**Sorting Algorithms Empirical Time Efficiency Comparison**

**Objective**: To analyze how different algorithms solving the same problem produce different computational results, and to experiment with empirical analysis of algorithms

Introduction:

Sorting is one of the most common tasks performed by our computers. In this project, I will be analyzing the efficiency of sorting algorithms (Bubble sort, Insertion Sort, Quick Sort, Merge Sort, and Selection Sort). Let’s briefly define what these algorithms are and their time complexities.   
Bubble sort is the simplest sorting algorithm that works by repeatedly swapping elements if they are in wrong order. The average time complexity of bubble sort is O(n^2). The selection sort sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning. At every iteration of selection sort, the minimum element from the unsorted subarray is picked and moved to the sorted subarray. The average time complexity of selection sort is O(n^2). Insertion sort is a sorting algorithm that builds the final sorted array one at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The worst-case time complexity of insertion sort is O(n^2). Merge sort is a divide and conquer algorithm, which divides the input array into two halves, calls itself for those two halves and then merges the two halves. The average case time complexity of merge sort is O(n logn). Quick Sort is Divide and Conquer Algorithm, which picks an element as pivot and partitions the given array around the picked pivot. The average case time complexity of quick sort is O(n logn).

To complete this project, I primarily used Eclipse to run my java program and excel to create the charts. The first step in this project was to generate files with random numbers ranging from 0 to 50000000. I wrote a java program to accomplish this task. The next step was to implement the algorithms. I used the reference from geekforgeek.com website to implement the sorting algorithms in Java. Then, I wrote a java program that creates a user interface to let users select the input size, and the type of algorithm, which would ultimately return the time it takes for that algorithm to process the selected input size.

* **Feature of the program:**

1. The program is user friendly as the user interface allows user to select specific input size and the algorithm type, they want to run
2. The program helps in computing the total time taken by a specific algorithm to sort a specific input size

Here is a picture of a user selecting the data size and algorithm type to compute the running time

Graphical user interface, text, application

Description automatically generated

**Observation**

**Bubble Sort**

Chart

Description automatically generated  
  
  
From the above chart, we can see that as our input size increases, the time it takes for the bubble sort to sort these numbers increases. As in the chart above, we see that it takes 403109019 nano seconds (0.4 seconds) to sort the file with 5000 random numbers, 2688188605nano seconds (2.69 seconds) to sort 25000 random numbers, 36794199685 nano seconds(36.79 seconds) to sort 80000 random numbers, 1.466\*10^(11) nano seconds ( 2.44 minutes) to sort 150000 random numbers, and 6.22955\*10^11 ns (10.38 minutes) to sort the file with size 300000. So, we can conclude from our observation that as input size increases, the time it takes for bubble sort to sort the contents increases considerably faster. I think that the reason behind bubble sort taking a lot of time is that bubble sort makes multiple passes through our list. It compares adjacent numbers and exchanges those that are out of order. In each pass, it places the next largest value in its proper place. This results in the time complexity of bubble sort to be O(N^2). Because of these numerous comparisons and swaps, bubble sort takes a long period of time when sorting our data.

**Selection Sort**

Chart

Description automatically generated

As in the chart above, we see that it takes 690348366.8nano seconds ( 0.69 seconds) to sort the file with 5000 random numbers, 2978624156 nano seconds (2.97 seconds) to sort the file with 25000 random numbers, 52027440796 nano seconds (52 seconds) to sort the file with 80000 random numbers, 2.2274\*10^11 seconds (3.71 minutes) to sort the file with 150000 random number, and 7.3148\*10^11 nano seconds (12.19 minutes) to sort the file with 300000, So, we can see that as our input size increases, the time it takes for the selection sort to sort these numbers increases. I think that our selection sort takes a lot of time while sorting the data because it repeatedly loops over the array to find the minimum element in the array and then, moves to it to proper position in the array. The number of times selection sort passes through an array is one less than the number of items in the array. Furthermore, selection sort is in-place algorithm and has O(n^2) time complexity, which makes it inefficient on large lists.

**Insertion Sort**

Graphical user interface, chart

Description automatically generated

As in the chart above, we see that it takes 79856968.8nano seconds ( 0.07 seconds) to sort the file with 5000 random numbers, 621124983.8nano seconds (0.62seconds) to sort the 25000 random numbers, 6589640098 nano seconds (6.54 seconds) to sort 80000 random numbers, 48259447390 nano seconds (48.26 seconds) to sort 150000 random numbers, and 3.92818\*10^11nano seconds (6.54 minutes) to sort the file with 300000 random numbers. I think that this algorithm is taking long period of time because the insertion sort algorithm iterates through the elements in our data and removes one element per iteration, finds the place the element belongs in the array and then places it there. This process grows a sorted list from left to right. While doing so, the algorithm takes a lot of time while sorting our data.

**Quick Sort**

Chart, line chart

Description automatically generated

As in the chart above for the quick sort, it takes 8249079ns (0.00824 seconds) to sort the file with 5000 random numbers,12895850ns (0.01289 seconds) to sort 25000 random numbers, 38067596 nano seconds(0.03807 seconds) to sort 80000 random numbers, 60323403.8nano seconds(0.06 seconds) to sort 150000 random numbers, and 106475067.2 ns (0.10648 seconds) to sort the file with 300000 random numbers. Quick sort has been able to sort our random data very fast in comparison to the algorithms before because we know that quick sort is a divide and conquer algorithm which relies on a partition operation, where a pivot is selected and all the elements smaller than the pivot are moved before it and all greater elements are moved after it. Furthermore, the average time complexity for Quick Sort is O(n logn). Through this observation, we can say that Quick Sort vastly outperforms selection, bubble, and insertion sort.

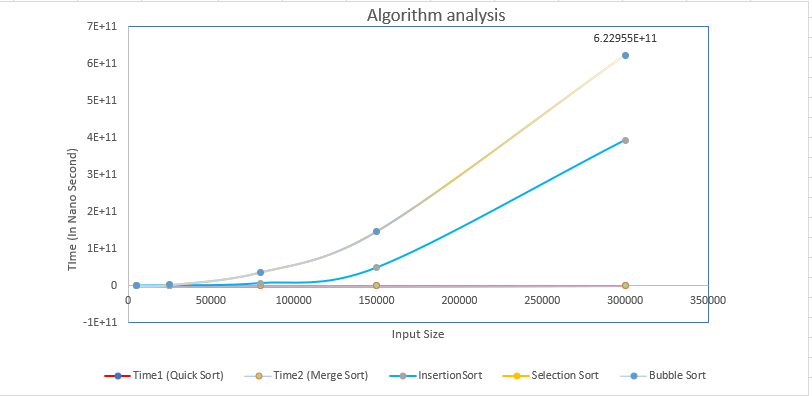
**Merge Sort**

Chart, line chart

Description automatically generated

The above chart represents the efficiency of merge sort algorithm while sorting files of various sizes with random numbers. As seen in the chart above, it takes 9198749.2 ns (0.009 seconds) to sort the file with 5000 random numbers, 22617497 nano seconds (0.0226 seconds) to sort the file with 25000 random numbers, 55824784.8 nano seconds (0.0558 seconds) to sort the file with 80000 random numbers, 73331729.6 nano seconds(0.073 seconds) to sort the file with 150000 random numbers, and 120556230.8ns (0.12 seconds) to sort the file with 300000. Based on the data, we can notice that merge sort is considerably faster than insertion, selection, and bubble sort. I think this is because merge sort is a divide and conquer algorithm and in our program, merge sort is dividing our unsorted list into n sub-lists, each containing one element and repeatedly merge sub-lists to produce new sorted sub-lists until there is only one sub-list remaining. This leads to the time complexity of merge sort to have O(n logn) time complexity, which is very efficient.

**Comparing all algorithms in single graph**

  
  
  
Note

* The yellow line (Selection Sort) is overlapping the blue line (Bubble Sort)
* The orange line (Quick Sort) is overlapping the green line(Merge Sort)

In the chart above, we can compare the time it took for all our 5 algorithms to sort the files with random numbers. We can notice that selection sort and bubble sort overlap each other and take the highest amount to time to sort the numbers. Insertion Sort is faster than bubble sort and selection sort. On the other hand, Merge Sort and Quick Sort are the fastest ones, taking the least amount of time to sort our files. We also notice that Insertion Sort, Bubble Sort, and Selection Sort take a lot of time, when sorting large files. On the other hand, the time it took for merge sort, and quick sort to sort large files is the least and is very close to one another. The above graph also supports that the average case running time of merge sort, and quick sort O(n\* logn) is far better than the average running time of bubble sort, insertion sort, and quick sort, O(n^2), while sorting random numbers.   
  
  
  
  
  
  
Extra Credit

**Comparing Algorithm efficiency for sorting data sets containing sorted increasing numbers**

A picture containing graphical user interface

Description automatically generated

Chart, line chart

Description automatically generated

Note- Merge Sort, Insertion Sort, and Quick Sort perform very similar. In the chart above, quick sort overlaps merge sort and insertion sort.

As seen in the chart above, selection sort takes a huge among of time while sorting large files. Bubble sort also takes a lot of time while sorting the files, however after it reaches certain point, the time it takes for it to sort larger files remains constant. Merge Sort performs better than bubble sort and selections sort. Insertion Sort perform the best, as it takes the least amount of time while sorting the data. However, this result fails to support that the quick sort worst case occurs when array is sorted in ascending order because as seen it the data, quick sort performs considerably better than bubble sort and selection sort.

Chart, line chart

Description automatically generated

Furthermore, when comparing our merge sort, quick sort, and insertion sort, we can see that **insertion sort outperforms merge sort and quick sort. This is because of the fact that our data is already sorted in ascending order and that the insertion sort algorithm just iterates over N elements to check if all the elements are in proper place or not, making it the most efficient algorithm in this case.**

**Comparing Algorithm efficiency for sorting data sets containing sorted numbers in descending order**

Chart, line chart

Description automatically generated

From the chart above, we can see that bubble sort takes the most amount of time while sorting the files with sorted data in descending order. Selection Sort and Insertion Sort performs better than bubble sort. Merge sort and Quick Sort perform better than all other algorithms in this case.

Chart, line chart

Description automatically generated

As seen it the chart above, quick sort performs fasters than merge sort. However, this doesn’t align with our pre-existing knowledge of quick sort having the worst case while sorting the sorted array in descending order. It might be because of our implementation of quick sort algorithm in our program.

**So, how many times are the sorting algorithms faster than one another?**

Based on the data above, while sorting the files with random data of size 5000, 25000, 80000, 150000,300000, quick sort is almost two times faster than merge sort. Similarly, for the same files, selection sort was 1.5 times slower than bubble sort. When we compare insertion sort, and quick sort, quick Sort is 8.5 times faster for the file of size 5000,48 times faster for the file of size 25000, 172 times faster for the file of size 80000, 804 times faster for the file of size 150000, and 3924 times faster for the file of size 300000.When we compare bubble sort and quick sort, quick sort is almost 5 times faster for the file of size 5000,208 times faster for the file of size 25000, 968 times faster for the file of size 80000, 2440 times faster for the file of size 150000, and 6228 times faster for the file of size 300000.

**Conclusion**

Based on the results above, we can say that the efficiency of a particular algorithm depends on what type of array it is sorting (unsorted array, sorted array in ascending order or sorted array in descending order). For example, for an unsorted array, bubble sort, selection sort, and insertion sort perform the worst, whereas the merge sort and quick sort performs the best. This is also supported by our pre-existing knowledge of these two algorithms being the fastest ones. Furthermore, we can also conclude that insertion sort is the most efficient algorithm when it comes to sorting a sorted array (ascending order). Despite of our common understanding of fastest sorting algorithms being quick sort, and merge sort, this project shows that for some scenarios, such as a sorted array, insertion sort can perform much better. Finally, we can conclude that the average case running time of merge sort, and quick sort (n \* logn) is much better than the average case running time of bubble sort, insertion sort, and selection sort O(n^2).

**Suggestions for Future Enhancement and Modification**

I think that there can be some enhancement made to this project in future. While, I was running the sorting algorithms, there were multiple programs running in the background in my computer, which might have affected the running time of algorithms. So, in future, stopping all background programs while running the program can definitely provide better accuracy in our results.

Reference (IEEE citation)

“QuickSort,” *GeeksforGeeks*, 04-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/quick-sort/. [Accessed: 31-Oct-2020].

“Bubble Sort,” *GeeksforGeeks*, 30-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/bubble-sort/. [Accessed: 25-Oct-2020].

“Selection Sort,” *GeeksforGeeks*, 30-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/selection-sort/. [Accessed: 20-Oct-2020].

“Insertion Sort,” *GeeksforGeeks*, 30-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/insertion-sort/. [Accessed: 15-Oct-2020].

“Merge Sort,” *GeeksforGeeks*, 30-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/merge-sort/. [Accessed: 17-Oct-2020].