Data Structures and Algorithms
CS-206

Binary Tree

Instructor
Dr. Maria Anjum
Assistant Professor

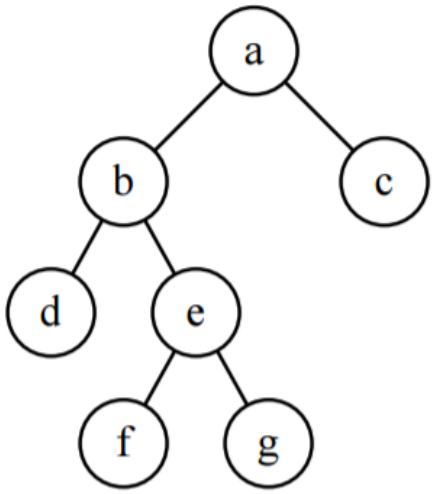
Department of Computer Science
Lahore College for Women University

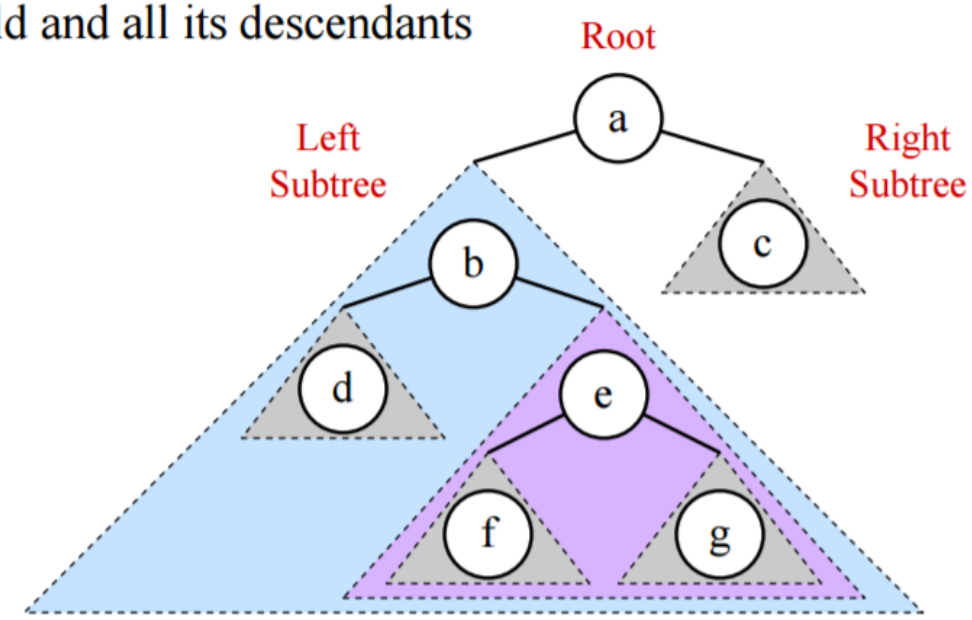
Topics Covered

- Trees
- **Binary Tree**
- Balanced Trees
- AVL Trees
- Spanning Trees

Binary Tree

- A **binary tree** is a tree in which
  - Every node has at most 2 children, called the **left child** and **right child**
- A binary tree can be defined recursively as
  - **Root node**
  - **Left subtree:** left child and all its descendants
  - **Right subtree:** right child and all its descendants





Binary Tree Cont.

- A binary tree is made of nodes, where each node contains
  - a "left" pointer,
  - a "right" pointer, and
  - a data element.
- The "root" pointer points to the topmost node in the tree.
- The left and right pointers recursively point to smaller "subtrees" on either side.
- A null pointer represents a binary tree with no elements -- the **empty tree**.

Binary Tree Recursive Definition Cont.

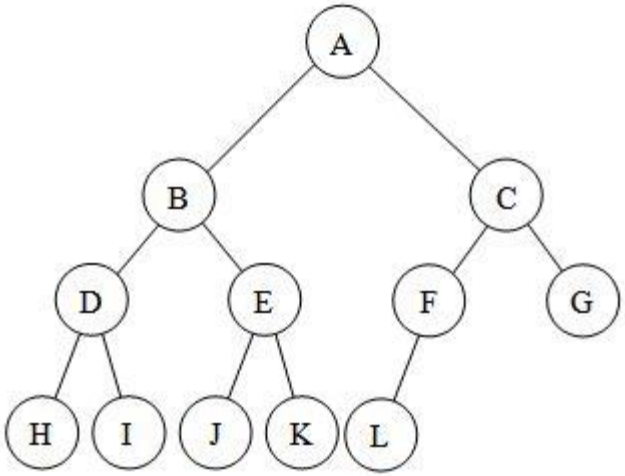
The formal recursive definition is:

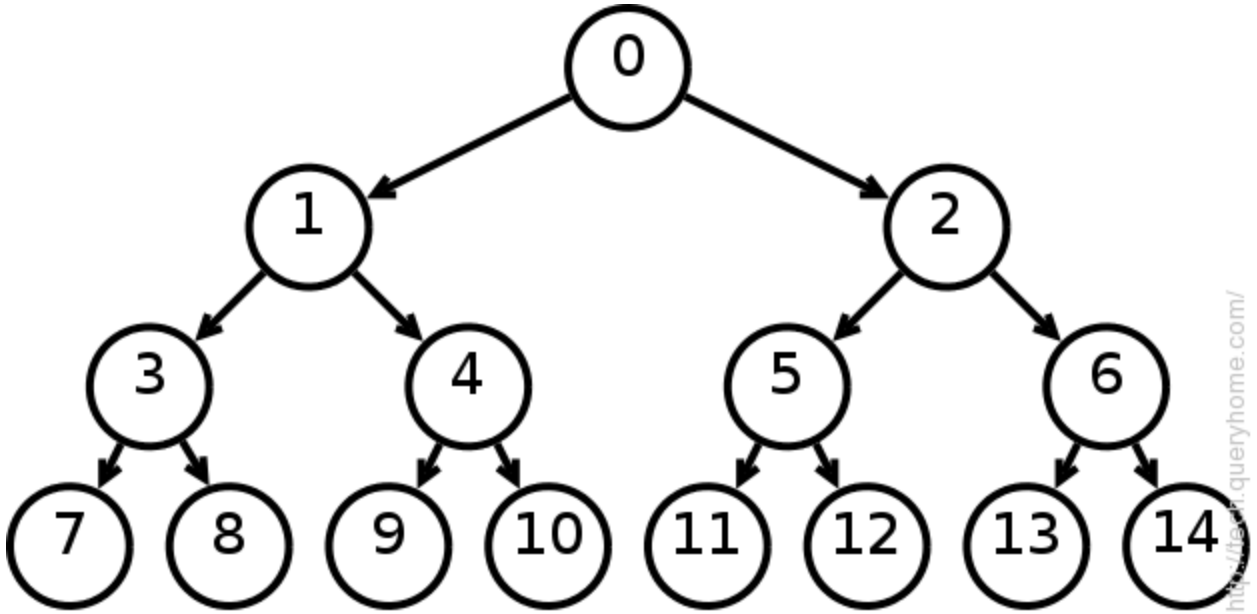
a **binary tree** is either empty (represented by a null pointer), **or** is made of a single node, where the left and right pointers (recursive definition ahead) each point to a **binary tree**.

Binary Trees

- A binary tree is restricted to only having **0, 1, or 2** children.
  - A **complete binary tree** is one where all the levels are full with exception to the last level and it is filled from left to right.
  - A **full binary tree** is one where if a node has a child, then it has two children.

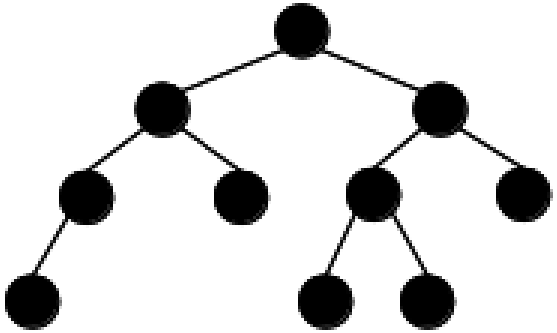
Complete and Full Binary Tree



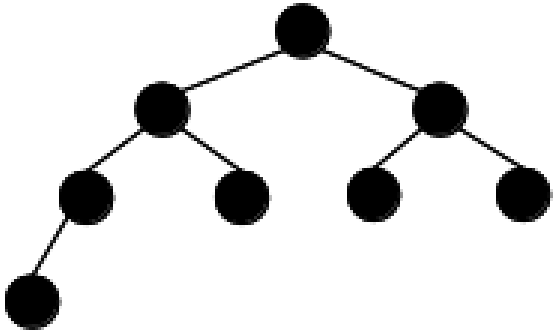


Example

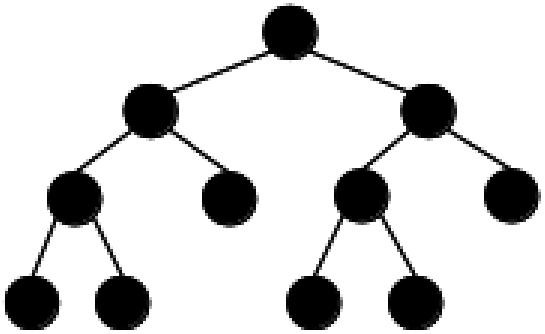
Neither complete nor full



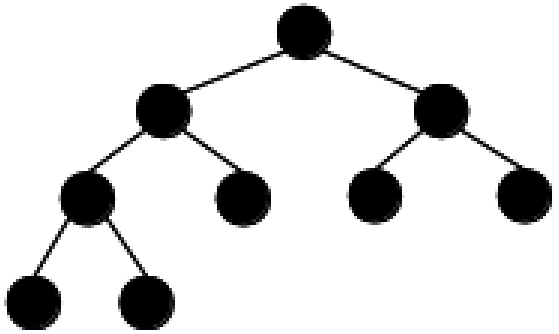
Complete but not full



Full but not complete



Complete and full

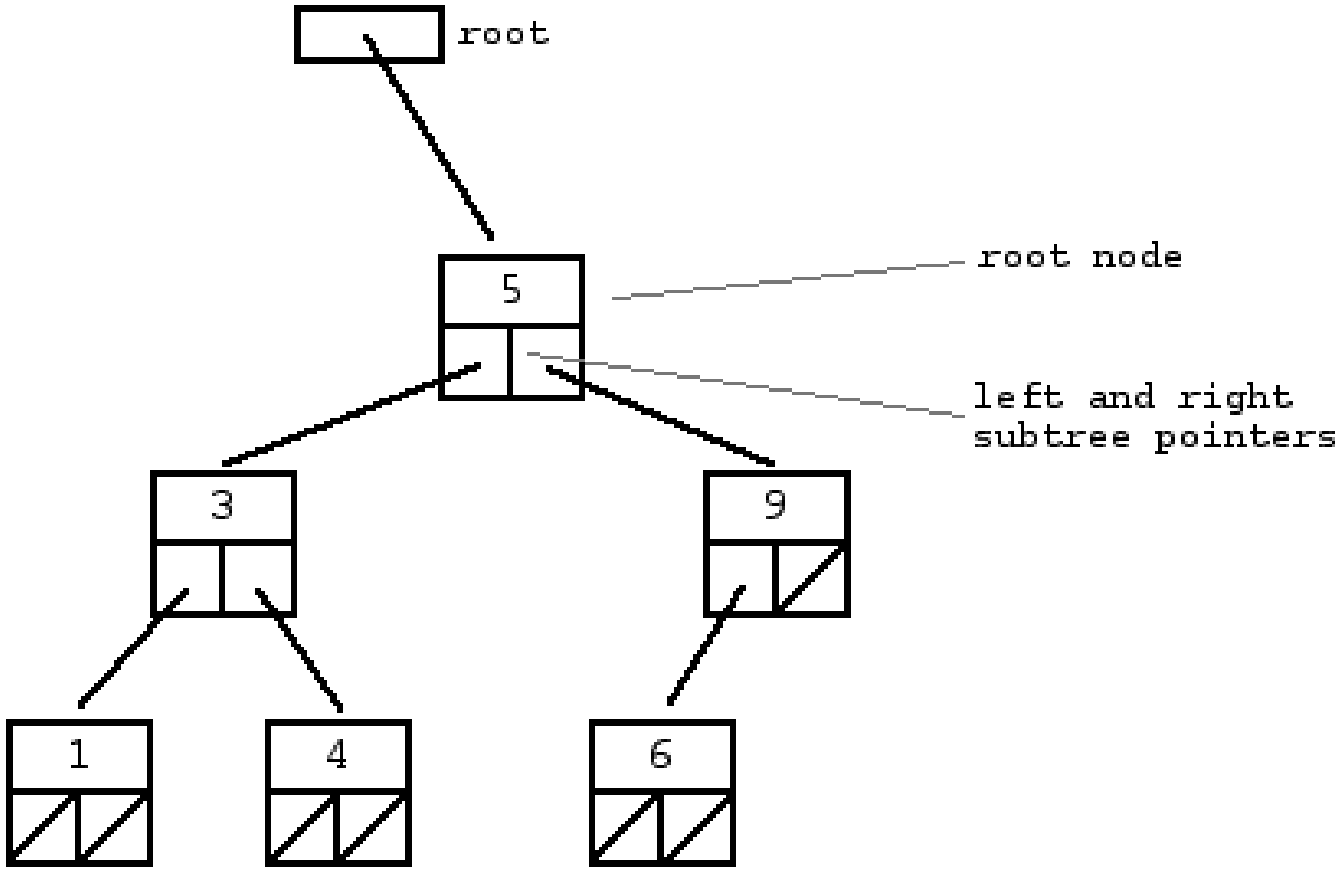


Binary Search Trees (BST)

A binary search tree (BST) is a binary tree that has the following property:

- For each node *n* of the tree, all values stored in its left subtree are less than value *v* stored in *n*, and all values stored in the right subtree are greater than *v*.

Example of Binary Search Tree



Example Binary Search Tree Cont.

- The "root" node is a 5,
- its left subtree nodes (1, 3, 4) are <= 5, and
- its right subtree nodes (6, 9) are > 5.
- Recursively, each of the subtrees must also obey the binary search tree constraint:
- in the (1, 3, 4) subtree, the 3 is the root, the 1 <= 3 and 4 > 3.
- Watch out for the exact wording in the problems -- a "binary search tree" is different from a "binary tree".
- The nodes at the bottom edge of the tree have empty subtrees and are called "leaf" nodes (1, 4, 6) while the others are "internal" nodes (3, 5, 9).

Exercise

- 24, 21,16,27,0,1,5,9,50,4

1- First No. is root node which is ?

2- if No. > root

place on right side

3- if No. < root

place on left side

*If equal then place on right side.

Exercise

- 50, 10, 40, 60, 70, 80, 20
- 85, 20, 15, 19, 100, 75, 2
- 4, 9, 0, 1, 19, 50, 51

Binary Tree Traversals

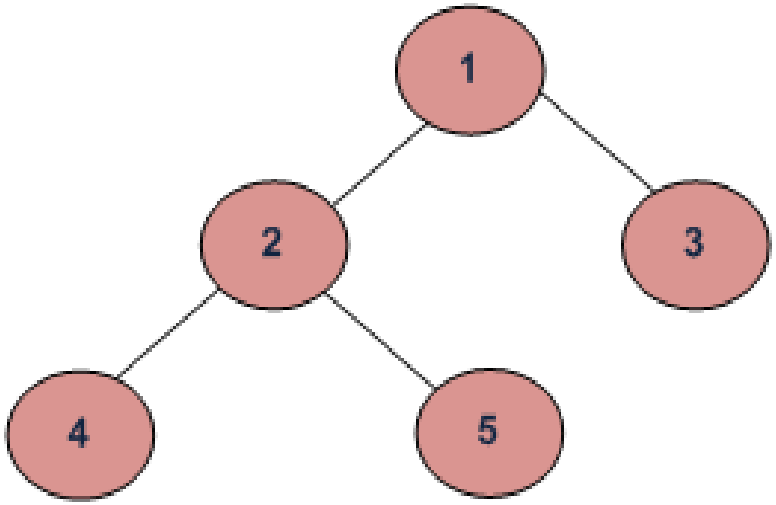
- A traversal is where each node in a tree is visited and visited once
- For a tree of n nodes there are n! traversals
- Of course most of those are hard to program
- There are two very common traversals
  - Breadth First
  - Depth First

Breadth First

- In a breadth first traversal all of the nodes on a given level are visited and then all of the nodes on the next level are visited.
- Usually in a left to right fashion
- This is implemented with a queue

Breadth First

- Breadth First or Level Order Traversal :
- 1 2 3 4 5



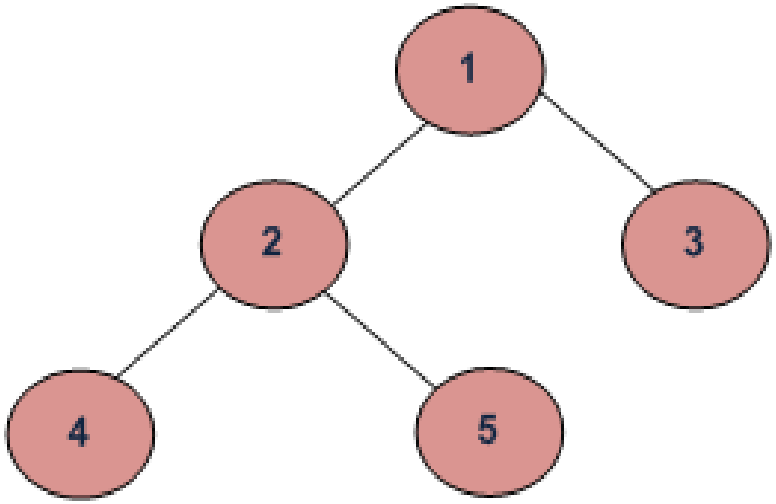
Depth First

- In a depth first traversal all the nodes on a branch are visited before any others are visited
- There are three common depth first traversals
  - Inorder
  - Preorder
  - Postorder
- Each type has its use and specific application

BST Traversals

- In-order (Left, Root, Right)
- Pre-order (Root, Left, Right)
- Post-order (Left, Right, Root)

- In-order – 4, 2, 5, 1, 3
- Pre-order 1, 2, 4, 5, 3
- Post-order 4, 5, 2, 3, 1



Insertion in BST

- In order to build a tree you must be able to insert into the tree
- In order to do this you need to know where the nodes goes
- Typically the tree is searched looking for a null pointer to hang the new element from
- There are two common ways to do this
- Use a look ahead or check for null as the first line in the code

Insertion in BST

- A naïve algorithm for inserting a node into a BST is that,
- we start from the root node,
- if the node to insert is less than the root,
  - we go to left child, and
  - otherwise we go to the right child of the root.
- We continue this process (each node is a root for some sub tree) until we find a null pointer (or leaf node) where we cannot go any further.
- We then insert the node as a left or right child of the leaf node based on node is less or greater than the leaf node.
- We note that a new node is always inserted as a leaf node.

Searching in BST

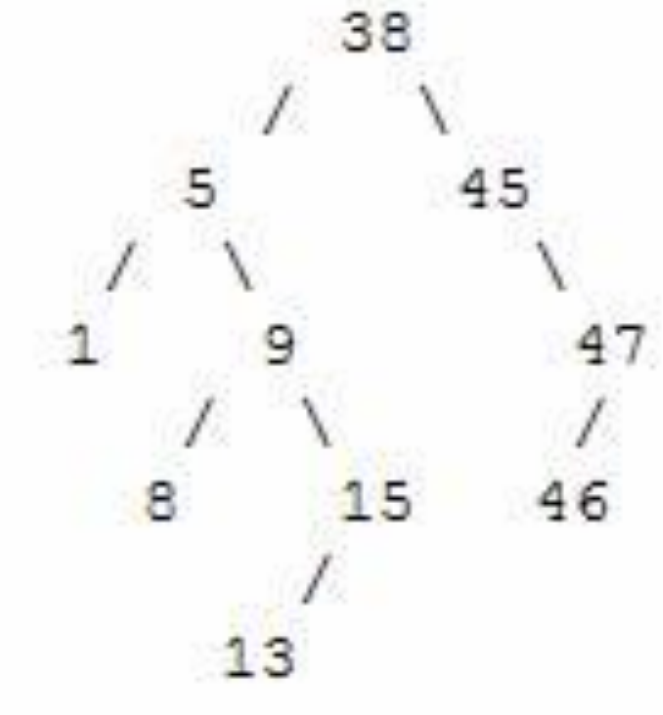
- Searching for a node is similar to inserting a node. We start from root, and then go left or right until we find (or not find the node).

Deletion in BST

- A BST is a connected structure.
- That is, all nodes in a tree are connected to some other node.
- For example, each node has a parent, unless node is the root.
- Therefore deleting a node could affect all sub trees of that node.

Deletion in BST cont.

- For example, deleting node 5 from the tree could result in losing sub trees that are rooted at 1 and 9.
- Hence we need to be careful about deleting nodes from a tree.



Deletion in BST cont.

- The best way to deal with deletion seems to be considering special cases.
  - What if the node to delete is a leaf node?
  - What if the node is a node with just one child?
  - What if the node is an internal node (with two children).

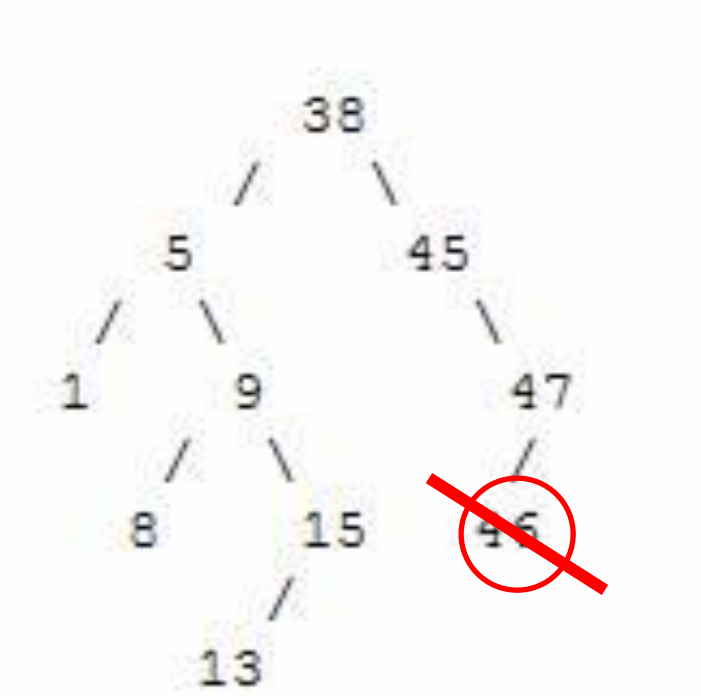
Deletion in BST cont.

- **Case 1 : The node to delete is a leaf node**

Deletion in BST cont.

- **Case 1 : The node to delete is a leaf node**

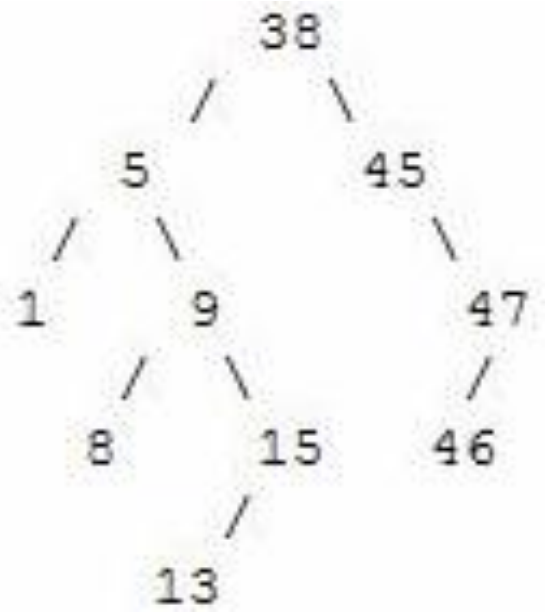
This is a very easy case. Just delete the node. We are done



Deletion in BST cont.

- **Case 2 : The node to delete is a node with one child.**
- This is also not too bad. If the node to be deleted is a left child of the parent, then we connect the left pointer of the parent (of the deleted node) to the single child.
- Otherwise if the node to be deleted

is a right child of the parent, then we connect the right pointer of the parent (of the deleted node) to single child.



Deletion in BST cont.

- **Case 3: The node to delete is a node with two children**
- This is a difficult case as we need to deal with two sub trees.
- But we find an easy way to handle it. First we find a replacement node (from leaf node or nodes with one child) for the node to be deleted.
- We need to do this while maintaining the BST order property.
- Then we swap leaf node or node with one child with the node to be deleted (swap the data) and delete the leaf node or node with one child (case 1 or case 2)

Deletion in BST cont.

- **Case 3: The node to delete is a node with two children**

Next problem is finding a replacement leaf node for the node to be deleted. We can easily find this as follows.

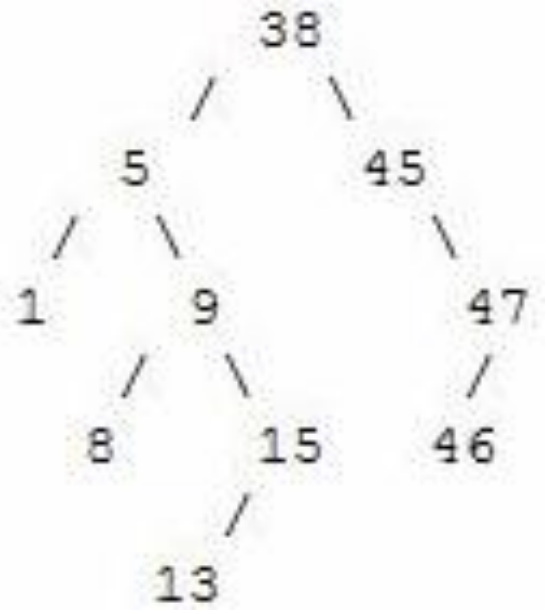
If the node to be deleted is N, the find the largest node in the left sub tree of N or the smallest node in the right sub tree of N.

These are two candidates that can replace the node to be deleted without losing the order property. For example, consider the following tree and suppose we need to delete the root 38.

Deletion in BST cont.

- **Case 3: The node to delete is a node with two children**
- Then we find the largest node in the left sub tree (15) or
- smallest node in the right subtree

(45) and replace the root with that node and then delete that node.



Time complexity in big O notation

Algorithm	Average	Worst Case
Space	Θ(n)	O(n)
Search	Θ(log n)	O(n)
Insert	Θ(log n)	O(n)
Delete	Θ(log n)	O(n)

Recall Pervious Concepts

- Recursion
- Pointers
- Linked List

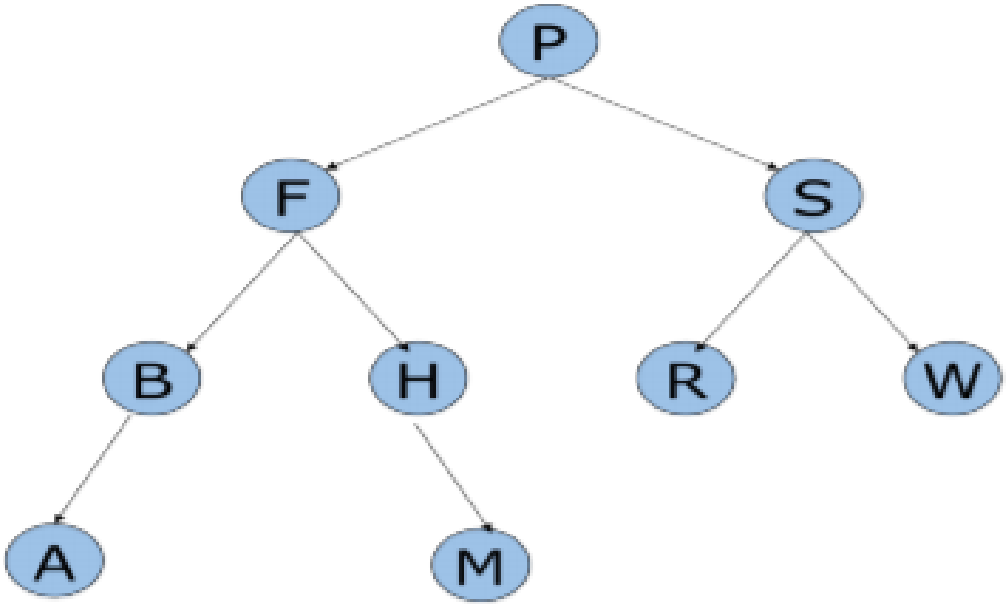
Assignment # 1

1. Convert into a binary search tree
2. Write down each of the three traversal techniques for each question listed in next slide.
   - in-order,
   - pre-order and
   - post order

Assignment # 1

- 300, 210, 400, 150, 220, 370, 450, 100, 175, 215, 250, 325, 390, 425, 470
- 10, 20, 1, 0, 2, 9, 4, 50, 70
- 23, 11, 21, 43, 91, 29, 82, 75
- 75, 82, 91, 19, 69, 74, 52, 0, 10
- 63, 57, 89, 72, 39, 46, 58, 16, 7, 92
- 47, 4,5,101,94,93,103,8,17,19,11,32,57

Assignment # 2



References and Reading Material

- http://www.ugrad.cs.ubc.ca/~cs221/2009W2/notes/set07_binary_trees_BST_ed_2perPage.pdf
- http://web.york.cuny.edu/~kzaman/cs291/trees.pdf
- https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html
- http://cslibrary.stanford.edu/110/BinaryTrees.html
- https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm
- https://www.ius.edu.ba/sites/default/files/u1251/6._tree-traversals.pdf
- http://www.cs.cmu.edu/~clo/www/CMU/DataStructures/Lessons/lesson4_3.htm