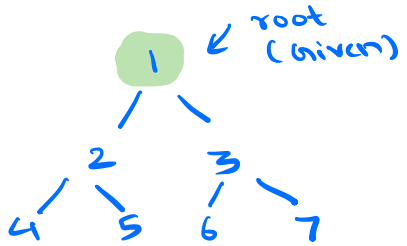
1. Path to Given Node

Given a Binary Tree A containing N nodes, find path from Root to a given node B.

Note- Assume B always exists in A and nodes have unique data values.

A =



B = 5

Output → [1 2 5]

root B

How to find if B exists in A?

Traverse all the nodes → order traversal

Parent → Child

Preorder Traversal

A =

res [ ] B=5

1's Left

- curr → 2

cur.val == B (1 ≠ 5)

add cur.val to res

res = [1]

2 ≠ 5

add 2 to res

res = [1 2]

2's Left

- curr → 4

4 ≠ 5

add 4 to res

res = [1 2 4]

- 4's Left

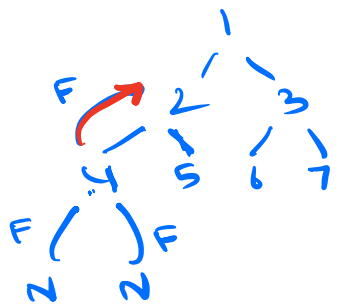
curr = null

if NULL return false

- 4's Right

curr = null

if NULL return false

- 

Back on 4 (after going to left & right)

res = [1 2 4 ...]

find 5 (not found)

X (Remove 4)

res = [1 2]

return false

2's Right

- curr = 5

res = [1, 2]

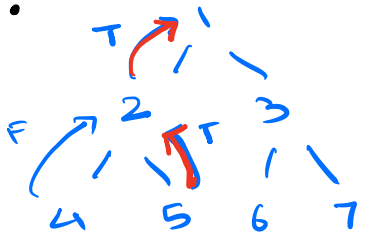
5 = 5 ✓

B = 5

res = [1 2 5]

return true if cur.val = B

Back to 1

- 

curr = 1

hasPath(cur.left) → T ✓

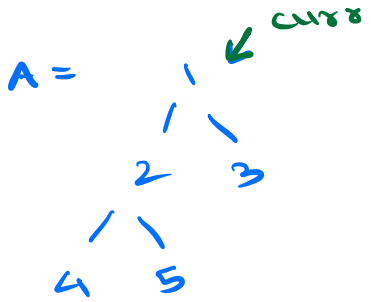
hasPath(cur.right) ✗ don't process

return true;

res = [1 2 5]

(as 1's left found 5, do not go to 1's right)

Ex

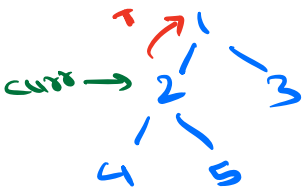


B = 2 res = [ ]

res = [1]

1 ≠ 2

go to left child → true



res = [1 2]

2 = 2

return true

Code

helps know whether B found in root's subtree

```
boolean hasPath(TreeNode root, ArrayList<Integer>
res, int B) {
    if (root == NULL)
        return false;
    res.add(root.val);
    if (root.val == B)
        return true;

    if (hasPath(root.left, res, B) ||
     hasPath(root.right, res, B))
    return true;

    res.remove(res.size() - 1);
    return false;
}
```

only go to right subtree, if B not found in left subtree

B not = root & not there in root's subtree

```
public ArrayList<Integer> solve (TreeNode A,
                                 int B)
{
    ArrayList<Integer> res = new
                        ArrayList<Integer>();

    hasPath(A,res,B);
    return res;
}
```

TC O(N) → go through entire tree

SC O(N)
→ function call stack for recursion

- 6 10 8 7 9 2 1 — N elements

Linear search O(N)

Binary search — Sort — O(N log N)
— Apply search — O(log N)

O(log N) → O(N log N)

Q. Maintain a sorted arr

6

10

① Find pos
② Insert

arr

6

0	1
6	10

A= 6 7 8 9 10
0 1 2 3 4

Insert 2 → 0 idx

- Search for idx takes O(log N)

A= 2 6 7 8 9 10
0 1 2 3 4 5

→ shift elements

O(N)

Total time to maintain a sorted arr → O(N) (log N + N)

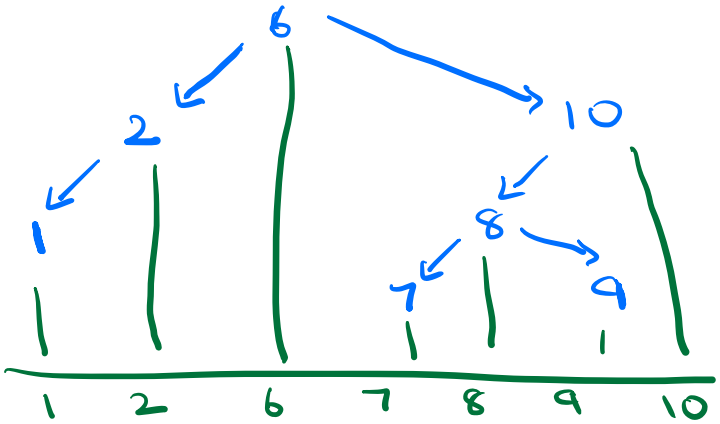
as good as linear search O(N)

- h → 2 → 6 → 7 → 8 → 9 → 10 (using a linked list)

Insert	Search
O(1)	O(N)

Ex 6 10 8 7 9 2 1 Search O(log N) Insertion O(1)

• 6



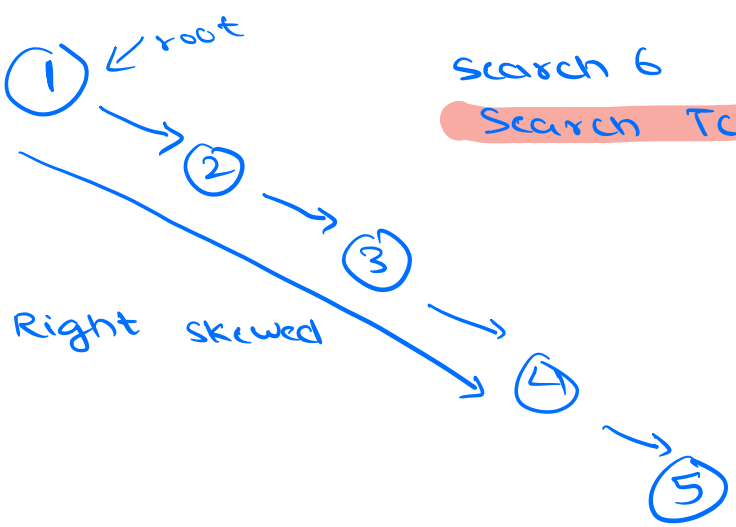
BST

Inorder → sorted arr

Node
Node.left < Node
Node.right > Node

Node.left < Node < Node.right

Search in BST → TC $O(\log_2 N)$



Search 6

Search TC O(N)

Input Stream
1
2
3
4
5

Insertion in BST — Search $O(\log_2 N)$
— Insert $O(1)$

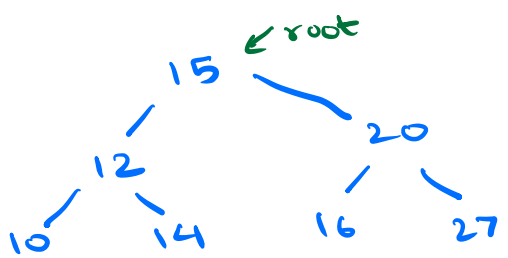
$O(\log_2 N)$

Worst case → $O(N)$

2. BST nodes in a range

Given a binary search tree of integers. You are given a range B and C. Return the count of the no. of nodes that lie in the given range.

A =



B = 1 C = 100

B = 13, C = 20

count of nodes >= 13 and <= 20

Output → 4 (14, 15, 16, 20)

1st approach

a. Inorder traversal → (store arr) $O(N)$

10 12 | 14 15 16 20 | 27

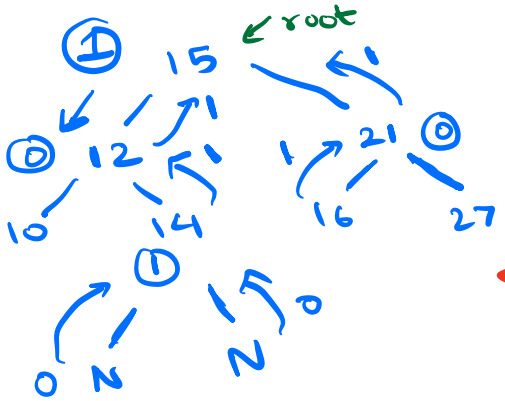
13 4 20

b. Finding idx of B and C $O(\log_2 N)$

c. return $idx_C - idx_B - 1$

TC → $O(N)$

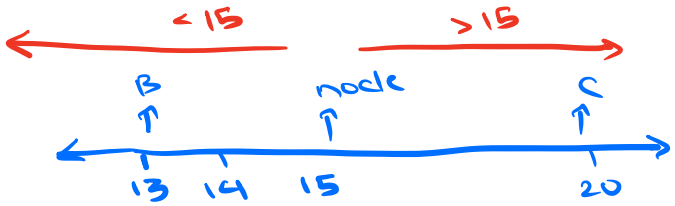
SC → $O(N)$ → Extra arr we create to store inorder traversal



curr → 15

B = 13, C = 20

cnt = 0



13 <= 15 <= 20

cnt = 1 + no of nodes in left + right

If B <= node.val <= C

then go to left and right

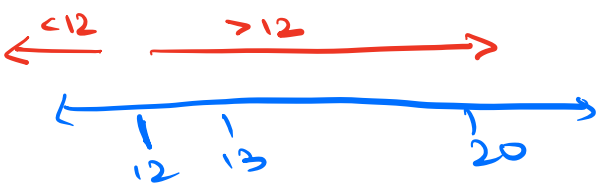
15

cnt = 1 + 1 + right

14 16

= 1 + 1 + 1

= 3 → final

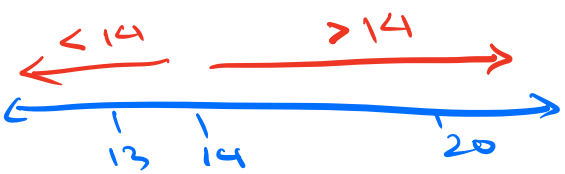


- curr → 12 (15's left)

if node.val < B

cnt = 0 + no of nodes right

go to right



- curr → 14 (12's right)

13 <= 14 <= 20

cnt = 1 + no of nodes left + right

go to left & right

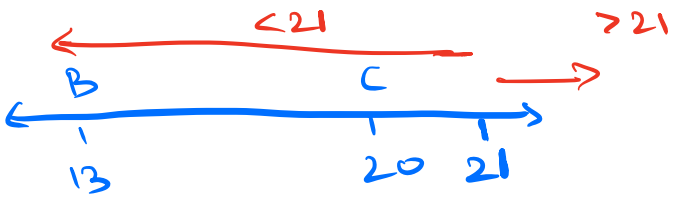
cnt = 1 + 0 + 0

= 1

- curr = null
(14's left & right)

if node = NULL return 0

- curr = 21
(15's right)



* We've traversed 15's left

20 < 21 C < node.val

cnt = 0 + nodes on left of 21

C < node.val go to left

- curr = 16

13 <= 16 <= 20

cnt = 1 + go to left + right

= 1 + 0 + 0 = 1

return 1

Code

```
public int cNodes(TreeNode root, int B, int C){
    if(root == NULL)
        return 0;
    if(root.val < B)
        return cNodes(root.right, B, C)
    else if(root.val > C)
        return cNodes(root.left, B, C)
    else if(root.val >= B && root.val <= C)
        return 1 + cNodes(root.left, B, C) +
                   cNodes(root.right, B, C)
    return 0;
}
```

TC O(N)
SC O(N)
↓
worst case

TC O(h) → log N to N
SC O(h) → log N to N

h → height of BST

h → log n
h → n (skewed)

3. Flatten Nested List Iterator

You are given a nested list of integers nestedList. Each element is either an integer or a list whose elements may be integers or other lists. Implement an iterator to flatten it.

- NestedIterator(List nestedList) initializes iterator with nestedList.
- int next() → returns next integer in nestedList.
- boolean hasNext() → returns true if some integers left in nestedList, otherwise false.

NestedList — int
NestedList

Iterator → obj which helps us go to each element

Ex A = [1, 2, 3, 4, 5, 6]

```
for (idx = 0 ; idx < N ; idx++) {
    print (A.get(idx))
}
```

- int idx = 0

next() return 1 // A[0] A[idx] idx++
next() return 2 // A[1]
next() return 3

↓

print cur ele, move to next

boolean hasNext()

↓

whether more elements in arr

A = [1, 4, 6]
idx

hasNext() → true

< if (idx < N)
return true;
else
return false;
>

→ Nested List

Ex A = [1, [2,3, [4,5]], 6]
0 1 2
Int Int

Output B = [1, 2, 3, 4, 5, 6]

Initially B = [ ]

- A[0].isInteger() → T
  ↓ given
  1 (Integer)
  add to B

B [1]

- A[1].isInteger() → F

[2, 3, [4, 5]]

[4, [4, 5]]

```
for (i = 0; i < A[i].list.size; i++)
    <
        for
    >
// need endless for loops
   to process nestedList
```

Given

Code

```
public class NestedIterator {
    ArrayList<Integer> B;
    int idx;
    int N;

NestedIterator (ArrayList<NestedInteger>
                nestedList) {
    ArrayList<Integer> B = new ArrayList<Integer>
                                    ();
     rec (nestedList)
     idx = 0
     N = B.size();
    }

int next() {
    return B.get(idx++);
    }

boolean hasNext() {
    return idx < N;
  }
```

populating B

```
void rec( ArrayList<NestedInteger> nestedList)
{
    for (int i=0 ; i<nestedList.size() ; i++) {

        if (nestedList.get(i).isInteger())
            B.add (nestedList.get(i).
                   getInteger())
        else
            rec (nestedList.get(i).getList())
    }
}
```

TC O(N) → no of integers in list

SC O(N)
↳ size of B (output arr)