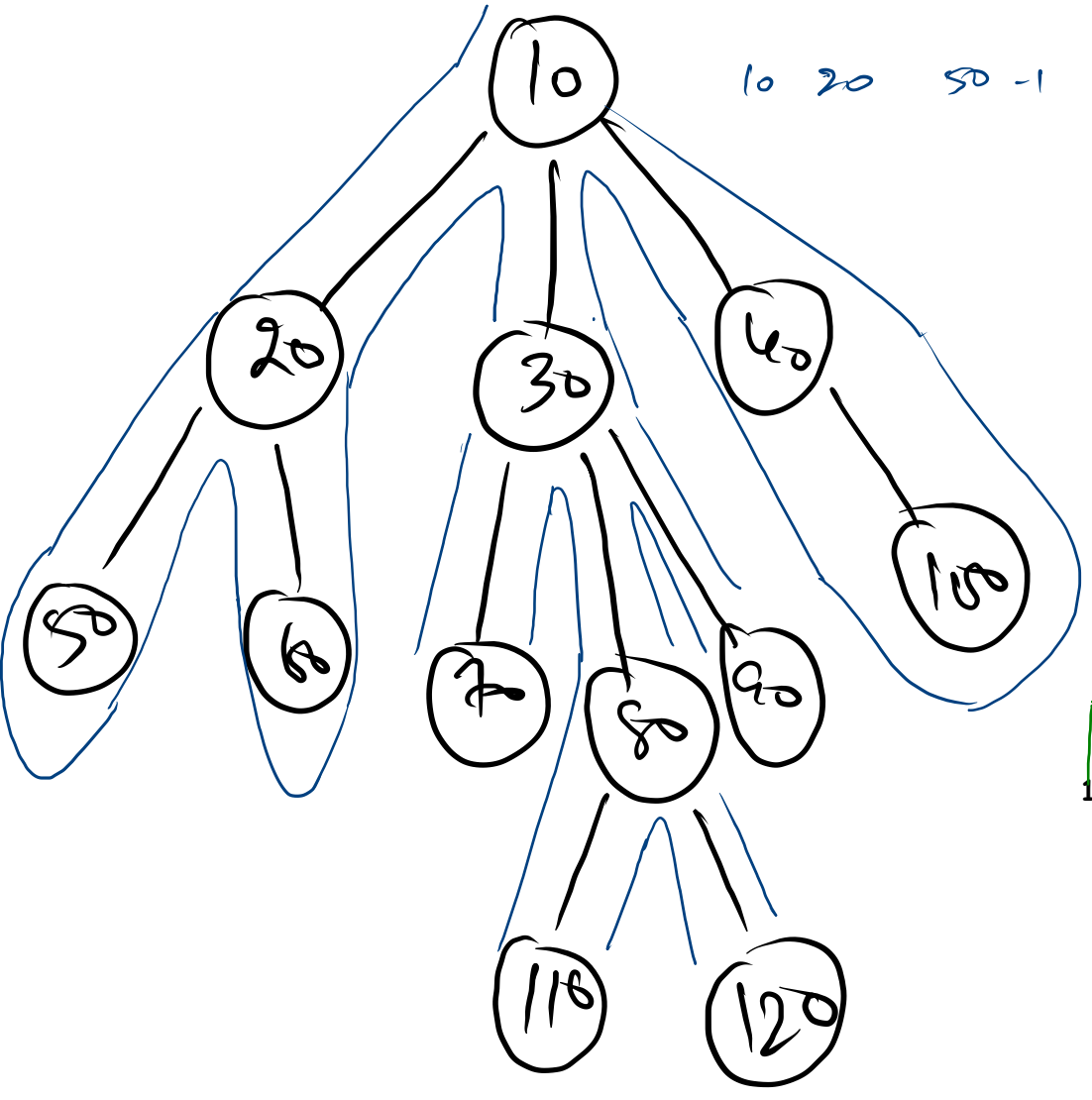
Tree → No Cycle



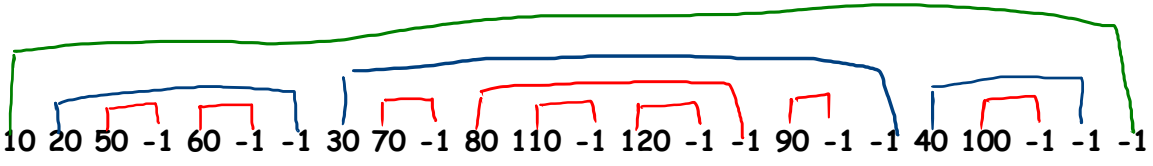
→ Node → data
→ ArrayList

→ Root

```
public static class Node {
    int data;
    ArrayList<Node> children;
}
```

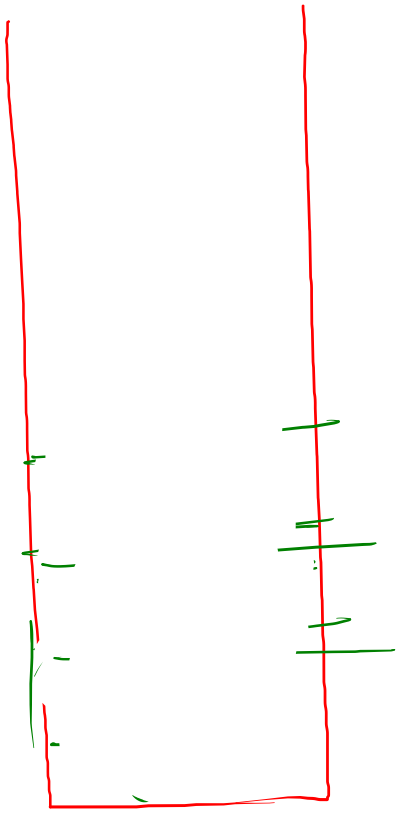


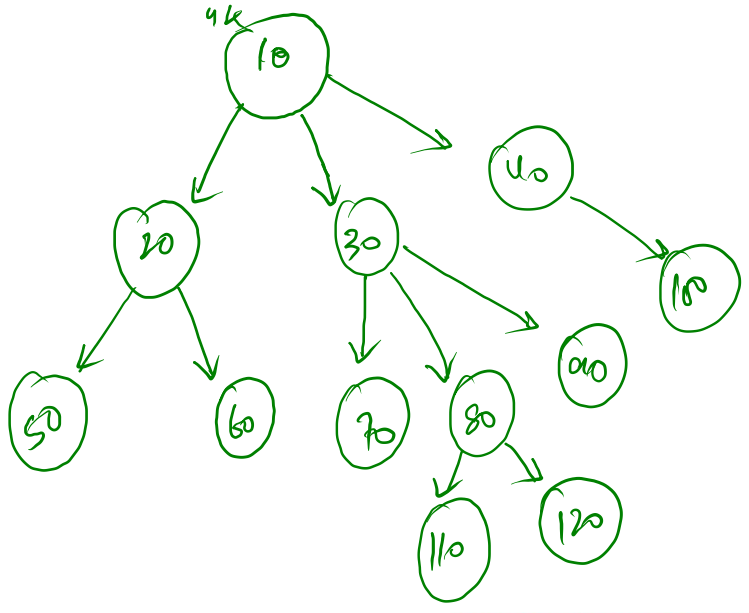
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1



Construct A Generic Tree

root = 4k





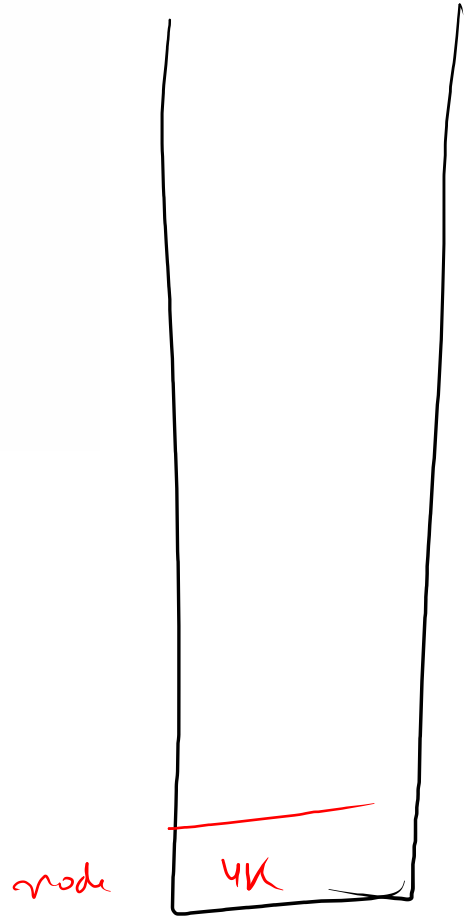
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1

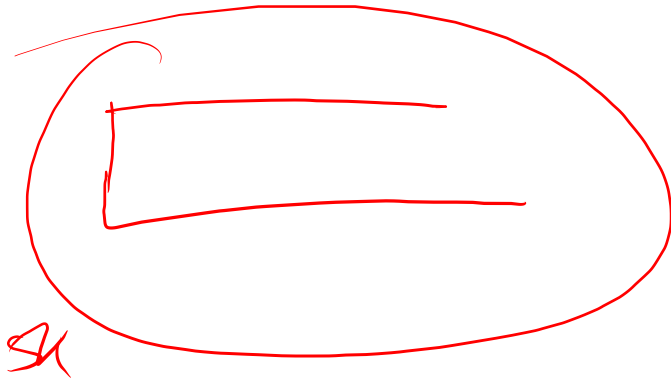
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

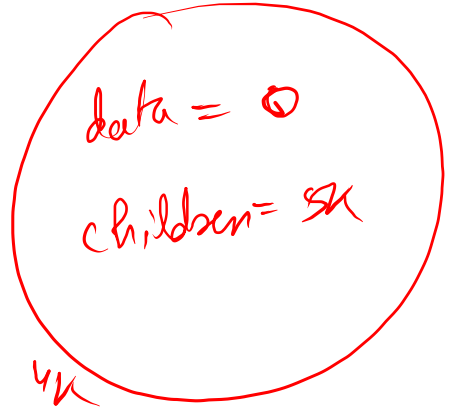
```java
public static Node construct(int input[]){
    Node root = new Node(input[0]);

    Stack<Node> st = new Stack<>();
    st.push(root);

    for(int i = 1 ; i < input.length ; i++){
        if(input[i] == -1){
            st.pop();
        }else{
            Node newNode = new Node(input[i]);
            st.peek().children.add(newNode);
            st.push(newNode);
        }
    }

    return root;
}
```

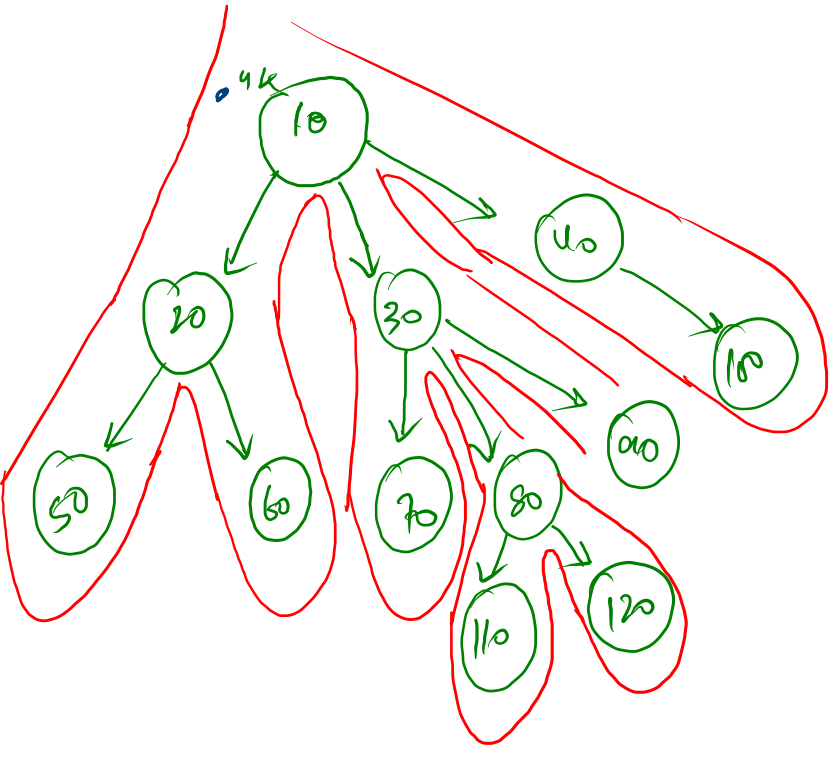
```java
public static class Node{
    int data;
    ArrayList<Node> children;

    Node(){
        children = new ArrayList<>();
    }
    Node(int val){
        data = val;
        children = new ArrayList<>();
    }
}
```

Node node = new Node();









10 → 20 30 40

20 → 50 60

50 →

60 →

30 → 70 80 90

70 →

80 → 110 120

110 →

120 →

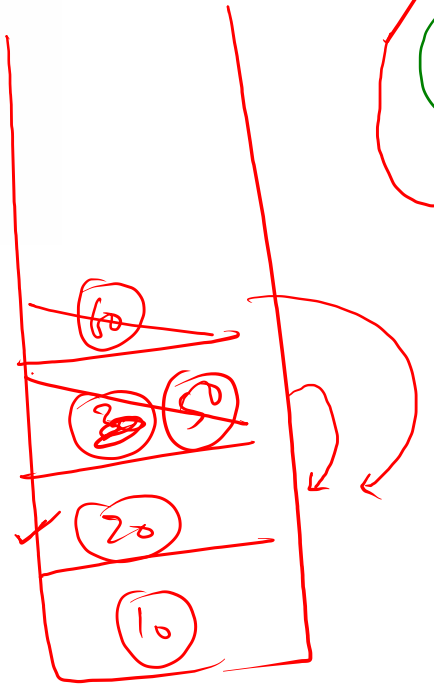
90 →

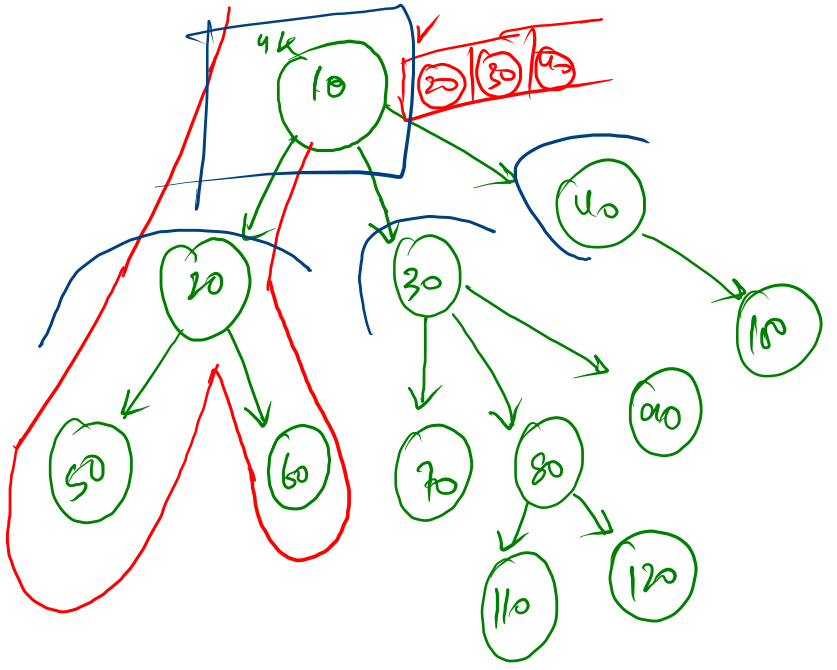
40 → 100

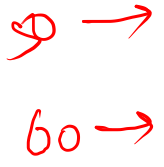
100 →

```java
public static void display(Node node){
    System.out.print(node.data +" -> " );
    for(Node child : node.children){
        System.out.print(child.data+" ");
    }
    System.out.println();

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

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





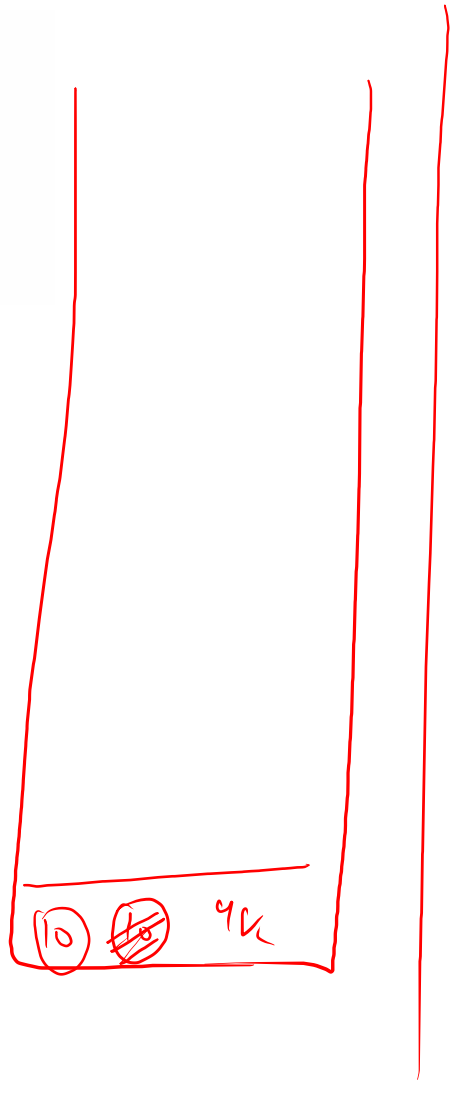


```java
public static Node construct(int input[]){
    Node root = new Node(input[0]);

    Stack<Node> st = new Stack<>();
    st.push(root);
```

push

- pop
- peek

```java
Node(int val){
    data = val;
    children = new ArrayList<>();
}
```

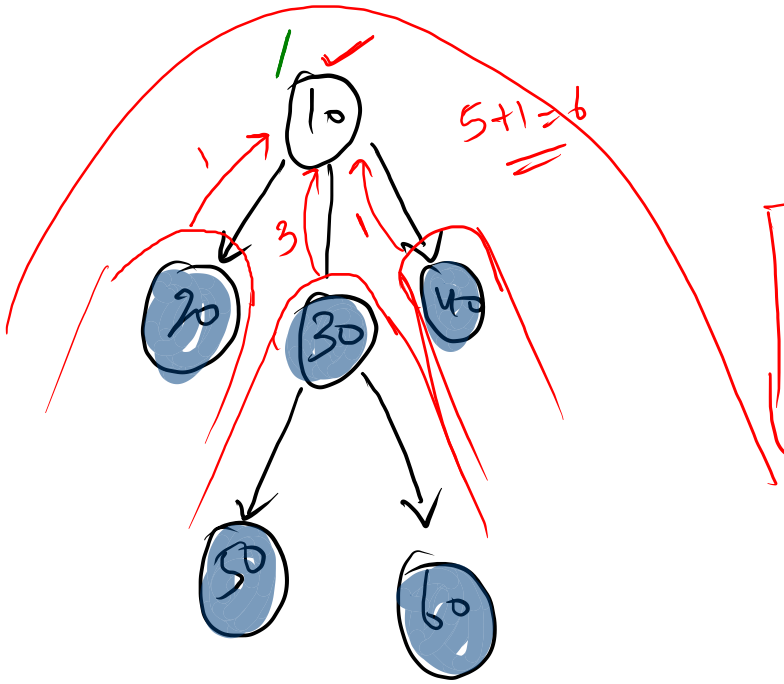


10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1

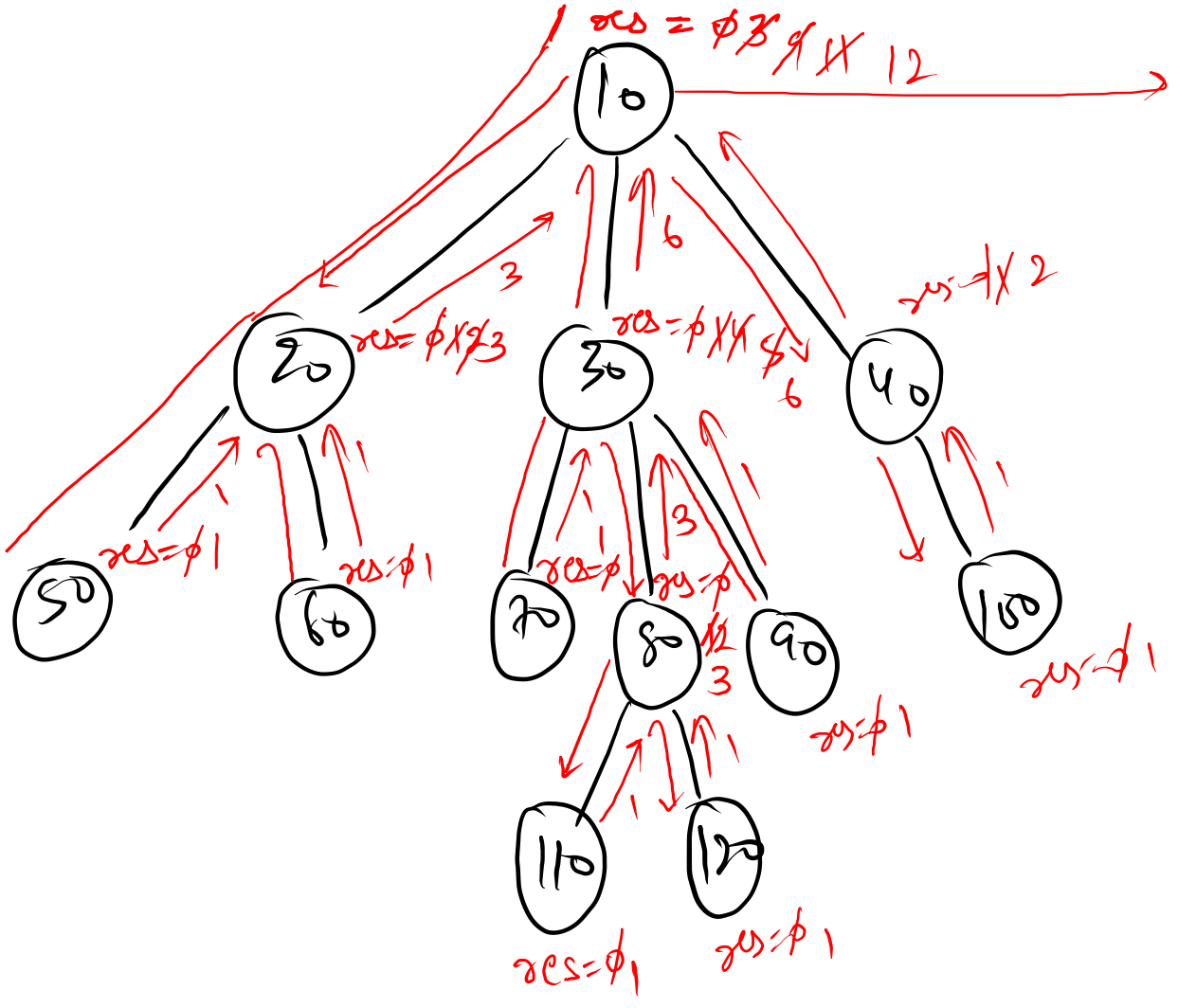
```
12
10 20 -1 30 50 -1 60 -1 -1 40 -1 -1
```

Size of Tree ⇒ No. of nodes in a Tree

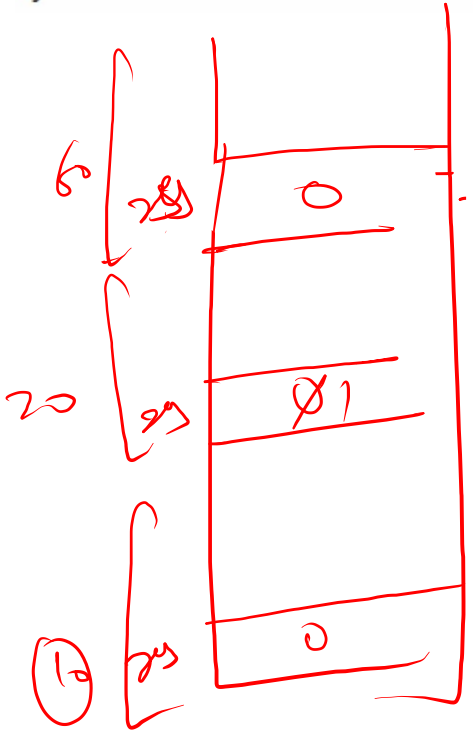
Expect → no. of nodes of a tree (root)

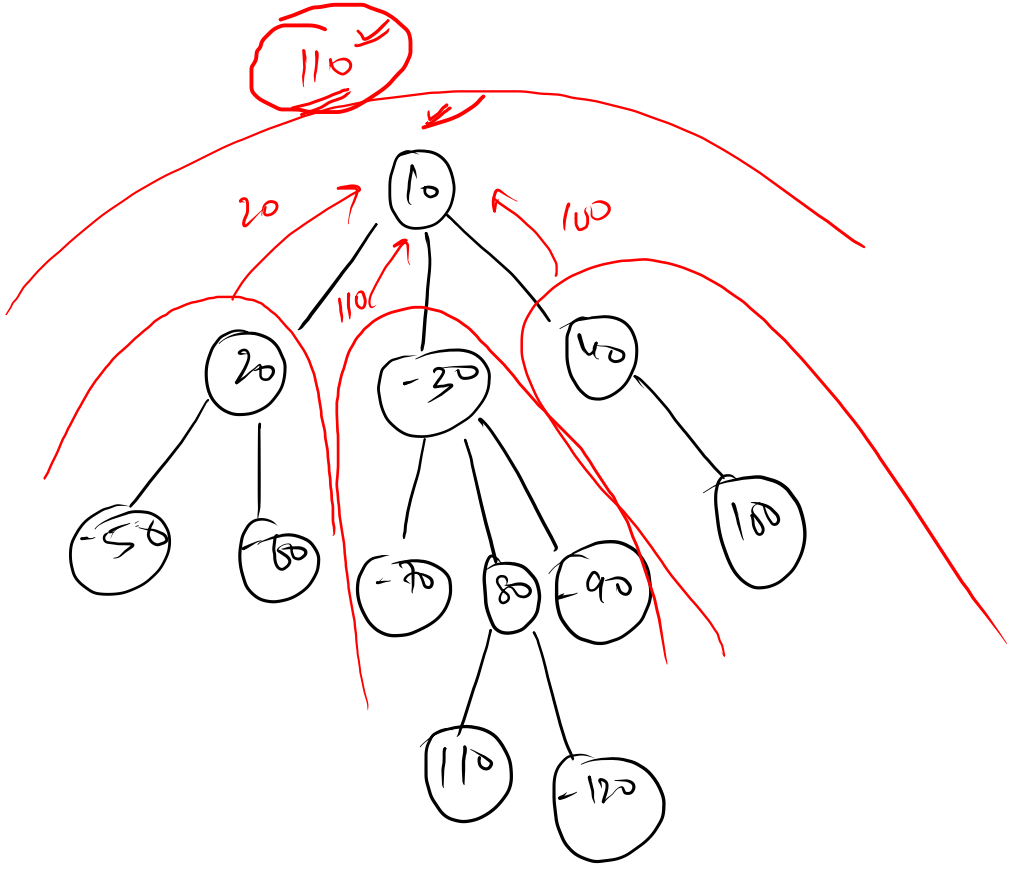


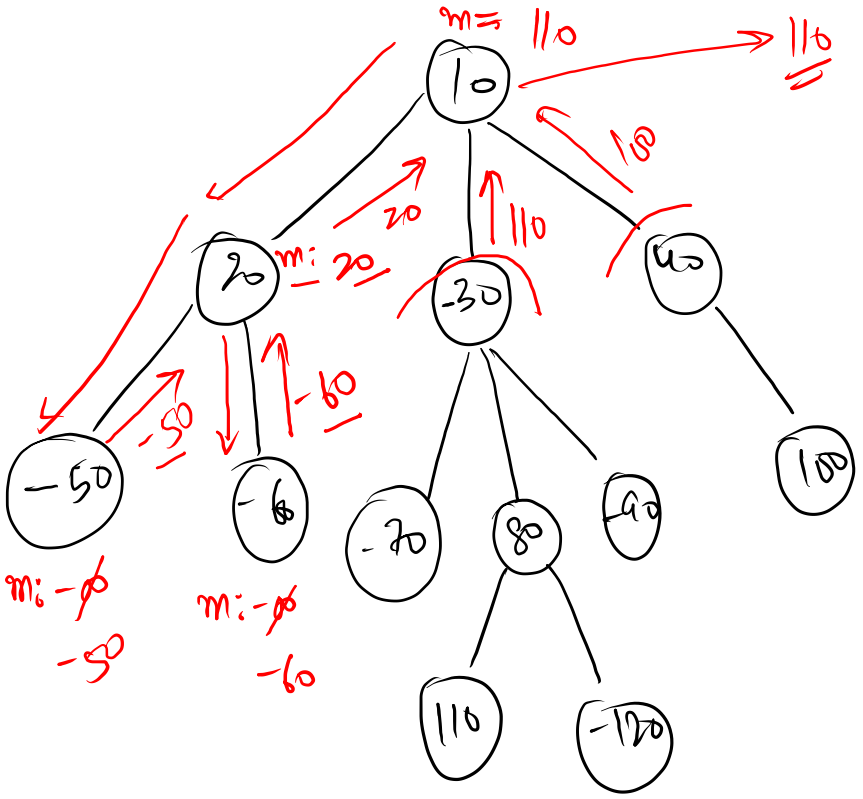
no. of nodes of Tree ⇒ [Σ (no. of nodes of child-Trees)] + 1



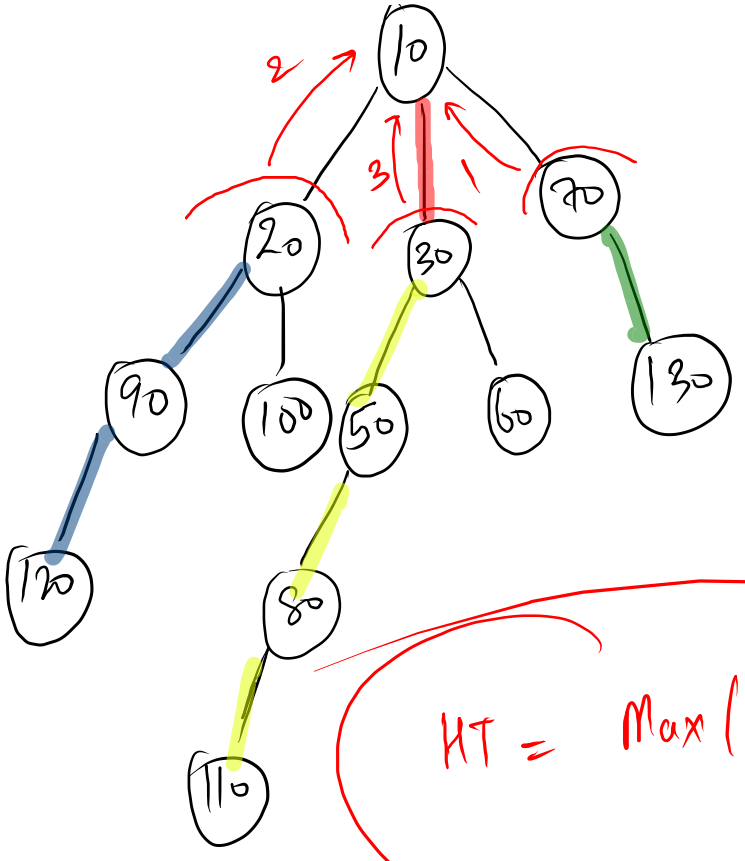

```java
public static int size(Node node){
    int res = 0;
    for(Node child : node.children){
        res += size(child);
    }

    res += 1;

    return res;
}
```




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




```java
public static int max(Node node) {
    int myMax = Integer.MIN_VALUE;

    for(Node child : node.children){
        myMax = Math.max(myMax,max(child));
    }

    myMax = Math.max(myMax, node.data);

    return myMax;
}
```



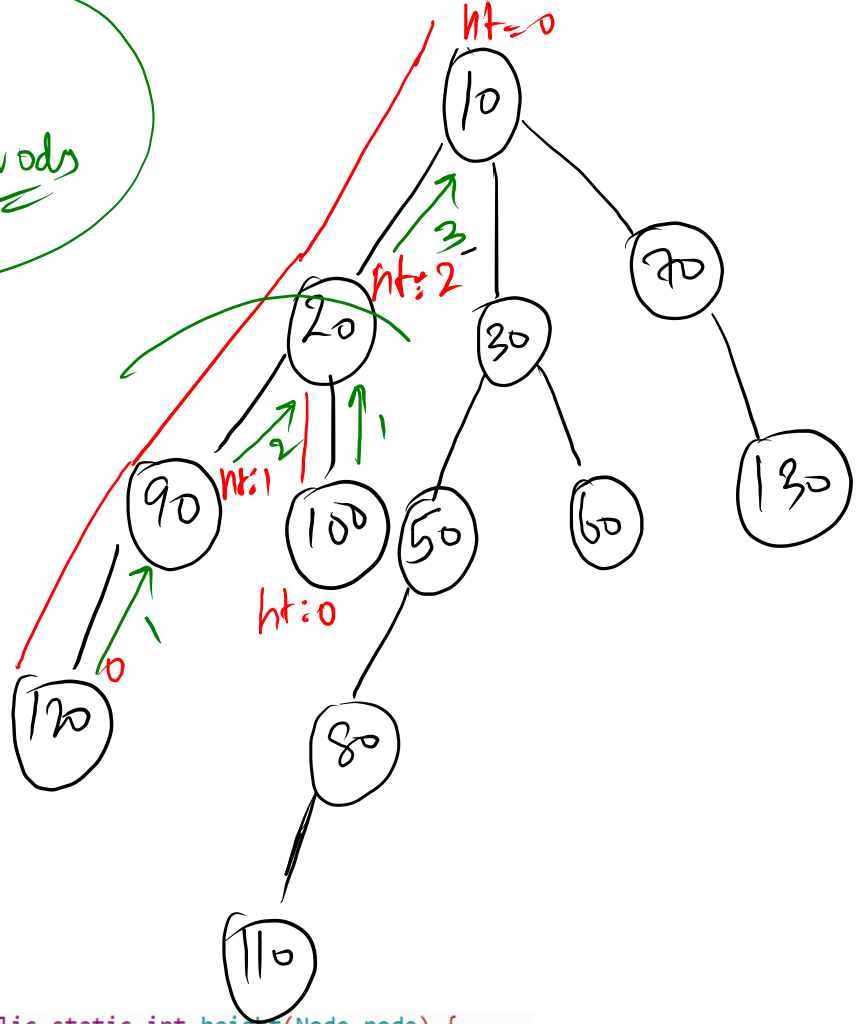
Height of a generic Tree

Distance of root Node from it's deepest Node

- Edges : 4
- Nodes : 5

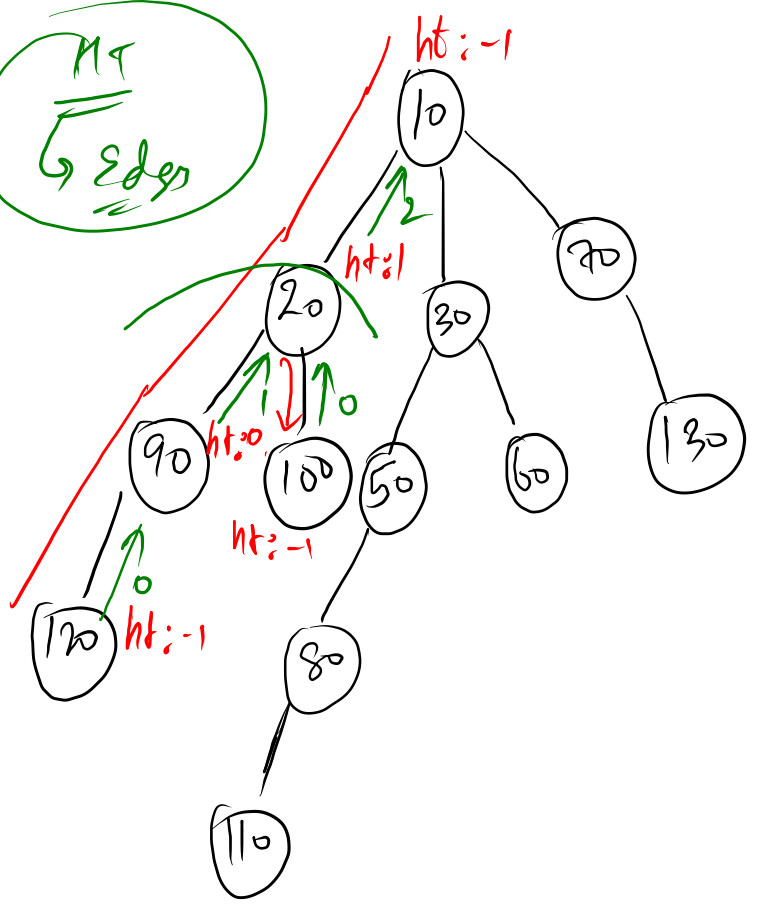
HT = Max (childHT) + 1

HT
↳ Nodes




```
public static int height(Node node) {
  int ht = 0;
  for(Node child : node.children){
      ht = Math.max(ht,height(child));
  }

  return ht+1;
}
```

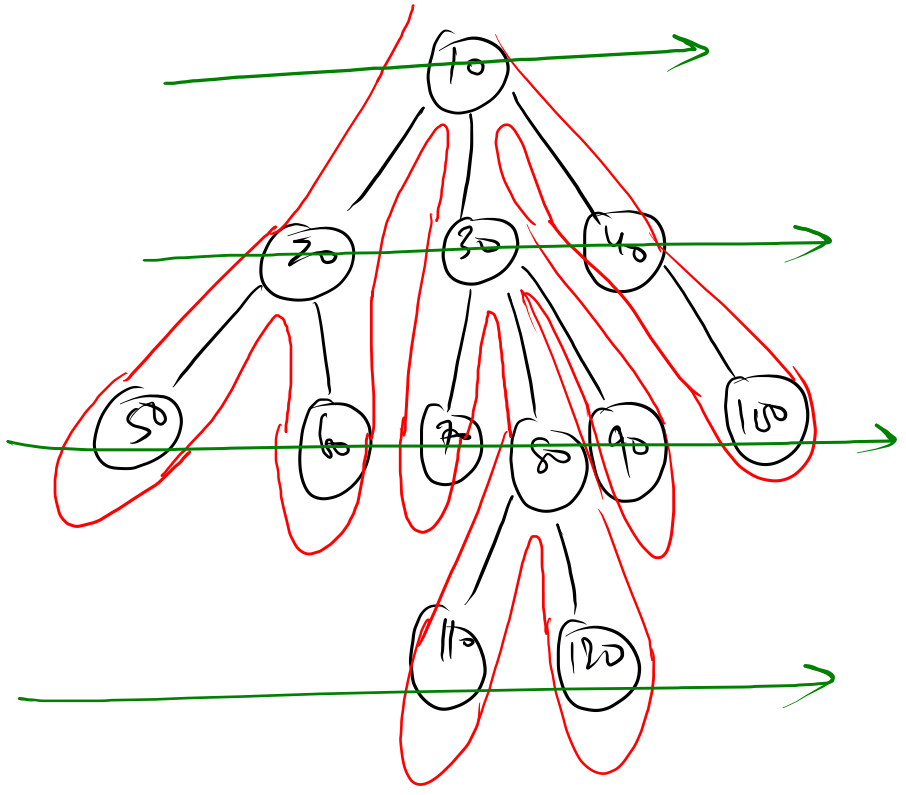
HT
↳ Edges




```
public static int height(Node node) {
  // int ht = 0;
  int ht = -1;
  for(Node child : node.children){
      ht = Math.max(ht,height(child));
  }

  return ht+1;
}
```

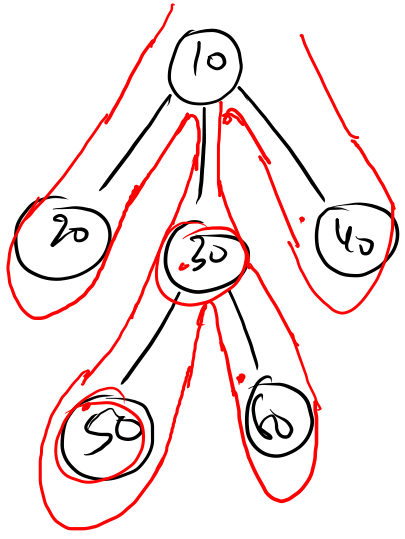
Traversal

- Stack
  - Preorder
  - Postorder
  - Level order



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

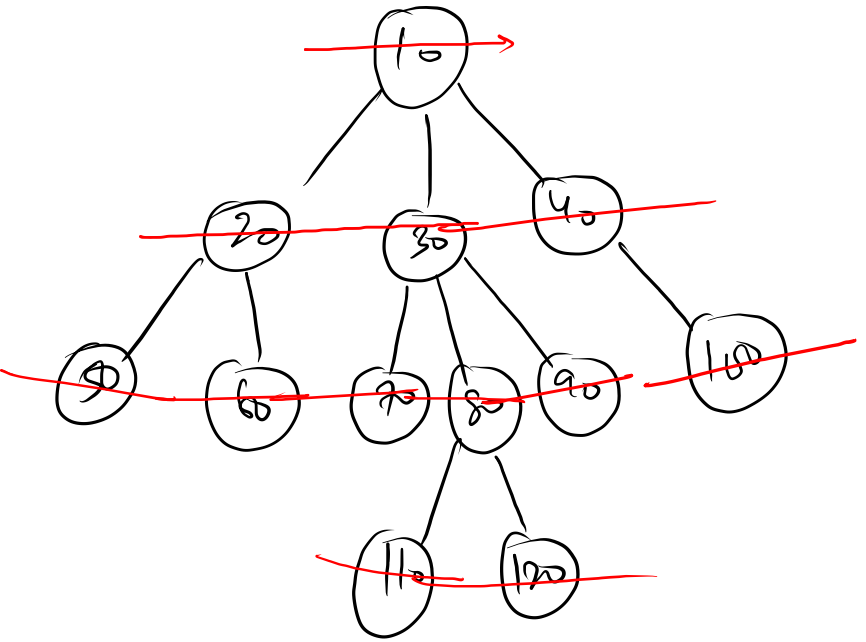
Preorder	Postorder
10	50
20	60
50	20
60	70
30	110
70	120
80	80
90	90
110	30
120	100
40	40
100	10



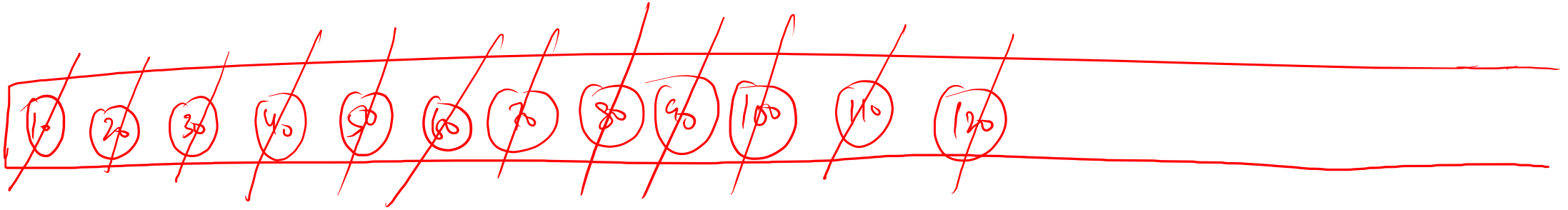

```java
public static void traversals(Node node){
  // pre-area
  System.out.println("Node Pre "+node.data);

  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);
  }

  // post-area
  System.out.println("Node Post "+node.data);

}
```

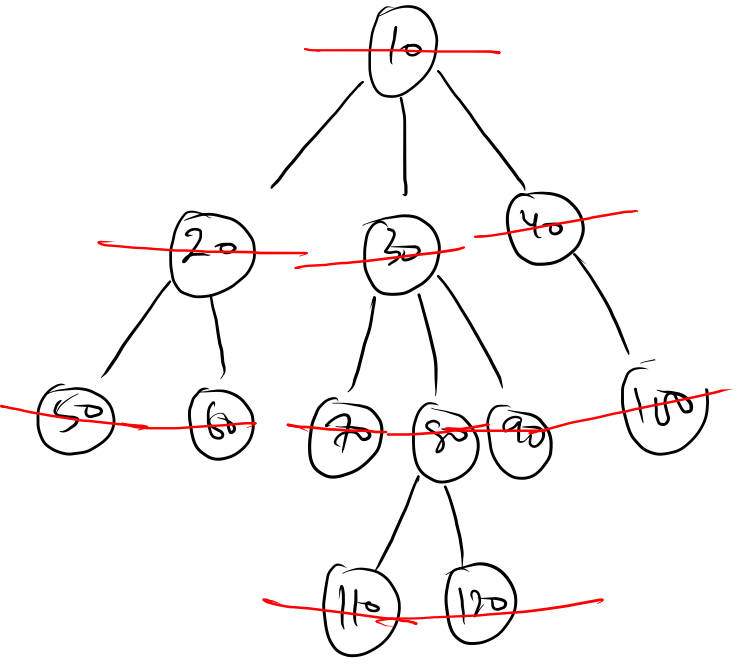
Node Pre 10
Edge Pre 10 -- 20
Node Pre 20
Node Post 20
Edge Post 10 -- 20
Edge Pre 10 -- 30
Node Pre 30
Edge Pre 30 -- 50
Node Pre 50
Node Post 50
Edge Post 30 -- 50
Edge Pre 30 -- 60
Node Pre 60
Node Post 60
Edge Post 30 -- 60
Node Post 30
Edge Post 10 -- 30
Edge Pre 10 -- 40
Node Pre 40
Node Post 40
Edge Post 10 -- 40
Node Post 10

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

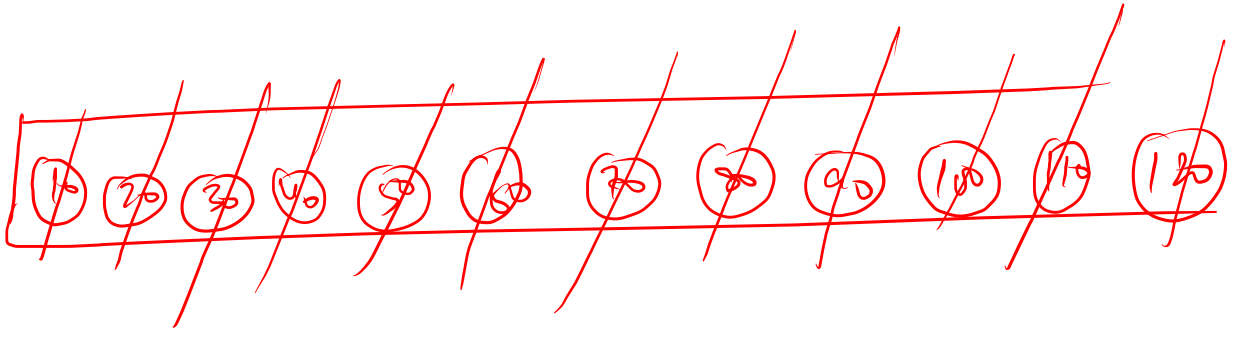


r
p
a

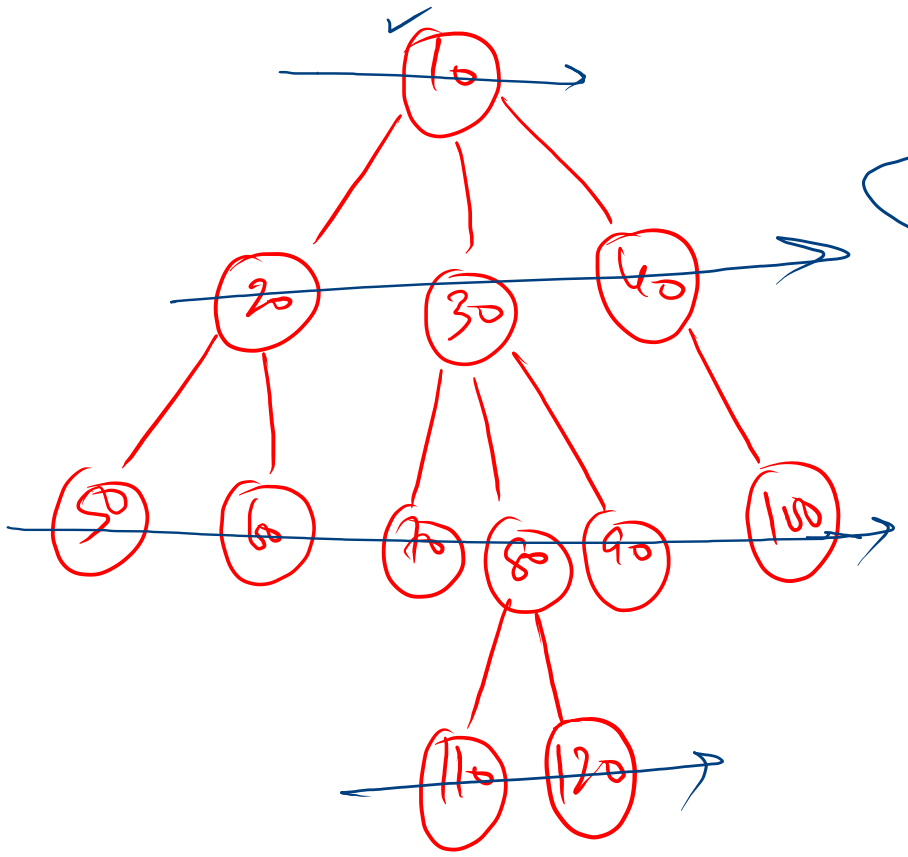


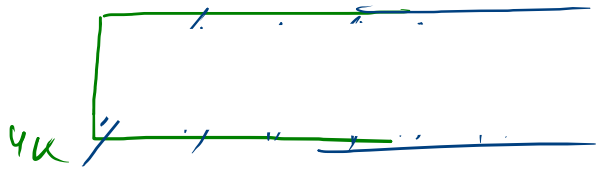


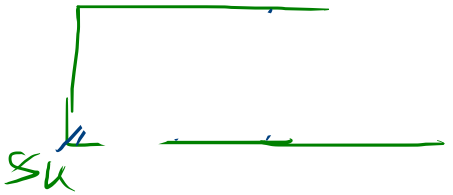
10 20 30 40 50 60 70 80 90 100 110 120.




```java
public static void levelOrder(Node root) {
  Queue<Node> queue = new ArrayDeque<>();

  queue.add(root);

  while (queue.size() > 0) {
    Node fnode = queue.remove();//remove
    System.out.print(fnode.data + " "); // print

    // add children to queue
    for (Node child : fnode.children) {
      queue.add(child);
    }
  }
  System.out.println(".");
}
```







mainQ = 4K 5K 4K

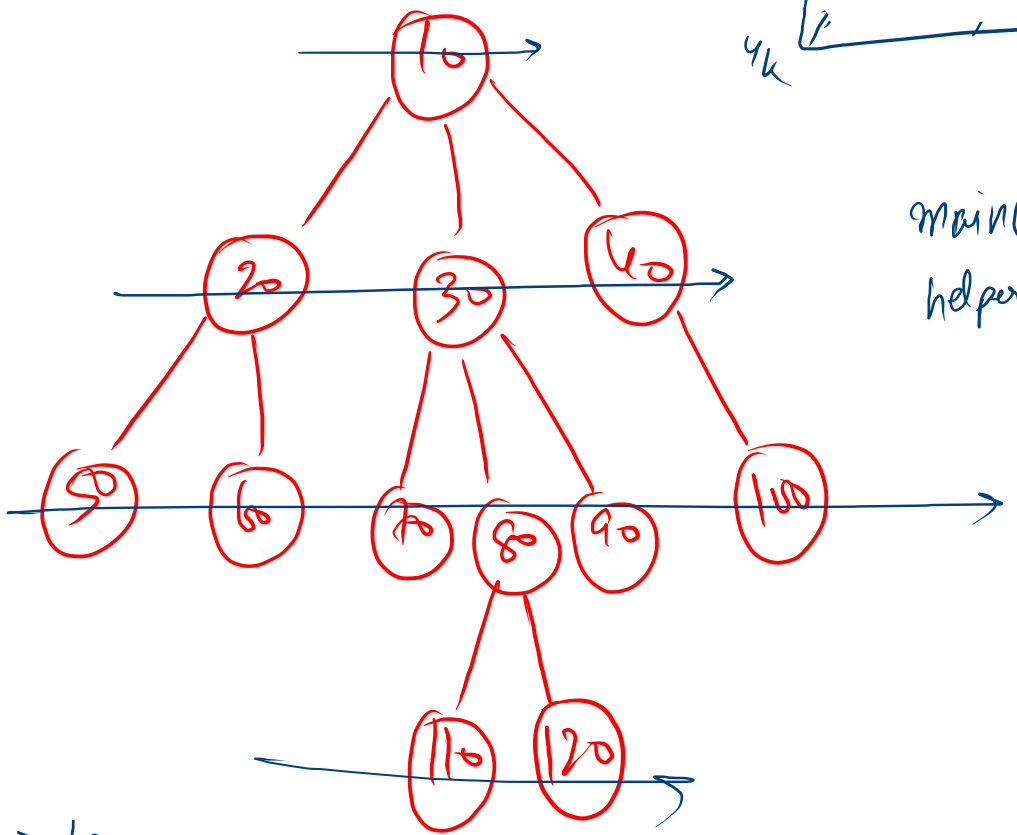
HelperQ = 5K 4K 5K

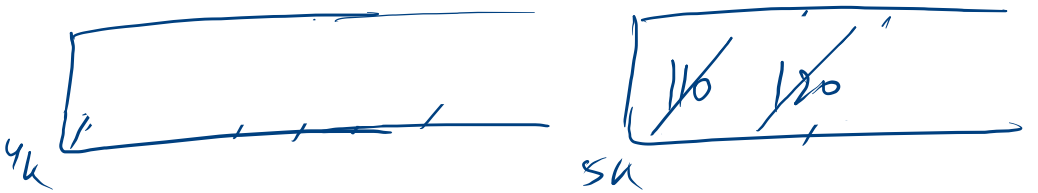
10

20 30 40

50 60 70 80 90 100

110 120





mainQ : 4k

helperQ : 5k

```java
public static void levelOrderLinewise(Node root){
    Queue<Node> mainq = new ArrayDeque<>();
    Queue<Node> helperq = new ArrayDeque<>();

    mainq.add(root);

    while(mainq.size() > 0){
        Node fnode = mainq.remove();//remove
        System.out.print(fnode.data+" "); // print

        // add children to helperq
        for(Node child : fnode.children){
            helperq.add(child);
        }

        if(mainq.size() == 0){
            System.out.println();

            Queue<Node> tempq = mainq;
            mainq = helperq;
            helperq = tempq;
        }
    }
}
```

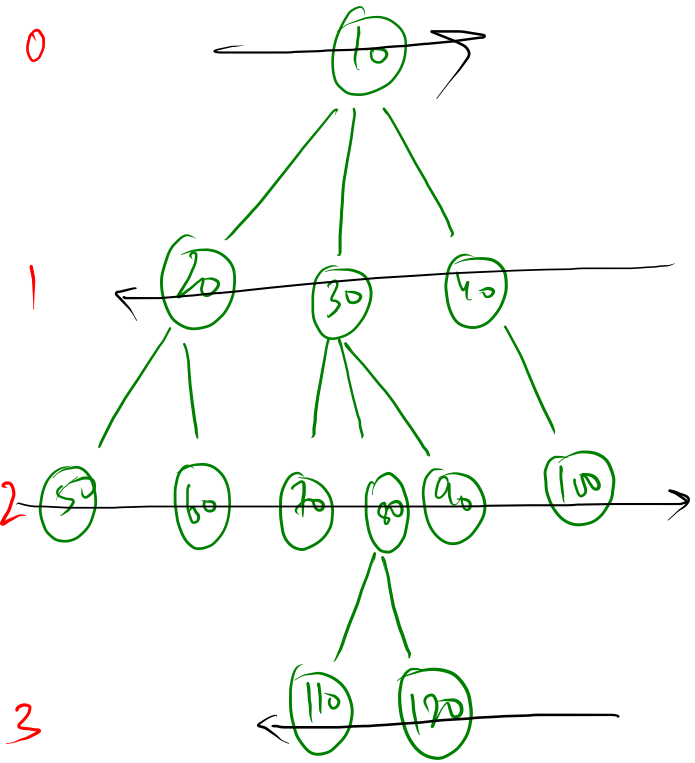
→ 10

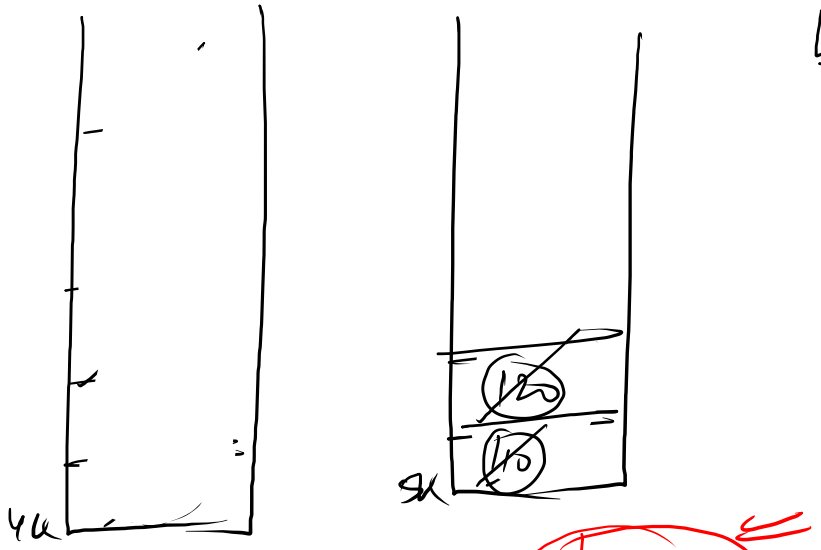
→ 20 30 40

→ 50 60 70 80 90 100

→ 110 120

→





helpers = 8k

mains = 4k

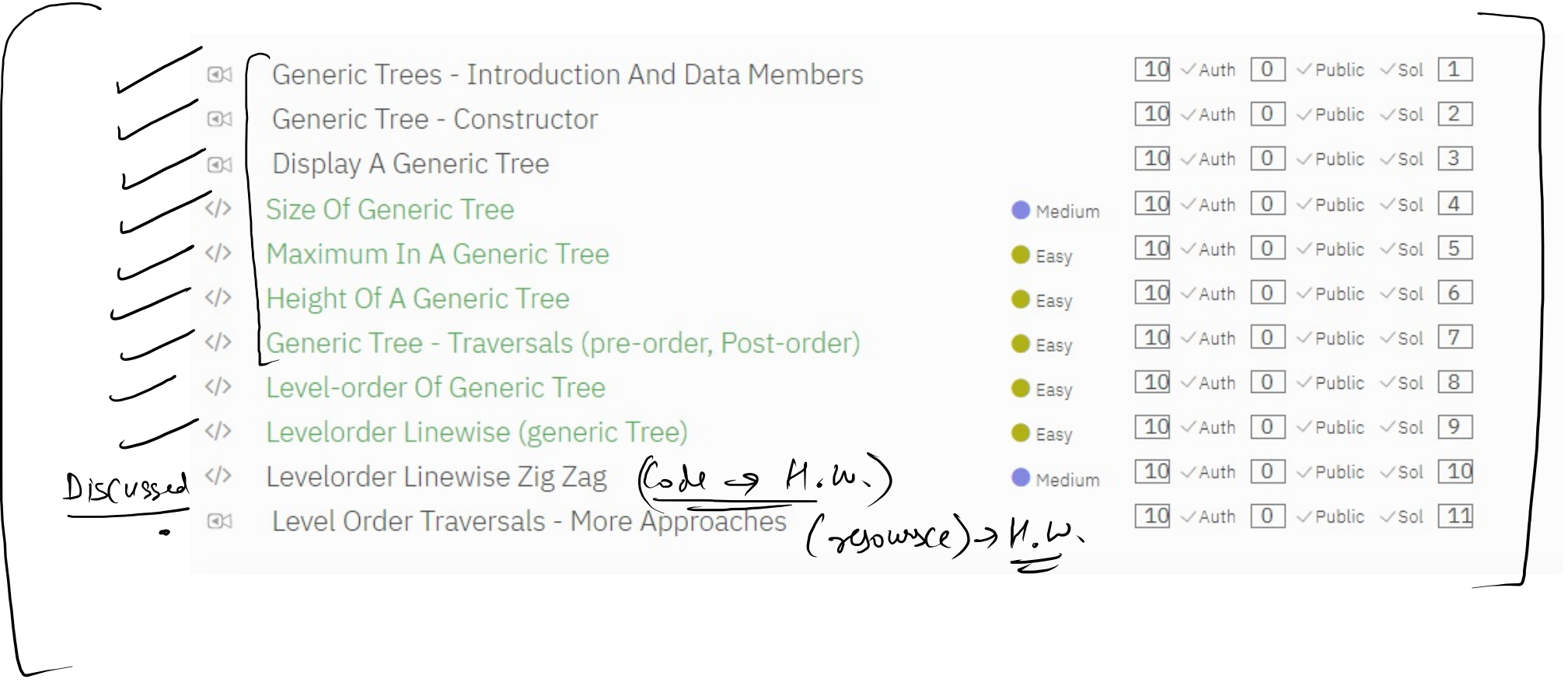
H.W.

Level = ~~0~~ X ~~2~~ 3
4

```
if (level == Even){
    add to helper
    normal
}else{
    add to helper
    in reverse format
}
```

```
if (mains.size ==0) {
    level++
    nextline
    mains <-> helpers
}
```

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


Generic Trees - Introduction And Data Members
10 Auth 0 Public Sol 1
Generic Tree - Constructor
10 Auth 0 Public Sol 2
Display A Generic Tree
10 Auth 0 Public Sol 3
Size Of Generic Tree
Medium
10 Auth 0 Public Sol 4
Maximum In A Generic Tree
Easy
10 Auth 0 Public Sol 5
Height Of A Generic Tree
Easy
10 Auth 0 Public Sol 6
Generic Tree - Traversals (pre-order, Post-order)
Easy
10 Auth 0 Public Sol 7
Level-order Of Generic Tree
Easy
10 Auth 0 Public Sol 8
Levelorder Linewise (generic Tree)
Easy
10 Auth 0 Public Sol 9
Levelorder Linewise Zig Zag
Medium
10 Auth 0 Public Sol 10
Level Order Traversals - More Approaches
10 Auth 0 Public Sol 11
Discussed
(Code → H.W.)
(resource) → H.W.