CMPSC 412 – Lab-2 (25 points)
Searching and Sorting

Due: in 1 week

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. Write and implement the algorithm for Linear search, Binary search, Insertion sort, Selection sort and Bubble sort. Calculate the time complexity for these searching and sorting algorithms.

   Table-1: tabulate the time complexity for these algorithms with best and worst time complexities.

Method	Time Complexity
Linear Search	Best case: O(1), Worst case: O(n)
Binary Search	Best case: O(1), Worst case: O(log(n))
Insertion Sort	Best case: O(n), Worst case: O(n^2)
Selection Sort	Best case: O(n^2), Worst case: O(n^2)
Merge Sort	Best case: O(nlog(n)), Worst case: O(nlog(n))
Bubble Sort	Best case: O(n), Worst case: O(n^2)

2. 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.

```
# Read text file and store
jsonFile = open(r'C:\Users\Jacob\Documents\School\Data Structures\students.txt')
students = json.load(jsonFile)
studentList = students['Students']
```

I am parsing with Json, then storing the information in a list of dictionaries. I have included the text file in the submission.

- 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 list using student id or first name or last name. Implement the sorting using selection sort, insertion sort, bubble sort and merge sort, and print out how much cpu time it took to sort the data. You can import a library to calculate the time.
  Show an example for searching a value using linear search. Table-2: Tabulate your recorded time for the linear search and all the four sorting algorithms i.e., selection sort, insertion sort, bubble sort and merge sort.

  Linear Search Code:

```
def linearSearch(studentIDNumber, studentList):

    # Iterate through the students dictionary, searching for the correct ID
    for student in studentList['Students']:

        # If found, print
        if (studentIDNumber == student['studentID']):

            returnName = student['firstName']
            print(f'Student ID {studentIDNumber} was found, their name is {returnName}')

            return student

    # Catch bad ID
    return 'Student not found'
```

  I randomly generate the ID to be searched between 1 and 20.

  Example Output:

```
Student ID 14 was found, their name is Byllethiel
Operation took 0.5009765625 milliseconds
```

Method	Time taken
Linear Search	0.5 milliseconds
Selection Sort	2.7 milliseconds
Insertion Sort	2.5 milliseconds
Merge Sort	11 milliseconds
Bubble Sort	3.5 milliseconds

- Table-3: Now apply all the four sorting algorithms on the sorted data (i.e., sort the text file according to student id / first name / last name where the text file is already sorted) and tabulate your recorded time. Print out how much CPU time it took to sort the data.

Method	Time taken
Selection Sort	2.5 milliseconds
Insertion Sort	2.5 milliseconds
Merge Sort	12.5 milliseconds
Bubble Sort	2 milliseconds

- Table-4: Create a different text file and have 20 rows of same student details. Apply the sorting algorithm and tabulate your readings. Print out how much CPU time it took to sort the data.

Method	Time taken
Selection Sort	2.5 milliseconds
Insertion Sort	2 milliseconds
Merge Sort	11.5 milliseconds
Bubble Sort	3 milliseconds

- Write a conclusion paragraph about what you understood from this lab exercise. What did you understand from table-1, table-2, table-3 and table-4?

While mergesort is generally accepted as the fastest sorting method of the ones we decided to cover for this lab, this does not necessarily hold true for smaller groups of data. We can see that when the list is already sorted, when it's small, or when all of the data is identical, merge sort actually takes the longest of all of the sorting algorithms. As the size of the database increases, merge sort will become more efficient. This is a good case study on why we use different sorting algorithms instead of the same one for every situation.