**Course Project**

**DeVry University**

**College of Engineering and Information Sciences**

**Course Number: CEIS295**

# Module 4: Sorting Algorithms

# Objectives

* To use Client data type in sorting algorithms
* To compare sorting algorithms using data to determine real-world speeds
* To develop an Excel table that displays relevant real-world speeds

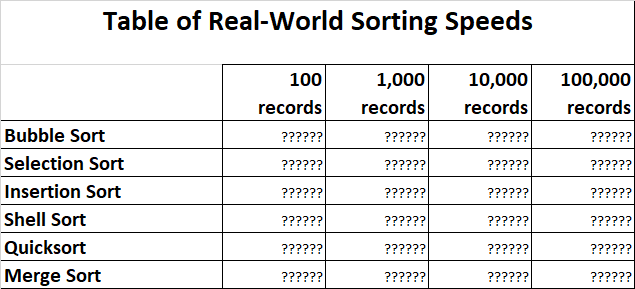
Next week, you are going to discover that you can find client records ***significantly*** faster if the data is sorted. However, it takes time to sort the data. It is important to use fast sorting algorithms to get the benefits of the advanced searching algorithms that we are going to discuss next week.

Some algorithms are very simple to understand and implement. Other algorithms are extremely complex and you must implement them carefully. Is the complexity worth it? Let’s find out!

We are going to use this time measurement technique to test the algorithms’ speeds this week and over the next few weeks. In addition, we are going to test the algorithms using a small, but real-world sized, dataset.

# Steps

1. Create a new folder in your CEIS295 folder called “Week 4 Project”. Create an Excel table in this folder so we can record the time that it takes to perform sorting algorithms on real-world data. We are going to compare real-life speeds for the following sorting algorithms:  
   1. Bubble Sort
   2. Selection Sort
   3. Insertion Sort
   4. Shell Sort
   5. Quicksort
   6. Merge Sort

Your Excel table should look something like this:  
  


1. Download the ClientData100.csv file, ClientData1000.csv file, ClientData10000.csv file, and ClientData100000.csv file and place the files in your “Week 4 Project” folder. Open the files and look at the data. We will read the data in these files so we can work with real-world sized datasets to test our algorithms.
2. In this same folder, create a Client class based on the following UML diagram. The **\_\_lt\_\_** method means “less than” and it should return True if this object is less than the other object (parameter). You need to **use the last name and the first name** to determine if this object is less than the other object. The **\_\_le\_\_** method means “less than or equal to” and it should return True if this object is less than or equal to the other object (parameter). The **\_\_eq\_\_** method means “equal to” and it should return True if this object is the same as the other object (parameter). Otherwise, it should return False. The **\_\_str\_\_** method should return the \_\_last\_name and \_\_first\_name in this format: Black, Jack  
   Also, make sure your name and the current date are listed at the top of the code.  
     
   
3. In this same folder, download the file called BubbleSort.py and place the code file in your “Week 4 Project” folder. Review the BubbleSort.py code carefully and do your best to understand the code. Download the other sorting files and place them in the same folder, including SelectionSort.py, InsertionSort.py, ShellSort.py, Quicksort.py, and MergeSort.py.
4. If you google for “sorting algorithms visualized”, you will find some excellent resources. For example:
   1. <https://visualgo.net/bn/sorting>
   2. <https://www.toptal.com/developers/sorting-algorithms>
5. First, let’s test the speed of the Bubble Sort algorithm. In this same folder, create a file called SortingActualSpeeds.py. Type your name and the current date at the top of the code. Then, import the BubbleSort class, the Client class, the date module from the datetime library, and the time module.
6. In the same SortingActualSpeeds.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 SortingActualSpeeds.py file, put these lines next in your code. This way, you can uncomment the line with the name of the file that you want to test by removing the hashtag.  
     
   #input\_file\_name = 'ClientData100.csv'  
   #input\_file\_name = 'ClientData1000.csv'  
   #input\_file\_name = 'ClientData10000.csv'  
   #input\_file\_name = 'ClientData100000.csv'
8. Read the records from the ClientData100.csv records into Client objects and place the Client objects into a list called clients. Now, check to see how long it takes the Bubble Sort algorithm to sort the list. Type this time value into your Excel table.
9. Then, use the ClientData1000.csv file and check to see how long it takes to sort 1000 records. Use the ClientData10000.csv file to check to see how long it takes to sort 10000 records. Write these times into your Excel table. **If you have a very fast computer**, you can use the ClientData100000.csv file to check to see how long it takes to sort 100000 records**. If you have a slow computer, simply put “Over 1 Hour”.** On my i5-2500k computer, the process took one hour and twenty-seven minutes.
10. Test the speed for the other sorting algorithms. Put the time that it takes to sort the records in the Excel table. When you sort the biggest data file, “ClientData100000.csv”, that contains 100,000 records, you will discover that the BubbleSort, SelectionSort, and InsertionSort are very slow. If you actually run these sort routines on 100,000 records, it will take over an hour for each of them. Instead, simply write “Over 1 Hour” on your Excel table in the 100,000 column for these sort routines. On the other hand, the ShellSort, Quicksort, and Merge Sort are very fast! You can use these sort routines to sort the 100,000 records in just a few seconds!   
    1. Bubble Sort – over 1 hour
    2. Selection Sort – over 1 hour
    3. Insertion Sort – over 1 hour
    4. Shell Sort – fast
    5. Quicksort – fast
    6. Merge Sort – fast

# Deliverables Part 4

* Complete the Module 4 Course Project Presentation deliverable
  + Client.py code – did you implement \_\_lt\_\_, \_\_le\_\_, and \_\_eq\_\_?
  + SortingActualSpeed.py code
  + Screenshot showing the code running with your name and date in the output
  + Excel “Table of Sorting Speeds” table