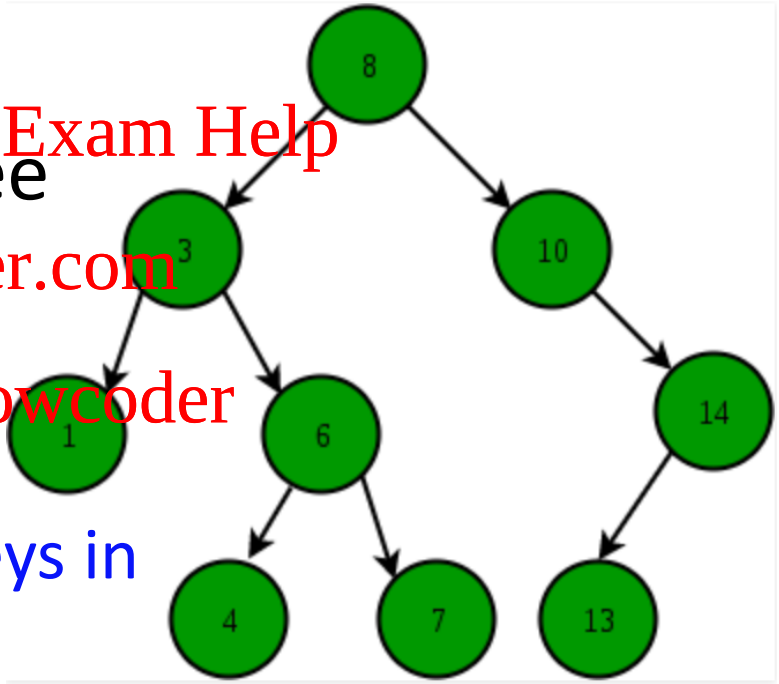
BST & AVL

Juan Zhai

juan.zhai@rutgers.edu

Inorder traversal (left, root, right)

1. Traverse the left subtree
2. Visit the root
3. Traverse the right subtree

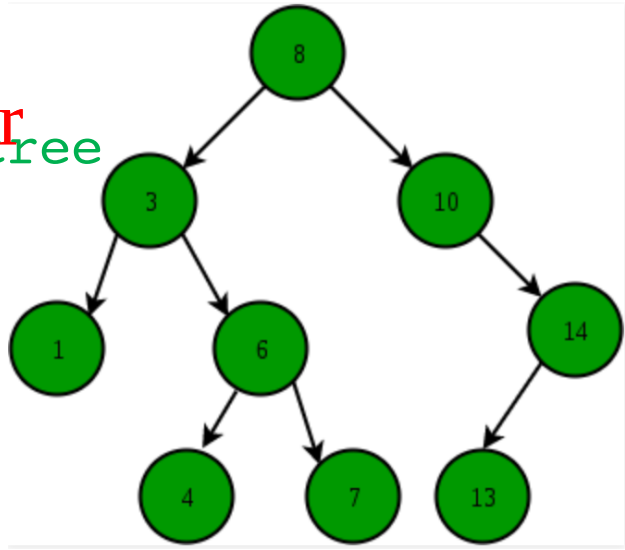


1, 3, 4, 6, 7, 8, 10, 13, 14

Inorder traversal prints all the keys in ascending order.

Inorder traversal (left, root, right)

```
public void inorder(Node root) {
        //check if bst is empty
        if(root == null)
          return ;
        //firstly recur on the left subtree
        inorder(root.left);
        //then print the current node
        System.out.println(root.key);
        //lastly recur on the right subtree
        inorder(root.right);
}
```

	Running time	
	Best	Worst
Traversal Nodes	O(n) [1]	O(n) [1]

[1] Must visit each node, assuming O(1) time per visit

Tree sort

- Tree sort is a sorting algorithm that is based on binary search tree data structure.
- First creates a binary search tree from the elements of the input set via insertion
- Then performs an in-order traversal on the created binary search tree to get the elements in ascending order.
- Sakai code

Tree sort

- Worst case: skewed tree
  - Insert n elements to form a tree:
    - Adding the comparison times of inserting each node:
      0+2+4+6+…+2(n-2)+2(n-1)
      = 2*(1+2+3+…+(n-2)+(n-1))
      = 2* (1+(n-1))*(n-1)/2
      = n*(n-1)
    - $O(n^2)$
  - Inorder traversal:
    - O(n) for all kinds of BST
  - $O(n^2) + O(n) \rightarrow O(n^2)$

Handle duplicates

- Sol 1.1: Left subtree has keys less than or equal to key in root
- Sol 1.2: Right subtree has keys greater than or equal to key in root
- Return first match
- Return all matches: *Traverse from root to leaf*

- Sol 2: At each node, maintain a singly linked list of all objects with the same key.
  - Hit the key, then get all the objects with the same key

AVL Tree

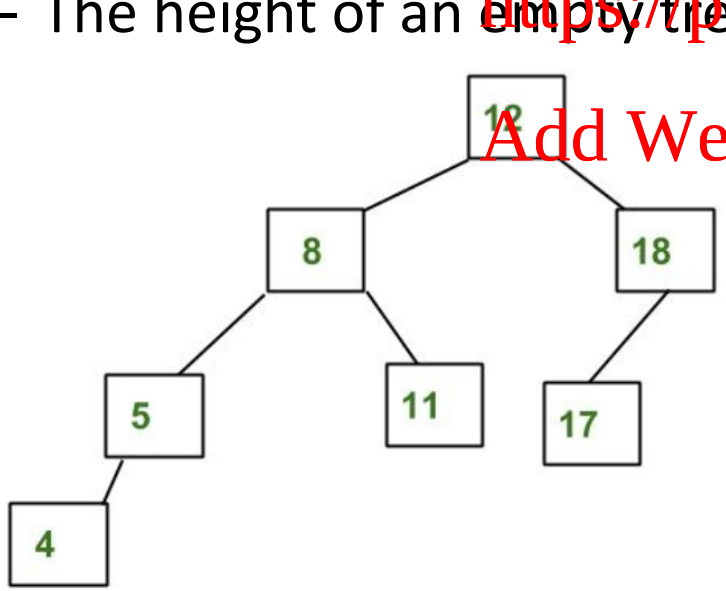
Self-balancing Binary Search Tree

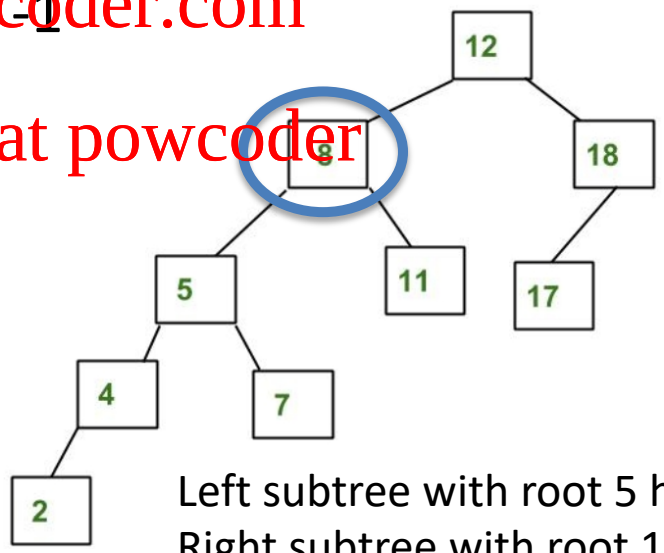
Skewed Binary Search Tree

- Search/insert/delete: worst case O(n)
- To maintain the O(log *n*) time, the height of the tree needs to be maintained at O(log *n*).
- To achieve the balanced condition, the binary search tree would have to be rebalanced after each insertion or deletion.
- Balanced binary search tree:
  - AVL tree, name after its inventor
  - Red-black tree

AVL Tree

- AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.
  - Height: the maximum level at which there is a node
  - The height of an empty tree is -1





Left subtree with root 5 has height 2
Right subtree with root 11 has height 0

AVL Tree: Recursive definition

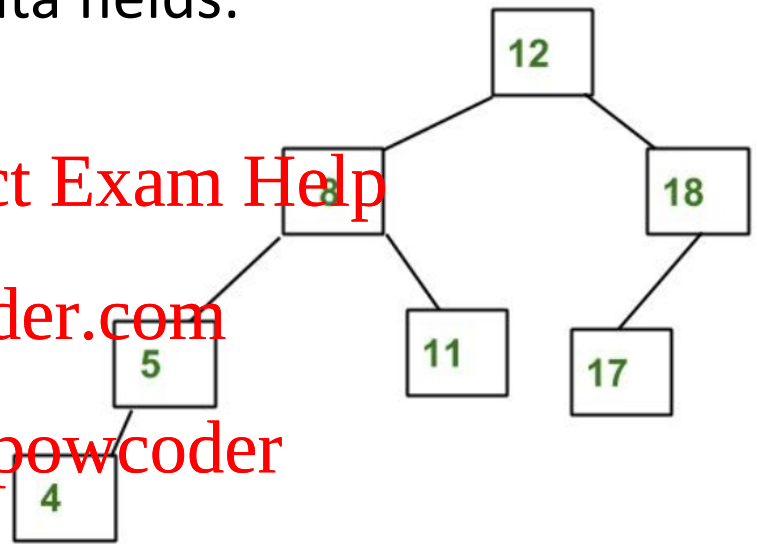
- An AVL tree is a binary search tree in which the left and right subtrees of the root are AVL trees whose heights differ by at most 1

Balance factor

- An AVL tree needs to keep track of the difference in the heights of the subtrees of each node.
- Associate a balance factor with each node.
  - Equal high, '-': the left and right subtrees of the node are of equal height
  - Right high, '\': height of the right subtree of the node is one more than that of its left subtree
  - Left high, '/': height of the left subtree of the node is one more than that of its right subtree

ALV Node

- AVL node holds the following data fields:
  - data
  - left
  - right
  - parent
    - 12 is the parent of both 8 and 18
  - bf
    - 4, 11, 17 have '-'
    - All others have '/'
  - height: the maximum level at which there is a node
    - 4, 11, 17: 0
    - 5, 18: 1
    - 8: 2
    - 12: 3



Search

- Searching proceeds as in any binary search tree since a AVL tree is a binary search tree.