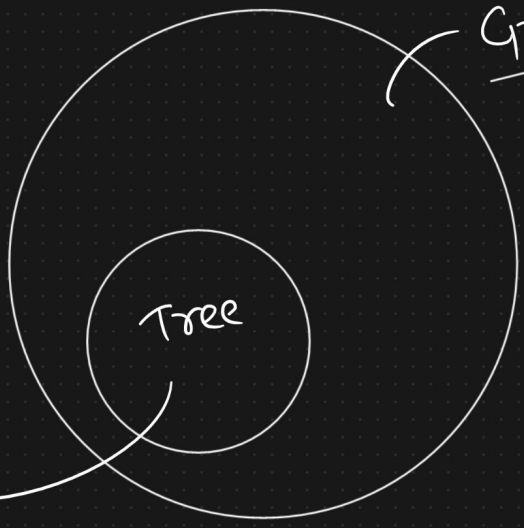
Binary Search Tree

1. Tree
2. Tree Traversal
3. Recursion

Data Structure

- Linear
  - Array, Linked List)
  - Stack & Queue
- Non-Linear
  - Tree,
  - Graph



Binary Tree

(0, 1, 2)

Atmost 2 child node

Binary Search Tree

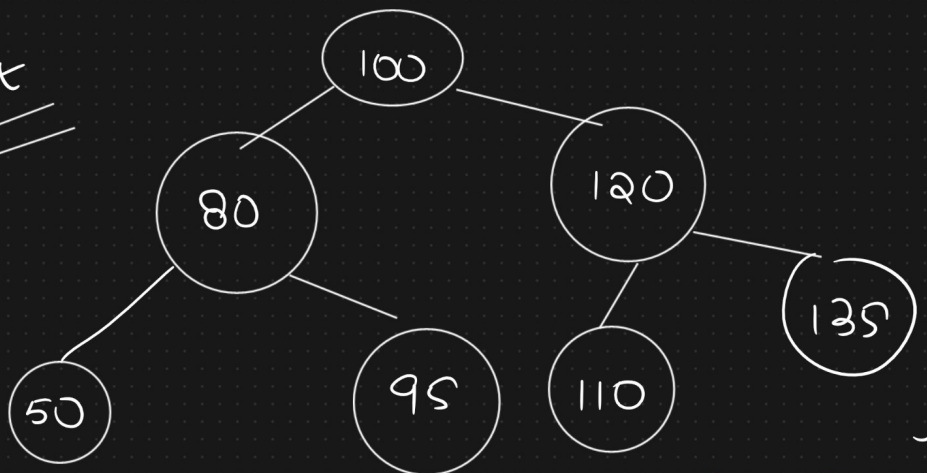
No duplicates in BST

Property

1. Left Subtree ← data < Parent node data
2. Right Subtree ← data > Parent node data

100, 80, 120, 50, 95, 135, 110

Insert



Balanced Binary Search Tree

0 1 2 3 4 5 6

50, 80, 95, 100, 110, 120, 135

3. Inorder traversal (BST)

- Left subtree (Recursively)
- root.val
- Right subtree (Recursively)

Sorted array (Ascending Order)

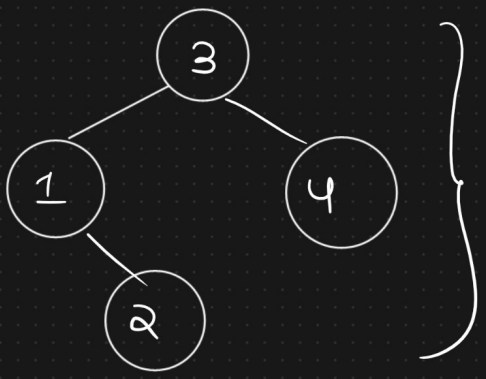
k = 3 — 3rd smallest element

k-1

k-1

result[2]

Input: root = [3,1,4,null,2], k = 1

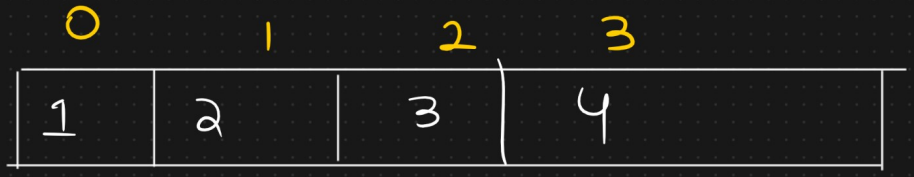


Result = 1

BST

rec(k-1)

Inorder Traversal



1. Validation BST
2. Kth Smallest finding

Inorder traversal

rec

Complexity Analysis

Time complexity => O(n)

space complexity = O(n)

$T(n/2)$ —— inOrder(node.left) —— ①

c —— res.append(node.val) — ②

$T(n/2)$ —— inOrder(node.right) — ③

Balanced BST

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

$$= O(n)$$

Imbalanced BST

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

$$= O(n)$$

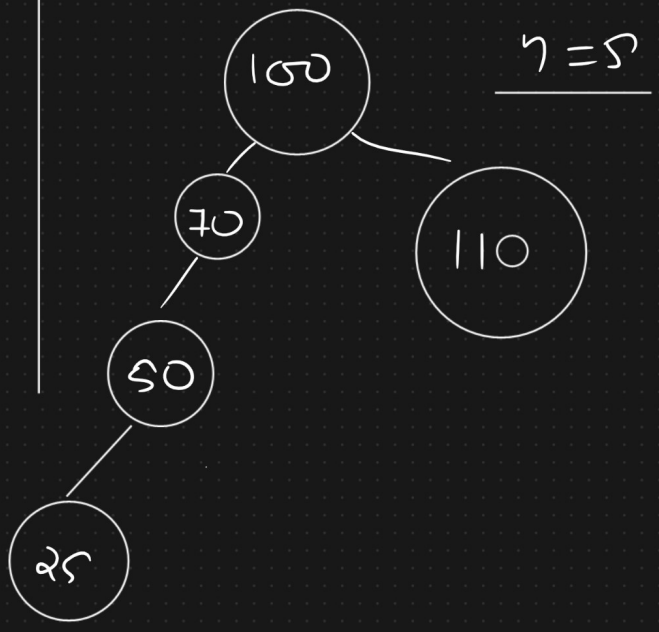
Balanced BST

↳ Almost equal num of nodes in left & right subtree

Imbalanced BST

↳ Unequal distribution of num of nodes in left & right subtree

100, 70, 50, 25, 110



n = 5

height

O(h)

Insertion

Balanced BST	Imbalanced BST
$T(n) = T(n/2) + c$	$T(n) = T(n-1) + c$
$= O(\log_2 n)$	$= O(n)$

Why??

- Searching
  - Balanced BST
  - $O(\log_2 n)$
- Realtime application

- Database
  - File Indexing
- Data Science
  - clustering