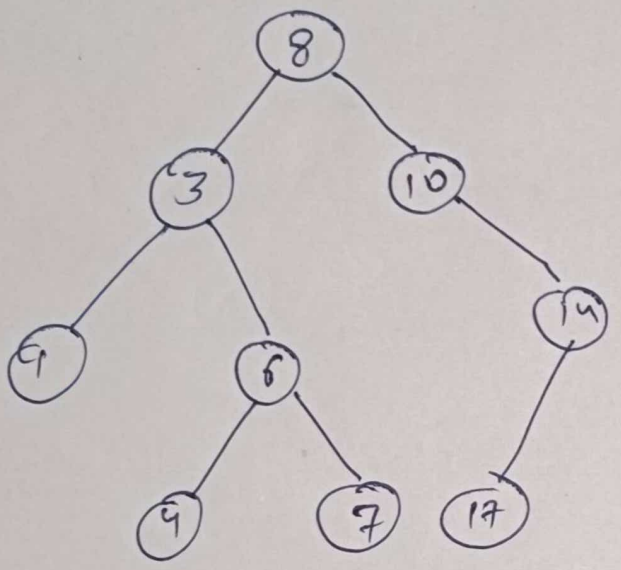
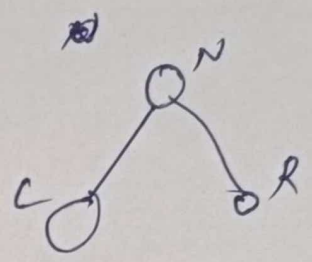
Binary Search Tree





$L < N < R$

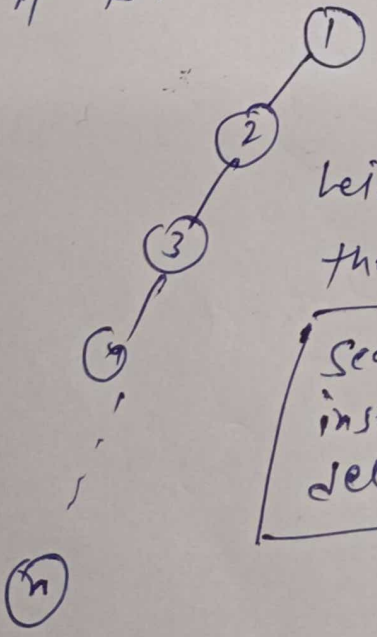
Ideally duplicates are not allowed in BST

if it presents then $L <= N < R$ or $L < N <= R$

choose any one

Why BST over The BT

if BT like



height of BT = N

then

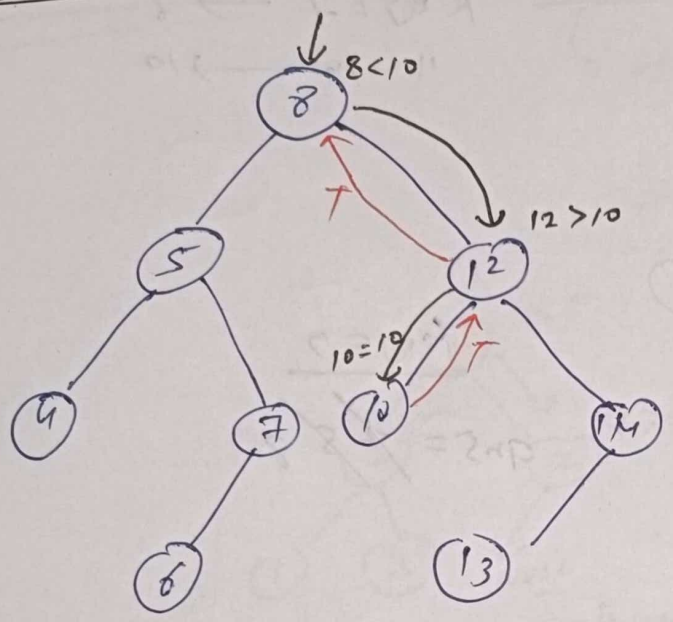
search →	O(N)
insert →	O(N)
delete →	O(N)

But in BST generally not always the height is $\log(N)$

then

search →	O(logN)
insert →	"
delete →	"

Search in BST

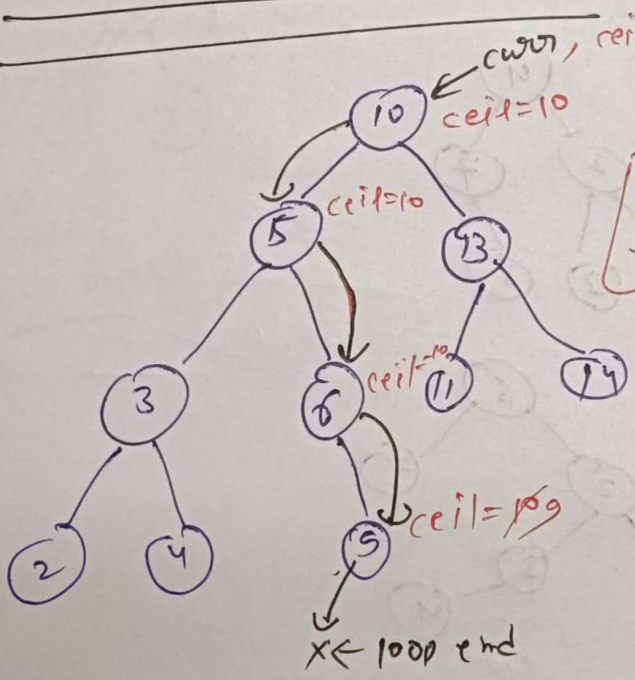


Search = 10

$T.C. = O(\log N)$

$S.C. = O(\log N)$

Ceil in BST



$T.C. = O(\log N)$
$S.C. = O(1)$

key = 8, ans = 9
key = 11, ans = 11
key = 12, ans = 13

Approach

key = 8 ciel = ~~-1~~ ~~10~~ 9 (initially)

1st iteration curr = 10 10 > 8 ciel = 10

curr = 5 (curr → left) & curr = 5 8 7 5

2nd iteration curr = 5 no change in ceil.

curr = 6 (curr → right) curr = 6 6 < 8 no change in ceil

curr = curr → right = 9

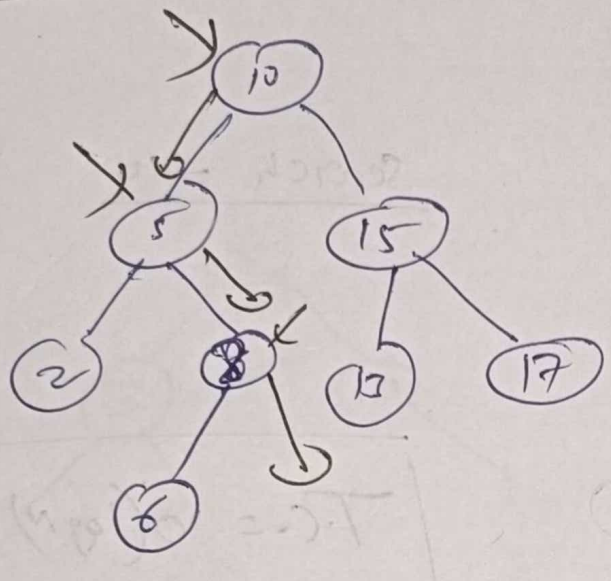
4th iteration curr = 9 9 7 8 ceil = 9

curr = curr → left = NULL

end 1000

3rd iteration

Floor in BST



key = 7 ⟶ 6

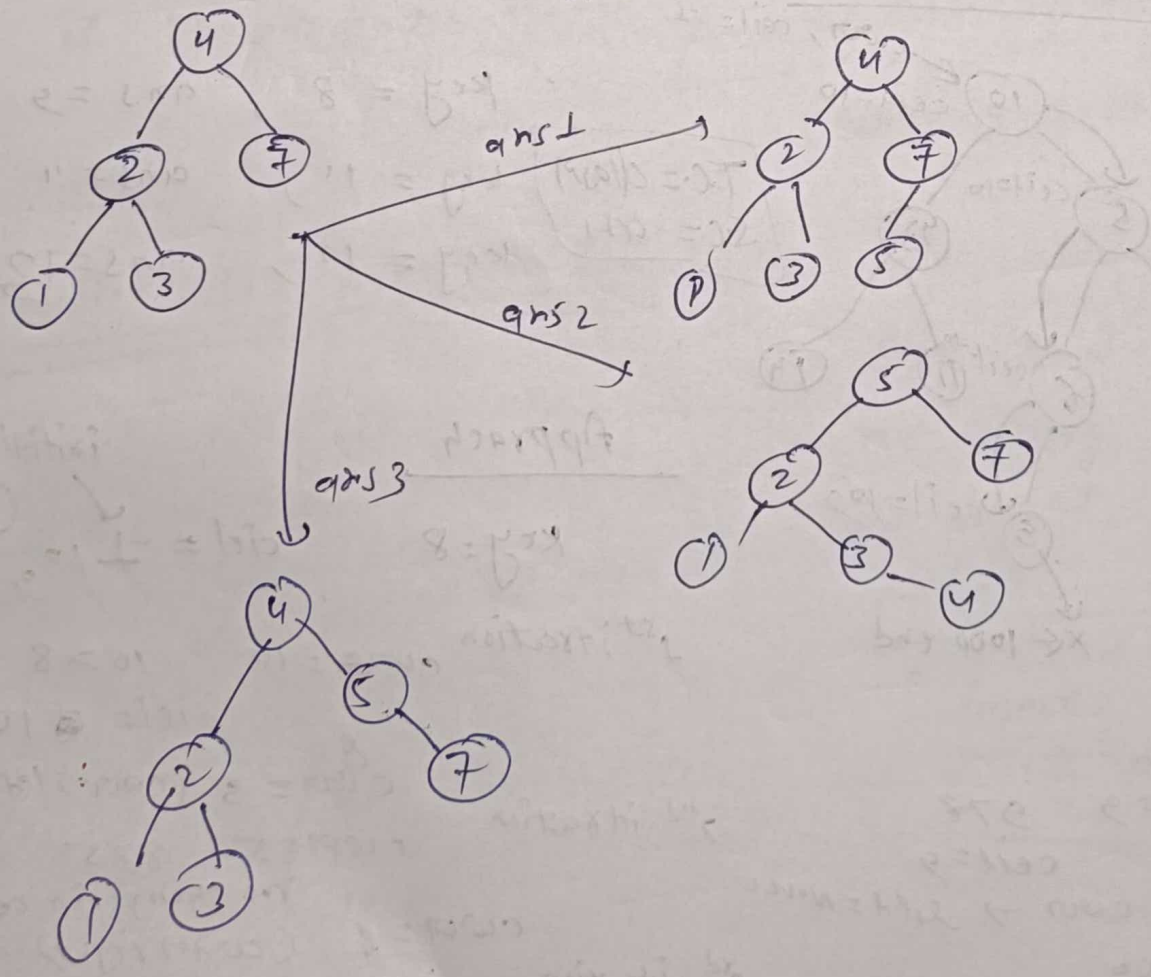
" 14 ⟶ 10

key = 3

ans = ~~1~~ ~~8~~ 8

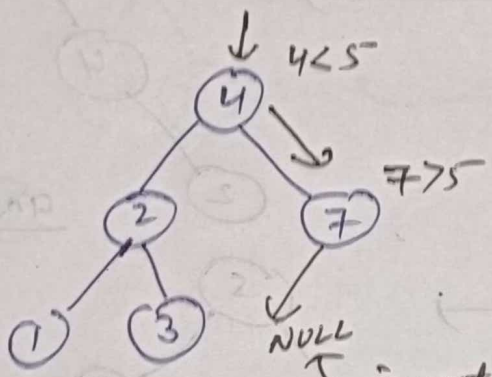
Insert a node in BST

node = 5

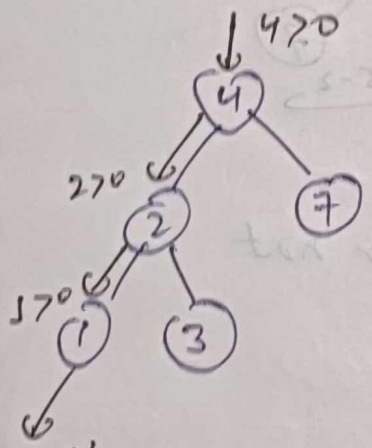


Approach :→ Find where it (node) can be & insert it

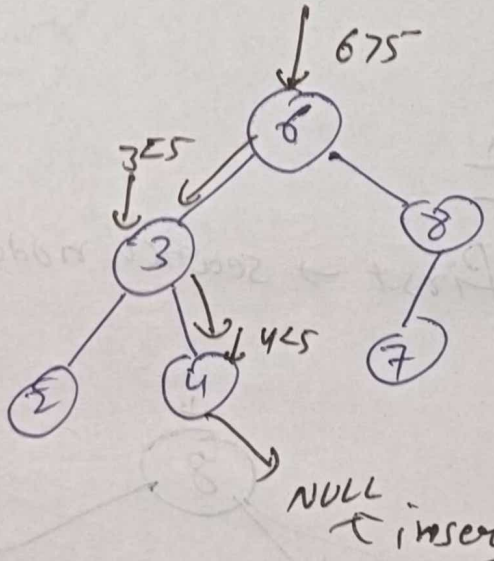
↳ always be a leaf position



node = 5



node = 0

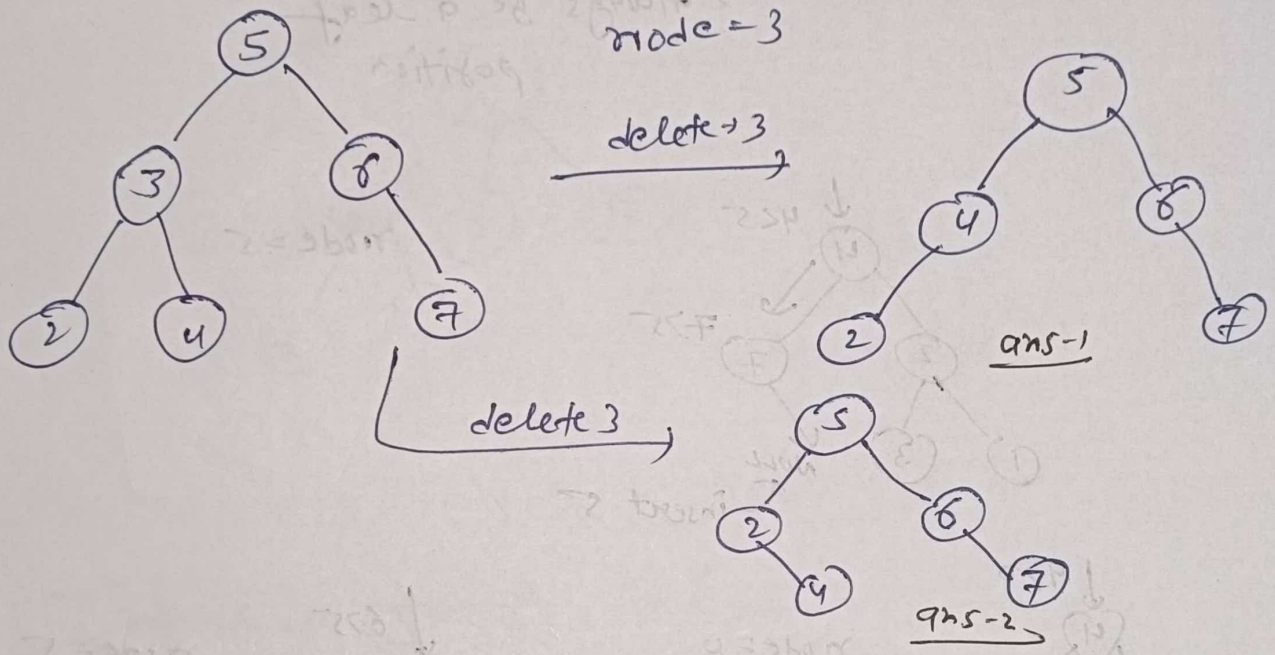


node = 5

T.C. = O(log N)

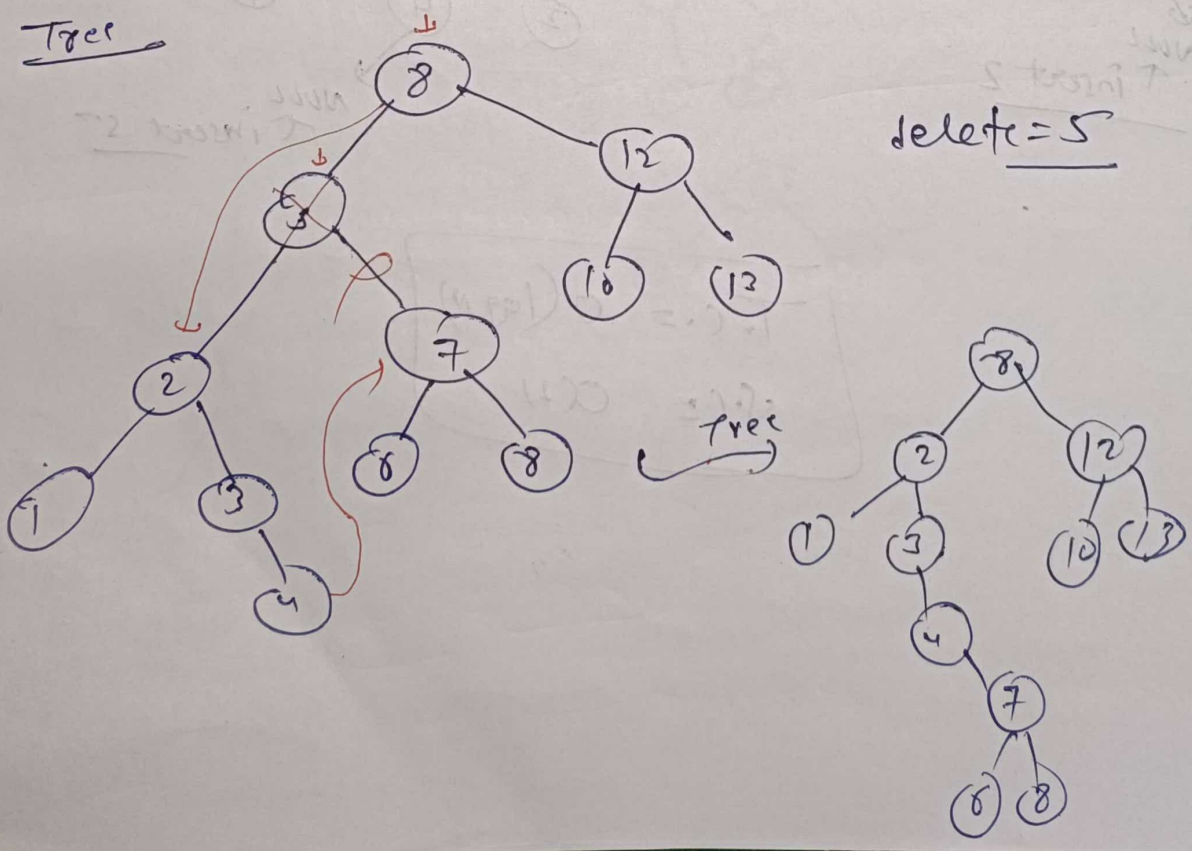
S.C. = O(1)

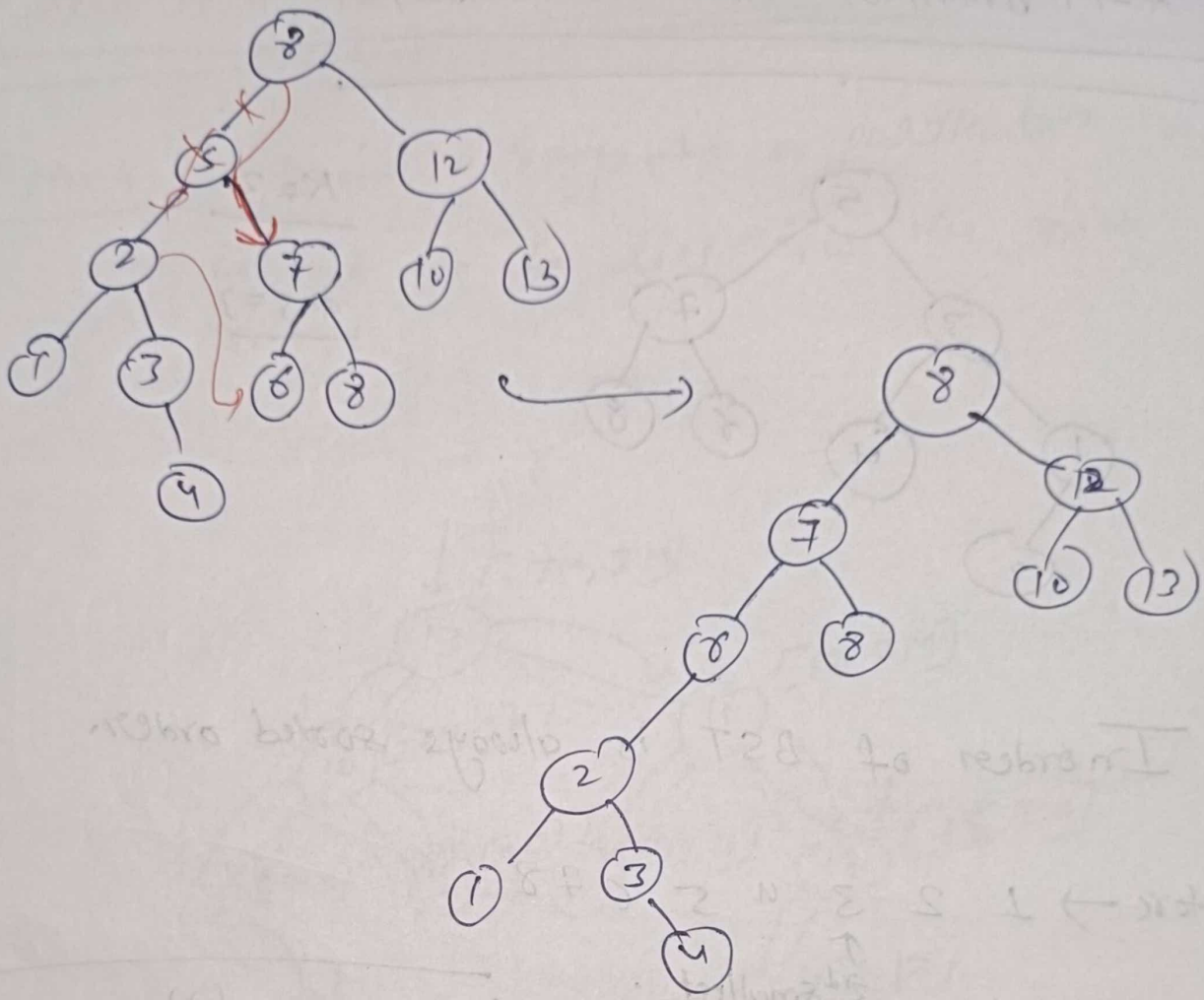
Delete a Node in BST



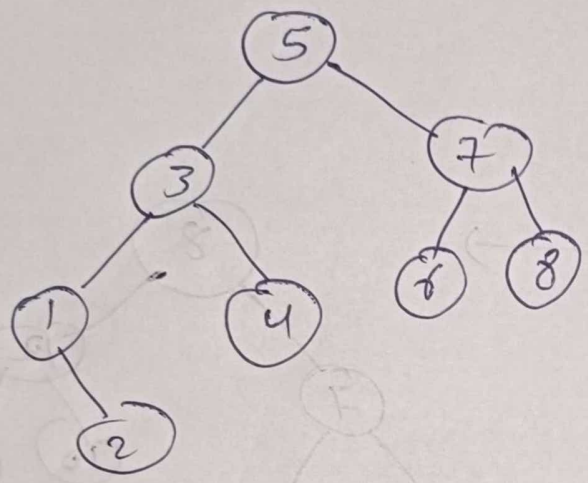
Approch

first → search node if exist or not




8
5
12
2
7
10
13
1
3
6
8
4
8
7
12
10
13
6
8
2
1
3
4

K-th smallest element in BST



K = 3

ans = 3

Inorder of BST is always sorted order.

store → 1 2 3 4 5 6 7 8

↑ 3rd smallest element.

T.C. = O(N)
S.C. = O(N) + O(N)

Other

take a counter = 0

when you visit a node in inorder cnt++

if (cnt == k)

ans = node → val.

T.C. = O(N)
S.C. = O(N)

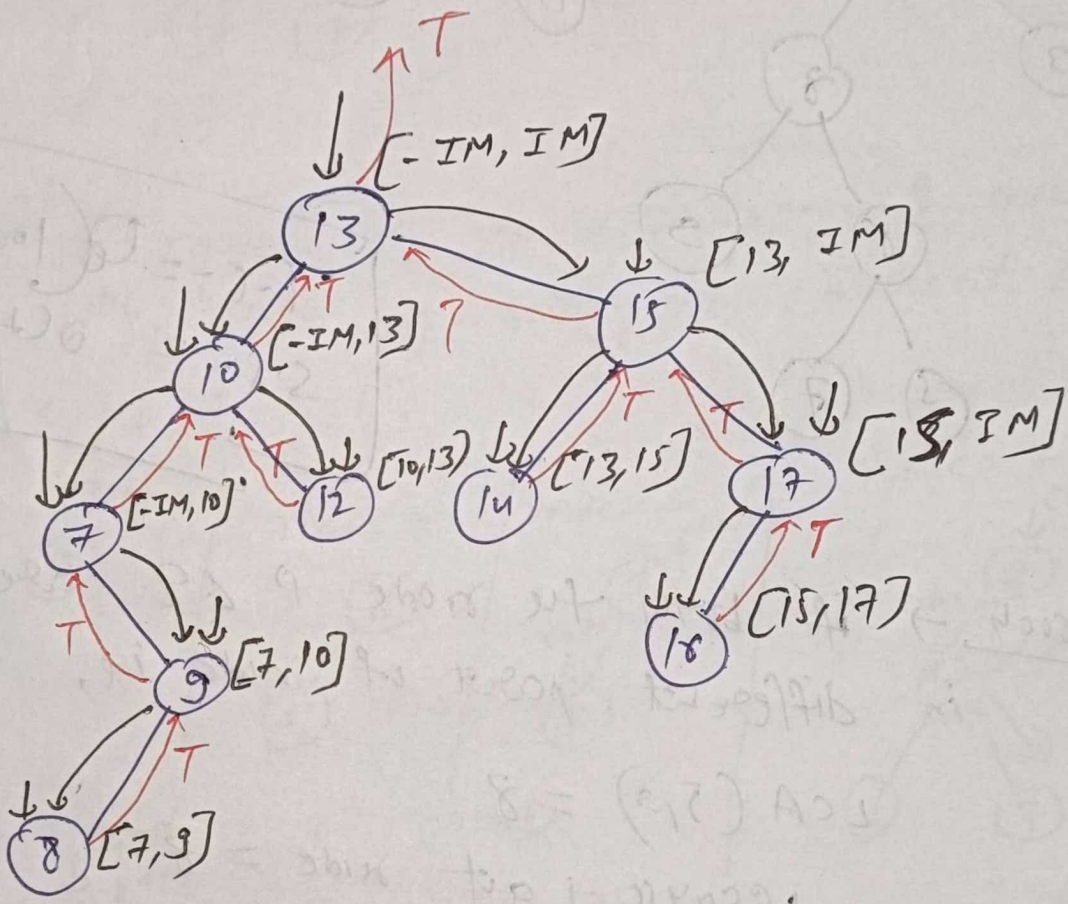
← inorder recursive / iterative

← morris Traversal.

T.C. = O(N)
S.C. = O(1)

Check if a tree is a BST or BT

Approach :- provide a range to a particular node i.e. it can be lies b/w the given range.



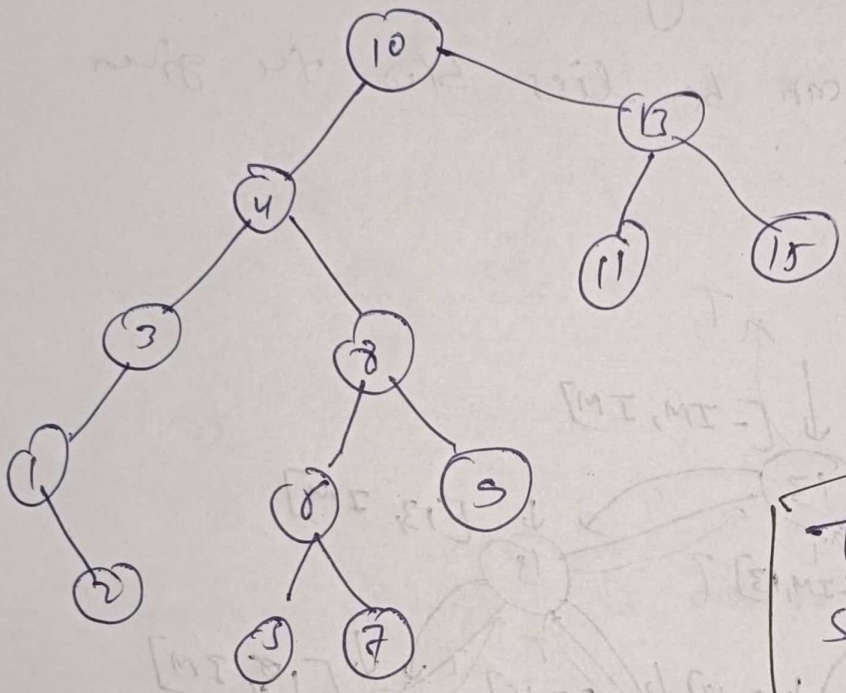
if we go left ⟶ update INT_MAX

if ——— right ⟶ ——— INT_MIN

T.C. = O(N)

S.C. = O(1) ← morris Traversal

LCA in BST



$T.C. = O(\log N)$

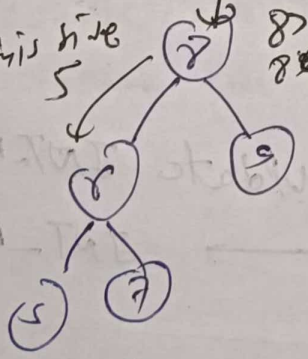
$S.C. = O(1)$

Approach → if both the node P & Q are lies in different part of node i.e.

LCA (5, 9) = 8

because + our node = 8

this side 5

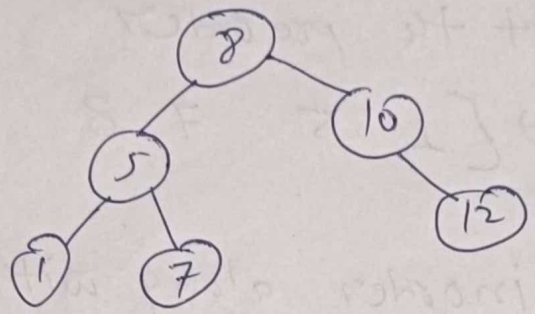


5 and left subtree → 9 on right subtree

the first point of partition

Construct a BST from preorder traversal

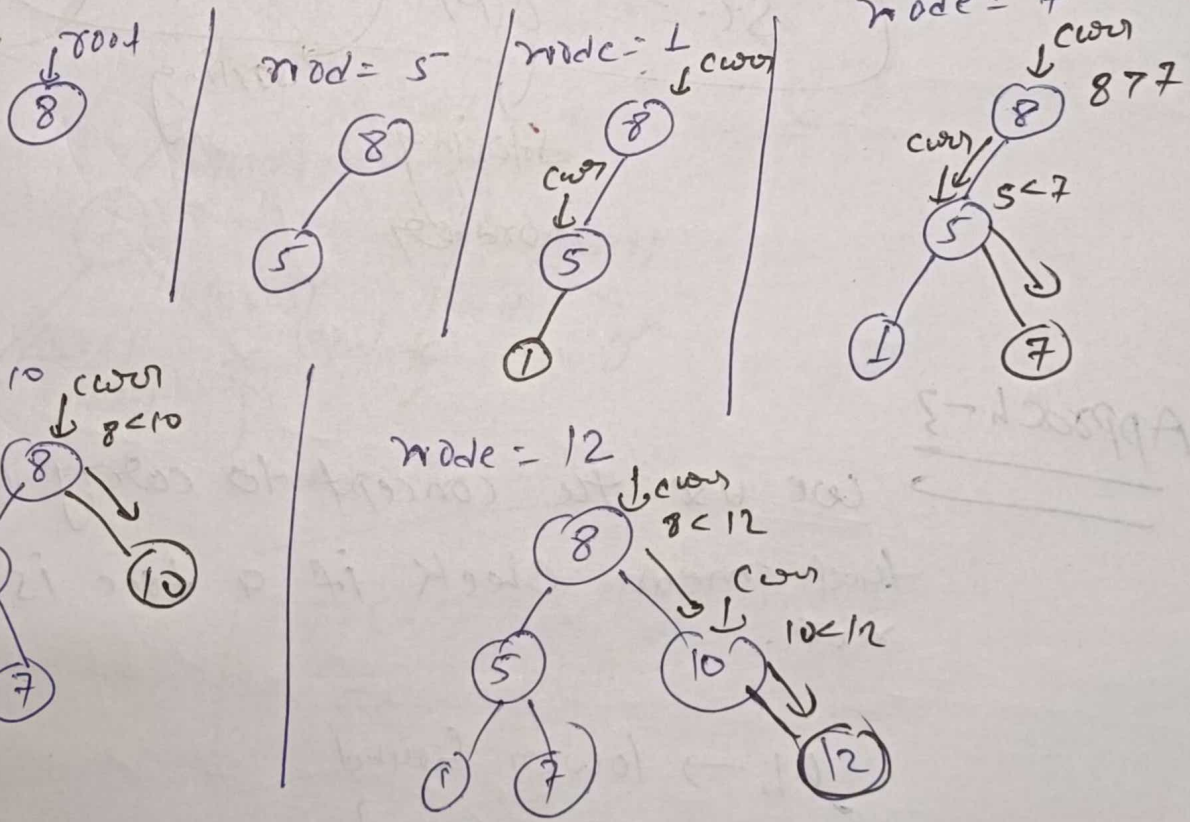
preorder → [8, 5, 1, 7, 10, 12]



Approach-1

Traverse the preorder array & insert the node by getting the correct position

[8, 5, 1, 7, 10, 12]



T.C. = $O(N^2)$

S.C. = $O(1)$

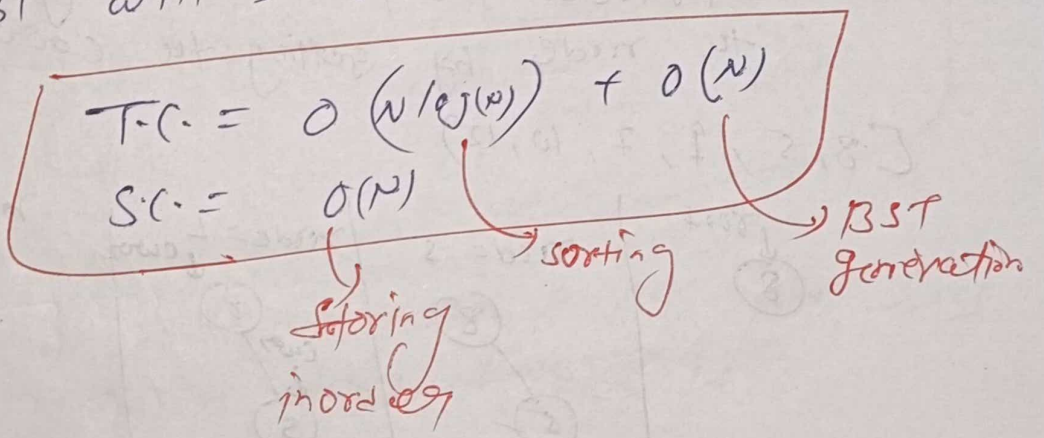
Approch-2

preorder → [8 5 1 7 10 12]

↓ sort the preorder

Inorder → [1 5 7 8 10 12]

Now we have inorder along with preorder & we can generate a unique Binary tree & that unique BT will be our BST



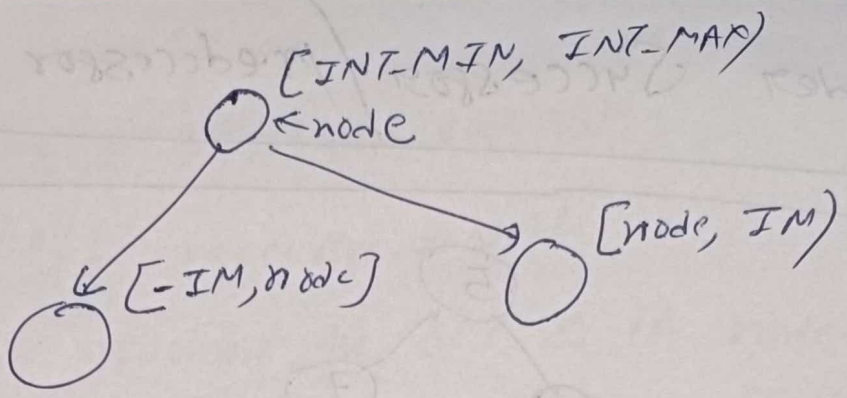
Approch-3

→ we use the concept to carry (lb, ub) that mean check if a tree is BST or not.

lb → lower bound
ub → upper bound

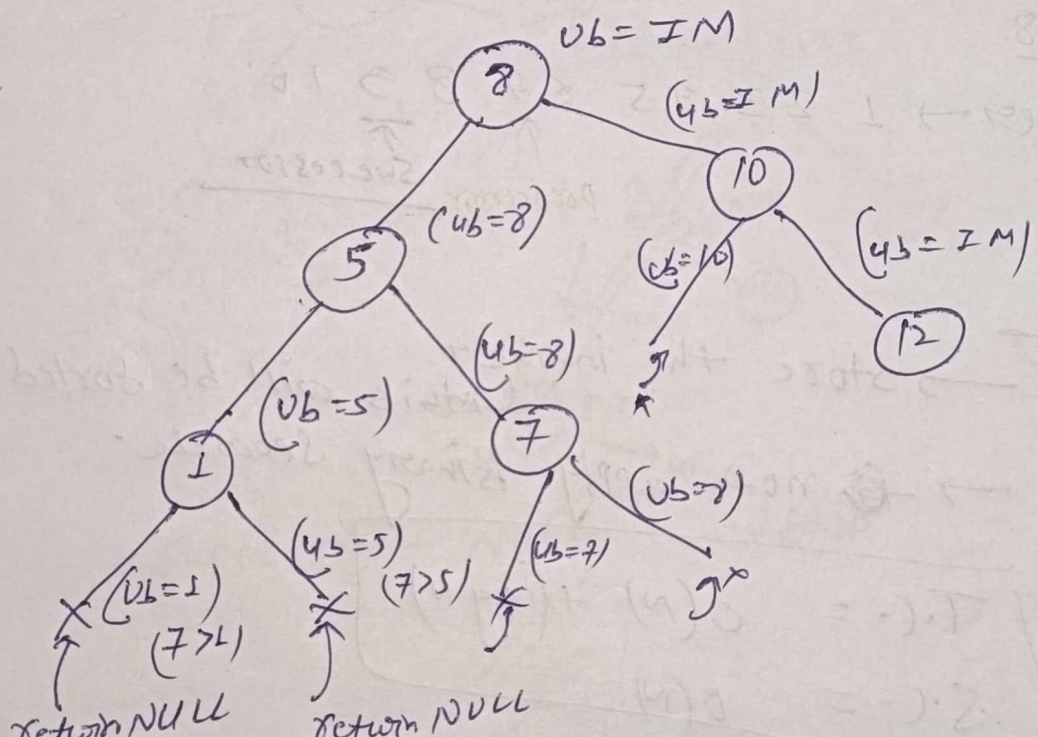
→ Here we only required ub.

is Tree A BST



Sol^{n}

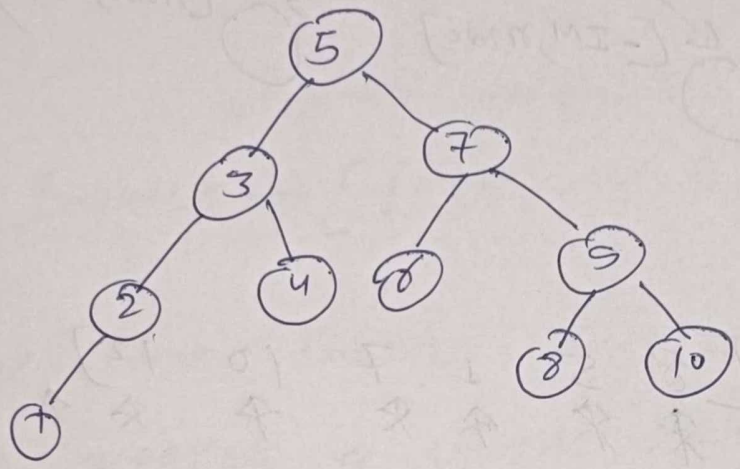
preorder



$T.C. = O(3N) \approx O(N)$

$S.C = O(N)$

Inorder Successor / predecessor



Val = 8

inorder → 1 2 3 4 5 6 7 8 9 10

predecessor ↑ ↑ Successor

Brute force

→ store the inorder (this will be sorted)

→ now apply Binary search.

T.C. = O(N) + (log N)

S.C. = O(N)

Approach - 2

Traverse the BST using inorder & the first value which is > 8 store it & stop bcz this is your answer

T.C. = O(N) → morris Traversal.

S.C. = O(1)

Approch -3

take a successor variable

initially successor = NULL

Now traverse the BST & if node->val > val

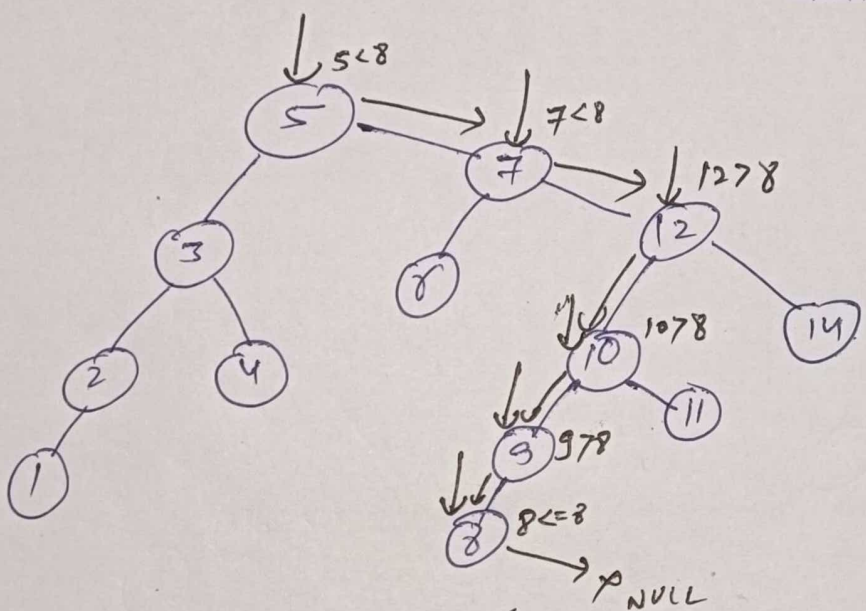
successor = node->val

& go node->left

else go->right

until NULL

ex

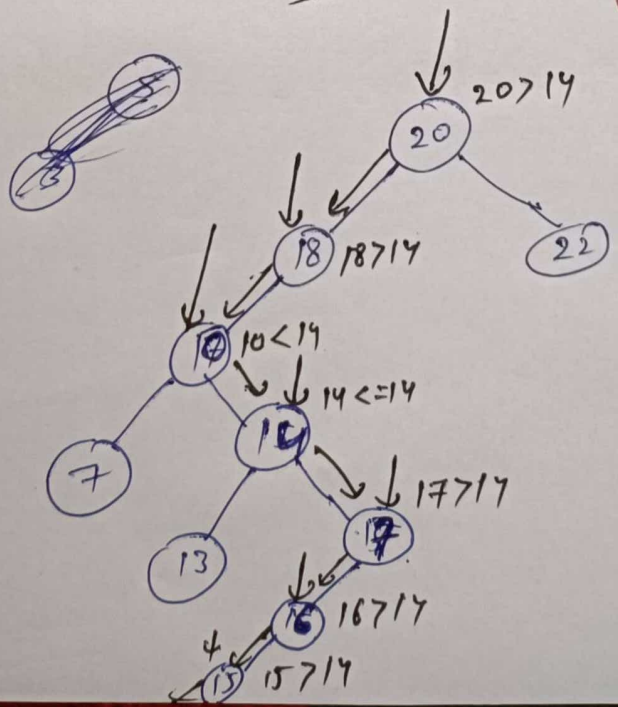


Val = 8

successor = NULL
12
10
9

T.C. = O(H)
S.C. = O(1)

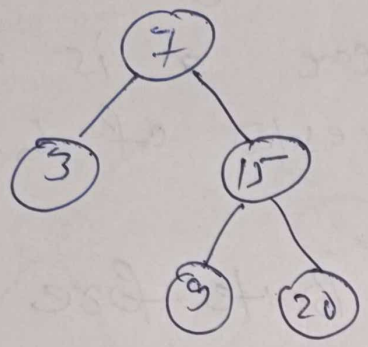
ex



Val = 14

successor = NULL
20
18
17
16
15

BST Iterator



BST Iterator (7)

next → 3

next → 7

has next → True

next → 9

has next → True

next → 15

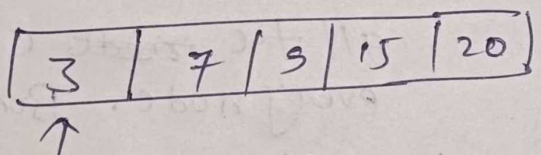
has next → True

next → 20

has next → false

Approach-1

store the inorder



initially pointer

for next & has next()

T.C. = O(1)

S.C. = O(N)

T.C. = O(N) for inorder.

follow up !→ don't store the inorder.

Approach-2

Inorder → left node right

To solve this problem we can borrow the logic of inorder.

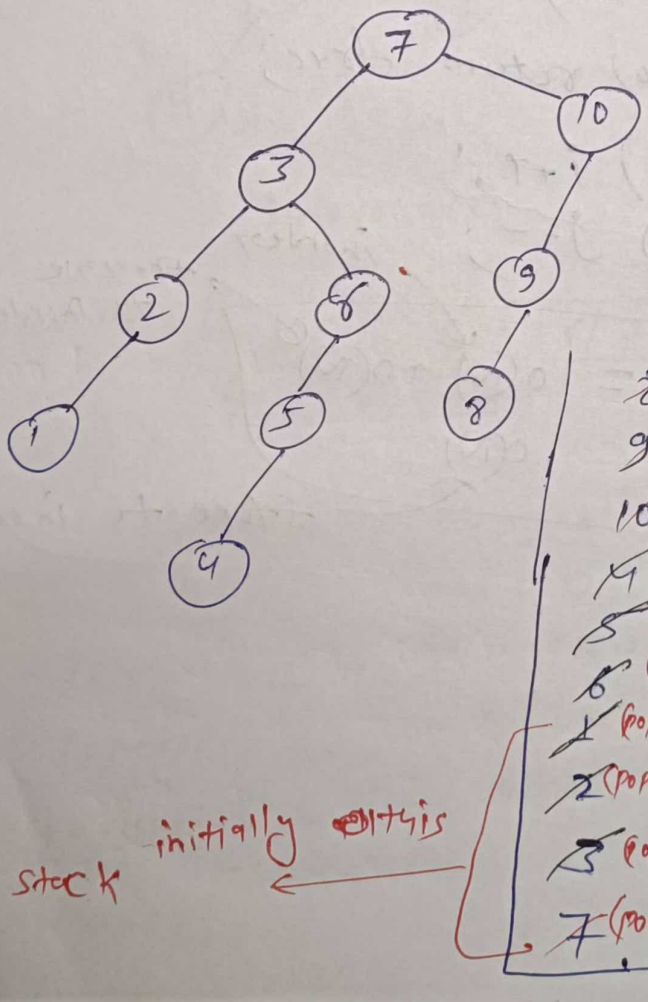
i.e., first we store all extreem left into stack & ~~which~~ which next() is called the return st.top(); & st.pop()

i.e. left node right

this is done

this means next() called

now put node> right & all left onto stack.



BST iterator (7)

next → 1
1→right = NULL

next → 2
2→right = NULL

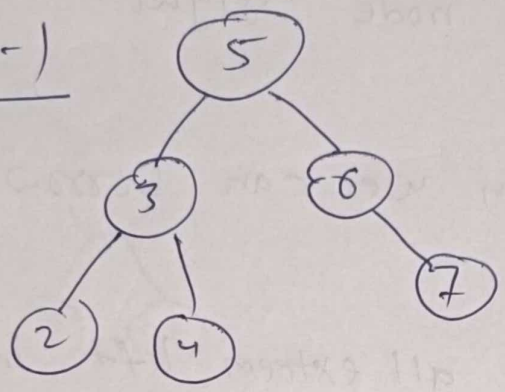
next → 3
3→right = 6
6→left = 5
insert all left onto stack

next → 4
next → 5
next → 6
next → 7
insert 7→right = 10 & all 10's lefts

next → 8
next → 9
next → 10

Two Sum In BST

Approach-1



Store the inorder of BST & apply two pointer approach

2 3 4 5 6 7

```
sum = in[i] + in[j]
if ( sum == k) return true;
if ( sum < k) i++;
if ( sum > k) j--;
```

inorder

traverse the inorder

T.C. = O(N) + O(N)

S.C. = O(N)

store the inorder

Approch-2

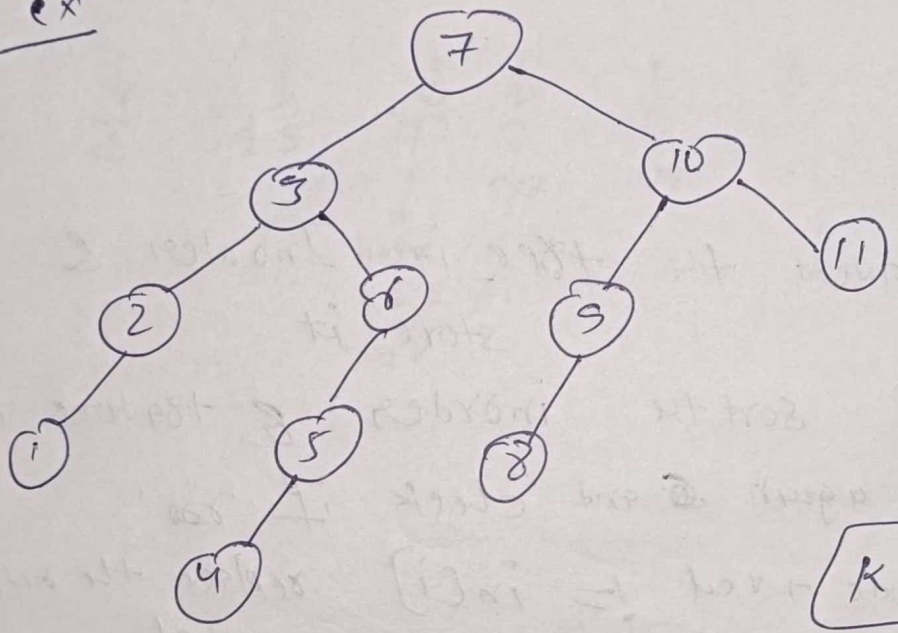
We use the concept of BST Iterator

i.e.

next() ← this will give you the next value in inorder

before() ← this will give the last value in inorder

ex



T.C. = O(N)

S.C. = O(H) × 2

(next + before)

k = 16

i → next() j → before()

inorder → | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

i = next() = 1 j = before() = 11

1 + 11 = 12 < 16

so increas i

i = next() = 2 2 + 11 = 13 < 16 ✓

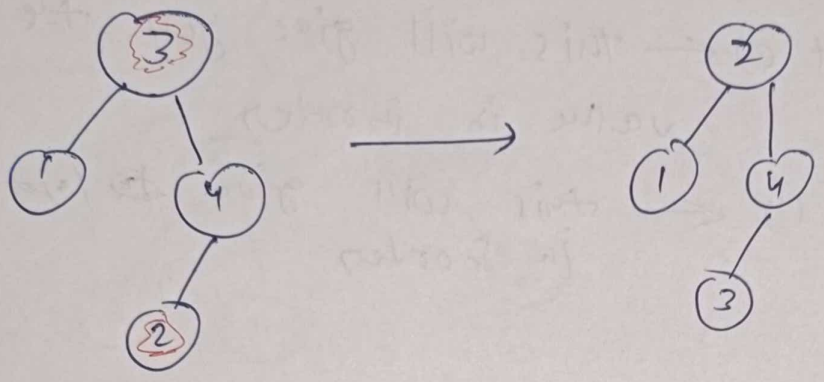
i = next() = 3 3 + 11 = 14 < 16 ✓

i = next() = 4 4 + 11 = 15 < 16 ✓

i = next() = 5 5 + 11 = 16 true;

Recover BST

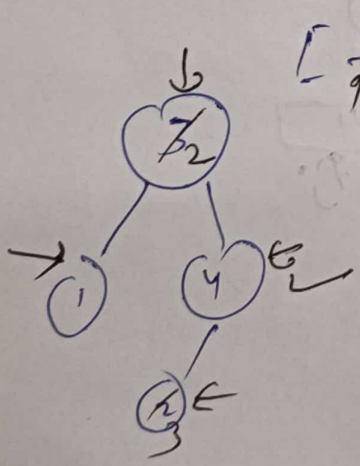
ex



Brute force :-

Traverse the tree in Inorder & store it

Now sort the inorder & traverse the tree again and check if node → val != in[i] replace the node val.



[1 2 3 4]

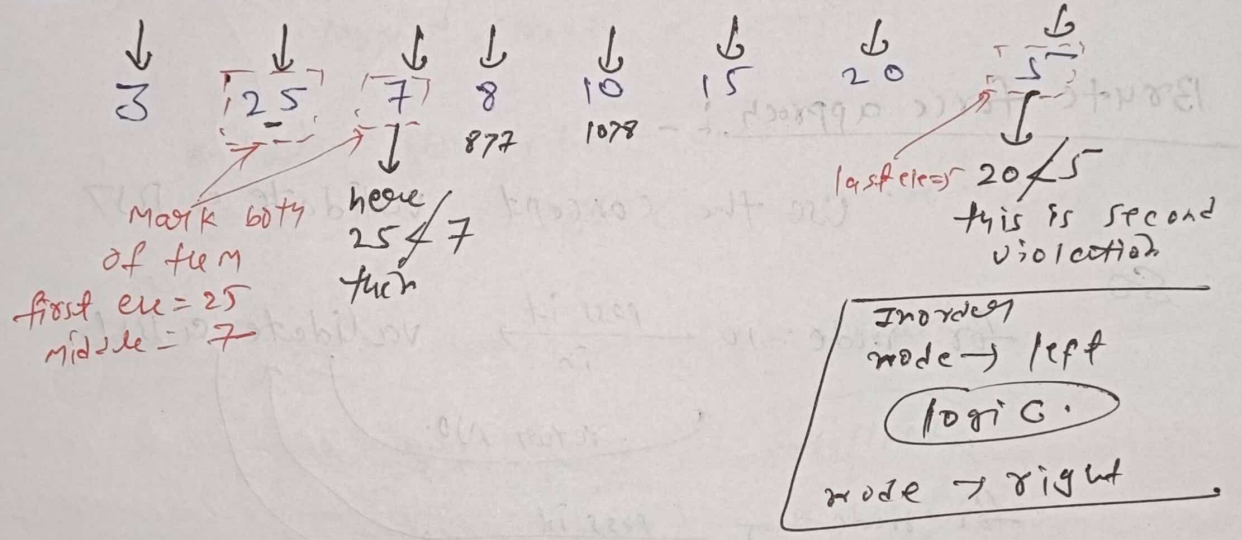
T.C. = O(N) + O(NlogN) + O(N)

S.C. = O(N)

Optimal Approach :-

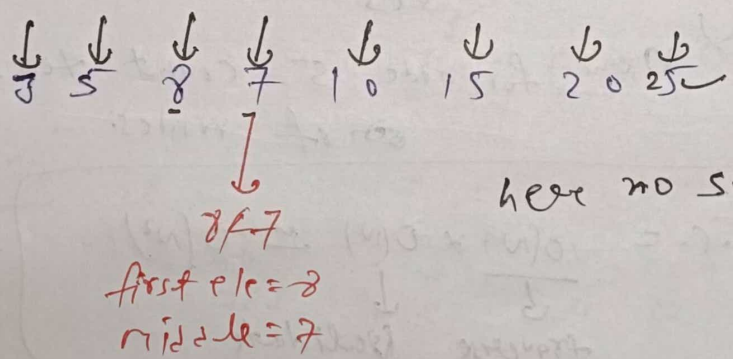
If we think about a sorted array where only two elements are swaped then there are only two cases. & inorder of BST is sorted so we can figure out these two nodes in these two cases.

(1.) Swapped nodes are not adjacent



Inorder
node → left
logic.
node → right

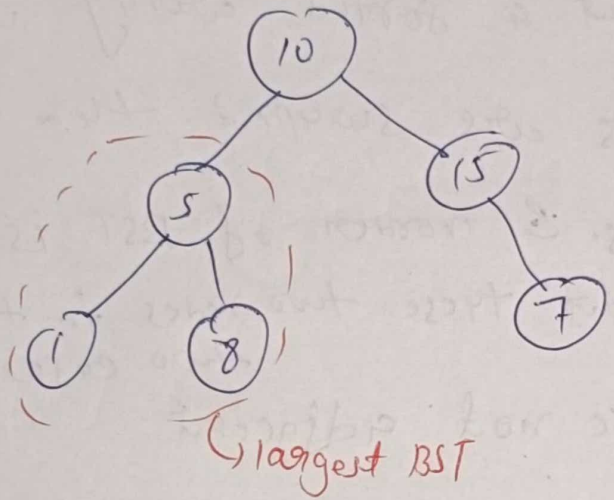
(2.) Swapped nodes are adjacent.



here no second violation.

T.C. = O(N)
S.C. = O(1)

Largest BST in BT



Brute force approch :-

Use the concept validate a BST

So

for node = 10 —pass it in→ validate a BST

← return NO.

for node = 5 —pass it in→ validate a BST

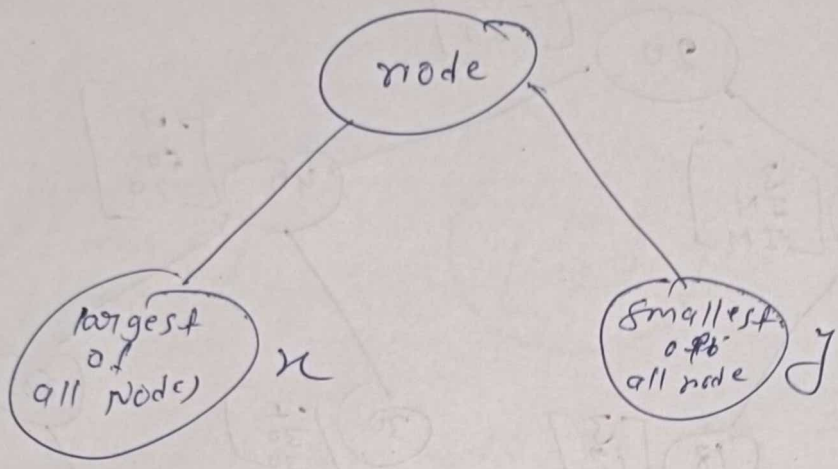
← Yes

Now for node 5 count the number of nodes.

$T.C. = O(N) * O(N) \approx O(N^2)$

- $O(N)$ → traverse tree
- $O(N)$ → validate BST

Approach-2

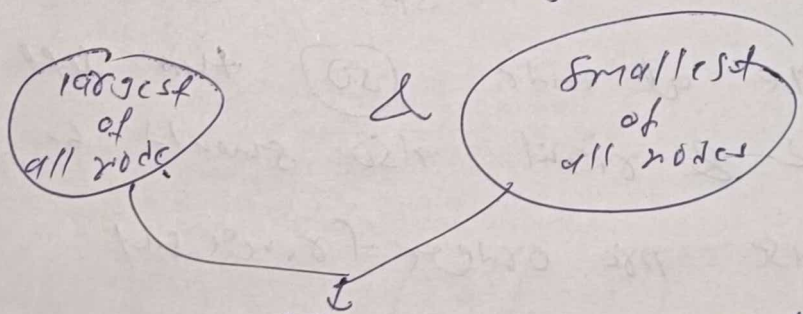


largest < node < smallest

↳ how you check size

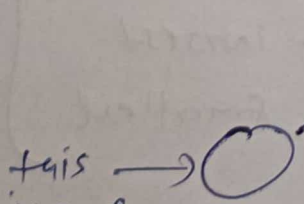
size = 1 + x + y

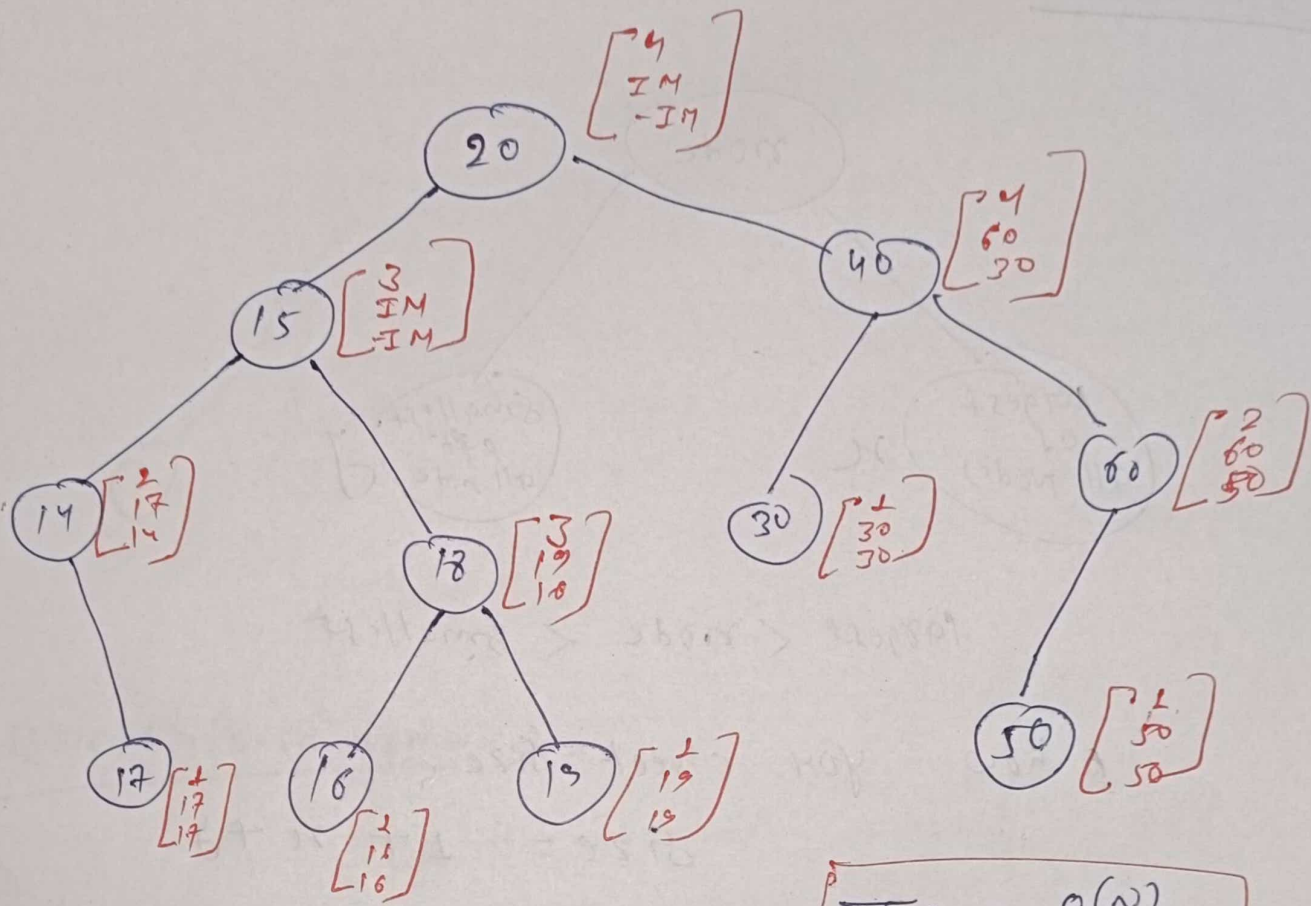
Ques → how can we say that



Soln → if we start from bottom node & if bottom guy is bst then if we add some node ∪

if it satisfied the cond'n then it is also, bst





not considered recursive stack space

T.C. = O(N)
S.C. = O(N)

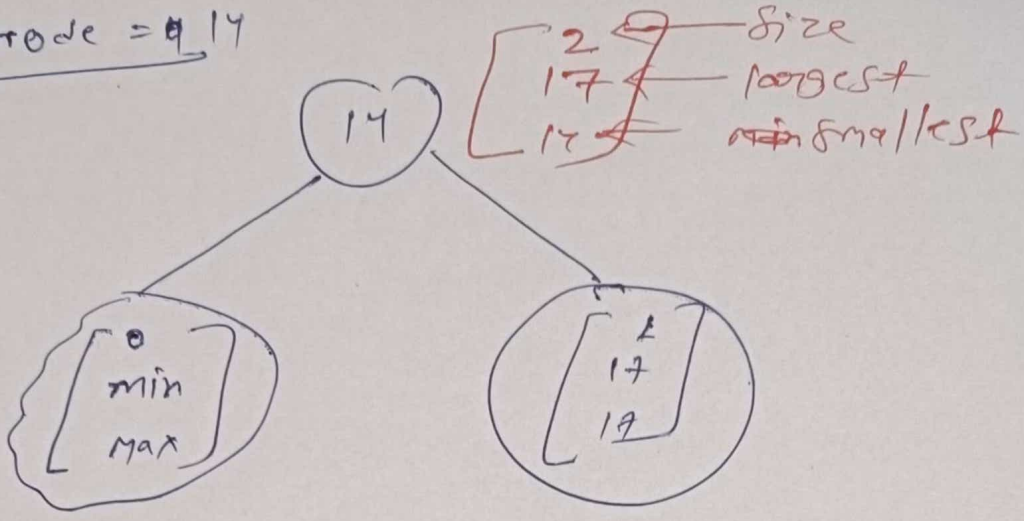
if we are at node (50) then left has to be computed & right also should be computed so we use pre order traversal

Preorder ⟶ left right root

for a leaf node → [1 ← size, 17 ← largest, 17 ← smallest]

for a NULL node → [0 ← size, INT_MIN ← largest, INT_MAX ← smallest]

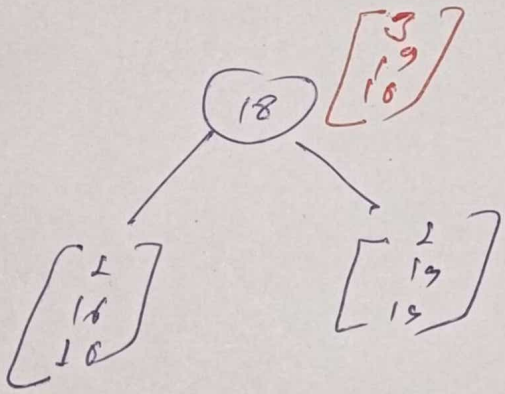
node = 14



min < 14 < 17 ⟶ True

then the node 14 can make a BST

node = 18

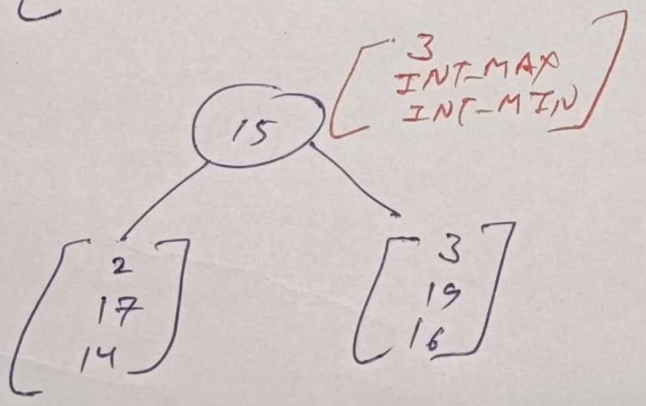


16 < 18 < 19

size = 1 + (1) + (1)

= 3

node = 15



17 < 15 < 16

↓ voilation

Imp when condⁿ is voilated then pass largest = INT_MAX, smallest = INT_MIN so that comparision could be true in future.

node = 20

IM < 20 < 30

↓ voilate

size = max (left, right)