**Course Project**

**DeVry University**

**College of Engineering and Information Sciences**

**Course Number: CEIS295**

# Module 6: Binary Search Trees Real-World Speeds

# Objectives

* To add Client data type to the BinarySearchTree
* To develop Python code that tests the BinarySearchTree’s real world speeds
* To update an Excel table that displays relevant real-world speeds

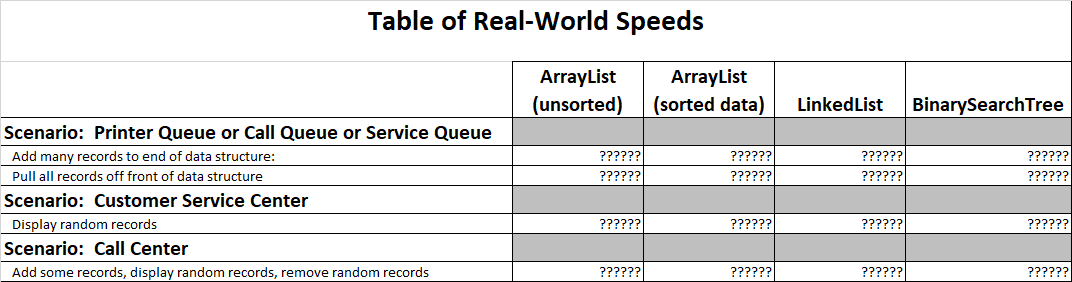
We have discovered that ArrayLists are extremely fast for retrieving data. Unfortunately, they are extremely slow to add and remove data, especially from the front of the data structure. LinkedLists are extremely fast in adding and removing data, especially from the front and end of the data structure. Unfortunately, they are extremely slow for retrieving data from the middle.

The Binary Search Tree provides a compromise between the ArrayList and the LinkedList. Retrieving information from the Binary Search Tree is very fast (O(log n)). It is not extremely fast, but it is considered very fast for data retrieval. In addition, a balanced Binary Search Tree is very fast for adding and removing data (O(log n)). If you need to do it all, then the Binary Search Tree may be the data structure for you to use.

Last week, we determined that the fastest searches can be conducted on sorted data using a Binary Search. Another benefit of the Binary Search Tree is that the data is maintained in sorted order!

Let’s use our time measurement technique to test the Binary Search Tree algorithm’s real-world speed. How does the Binary Search Tree compare in speed with the ArrayList and the Linked List using real-world data? Let’s find out!

# Steps

1. Create a new folder in your CEIS295 folder called “Week 6 Project”. Copy the Excel table to this folder that we used in Week 2 for our LinkedList project. We can use this Excel table to record the time that it takes to perform real-world processes based on common scenarios. We are going to compare the Binary Search Tree data structure with the ArrayList and the LinkedList. Update your Excel table. It should look something like this one, but it should contain the actual times that you already recorded the ArrayList and LinkedList:  
     
   
2. Download the ClientData.csv file and place the file in your “Week 6 Project” folder. We will read the data in this file so we can work with a real-world sized dataset to test our algorithm.
3. Copy the Node.py file and the BinarySearchTree.py files to this folder. We will use the BinarySearchTree code to check the speed of this algorithm.
4. Copy your Client class to this folder. Update the Client class to include the required methods to be used in a Binary Search Tree.
   1. \_\_lt\_\_ - means “less than” and should compare client\_id
   2. \_\_eq\_\_ - means “equal to” and should compare client\_id
   3. \_\_str\_\_ - means “string” and should return the string version of the client\_id
5. In this same folder, create a file called BinarySearchTreeListActualSpeed.py. Type your name and the current date at the top of the code. Then, import the BinarySearchTree class, the Client class, the date module from the datetime library, the time module, the random module, and the sys module.
6. In the same BinarySearchTreeListActualSpeed.py file, display your name and the current date in the output to show that you are the author of this code.
7. In the same BinarySearchTreeListActualSpeed.py file, create a list and read the records from the ClientData.csv file into Client objects and place the Client objects into the list.
8. In the same Python code file, create an BinarySearchTree object. Let’s test the first scenario. If you accept jobs at the back of the line and then process the jobs from the front of the line, which data structure is best? Write code to test how much time that it takes to add the Client objects from the list into the BinarySearchTree data structure. Add this time to the spreadsheet into the “Add many records to end of data structure” row in the Excel table.
9. Continue the BinarySearchTreeActualSpeed code and write code to test how much time it takes to remove all of the Client objects from the front of the BinarySearchTree, one object at a time. Add this time to the spreadsheet into the “Pull all records off front of data structure” row in the Excel table.
10. Now, let’s test the second scenario. A customer service center receives calls from random customers. The customer service center must be able to quickly pull up the customer’s record. Is the BinarySearchTree good for this scenario? Let’s see how long it takes to pull up random records. Write code to add all of the customers to the LinkedList again since we deleted all of the customers on the last step. Now, write code to test how much time it takes to display many random records. Add this time to the spreadsheet into the “Display random records” row in the Excel table.
11. Let’s consider the third scenario. Consider a credit card call center. The representatives may sign up new credit card accounts (add records). The representatives may answer customers’ questions (display random records). Finally, the representatives may delete a paid-off account at the request of the customer (remove records). Write code to test how much time it takes to add the Client records to the BinarySearchTree, then randomly display records, and then randomly delete records.

# Deliverables Part 6

* Complete the Module 6 Course Project Presentation deliverable
  + BinarySearchTreeActualSpeed.py code
  + Screenshot showing the code running with your name and date in the output
  + Updated Excel “Table of Speeds” table