Today's agenda

- ArrayList
- Size of the tree
- Sum of all nodes
- level order
- reverse level order

Array smartly used

ArrayList → Dynamic Array

int[] arr = new int[];

→ List<Integer> ls = new ArrayList<>();

name

or

→ ArrayList<Integer> ls = new ArrayList<>();

ls

0	1	2
10	20	30

ls.add(10);
ls.add(20);
ls.add(30);

ls.size() → 3

ls.get(1) → 20

→ O(n)

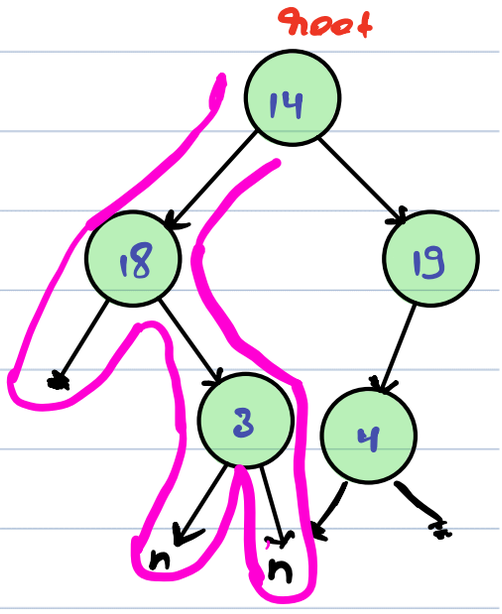
ls.remove(idx) → idx th index

→ last index

→ O(1)

Q) Size of a tree

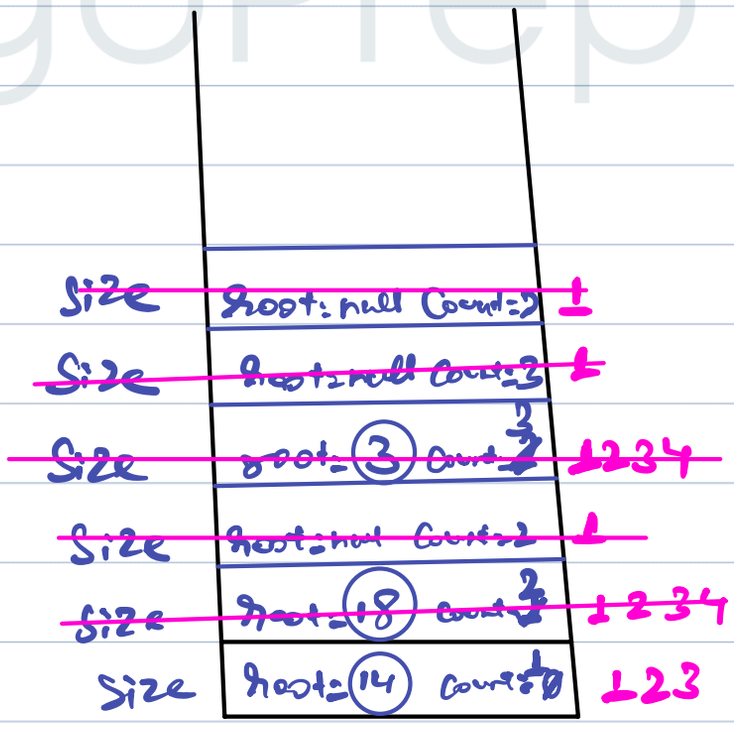
↳ Given a tree, calculate no. of nodes in it.



Idea 1

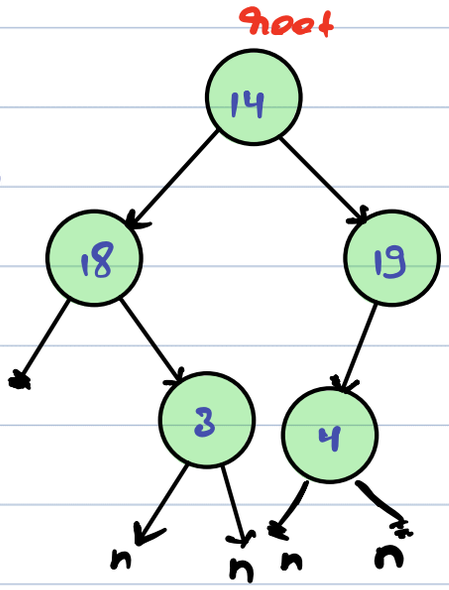
Global variable

```
int count=0;
void size (Node root){
1  if (root == null){return;}

2  count++;
3  size (root.left);
4  size (root.right);

}
```

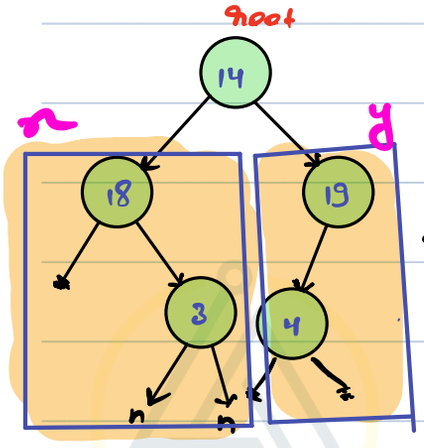


Idea 2

Faith: Given root node, find and return the size of tree.



Main logic:



→ x+y+1

```
int size (Node root){
    if(root==null){return 0;}
    int x= size(root.left);
    int y = size (root.right);
    return x+y+1;
}
```

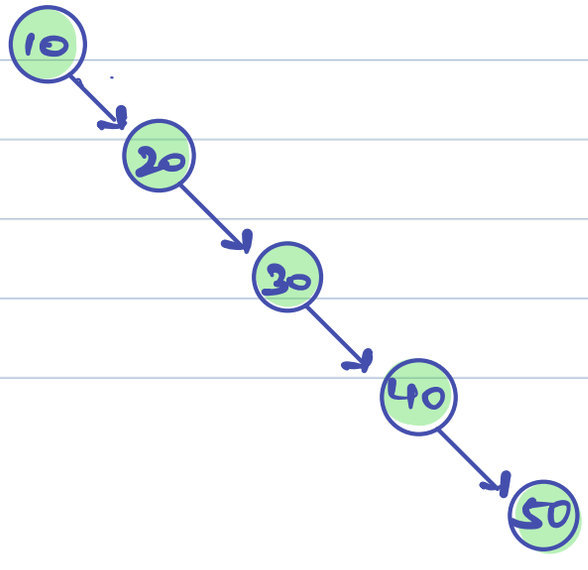
T.C: O(N)

S.C: O(height of tree) = O(N)

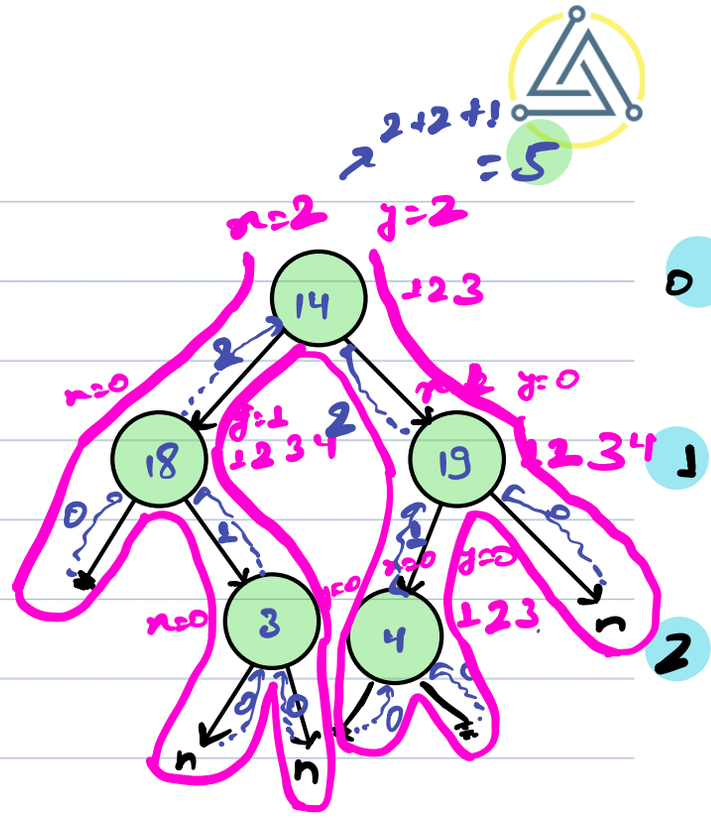
base case:

```
if(root==null){return 0;}
```

Skewed tree:



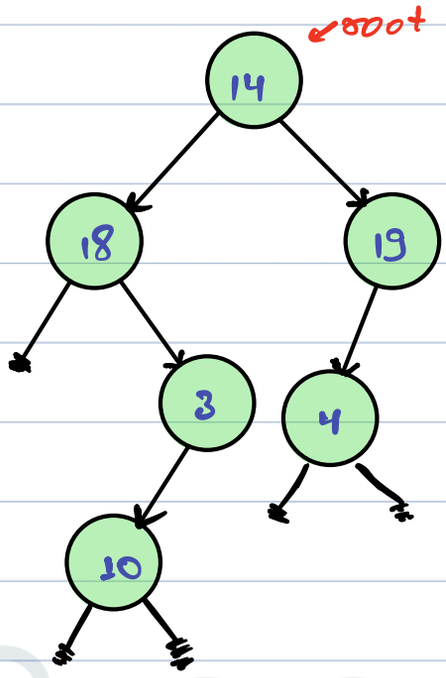
```
int size (Node root){
1  if(root==null){return 0;}

2  int x= size(root.left);
3  int y = size (root.right);

4    return x+y+1;

}
```



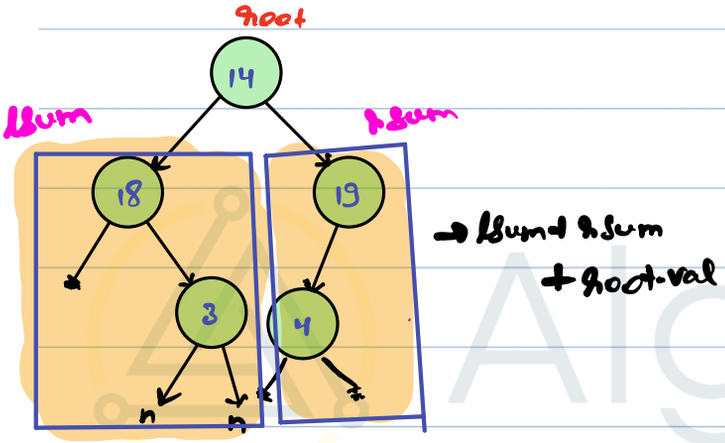
Q) Sum of a tree

↳ Given a tree, calculate sum of all nodes data in it.

Faith: Given root node, find and return the sum of tree.



Main logic:

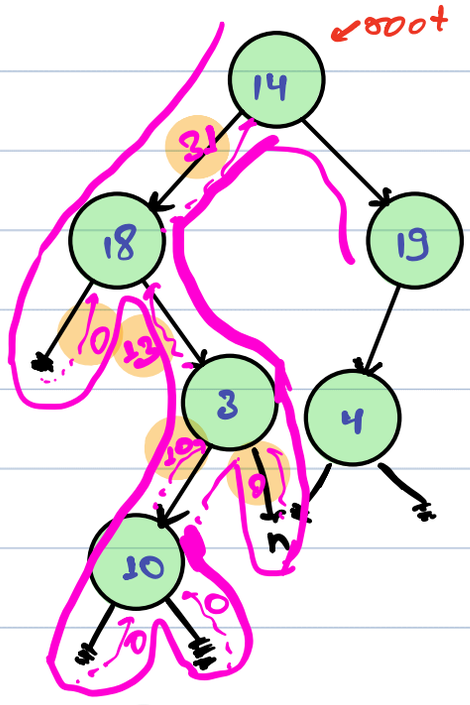


→ lSum + rSum + root.val

base case:

```
if(root==null){return 0;}
```

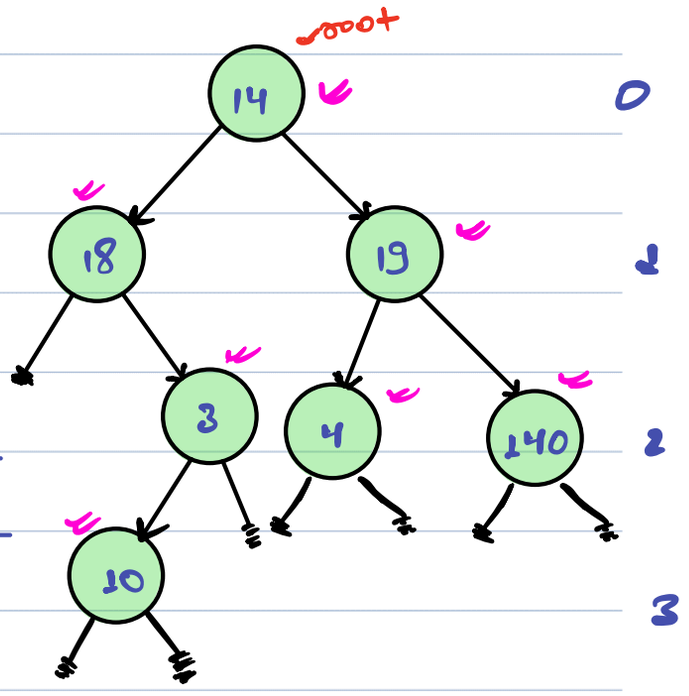
```
int Sum (Node root){
 if(root==null){return 0;}

  int lSum = Sum (root.left);
  int rSum = Sum (root.right);

  return lSum + rSum +
         root.val;
}
```




```
int Sum (Node root){
if(root==null){return 0;}

int lsum = Sum (root.left);
int rsum = Sum (root.right);

return lsum + rsum +
       root.val;
}
```

Break till 10:28 Am

Q) Level order of tree



14 18 19 3 4 140 10

Queue <Node> q;

~~14~~ ~~18~~ ~~19~~ ~~3~~ ~~4~~ ~~140~~ ~~10~~

rem: 10

14 18 19 3 4 140 10

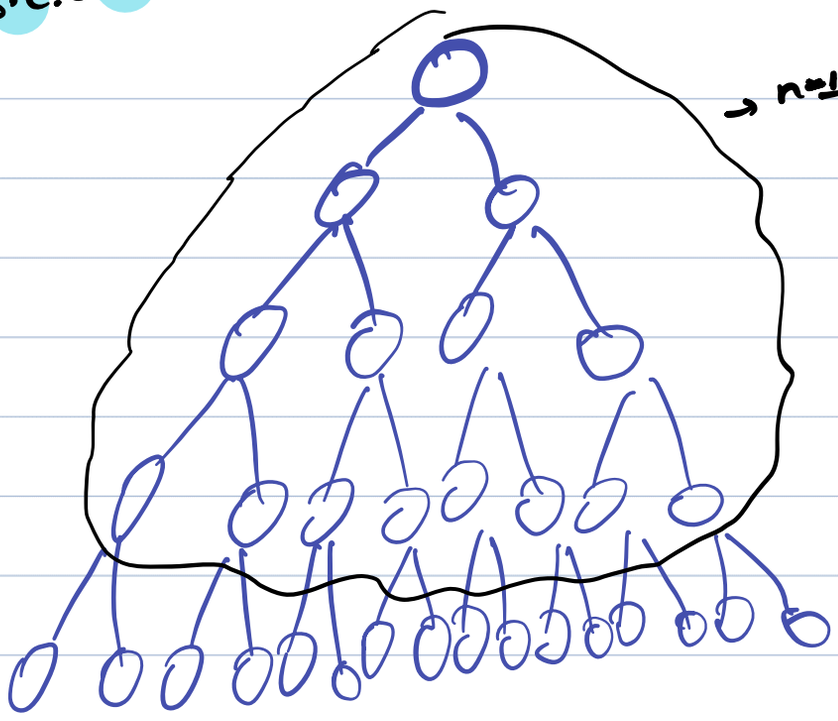
//Pseudo code

```
void levelorder (Node root){
    Queue <Node> q;
    q.add(root);
    while (q.size() > 0){
        Node rem = q.remove();
        System.out.println (rem.val);
        if (rem.left != null){ q.add(rem.left);}

        if (rem.right != null) {q.add(rem.right);}
    }
}
```

T.C: O(N)

S.C: O(N)

Why S.C: O(N)



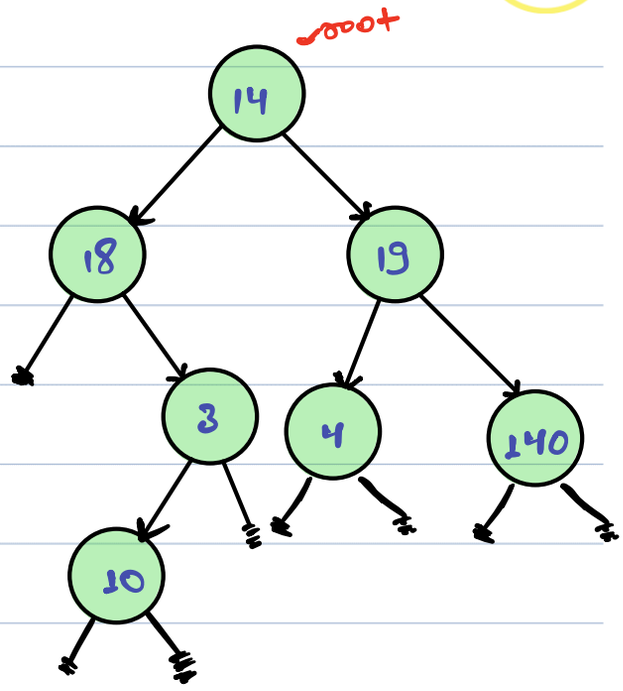
≈ 2n-1

↓

O(N)

→ O(N)

Q) Level order of tree 2



14

18 19

3 4 140

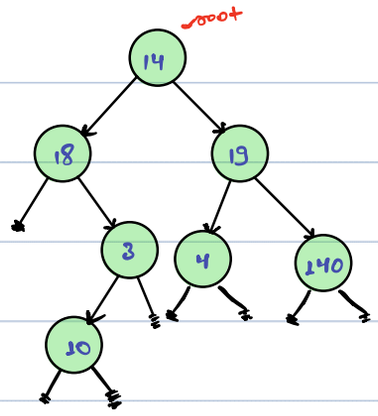
10

```
void levelOrder2 (Node root){
    Queue<Node> q;
    q.add(root);

    while (q.size() > 0) {
        int n = q.size();
        for(int i=1; i<=n; i++) {
            Node rem = q.remove();
            System.out.print(rem.val + " ");
            if (rem.left != null) { q.add(rem.left); }

            if (rem.right != null) { q.add(rem.right); }
        }
        System.out.println();
    }
}
```

T.C: O(N)

S.C: O(N)

```
void levelOrder2 (Node root){
    Queue <Node> q;
    q.add(root);

    while (q.size() > 0) {
        int n = q.size();
        for(int i=1; i<=n; i++) {
            Node rem = q.remove();
            System.out.print(rem.val);
            if (rem.left != null){ q.add(rem.left);}

            if (rem.right != null) {q.add(rem.right);}
        }
        System.out.println();
    }
}
```



0

q

rem: 18

n = 2 1

14

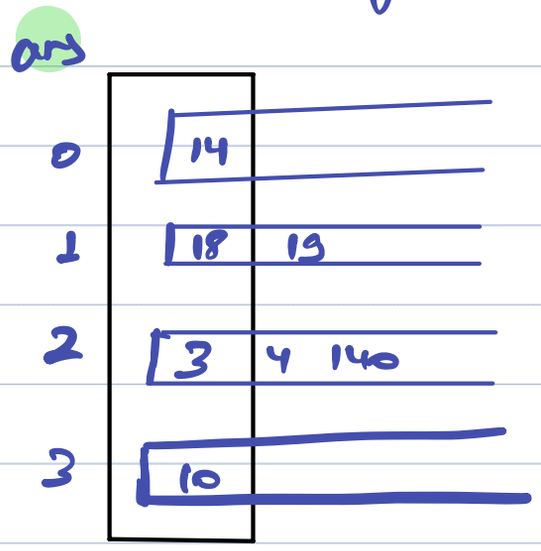
18 19

3 4 140

10

→

→ List<List<Integer>> ans = new ArrayList<>();



14
18 19
3 4 140
10

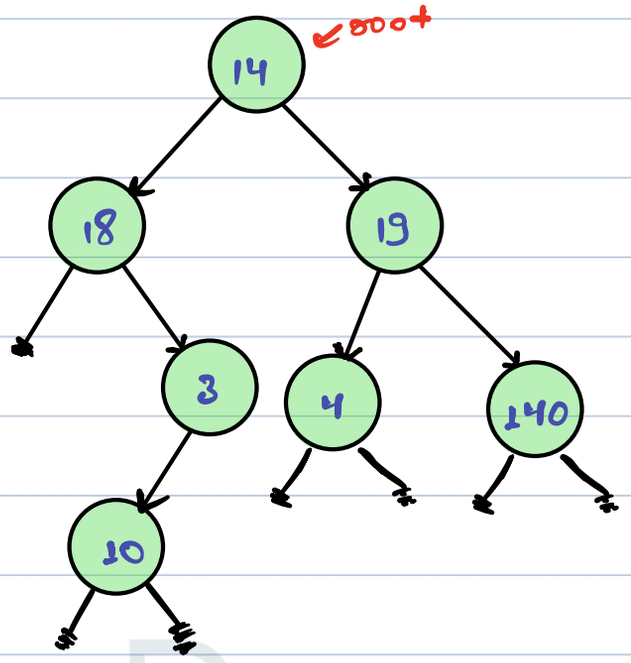
List<List<>>

```
levelorder2 (Node root){
    Queue<Node> q;
    q.add(root);
List<List<Integer>> ans = new ArrayList<>();
    while (q.size() > 0) {
        int n = q.size();
List<Integer>temp= new ArrayList<>();
        for(int i=1; i<=n; i++) {
            Node rem= q.remove();
            temp.add(rem.val);
            if (rem.left !=null){ q.add(rem.left);}

            if (rem.right !=null) {q.add(rem.right);}
        }
        ans.add(temp);
    }
    return ans;
}
```

Q) Reverse level order

↳ Print the level order from last level to first.

(10) (3 4 140) (18 19) (14)



↳ level order:

14 18 19 3 4 140 10

↓ reverse

(10) (140 4 3) (19 18) (14)

→ R-L level order

~~14~~ ~~18~~ ~~19~~ ~~140~~ ~~4~~ ~~3~~ ~~10~~

14 19 18 140 4 3 10

↓ reverse

10 3 4 140 18 19 14