Recap:

① Structure of BT | types of trees → 8 recursive codes

② CRUD → Traversals → BFT → LOT → 8 variants.
 DFT → Pre, In, Post. (level switch)

③ Construction of BT (2 traversals, in. mandatory)

④ Catalan Number → # structurally different trees

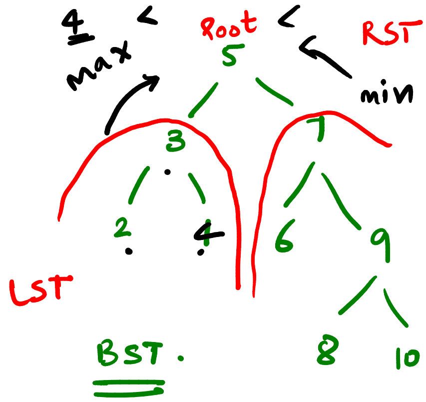
⑤ Hierarchy of tree → LCA | all ancestors
 root to node path

⑥ Figure out all branches.

BST. BT → structure & handling of BT. Nothing → how data/labels in nodes is filled.

Constraint: $\boxed{\text{all nodes in the left subtree} < \text{root node} < \text{all nodes in the right subtree}}$

Binary Search Tree.



Q.1 If tree is given (root), tell if it is a BST or not?

* min max Recursion.

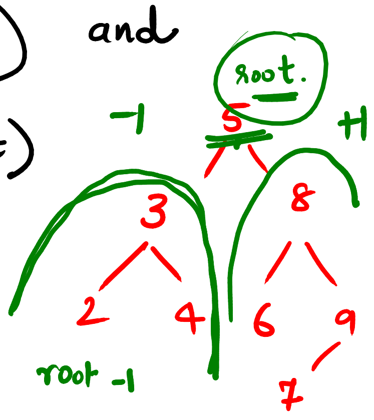
Idea: Leaf nodes.

min → left
max → right
correct.
left - right

bool isBST (root, min_left, max_right) :
 if root == null : return true
 if (root.data < ~~max~~ min_left || root.data > ~~min~~ max_right) :
  return false

leave

 return isBST (root.left, ~~max~~ min_left, root.data - 1) and
  isBST (root.right, root.data + 1, ~~min~~ max_right)



min_left < 4

Alternatively: *. 6+

Inorder traversal: Sorted order:

ans [] → 2 3 4 5 6 7 8 9 ⇒ if o/p is sorted O(n) traversal. BST.

CRUD.

R: Level → ×
Pre → ×
In → sorted order of nodes in array.
Post → ×

Q. Goldman Sachs Question:

BST and I give you a sum.
2 nodes that sum upto given sum.

2Sum Problem in BST.

search (root, key):

BST:

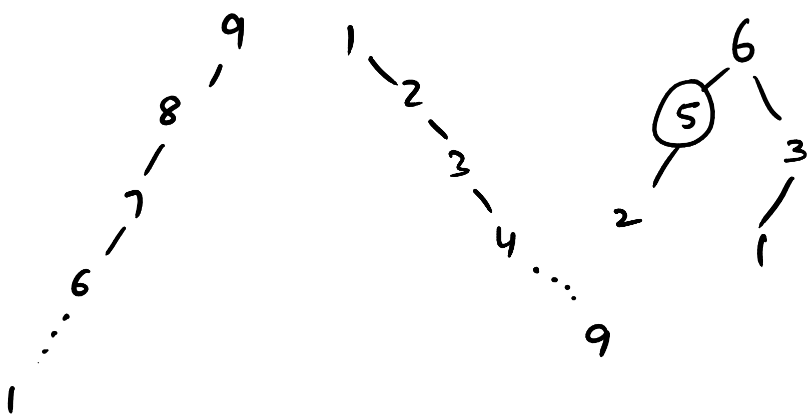
if root.data == key : return True
else if root.data < key :
    search (root.right, key)
else:
    search (root.left, key)

BT : O(n)

T.C. :

× O(log n) ← best you can achieved.

Even BSTs can be skewed. O(n) worstcase.



x

x search (root, sum-x)

n × n → O(n^2)

True
search (root, 5)

10

Best: Inorder traversal → sorted array O(n)
2 sum problem on sorted array → O(n)

T.C. O(n)

Any array question on sorting → BST question by input as BST.

i/p → normal tree → traversal O(n) → array → sorted
O(nlogn)

*

BST → traversal | sorted O(n)

random array → sorted O(n log n)

Create (C) : BST not every empty position can be a potential node position ✗ ∵ maintain constraint

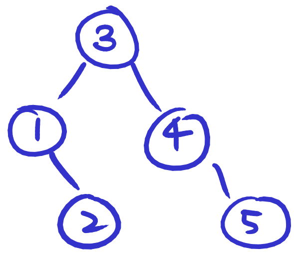
if new_node > root → go right

< root → go left

leaf append.

3 1 4 .5 2

BST



Catch.

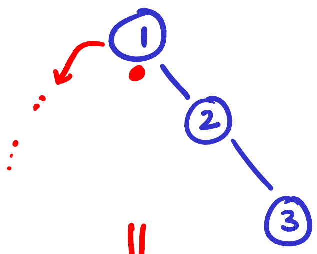
Tree may end up getting skewed.

BT → IDC if tree is skewed or not. Search O(n)

BST → do care if tree is skewed or not. Search O(height)

Rotations. Balanced ⇒ for all nodes $|h(\text{left ST}) - h(\text{right ST})| \leq 1$

i/p: 1 2 3 4 5 6

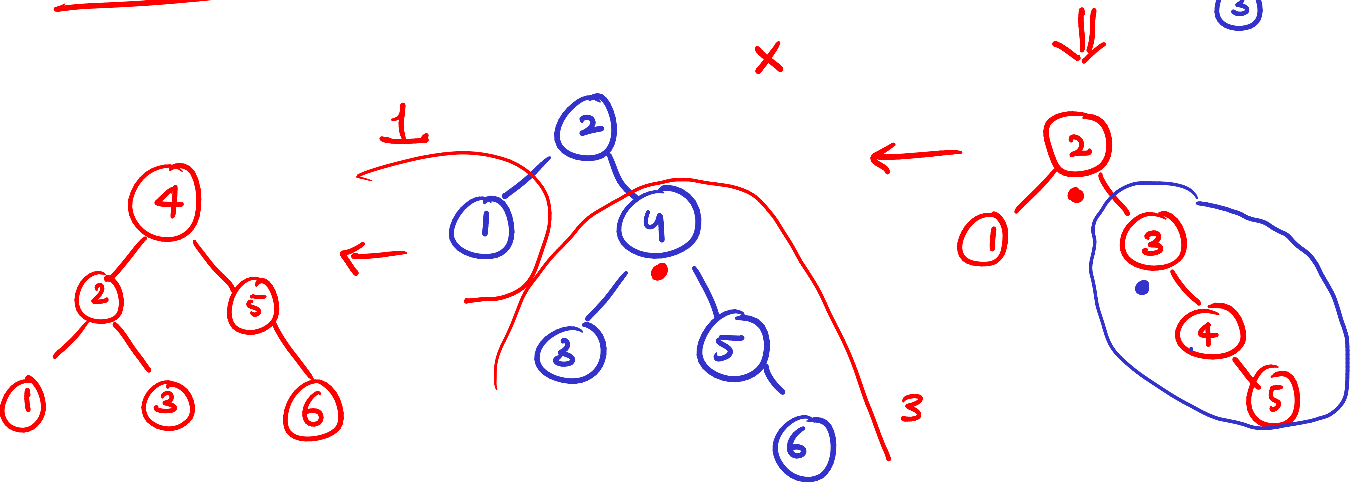


h(lst) 0

h(rst) 2

unbalanced.

O(1) rotation is performed.



h(lst): 1

h(rst): 3

Visualizer:

AVL Trees ⟹ self balancing BSTs.

short form of scientists.

height = O(log n).

Update : update (root , n_1 , n_2) $\longrightarrow$ replace n_1 by n_2

= delete (root , n_1) + insert (root , n_2) already discussed.

Final boss in BST . Delete (root , n_1) .

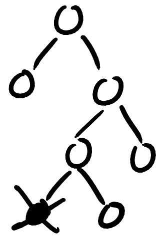
delete a node.

3 possibilities :

1. Trivial one. (0 children)

Delete the leaf node. Just discard it.

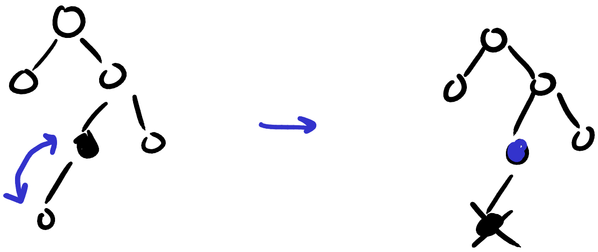
BST



2. One child.

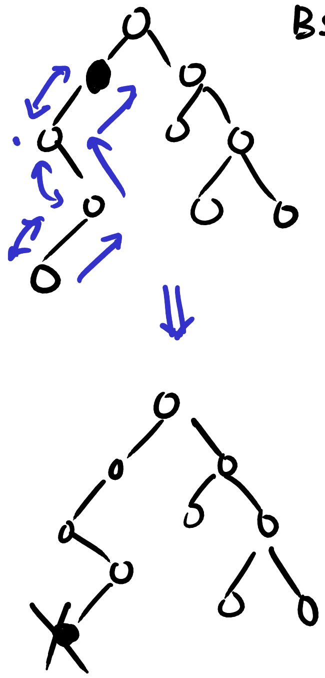
constant parent $\longleftrightarrow$ child swap , till your node to be deleted becomes a leaf node.

Eg (a)



Eg(b)

BST .

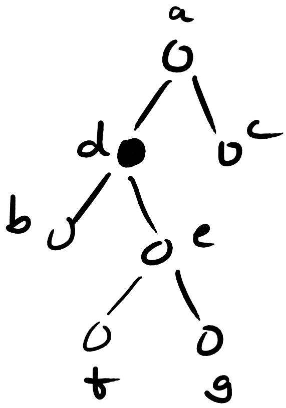


3. 2 children exist.

[a b c d e f g]

inorder

[a b c e f g]



If I want to delete node X , I want after deletion inorder traversal array to be sorted as well.

SORTED inorder traversal.

left subtree P X S right subtree

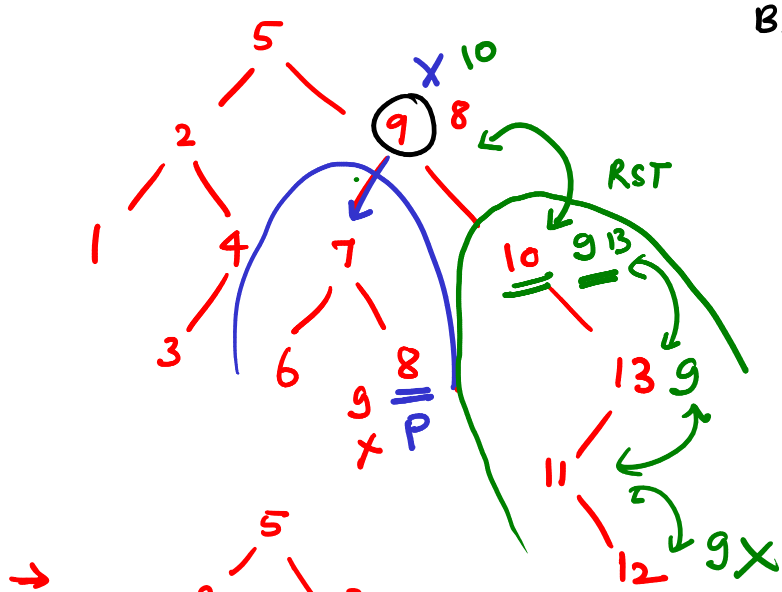
[_ _ _ _ _ X _ _ _ _]

delete this

X has 2 children

P $\Rightarrow$ max element of the left ST : Predecessor (X)

S $\Rightarrow$ min element of the right ST : successor (X)

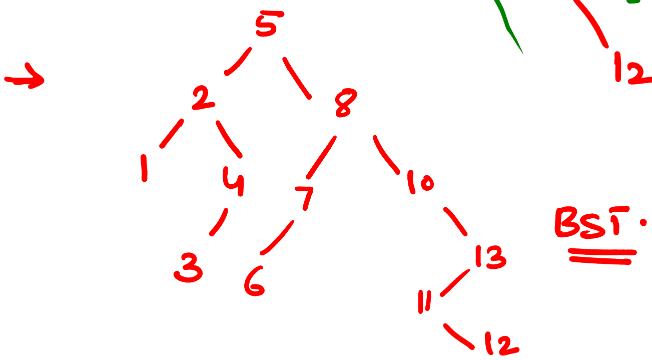


BST : Inorder :

1 2 3 4 5 6 7 8 ~~9~~ 10 11 12 13

Predecessor ⇒ max value in left ST.

Successor ⇒ min value in right ST.



Inorder :

1 2 3 4 5 6 7 8 10 11 12 13.

logic :

1. if root == null : return delete(root, key)

2. if root.data == key :
   pre = rightmost child of the left subtree or left child (right → cant go)

   OR

   succ = leftmost child of the right subtree or right child (left → cant go)

   swap (root, pre) / swap (root, succ)

3. search for key :
   if root < key : → right subtree
   else : → left subtree.

4. Check if key is a leaf node :
   → discard it.
   } Base cond^n.

CRUD	BT	BST
1. Search (key)	$O(n)$	$O(\text{height})$
2. Create (n nodes)	$O(n^2)$	$O(n \cdot \text{height})$
Insert (1 node)	$O(n)$	$O(\text{height})$
3. Read	$O(n)$	$O(n)$
4. Delete	$O(n)$	$O(\text{height})$
5. Update = 1 delete + 1 insert	$O(n)$	$O(\text{height})$.

By having AVL Tree : always achieve height = $\underline{\underline{\log n}}$.

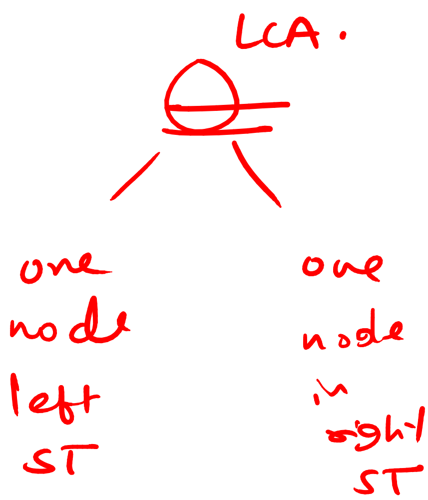
Exercise :

LCA(n_1, n_2) in BST.

if n_1 and n_2 < root :
go left

else if n_1 and n_2 > root :
go right

else if n_1 < root < n_2 OR
n_2 < root < n_1 : root → LCA



BST : TC. $O(\text{height})$.

— BST Concludes —