Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report 2

[Code No: COMP 314]

Submitted by:

Jiwan Humagain

Roll No:19

Group: Computer Engineering

Level: III year/II sem

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date:2024-05-23

3. Generate some random inputs for your program and apply both quick sort and merge sort algorithms to sort the generated sequence of data. Record the execution times of both algorithms for best and worst cases on inputs of different sizes. Plot an input-size vs execution-time graph.

Description

In this lab we learnt about the time complexities of merge and quick sort using different test cases and making their graphs. Initially, we learnt about testing the program using different test cases. After that we write code for merge and quick sort and use test cases to test the correctness of the code.

We then generate random numbers and store them in an array and then we plot the graph of those data vs the time required to sort those data using both selection and insertion sort. That was all we did in this lab.

Output

```
m/algo lab/Lab 2/merge sort/test.py"
.....
----------------------------------------------------------------------
Ran 5 tests in 0.020s

OK
```

Image 1: Testing different function of merge sort using unittest

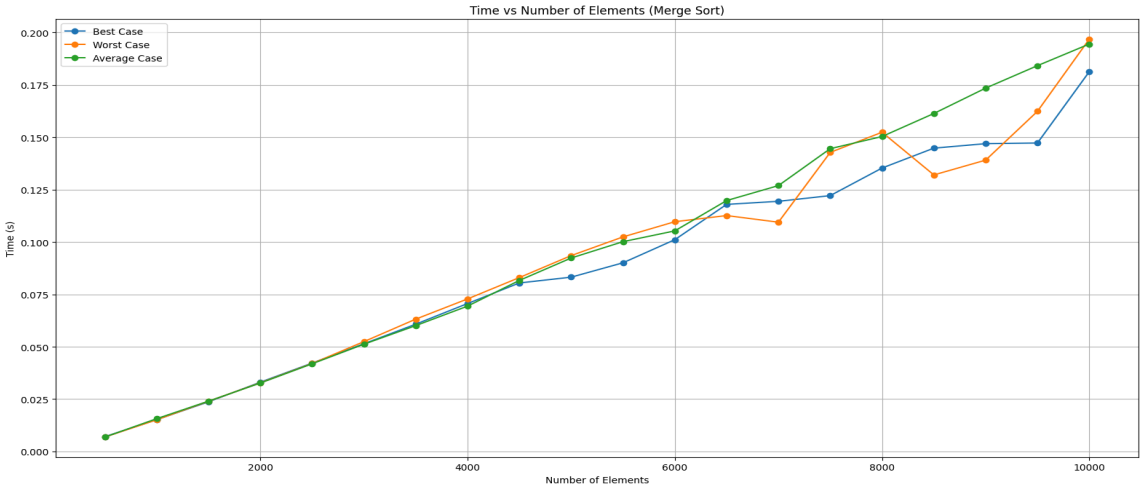

Image 2: Merge sort cases

```
m/algo lab/Lab 2/quick sort/test.py"
.....
----------------------------------------------------------------------
Ran 5 tests in 0.008s

OK
```

Image 1: Testing different function of quick sort using unittest

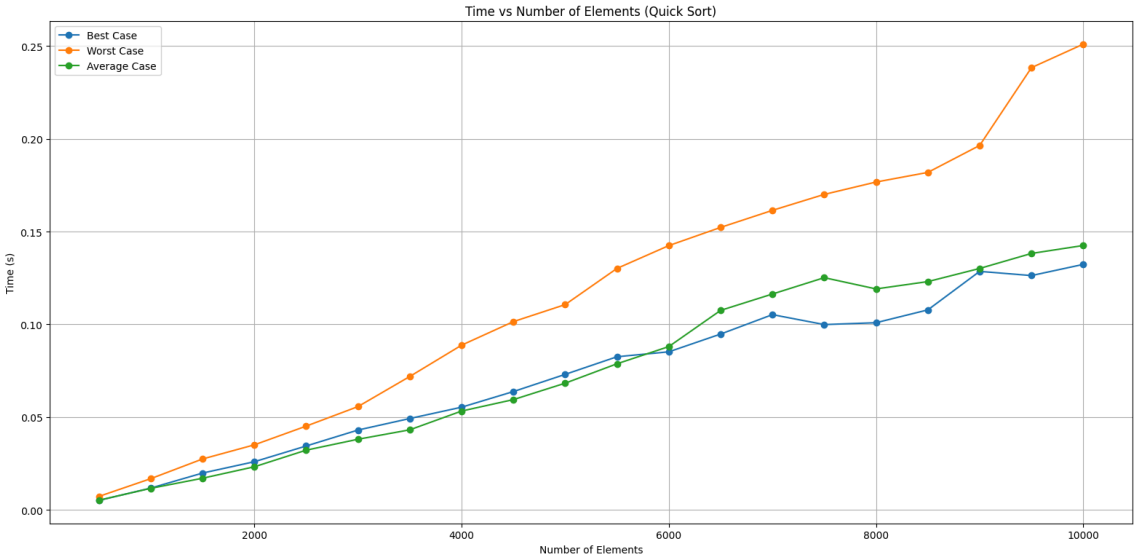

Image 4: Quick sort cases

It is seen clearly that the best case average case and worst case for quicksort is O(n* log n), O(n* log n), O(n^2) respectively. Similarly for merge sort all three cases are same i.e. O(n* log n).

4. Compare your results with those from Lab 1, and explain your observations.

Merge Sort and Quick Sort are efficient, with average time complexities of O(n log n), suitable for large datasets. Merge Sort is stable and consistent across all cases but uses more memory. Quick Sort is generally faster due to better cache performance but can degrade to $O(n^2)$ in the worst case without good pivot selection, although this is rare with optimizations like median-of-three. Selection Sort and Insertion Sort are simpler, with $O(n^2)$ time complexities, making them inefficient for large datasets. However, Insertion Sort performs well on small or nearly sorted arrays due to its O(n) best-case complexity.