BST Questions

lecture 1

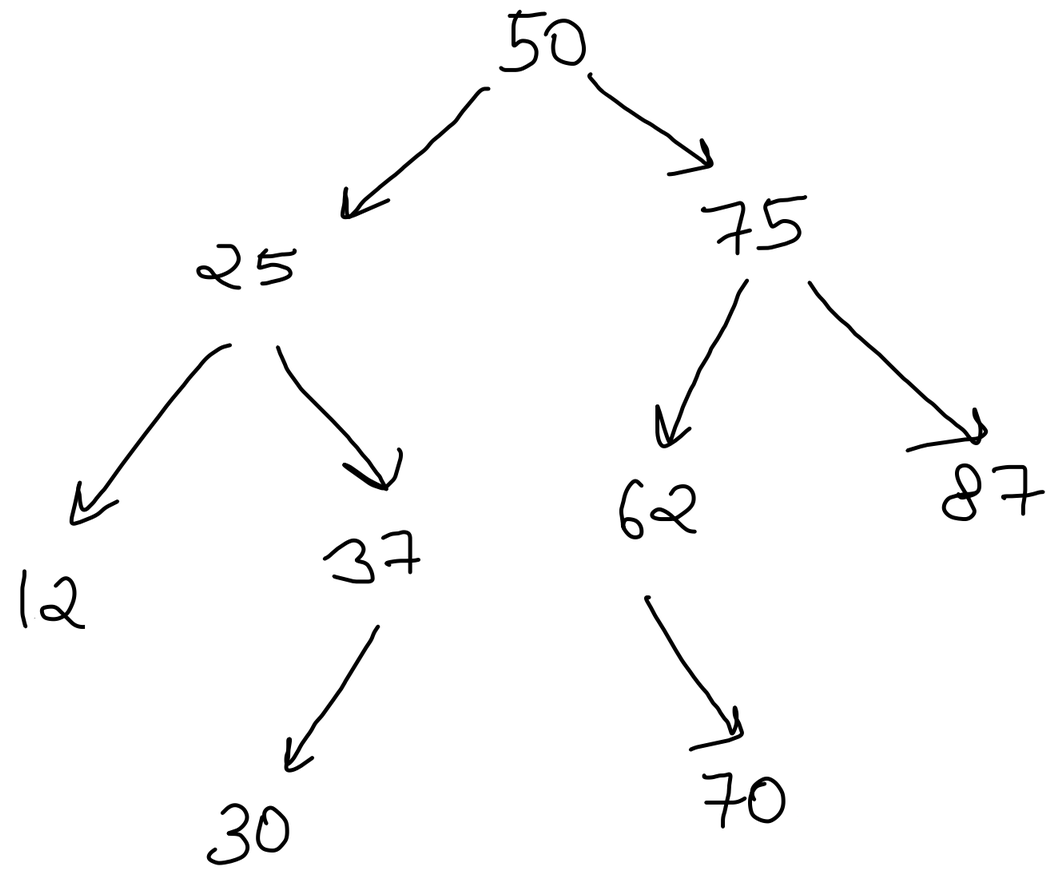
- Is A Binary Search Tree
- Is Balanced Tree
- Largest Bst Subtree
- Size, Sum, Max, Min, Find In Bst

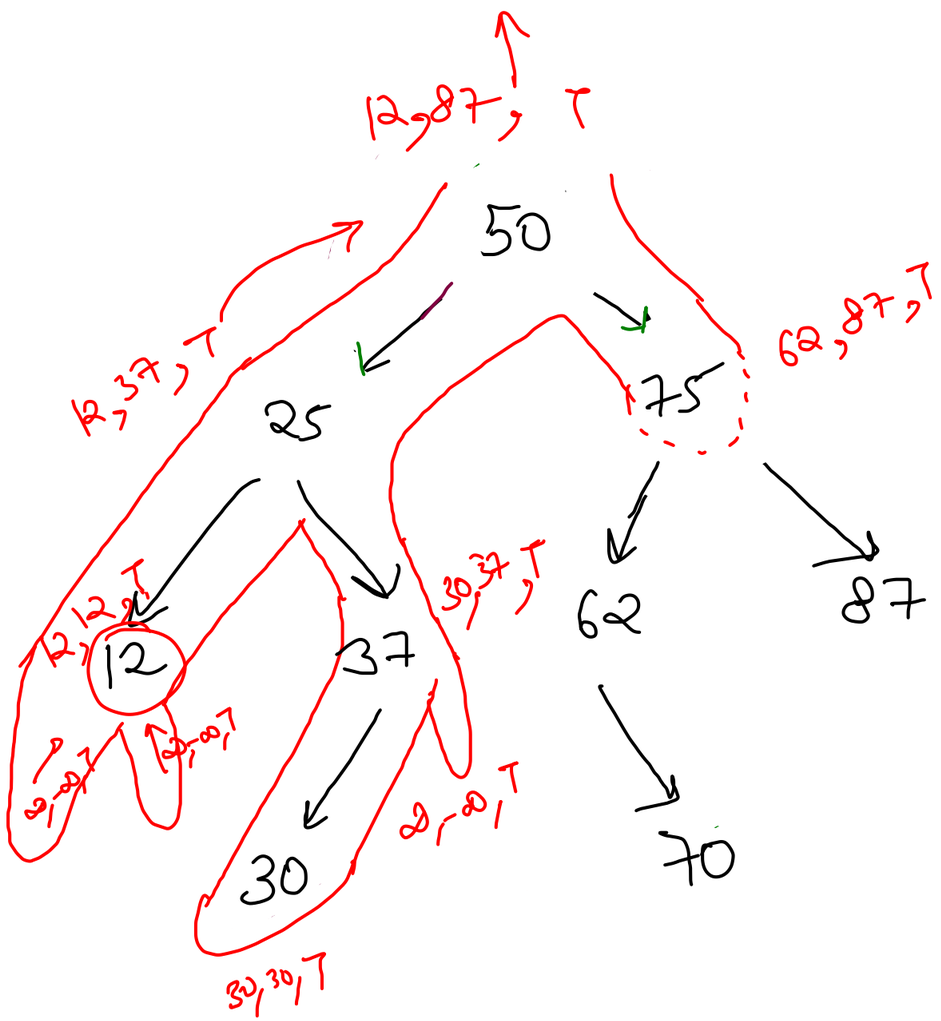
lecture 2

- Binary Search Tree - Introduction
- Binary Search Tree - Constructor
- </> Add Node To Bst

lecture 3

- </> Remove Node From Bst
- </> Replace With Sum Of Larger
- </> Lca Of Bst
- </> Print In Range
- </> Target Sum Pair In Bst
- Target Sum Pair - Bst - Alternate Approaches
- Iterable And Iterator





Binary Search Tree # unique nodes

① left nodes < root node < right nodes

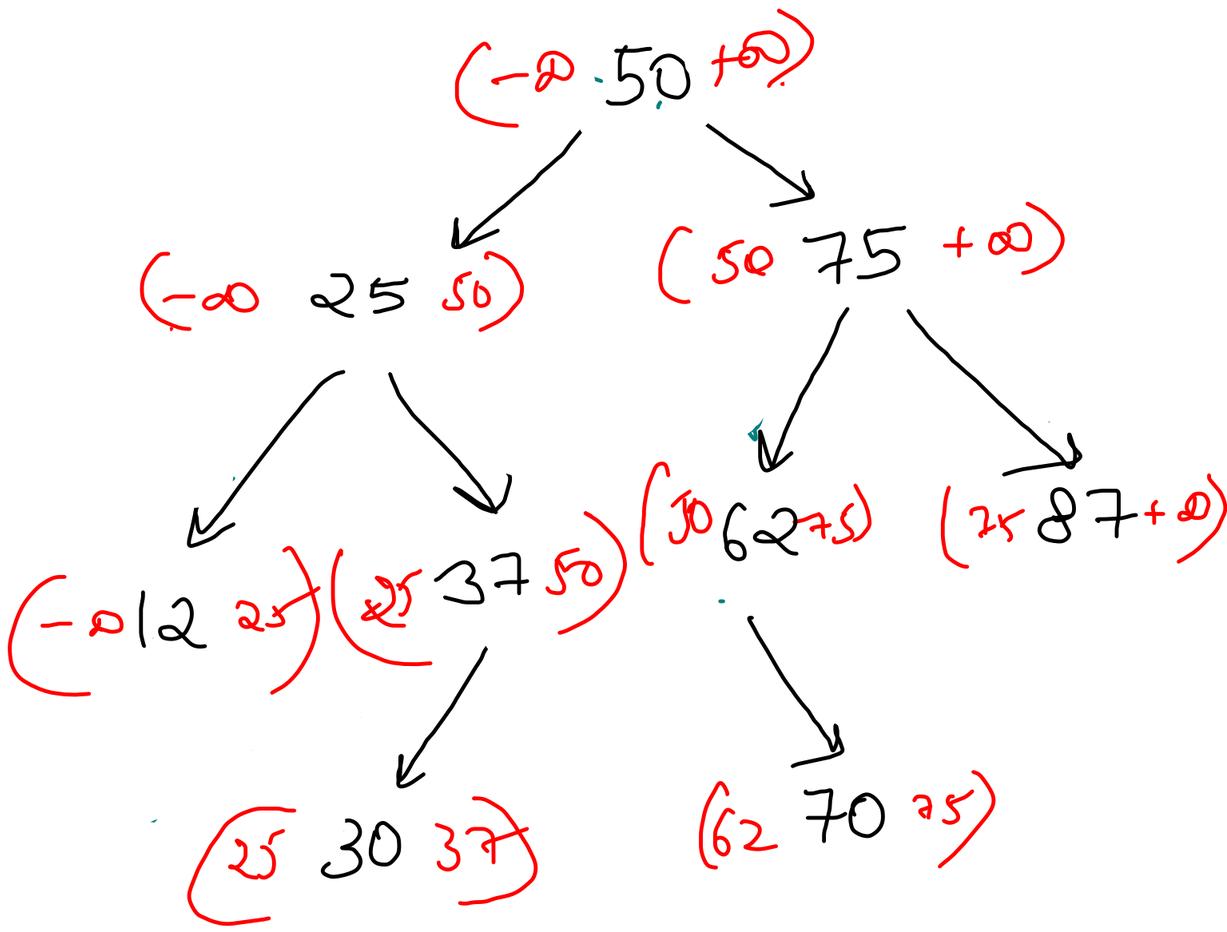
isBinaryTree ABST?

1st approach

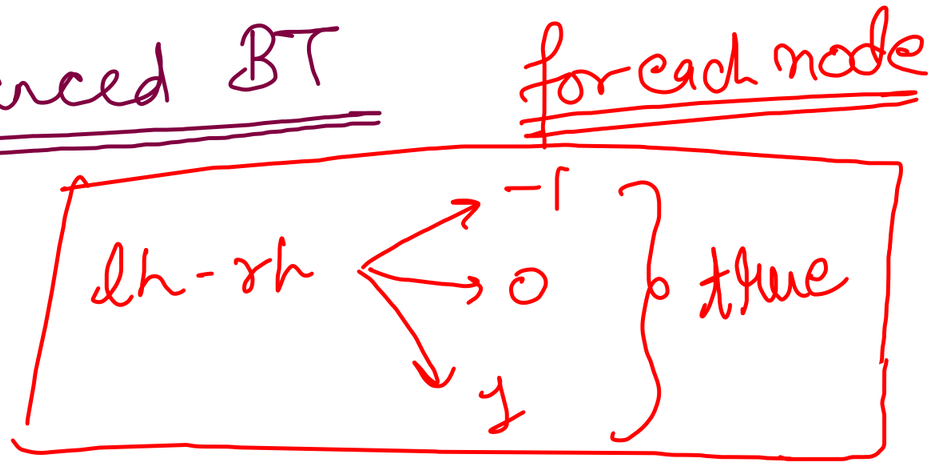
```
public static isBSTPair isBST(Node root){
    if(root == null) return new isBSTPair();

    isBSTPair left = isBST(root.left);
    isBSTPair right = isBST(root.right);

    isBSTPair curr = new isBSTPair();
    if(left.max < root.data && root.data < right.min
       && left.isBST && right.isBST){
        curr.isBST = true;
    } else {
        curr.isBST = false;
    }

    curr.min = Math.min(root.data, Math.min(left.min, right.min));
    curr.max = Math.max(root.data, Math.max(left.max, right.max));
    return curr;
}
```

2nd approach

```java
public static boolean isBST(Node root, int min, int max){
    if(root == null) return true;

    if(root.data > min && root.data < max){
        boolean lres = isBST(root.left, min, root.data);
        boolean rres = isBST(root.right, root.data, max);
        return lres && rres;
    }

    return false;
}
```

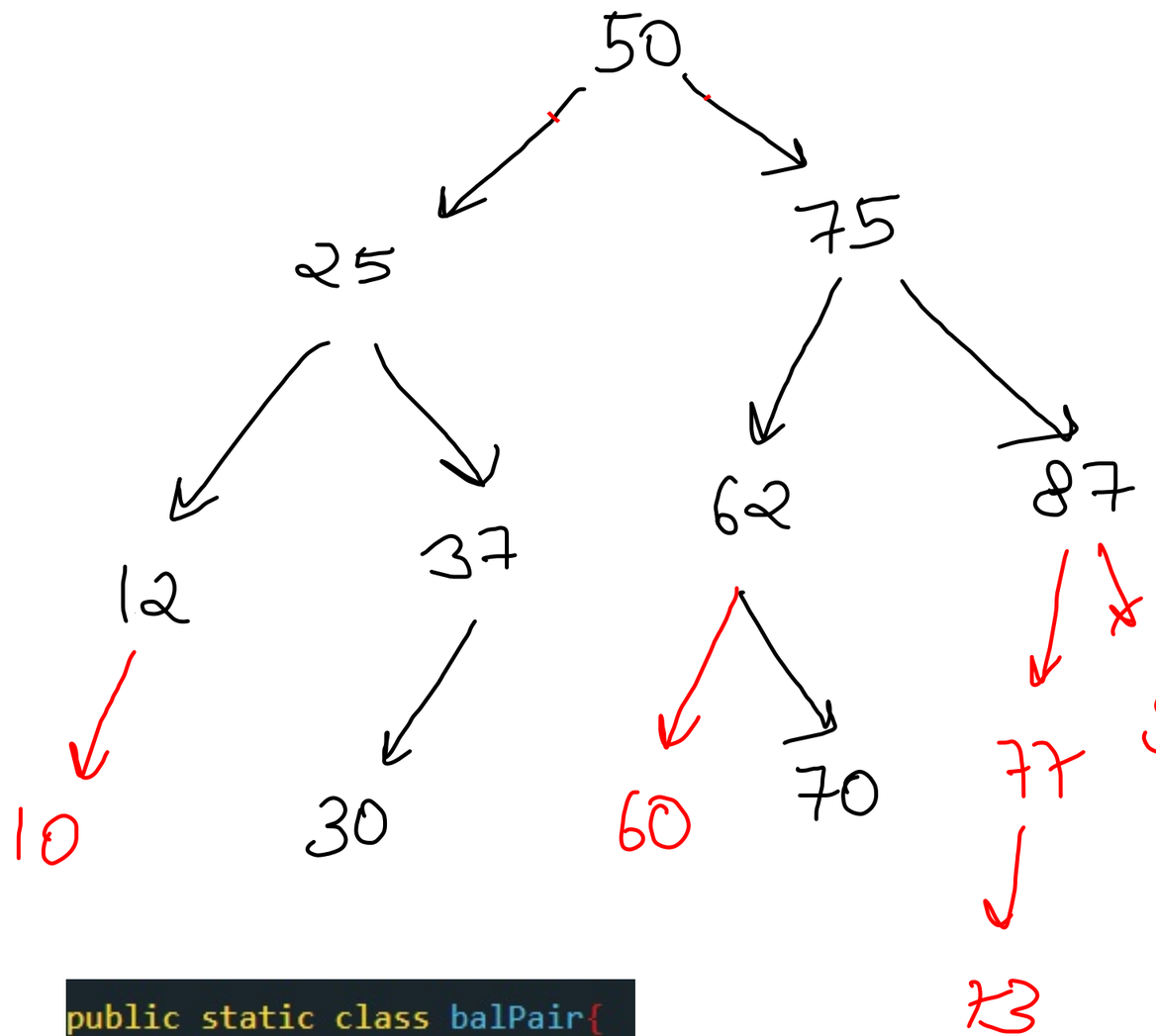


Balanced BT




```java
public static balPair isBalanced(Node root){
    if(root == null) return new balPair();

    balPair left = isBalanced(root.left);
    balPair right = isBalanced(root.right);

    balPair curr = new balPair();

    if(left.isBal && right.isBal && Math.abs(left.height - right.height) <= 1){
        curr.isBal = true;
    } else {
        curr.isBal = false;
    }

    curr.height = Math.max(left.height, right.height) + 1;
    return curr;
}
```

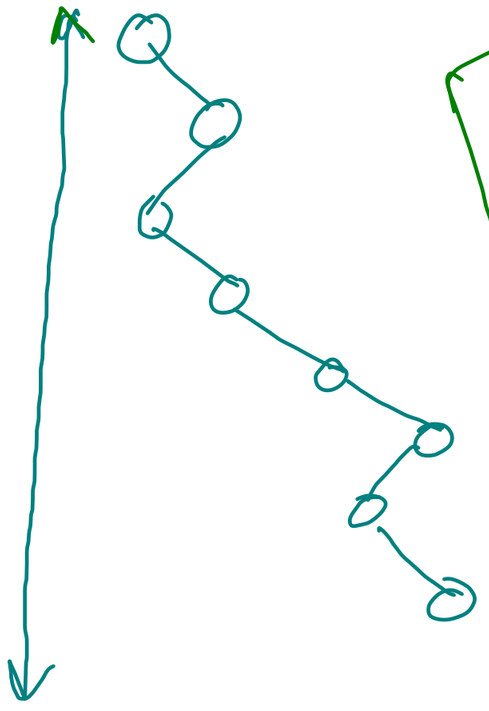
TC ⇒ O(N) SC ⇒ O(h)




```java
public static class balPair{
  int height = 0;
  boolean isBal = true;
}
```

nodes to height formula

N nodes



Skewed tree
$h_{max} = N$

Recursion (Space)

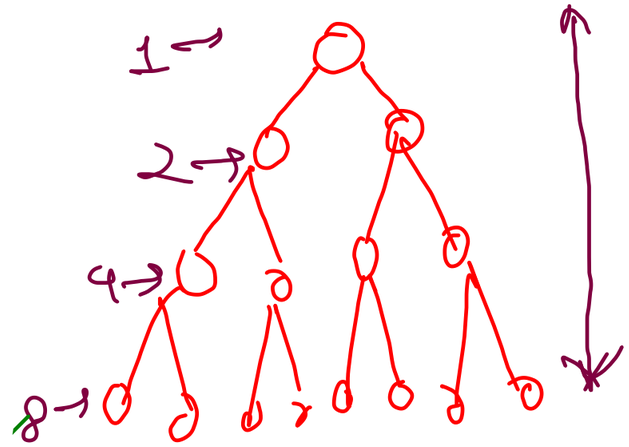
O(h)

O(height or depth)

Skewed
$O(N)$

balanced
$O(\log_2 N)$

$$\begin{cases} \Omega(\log_2 N) \rightarrow \text{Best case} \\ \Theta(\log_2 N) \rightarrow \text{Avg case} \\ O(N) \rightarrow \text{worst case} \end{cases}$$



balanced tree

$2^0 + 2^1 + 2^2 + 2^3 + \dots 2^{h-1} = N$

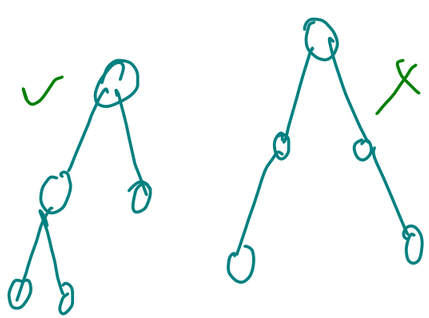
$2^h = N$

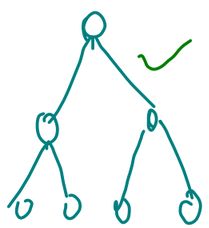
$h_{min} = \log_2 N$

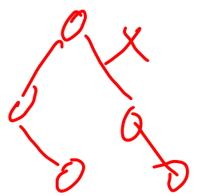
Strict BT

- 0 children
- 2 children

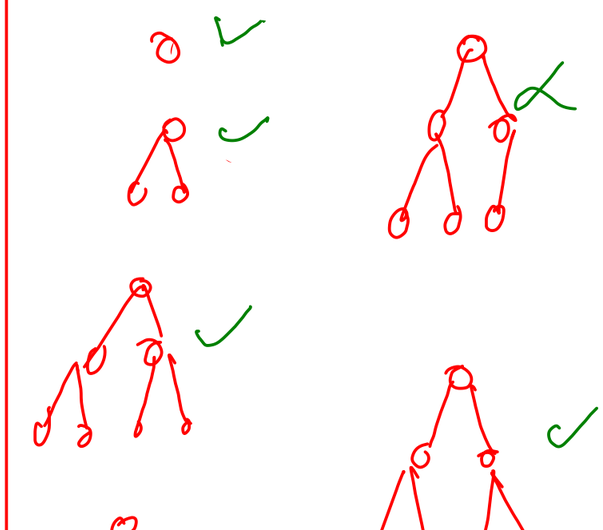
1 child ~~1 child~~

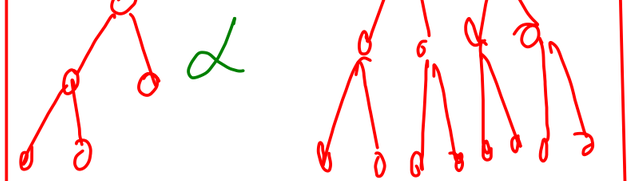






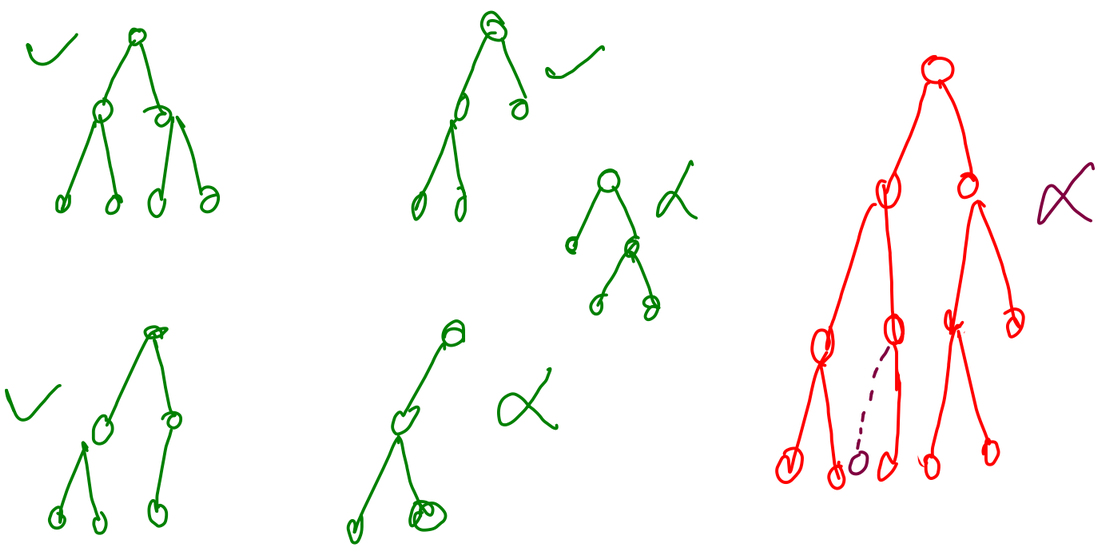
full BT

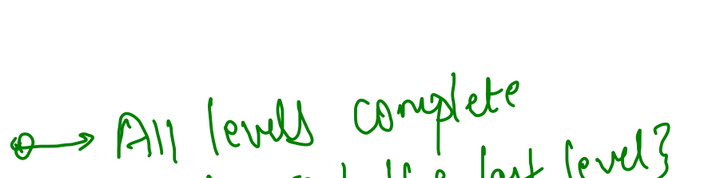




All levels are complete {max^m nodes}

Complete BT





→ All levels complete {except the last level}

→ If last level is not complete, all nodes should be as left as possible

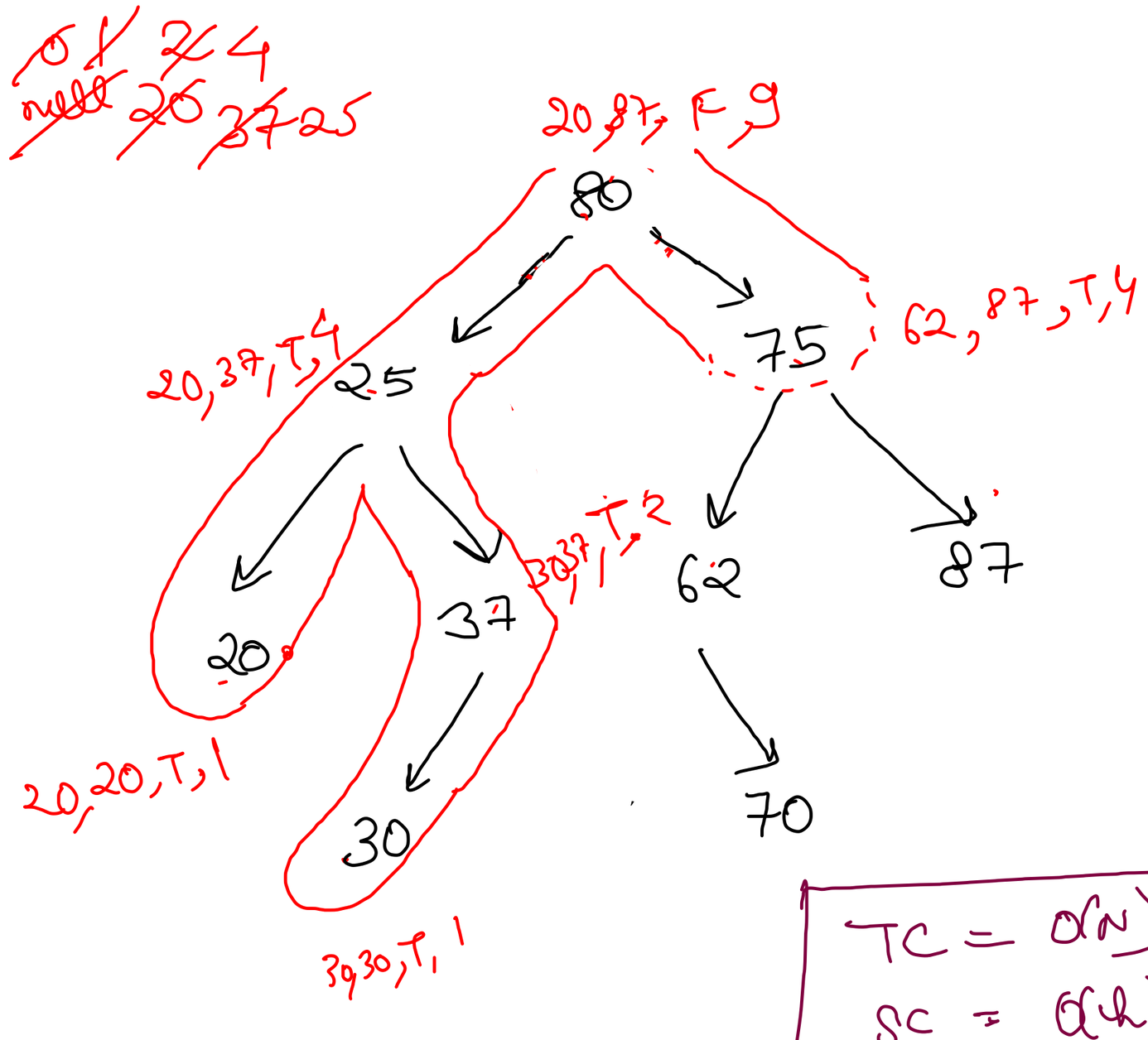
All full BT are complete BT

but complete BT may or may not be full.

{Break till 11:00 AM}

Largest BST Subtree

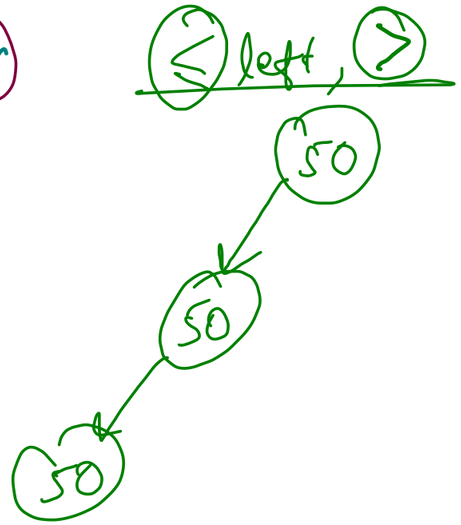
```java
public static class isBSTPair{
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    boolean isBST = true;
    int count = 0;
}

static int maxCount = 0;
static Node largestBST = null;

public static isBSTPair largestBST(Node root){
    if(root == null) return new isBSTPair();

    isBSTPair left = largestBST(root.left);
    isBSTPair right = largestBST(root.right);

    isBSTPair curr = new isBSTPair();
    curr.count = left.count + right.count + 1;

    if(left.max < root.data && root.data < right.min
        && left.isBST && right.isBST){
        curr.isBST = true;

        if(curr.count > maxCount){
          maxCount = curr.count;
          largestBST = root;
        }

    } else {
        curr.isBST = false;
    }

    curr.min = Math.min(root.data, Math.min(left.min, right.min));
    curr.max = Math.max(root.data, Math.max(left.max, right.max));

    return curr;
}
```



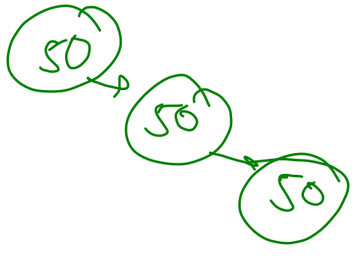
TC = O(N)
SC = O(h)

BST nodes duplicate

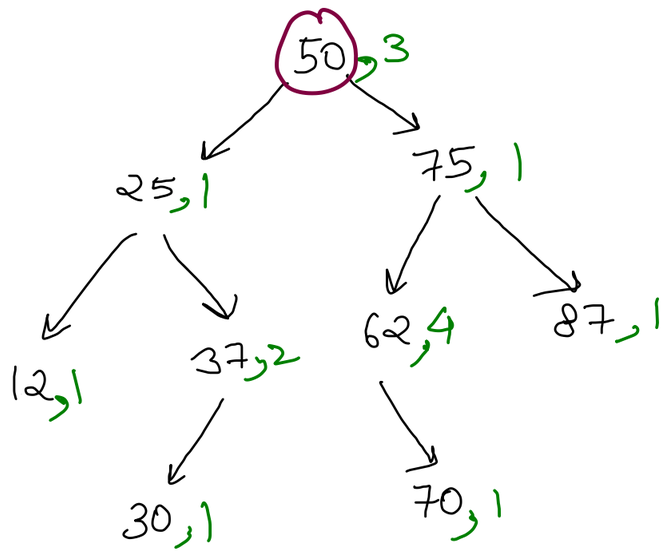
① ≤ left, > right



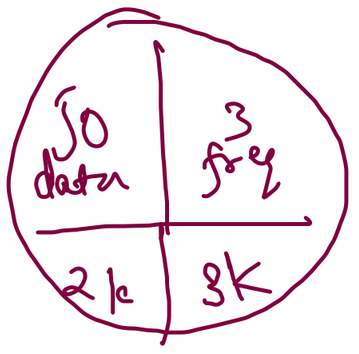
② < left, ≥ right



③ frequency

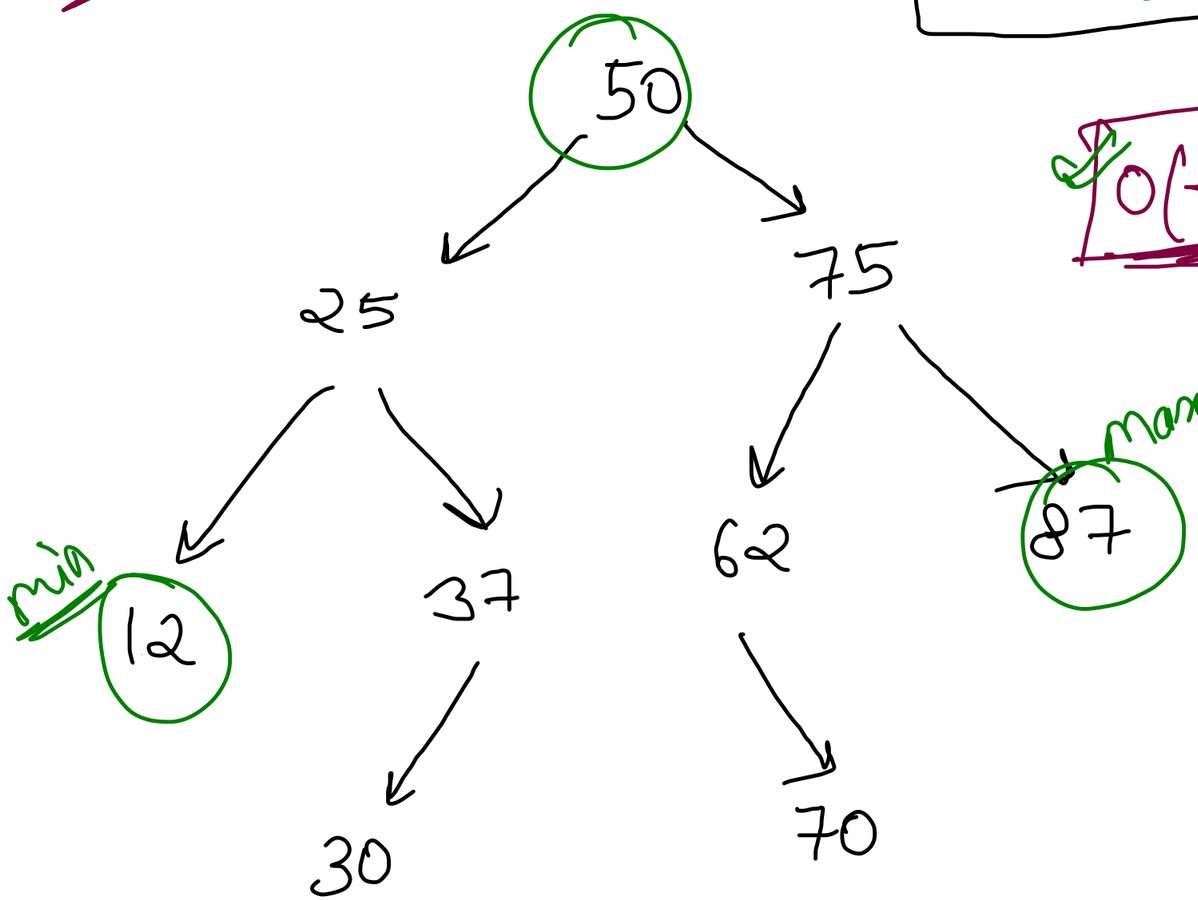


node { → → → int freq



Max & Min

Time O(N) ⟸ { size, sum ⇒ same as BT



O(h)

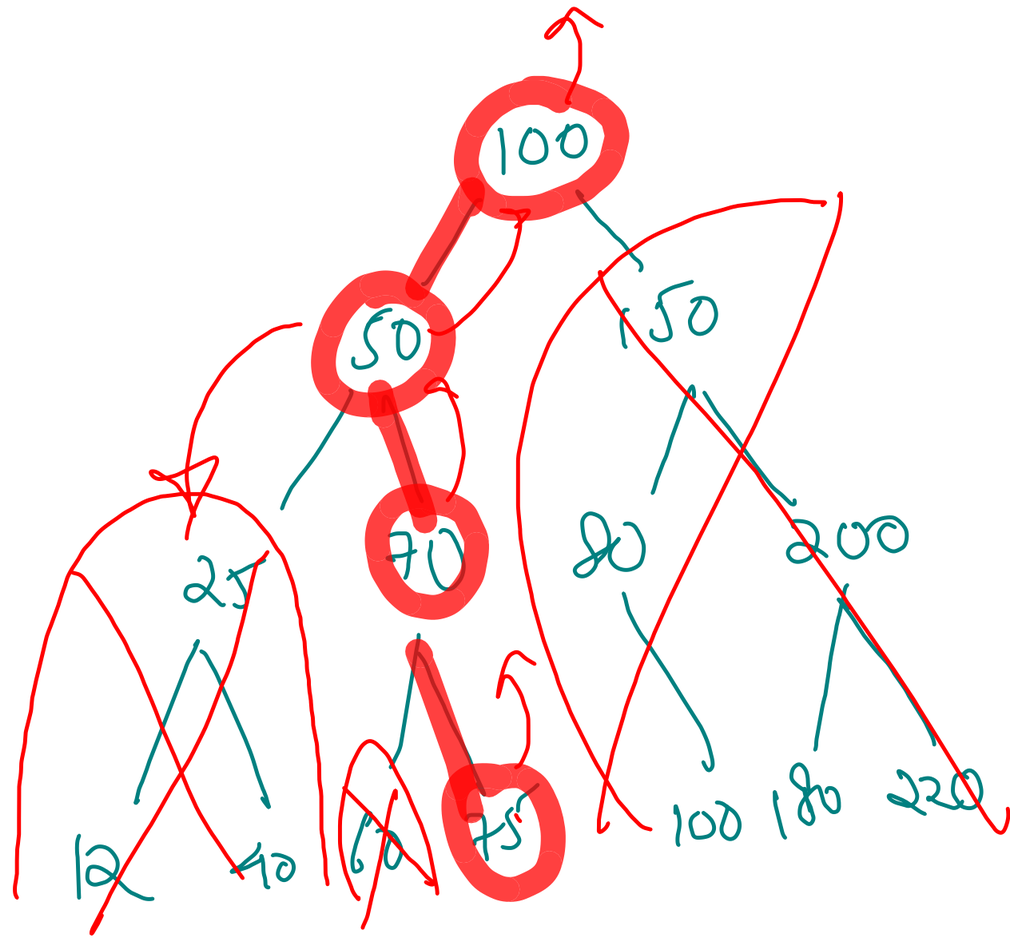
Min
↳ Move as left as possible

Max
↳ Move as right as possible

```
public static int max(Node node) {
    if(node.right == null)
        return node.data;
    return max(node.right);
}

public static int min(Node node) {
    if(node.left == null)
        return node.data;
    return min(node.left);
}
```

Find / Search in BST

```java
public static boolean find(Node node, int data){
    if(node == null) return false;

    if(node.data == data) return true;

    if(data < node.data)
      return find(node.left, data);

    return find(node.right, data);
}
```



BST search → O(h) = Balanced BST O(log₂N)

75

#Tentative Level 1 Schedule

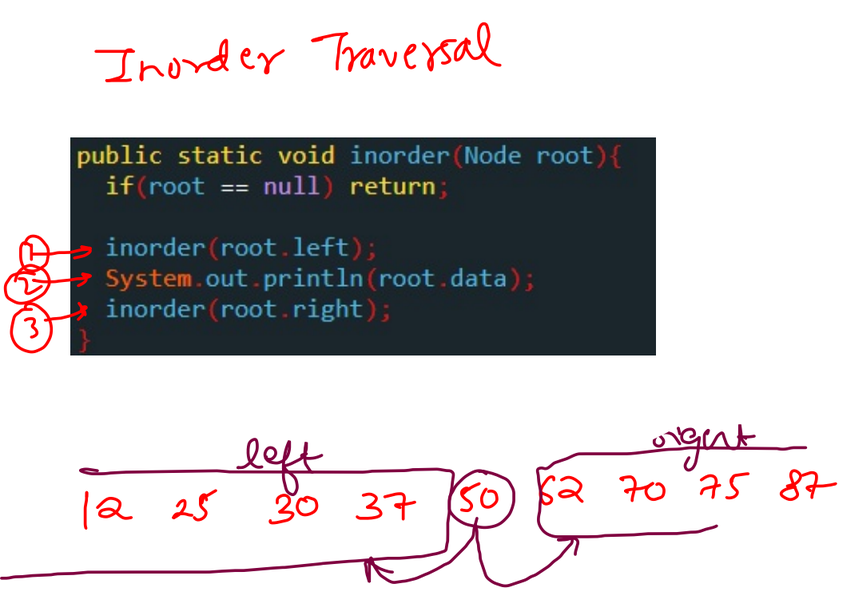
1. 23rd october :→ BST Lecture 2 (Saturday)
2. 24th october :→ BST Lecture 3 (Sunday Morning)
   Hashmap Lecture 1 (Sunday Evening)
3. 28th october :- Hashmap Lecture 2 (Thursday)
4. 30th october :→ Heap Lecture 1 & 2 (Saturday Morning & Evening)
5. 31st October :→ Morning class :→ Remaining Questions.
   (Binary Search, + other questions)
   ↳ Evening class off
6. 4th November :→ Diwali off (Thursday)
7. 6th November :→ Bhaiya Dhuj off (Morning and/or Evening class)
8. 7th Nov - 14th Nov : Graph Lectures Resumes {6 Lectures}
9. 18th nov (Thursday) & 20th nov (Saturday AM & E) :→ Level 1 revision off
10. 21st nov (Sunday) :→ LEVEL 2 start

9-5
↑
x → Discuss + code discuss
y → 2-3
z → 1-2 HW

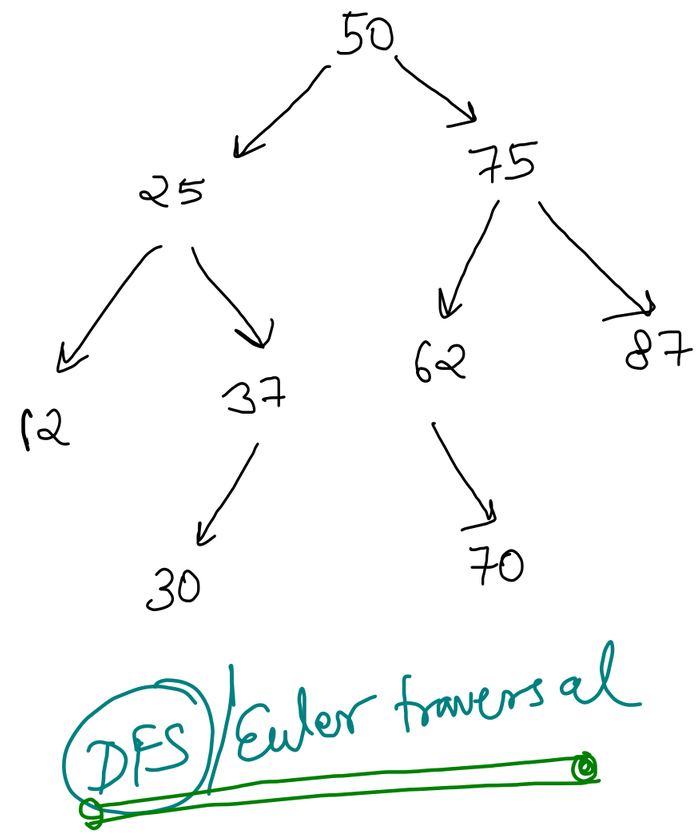
(2) Property :→ Inorder traversal of BST will always give sorted order.

Inorder Traversal

```
public static void inorder(Node root){
  if(root == null) return;

  inorder(root.left);
  System.out.println(root.data);
  inorder(root.right);
}
```



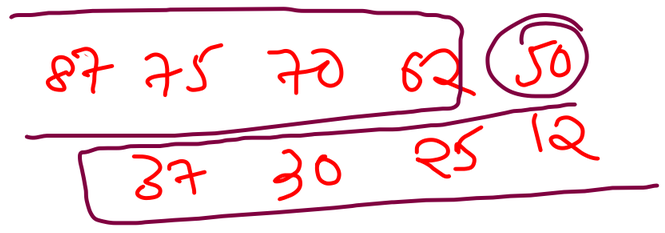
Increasing order



DFS / Euler traversal

Reverse-Inorder Traversal

```
public static void revInorder(Node root){
  if(root == null) return;

  revInorder(root.right);
  System.out.println(root.data);
  revInorder(root.left);
}
```



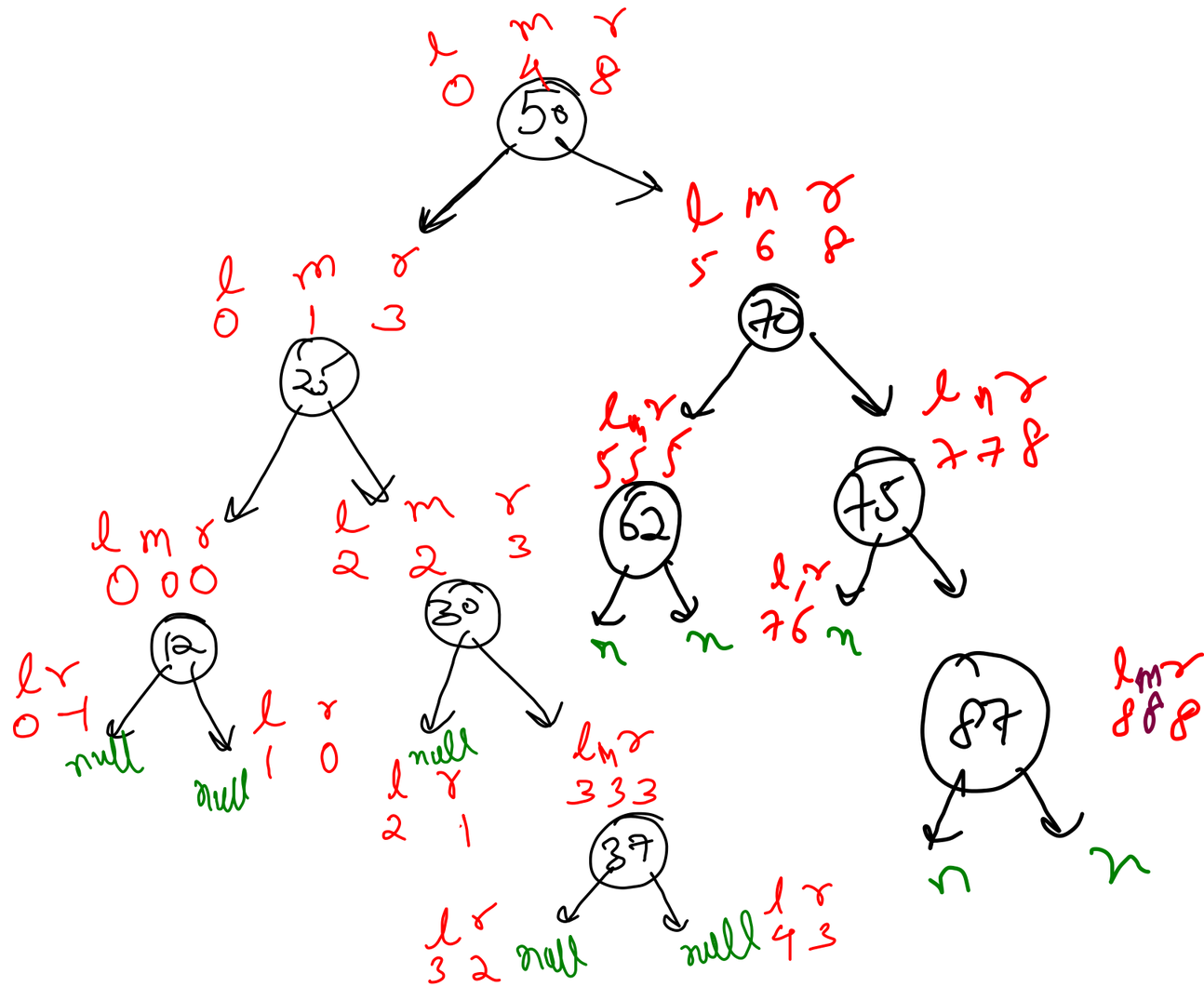
Decreasing order

Construction

{ 12 , 25 , 30 , 37 , 50 , 62 , 70 , 75 , 87 }

l = 0 ; r = n-1

```java
public TreeNode construct(int[] nums, int l, int r){
    if(l > r) return null;

    int mid = (l + r) / 2;

    TreeNode root = new TreeNode(nums[mid]);

    root.left = construct(nums, l, mid - 1);
    root.right = construct(nums, mid + 1, r);
    return root;
}

public TreeNode sortedArrayToBST(int[] nums) {
    TreeNode root = construct(nums, 0, nums.length - 1);
    return root;
}
```



O(N) Time Comp

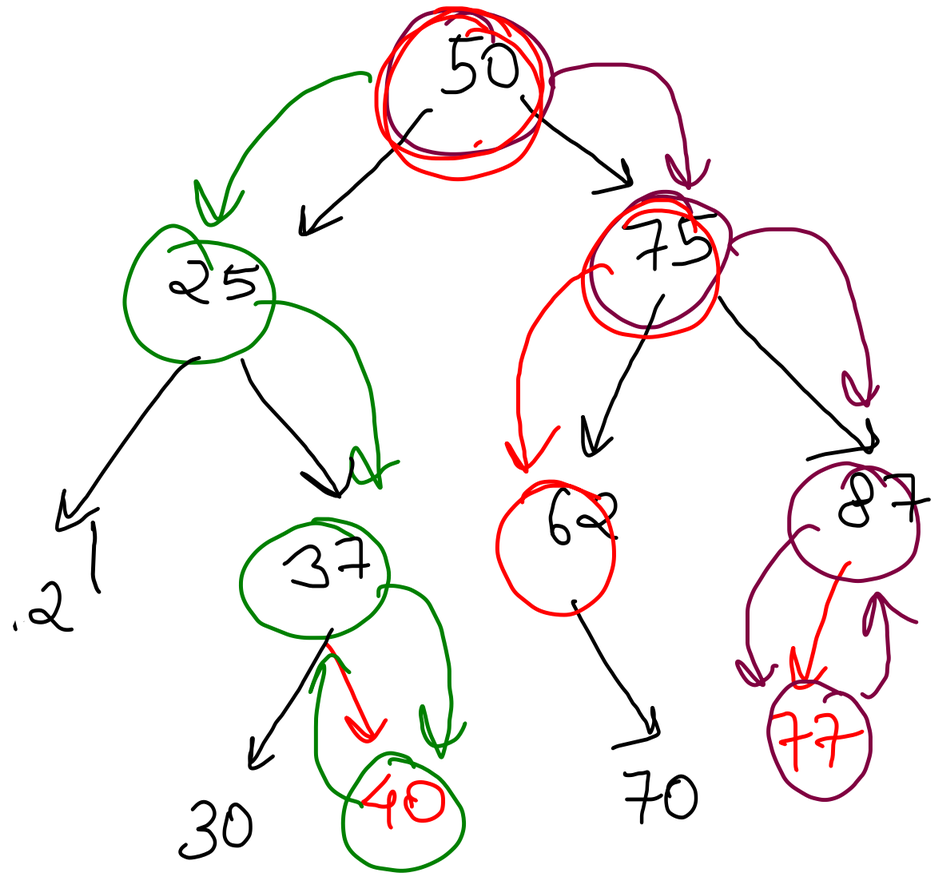
O(h) = O($\log_2 N$) Recursion

Insert Nodes / Add Nodes

O(1) ⟹ { $O(\log_2 n)$ balanced
O(n) skewed

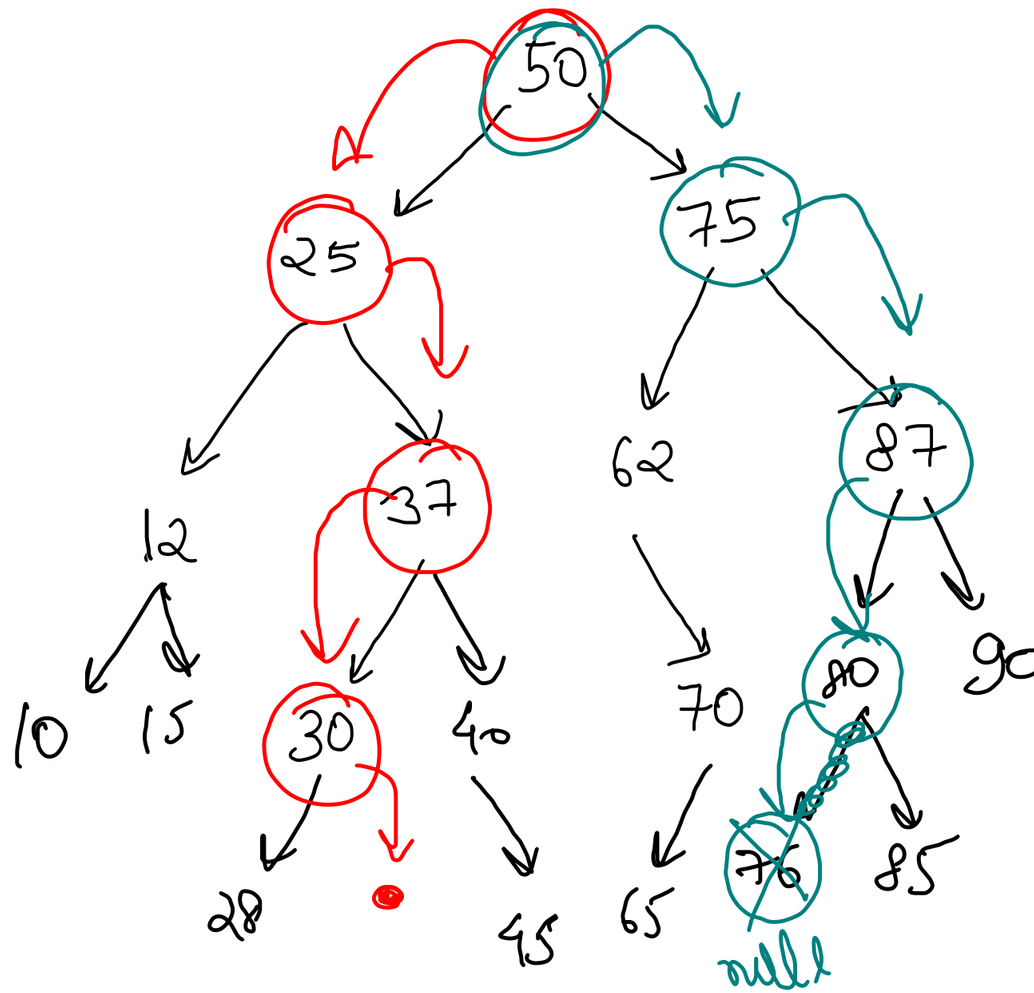
Time

```
public static Node add(Node node, int data) {
    if(node == null){
        Node newNode = new Node(data, null, null);
        return newNode;
    }

    if(node.data == data) return node;

    else if(node.data < data)
      node.right = add(node.right, data);
    else
      node.left = add(node.left, data);

    return node;
}
```



Remove node in BST

- Node does not exist (31)
- leaf node (0 child) (76)
- single child (1 child)
- 2 children node




```
if (node == null) return null;
if (node.data == data) {
        leaf node: return null;
} else if (node.data > data)
        node.left = remove(node.left, data)
  else {
        node.right = remove (node.right.data)
  }
```

Remove node in BST

O(h) time

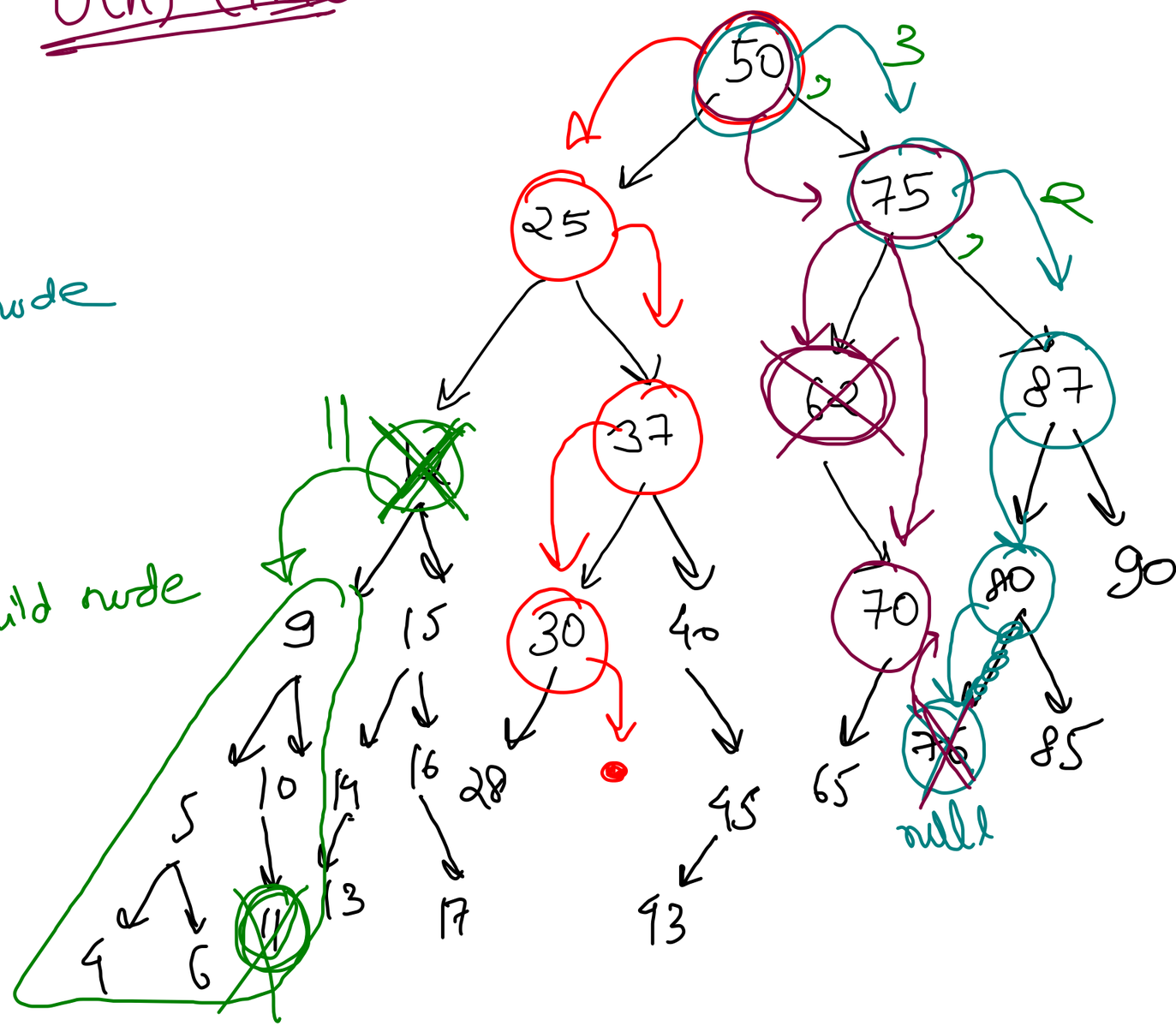
```
public static Node remove(Node node, int data) {
  if(node == null){
      // if node does not exist
      return null;
  }

  if(node.data == data){

      if(node.left == null && node.right == null){
          // leaf node
          return null;

      } else if(node.left == null){
          // only right child
          return node.right;

      } else if(node.right == null){
          // only left child
          return node.left;

      } else {
          // 2 child
          int inorderPredecessor = max(node.left);
          node.data = inorderPredecessor;
          node.left = remove(node.left, inorderPredecessor);
      }

  } else if(node.data < data)
      node.right = remove(node.right, data);
  else node.left = remove(node.left, data);

  return node;
}
```

does not exist

leaf node

single child

2 child node



Replace with sum of largers

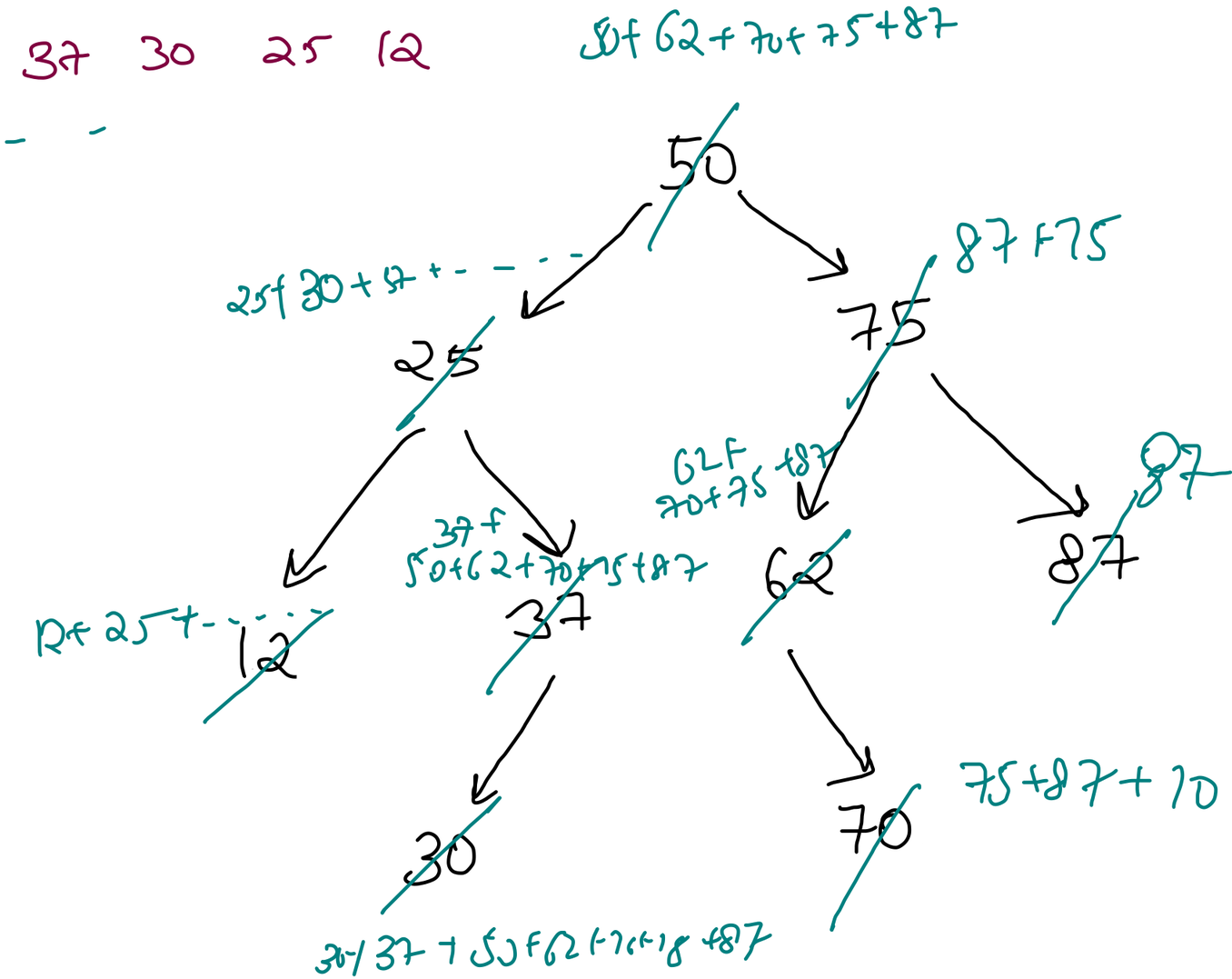
Reverse Inorder : 87 75 70 62 50 37 30 25 12

static sum = 0

87 | 87 f 75 | 87 f 75 f 70 | 87 f 75 f 70 f 62 | - - - -

leetcode variation

```
class Solution {
    static int sum = 0;
    public TreeNode bstToGst(TreeNode root) {
        sum = 0;
        return bstToGstHelper(root);
    }

    public TreeNode bstToGstHelper(TreeNode root){
        if(root == null) return null;

        root.right = bstToGstHelper(root.right);

        sum += root.val;
        root.val = sum;

        root.left = bstToGstHelper(root.left);

        return root;
    }
}
```



LCA of BST

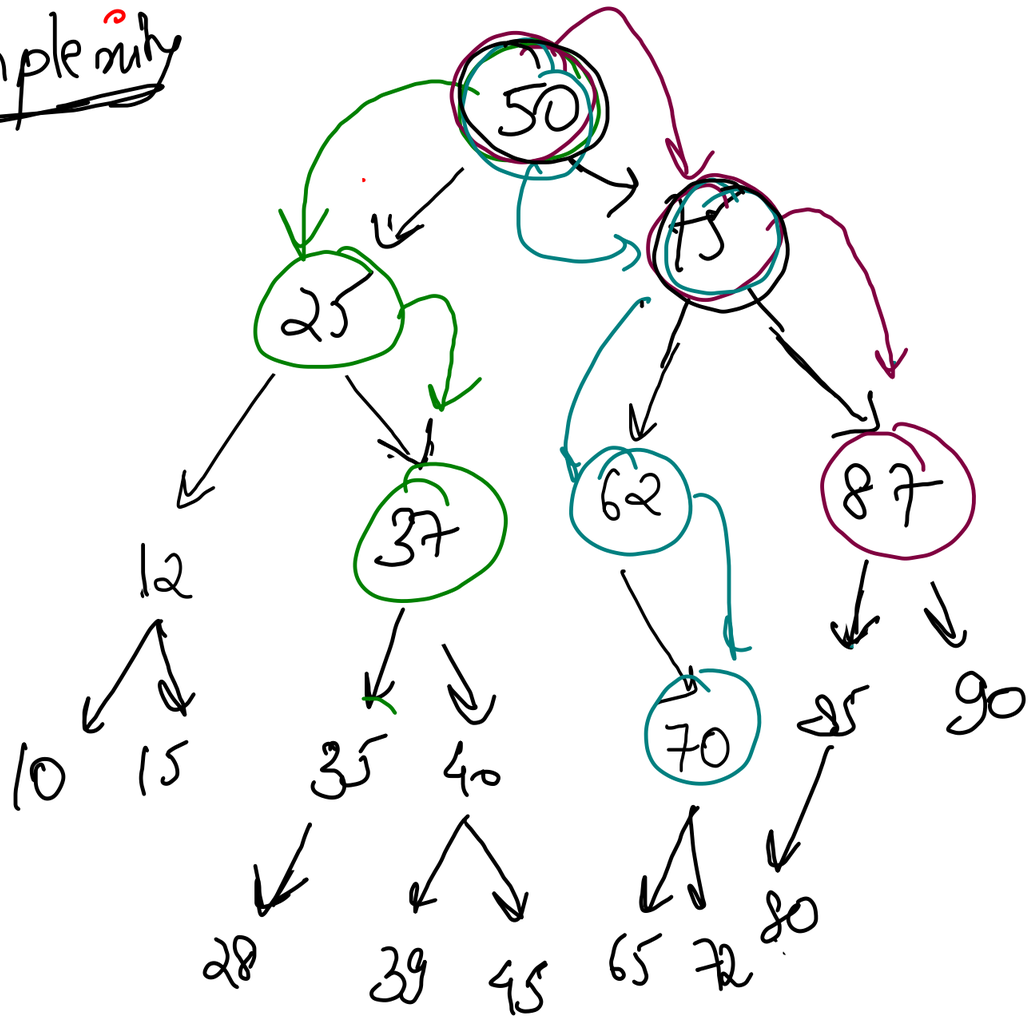
$d1 < d2$

LCA(9,5) = LCA(6,9) ∴ if d2 > d1 swap(d1,d2)

* It will fail if any one node does not exist

O(h) time complexity

```
public static int lca(Node node, int d1, int d2) {
        if(node.data == d1 || node.data == d2){
            return node.data;

        } else if(d1 < node.data && d2 < node.data){
            return lca(node.left, d1, d2);

        } else if(d1 > node.data && d2 > node.data){
            return lca(node.right, d1, d2);

        } else {
            // d1 < node.data < d2 or d2 < node.data < d1
            return node.data;
        }
}
```



① LCA(35 & 40) = 37

② LCA(87 & 80) = 87

③ LCA(65 & 72) = 70

Print in Range

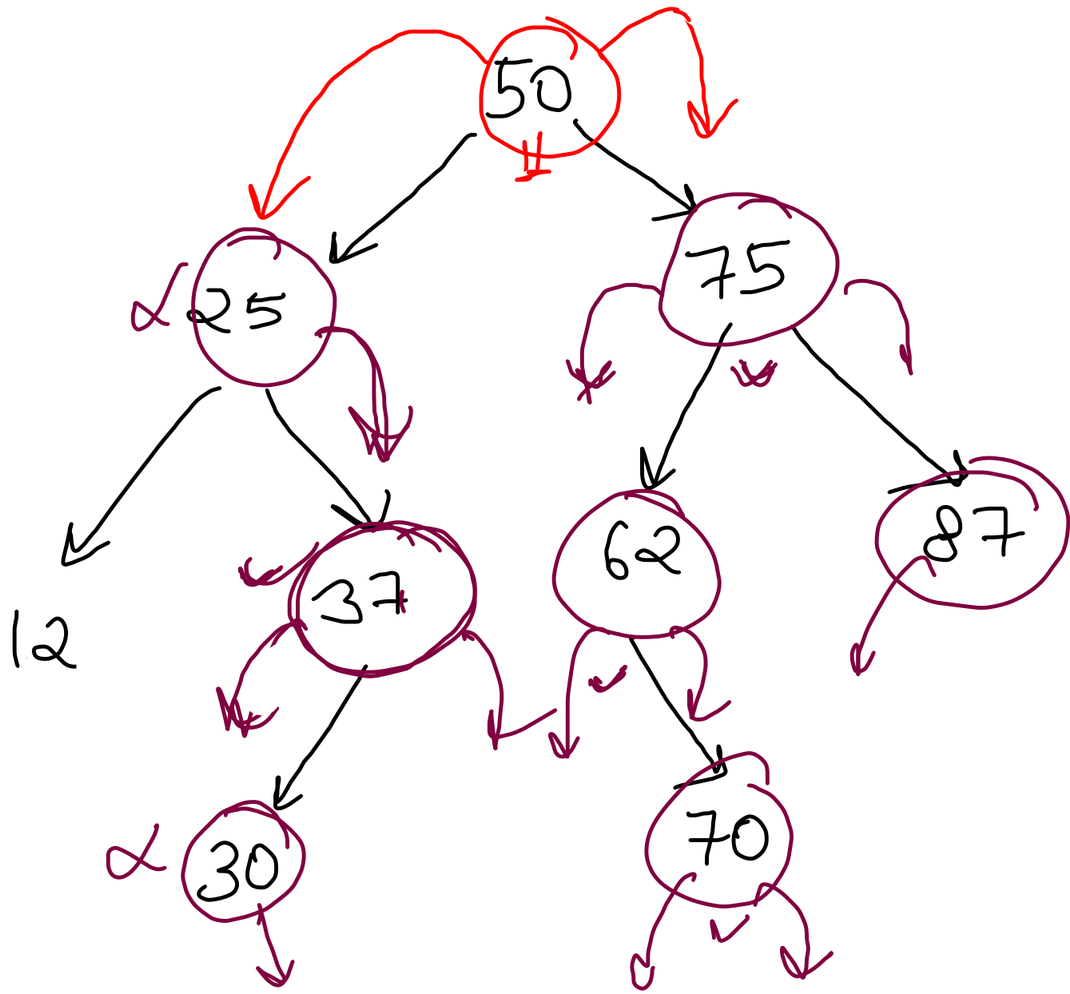
25 [35, 80]

37, 50, 62, 70, 75

```java
public static void pir(Node node, int d1, int d2) {
    if(node == null) return;

    if(!(node.data < d1))
        pir(node.left, d1, d2);

    if(node.data >= d1 && node.data <= d2)
        System.out.println(node.data);

    if(!(node.data > d2))
        pir(node.right, d1, d2);
}
```

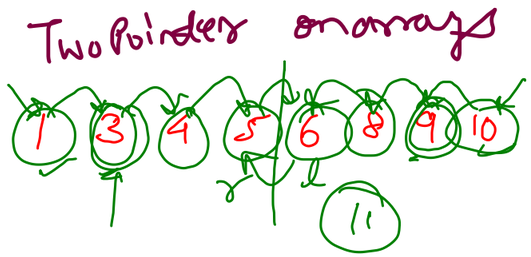
Best case: O(h)

Worst case: O(n)



Target Sum Pair

Two Pointers on arrays



{1,9}
{3,8}
{5,6}

{2 3 4 5 7 8 9 10 12 13 15 16 18 20}

- Approach 1:

Time : O(n*h)

Space : O(h) recursion

- Approach 2:-

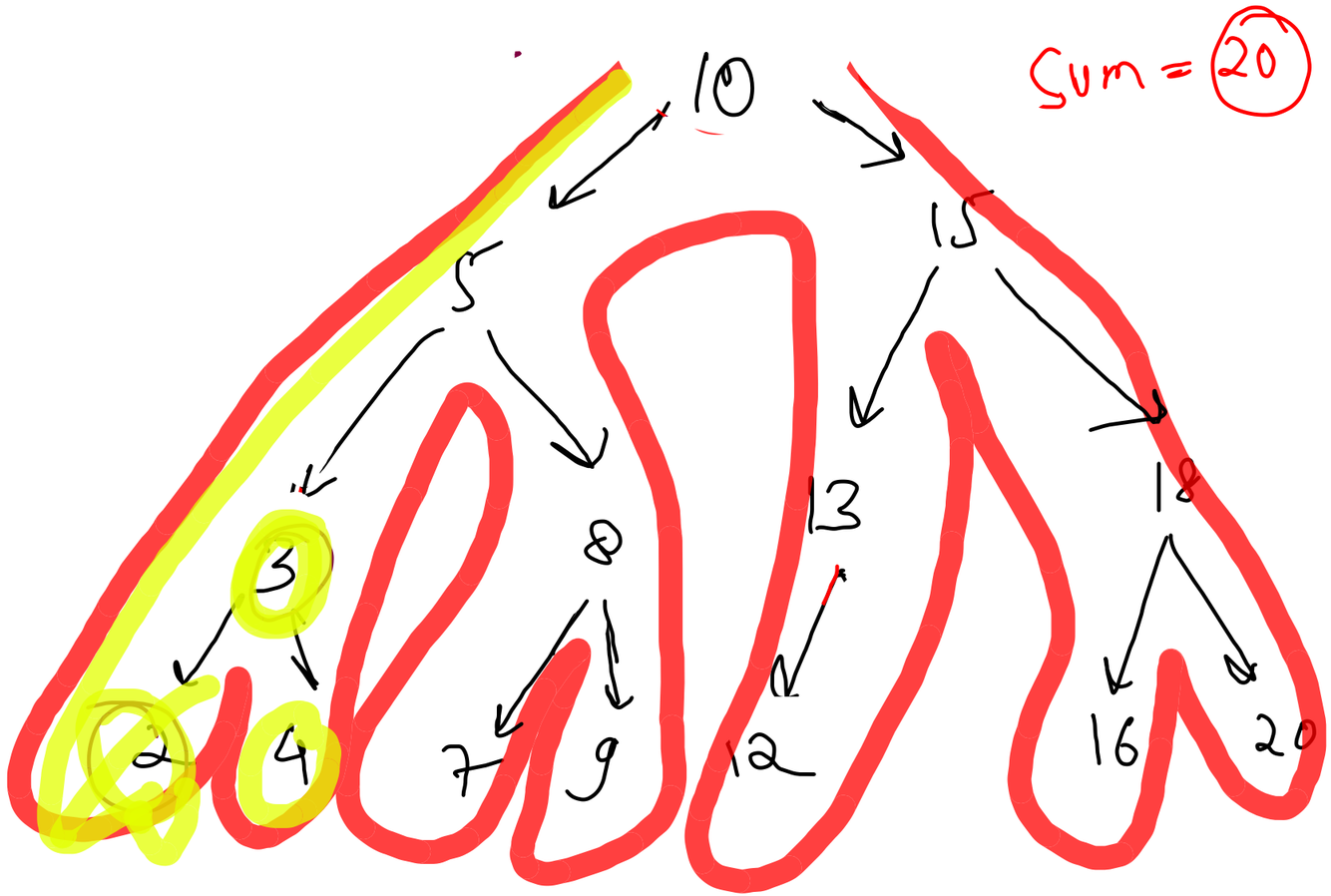
Time : O(n)

Space : O(n) extra space

Approach 3:

Time :→ O(n)

Space :→ O(h) stack



```java
public static boolean find(Node curr, int target){
  if(curr == null) return false;
  if(curr.data == target) return true;

  if(curr.data < target) return find(curr.right, target);
  return find(curr.left, target);
}

public static void approach1Helper(Node curr, int target, Node root){
    if(curr == null) return;

    approach1Helper(curr.left, target, root);

    if(curr.data >= target/2) return;

    int complement = target - curr.data;
    if(find(root, complement) == true){
        System.out.println(curr.data + " " + complement);
    }

    approach1Helper(curr.right, target, root);
}
public static void approach1(Node root, int target){
   approach1Helper(root, target, root);
}
```

O(h)

O(n*h)

Approach 1

For each node,

find complement.

Also node < sum/2

Time :→ O(n * h)

Space :→ O(h)

Recursion.

```java
public static void inorder(Node root, ArrayList<Integer> arr){
  if(root == null) return;

  inorder(root.left, arr);
  arr.add(root.data);
  inorder(root.right, arr);
}

public static void approach2(Node root, int target){
  ArrayList<Integer> arr = new ArrayList<>(); // o(n) space
  inorder(root, arr); // insert elements in sorted order in arraylist - O(n)

  // Two Pointer on arraylist - o(n)
  int left = 0, right = arr.size() - 1;

  while(left < right){
      int sum = arr.get(left) + arr.get(right);
      if(sum == target){
          System.out.println(arr.get(left) + " " + arr.get(right));
          left++; right--;
      } else if(sum < target){
          left++;
      } else {
          right--;
      }
  }
}
```

Approach 2

→ Store inorder in array

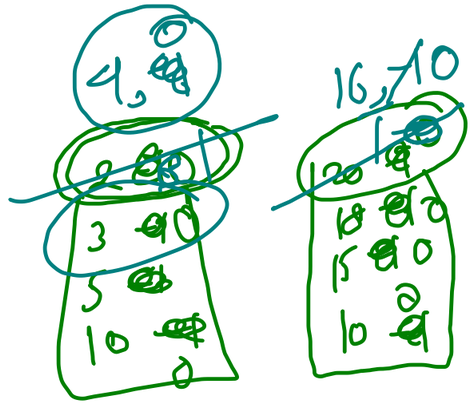
→ Apply two pointer on array

Time ⇒ O(n) {inorder + 2 Pointer}

Space ⇒ O(n) extra space (array)

Time :→ O(n)
space

```
public static void approach3(Node root, int target){
    Stack<Pair> inorder = new Stack<>();
    inorder.push(new Pair(root, -1));

    Stack<Pair> reverseInorder = new Stack<>();
    reverseInorder.push(new Pair(root, -1));

    int left = iterativeInorder(inorder);
    int right = iterativeReverseInorder(reverseInorder);

    while(left < right){

        if(left + right == target){
          System.out.println(left + " " + right);
          left = iterativeInorder(inorder);
          right = iterativeReverseInorder(reverseInorder);

        } else if(left + right < target){
          left = iterativeInorder(inorder);
        } else {
          right = iterativeReverseInorder(reverseInorder);
        }
    }
}
```



Approach 3

2,18
4,16

5,15
8,12