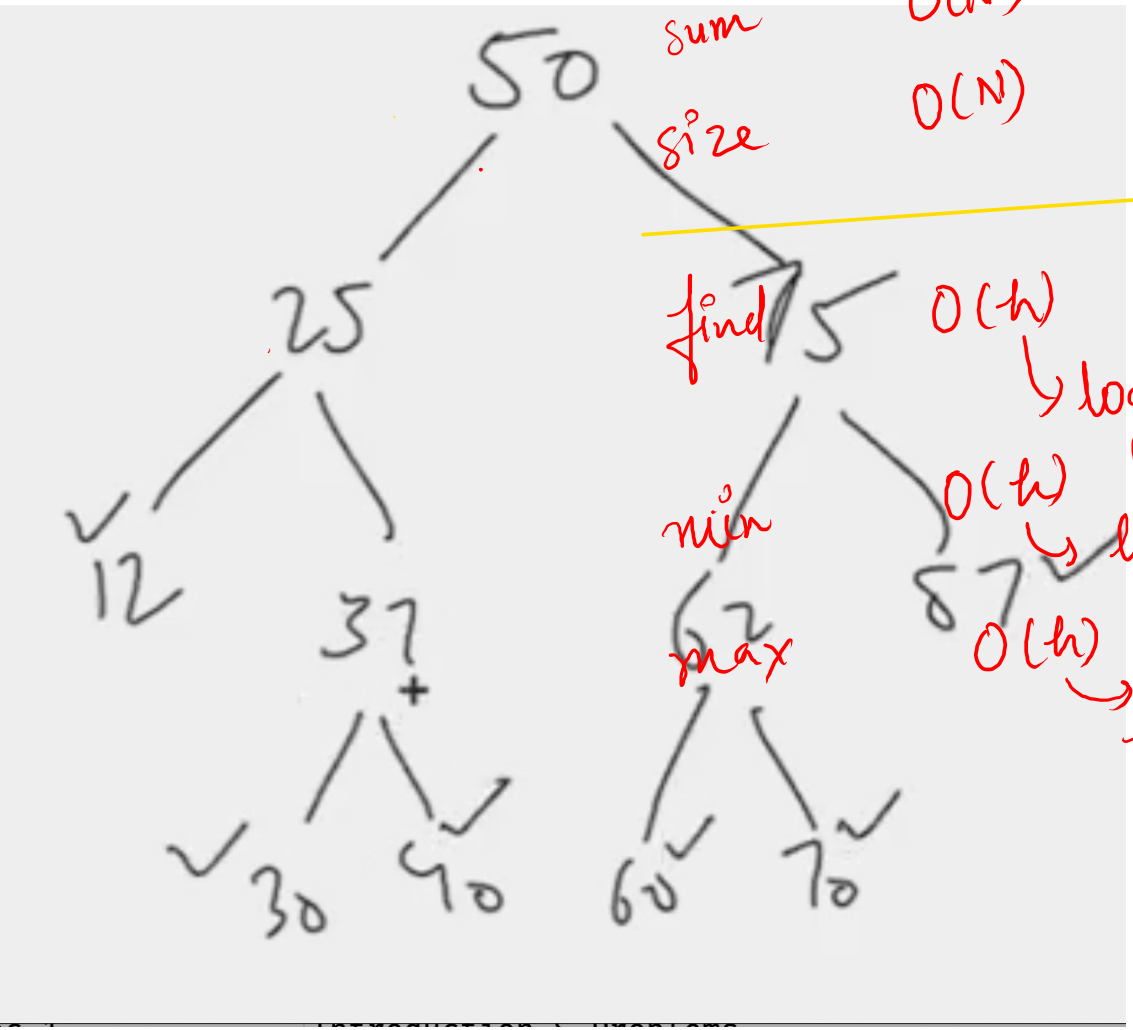
Binary Search Tree



BST

O(N)

O(N)

BT

O(N)

O(N)

} Independent On Node value

O(h)

↳ log(n)

O(h)

↳ log(n)

O(h)

↳ log(n)

O(N)

O(N)

O(N)

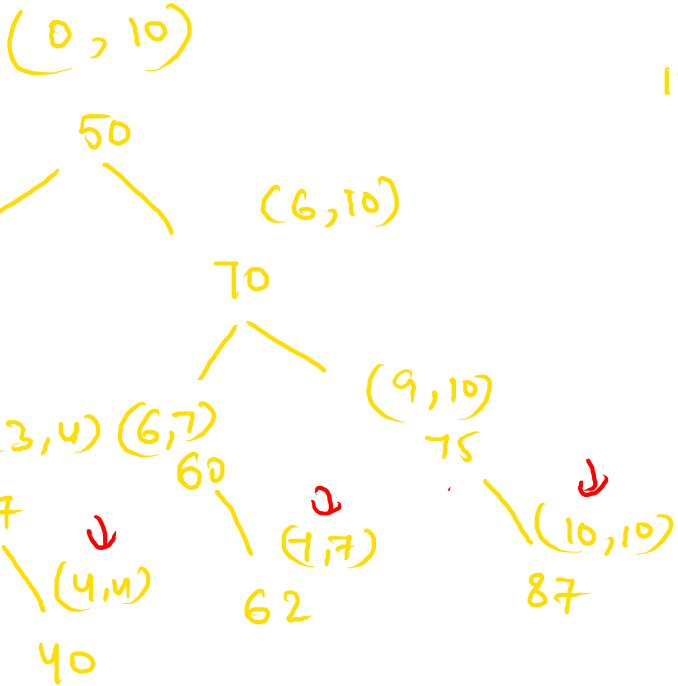
} dependent On Node value, ∴ BST is faster

arr

0	1	2	3	4	5	6	7	8	9	10
12	25	30	37	40	50	60	62	70	75	87

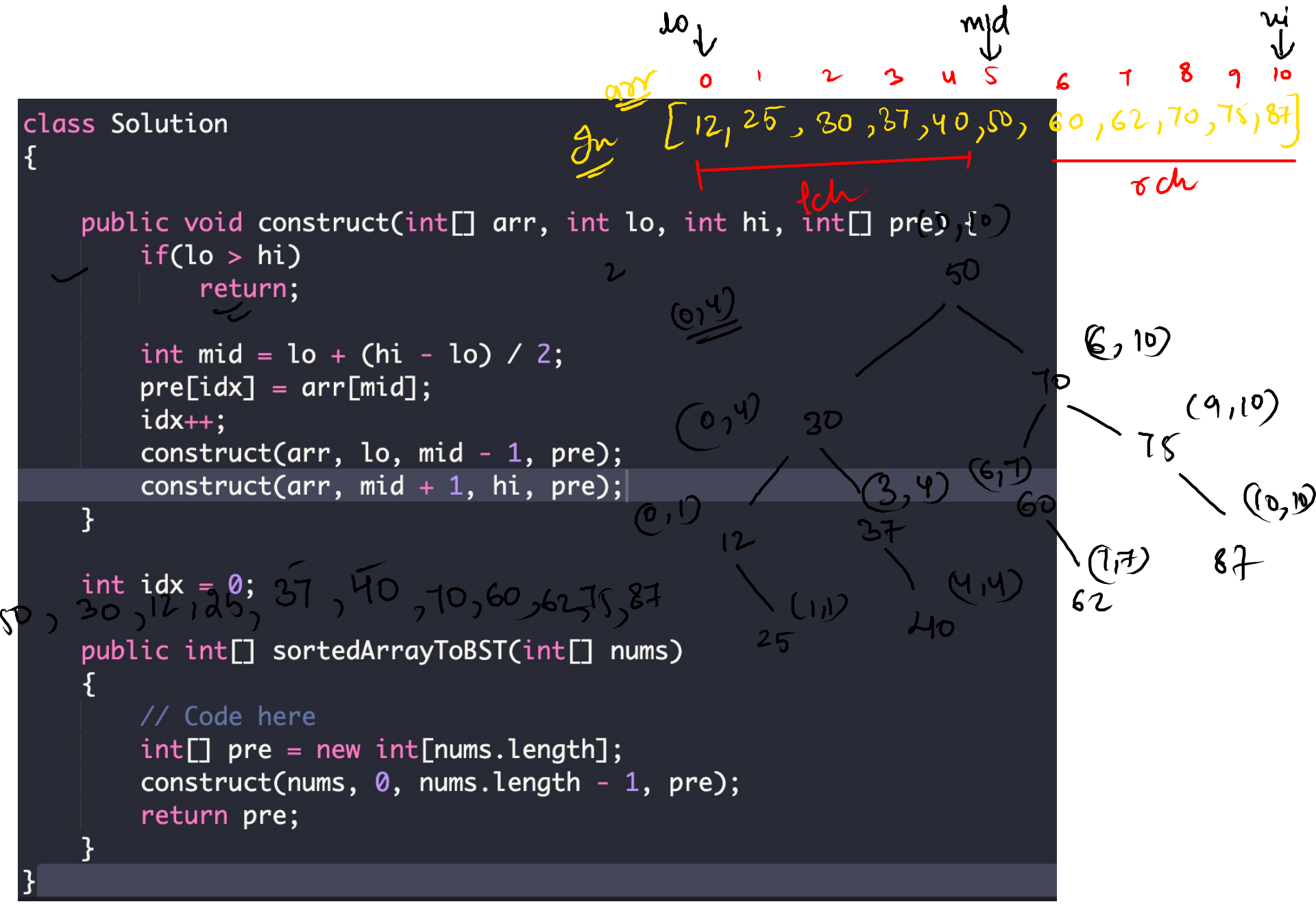
lch

rch



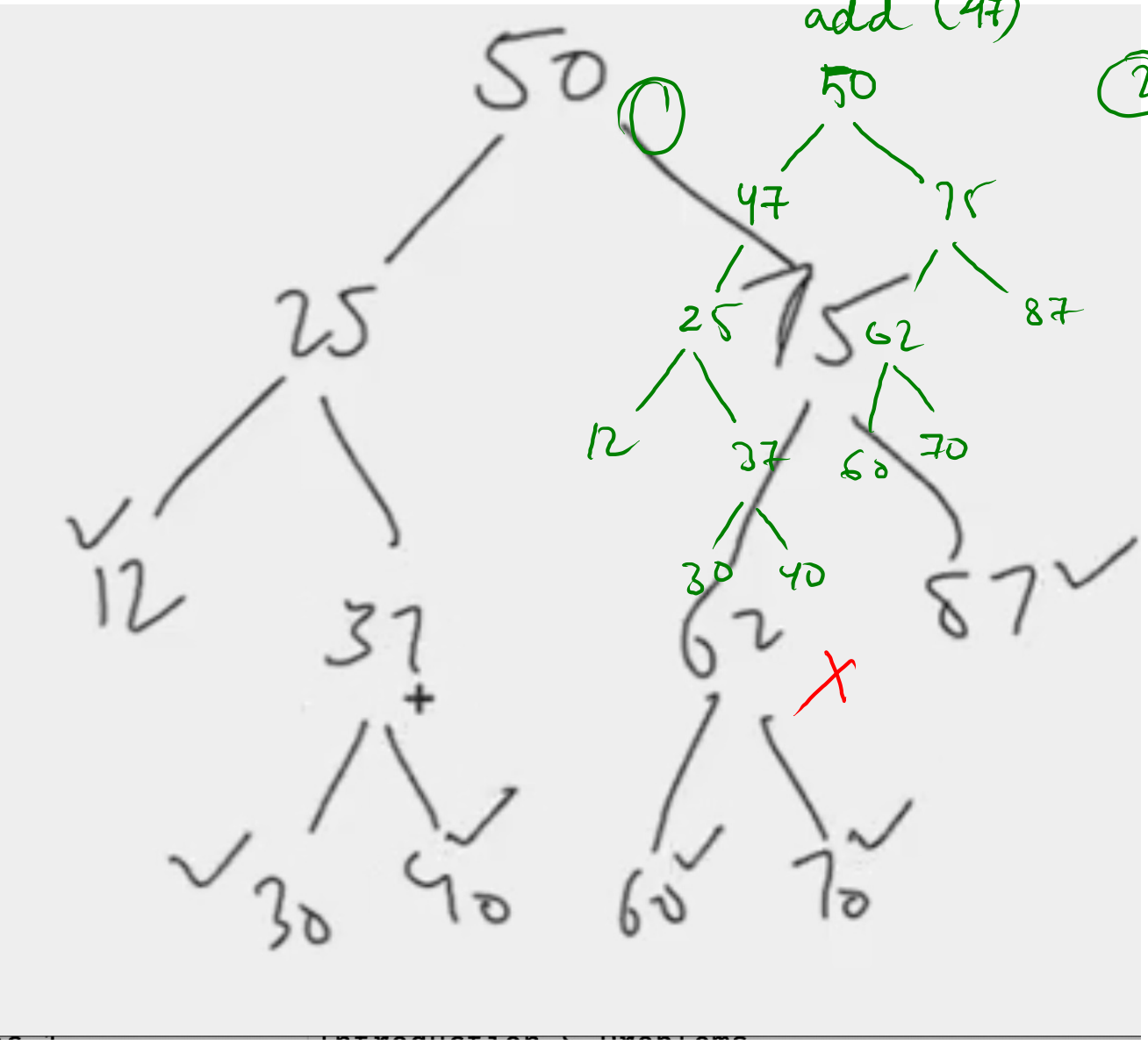
Base case!

(lo > hi)

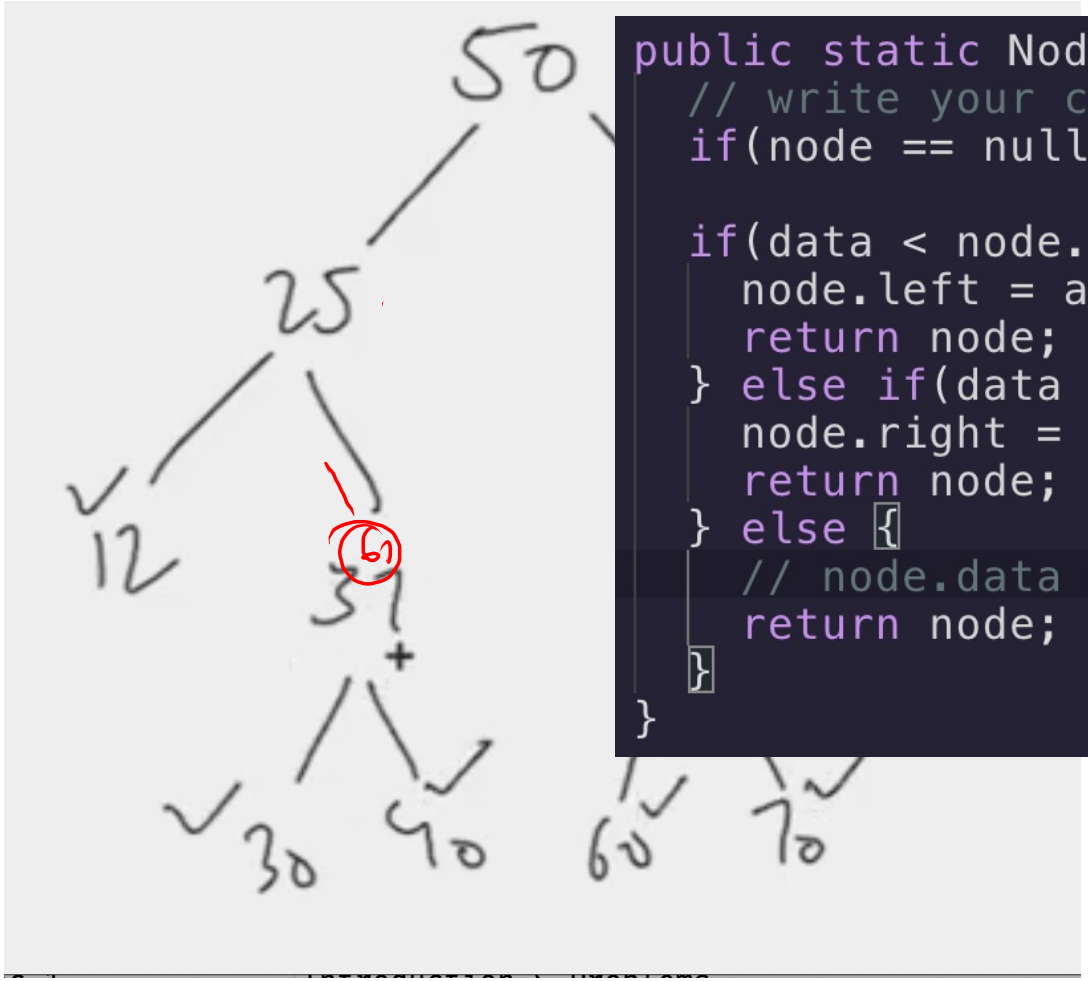

lo
mid
hi
0 1 2 3 4 5 6 7 8 9 10
arr
[12, 25, 30, 37, 40, 50, 60, 62, 70, 75, 87]
lch
rch
class Solution
{
public void construct(int[] arr, int lo, int hi, int[] pre) {
if(lo > hi)
return;
int mid = lo + (hi - lo) / 2;
pre[idx] = arr[mid];
idx++;
construct(arr, lo, mid - 1, pre);
construct(arr, mid + 1, hi, pre);
}
int idx = 0;
public int[] sortedArrayToBST(int[] nums)
{
// Code here
int[] pre = new int[nums.length];
construct(nums, 0, nums.length - 1, pre);
return pre;
}
}
50 30 12 25 37 40 70 60 62 75 87
(0,10)
(0,4)
(6,10)
(0,1)
(3,4)
(6,7)
(9,10)
(1,1)
(4,4)
(7,7)
(10,10)
50
30
70
12
37
60
75
25
40
62
87

Que Construct BST from a Sorted Array (OR Inorder traversal)

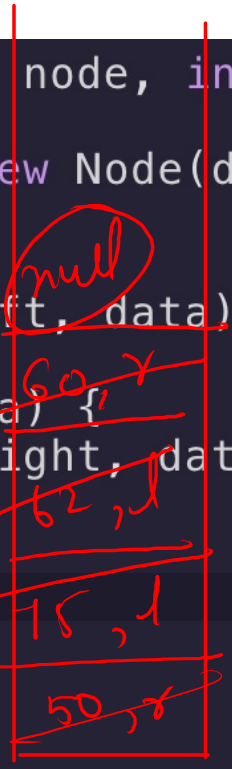
BST


50
25
12
37
30
40
62
60
70
87
add (47)
①
50
47
75
25
62
87
12
37
60
70
30
40
②
47
25
80
12
37
75
62
87
30
40
60
70

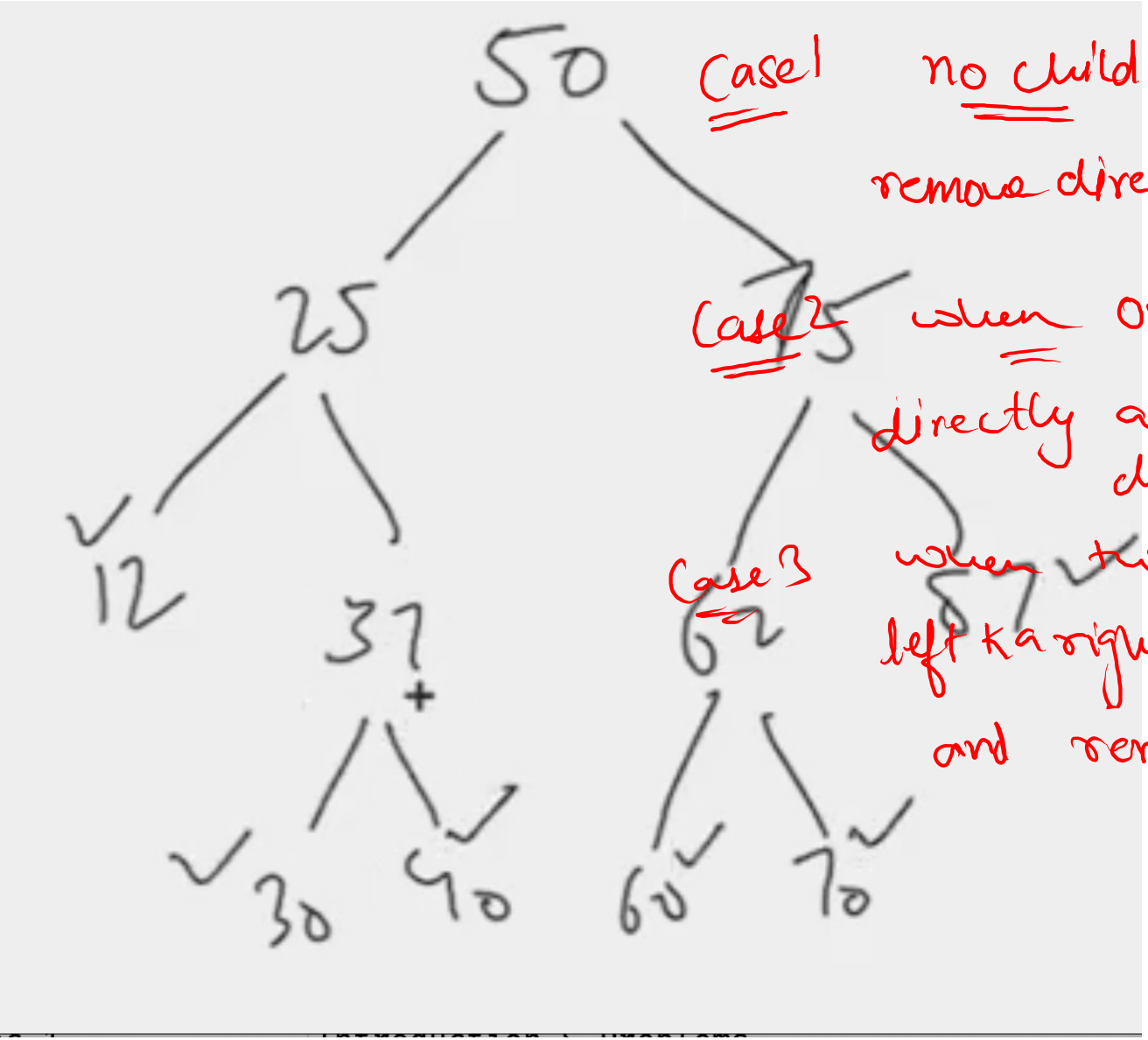
add (61)




```java
public static Node add(Node node, int data) {
  // write your code here
  if(node == null) return new Node(data);

  if(data < node.data) {
    node.left = add(node.left, data);
    return node;
  } else if(data > node.data) {
    node.right = add(node.right, data);
    return node;
  } else {
    // node.data == data
    return node;
  }
}
```



Que Remove from A BST



Case1 no child

remove directly

Case2 when one child

directly attach with that child

Case3 when two child.

left ka rightmost as node.

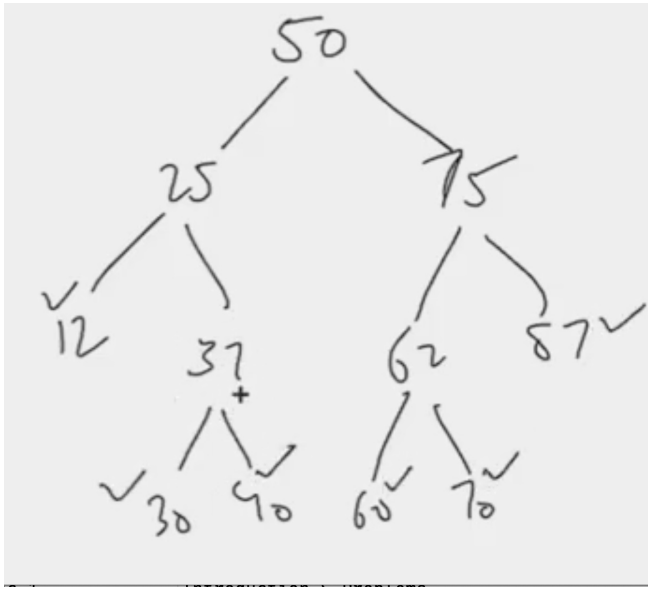
and remove (left ka Rightmost)

Lecture 2

Binary Search Tree

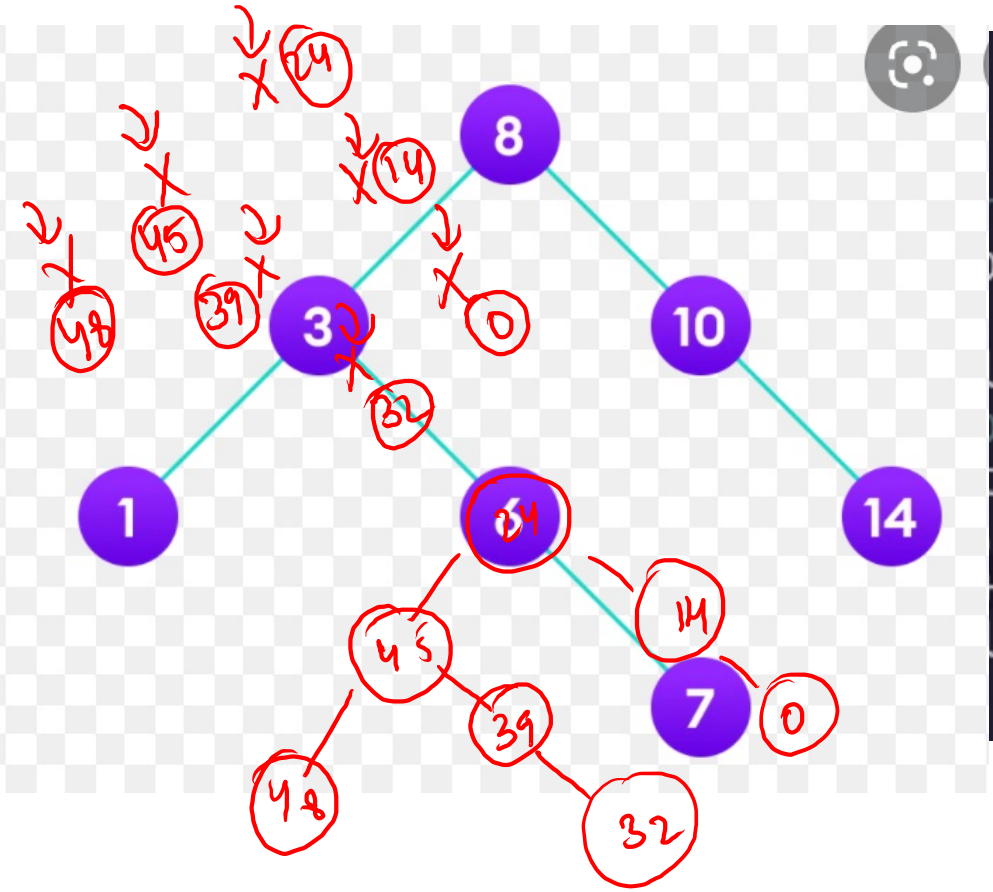
Que Replace with larger Sum

Static Sum



In (smaller) L N R → larger (increasing Order)

Reverse In R N L (decreasing order)



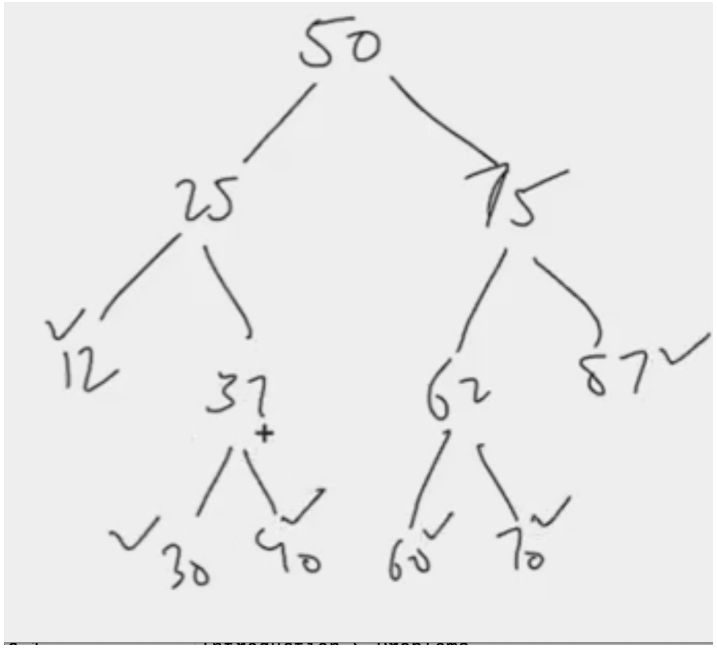

```
int sum = 0;
static void rwsol(Node node){
rite your code here
ode == null) return;

(node.right);
rk
temp = node.data;
data = sum;
= temp;
(node.left);
```

decreasing

R N L

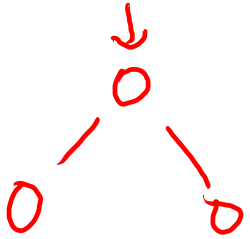
Qu

LCA



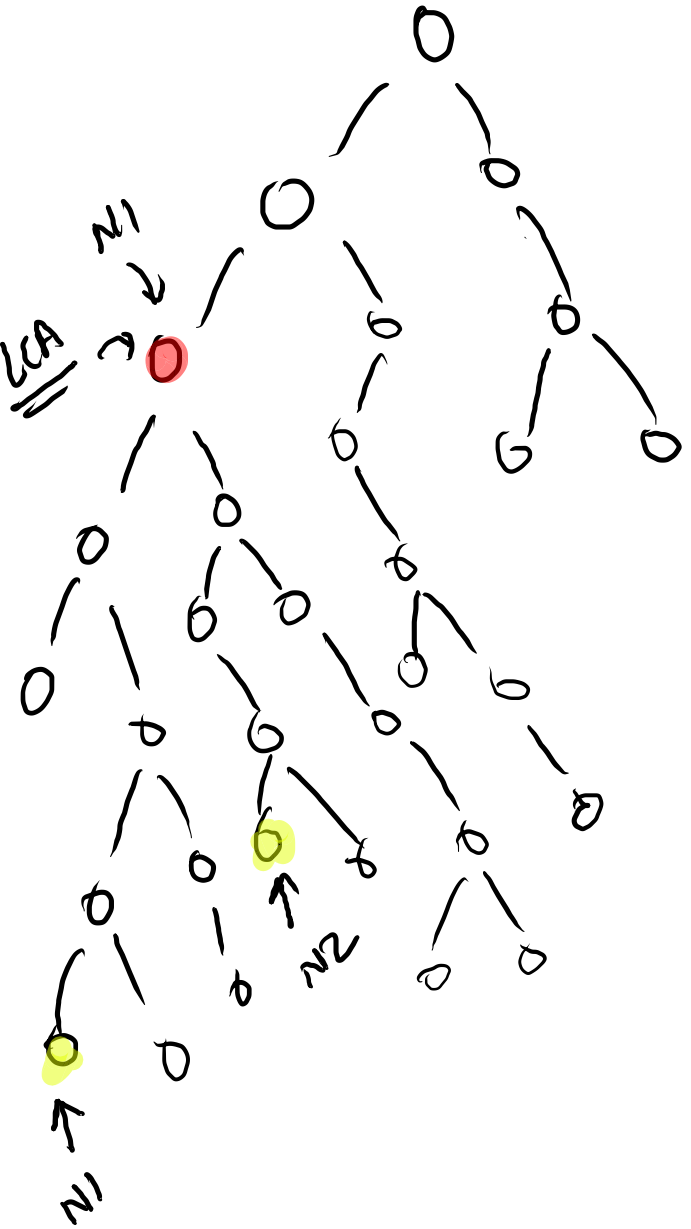
TC : { O(h) ; log(N)

Case1



Case2

Case3

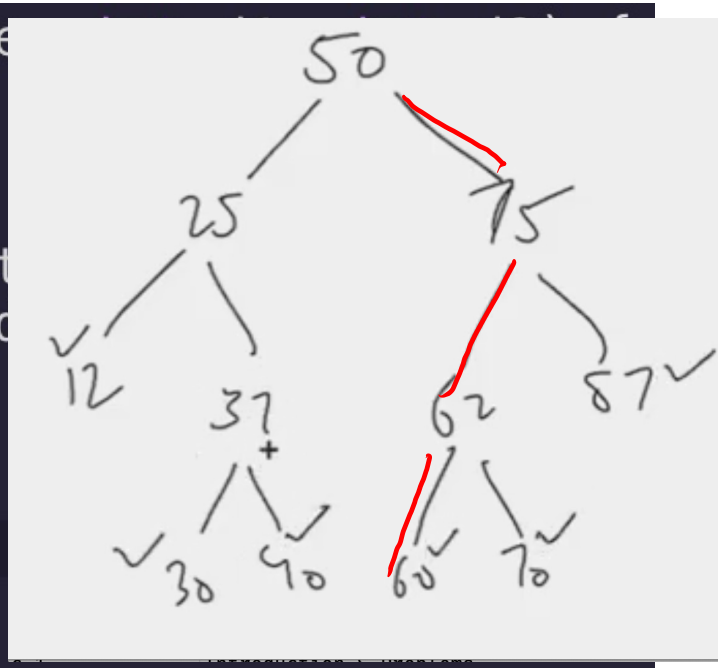


BST → {log(N)}

LCA

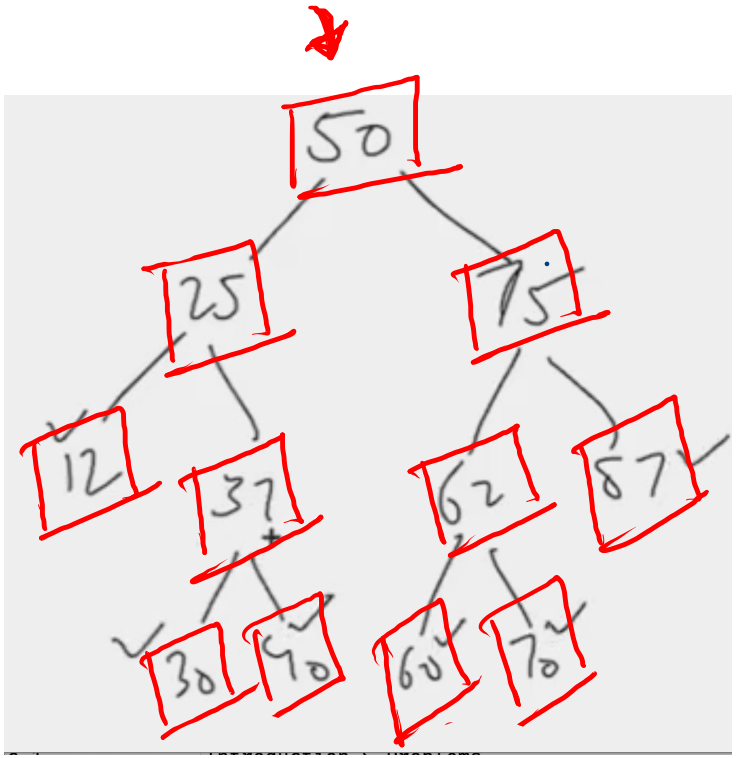
TC = {O(h)
↳ log(N)}

```java
public static int lca(Node node
  // write your code here
  if(node == null) return -1;

  if(node.data > d1 && node.dat
    return lca(node.left, d1, d
  } else if (node.data < d1 &&
    return lca(node.right, d1,
  } else {
    return node.data;
  }
}
```



O(h) → log(n)

Q.ue Target Sum Pair



Target → 100

Pair ⇒ {25, 75}, {30, 70},
{40, 60}

```java
public static boolean find(Node root, int val) {
  if(root == null) return false;
  if(root.data > val) return find(root.left, val);
  else if(root.data < val) return find(root.right, val);
  else return true;
}

public static void dfs(Node root, Node node, int tar) {
  if(root == null) return;

  dfs(node.left, root, tar);

  int comp = tar - node.data;
  if(comp > node.data) {
    if(find(root, comp) == true) {
      System.out.println(node.data + " " + comp);
    }
  }

  dfs(node.right, root, tar);
}
```

Approach 2

12	25	30	37	40	50	60	62	70	75	87

100

target

TC = { O(N)

SC = { O(N)

25, 75

```
if sum < target {
    left ++;
}
else if sum > target {
    right --;
}
else {
    print (arr[left], arr[right]);
    left ++; right --;
}
```

Approch 1

TC = { O(N*h)
       O(N log N)

SC = { O(h)

Approch 2

TC = { O(N)

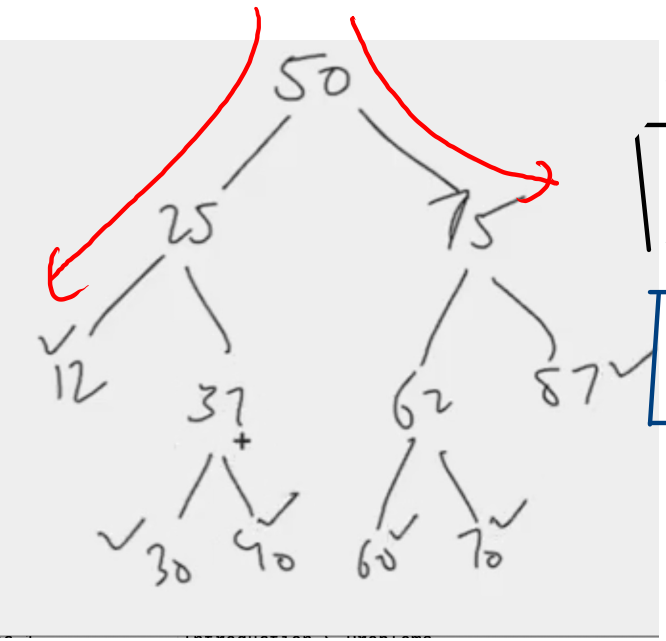
SC = { O(N)

Best Approch

Approch 3

TC = { O(N)

SC = { O(h)



12, 25, 30, 37, 40, 50, 60, 62, 70, 75, 87

87, 75, 70, 62, 60, 50, 40, 37, 30, 25, 12

log(N)

~~1412~~

2511

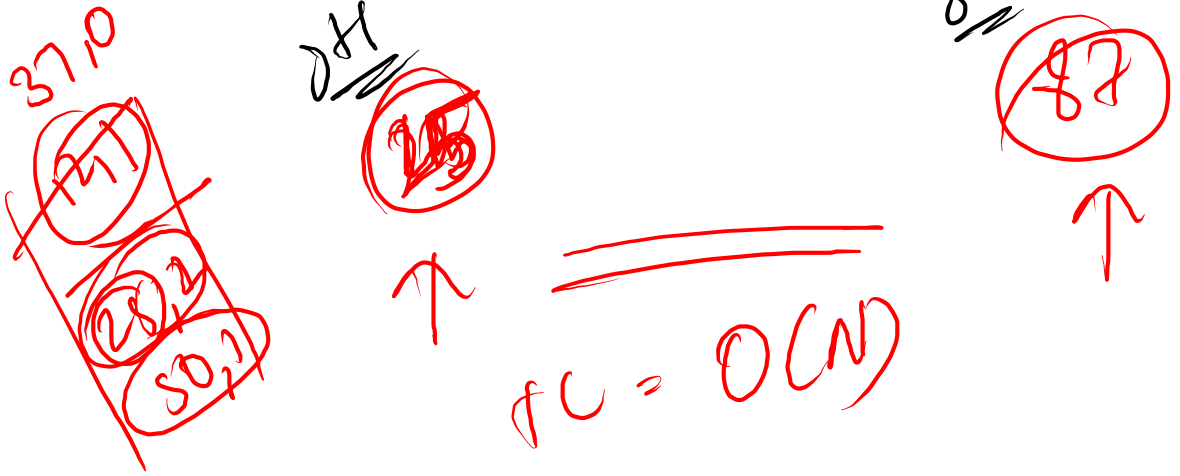
5011

Stack 1 → Normal In

Stack 2 → Reverse In

inorder

deinorder



TC = O(N)