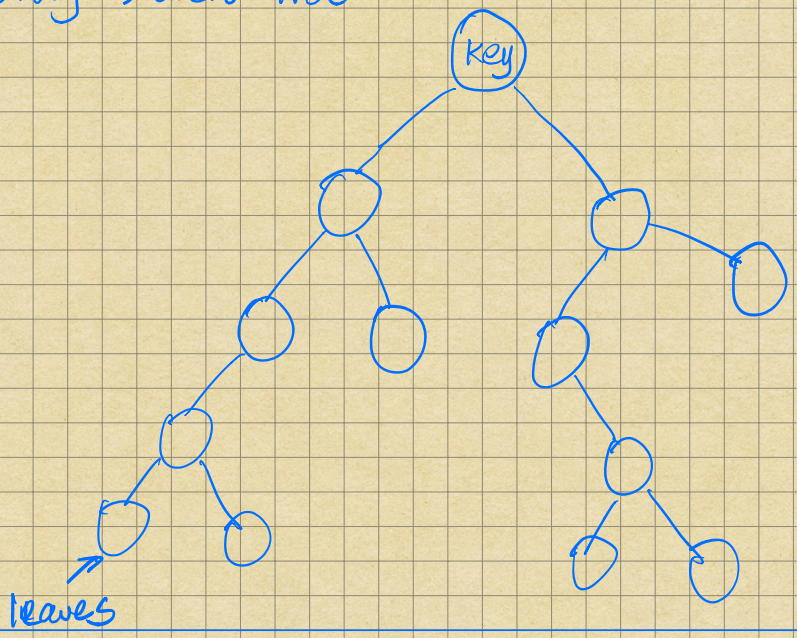
Binary Search Tree

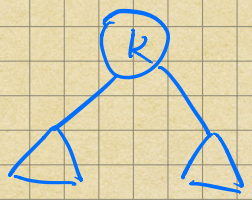


keys are comparable, for each pair of keys

$$k_1, k_2 : \begin{cases} k_1 < k_2 \\ k_2 < k_1 \\ k_1 = k_2 \end{cases}$$

Binary Search Tree : (BST)

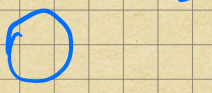
1. Binary Tree with data composed of comparable objects.
2. For each node:



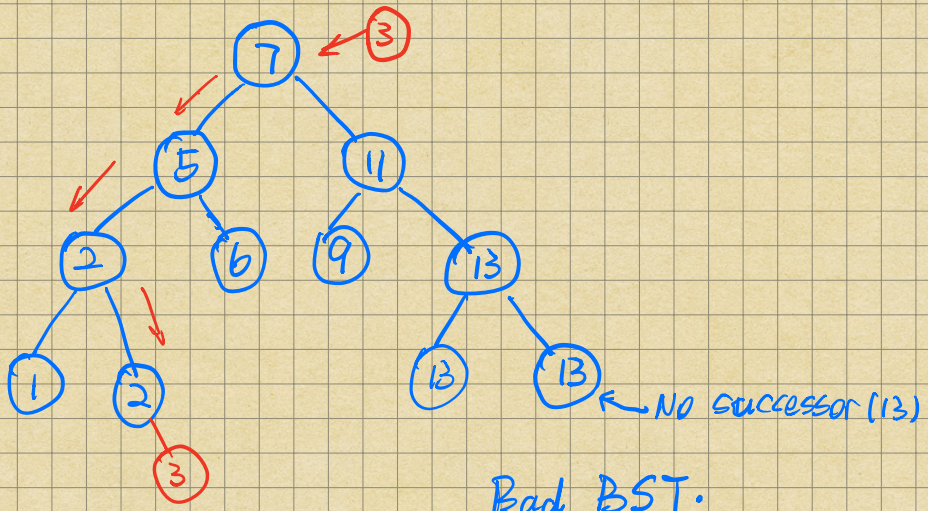
- for each node with k, every node in left sub tree has a smaller or equal keys.
- every node in the right sub tree has a larger or equal keys

3. The left and the right sub trees are also BST

4. The empty tree is a BST and a single node in BST.



ex).



(1). search

(2) insert

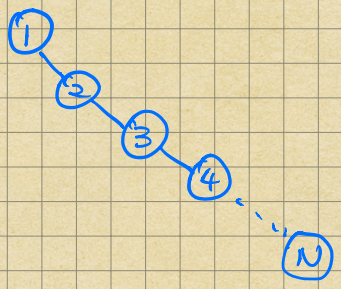
(3). delete

(4). successor / predeccessor

(5). data in a range

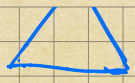
(6). Min / max

Bad BST:

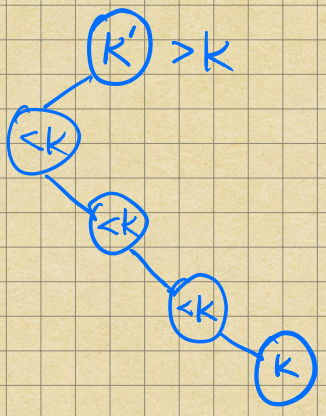


- depth = d
- search = O(d).
- insert = O(d) (may increase depth by 1)
- Min / max = O(d)
  ↳ keep going left
- successor / predeccessor:

  successor(k) = min{key : key > k}
  - Case ①: right sub tree exists



- Case ②: right sub tree doesn't exist



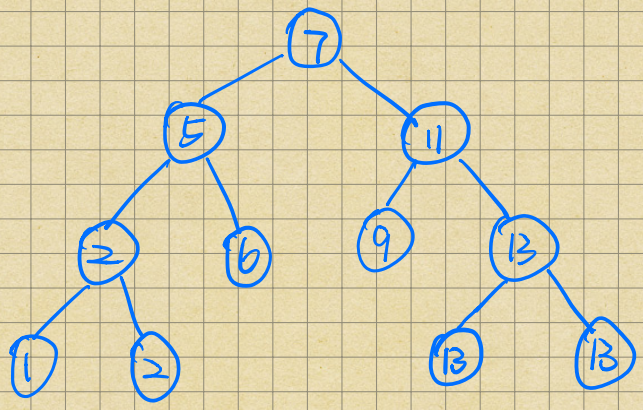
Predecessor(k) = max {key: key<k}.

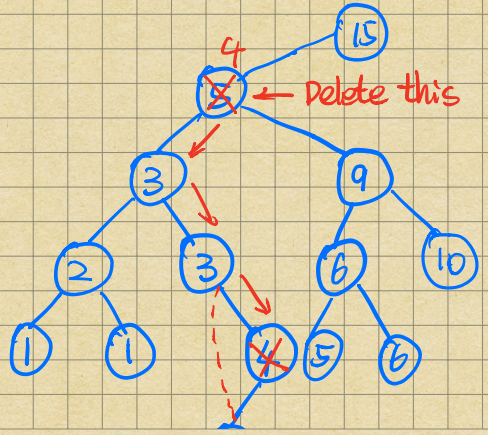
case ①:

case ②:

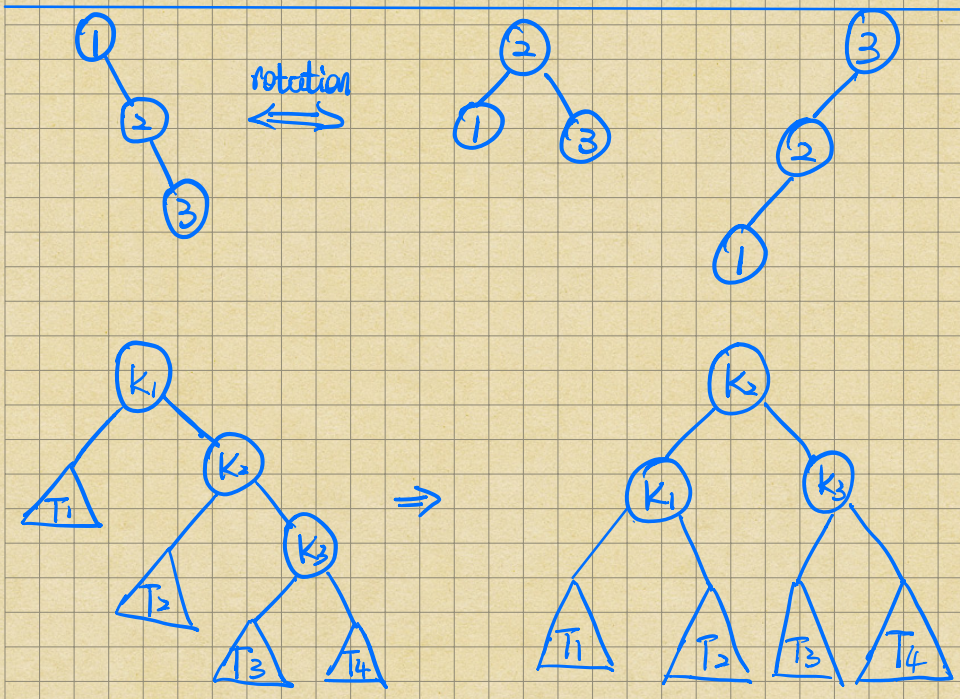
- if k = Pre(k'), then k' = succ(k)

- Delete:





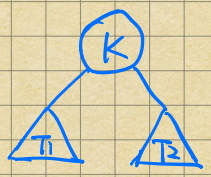
Delete is O(d), may d=d-1



$T_1 < K_1 < T_2 < K_2 < T_3 < K_3 < T_4$

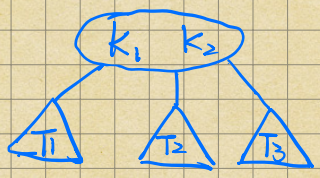
Two-Three-Trees

- Nodes: 2-node



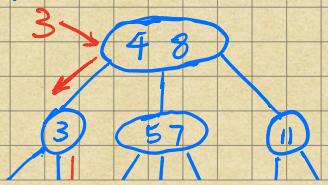
$T_1 \le K \le T_2$

3-node : $K_1 < K_2$



$T_1 \le K_1 \le T_2 \le K_2 \le T_3$

ex). 2-3-trees



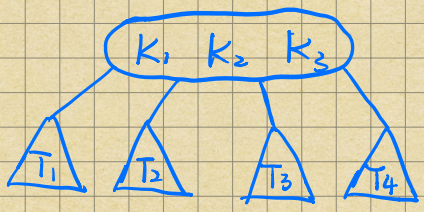
if N nodes, depth $\ge \log_2 N$

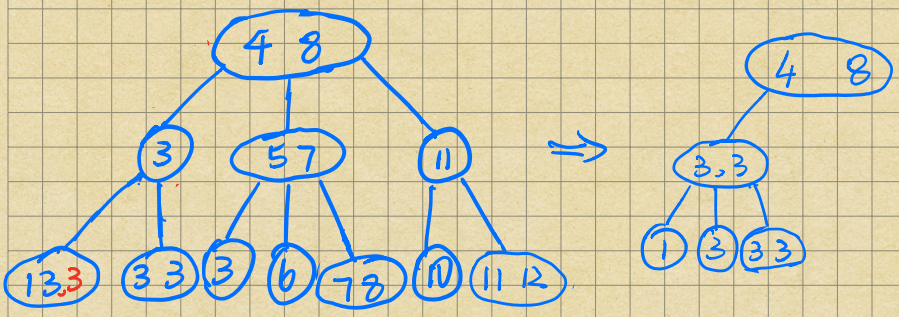
$\log_3 N \le$ depth $\le \log_2 N$

depth = O(lgN)

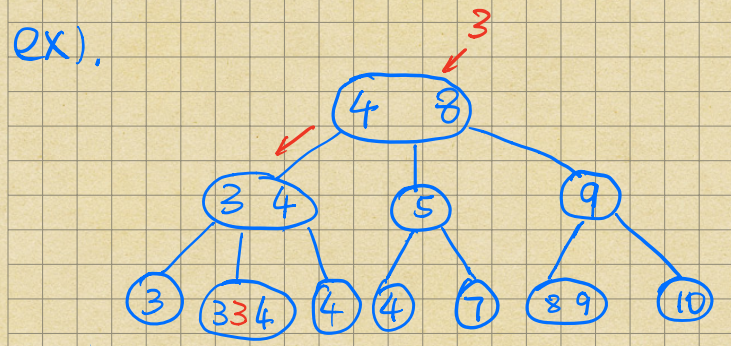
- search = O(lgN)

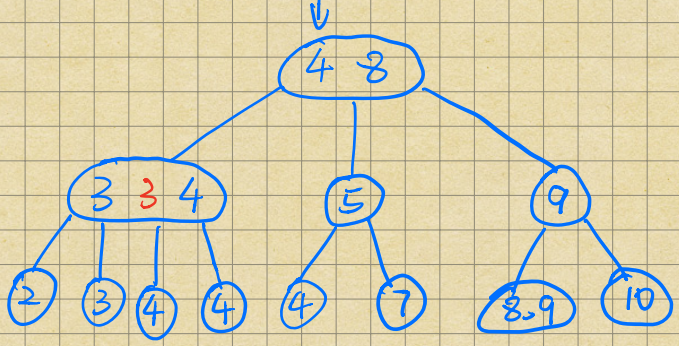
ex). 4 nodes

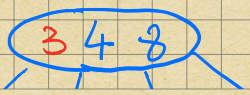


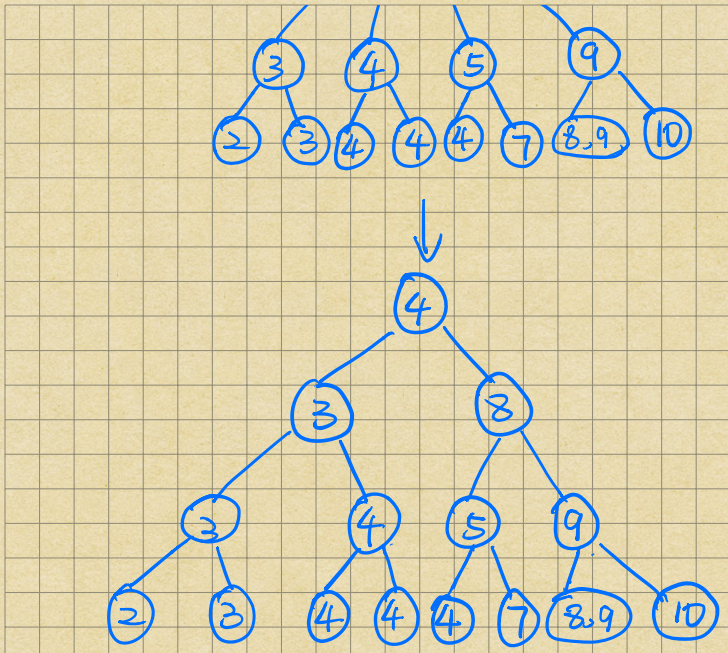


ex).



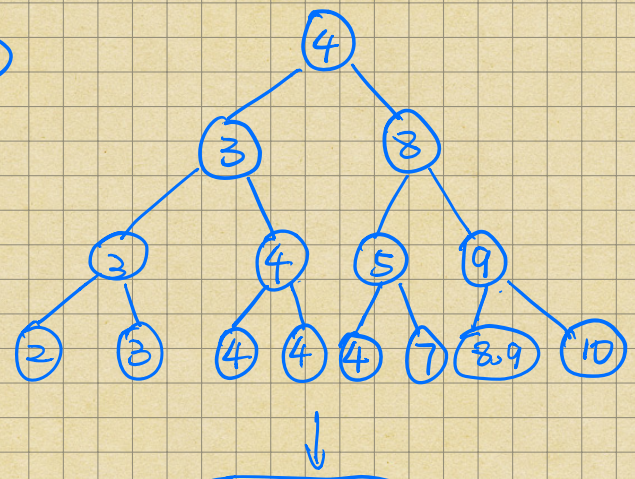


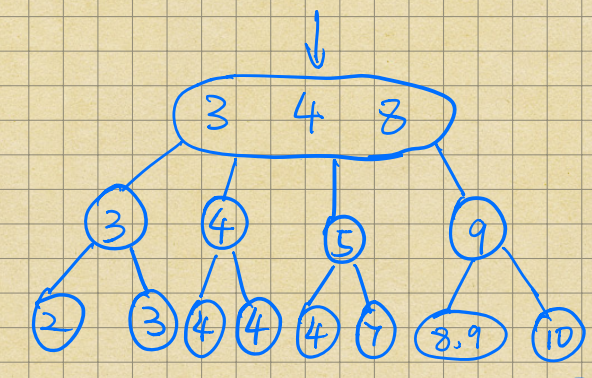




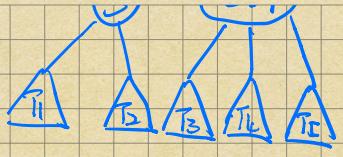
- delete min

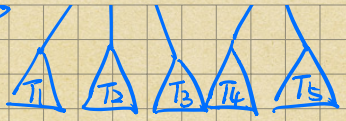
Case ①





Case ②


T1
T2
T3
T4
T5


T1
T2
T3
T4
T5