Search Trees

How fast can we search data of size n?

Ans: ▷ Without preprocessing: Select, $O(n)$

▷ With preprocessing: PQs, $O(1)$, only for max/min

▷ Sub-linear Search Time

▷ Binary Search Tree

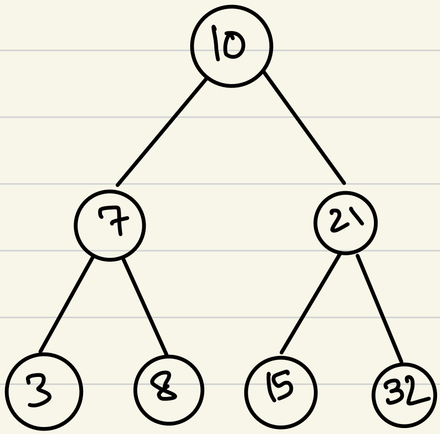
For a binary tree T,

- ▷ Root: T.root
- ▷ All internal nodes have degree $\in \{1, 2\}$
- ▷ Leaf nodes have degree 0
- ▷ $\forall$ node $x \in T$,
  - ▷ $x.p \longrightarrow$ parent
  - ▷ $x.l \longrightarrow$ left child
  - ▷ $x.r \longrightarrow$ right child
- ▷ Total no. of nodes $= n =$ T.size

watch out! Problematic calculation

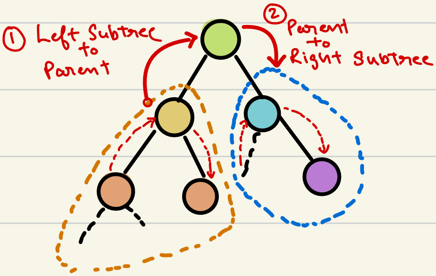
$\therefore$ We define Binary search tree (BST) as,

$$\boxed{\begin{gathered}\forall \text{ node } y \in T, \ x \in \text{left}(y),\ z \in \text{right}(y) \\ x.\text{key} \leq y.\text{key} \leq z.\text{key}\end{gathered}}$$

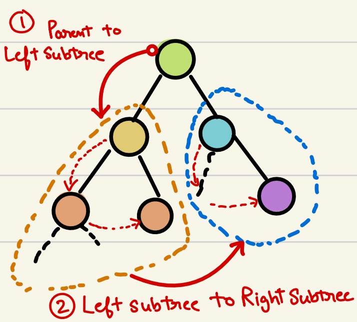


Tree Traversal

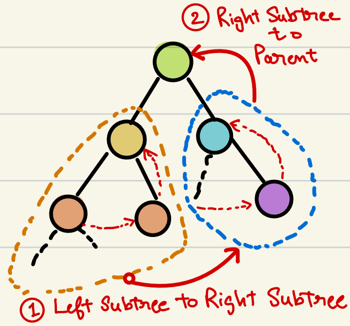
Inorder



Preorder



Postorder



▷ Dictionary Data Structure

Key Operations

- ▷ Min (T)
- ▷ Max (T)
- ▷ Insert (T, v) → Value to be inserted
- ▷ Delete (T, x) → Pointer to node
- ▷ Pred (T, x) : Predecessor
- ▷ Succ (T, x) : Successor
- ▷ Search (T, v) : Find $x \in T$, s.t., $x.\text{Key} = v$

▷ BST as Dictionary

- ▷ Fetch Min/Max :

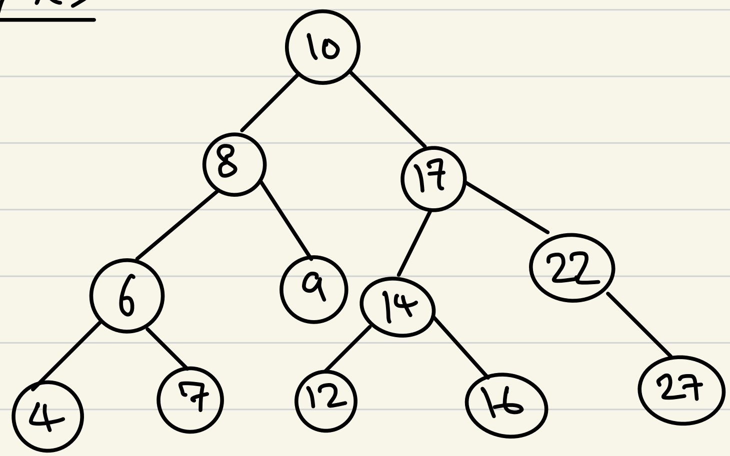
  Worst : $O(n)$, Best : $O(\log n)$

Instead of "n", we can use height "h" to better assess runtim.

$$\boxed{T_{Min/Max} = O(h) \;;\; \log n \le h \le n}$$

▷ $T_{search} = O(h)$ | If we're unlucky, $h \to h+1$

Succ(T, x)



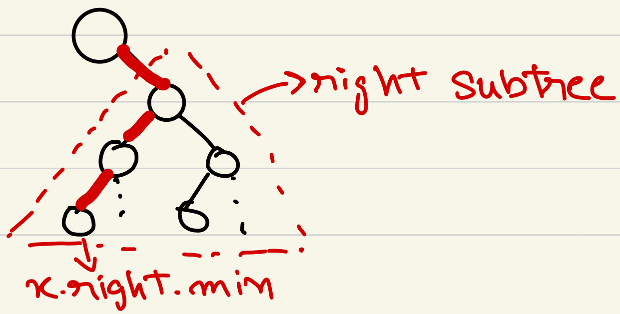
▷ Succ(6) = 7

▷ Succ(9) = 10

▷ Succ(10) = 12

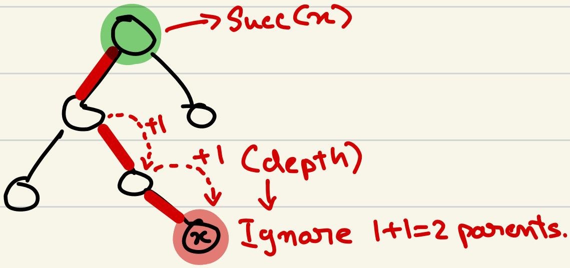
Case 1 | x.right ≠ null

return x.right.min



Case 2 | $x.right = null$

Return the larger parent.



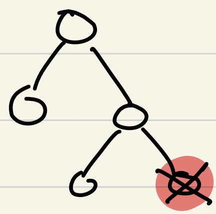
$$\boxed{T_{succ} = O(h)}$$

Delete(x)

Case 1 | $Deg(x)=0$ (Leaf node)

▷ Delete the node x.

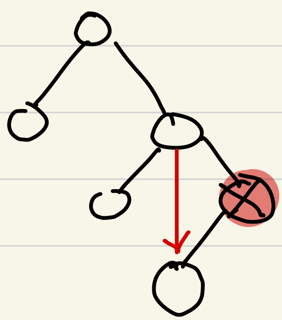
▷ $T=O(1)$



Case 2 | $Deg(x)=1$

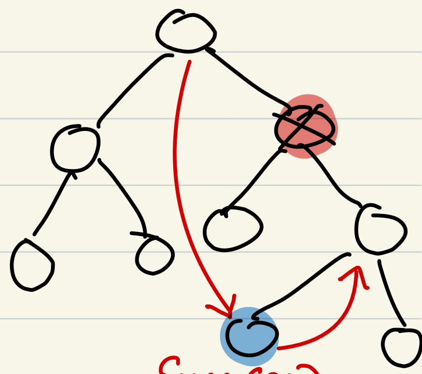
▷ Delete the node x.

▷ $x.parent \longrightarrow x.child$



Case 3 | Deg(x) = 2

- ▷ $y = Succ(x)$
- ▷ x.parent → y
- ▷ y.right → x.right
- ▷ Delete "x"
- ▷ $T = O(h)$



▷ BST SORT

```
BST-Sort(A, n)
    T ← init BST
    for i = 1 to n
        Insert(T, A[i])
    end for
```

Generate BST

$A \longleftarrow$ InorderTraversal(T)

return elements in sorted order

Runtime

$$\underbrace{T_{BST\text{-}Sort} \leq O(n)}_{\text{Inorder Traversal}} + \underbrace{n * T_{Insert}}_{\text{Generating BST}} = O(n \cdot h)$$

where,

$$\boxed{\begin{array}{l} \triangleright \text{ Worst Case: } h = \Omega(n) \\ \therefore T_{BST\text{-}Sort} \geq O(n^2) \end{array}}$$

Can we do better than $O(n^2)$?

Ans: Yes! Insert elements in random order

The implication is that root value will have rank close to $n/2$.

▷ Strong correspondence b/w rand-root & rand-pivot from rand-QS

$$E[T_{worst\text{-}case}] = T_{Rand\text{-}QS} + O(n)$$

$$\boxed{E[T_{\text{worst-case}}] = \Theta(n\log n)}$$

$$\boxed{E[\text{Height} \mid n] = O(\log n)}$$ → refer book for proof

Application

- Useful for dictionary with random order insertion
  - Search = $O(\log n)$

Essentially, a BST is more often balanced than not.

▷ 2-3 TREE

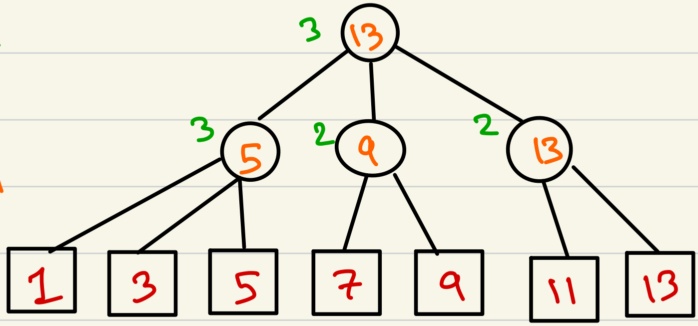
Tree T where:

- Every internal node has degree $\in \{2,3\}$ (no degree 1)
- Internal nodes contain the max value in their subtree.
- All leaf nodes' data in sorted order at same height "h".

Degree

Data

Extra



▷ $\forall$ internal node x,

$x.max$ = maximum leaf value in sub-tree

Lemma: $\forall$ valid 2-3 tree T

$$\log_3(n) \leq h_T \leq \log_2(n)$$

s.t. n = no. of leaf nodes

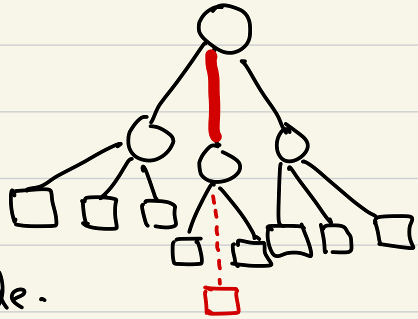
$\Rightarrow h = \Theta(\log(n))$ for worst case

$\therefore$ All operations are done in $O(\log n)$

Search

$T(n) = O(\log n)$

Insert

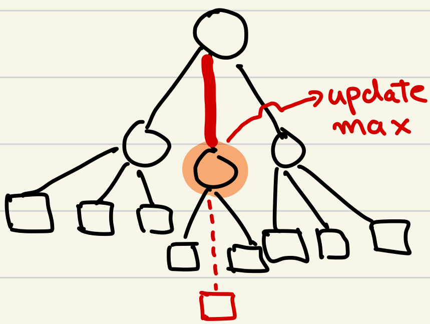


① Find the correct location for insertion using search.

② Insert the value as new node.

③ Adjust the tree

Case 1 | Less than 3 leaf nodes at target internal node

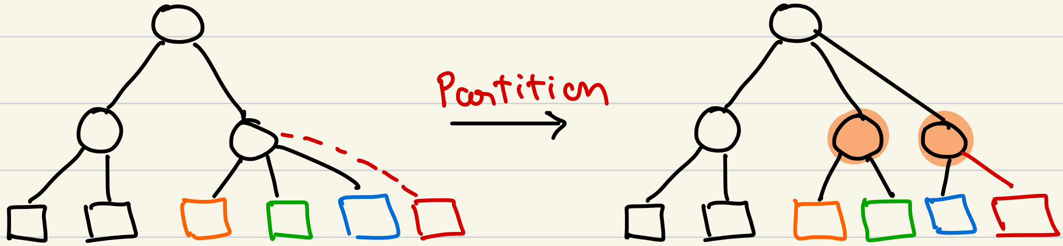


1. Simply insert new leaf node
2. No adjustment needed.
3. Update max value across all parent nodes.

Case 2 | 3 leaf nodes at target internal node

1. Insert new node
2. Replace parent node with 2 nodes, with the left one getting 2 left-most children & right one getting the rest.

3. Update all parent values



$$\boxed{T(n) = O(h) = O(\log n)}$$

▹ If we split root, create a parent node, and make it the new root.

Delete

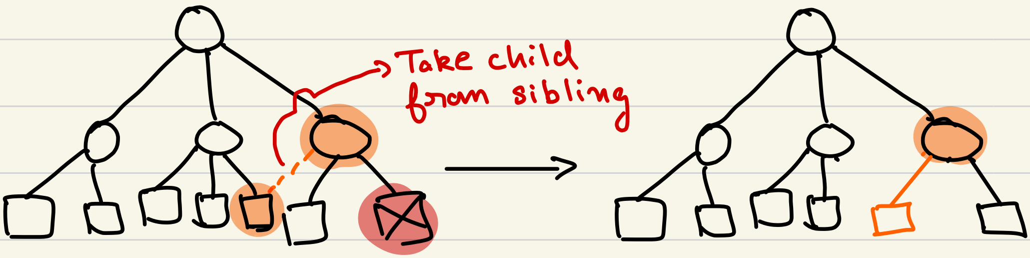
1. Delete leaf node
2. Update parent values.

Case 1 | Target's parent has degree 3

Follow the same as above

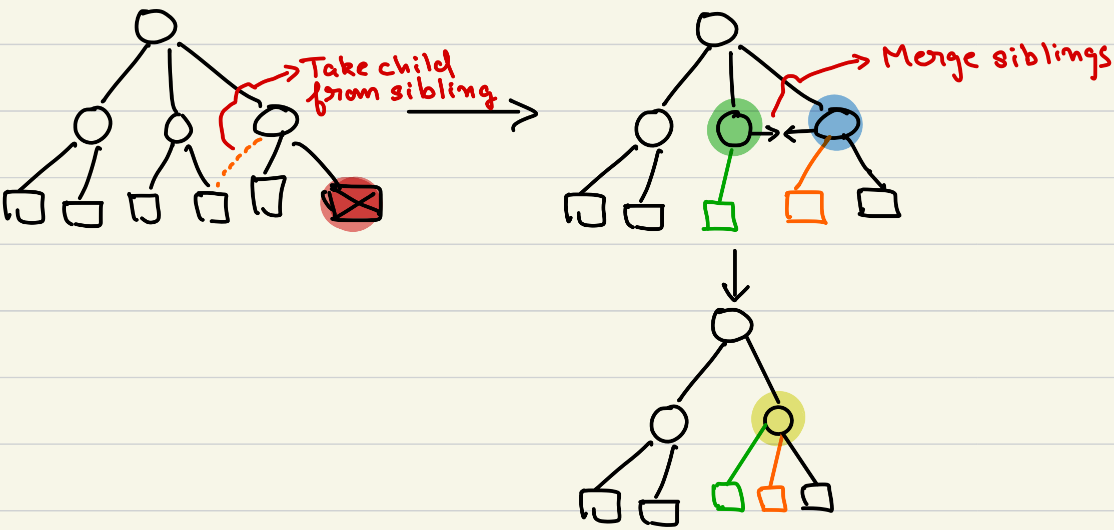
Case 2 | Target's parent has deg=2, immediate sibling has deg = 3

1. Delete leaf node
2. Take the closest child of imm. sibling
3. Update all parent values



Case 3 | Target's parent has deg=2, immediate sibling has deg=2

1. Delete leaf node
2. Take the closest child of imm. sibling
3. Merge the siblings
4. Repeat 1-3 for parents until sibling has deg=3
5. If root has a single child, delete it, make child root.
6. Update parent values.



▷ AUGMENTED DATA STRUCTURE

▹ Store useful things in internal nodes which help answer more complex queries.
Tradeoff: High cost of insertion/deletion.

For eg: Maintaining number of nodes in the current sub-tree, in the root node of the sub-tree

▹ Allows quick reference for queries related to num. of nodes.

▹ Maintenance cost is trivial

▹ Applications: Rank of element, K^{th} smallest element