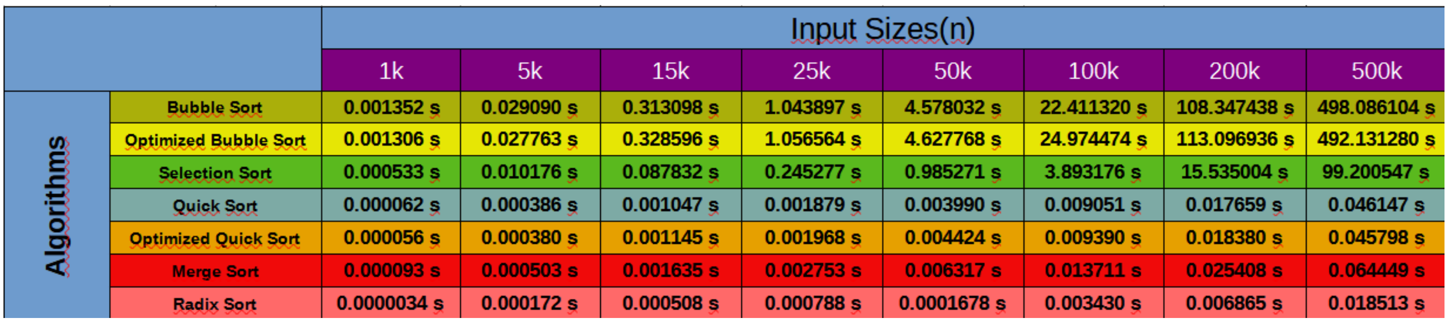
# CSE-311 Project

Hasan Güzelşemme
20220702052

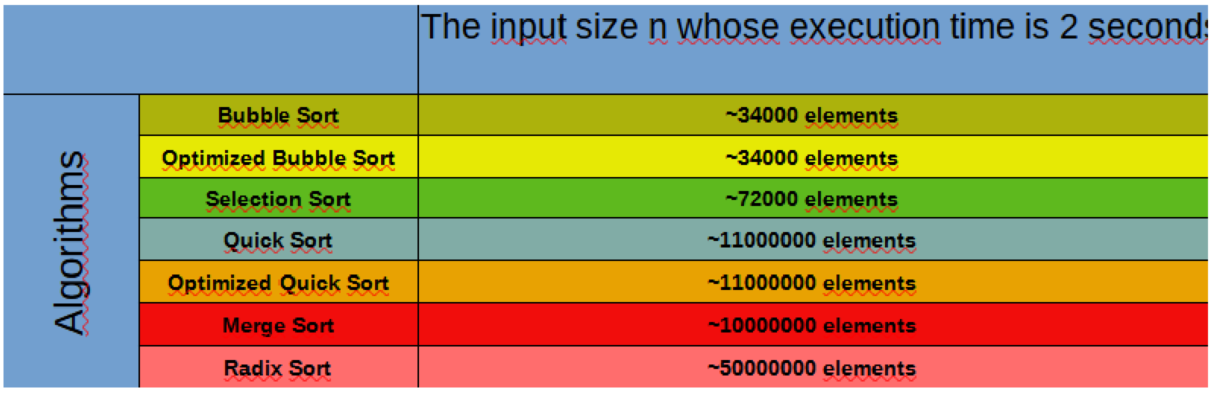
During this project, I have calculated the time taken for 7 sorting algorithms to sort arrays of certain sizes. Here are the results obtained:



| | | Input Sizes(n) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1k | 5k | 15k | 25k | 50k | 100k | 200k | 500k |
| Algorithms | Bubble Sort | 0.001352 s | 0.029090 s | 0.313098 s | 1.043897 s | 4.578032 s | 22.411320 s | 108.347438 s | 498.086104 s |
| | Optimized Bubble Sort | 0.001306 s | 0.027763 s | 0.328596 s | 1.056564 s | 4.627768 s | 24.974474 s | 113.096936 s | 492.131280 s |
| | Selection Sort | 0.000533 s | 0.010176 s | 0.087832 s | 0.245277 s | 0.985271 s | 3.893176 s | 15.535004 s | 99.200547 s |
| | Quick Sort | 0.000062 s | 0.000386 s | 0.001047 s | 0.001879 s | 0.003990 s | 0.009051 s | 0.017659 s | 0.046147 s |
| | Optimized Quick Sort | 0.000056 s | 0.000380 s | 0.001145 s | 0.001968 s | 0.004424 s | 0.009390 s | 0.018380 s | 0.045798 s |
| | Merge Sort | 0.000093 s | 0.000503 s | 0.001635 s | 0.002753 s | 0.006317 s | 0.013711 s | 0.025408 s | 0.064449 s |
| | Radix Sort | 0.0000034 s | 0.000172 s | 0.000508 s | 0.000788 s | 0.0001678 s | 0.003430 s | 0.006865 s | 0.018513 s |

Here you can see how the given algorithms behave as the input size grows.

Also, the maximum input sizes that each algorithm can sort in a two second time window is given below:



| | | The input size n whose execution time is 2 second |
|---|---|---|
| Algorithms | Bubble Sort | ~34000 elements |
| | Optimized Bubble Sort | ~34000 elements |
| | Selection Sort | ~72000 elements |
| | Quick Sort | ~11000000 elements |
| | Optimized Quick Sort | ~11000000 elements |
| | Merge Sort | ~10000000 elements |
| | Radix Sort | ~50000000 elements |

From these observations we can get some idea about the given sorting algorithms. These observations are as follows:

**Bubble Sort/Optimized(Improved) Bubble Sort:** Despite being easy to implement, these two sorting algorithms works best only and only for smaller sized arrays. As the arrays grow, the function follows $n^2$ growth rate and gets worse as input size goes to infinity.

The optimized/improved version of it checks if the array is already sorted or not. If so it stops execution. However there is no practical use for it.

There are no practical usage for these algorithms except educational purposes.

**Selection Sort:** Similar to Bubble Sort, Selection Sort follows the $n^2$ growth rate. But selection sort excels in larger data sets compared to Bubble Sort because it has less number of swaps. Still, it is not feasible to choose among other algorithms when it comes to larger data sets. It is also used when data sets are minimal or educational purposes.

**Quick Sort/Optimized(Improved) Quick Sort:** A significant improvement compared to the aforementioned algorithms, Quick Sort is significantly faster. It follows the growth rate n*log(n) which is quite faster when it comes to larger data sets. You can see the drastic change of inputs sizes above.(where $n^2$ growth rate algorithms can sort inputs sizes of 5 digits, Quick Sort can sort input sizes of 8 digits.)

QuickSort is used in computer graphics: for image rendering and data visualiation among other things

**Merge Sort:** Another function that follows n*log(n) time complexity, Merge Sort falls just behind the Quick Sort algorithm performance-wise. However, If stability is needed, it would be wiser to use Merge Sort because it is a stable sorting algorithm, where Quick Sort isn't. It is good to use this algorithm on larger data sets.

This algorithm is used in file sorting within external storage systems, such as hard drives.

**Radix Sort:** Compared to rest of the algorithms, this algorithm shows the best performance. This was not an algorithm we saw before. It distributes the elements into buckets based on each digit's value. By repeatedly sorting the elements by their significant digits, from the least significant to the most significant, Radix Sort achieves the final sorted order. It follows the time complexity O(d * (n + b)), where d denotes number of digits, n denotes the input size and b denotes the base of the number system that is being used. It is certainly the best option when it comes to larger data sets out of the 7 algorithms we've compared.

In the modern era, radix sorts are most commonly applied to collections of binary strings and integers.

I have also made sure that generated numbers on the list are between 1 and 65936 in order for radix sort to work properly.

**Youtube Link for the 2 second input execution:** https://youtu.be/QuMYyC30EMY