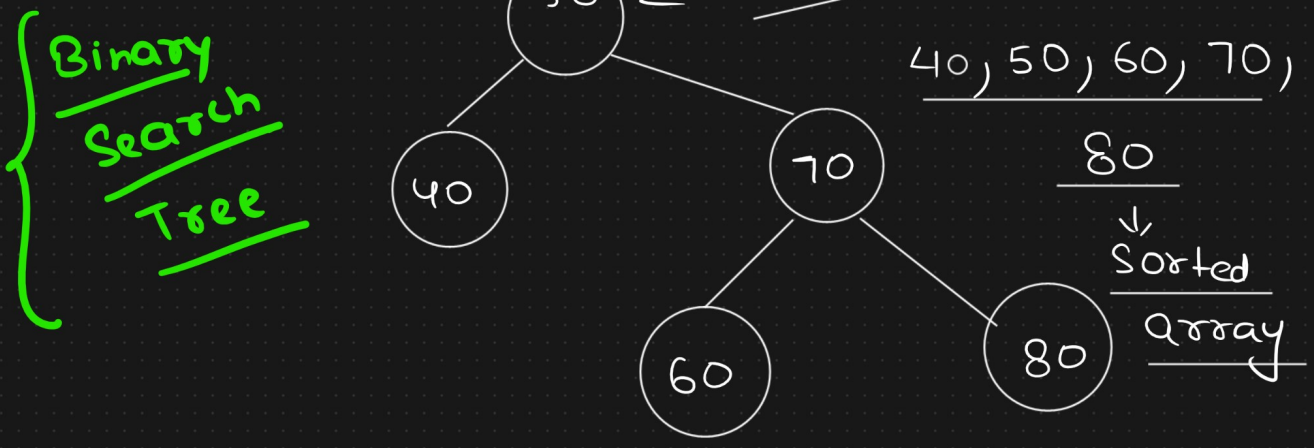
Binary Search Tree

1) Properties – Done ✓ — Approach
2) Operations: Insertion, Deletion, Searching – Done
3) Leetcode ✓ – Done

{ 50, 40, 70, 60, 80 }



Properties

1) Left Subtree < Parent Node

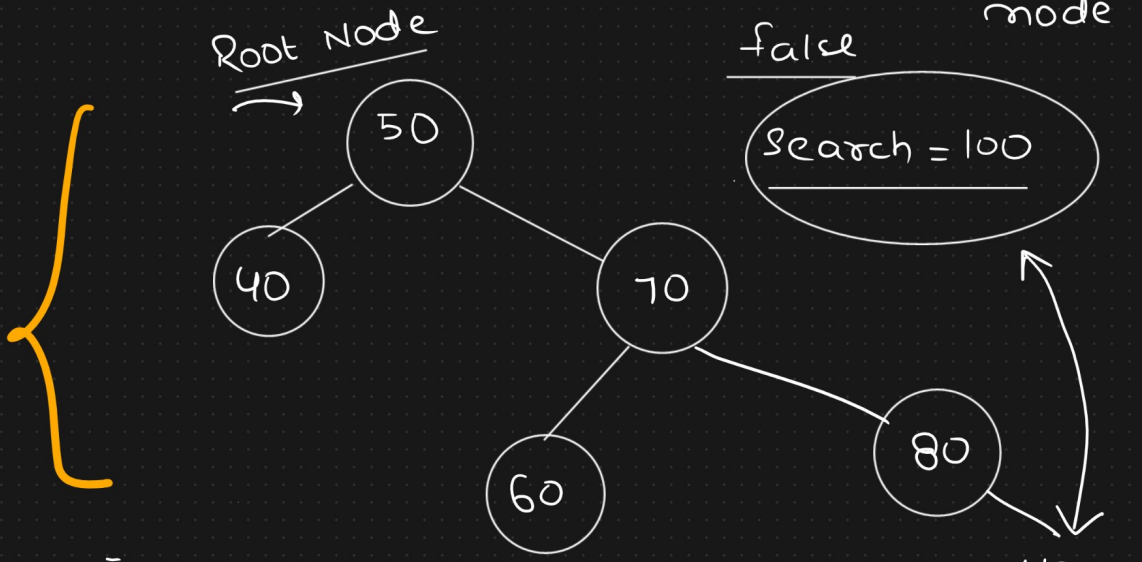
   Right Subtree > Parent Node

2) No duplicate elements in BST

3) Inorder traversal of BST
   ↳ Sorted array

Insertion

(new key) → Insertion → always at the leaf node



Inorder → 40, 50, 60, 70, 80

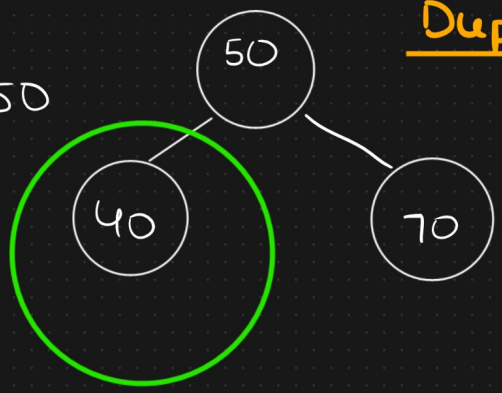
↳ BST

50, 40, 70, 40, 60, 80

Duplicate

key = 40

root.data = 50



50 < 40

No

↳ else

↳ 50 > 40 → True

root

40

40

root.left = insertBST(root.left, key)

Time complexity

$\Theta(h)$

Insertion / Searching

Balanced tree

Best case/average → $\Theta(\log n)$

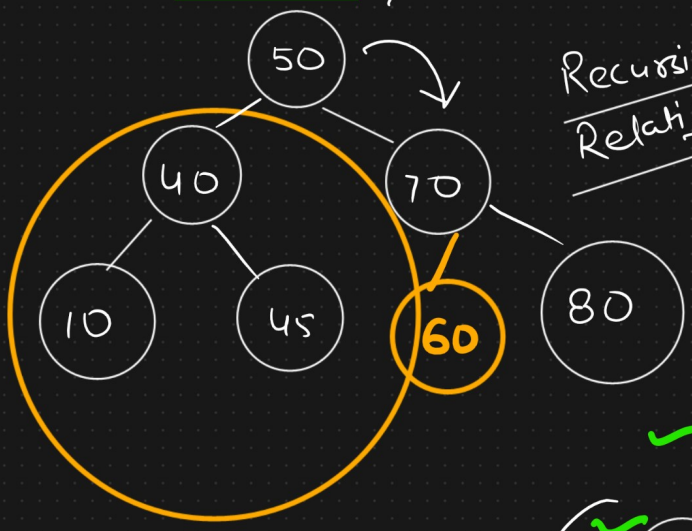
$x = 60$

Balanced

Worst case → $\Theta(n)$

Skewed tree

Valid BST / NOT



Recursive Relation

Left Right

$T(n) = T(n/2) + c$

$= \Theta(\log n)$

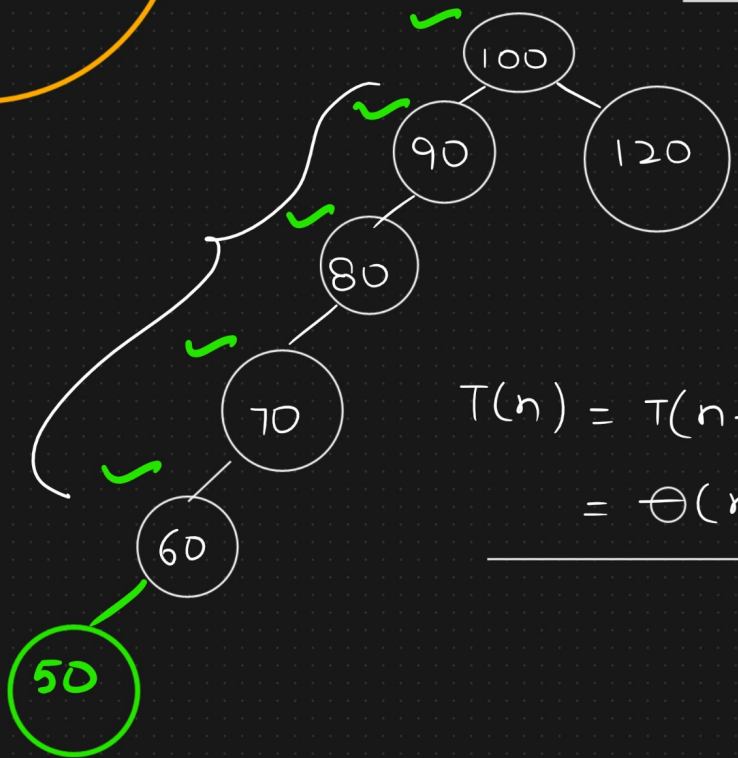
$x = 50$

$a = 1$

$b = 2$

$\log_b a$

$\log_2 1 = 0$



→ Left skewed BST

$T(n) = T(n-1) + c$

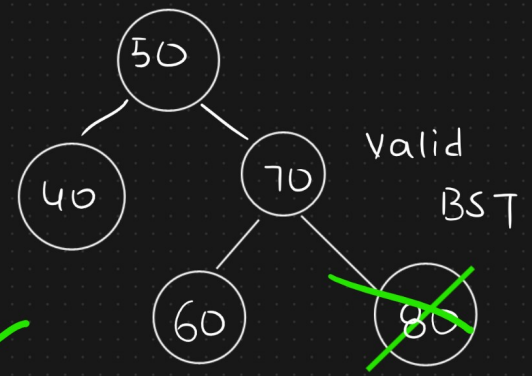
$= \Theta(n)$

$\Theta(n)$

Deletion

1) Leaf Node (No child)

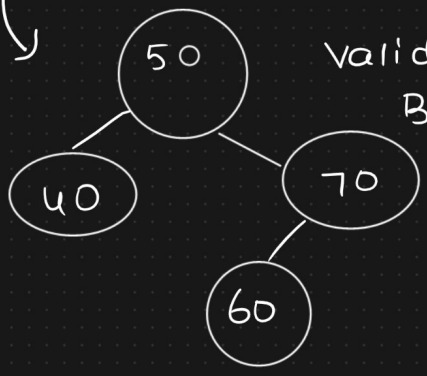
Delete 80



Valid BST

1) search (80) ✓

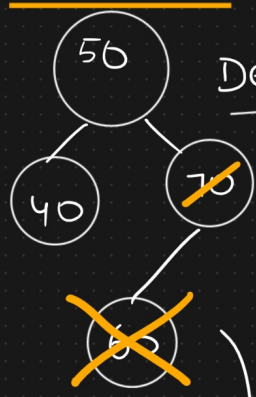
2) Delete



Valid BST

2) one child

Delete 70

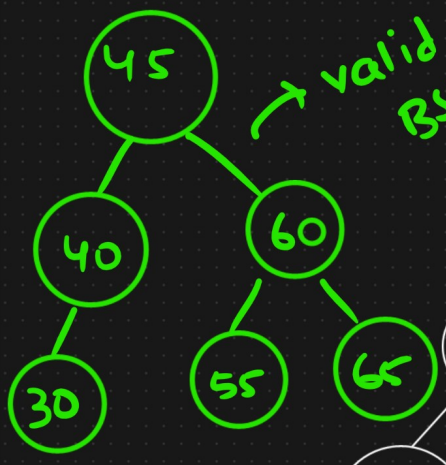


Valid BST

1) search (70) ✓

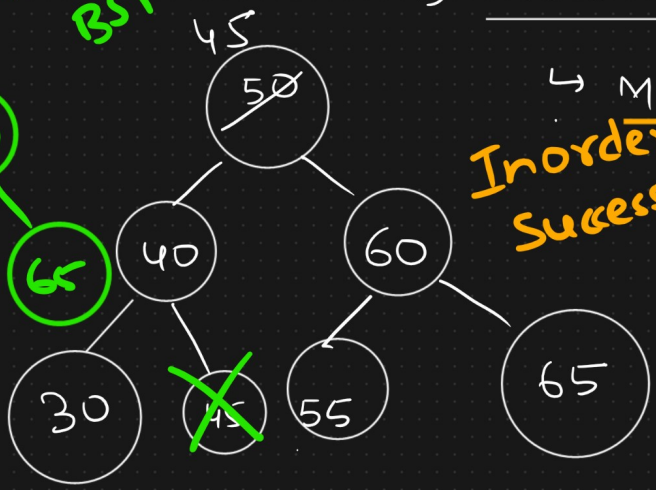
2) copying the child data & Paste it over Parent data

3) Delete single child node



valid BST

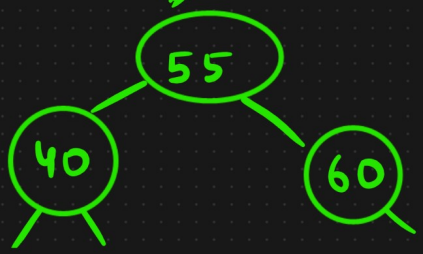
3) Delete 50



↳ Min. value from (55) Inorder Successor right subtree

OR

↳ Max. value from Left subtree (45) Inorder Predecessor



* Task → Try to implement deletion in BST
(no child,
one child,
two child)