Recitation 13

Coming Up

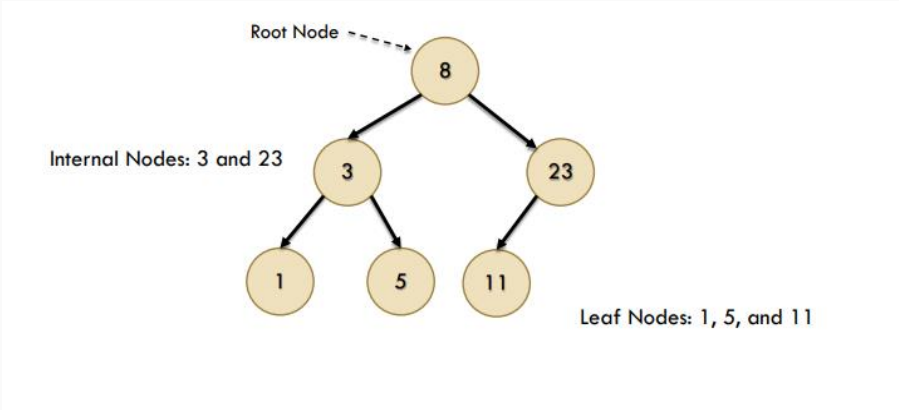
- HW4 is due on May 6th, 2026

Binary Search Trees

- Functionality
  - Search
  - Insert
  - Remove
- Properties
  - Depth

Overview

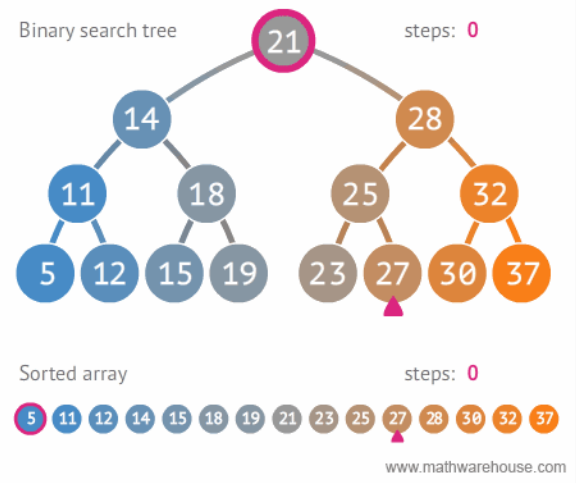
- Dynamic, Hierarchical (Tree) Data Structure
  - Designed to be in-memory
- Nodes are organized following a strict ordering
  - Left subtree - contains only nodes with values less than the node's value
  - Right subtree - contains only nodes with values greater than the node's value
- Each node is permitted 2 children (binary)



Search

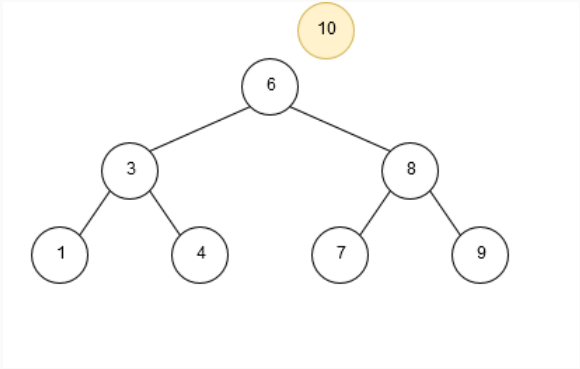
- O(log n) - average case
- O(n) - worst case

- Idea: Starting at the root, repeatedly compare the target node to the current node. Move left/right depending on if the target node is larger/smaller.
  - Keep doing this until the value is found or a null pointer is reached.



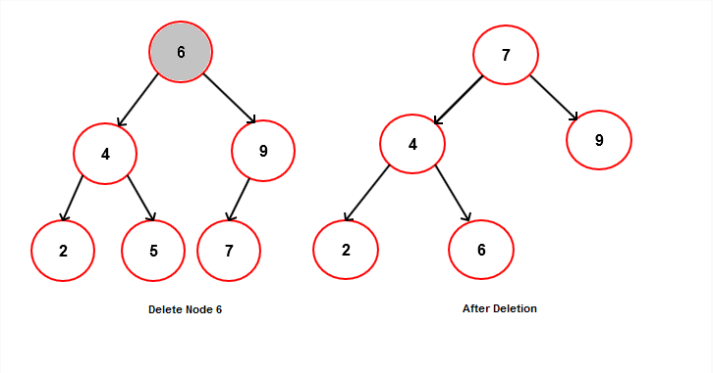
Insert

- O(log n) - average case
- O(n) - worst case
- Idea: Starting at the root, repeatedly move left or right based on comparisons between the target node and the current node. Once a null pointer is reached, insert the node there while adhering to BST rules.



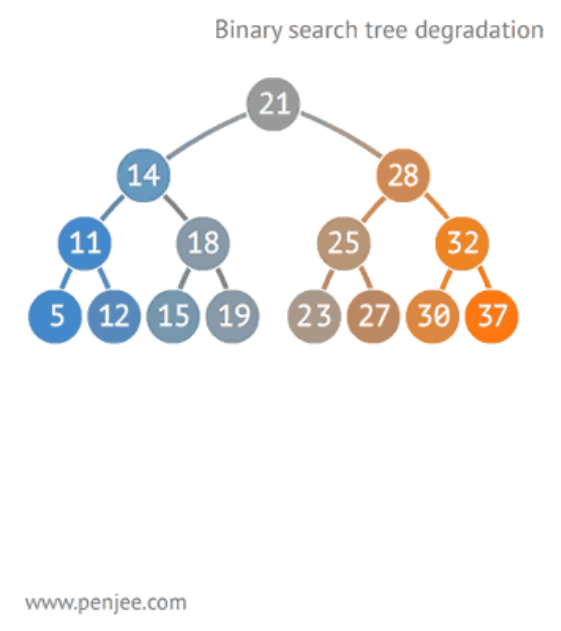
Delete

- O(log n) - average case
- O(n) - worst case
- Three Cases: Node has...
  - No children – simply remove the node
  - One child – replace it with the its child (link the parent and the child together).
  - Two children – replace it with its predecessor/successor



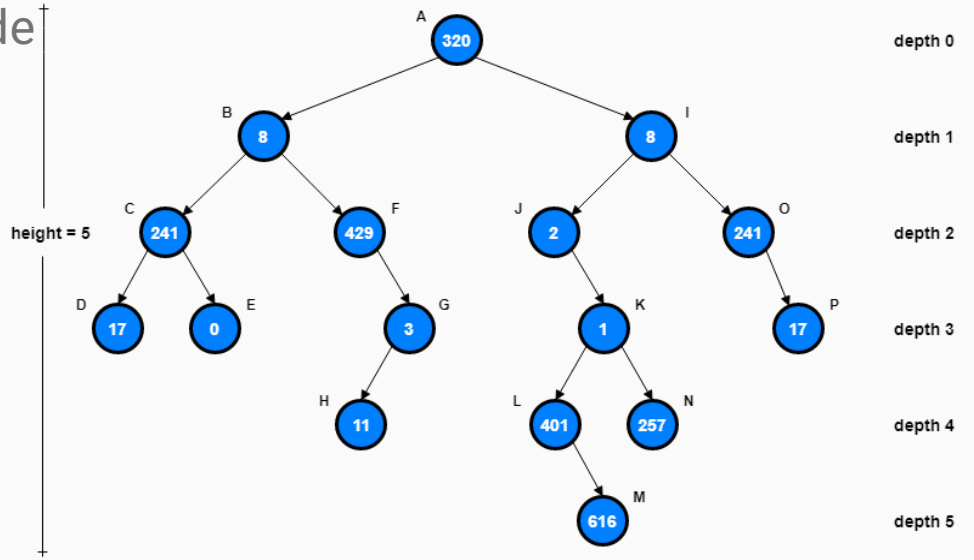
Degradation of BST Tree

- Time complexity is directly influenced by the height of the tree.
- Balanced BST – O(log n)
- Skewed BST – O(n)
  - Basically turns into a LinkedList



Depth

- Number of edges from root to node
- Root = depth 0



Recitation 13 Activity

- Worksheet located on canvas.