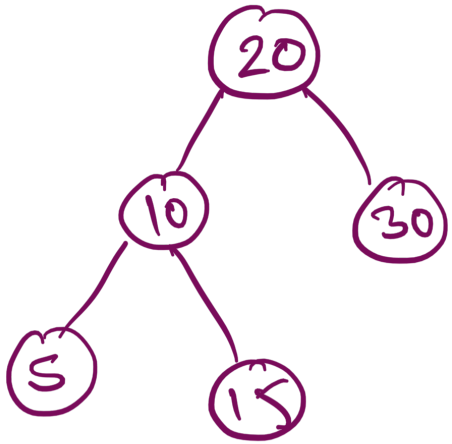
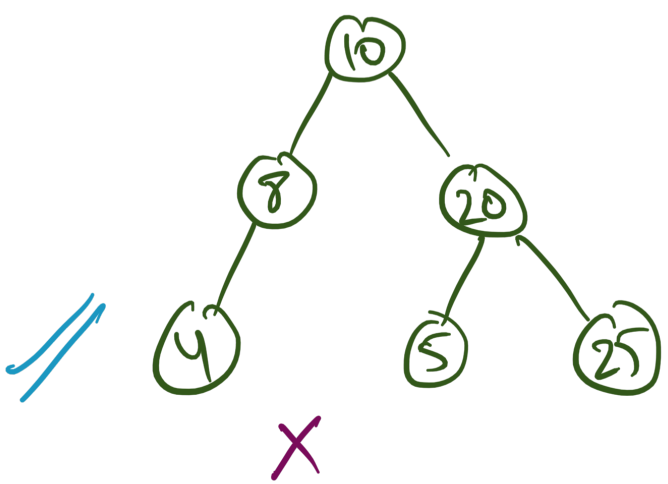
Binary Search Tree - I

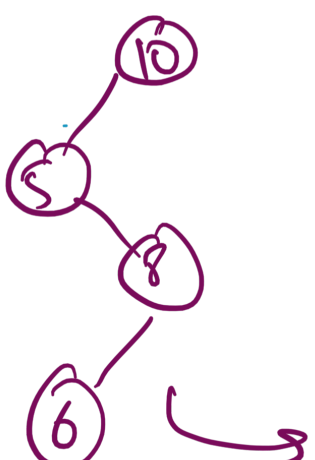


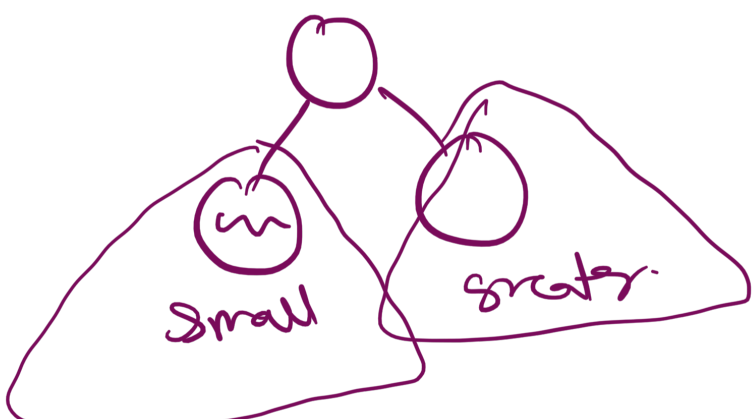
Binary Tree

A BST is a binary tree where each element in the left subtree is smaller and each element in right subtree is greater than the current node.
This is true for each node.



X

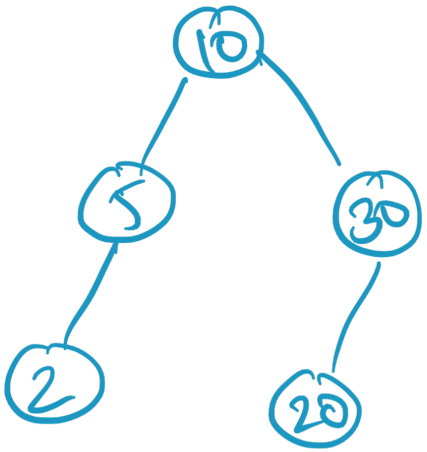




Inorder (LNR)

5 6 8 10

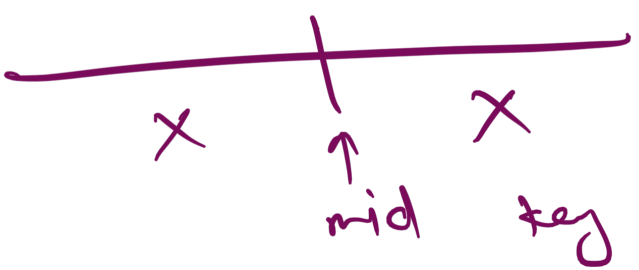
Sorted.



2, 5, 10, 20, 30

Inorder Traversal of a BST is a sorted array.

Range (4, 32)



Comparison with other Data Structures

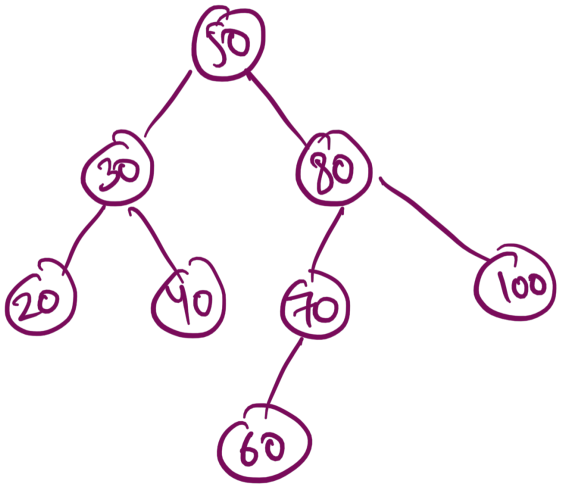
	Search	Insert	Delete
Array	O(n)	O(n)	O(n)
Linked List	O(n)	O(1) / O(n)	O(n)
HashSet	O(1)	O(1)	O(1)
BST	O(log n)	O(log n)	O(log n)

- HashSet
- TreeSet
  - ↓
  - sorted
  - BST
  - AVL Tree

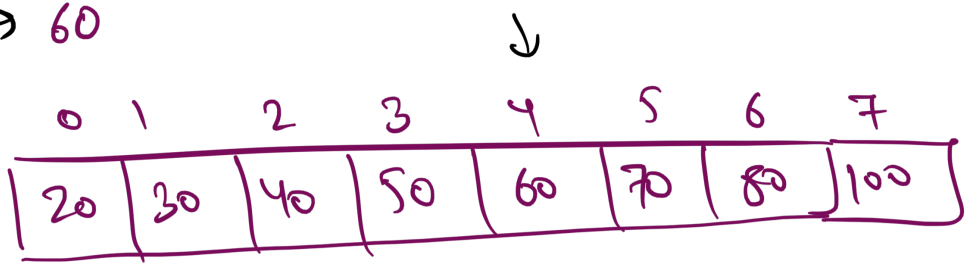
HashMap

TreeMap

Search in a BST



Key → 60



l = 0
r = 7

mid = 3

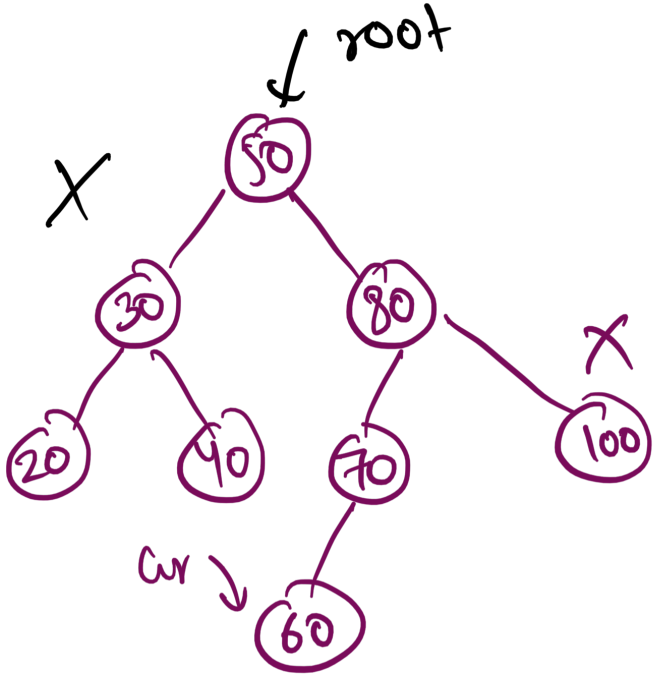
l = 4
r = 7

mid = 5

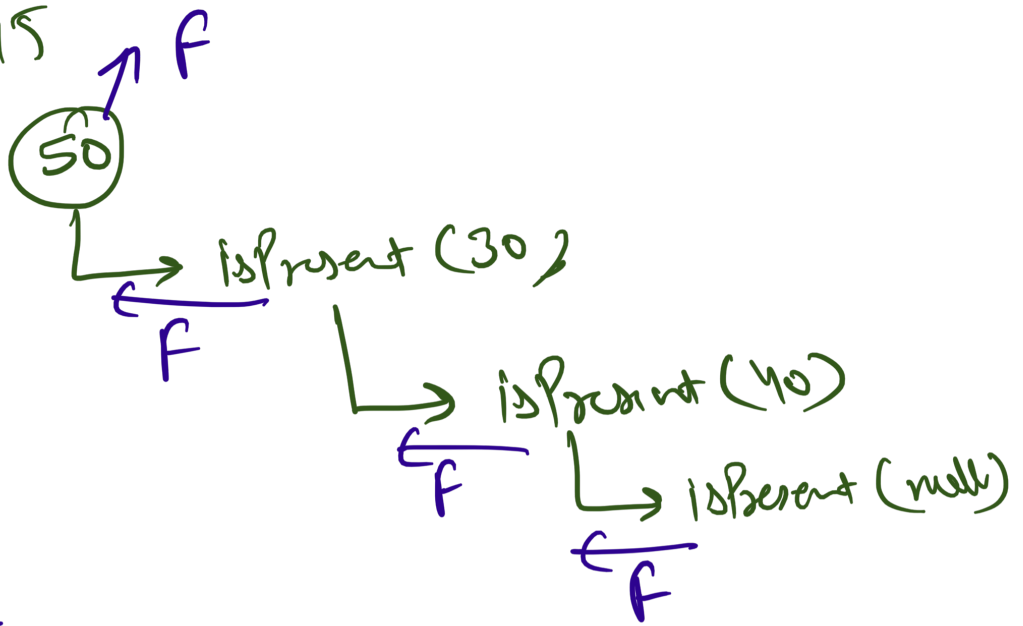
l = 4
r = 4

mid = 4

$\log_2(7)$



key = 45

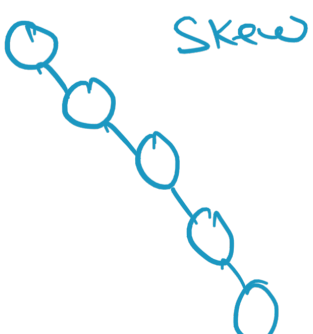


50
30
40

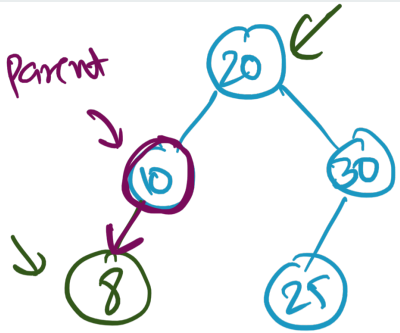
T.C. → O(log n)
S.C. → O(log n)
} Average case

n
n/2
n/4 ... 1

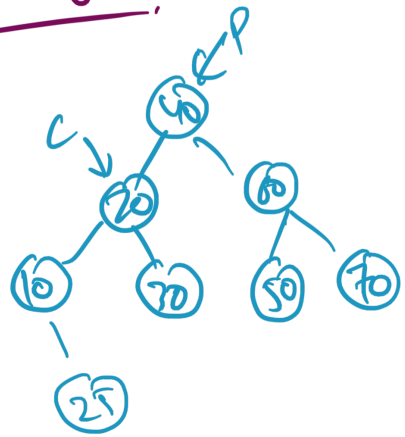
$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$



Insert in a BST



Insert will always create a leaf node

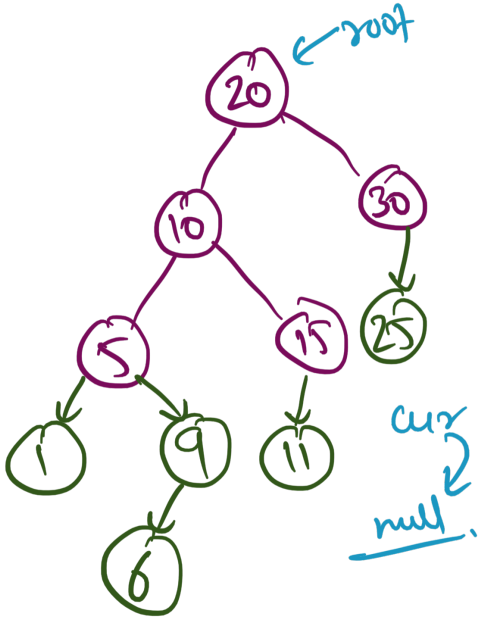


25

8

par = 11

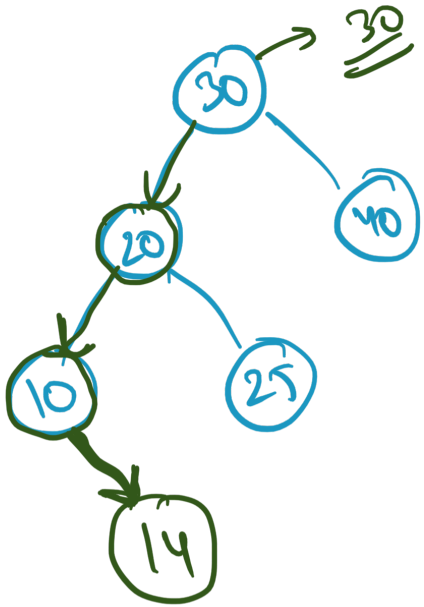
cur = null



12

cur

null



14.

insert (30)

left

insert (20)

20

left

insert (10)

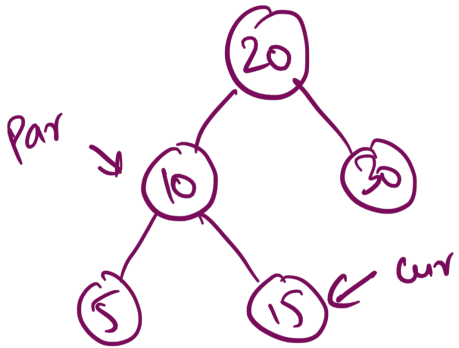
10

right

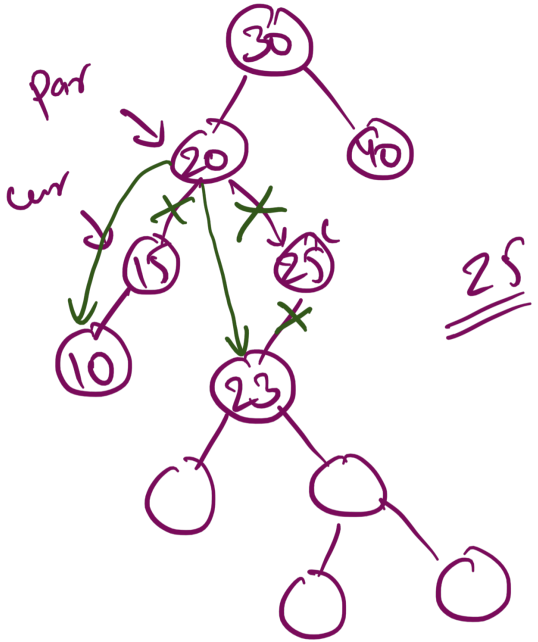
insert (null)

14

Delete in a BST

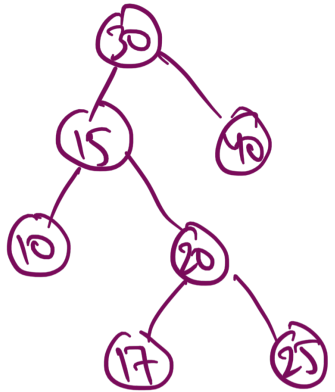


15 → (1) delete a leaf node



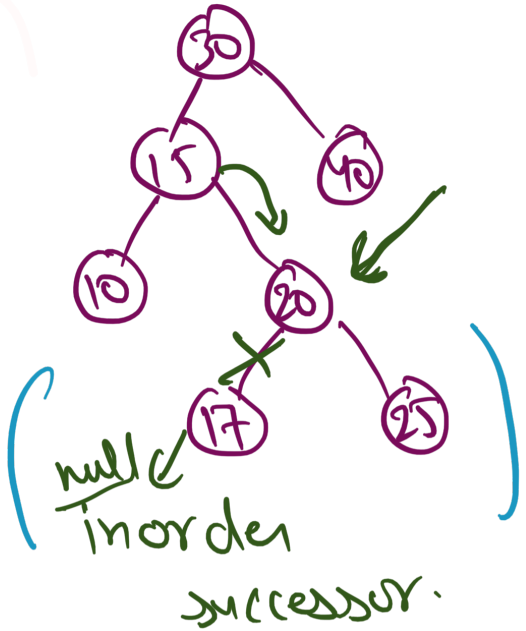
25

(2) delete with one child node

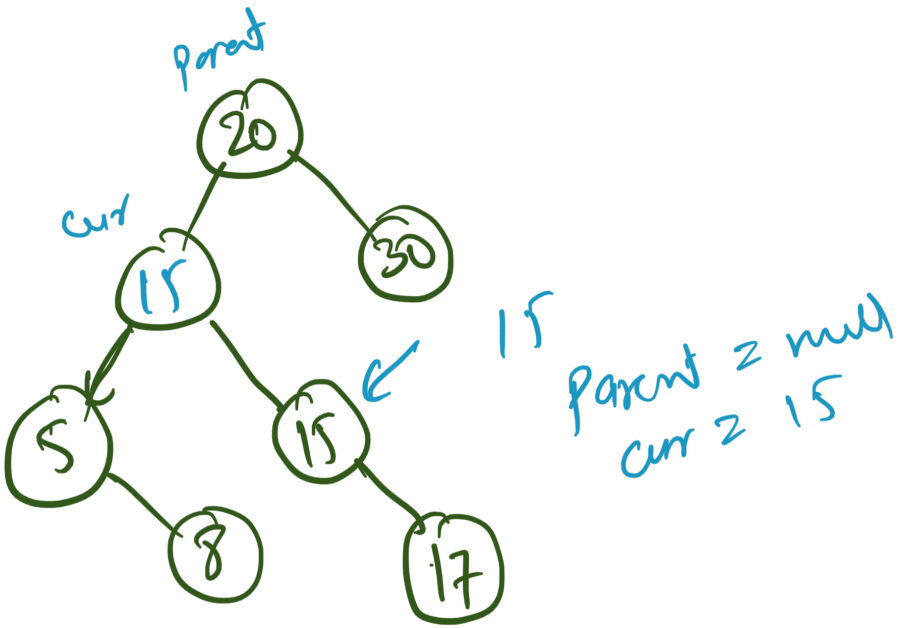


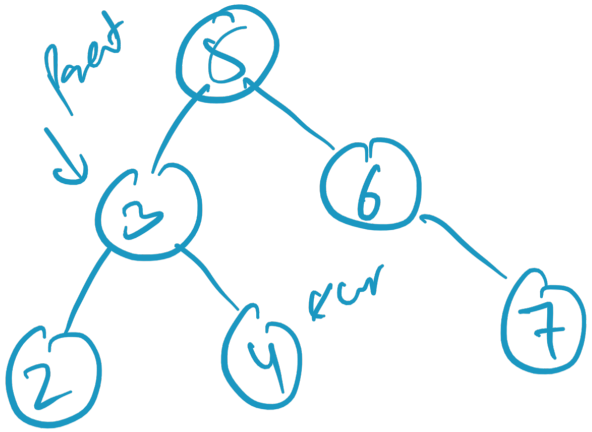
inorder successor.

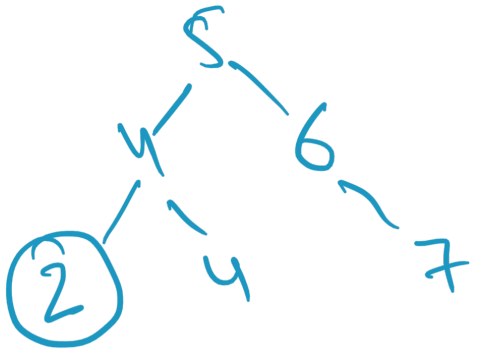
(3) delete with both children

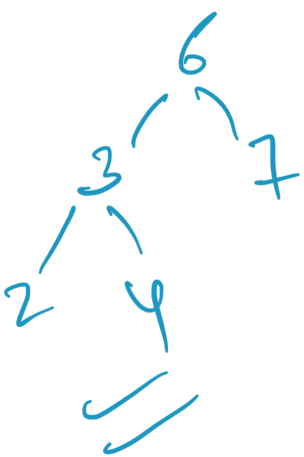

30
15
40
10
20
17
25
null
inorder
successor.


10
15
17
20, 25, 30, 40


parent
20
cur
15
30
5
15
8
17
15
parent = null
cur = 15


fast
5
3
6
2
4
rcur
7


5
4
6
2
4
7


6
3
7
2
4


5
3
6
2
4
7

Practice Problems

1. Check if a Binary Tree is BST.
2. Convert a Sorted Array to Balanced BST.
3. Check if a given array can represent Preorder Traversal of Binary Search Tree
4. Second largest element in BST
5. Print BST keys in the given range