**Course Project**

**DeVry University**

**College of Engineering and Information Sciences**

**Course Number: CEIS295**

# Module 1: ArrayList Real-World Speeds

# Objectives

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

We are discussing the algorithm speeds using theoretical nomenclature, including Big-O notation. At the same time, it is important to test the algorithm speeds using real-world data. Does the algorithm perform as expected? Write code to test the algorithm and find out!

You can test the speed of a process by checking the current time. Then, run the process. Finally, check the current time again. Subtract the current time at the start of the process from the current time at the end of the process. The different in time will be the time that the process took to complete.

For example, how long does it take to say “Hello Everyone!” a total of 1000 times? Run this code in your Python IDE:

#Name: Your name

#Date: Your date

import time # use time library to time the code executions

# get current time before the process

start\_time = time.time()

# run the process

for i in range(1000):

print( "Hello Everyone!" )

# get current time after the process

end\_time = time.time()

# subtract start time from end time to get time used by process

total\_time = end\_time - start\_time

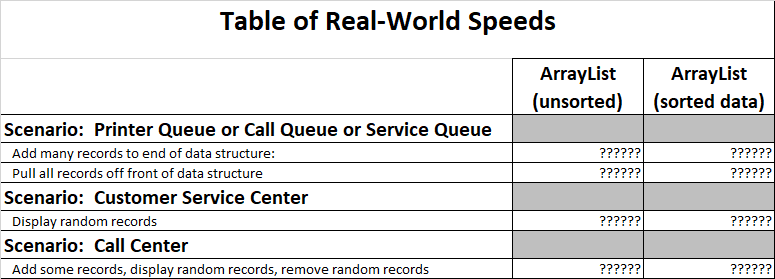
# Show the result. Note: .6f means “show six decimal places”

print("\nSeconds run 1000 times: {:.6f}".format(total\_time))

On my computer that has an Intel i5-2500K processor running at 3.30GHz, the process only takes 0.04 seconds. Nice! How long does it take on your computer? We use Big-O notation to talk about theoretical speeds because your speed may be very different than my speed on this process. If you are using an AMD Ryzen 9 3950X based system, your speed will be significantly faster than my speed. On the other hand, if you are using a computer with a Celeron processor running at 1.20GHZ, the process will take significantly longer. In other words, all processes will run faster on a faster computer, and all processes will run slower on a slower computer. The Big-O notation gives us a way to discuss the relative speed of different algorithms without discussing the hardware that the process will run on.

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 1 Project”. Create an Excel table in this folder so we can record the time that it takes to perform real-world processes based on common scenarios. We are going to compare the ArrayList data structure that we created this week with data structures that we are going to create over the next few weeks. In addition, we are going to compare the ArrayList search feature when you use unsorted data versus when you use sorted data. The Excel table should look something like this:  
     
   
2. In this same folder, create a file called TimeProcess.py and place the code listed above in the file. Run the code and check to see how long it takes your computer to say “Hello Everyone!” one thousand times. Make sure you understand how to check the time of a process based on this technique. Also, make sure your name and the current date are listed at the top of the code.
3. Download the ClientData.csv file and place the file in your “Week 1 Project” folder. Open the file and look at the data. We will read the data in this file so we can work with a real-world sized dataset to test our algorithms. This dataset is considered “small” based on real-world data, but the dataset will give us a way to test our algorithm speed.
4. Download the Quicksort.py file and place it in your “Week 1 Project” folder. We are going to use this code to sort our data and check the speed difference when you use sorted versus unsorted data with an ArrayList.
5. In this same folder, create a Client class based on the following UML diagram. The \_\_eq\_\_ method means “equals” and it should return True if this object is the same as the other object, which is received in the parameter. Otherwise, it should return False. The \_\_str\_\_ method means “string” and should return the \_\_cliend\_id, \_\_first\_name, and \_\_last\_name in this format: 1000, Black, Jack  
   Also, make sure your name and the current date are listed at the top of the code.  
     
   
6. Copy the ArrayList.py file to this folder. We will use the ArrayList code to check the speed of this algorithm.
7. In this same folder, create a file called ArrayListActualSpeed.py. Type your name and the current date at the top of the code. Then, import the ArrayList class, the Client class, the Quicksort class, the date module from the datetime library, the time module, the random module, and the sys module.
8. In the same ArrayListActualSpeed.py file, display your name and the current date in the output to show that you are the author of this code.
9. In the same ArrayListActualSpeed.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.
10. In the same Python code file, create an ArrayList 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 ArrayList data structure. The ArrayList automatically adds the new records to the end of the data structure so add this time to the spreadsheet into the “Add many records to end of data structure” row in the Excel table.
11. Continue the ArrayListActualSpeed code and write code to test how much time it takes to remove all of the Client objects from the front of the ArrayList, 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.
12. 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 ArrayList 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 ArrayList 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.
13. 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 ArrayList, then randomly display records, and then randomly delete records. Add this time to the spreadsheet into the bottom row.
14. If the data is sorted, will the ArrayList run faster? If you have sorted data, you can use the search\_sorted( ) method. We will talk about different searching methods in a couple of weeks! Let’s check the difference in speed for the ArrayList when you use sorted and unsorted data.
15. In your ArrayListActualSpeed.py code, add this line after you read the client records to sort the clients list:  
      
    Quicksort.sort(clients)
16. Then, find your search methods and change them from search( ) to search\_sorted( )
17. Run the code and add the speeds to your Excel table in the “ArrayList (sorted data)” column.

# Deliverables Part 1

* Complete the Module 1 Course Project Presentation deliverable
  + TimeProcess.py code
  + Client.py code
  + ArrayListActualSpeed.py code
  + Screenshot showing the code running with your name and date in the output
  + Excel “Table of Speeds” table