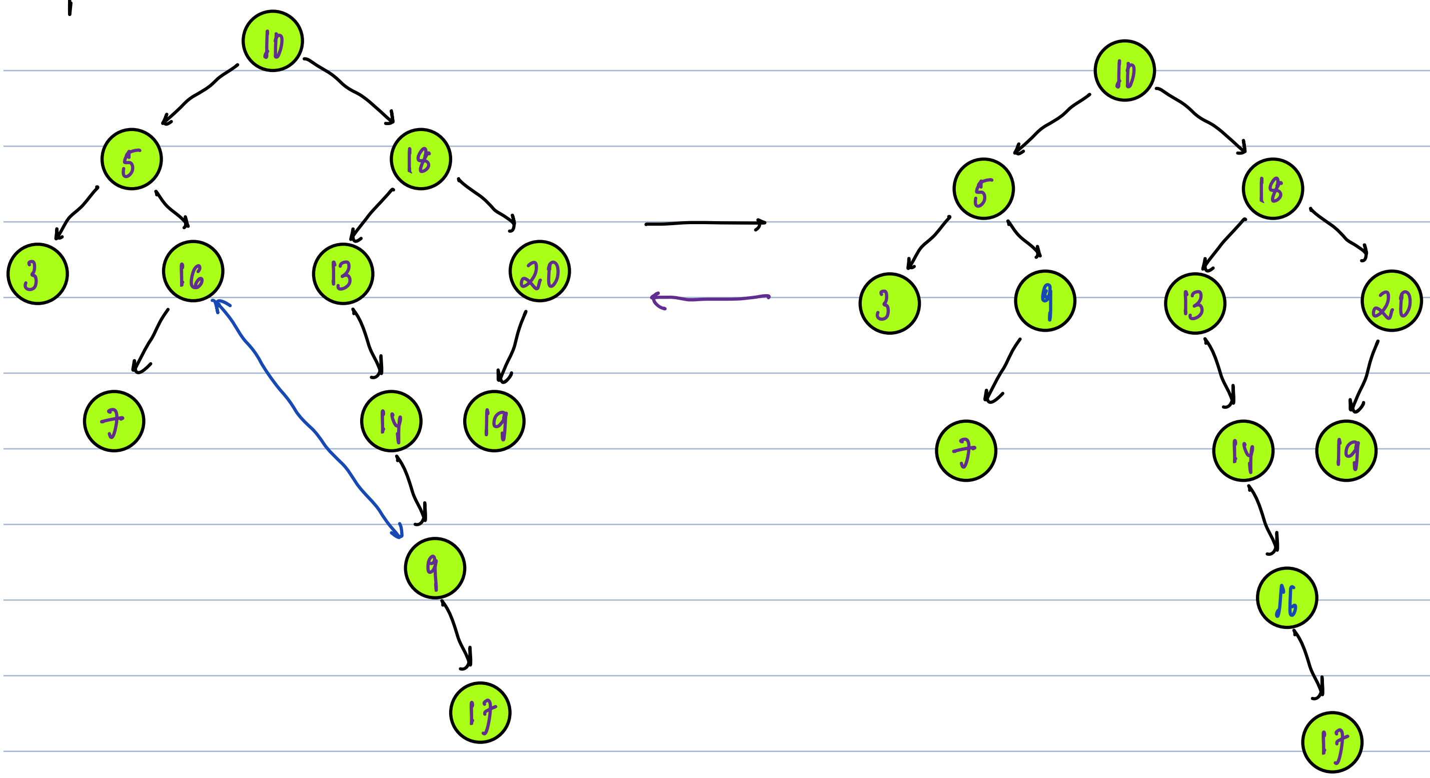
Todays Content

1. ReOrder BST
2. Largest BST in given BT

Recover BST:

Given a BT, which is formed by swapping 2 distinct nodes data in BST recover original BST, by swapping 2 nodes data back.

Input:



#Idea: Apply InOrder in Tree

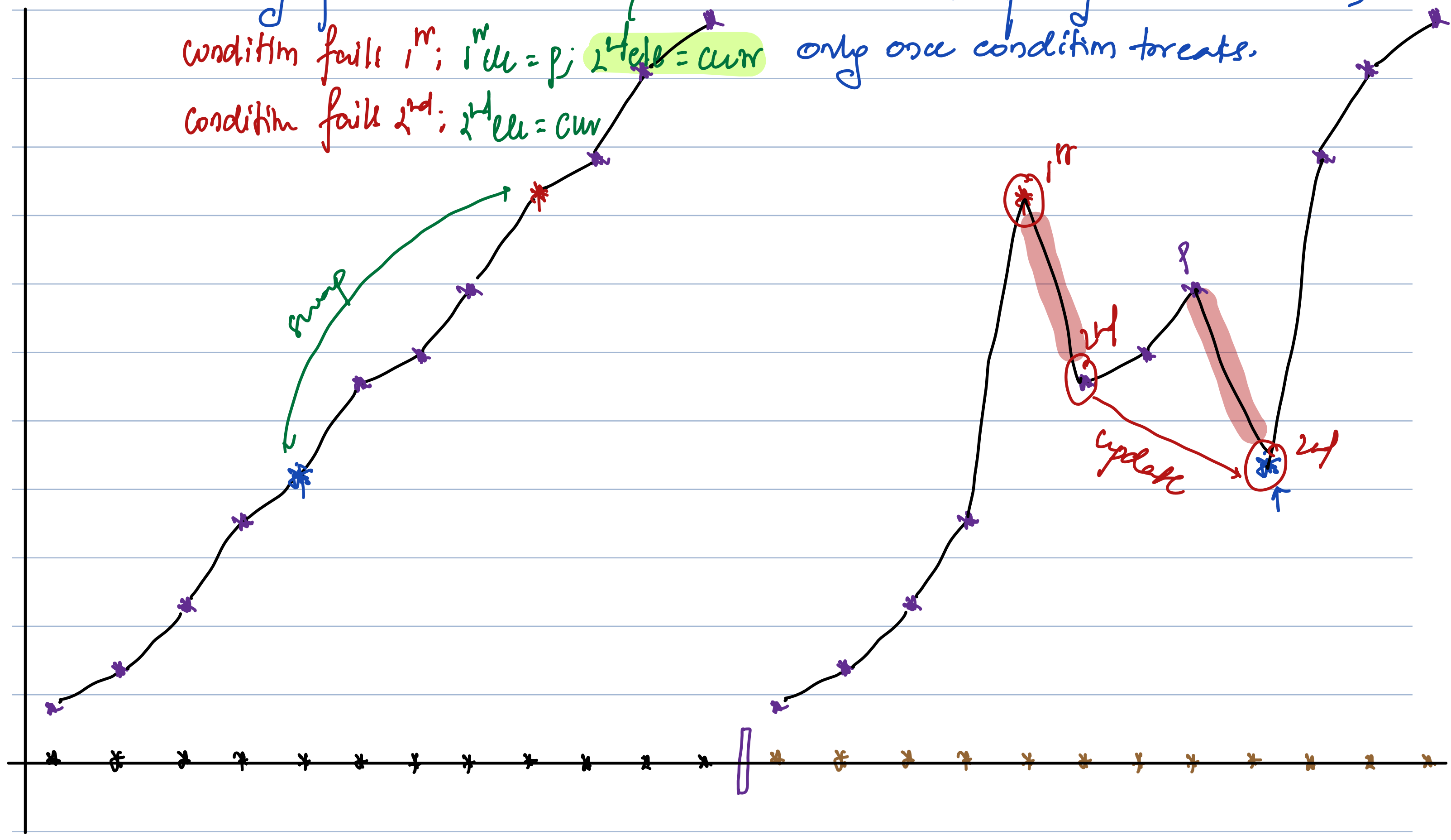
Ideally: prev & curr;

condition fails 1st; 1st ele = p; 2nd ele = curr

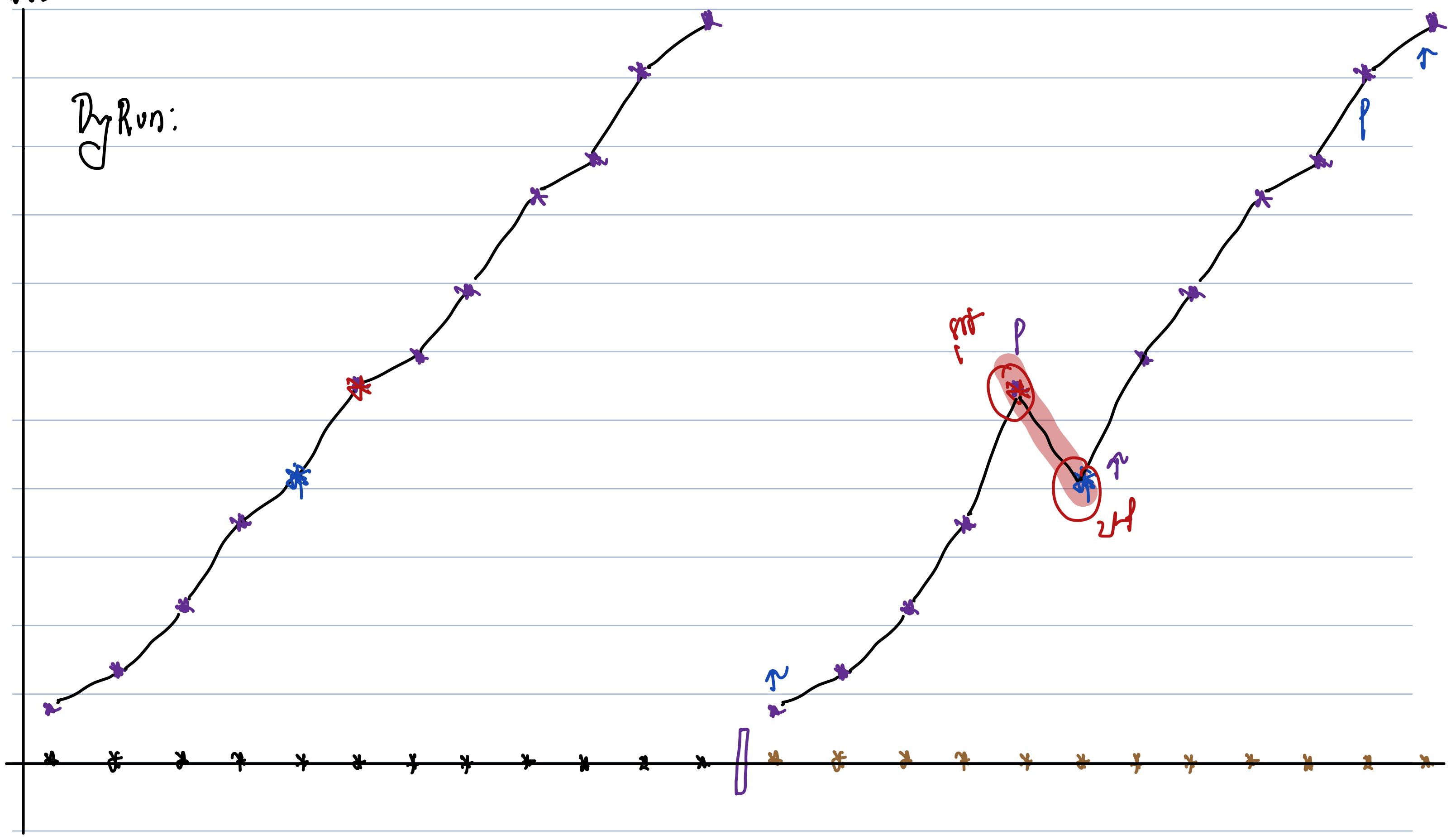
condition fails 2nd; 2nd ele = curr

To handle edge case:

When we swap adjacent elements, only once condition breaks.



#Issue:



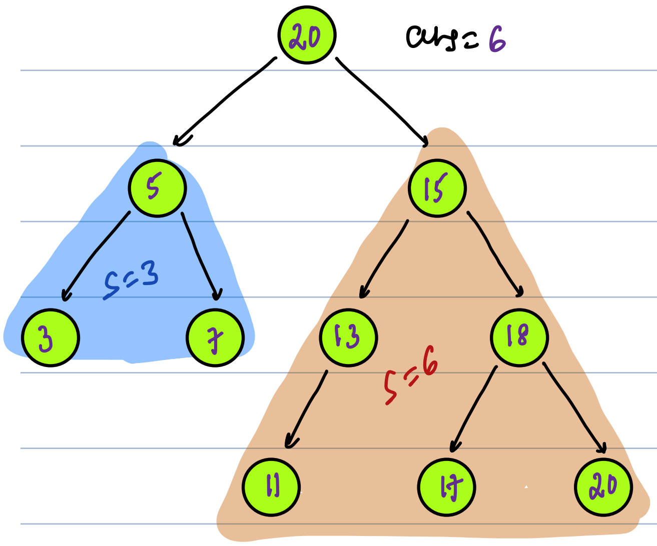
#Idea Apply InOrder in Tree } # Discussed in checking, if inorder is sorted.
Ideally: prev < curr;
condition fails 1st; 1st ele = p; 2nd ele = curr
condition fails 2nd; 2nd ele = curr

```
Node *p = nullptr, *f = nullptr, *s = nullptr;
void InOrder(Node *root){  TC: O(N)
        if(root == nullptr){return;}
        InOrder(root->left);

        if(p != nullptr && p->data > root->data){ # p < curr->data && curr = root.
            if(f == nullptr){ # failing 1st time
                f = p; s = root;
            }
            else{
                s = root;
            }
        }
        p = root; # updating prev

}   InOrder(root->right);

Node* restore BST(Node *root){
    *p = nullptr, *f = nullptr, *s = nullptr;
    InOrder(root);
    Swap f->data && s->data;
    return root;
}
```

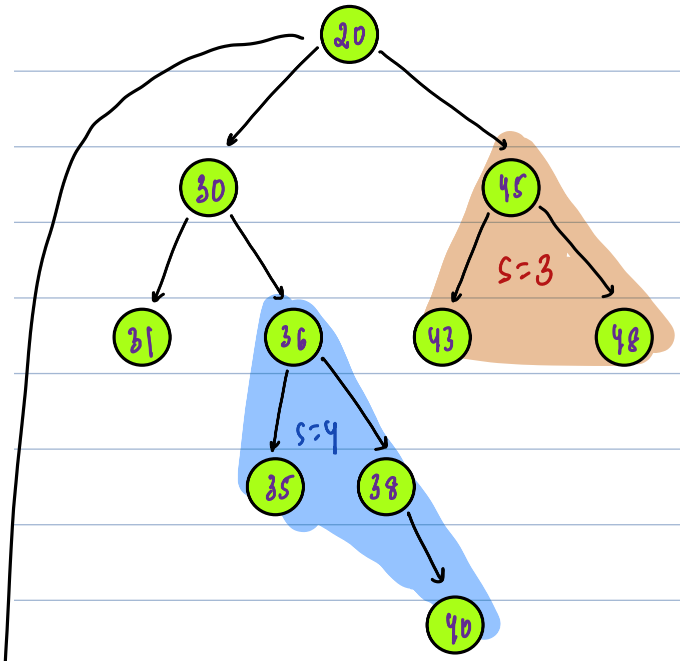
2Q Return size of largest subtree in BT which is BST

Ex1:

ans = 6



Ex2: ans = 4



Idea1; *Wrong approach

1. Apply Inorder on BT & store in arr[].

2. Iterate on arr[]:

Calculate longest part of arr[], which is increasing.

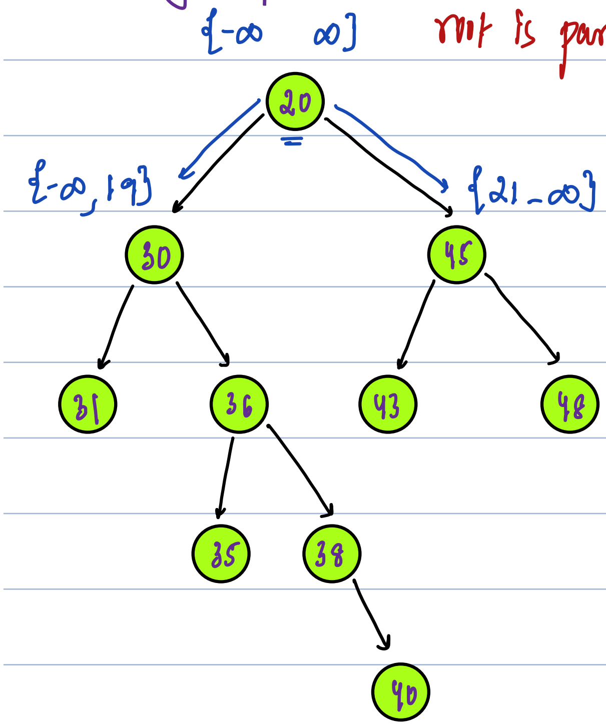
In: 31 30 35 36 38 40 20 43 45 48

It need not be a subtree.

```
Idea2: For every node;
    if( isBST(node) ){ # Subtree is BST
        ans = max(ans, size(node));
    }
}
```

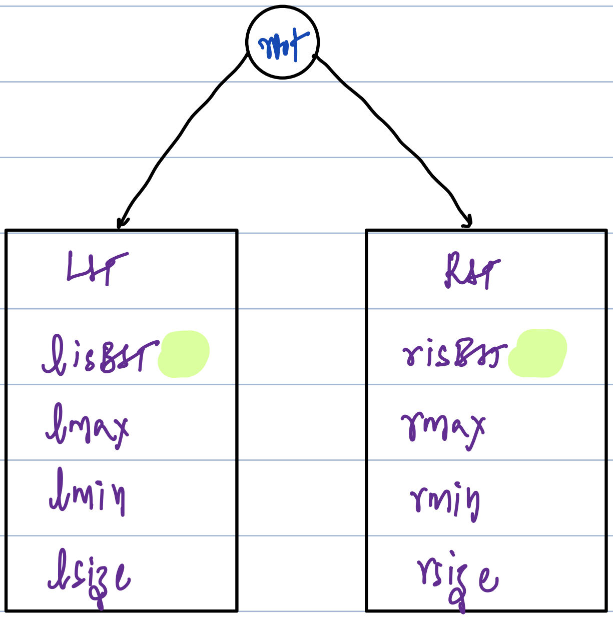
TC: $O(N*(N+N)) = O(N^2)$

Idea3: Try Top down: Here, we cannot do top-down because we are not sure, if root is part of BST, because of that any info we pass from root can result in wrong values.



Idea4: Try Bottom up.

Case1:



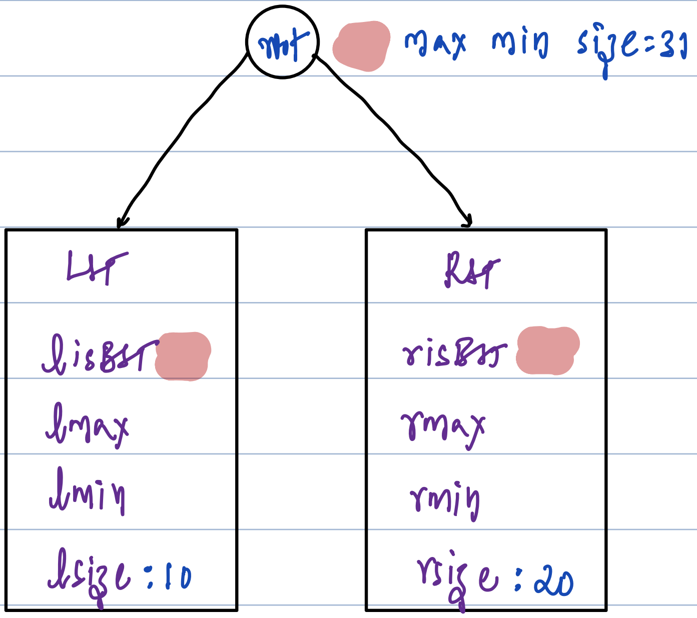
isBST : Is subtree BST or not

max : max of subtree

min : min of subtree

size: size of subtree

Case2:



#Issue: Finally root node will return:

isBST max min size:

↳ size of BST, It's not size of largest BST

#Sol: Take a global variable:

If a subtree is BST, compare it's size with ans.

```
int sans=0;
vector<int> largestBST(Node *root){ TC: O(N)
    if(root==null){
        vector<int> ans(4,0);
        ans[0]=1; ans[1]=+∞; #Min ans[2]=-∞; #Max ans[3]=0; #Size
    } return ans;
                                                      0    1    2    3
    vector<int> l = isBST(root->left); # l: isB  min  max  size
    vector<int> r = isBST(root->right); # r: isB  min  max  size
    vector<int> ans(4,0);
    if((l[0]==1) && (r[0]==1) && (root->data > l[2]) && (root->data < r[1])){
    } ans[0]=1;
    ans[1]= min(l[1], r[1], root->data);
    ans[2]= max(l[2], r[2], root->data);
    ans[3]= l[3]+r[3]+1;
    if(ans[0]==1){
    } sans= max(sans, ans[3]);
    return ans;
}


int solve(Node *root){
    sans=0;
    largest(root);
    return sans;
}
```

```
int solve(Node *root){
|
}
```

#Slight modification:

3Q Return root node of largest subtree which is BST