**CMPSC 413 – Lab-2** (25 points)

**Searching and Sorting**

**Due: in 1 week**

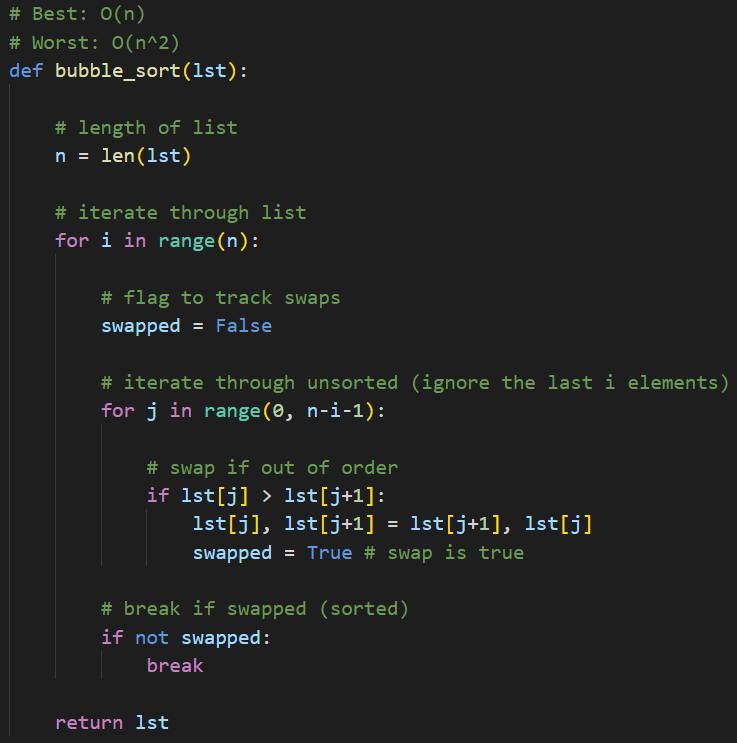
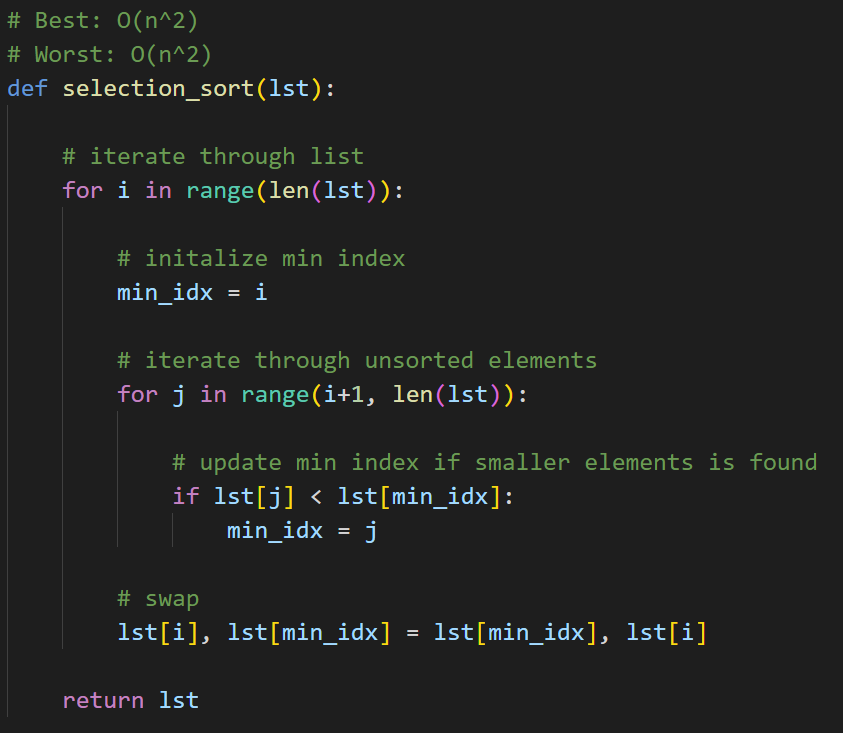
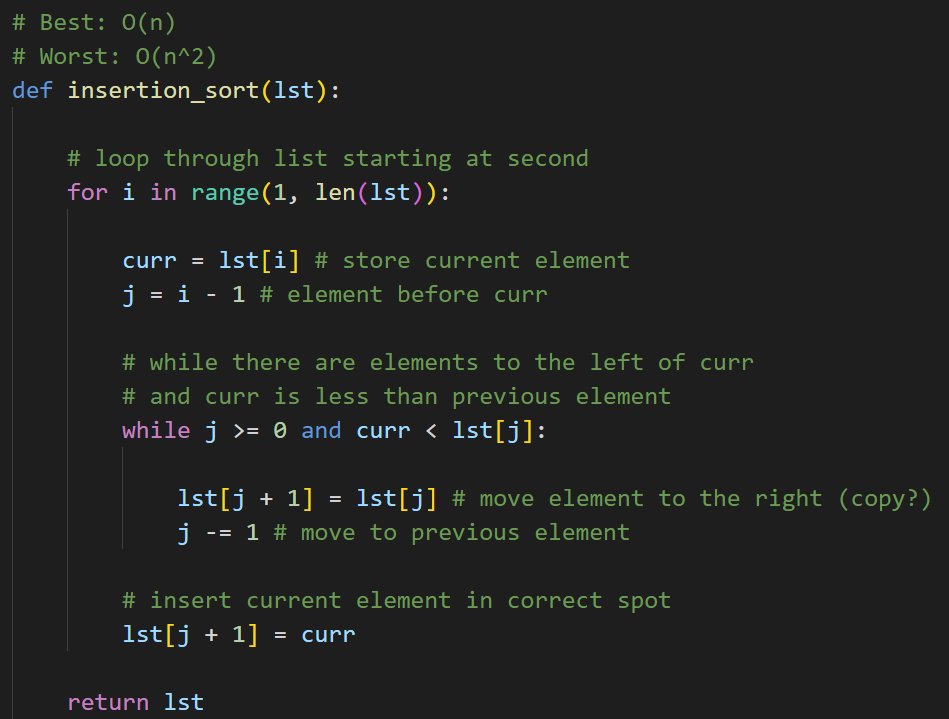
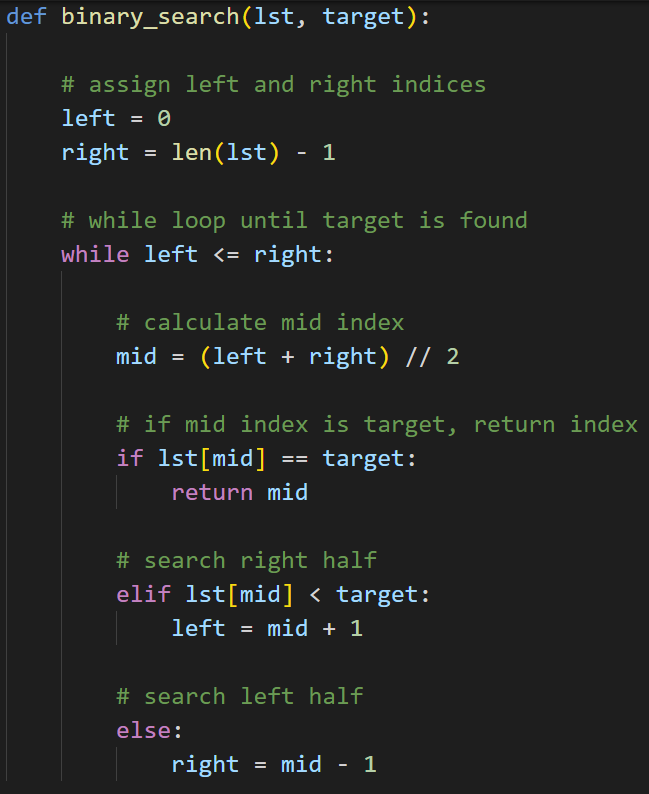
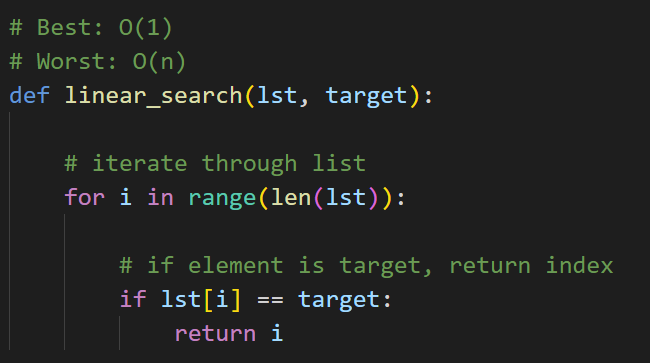
**Note:** attach screenshots of your program and results under each programming exercises. Please make sure that the screenshot is readable. Don’t attach a very small screenshot image.

**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.

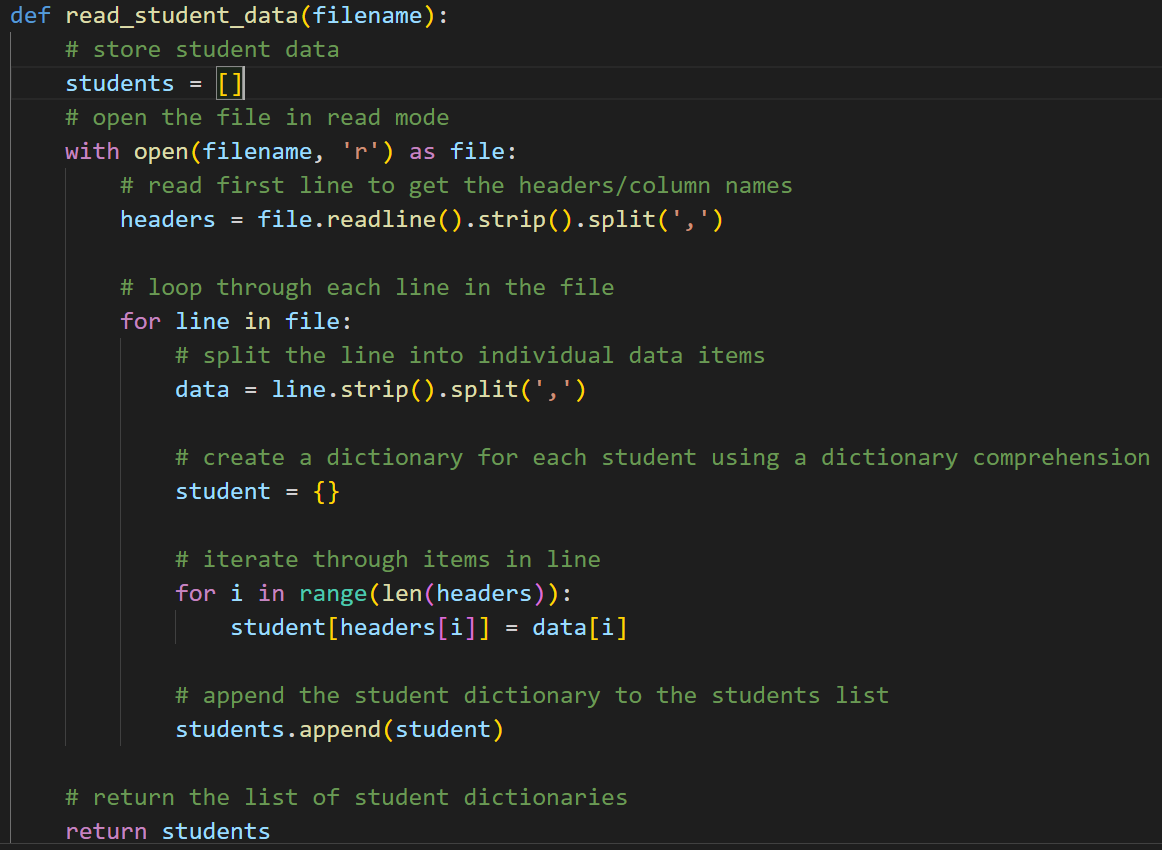
Table-1: tabulate the time complexity for these algorithms with best and worst time complexities.

| Algorithm | Best Case | Worst Case |
| --- | --- | --- |
| Linear Search | O(1) | O(n) |
| Binary Search | O(1) | O(logn) |
| Insertion Sort | O(n) | O(n^2) |
| Selection Sort | O(n^2) | O(n^2) |
| Bubble Sort | O(n) | O(n^2) |



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

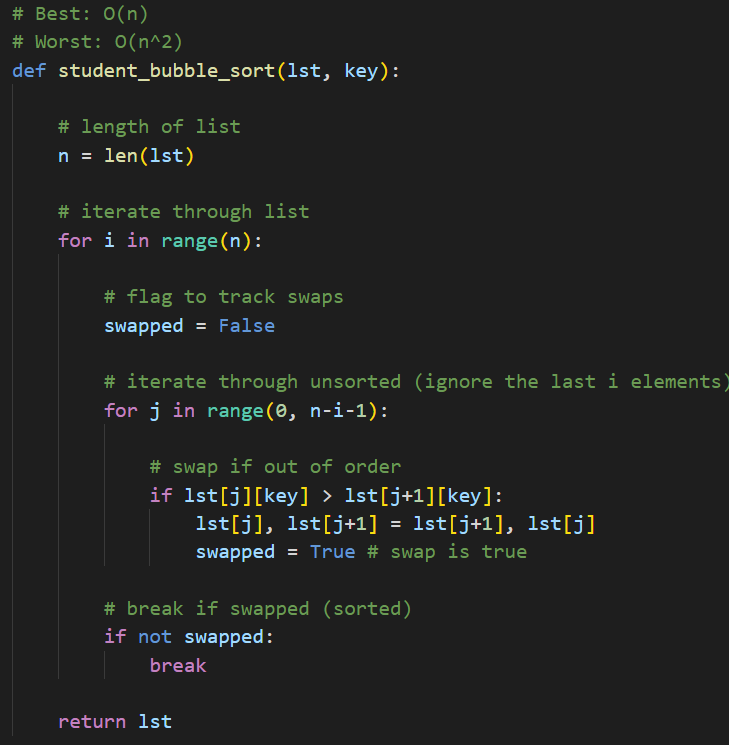
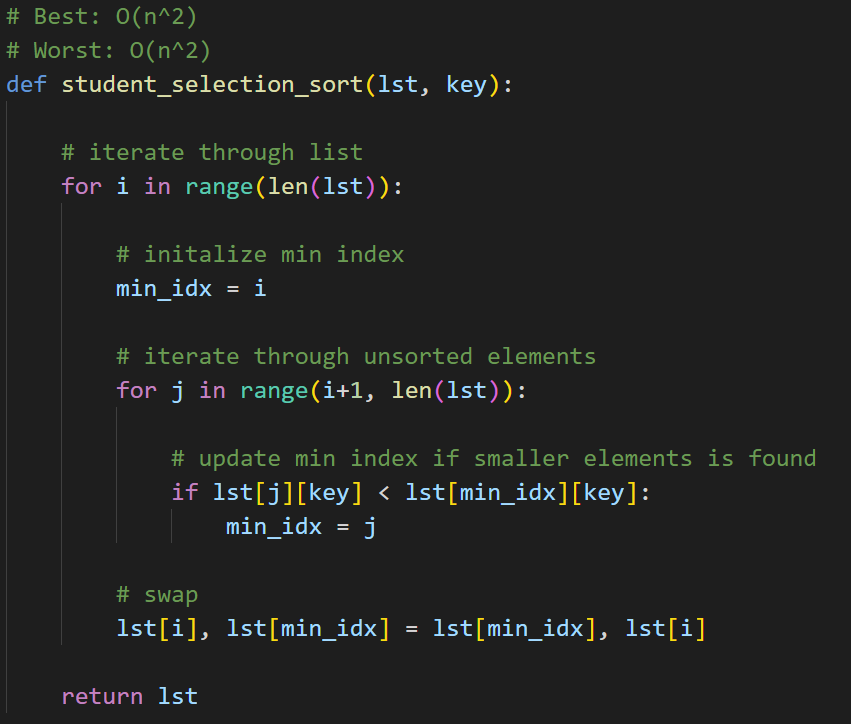
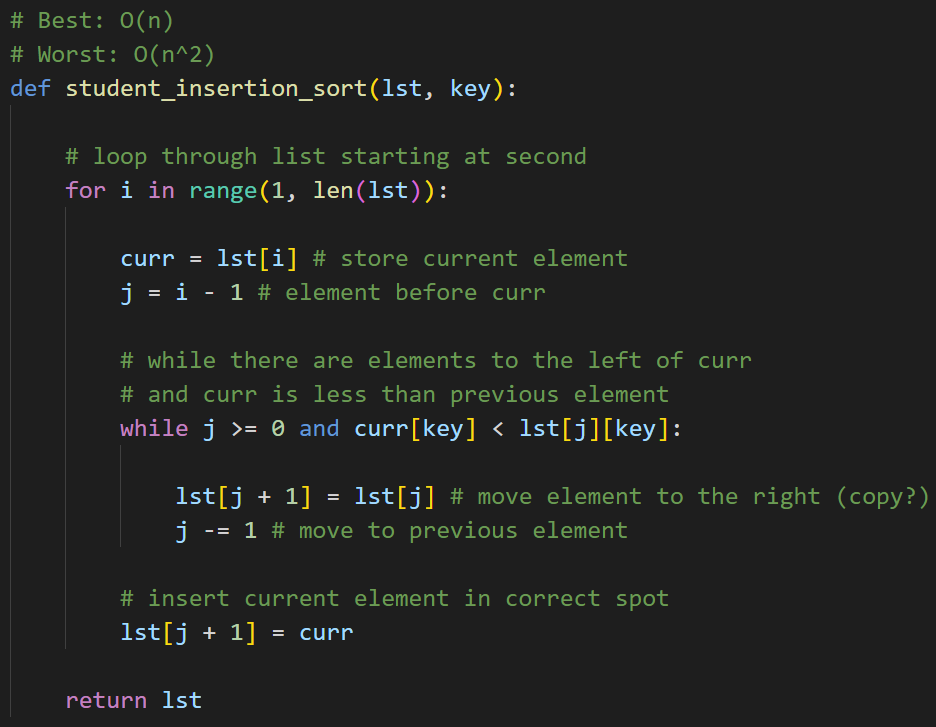
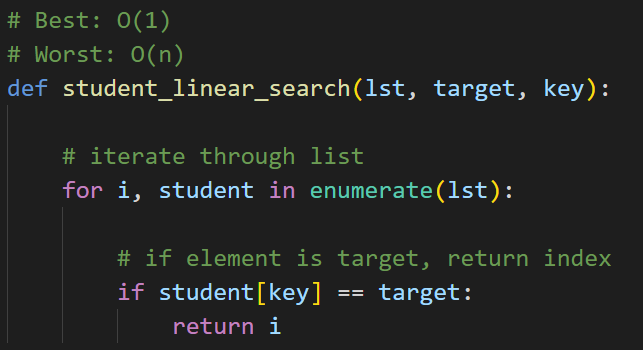
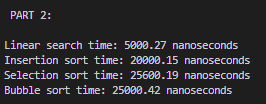
* Student ID
* first name
* last name
* email id
* Major
* Write a program to read the data from the text file. Choose an appropriate data type and data structure (lists, lists of list, dictionary) for storing the information in your program.



* 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 list using student id or first name or last name. Implement the sorting using selection sort, insertion sort and bubble sort, and print out how much cpu time it took to sort the data. You can import a library to calculate the time.

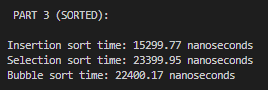
Show an example for searching a value using linear search. Table-2: Tabulate your recorded time for the linear search and all the four sorting algorithms i.e., selection sort, insertion sort and bubble sort.

| Algorithm | Time 1 |
| --- | --- |
| Linear Search | 5000.27 ns |
| Insertion Sort | 20000.15 ns |
| Selection Sort | 25600.19 ns |
| Bubble Sort | 25000.42 ns |



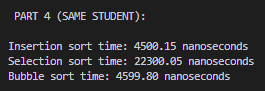
* Table-3: Now apply all the four sorting algorithms on the sorted data (i.e., sort the text file according to student id / first name / last name where the text file is already sorted) and tabulate your recorded time. Print out how much CPU time it took to sort the data.

| Algorithm | Time 1 |
| --- | --- |
| Insertion Sort | 15299.77 ns |
| Selection Sort | 23399.95 ns |
| Bubble Sort | 22400.17 ns |



* 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.

| Algorithm | Time 1 |
| --- | --- |
| Insertion Sort | 4500.15 ns |
| Selection Sort | 22300.05 ns |
| Bubble Sort | 4599.80 ns |



* 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?

* Table 1
  + This was basically to allow us to analyze the time complexities of the different algorithms, specifically the best and worst case. I believe the goal was to use this as a reference when doing steps 2,3, and 4 to compare actual results to the theoretical results.
* Table 2
  + This allowed us to see the actual results of using these algorithms. After optimizing bubble sort, all of the algorithms were pretty close together, with insertion sort being a bit more efficient than the other two, but they were all pretty close.
* Table 3
  + They all worked a bit faster on the pre sorted data. Again, insertion outperformed the other two, while bubble sort slightly outperformed selection sort.
* Table 4
  + This is how I would expect the results to go. I was expecting insertion and bubble to perform similarly, while selection was very inefficient in comparison.

I think it might have been the way I implemented bubble sort at first, but I was not expecting the results I got for that particular algorithm for parts 2,3, and 4 the first time I did this. The other times fell in line with expectations, though. With that said, I went back and added a swapped flag to bubble sort to break out of the loop if no swaps were made and the results were more aligned with expectations.

Overall, it was interesting to see the actual application differences between the algorithms in a real scenario, even if it is a small dataset. It was also interesting to see how optimization can greatly affect performance (bubble sort in this case).