# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Fall 2021** |
|  | | | | | |
| Points |  | Document author: | **Saishnu Ramesh Kumar** | | |
|  | Author’s email: | **Saishnu\_rk@mail.fresnostate.edu email** | | |
| Laboratory number: | **Session 1, 11am to 12:50pm** | | |
|  | | | | | |

**1. Statement of Objectives**

For this lab report, we were instructed to write two different C++ programs that dealt with sorting algorithms, specifically Selection and Merge Sort. Understanding the time complexity and how both algorithms work in the code format is important as it can be used for sorting a big list of numbers in a small amount of time. Being able to write the code for both, Selection and Merge would help develop a better understanding of these algorithms as well as the other types of sorting algorithms available.

**2. Experimental Procedure**

**Selection Sort:**

For selection sort, there were three functions introduced: swap(), selectionSort(), and printArray(). The swap function just contained the formula to swap the elements around. The selectionSort function contained the sorting algorithm section where the length of the array was checked and each number in the array list was checked and sorted out in order while the swap function was called to help with moving the elements of the array around. The printArray() function calls the array so that it could be outputted onto the terminal screen. For each array list, the original, semi-sorted, and reverse list, all of their execution times were also recorded as the speed at which the selection sort may sort each problem would either be faster or slower. Moreover, the time complexity of Selection Sort is O(n2) which would mean that it is not the most efficient algorithm as compared to merge sort.

**Merge Sort:**

As for merge sort, there were 4 functions used for the program: swap(), merge(), mergeSort(), and printArray(). The swap function worked the same as the swap function used in the selection sort program. The merge function was created so that it could take the original array apart and divide it into a subarray to help with the sorting, it would use the divide and conquer method where the array list would get broken down until each element is on its own and it would then merge back but in an ordered styled. The mergeSort function is there to recursively call on itself and to call the merge function for the sorting phase. Also, the printArray function would just output the entire list on the terminal. For merge sort, the time complexity for this algorithm is O(nlogn), which would mean that this algorithm is much more efficient in sorting bigger lists of elements as compared to selection sort.

**3. Analysis**

**Selection Sort Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Selection Sort** | **Time Complexity** | **Execution Time for Original (secs)** | **Execution Time for Semi-Sorted (secs)** | **Execution Time for Reverse (secs)** |
| - | O(n2) | 4e-07 | 2e-07 | 1e-07 |

**Merge Sort Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Merge Sort** | **Time Complexity** | **Execution Time for Original (secs)** | **Execution Time Semi-Sorted (secs)** | **Execution Time for Reverse (secs)** |
| - | O(nlogn) | 3.9e-06 | 1.1e-06 | 8e-07 |

**4. Encountered Problems**

There were numerous amounts of compiler errors, but it was solved after watching some videos as well getting some help from websites. I also faced some difficulties in attempting to write the code for merge sort, despite understanding the theoretical concept I struggled for a little when writing the code, but I was able to get it done.

**5. Conclusions**

In this lab, I learnt and got a better understanding of how to actually code both of these algorithms because I realized that even if I understand the theoretical concept of it well, trying to code these algorithms are a challenge. Understanding which algorithm is more efficient than the others as well and knowing the different time complexities for each of them were also interesting. I believe that I have got a better grasp of how algorithms work now, especially in the coding part of the topic.

**6. References**

1. Slides provided by TA during the lab session.
2. <https://www.geeksforgeeks.org/selection-sort/>
3. <https://www.geeksforgeeks.org/merge-sort/>
4. <https://www.youtube.com/watch?v=mB5HXBb_HY8>