**CMPSC 412 – Lab-4** (50 points)

**Searching and Sorting**

**Due date: 2/22/2022**

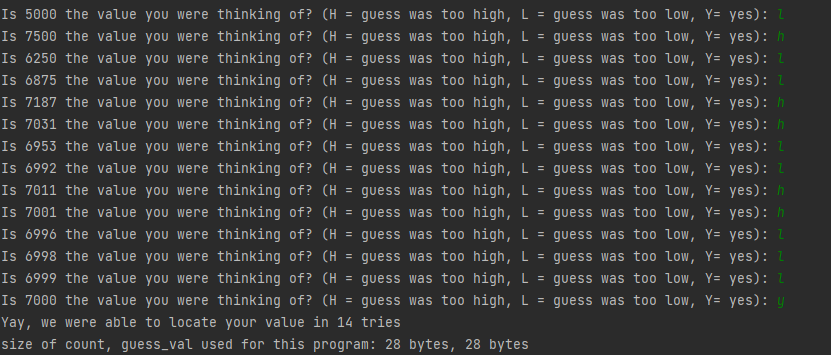
**Note:** attach screenshots of your program and results under each programming exercises. Please make sure that the screenshot is readable. Don’t attach a very small screenshot image.

**Lab Exercises:**

1. Implement and use binary search to make a "guess-the-number" game: You think of a number between 1 and 10000, your program guesses the number and you reply whether your guessed number is higher or lower. Your program makes another guess and so on until it gets the right number. Calculate the actual memory size for each of the data structures, variable you use in your program. You can use \_\_sizeof\_\_() or sys.getsizeof().

Example: print (a.\_\_sizeof\_\_())

print (sys.getsizeof(a))



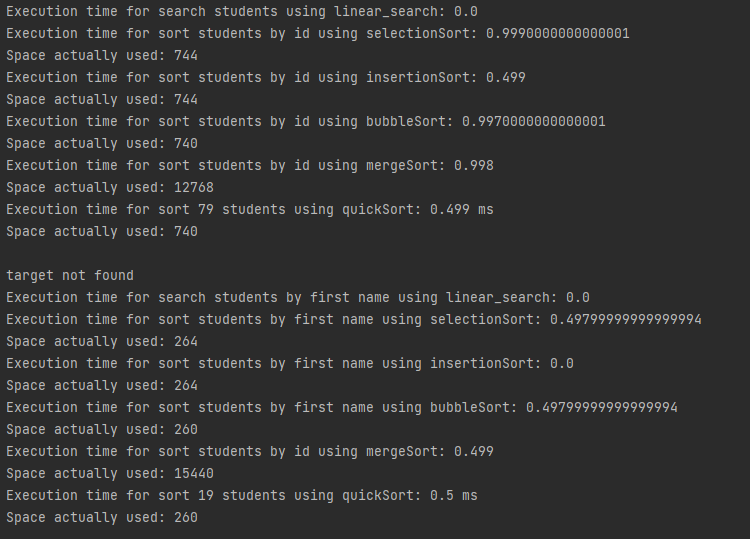
1. Create a database with the following details for at least 20 students and store it as a text file:

* Student ID
* first name
* last name
* email id
* Major
* Write a program to read the data from the text file. Choose an appropriate data type and data structure (lists, lists of list, dictionary) for storing the information in your program.
* Write a function which takes a parameter and sorts the entire list of students and displays all the details of all students. Your function should sort the text file using a) student id and b) first name, and save it in a different text file. Implement the sorting using selection sort, insertion sort, bubble sort, merge sort and quick sort (you can decide the pivot value). Print out how much cpu time it took to sort the data for each sorting algorithm. You can import a library to calculate the time. Calculate the actual memory size for your program.
* Table-1: Tabulate your recorded time for all the four sorting algorithms i.e., selection sort, insertion sort, bubble sort, merge sort and quick sort. Sorting according to a) student id and b) first name.
  1. Sort by student Id/ first name

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Selection | Insertion | Bubble | Merge | Quick |
| By Id | .999 ms | .499 ms | .997 ms | .998 ms | .499 ms |
| By FirstName | .498 ms | 0.0 ms | .498 ms | .499 ms | .5 ms |

* Table-2: Tabulate the memory size for all the four sorting algorithms i.e., selection sort, insertion sort, bubble sort, merge sort and quick sort. Sorting according to a) student id and b) first name.
  1. Memory size

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Selection | Insertion | Bubble | Merge | Quick |
| By Id | 744 bytes | 744 bytes | 740 bytes | 12768 bytes | 740 bytes |
| By FirstName | 264 bytes | 264 bytes | 260 bytes | 15440 bytes | 260 bytes |



* Write a conclusion paragraph about what you understood about the searching and sorting algorithms.

These algorithms were all run on a manually created user text file containing 240 total entries. Based on the results above, it appears that in terms of time complexity, it was more efficient to perform sorting based on the first name of the user, rather than based on their user id. Between the algorithms however, the execution time remained pretty constant across all (mostly because the input size is still relatively small). However, when testing the algorithms on inputs of various sizes, the difference becomes more apparent. Another important thing to note is that the memory usage for most of the algorithms is relatively stable, except for merge sort. The reason for that is that all the other algorithms perform swaps in the array in place, while merge sort creates new arrays of size n/2 for every recursive call. As a result, it is the most time efficient, however it requires significantly more space than other algorithms.