Lab 9: BST

Data Structures and Algorithms

In this lab, we will explore the Binary Search Tree – BST.

Your first task: Implement BST using the code given in the video lecture. Please change the name of the class from BinaryTree to BinarySearchTree. Keep in mind that there are several errors in the code given. Also, when I said you need two more lines of code, you actually need three lines.

Now, let test time for BST sort.

When you put any data in the BST and process it using in-order traversal, you actually perform sorting call Binary Search Tree sort. This is one of the best sort with O(n log n). However, it is very easy to find deformed case for this sort. Let explore it.

To test the BST insertion performance. You are to generate two set of data, insert each of them into a BST, and then measure the time of the insertions. The first set of data is an ordered data, from 1, 2, 3, .., N. The second set of data consist of N random data values between [0, N). You code may look like the followings:

First set of data	Second set of data

`for(int i=0; i<N; i++) {`
`  bst.insert(i);`
`}`
	`for(int i=0; i<N; i++) {`
	`  bst.insert((int)(N*Math.random()));`
	`}`

Where bst is your binary search tree. You will have to implement a way to measure time in millisecond yourself.

Now, you have to select N yourself to fill the following empirical data table below

N	Time (millisecond) for insert	
	First set of data	Second set of data

For example, your N can be 1 million up to 10 million.

Your second task: Perform the experiment using your N's of choice. You have to select N such that you can plot a graph and see the growth rate of time nicely. Your biggest N should take more than a minute to insert all N data from the first set.

Instruction: For task 1 and 2, implement BinarySearchTree.java and perform the experiment. Submit your BinarySearchTree.java and the data table.