

Must Do Binary Search Tree problems for Placement

Binary Search Trees	<u>Find a value in a BST</u>
Binary Search Trees	<u>Deletion of a node in a BST</u>
Binary Search Trees	<u>Find min and max value in a BST</u>
Binary Search Trees	<u>Find inorder successor and inorder predecessor in a BST</u>
Binary Search Trees	<u>Check if a tree is a BST or not</u>
Binary Search Trees	<u>Populate Inorder successor of all nodes</u>
Binary Search Trees	<u>Find LCA of 2 nodes in a BST</u>
Binary Search Trees	<u>Construct BST from preorder traversal</u>
Binary Search Trees	<u>Convert Binary tree into BST</u>
Binary Search Trees	<u>Convert a normal BST into a Balanced BST</u>
Binary Search Trees	<u>Merge two BST [V.V>IMP]</u>
Binary Search Trees	<u>Find Kth largest element in a BST</u>
Binary Search Trees	<u>Find Kth smallest element in a BST</u>
Binary Search Trees	<u>Count pairs from 2 BST whose sum is equal to given value "X"</u>
Binary Search Trees	<u>Find the median of BST in O(n) time and O(1) space</u>
Binary Search Trees	<u>Count BST nodes that lie in a given range</u>
Binary Search Trees	<u>Replace every element with the least greater element on its right</u>
Binary Search Trees	<u>Given "n" appointments, find the conflicting appointments</u>

Binary Search Trees	<u>Check preorder is valid or not</u>
Binary Search Trees	<u>Check whether BST contains Dead end</u>
Binary Search Trees	<u>Largest BST in a Binary Tree [V.V.V.V IMP]</u>
Binary Search Trees	<u>Flatten BST to sorted list</u>

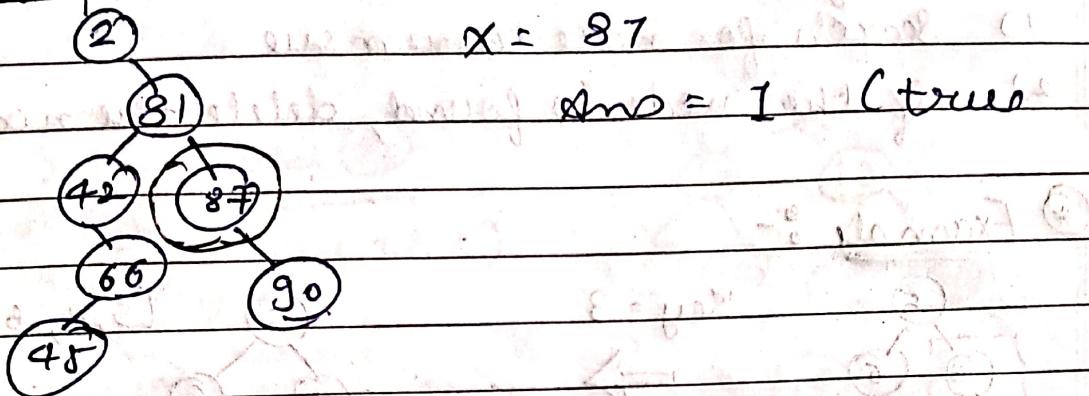
Binary Search Tree

Date: 22/3/22 Page no: _____

Ques - Search a node in BST

Given BST and a node value X , find if the node with value ' x ' is present in BST or not.

* Example :-



* Intuition :-

- ① Start from root
- ② compare searching element (x) with root.
if less than root, then recursively call left subtree.
- ③ else recursively call right subtree.
- ④ if the element to search is found anywhere, return true, else return false.

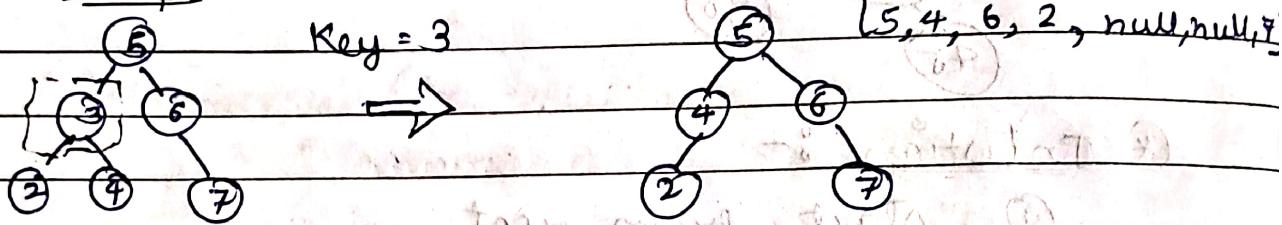
Ques :- Delete node in a BST

Given root node reference of BST & a key, delete node with given key in BST.
Return root node reference of BST.

Basically deletion can be divided into 2 stages:-

- 1) Search for node to remove
- 2) if the node is found, delete the node.

* Example :-



* Intuition :-

We've 3 cases for deleting node from BST :-

- (1) The node is a leaf node :-

In this case we just remove the node & return the root; deleting any leaf node doesn't affect remaining tree.

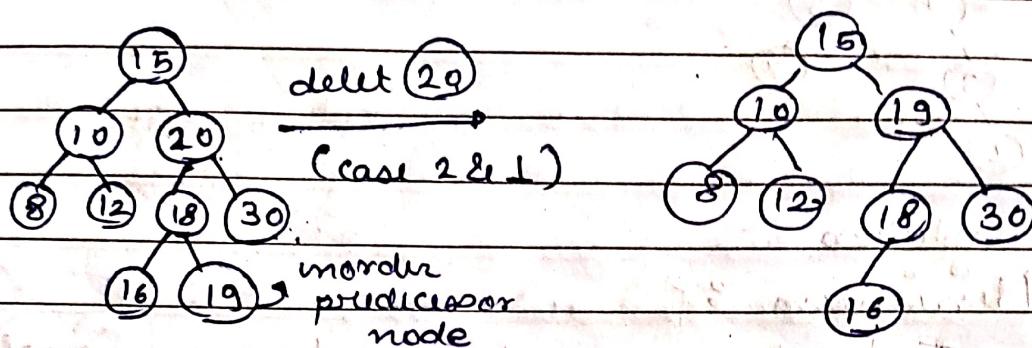
- (2) The node has one child :-

In this, we replace node with the child node and return the root.

(3) Node has 2 children :-

in this case, in order to conserve BST properties, we need to replace the node with its inorder successor, i.e. we need to replace it with either:

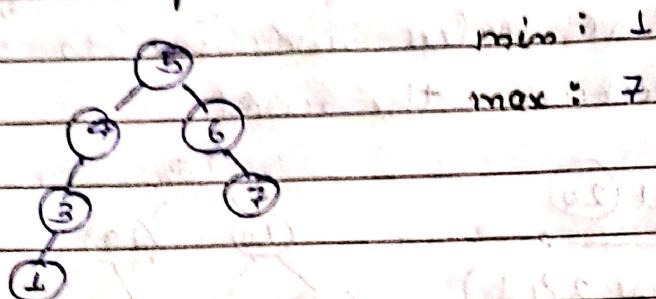
- (1) The greatest value node in its left subtree
- (2) The smallest value node in its right subtree & return the root.



Ques → find Min & max. value in BST

Given BST, find mini. and max. element in BST.

Example :-



min : 1

max : 7

② Intuition :-

Minimum :-

Just traverse the node from root to left recursively until left is null.

The node whose left is Null is the node with min. value

Maximum :-

Just traverse the node to right recursively until the right is null.

The node whose right is Null is the node with max. value.

Snippet :-

(1)

```

int maxValue (struct node* node)
{
    struct node* current = node;
    while (current->right != null)
        current = current->right;

    return (current->data);
}

```

(2)

```

int minValue (struct node* node)
{
    struct node* current = node;
    while (current->left != null)
        current = current->left;

    return (current->data);
}

```

Ques → Predecessor and Successor

There is BST given, with root node with key part as integer only.
find inorder successor & predecessor of given key.

If not found any return -1.

Example :-

Input - T = no. of test cases, each T contains 'n' denotes no. of edges of BST.
rest contains edges of BST.

Last line contains BST.

2 → T

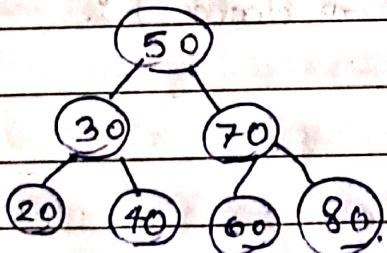
6 → n

50,30L 30,20L 30,40R 50,70R 70,60L 70,80R

65 → Key

Output

predecessor = 60 & successor = 70.



Approach :-

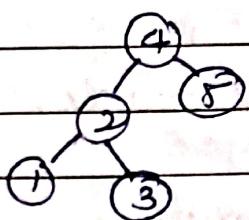
- ① If root is null, return.
- ② If key is found,
 - (a) if its left subtree is not null, the predecessor will be the right most child of left subtree or leaf child itself.
 - (b) if its right subtree is not null, the successor will be the left most child of right subtree or right child itself.
- ③ If key is smaller than the root node, set the successor as root. search recursively into left subtree.
→ else, set the predecessor as root. search recursively into right subtree.

Ques. Check for BST

* Binary search tree has following properties :-

- (1) Left subtree of node contains only nodes with keys less than the node's key.
- (2) Right subtree of node contains only nodes with keys greater than the node's key.
- (3) Both left and right subtrees also be BST.

Example:-



Ans = True

* Intuition :-

- (1) Do inorder traversal of given tree and store result in temp vector.
- (2) This method assumes that there are no duplicate values in tree.
- (3) Check if the temp array is sorted in ascending order, if it is, then tree is BST.

Implement :-

bool is BST (node * root)

(E) struct node * prev = null;

if (root)

(E) if (! isBST (root -> left))
return false;

if (prev != null & & root -> data <= prev -> data)
return false;

prev = root ;

(E) return isBST (root -> right);

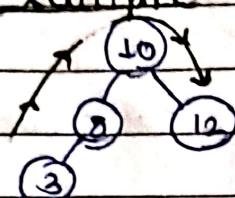
(3) return true ;

(3)

Ques - Populate inorder successor of all node

Given BT, write a funcⁿ to populate next pointer for all nodes. The next pointer for every node should be set to point to inorder successor.

② Example :-



next next next next
 $\text{Ans} = 3 \rightarrow 8 \rightarrow 10 \rightarrow 12 \rightarrow 1.$
 Inorder = 3, 8, 10, 12

① Intuition :-

- Take a prev variable as 'prev == null'.
- during inorder traversal, go till the bottom & when if (prev == null) point prev = bottom element.



→ for 8 check if (prev != null)
 then prev → next = root;
 (this will point to the
 next successor.)

- Then prev = root, means 8 will be prev.
 do the same for 10 & 12 also

Implementation :-

```
void solve (node* root, node* &prev)
```

(1) if (!root) return;

```
solve (root->left, prev);
```

if (prev != null) (2)

```
prev->next = root;
```

(3)

```
prev = root;
```

```
solve (root->right, prev);
```

(4)

```
void populateNext (node* root)
```

(5)

```
node* prev = NULL;
```

```
solve (root, prev);
```

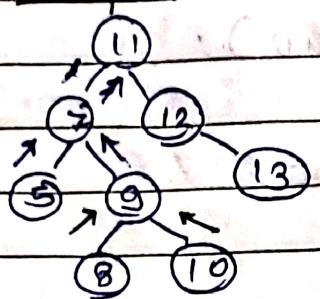
(6)

Ques

Find LCA of 2 nodes in BST

Given BST, & 2 node values. Find LCA of 2 nodes in BST

* Example :-



$$n_1 = 8$$

$$n_2 = 10$$

both 8 & 10 meeting out 9.

$$n_1 = 5$$

$$n_2 = 5$$

$$\rightarrow \text{LCA} = 9$$

Intuition :-

Here we'll not use $O(N)$ method.

We'll check for the 2 condition, if the given 2 condtn becomes false then we got our LCA

condtⁿ 1 - $n_1 < \text{root} \rightarrow \text{data}$ } left
 $n_2 < \text{root} \rightarrow \text{data}$

condtⁿ 2 - $n_1 > \text{root} \rightarrow \text{data}$ } right
 $n_2 > \text{root} \rightarrow \text{data}$

If condtn 1 & 2 become false then become false that mean 1 value is at left another at right, we found our LCA
eg. - (above example)

$$n_1 = 8 \quad n_2 = 10$$

(1) $8 < 11, 10 < 11 \Rightarrow$ move to left

$8 > 7, 10 > 7 \Rightarrow$ move to right

$8 < 9, 10 > 9 \Rightarrow$ LCA = 9

current node

② Implement :-

```
node* LCA ( node * root, int n1, int n2 )
{
```

```
    if (!root) return null;
```

```
    if (n1 < root->data and n2 < root->data)
```

```
        return LCA( root->left, n1, n2 );
```

```
    else if ( n1 > root->data and n2 > root->data)
```

```
        return LCA( root->right, n1, n2 );
```

```
    else return root;
```

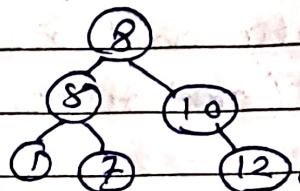
Ques - Construct BST from preorder traversal

Given a preorder traversal of BT, construct a BST.

Example :-

preorder: { 8, 5, 1, 7, 10, 12 }

BST -



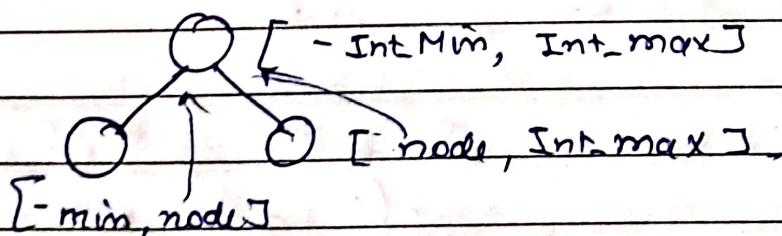
* Intuition :-

- ∵ preorder is sorted, so firstly we'll sort the entire given data.
- Now, you've both preorder (given) & inorder (from sorting), so now you can construct BST.

TC - $O(n \log n)$ + $O(n)$

SC - $O(n)$

* Efficient Sol



assume upper bound $ub = INTMAX$,

i.e = 8.

- "∴ preorder = root \rightarrow left \rightarrow right,
Here, there might be left might not be.
- i.e everything at left < 8 (ub).

⑧ up

i.e $8 < 5$ (left)

5 (ub = 8)

7 (ub = 5)

1 (ub = 5)

✗ null.

→ we don't require lower bound
bcz, 7 can't be planted
at 1 or 5, it is automatically
checking for lb.

- Same goes for 10 & 12.

* Implementation :-

TreeNode* fromPreorder (vector<int> A) ⑤

int i = 0;

return build (A, i, INT_MAX);

③

TreeNode* build (vector<int> &A, int &i, int bound)

⑥ if ($i == A.size() \text{ || } A[i] > bound$)

return NULL;

TreeNode* root = new TreeNode (A[i++]);

root \rightarrow left = build (A, i, root \rightarrow val);

root \rightarrow right = build (A, i, bound);

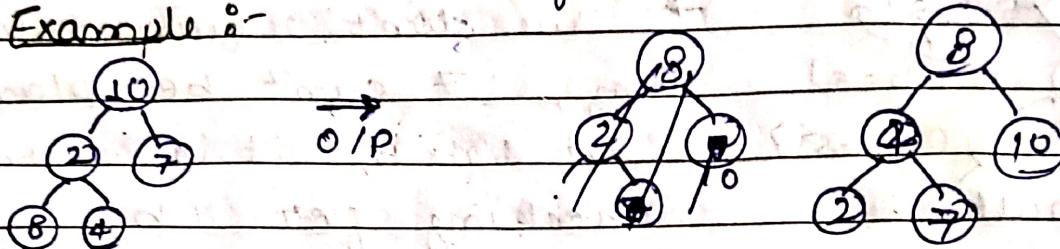
return root;

③

Ques - Convert Binary Tree to BST

Given BT, convert it into BST. It should be done in a way that it keeps the original structure of BT.

- Example :-



- Intuition :-

We can solve the above problem in 3 steps:

- (1) Create a temp array arr[], that stores inorder traversal of tree.
- (2) Sort arr[]. In this implementation Quick sort will be used: $TC(n^2)$.

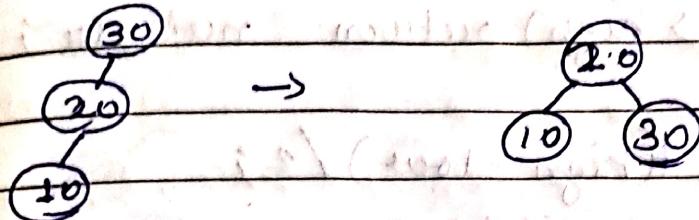
- (3) Again do inorder traversal of tree and copy array elements to tree node one by one.

$$TC = O(n \log n)$$

$$SC = O(n)$$

Convert normal BST into Balanced BST
 Given BST that may unbalanced,
 convert it into a balanced BST that
 has min. possible height.

① Example :-



② Intuition :-

- ① Traverse given BST, in inorder & store result in an array. ($TC = O(n)$).

Note that this array would be stored as inorder traversal of BST always produce sorted sequence.

- ② Build balance BST, from above created sorted array using recursive approach.
 $TC = O(n)$.

$Sc = O(n)$

② Snippets :-

Node * cons (node * root, vector < node > common,
int low, int high)

③

if (low > high) return nullptr;

int mid = (low + high - low) / 2;

root = common [mid];

// Recursive approach

root → left = cons (root, common, low, mid - 1);

root → right = cons (root, common, mid + 1, high);

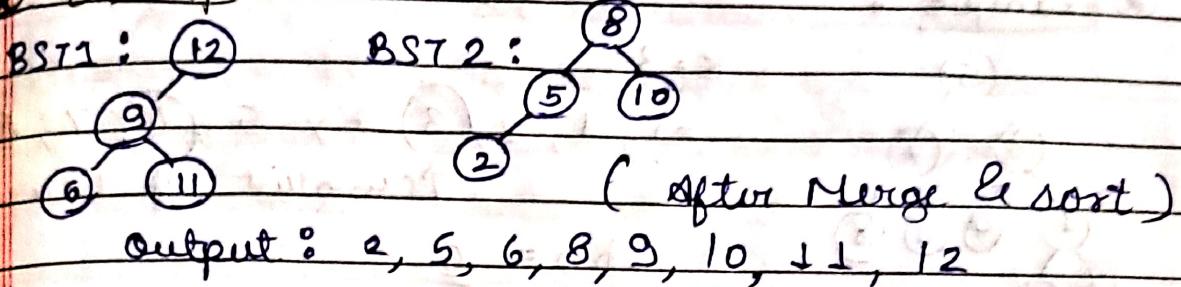
return root;

③

Merge 2 Balanced BST

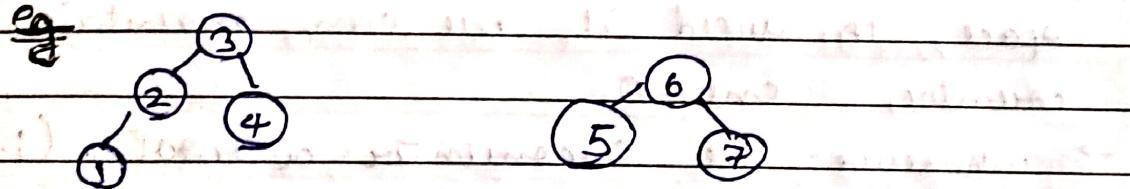
Given 2 BST, write a funcⁿ that merge 2 given balanced BST into single BST.

Example :-



Intuition :-

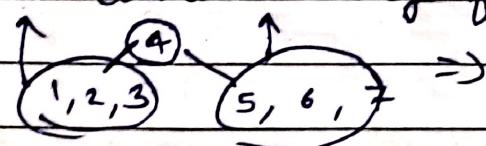
- ① find inorder of both trees, in an array
- ② merge both array.
- ③ Convert normal BST to Balanced BST concept. (prev. question)



array 1 = 1, 2, 3, 4 array 2 = 5, 6, 7

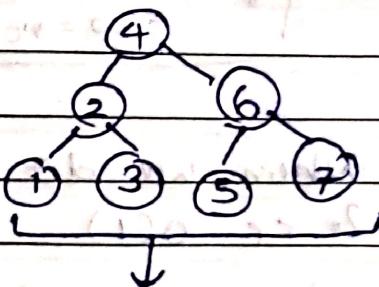
array 3 = 1, 2, 3, 4, 5, 6, 7

for divide array from middle



Send 1 to left send 5 to L

3 to right 7 to R (right)



Balanced BST

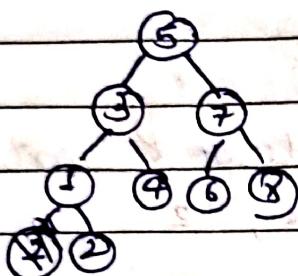
TC - $O(n+m)$

SC - $O(n+m)$

Ques \rightarrow k^{th} Largest element in BST

Given BST. write funcⁿ which will return k^{th} largest element without doing any changes in BST.

④ Example :-



$$k = 3$$

$\Rightarrow 1, 2, 3, 4, 5, 6, 7, 8$
↑ smallest ↑ (largest)

$$\text{ans} = 6$$

④ Intuition :-

\rightarrow Inorder of any given property is always in sorted order. i.e. left, root, right

\rightarrow Now to store BST, we are using extra space, to avoid it we can maintain counter, $\text{cnt} = 0$.

\rightarrow whenever we encounter a root, (i.e. left root right). Use $\text{cnt}++$

$\text{cnt}++$

Inorder \rightarrow if ($\text{cnt} == k$)
 $\text{ans} = \text{root}$

\rightarrow When we do Morris traversal $\text{TC} - O(n)$ & $\text{SC} - O(1)$.

\rightarrow So, we use Morris traversal instead of recursive or iterative.

*) Snippet :-

```
int ans; // global
```

```
void solve (node* root, int k, int & idx) .
```

(E)

```
if (!root) return;
```

```
solve (root -> left, k, idx);
```

```
if (n-1 == idx) (E) // n = size of tree
```

```
ans = root -> data;
```

```
idx ;
```

```
return;
```

(3)

```
else idx++;
```

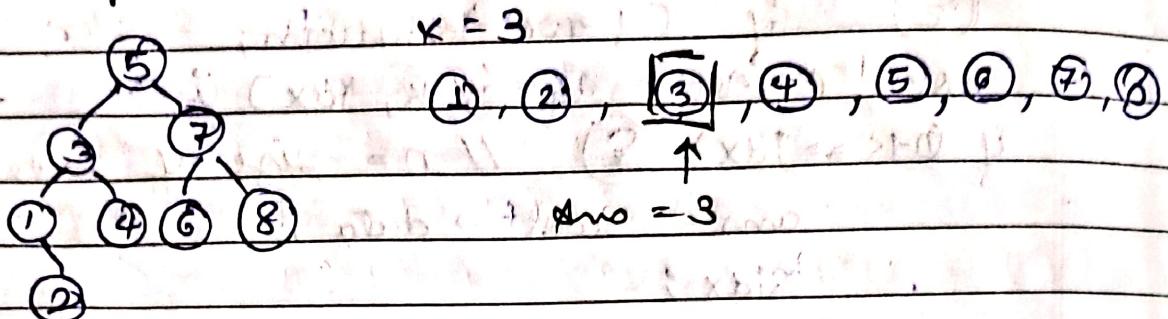
```
solve (root -> right, k, idx);
```

(3)

Ques. 1st Smallest element in BST.

Given k^{th} smallest int in BST. find k^{th} smallest element in BST

Example



Institutions

- we first find inorder traversal of given BST.
 - Maintain a counter if counter is equal index.
 - we'll get our answer.

$a[5] = 1, 2, 3, 4, 5, 6, 7, 8$

index - 0 1 2 ↑ 3 4 5 ↑ 6 7
small largest

egg small

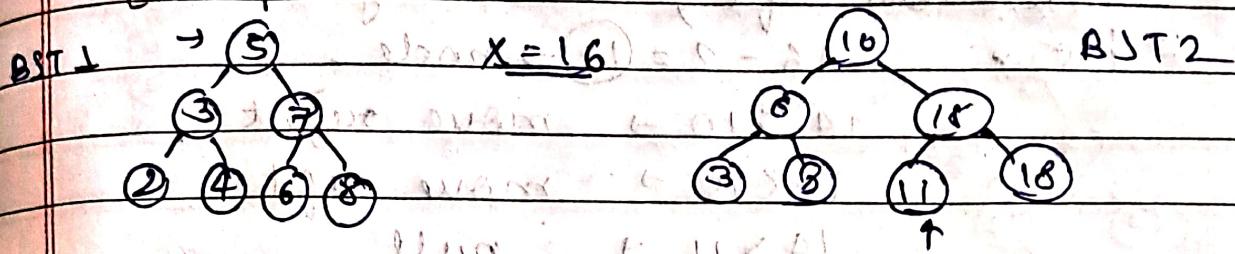
$$\boxed{K=3 \text{ (3rd largest)}} \Rightarrow \boxed{n-K \text{ smallest}} \\ \therefore 7-3 = 5^{\text{th}} \text{ ans} \checkmark$$

for largest

Count pairs from 2 BST whose sum is equal to given value "x".

Given 2 BST with N_1 & N_2 distinct nodes & a given value X . Task is return count of all pairs whose sum is equal to X from both the BST.

Example



$$16 - 5 = 11 \Rightarrow (16, (5, 11))$$

$$16 - 3 = 13 \times 0; 16 - 2 = 14 \times 0; 16 - 8 = 8 \Rightarrow (8, 8)$$

return count.

① Intuition :-

→ Take 1 unordered map & store the element inside the map, on BST 2.

→ Run inorder traversal on BST 1 and check if $(X - \text{element}) \in \text{map}$. i.e. $16 - 5 = 11$, present in map or not.

→ if present use count (count++),

② return count.

TC - $O(n+m)$

SC - $O(m)$.

① Optimal solⁿ (requires extra space)

→ Find inorder of BST 1, and then check for ; $(x - \langle \text{element} \rangle) = \text{node}$ that node present in BST 2 or not.

→ now if node is $\langle \text{element}$ move left, else right.

i.e. $16 - 2 = 14 \rightarrow \text{node}$.

$14 > 10 \rightarrow \text{move right}$.

$14 < 15 \rightarrow \text{move left}$

$14 > 11 \rightarrow \text{null}$

(not found)

$16 - 6 = 10, \Rightarrow (6, 10) \text{ found}$

$16 - 5 = 11 \rightarrow 11 > 10 \rightarrow \text{move right}$

$11 < 15 \rightarrow \text{left}$

(5, 11) found

TC - $O(n \times \Theta(\text{height(BST2)}))$

SC - $\Theta(\text{height}) \Rightarrow O(1)$

* Implementation :-

```

void solve (node *root1, node *root2, int k)
 $\Sigma$  if (!root1) return;
    solve (root1->left, root2, k);
    if (fetch (root2, k, -root1->data))
        c++;
    solve (root1->right, root2, k);
 $\Sigma$ 

```

```

int countPair (node *root1, node *root2,
               int x)
 $\Sigma$ 

```

$c = 0;$

solve (root1, root2, x)

return c;

(3)

27/3/22

Ques → Find median of BST in O(n) time

Given BST, find median of it.

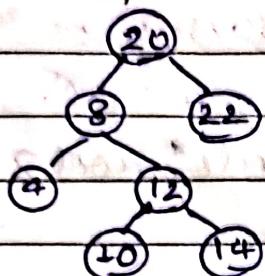
If no. of nodes are even :-

$$\text{median} = (n/2^{\text{th}} \text{ node} + (n+1)/2^{\text{th}} \text{ node}) / 2.$$

If no. of nodes are odd :-

$$\text{median} = (n+1)/2^{\text{th}} \text{ node}.$$

④ Example :-



$$\text{median} = 12$$

no. of nodes = 7 (odd)

⑤ Intuition :-

→ To find median, we need to find inorder of BST (because we'll get sorted order).

Then we'll find the median

→ ∵ we are not allowed to use extra space so for inorder, recursion & iterative approach can't be applied.

→ we can use Morris inorder traversal coz it doesn't require extra space.

Implementation :-

count no. of node in given BST, using morris Inorder.

Then perform morris Inorder one more time by counting nodes & by checking if count is equal to the median point.

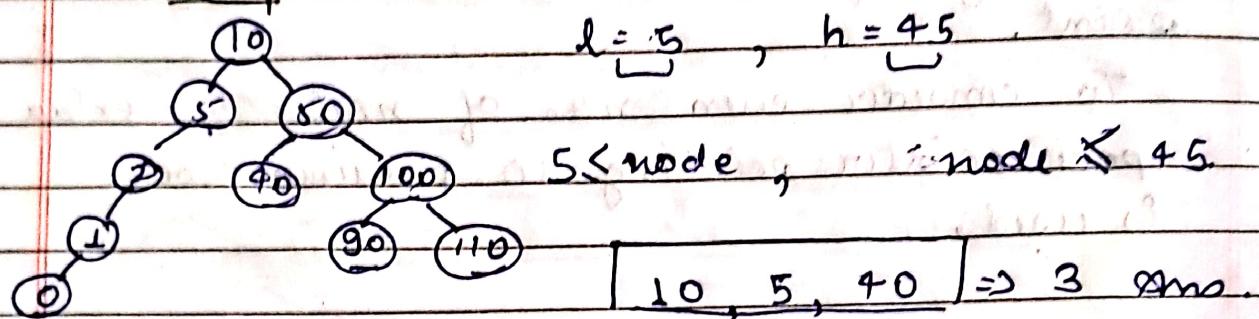
To consider even no. of nodes an extra space pointer pointing to previous node is used.

Ques + Count BST That lies in given range

Orinum BST, and using $[L, H]$, count no. of nodes in BST that lies in given range.

- ① Value smaller than root go left side.
- ② value greater or equal to root go right.

Example :-



④ Intuition :-

① Let say we pick 10 from above exm.
 $5 \leq 10 \leq 45$, then I'll add 1 with
 left recursion + right recursion
 $\Rightarrow 1 + (\text{left}) + \text{right}$.

② Let say we've 2, now check if
 $(x \leq l) \rightarrow (\text{true})$ (right recursion)
 (∴, no use of calculating 2 & below)

③ Let say we've 50 now check
 $(x > h) \rightarrow (\text{true})$ (left recurr.)
 (∴, no use of calculating value greater than
 h).

② Implementation :-

```
int getCount (node * root, int l, int h)
```

③

```
if (!root) return 0;
```

```
{ if (root->data >= l and root->data <= h)  
    return 1 + getCount (root->left, l, h)  
    + getCount (root->right, l, h);
```

④

```
else if (root->data < l)
```

```
    return getCount (root->right, l, h);
```

```
else
```

```
    return getCount (root->left, l, h);
```

⑤

29/3/22 page no: _____

Ques → Given n appointments, find all conflicting appointments

Given 'n' appointment, find all conflicting appointment.

* Example :-

appointment = $\{ \{1, 5\}, \{3, 7\}, \{2, 6\}, \{10, 15\} \}$
 $\{5, 6\}, \{4, 100\} \}$

output $\rightarrow \{3, 7\}$ conflict with $\{1, 5\}$

$[2, 6]$ conflict with $[1, 5]$

$[5, 6]$ " " $[3, 7]$,

$[4, 100]$ " " $[1, 5]$.

conflict means : if it conflict with any other previous app. in array.

* Intuition :

One by one process all appointment from the second appointment to last.

for every app. i , check if it conflicts with $i-1, i-2, \dots, 0$.

we use Interval Tree to solve problem in $O(n \log n)$ time.

④ Interval Tree

- ① Create Interval Tree, init. with first appointment
- ② Do the follow for all other appointment start from second one.
 - (a) Check if current app. conflict with any of the existing appointment in IT.
if conflicts, then print the current appointment.
 - (This step can be done in $O(\log n)$ time)
 - (b) insert current appointment in interval tree.

$Tc = O(\log n)$

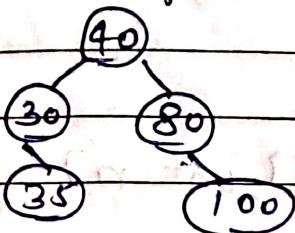
Ques Check if given array can be represent pre-order traversal of BST.

Given an array of no, return true if given array can represent pre-order traversal of BST else false.

* Example :-

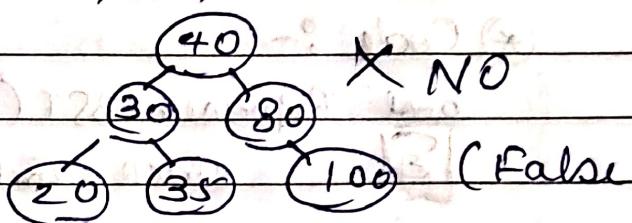
Input pre = {40, 30, 35, 80, 100}

ans/op = True.



Input \rightarrow {40, 30, 35,

20, 80, 100}



* Intuition :-

We can use stack to solve the prob.

Here, we find next greater element & after finding next greater, if we find smaller element, return false.

(1) Create empty stack, and initialize root as INT-MIN.

Do the follow, for every element $pre[i]$,

(a) if $pre[i]$ is smaller than current root, return false.

(b) Keep removing ele from stack while $\text{prev}[i]$ is greater than stack top. Make last removed item as new root (to be compared next).

At this point, $\text{prev}[i]$ is greater than the removed root.

(c) push $\text{prev}[i]$ to stack (all element in stack are in increasing order)

* Code :-

```
bool RepresentBST(int prev[], int n)
```

[1] stack<int> s;

int root = INT_MIN;

for (int i=0; i<n; i++) [2]

if ($\text{prev}[i] < \text{root}$)
return false;

while (!s.empty() && s.top() >= prev[i])

[3]

root = s.top();

s.pop(); [4]

③ s.push(prev[i]);

④ return true;

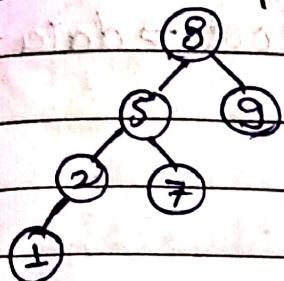
[5]

Ques. Check whether BST contains Dead end or not

Given BST, contains +ve int. value greater than 0. check whether BST contain dead end or not.

Dead end means we are not able to insert element after that node

* Example :-



O/p - Yes

1 is dead end bcoz we can't insert element after that.

* Intuition :-

→ we basically need to check if there is a leaf node with value x such that $x+1$ and $x-1$ exist in BST with exception of $x=1$.

→ For $x=1$, we can't insert 0 as the prob. say BST contains +ve integers.

IDEA

→ We first traverse the whole BST & store all node in a set. we also store all leaf in a separate hash to avoid re-traversal of BST.

→ Finally check for every leaf node X , if $X-1$ & $X+1$ present in [Set or not].

Recursive Approach :-

bool solve (node* root, int min, int max)

(E)

if (!root) return false;

if (min == max) return true;

return solve (root->left, min, root->data-1)

|| solve (root->right, root->data+1, max);

(3)

bool dead (node* root)

(E)

return solve (root, 1, INT_MAX);

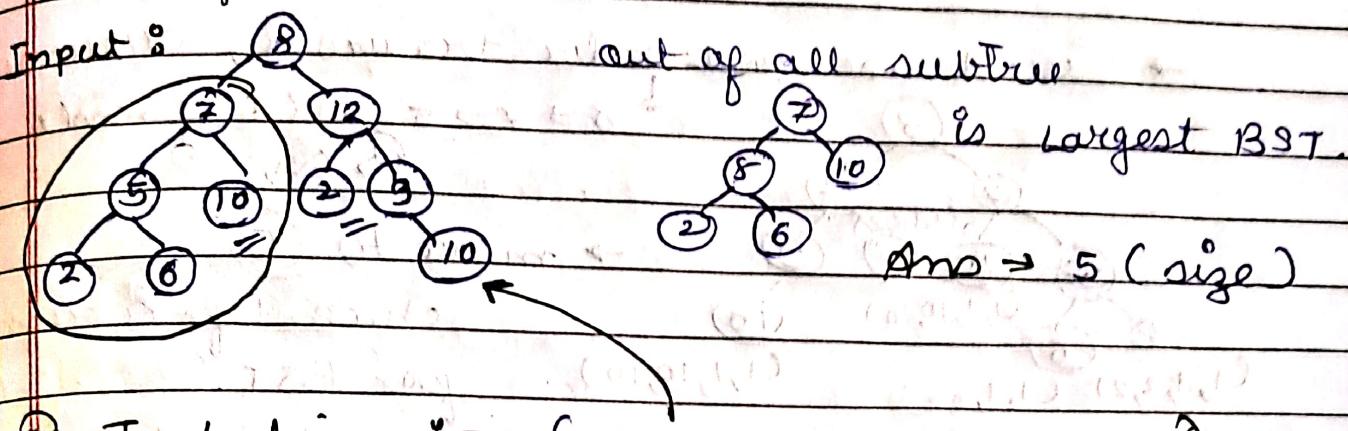
(3)

Largest BST in binary tree

Given a bt. find the size of its largest subtree that is a BST.

NOTE :- Here, size = no. of nodes in subtree

Example :-



① Intuition :- (Refer above example)

① Let say we pick 8 and check for the largest element in its left side, whether largest element of left is $>$ 8, if its greater than 8 then it can't be a BST.

② Same thing we'll check with right side whether all the elements we find the smallest element and that element is greater than 8, that mean all the elements are greater than 8 else can't be a BST.

Example

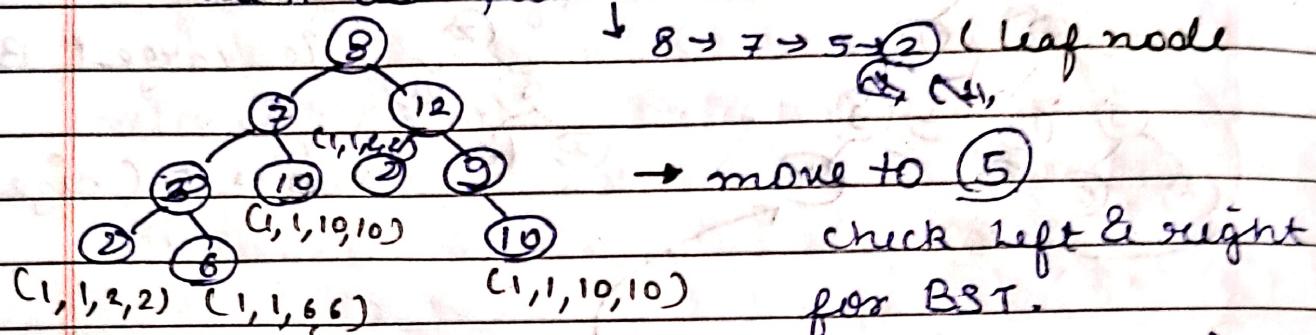
In left \Rightarrow compare 8 with 10

In right \Rightarrow compare 8 with 2

Approach :- $(BST, \text{size}, \text{min}, \text{max})$

- main & variable. (i) check whether particular node is forming BST or not. (1 and 0)
- (ii) store size
- (iii) minimum value, (iv) Maximum value

→ will use post-order traversal, recursively



(∴ 3rd variable is 1, 1 in

② & ⑥ hence ⑤ can form BST.)

⑤ become 1, 3, 2, 6)

→ Move to 7

If ($7 > 6$ and $7 < 10$) return true

⑦ become (1, 5, 2, 10).

→ move to 9 (have left null)

Null = (1, 0, max, min). ∵ 9 > min, 9 < max.

∴ 9 become → 1, 2, 9, 10)

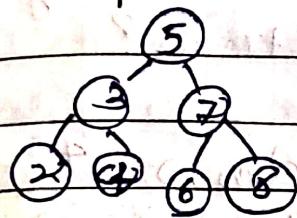
→ Move to 12

can't form bst coz $10 < 12$ on
right side

Ques :- Flatten BST to sorted list

Given BST, task is to flatten it to sorted list. Precisely, the value of each node must be lesser than the value of all the nodes at its right & its left node must be NULL after flattening.

Example :-



flatten - 2, 3, 4, 5, 6, 7, 8.

* Intuition :-

- re-create BST from its inorder traversal. It will take $O(N)$ extra space.
- we can apply inorder to BST as :

- ① Create a dummy node.
- ② Create variable 'prev' & make it point to the dummy node.
- ③ Perform inorder traversal & at each step :
 - ④ set $\text{prev} \rightarrow \text{right} = \text{current node}$
 - ⑤ set $\text{prev} \rightarrow \text{left} = \text{NULL}$
 - ⑥ set $\text{prev} = \text{current node}$

Improve sc to $O(1)$.