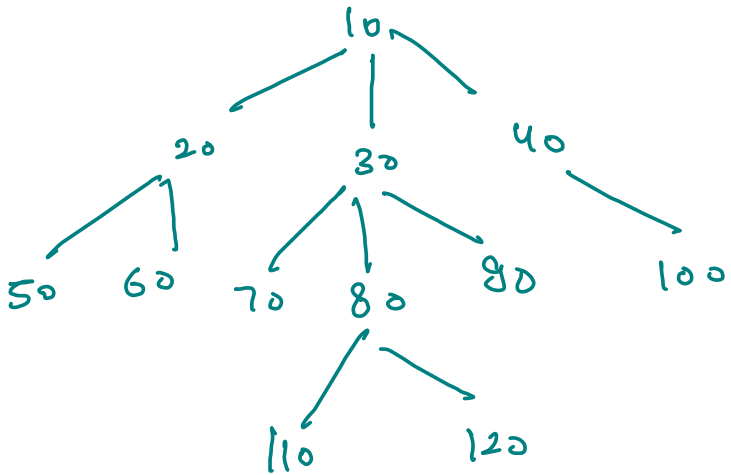
level order line wise



helperQ:

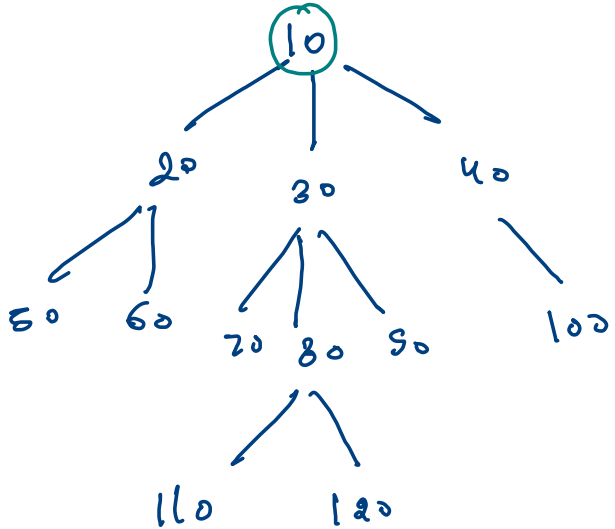
mainQ

10

20 30 40

50 60 70 80 90 100

110 120



~~80 60 70 80 80 100 110 120~~

```
while(que.size()>0) {
    int sz = que.size();
    while(sz-->0) {
        // get + remove
        // print
        // add children
    }
}
```

size of ~~3~~ ~~6~~ 2

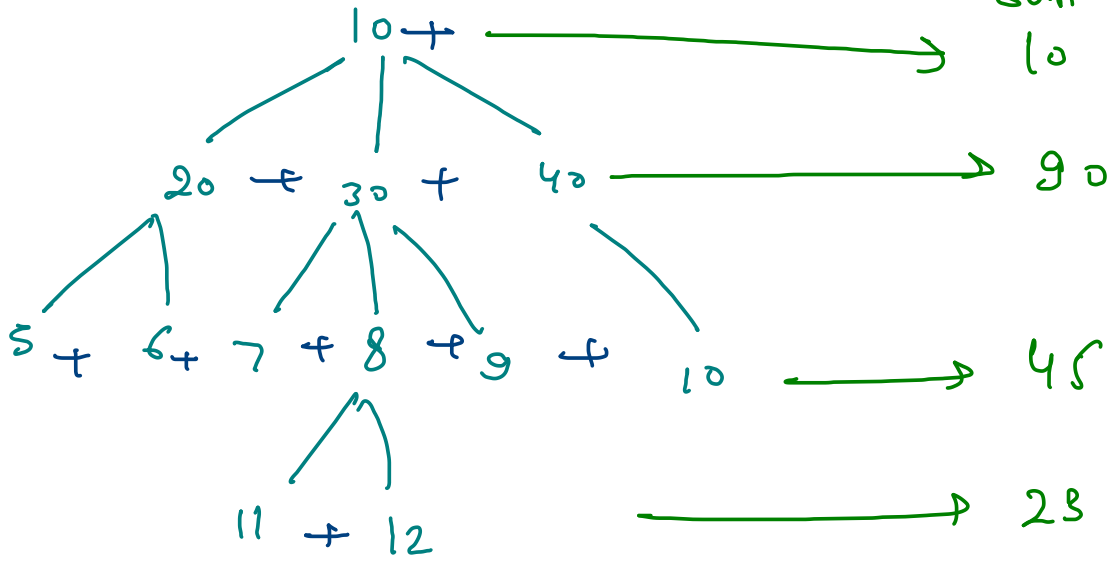
10

20 30 40

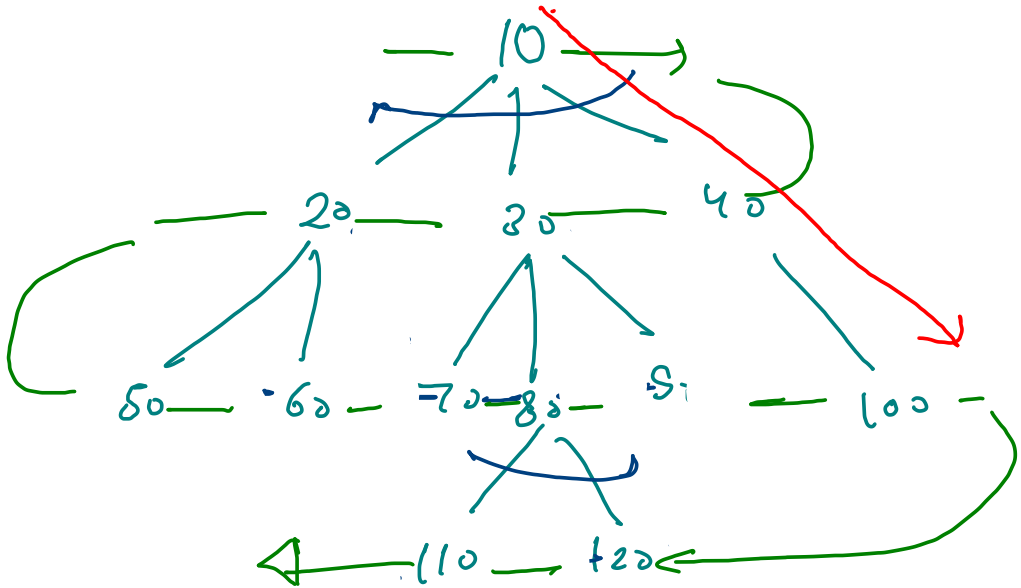
50 60 70 80 90 100

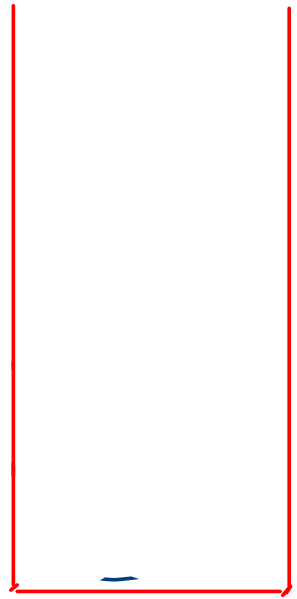
100 120

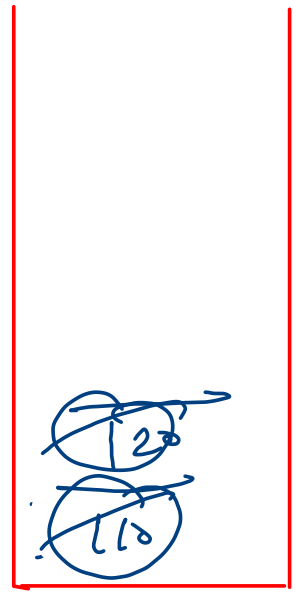
sum of Every level "print"



level order Zig Zag.



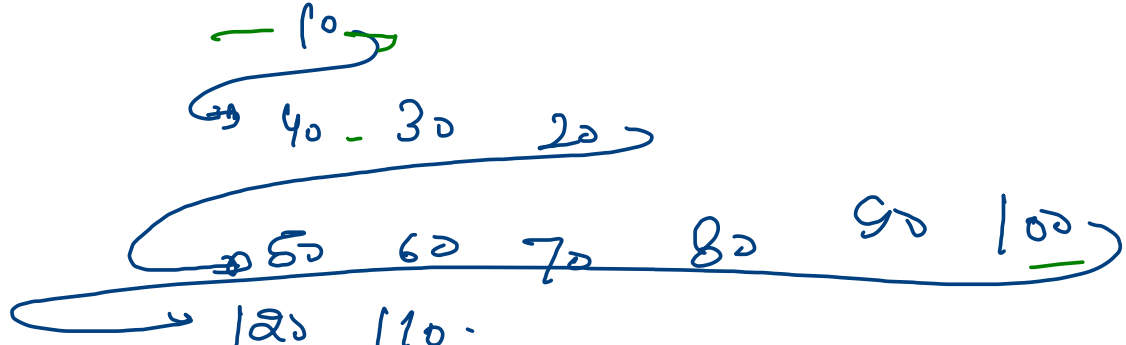




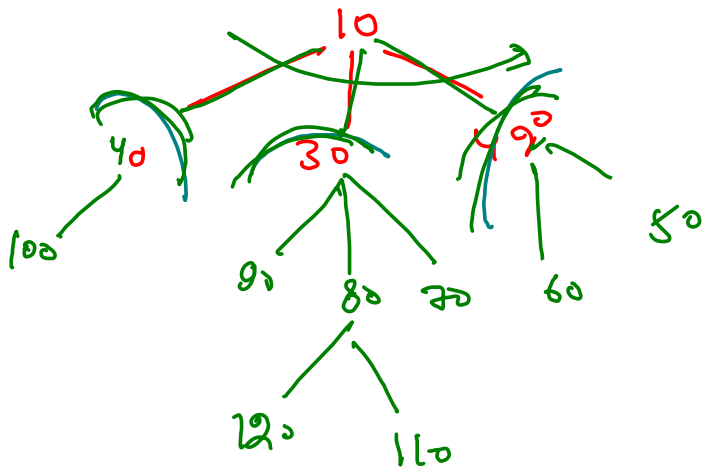
level = 0 1 2

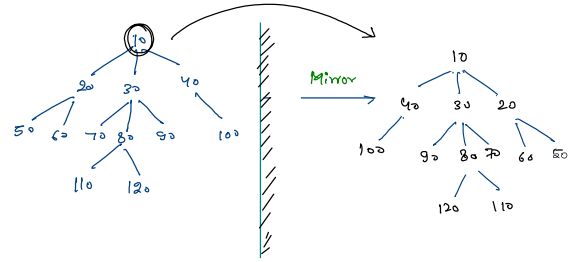
even → left to right

odd → right to left



Mirror Of Generic Tree



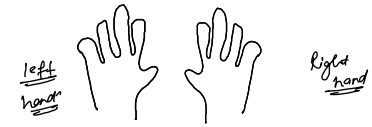


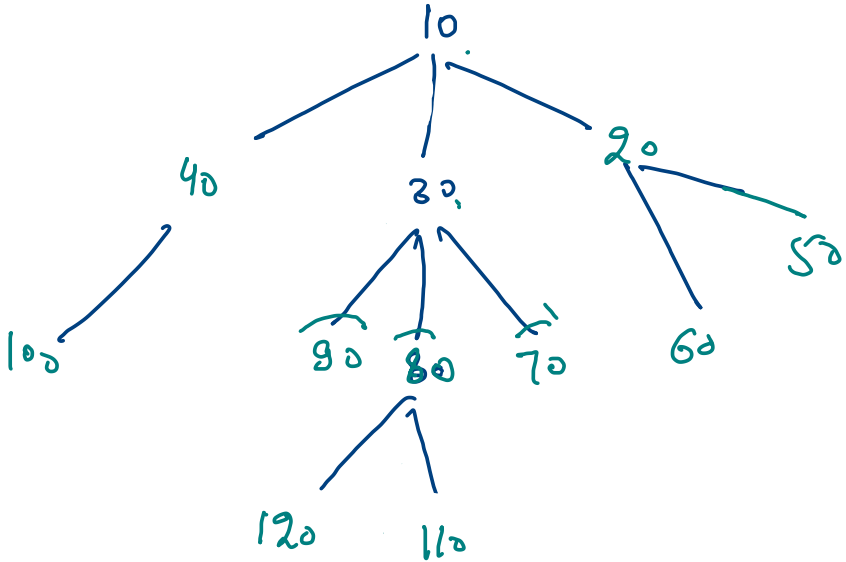
Expectation: mirror(10) → [

faith → { mirror(20), mirror(30), mirror(40)

Merging → self + children mirror

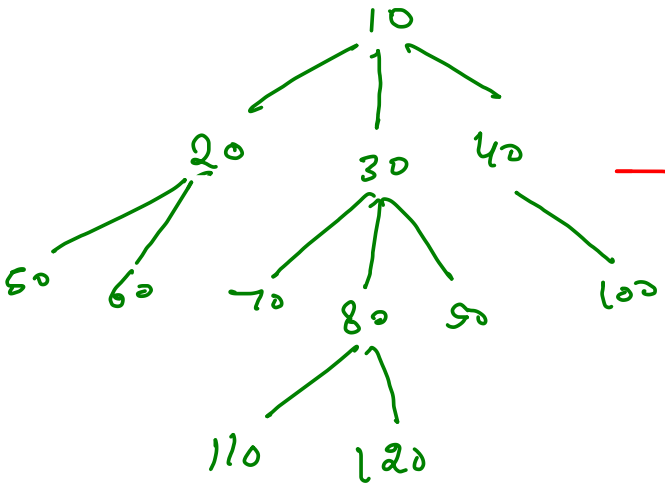
→ [revers of my child.






```java
public static void mirror(Node node){
    // faith
    for(Node child : node.children) {
        mirror(child);
    }
    // merging -> reverse nodde.children
    int left = 0;
    int right = node.children.size() - 1;

    while(left < right) {
        Node data1 = node.children.get(left);
        Node data2 = node.children.get(right);

        node.children.set(left, data2);
        node.children.set(right, data1);

        left++;
        right--;
    }
}
```

Linearize a Generic Tree



Linearize.

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

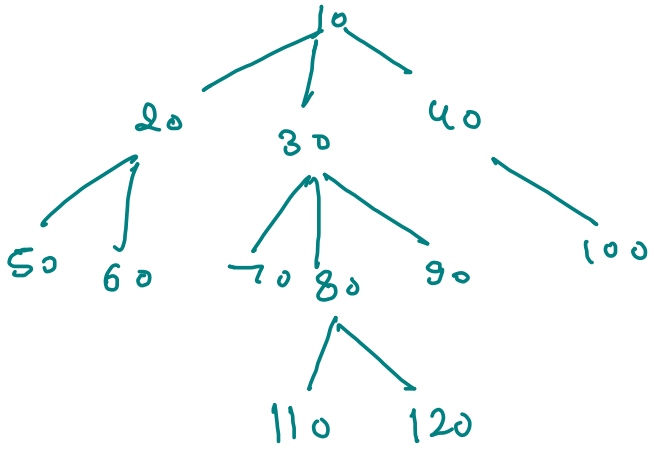
pre order.

convert Generic Tree into Linearise tree

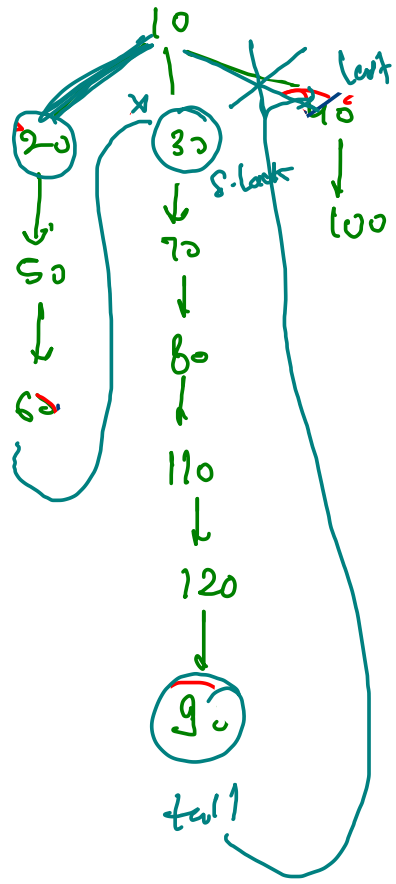
mirror Code:

```java
public static void mirror(Node node){
    // faith
    for(Node child : node.children) {
        mirror(child);
    }
    // merging -> reverse nodde children
    int left = 0;
    int right = node.children.size() - 1;

    while(left < right) {
        Node data1 = node.children.get(left);
        Node data2 = node.children.get(right);

        node.children.set(left, data2);
        node.children.set(right, data1);

        left++;
        right--;
    }
}
```

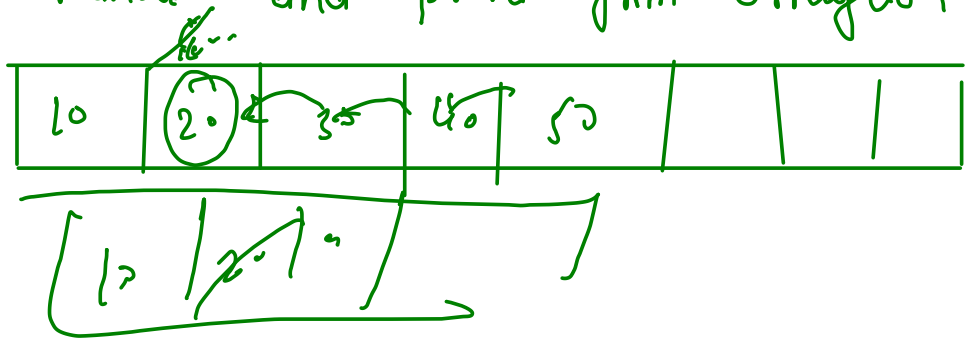
ArrayList<Node> children

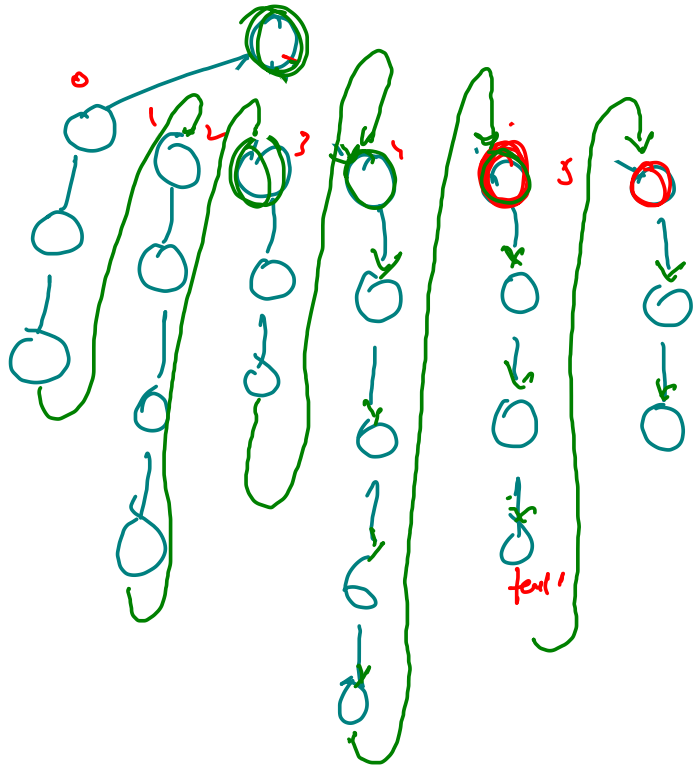


After faith



Remove and print from arraylist, only for loop.

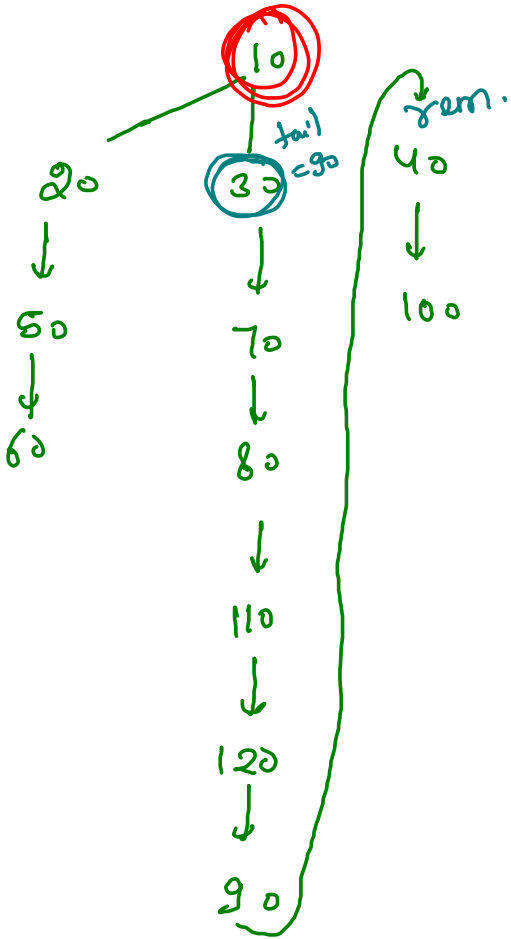




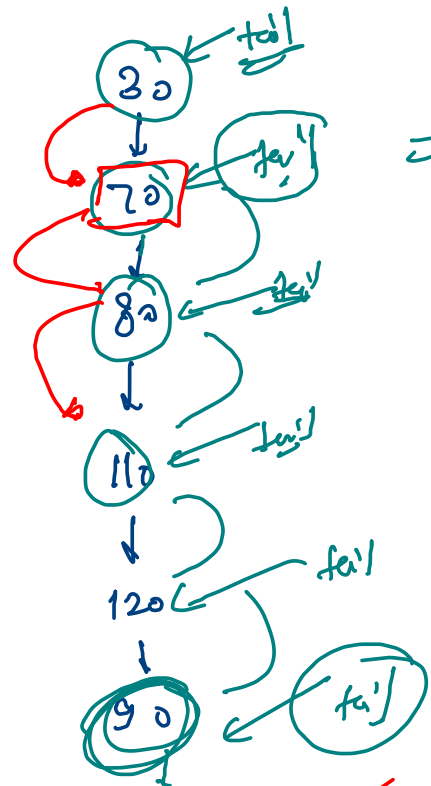
① last node remove + get

② get tail for second last node, which is currently last node.

③ tail.children(rem)




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

public static void linearize(Node root) {
    for(Node child : root.children) {
        linearize(child);
    }
    // make connection
    for(int i = root.children.size() - 2; i >= 0; i--) {
        Node rem = root.children.remove(i + 1);
        Node tail = getTail(root.children.get(i));
        tail.children.add(rem);
    }
}
```

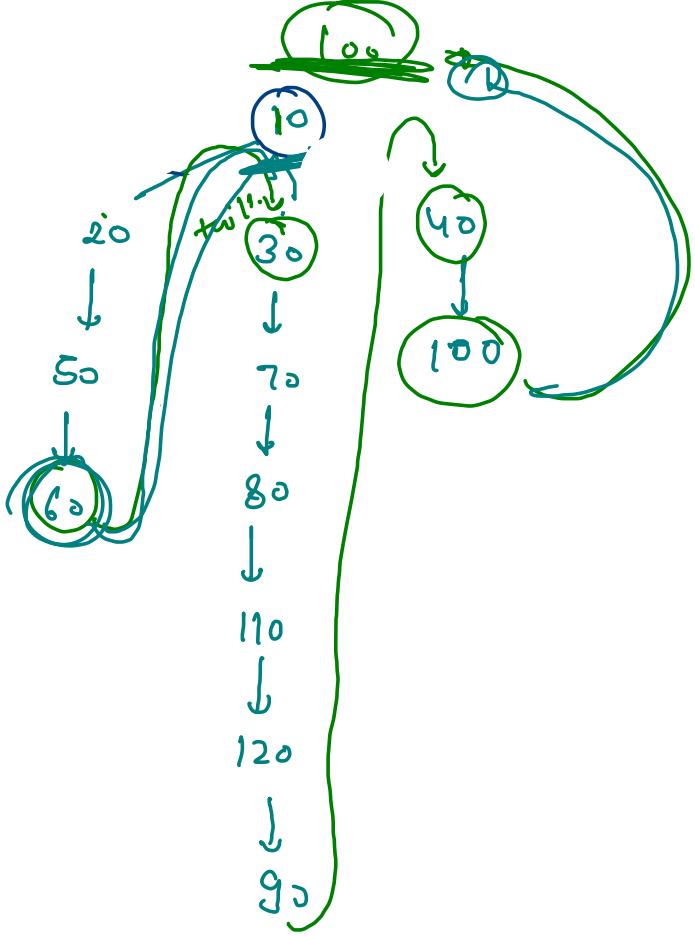


for every Element

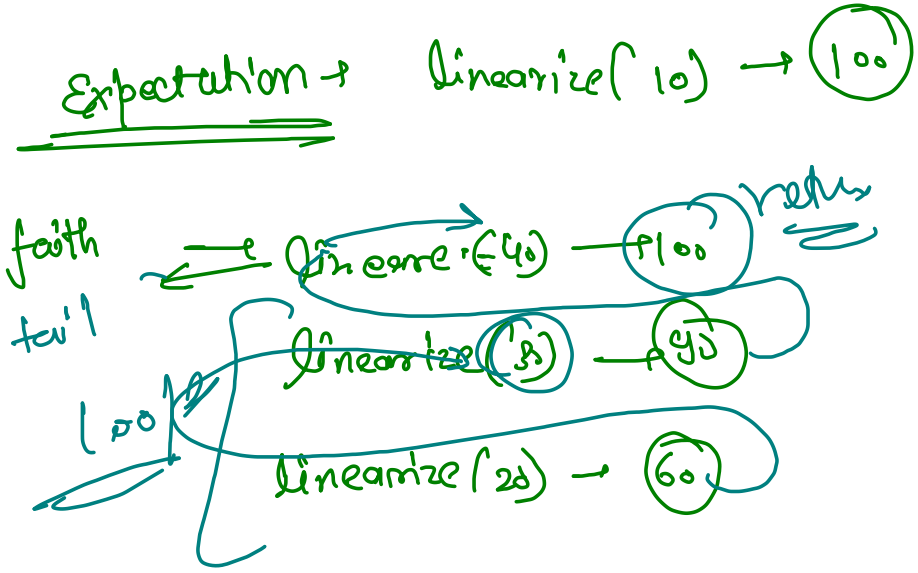
getTail → O(h)

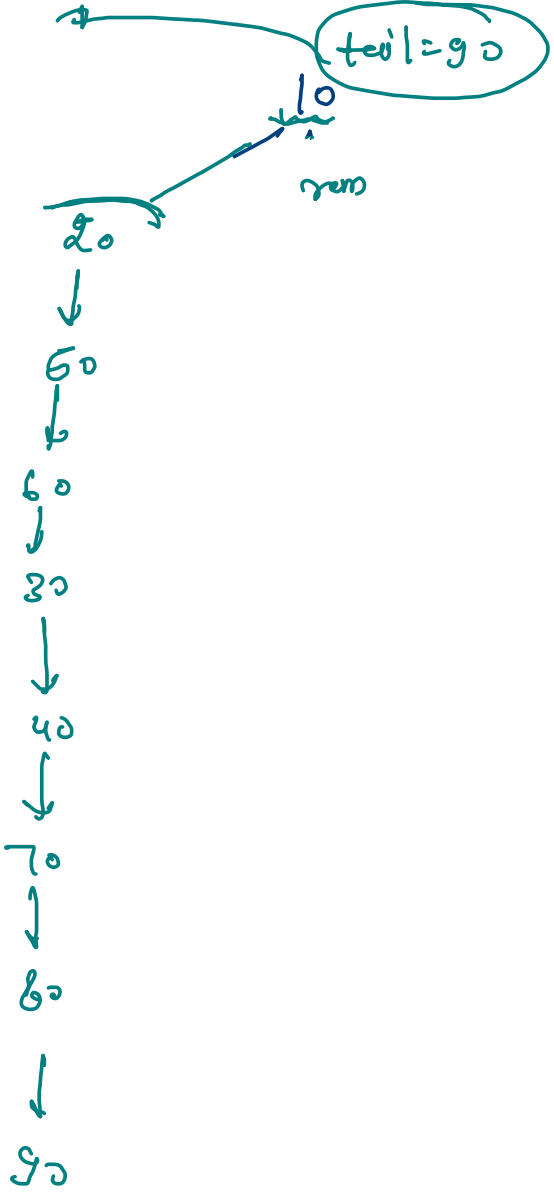
complexity ⇒ O(h)

overall complexity = $O(h^2)$



Expectation → linearize(10) → 100





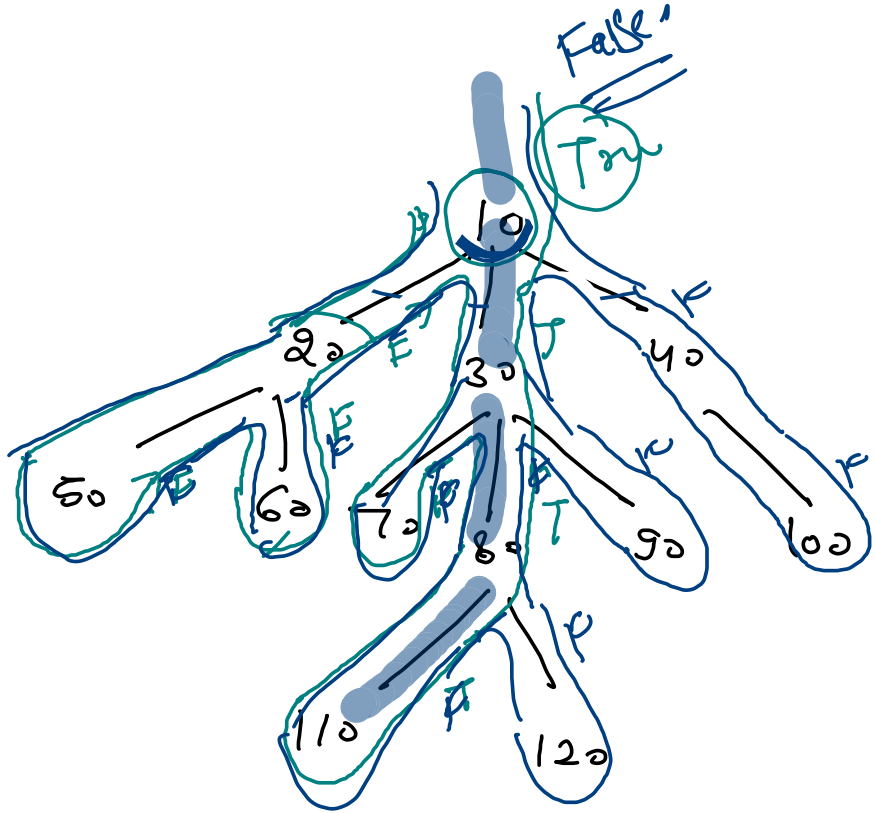

```java
public static Node lineariseBtr(Node root) {
    if(root.children.size() == 0) return root;
    int n = root.children.size();
    Node tail = lineariseBtr(root.children.get(n - 1));
    for(int i = n - 2; i >= 0; i--) {
        Node rem = root.children.remove(i + 1);
        Node ntail = lineariseBtr(root.children.get(i));
        ntail.children.add(rem);
    }
    return tail;
}
```

Normal Return type traversal.

only change is direction of traversal as reverse.

find -

→ 110
35



Expectation : find(10, data)
→ True
→ False

Self check

faith → find(20, data) → T / F

find(30, data) → T / F

find(40, data) → T / F

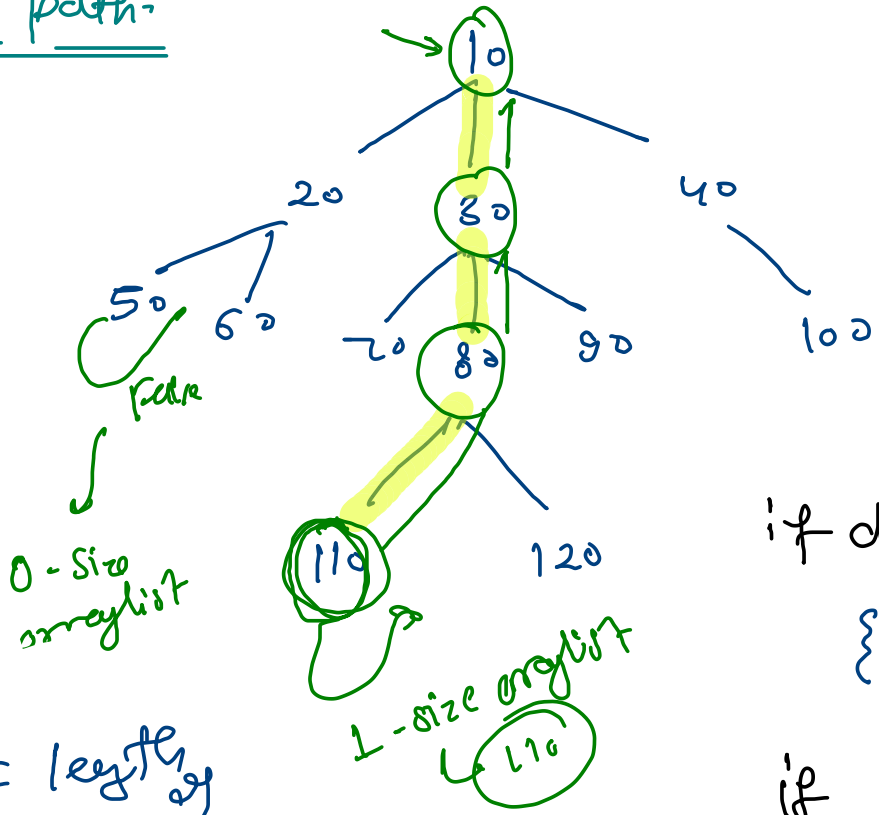
return

merging

1. Self check
2. Children check

Node to Root path.

find



110 → node to root path.

if data is present

{ 110, 80, 30, 10 }

if data is absent

{ } → Empty arraylist.

n = length of string

$T(n) = 2^n$

$O(n' \text{ string})$

$n' = 2^n$

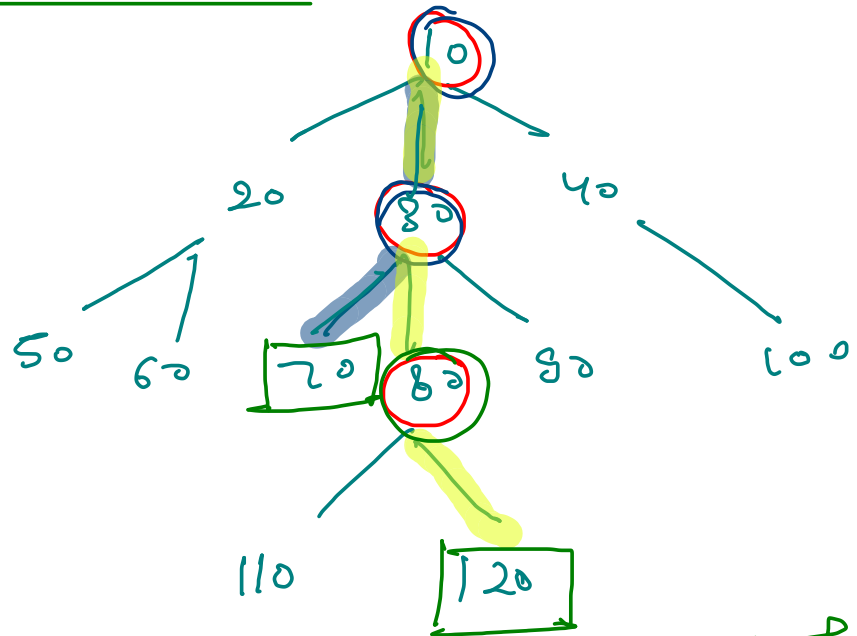
$T(n) = O(n')$

n = no. of nodes

$n \geq 3$

$2^3 \geq 8$

Lowest common Ancestor.



70 → Ancestor of 70

120 → Ancestor of 120

LCA → 30

→ LCA of (70, 120) = 30

node to root path →

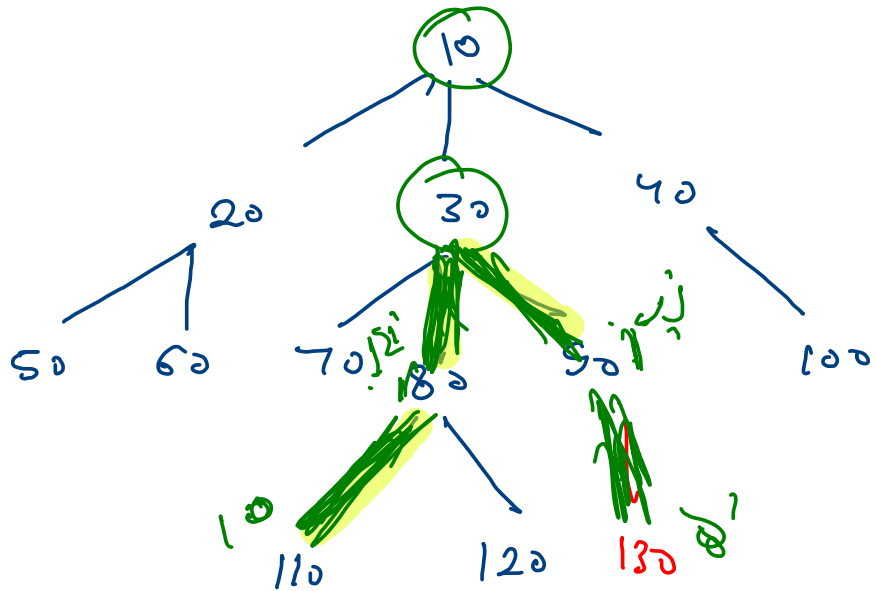
70 → 30 → 10

120 → 80 → 30 → 10

→ LCA of (80, 120) = 80

LCA

Distance b/w two Nodes in Generic Tree



distance b/w 110 & 90 = 3

if i != j

2

node → 110 80 90 10

0 1 2 3 i

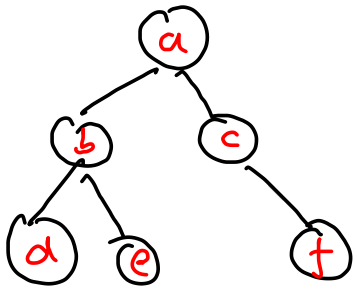
130 90 30 10

0 1 2 3 j

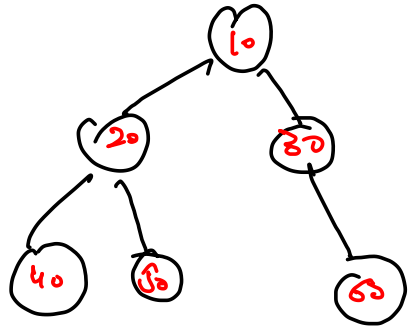
Structured Based problem

Similar

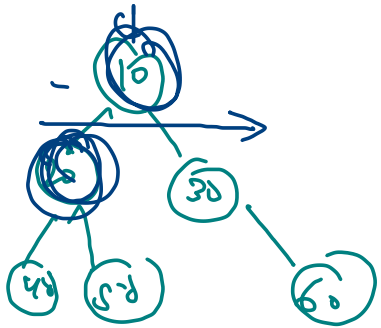
Tree1



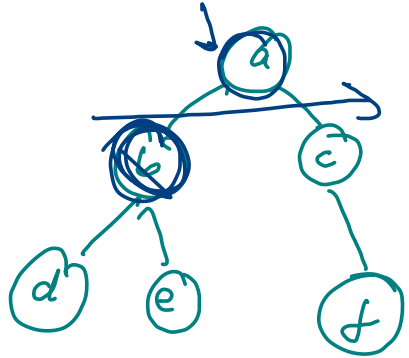
Tree2



root1



root2

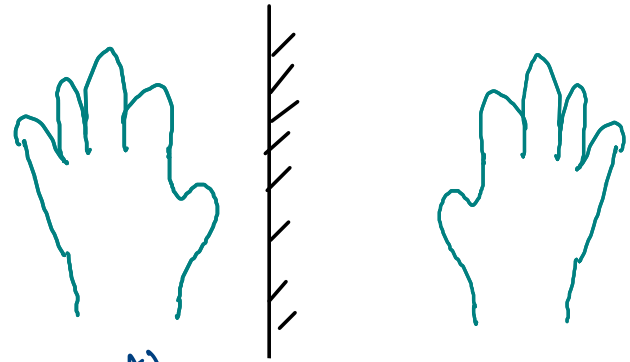


(1) no. of child is same or not

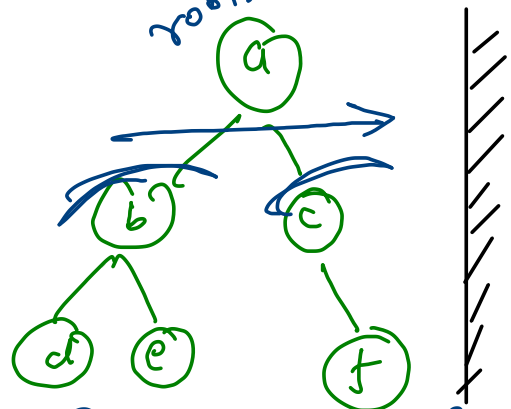
(2) (a, b), (30, 60) are similar, or not — false / ref false

at the end — true;

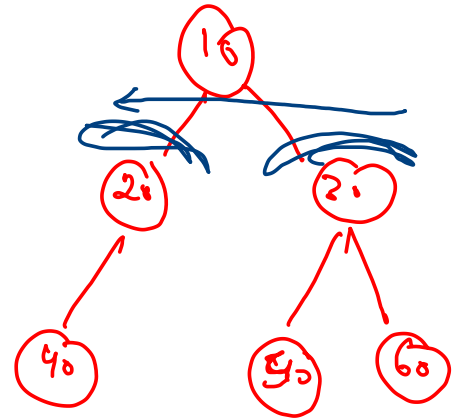
Mirror



root1



root2



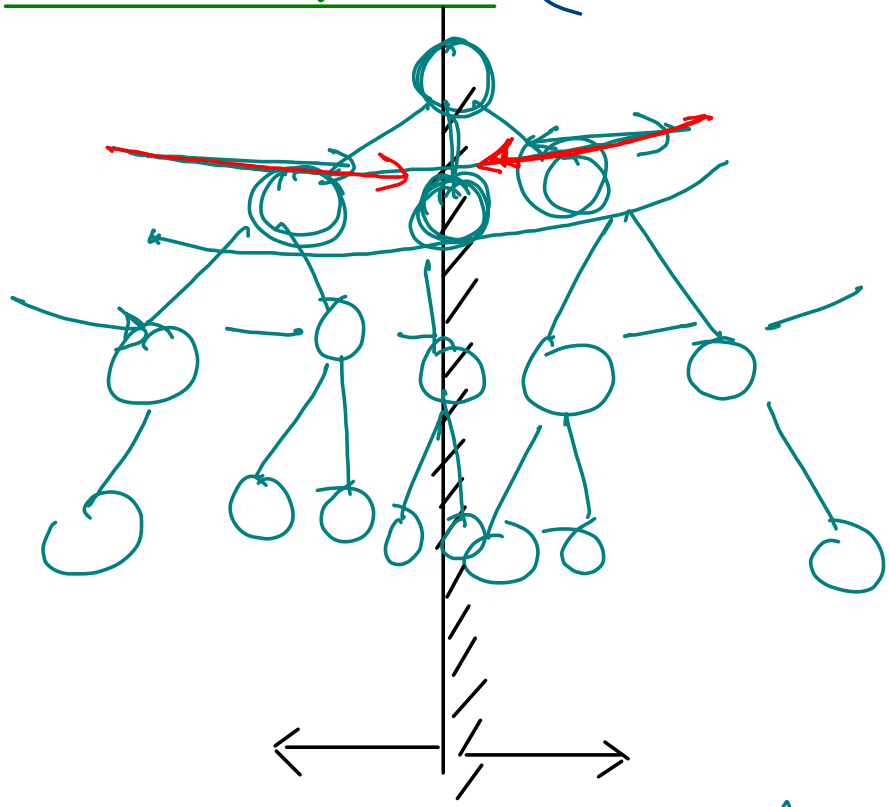
(1) root.child & root.child size same.

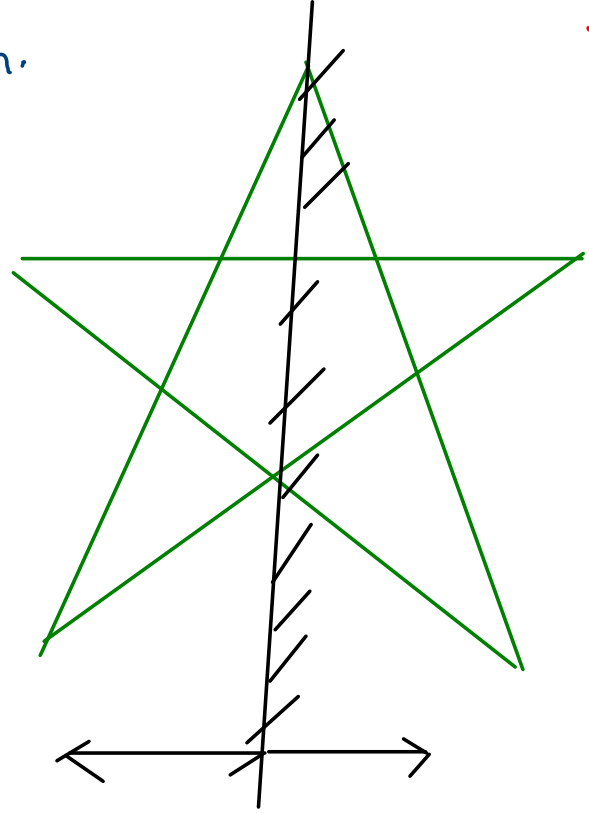
(2) root1 → (b, 30), (c, 60) — for mirror — if anything false — false

root2 ←

at the end — true.

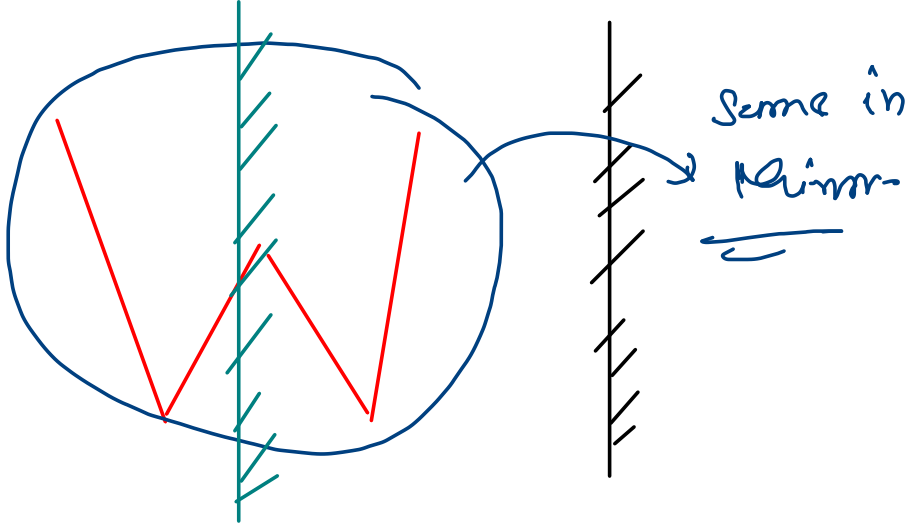
Is tree Symmetric (Structure based problem.



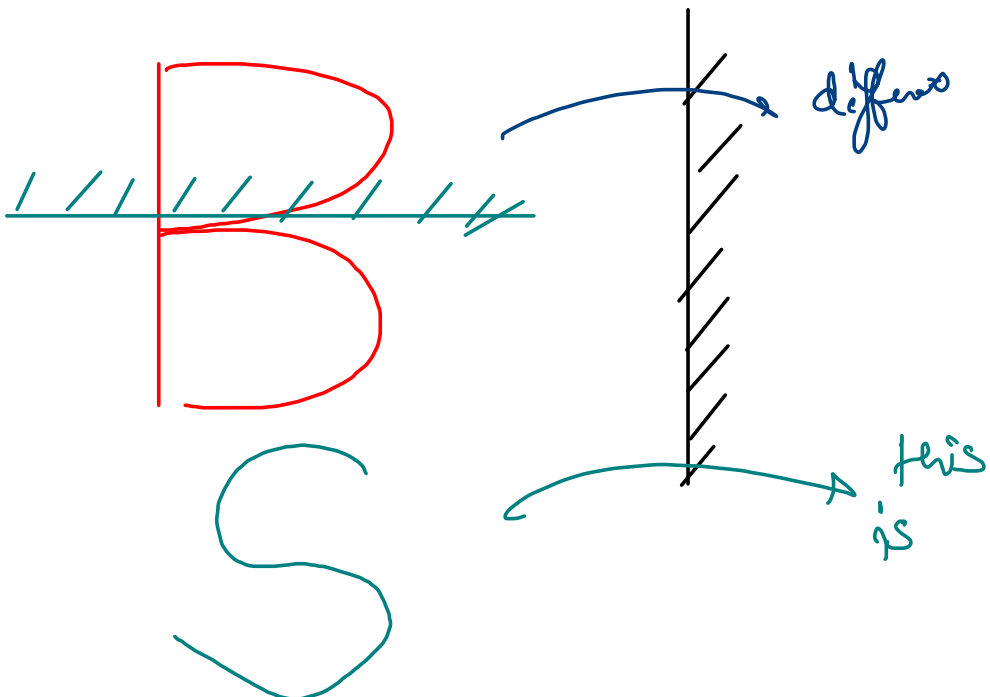


option ①

return are Mirror (root, root):



Symmetric



Not
Symmetric along
Vertical line of
Symmetry but
horizontally
Symmetric