Data Structures

Binary Search Trees 2

CS 225

September 24, 2025

Harsha Tirumala

Hello

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Announcements

- Resources - Lecture Handouts, notes
- Plagiarism - MP stickers (Overall good but a few FAIR cases)
- Exam 2 next week - practice exam 2 to be released today
  Topics - Tree traversals/BSTs
- Congrats on earning 2 EC points for providing feedback!

Learning Objectives

BST - Recap

BST Remove : Concept and Implementation

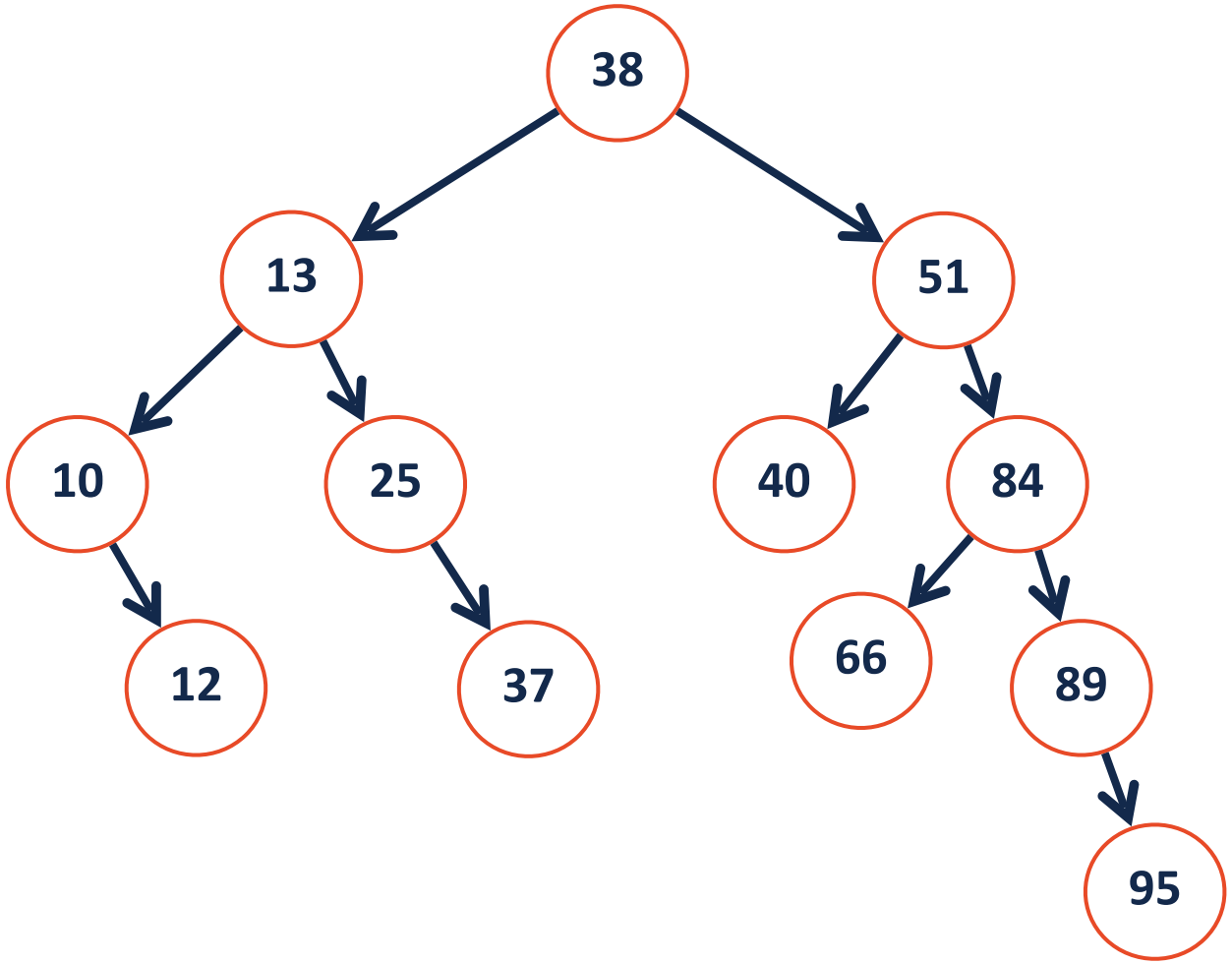
Discuss pros and cons of BST (and possible improvements)

Binary Search Tree (BST)

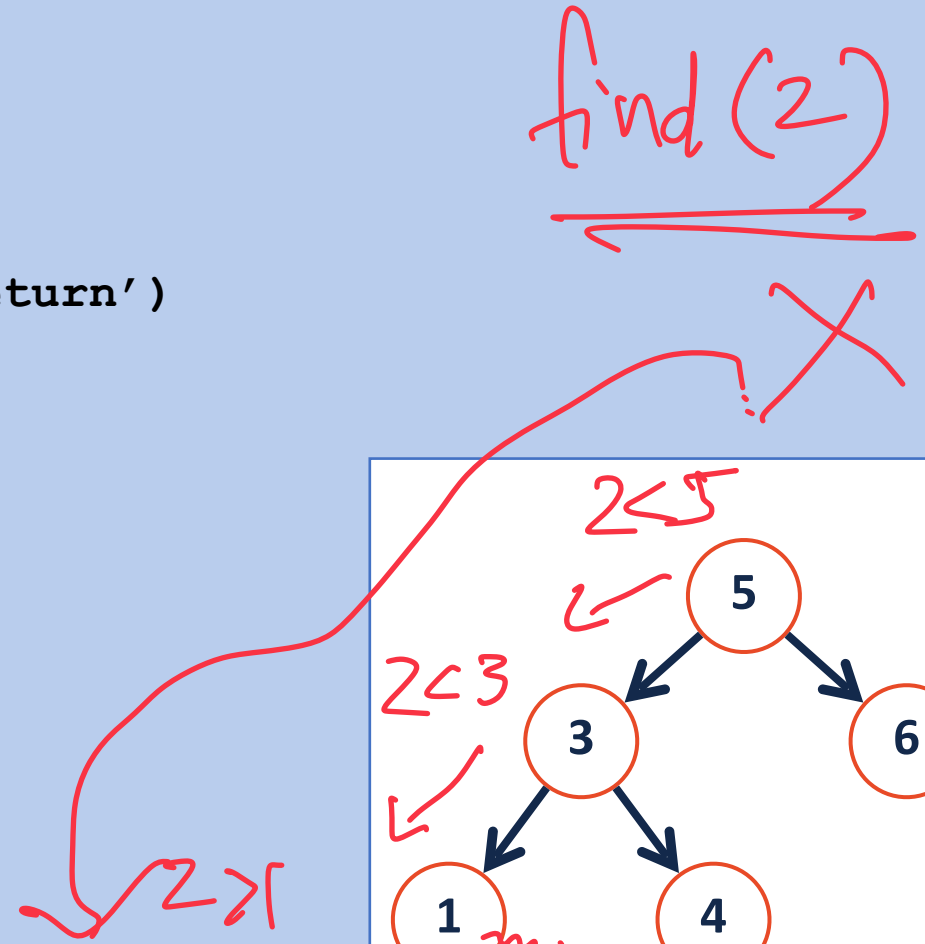
A **BST** is a binary tree $T = TreeNode(key, T_L, T_r)$ such that:

$$\forall n \in T_L,\ n.val < T.val$$

$$\forall n \in T_R,\ n.val > T.val$$



```
1  template<typename K, typename V>
2
3  TreeNode *& _find(TreeNode *& root, const K & key) {
4
5
6  // Base Case
7  if(root == nullptr || root->key == key){
8      return root;
9  }
10
11 // Recursive Step ("Combining step" is 'return')
12 if (root->key > key){
13     return _find(root->left, key);
14 }
15
16 return _find(root->right, key);
17
18
19 }
20
21
22
23
```



```
1  template<typename K, typename V>
2
3  void _insert(const K & key, const V & val) {
4
5
6    TreeNode *& tmp = _find(root, key);
7
8
9    tmp = new treeNode(key, val);
10
11
12 }
```

data

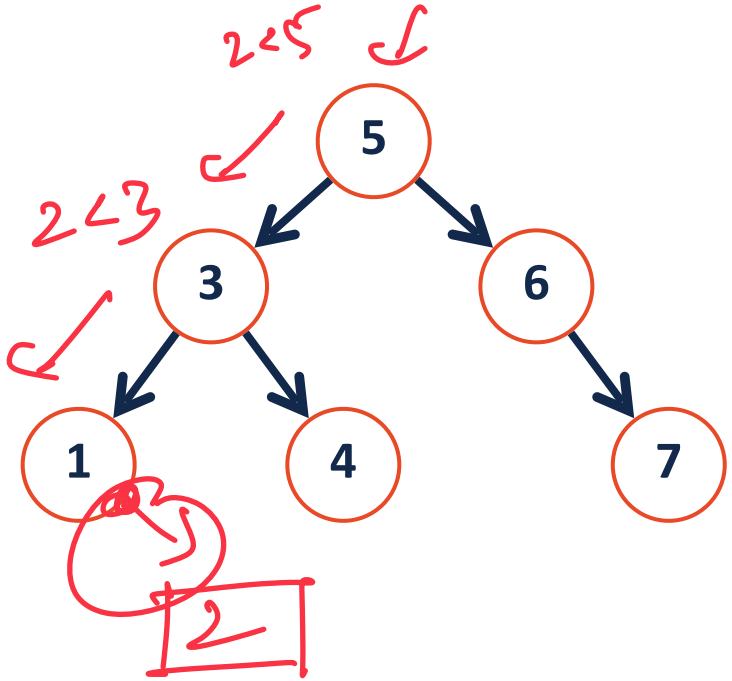
find insert

O(h)

insert(2)

O(h) ← find(_root, 2)

O(1) ← then insert

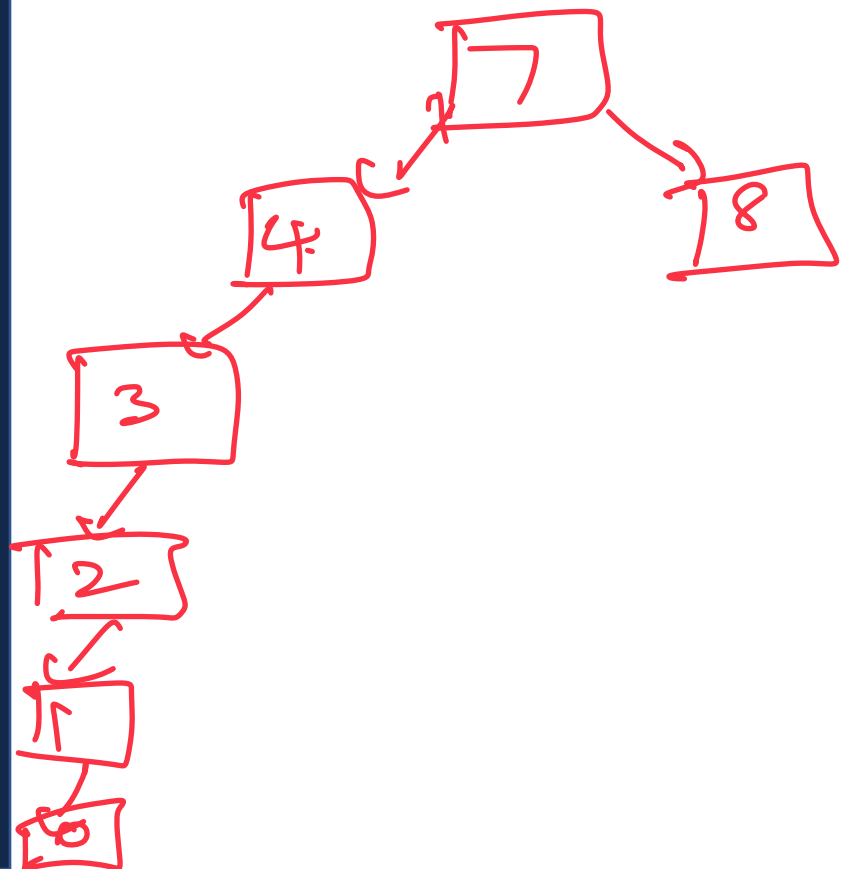


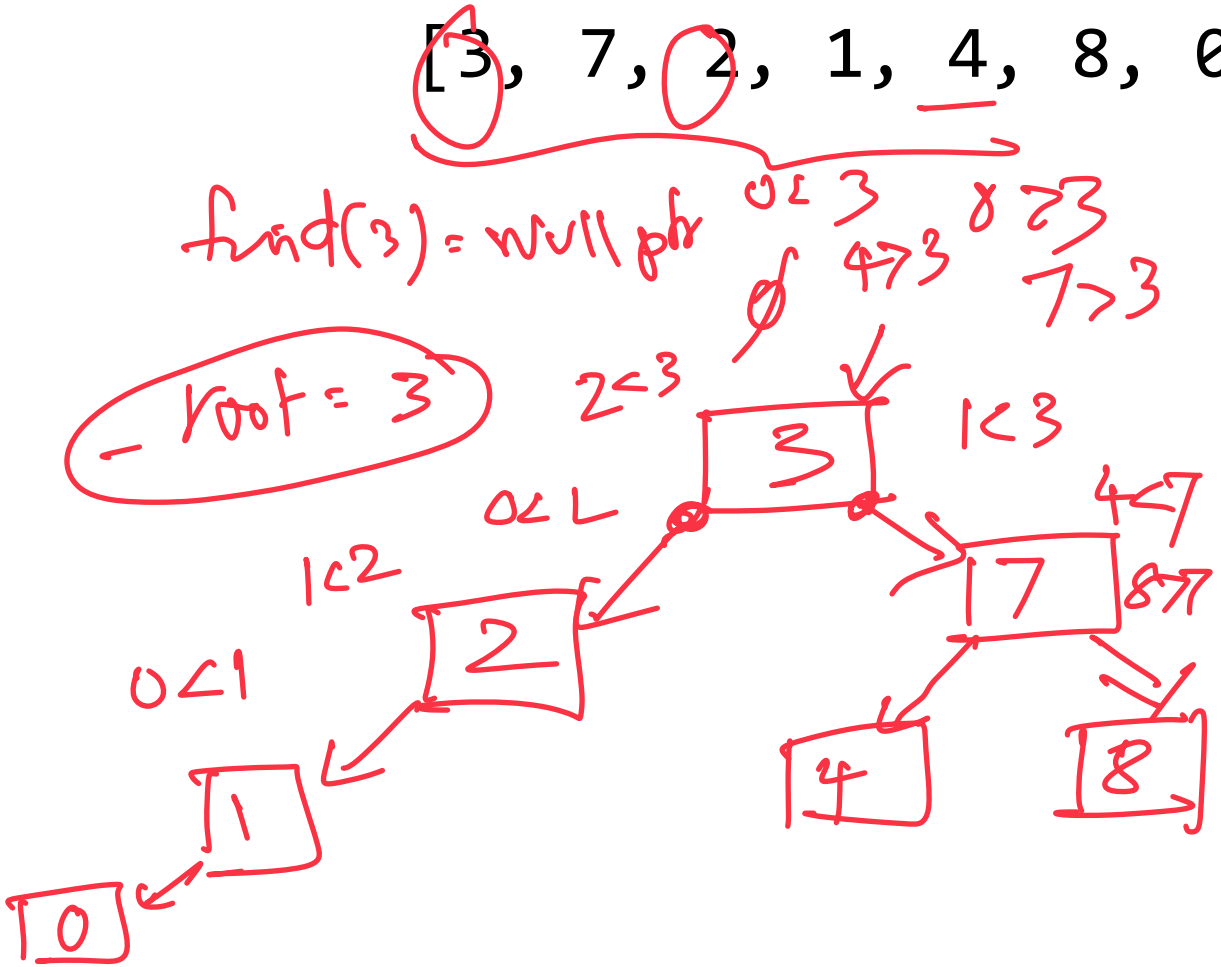
BST Insert

What binary tree would be formed by inserting the following sequence of integers:

`[7, 4, 3, 8, 2, 1, 0]`

`[3, 7, 2, 1, 4, 8, 0]`



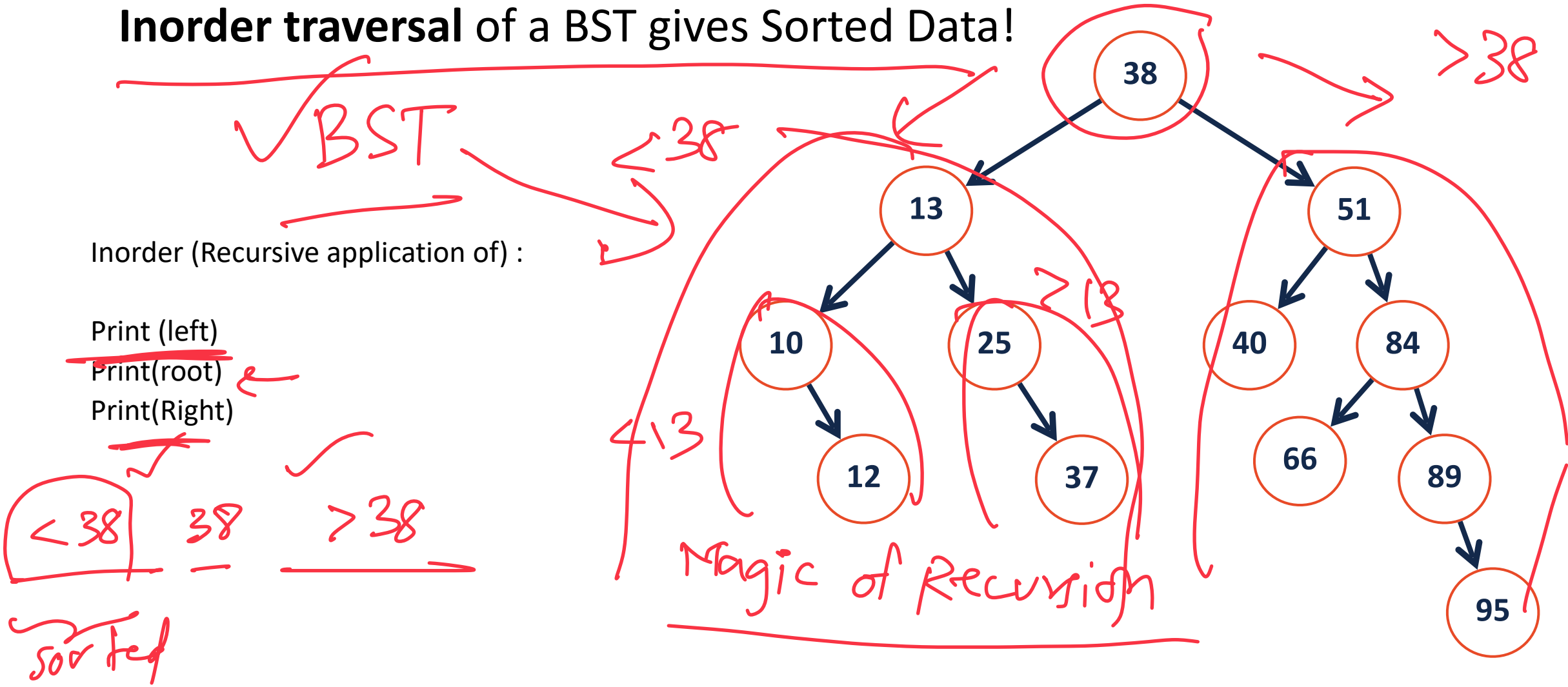


Sorted Data from BST

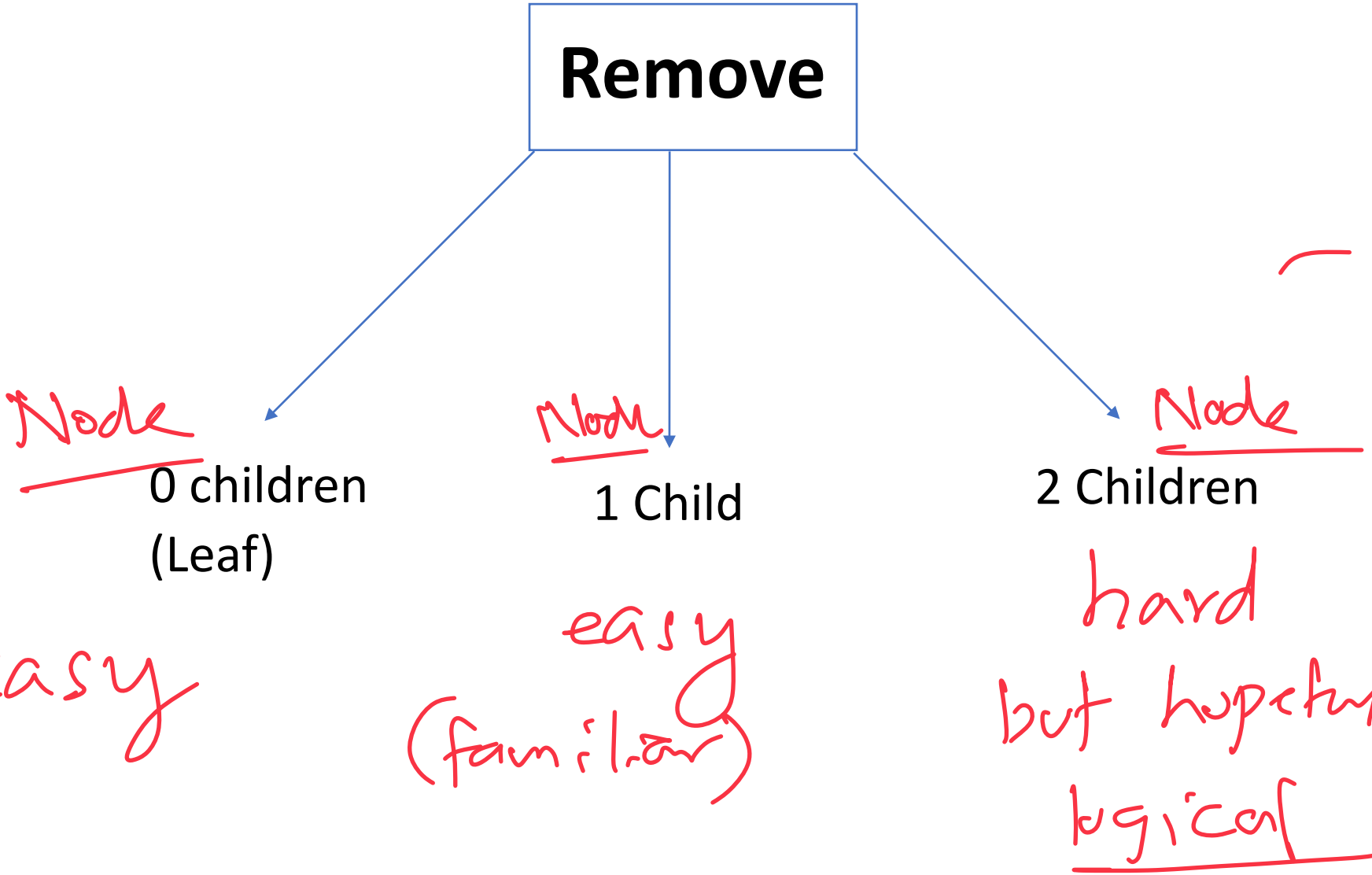
Inorder traversal of a BST gives Sorted Data!

Inorder (Recursive application of) :

Print (left)
Print(root)
Print(Right)

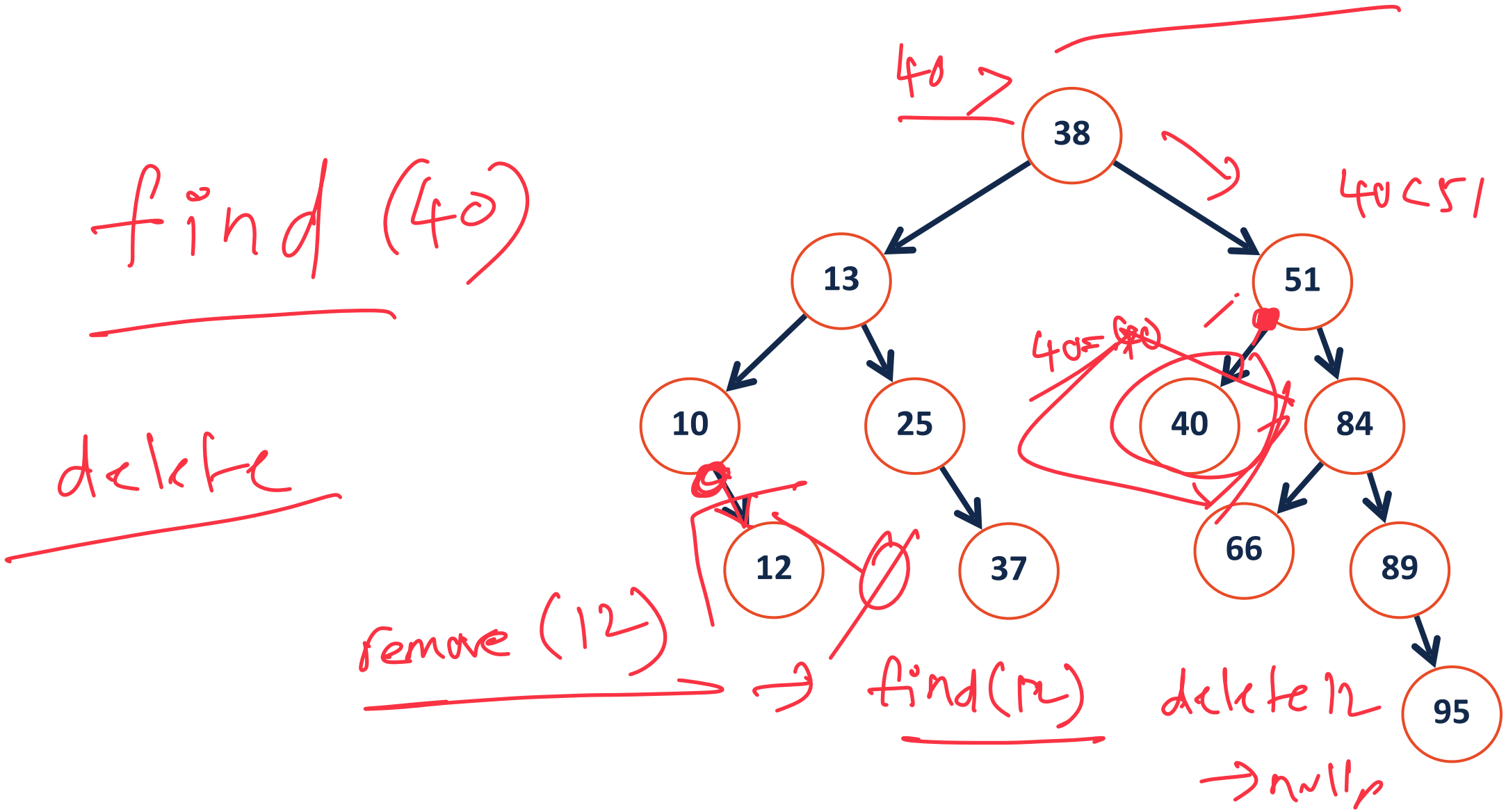


BST - Remove (cases)



BST Remove

remove(40)

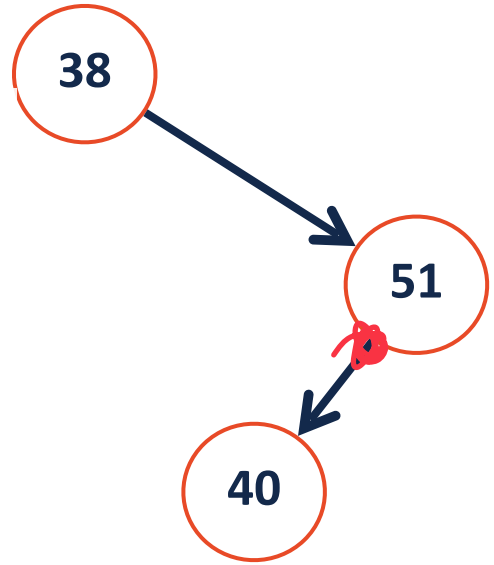


BST Remove

`remove(40)`

0-Child Case

```
TreeNode *& t = _find(root, 40);

delete t;

t = nullptr;
```

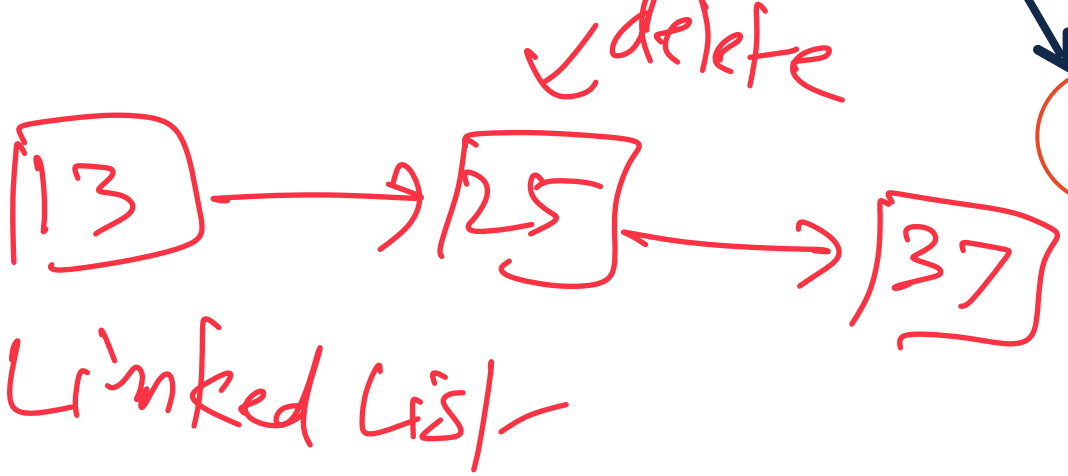


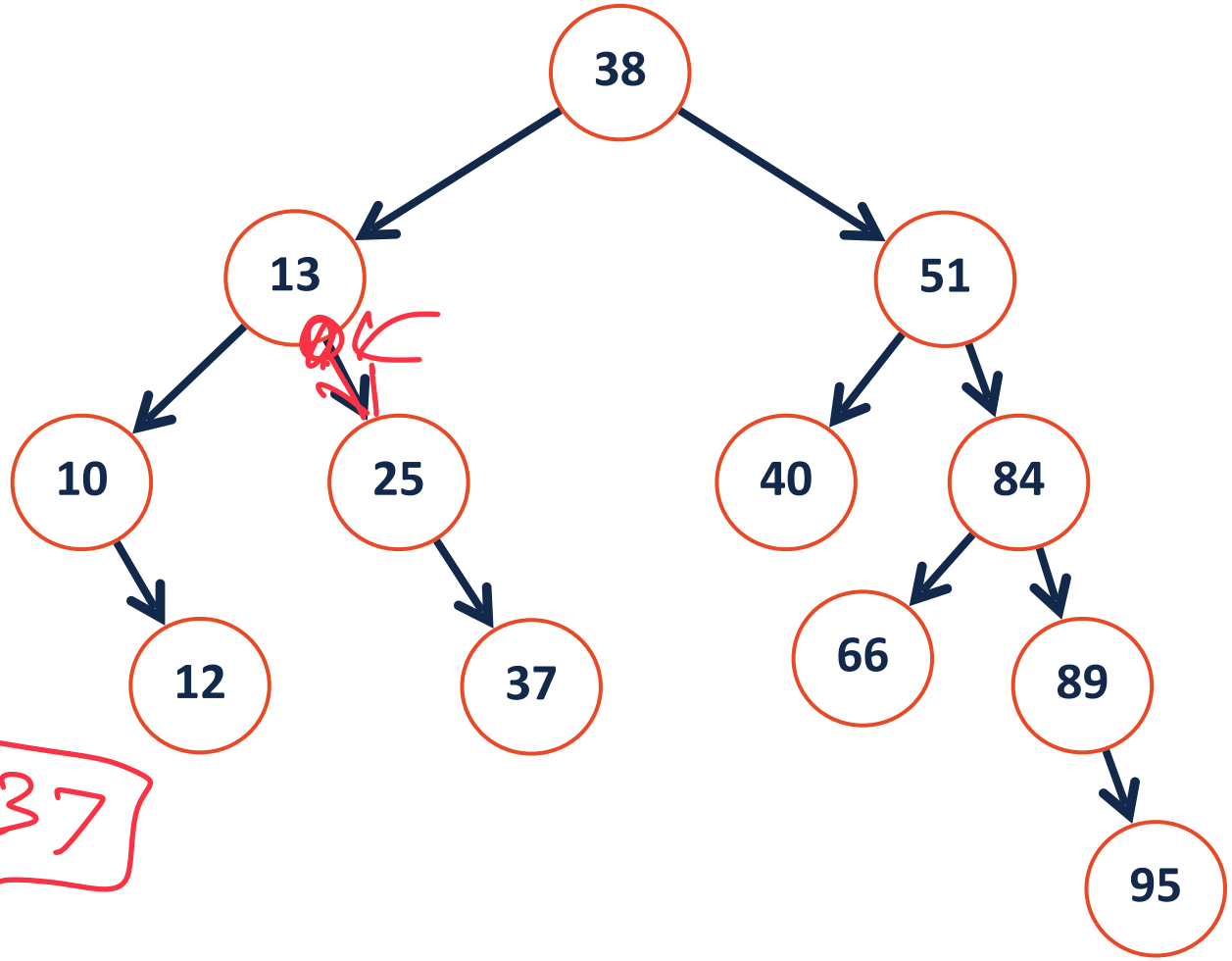
BST Remove

`remove(25)`

① find(25)

25 – leaf



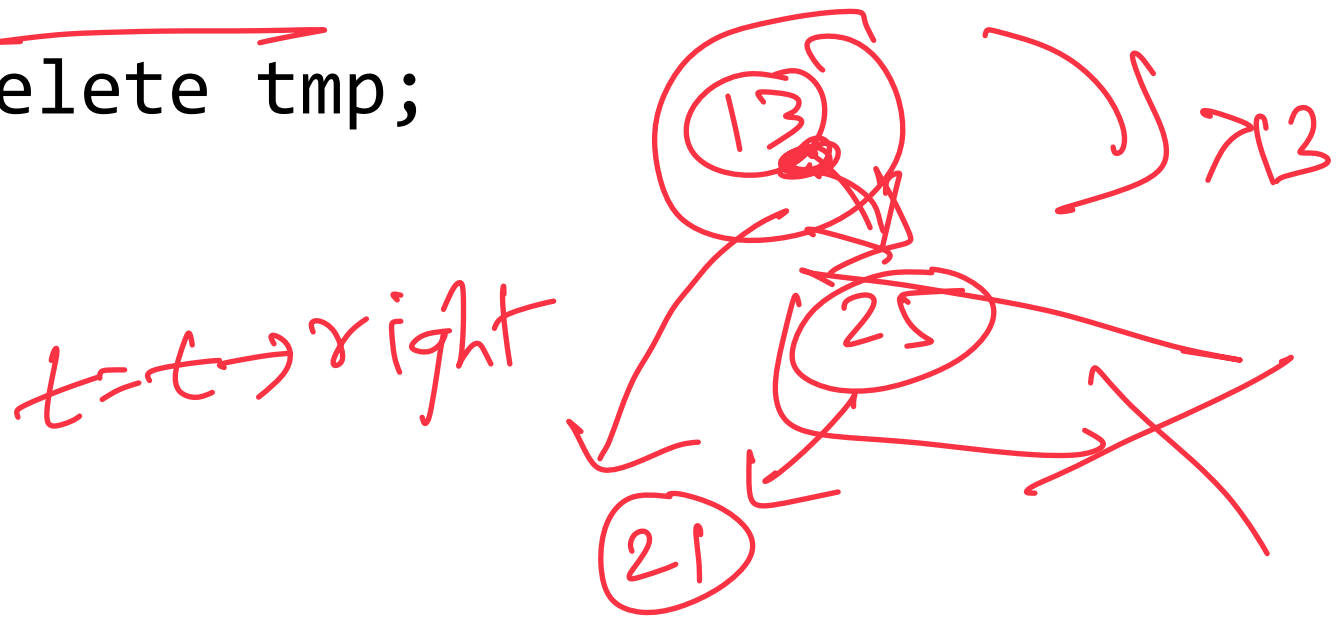


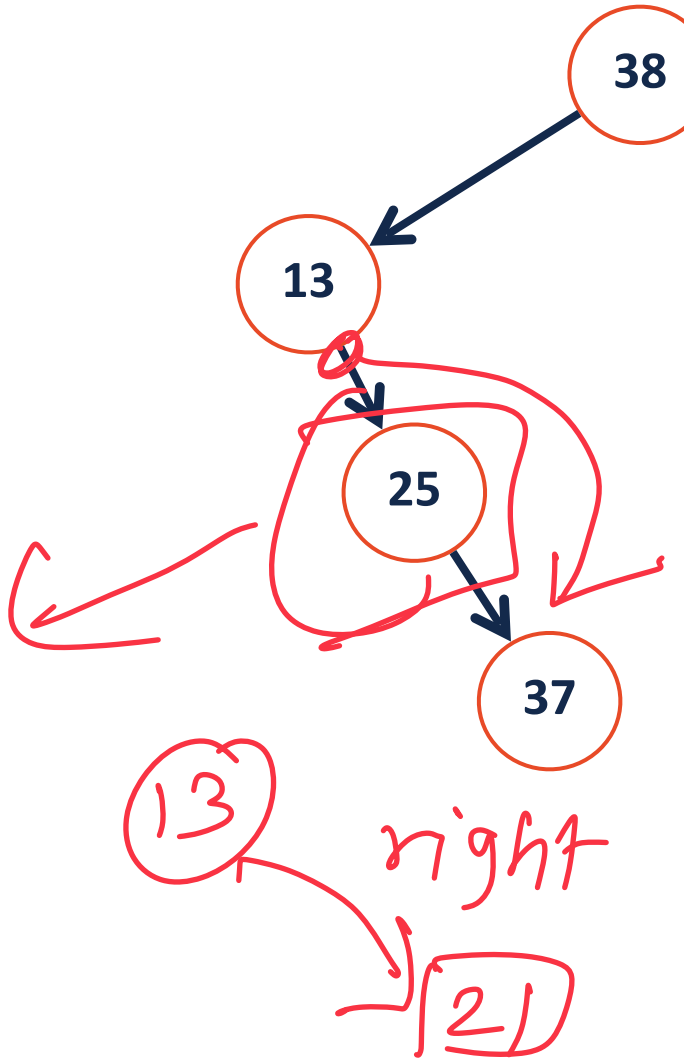
BST Remove

`remove(25)`

1-Child Case

```
TreeNode *& t = _find(root, 25);

TreeNode * tmp = t;

t = t->right;

delete tmp;
```





BST Remove

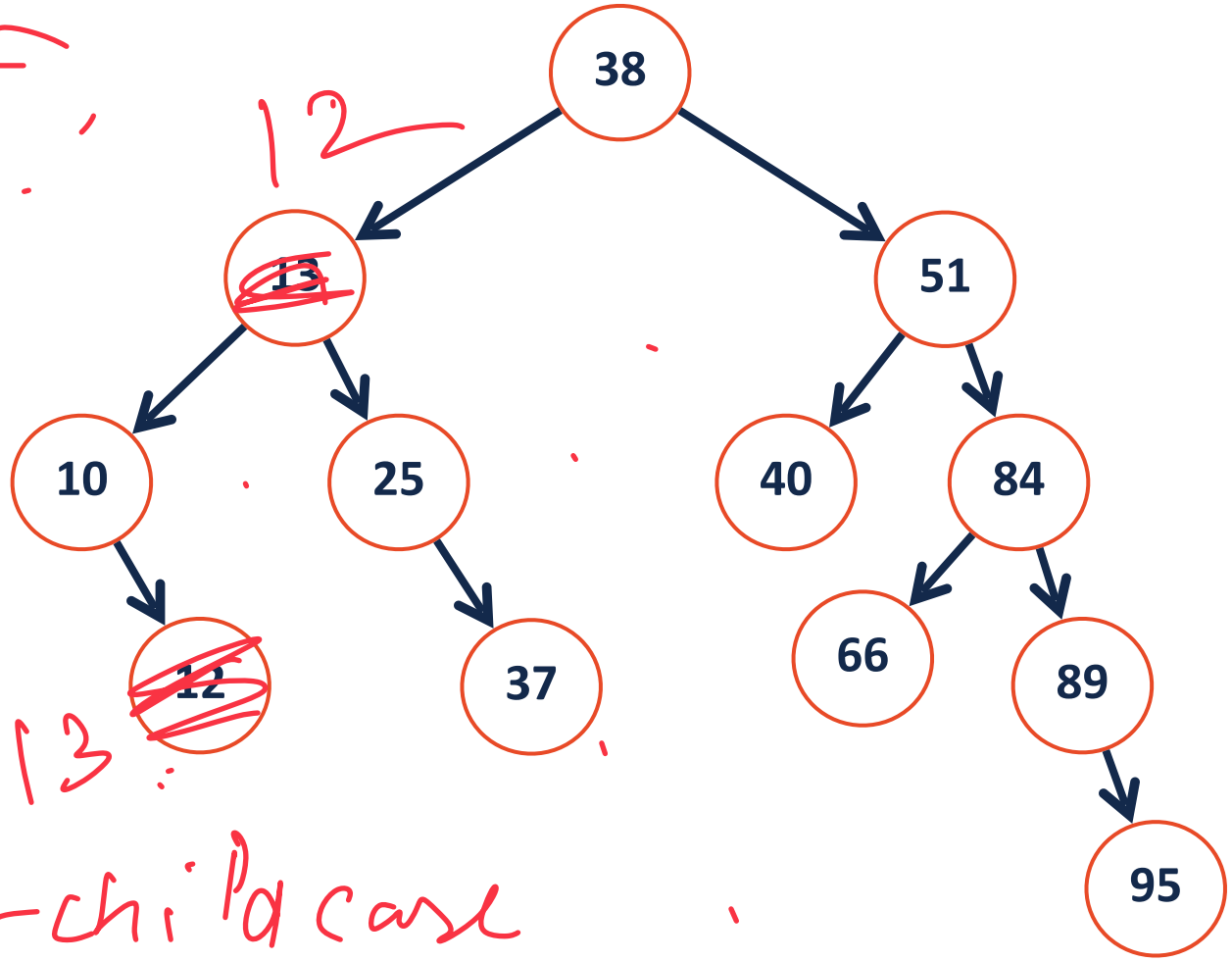
remove(13)

12 = Inorder predecessor of 13

<13 12 13

x

largest element on 13's left



0-child case

BST In-Order

In-Order Predecessor

Rightmost left child

IOP(38) = 37

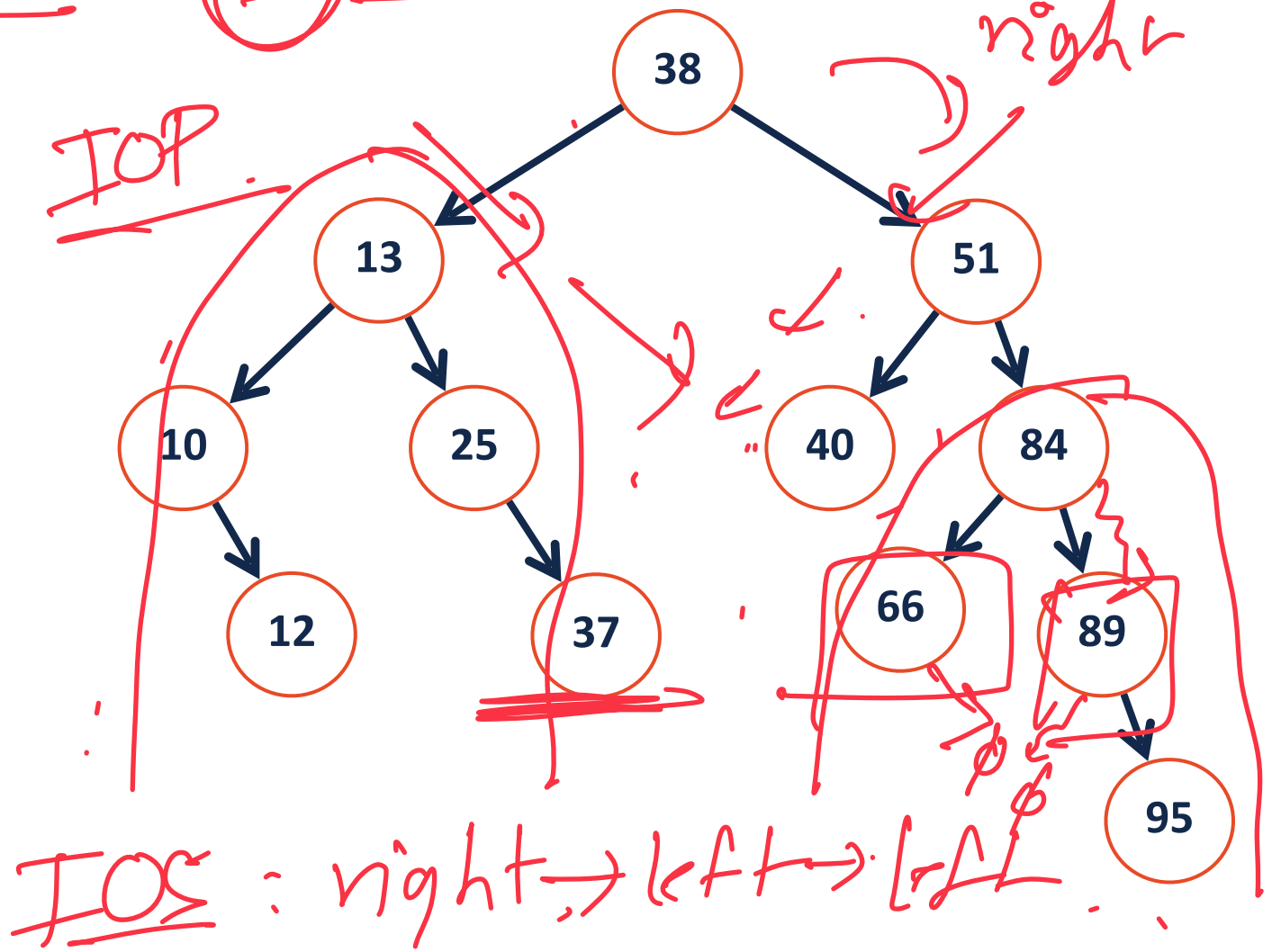
IOP(84) = 66

In-Order Successor

Leftmost right child

IOS(38) = 40

IOS(84) = 89



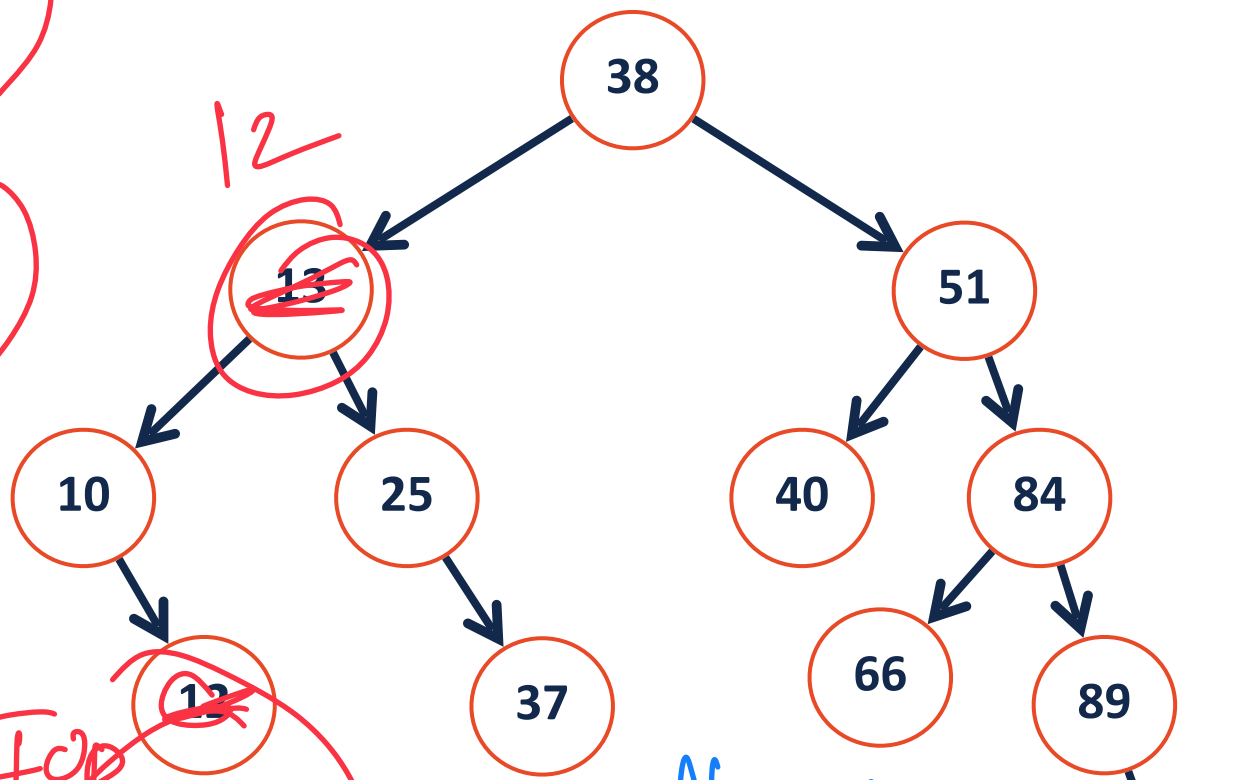
BST Remove - steps

find(13) : O(h)

findIop(13): O(h)

swap 13 ↔ O(1)

remove 13 O(1)



O(h)

12

Iop

13

lchild/Rchild

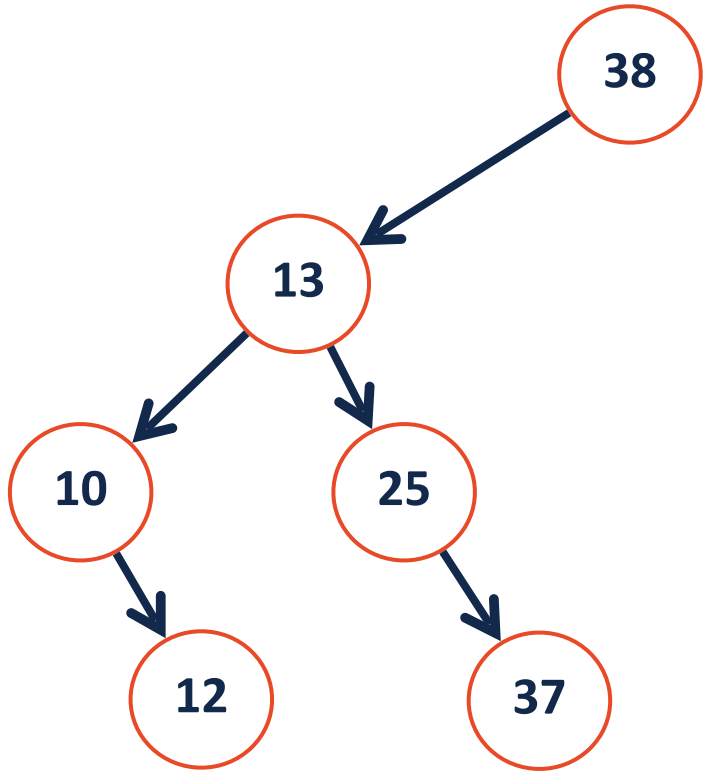
after swap, remove only needs O(1) time

BST Remove

`remove(13)`

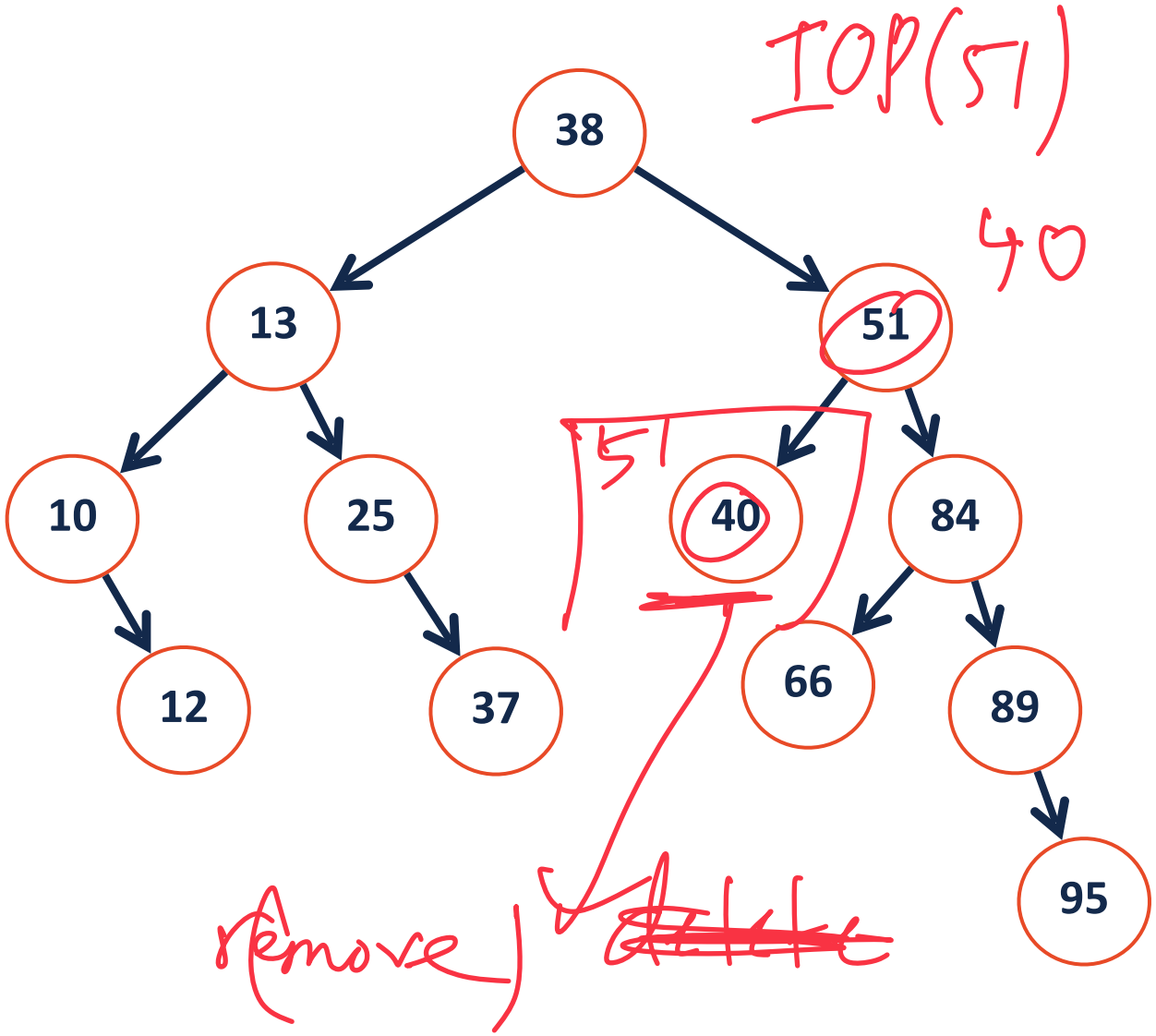
2-Child Case

```
TreeNode *& t = _find(root, 13);

TreeNode * IOP = getIOP(t);

swap(t, iop);

remove(13); //starting from t
```



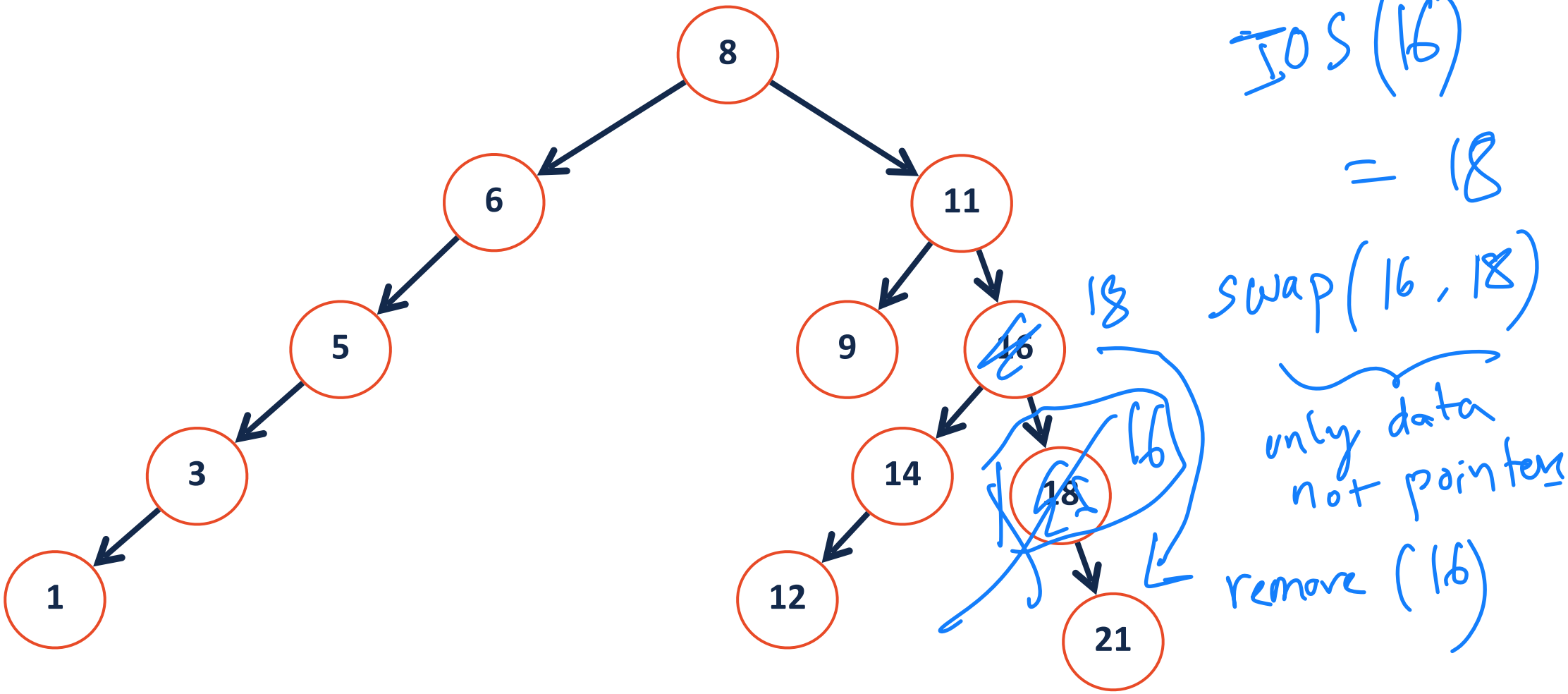
BST Remove

`remove(51)`

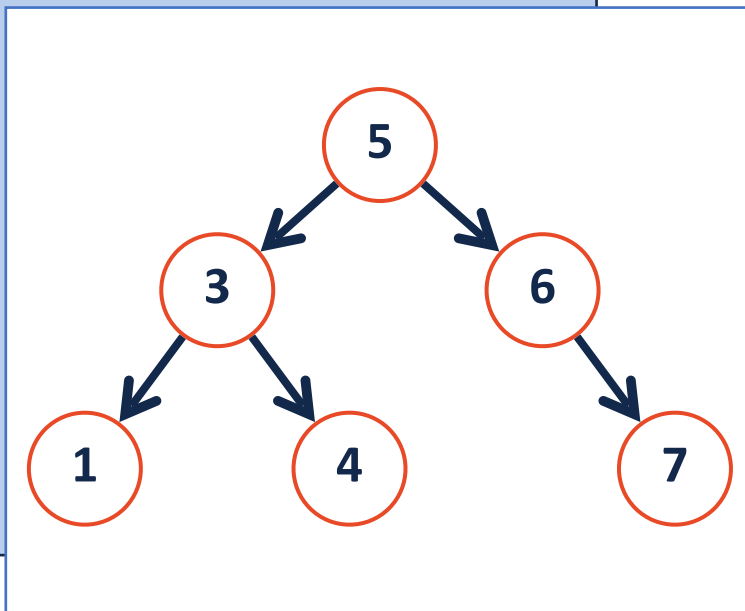


BST Remove

What will the tree structure look like if we remove node 16 using IOS?



```
1  template<typename K, typename V>
2
3  void _remove(TreeNode *& root, const K & key) {
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 }
```



BST Analysis – Running Time

Operation	BST Worst Case
find	**O(h)** Exploits structure of BST - each comparison leads to the next level until the query or NULL is found
insert	**O(h)** Implemented as find and insert - find takes O(h), inserting as a leaf O(1)
remove	**O(h)** Implemented as find and remove - find takes O(h), finding IOS/IOP takes a further O(h), swap O(1) and remove O(1)
traverse	**O(n)** BFS/DFS/Any traversal

Binary Search Trees vs Binary Trees

Operation	Binary Tree	Binary Search Tree
Insert	O(1)	O(h)
Remove	O(n)	O(h)
Traverse	O(n)	O(n)
Find	O(n)	O(h)

Binary Search Trees vs Binary Trees

Operation	Binary Tree	Binary Search Tree	BST Best-Case h = O(log n) Complete BT	BST Worst Case h = O(n) Unbalanced/Skewed BT
Insert	O(1)	O(h)	O(log n)	O(n)
Remove	O(n)	O(h)	O(log n)	O(n)
Traverse	O(n)	O(n)	O(n)	O(n)
Find	O(n)	O(h)	O(log n)	O(n)

Binary Search Tree : Optimal performance

Option A: Correcting bad insert order

The height of a BST depends on the order in which the data was inserted

Insert Order: [1, 3, 2, 4, 5, 6, 7]

Insert Order: [4, 2, 3, 6, 7, 1, 5]