|  |
| --- |
| Lab11 Tree - Solution |
|  |
| Exercise 1: Theory of Binary Tree (30 min) |
| Consider the following **binary tree**:   1. Is the above tree    1. a full binary tree,    2. a complete binary tree,    3. a binary search tree? 2. What node or nodes are 3. The tree's root? 4. Parents of 15 and 60? 5. Children of the node 15? 6. Siblings of node 60? 7. Ancestors of 40? 8. Descendants of 9? 9. Leaves? 10. What is the **left sub-tree of the root** of the tree? 11. Write the **preorder**, **inorder**, and **postorder traversals** of the above binary tree. 12. Which tree traversal algorithm visits the nodes in **ascending order** of the values stored in odes of a binary search tree? 13. The algorithm to **insert** a node into a binary search tree is given below:   Starting from the root,   1. if a new value is greater, than the node's value:    * if a current node has no right child, place for insertion has been found;    * otherwise, handle the right child with the same algorithm. 2. if a new value is less than or equal to the node's value:    * if a current node has no left child, place for insertion has been found;    * otherwise, handle the left child with the same algorithm.   What tree results after you insert the nodes 90, 25, 45, 27, 57, and 10 in that order with the above algorithm?   1. Based on the **search** algorithm for binary search algorithm given in the lecture notes, what is the search path for the following target integers:    1. 25    2. 90    3. 38 |
| Exercise 2: Test the Binary Tree and Binary Search Tree Implementation (30 min) |
| Run and test the implementations of the **Binary Tree** and **Binary Search Tree** given with the Lecture 11 notes (lec.11.zip) in the *BinaryNodeTree* and *BinarySearchTree* folders respectively.  In this implementation, **BinaryNodeTree** is a sub class of the **BinaryTreeInterface** and **BinarySearchTree** is a sub class of **BinaryNode Tree**. Try to understand the following core operations of both the Binary Tree and Binary Search Tree:   1. Insert a node 2. Search for a node with a specific value 3. Preorder traversal 4. Inorder traversal 5. Postorder traversal |
| Exercise 3: Implement Your Own Binary Search Tree ADT (60 min) |
| You are given a binary search tree class declaration (**BST.h**) that consists of the following methods / operations:   1. Default constructor 2. Destructor 3. Check if the tree is empty 4. Get number of nodes 5. Insert a node 6. Search for a node with a specific value 7. Preorder traversal 8. Inorder traversal 9. Postorder traversal   Provide the implementation for all the methods of the BST class in an implementation file named **BST.cpp**. Write a driver to test your implementation by allowing user to *input* a series of integers and *search* for a specific integer. The program should display the series of integers input by the user in *preorder*, *inorder* and *postorder* traversals as well as the *number of nodes* in the BST tree.  (For the tree node, use the **BinaryNode.h** and **BinaryNode.cpp** from Exercise 2). |