# Lab 6 : Binary Search Tree

# Week beginning 22nd October, 2018

The code for BST is given on shared drive.

1. Use debug mode to step through the insertSub method to see the recursive calls.

2. Change insertSub so that it handles duplicates as add method of Set interface.

3. Write a contains method. The header is:

boolean contains​(E element)

(Same as contains method of Collection interface)

(a) Write it either iteratively or recursively, whichever you find easier. Then write the other version.

(b) Which version – iterative or recursive - is more efficient? Explain your answer. You can use profiler to check this.

4. Use debug mode to step through the printSub method to see the recursive calls.

5. Change the printSub method to perform

* preorder traversal
* postorder traversal

Check that the output from these methods is what you expect.

6. Write a non-recursive version of preorder traversal. What data structure is required?

Use the following algorithm:

Push the root (reference to root node) onto the stack.

While the stack is not empty

pop the stack and visit it

push its two children (reference to children) //what order do you push children??

To use a stack, use Deque interface as Java recommends. Use LinkedList as implementation of Deque.

7. Perform breadth-first traversal of a binary search tree. What data structure is required?

Use the following algorithm:

Start with empty data structure

insert the root (reference to root node) onto the data structure

while the data structure is not empty

remove a node from the data structure and visit it

insert its two child nodes //what order do you insert the children?

8. Consider the difference between the algorithm for preorder traversal (depth-first traversal) and the algorithm for breadth-first traversal.