**CA284: New Systems Programming**

**Project Report**

**“Analysis of Sorting Algorithms”**

**STATEMENT OF NON-PLAGIARISM**

I hereby declare that all information in this assignment has been obtained and presented in accordance with academic rules and ethical conduct and the work I am submitting in this document, except where I have indicated, is my own work.

Student Number: 18314236

Student Name: Muhammad Zubair Asif

Date: 10/01/2021

**Signature**

1. **Introduction**

This is a report on the CA284: New-Systems-Programming Analysis of Sorting Algorithms project. For this project I decided to chose selections sort, bucket sort and heap sort algorithms. The reason I chose selection sort was because it is one of the reasonably simpler algorithms, and I wanted to start off light. I then chose to go with bucket and heap sort to increase difficulty. I chose each due to their differing time complexities; Heap - O(n log n), Bucket - O(n + k), Selection - O(n^2)

1. **Datasets**

For this project I decided to create a function alongside my algorithms which creates 7 random arrays of integers of varying data sizes and limits based off a specified range(displayed below). I decided to included the generate\_random function within the algorithms instead of in a separate program in order to run and test quicker with less of a hassle. The function uses seed values to generate the same random numbers for each algorithm allowing efficient comparison. In order to demonstrate good algorithm performance I chose varying different array sizes ranging from 100 to 100million.

…

100

1,000

10,000

100,000

1,000,000

10,000,000

100,000,000

1. **Algorithm Performance**

Before running the tests I had an inclination of how the programs would preform due to their time complexities.

|  |  |  |
| --- | --- | --- |
| SELECTION SORT | BUCKET SORT | HEAP SORT |
| 0.000 sec - 100  0.004 sec - 1,000  0.348 sec - 10,000  36.159 sec - 100,000  N/A - 1,000,000  N/A - 10,000,000  N/A - 100,000,000 | 0.016 sec - 100  0.013 sec - 1,000  0.014 sec - 10,000  0.018 sec - 100,000  0.030 sec - 1,000,000  0.383 sec - 10,000,000  3.460 sec - 100,000,000 | 0.000 sec - 100  0.000 sec - 1,000  0.006 sec - 10,000  0.081 sec - 100,000  1.119 sec - 1,000,000  14.760 sec - 10,000,000  221.892 sec - 100,000,000 |

*\*N/A = Did not finish/Took too long*

However, I was kind taken aback by some of the results.

Selection sort:

This algorithm was by far the worst out of the three algorithms. There was a clear fluctuation of time between 10,000 and 100,000, the other algorithms seemed to have gradual increase as the size of data increases, but selection sort was not as measured. Selection sort performed very poorly on larger data sizes it was very inefficient with data sizes above 100,000.

Bucket sort:

This was my first experience with bucket sort, and I was very impressed with the performance. It worked very well with both large and small data sizes. The increase in data size was substantial, however, the time for each to be sorted was very gradual. This was expected due to its time complexity.

Heap sort:

This algorithm was near instantaneous on data sets below 1,000. It was not as gradual with sorting times like bucket sort. As the size of the data increased so did the time to sort it, quite substantially in fact. There was a large fluctuation from 10,000,000 to 100,000,000.

1. **Negatives**

I do not believe that there were necessarily any negatives in fact I believe I regulated my project and my testing very well. However, there are some aspects which if I were to redo the project I would change. One of them being the decision of choosing selection sort. Selection sort is a very poor performer on larger data sets. It became very inefficient in obtaining results of data sets above 100,000. I would preferably chose some other algorithm maybe insertion, quick or even counting sort.

When first trying to implement the algorithms on data sets, I was facing segmentation faults. This caused major hindrance in the time taken to complete the project. The issue arose from maximum allowed space being less than the data size.

I would also try to be more creative with my data sets not just in range but also include negatives and even possibly letters. Furthermore, I would reverse the data and test it. Moreover, I would test half sorted data.

1. **Conclusion**

I had previously worked on sorting algorithms in other modules, However, I was never exactly sure how each algorithm worked and how long a certain algorithm would take on certain data sets. This project has given me a better understanding of time complexities and algorithm functionality. If I had more time at my disposal I would try to test a bigger range of data with more of a variance and I would add more algorithms as I believe three is not enough to fully compare data well, the more the algorithms the better the comparisons will be. In conclusion, the best performer was bucket sort, followed by heap sort and in last place is selection sort.