Analysis of Sorting Algorithms

Gabriel Choong Ge Liang

May 21, 2023

1.1 Array Size

Comparing each sorting algorithm with different array sizes.

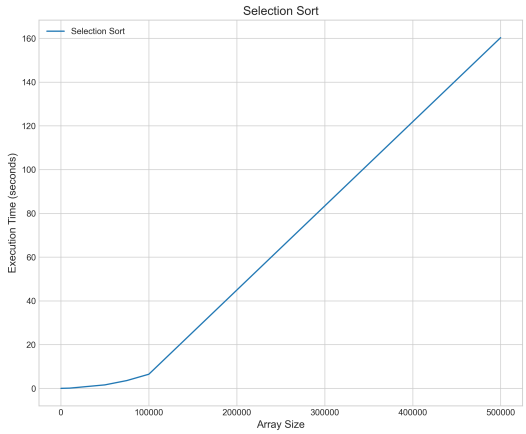

Figure 1: Comparison of Selection Sort with different array sizes.

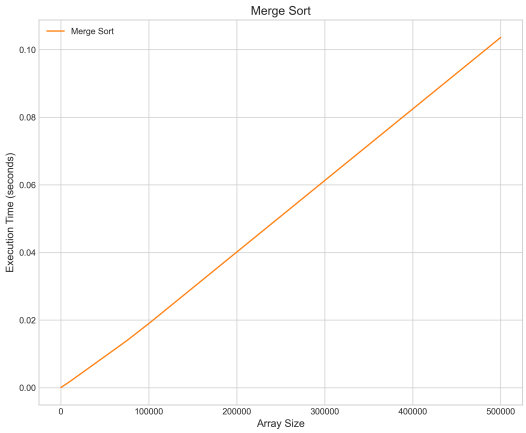

Figure 2: Comparison of Merge Sort with different array sizes.

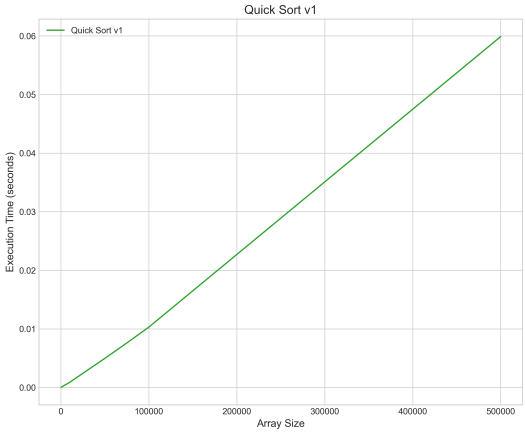

Figure 3: Comparison of Quick Sort V1 with different array sizes.

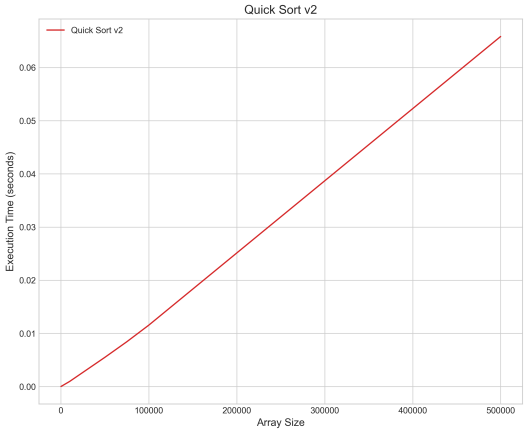

Figure 4: Comparison of Selection Sort V2 with different array sizes.

The graphs above compare the efficiency of sorting algorithms when the input array size varies. In Figure 1, the graph shows the quadratic function x^2. This is expected since Selection Sort has a time complexity of $O(n^2)$.

Figure 2 shows a curve that is close to a straight line. This is because Merge Sort has a time complexity of $O(n \log n)$. For large n, Merge Sort is more efficient than Selection Sort.

In Figure 3 and Figure 4, we compare two versions of Quick Sort algorithms, differing only in the pivot selection. The graph shows that the pivot selection slightly affects the efficiency of the algorithm. Theoretically, the second version should be more efficient than the first version. However, the graph shows that the first version is more efficient than the second version. This is because the second version has a higher constant factor due to the pivot swap with the last element of the array, while the first version does not require such a swap. Thus, the first version of Quick Sort is more efficient than the second version. Other factors, such as the random number generator used to generate the array, can also contribute to this difference in efficiency.

1.2 Algorithm

Comparing sorting algorithms on arrays with the same size.

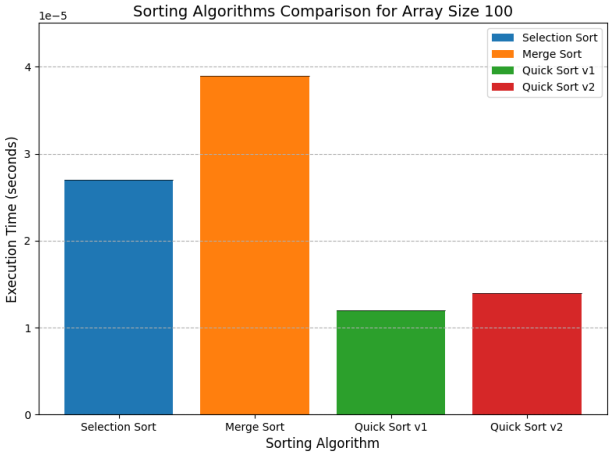

Figure 5: Comparison of Selection Sort, Merge Sort, and the two versions of Quick Sort when sorting an array of size 1000.

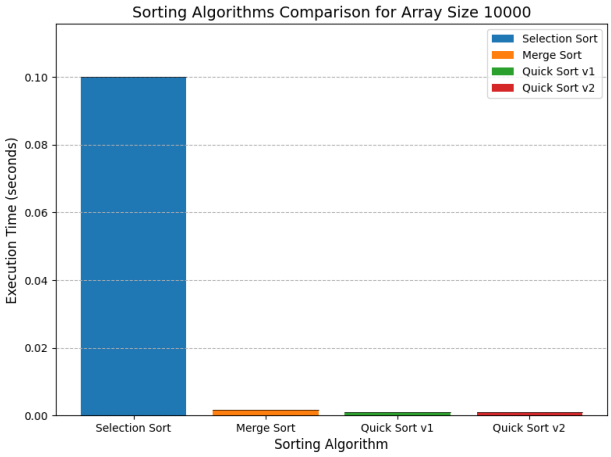

Figure 6: Comparison of Selection Sort, Merge Sort, and the two versions of Quick Sort when sorting an array of size 10000.

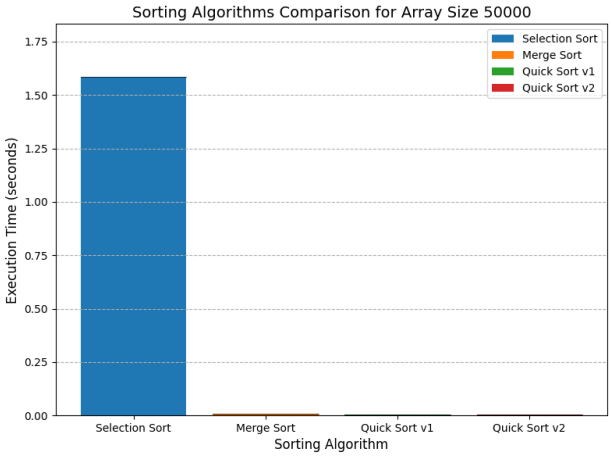

Figure 7: Comparison of Selection Sort, Merge Sort, and the two versions of Quick Sort when sorting an array of size 50000.

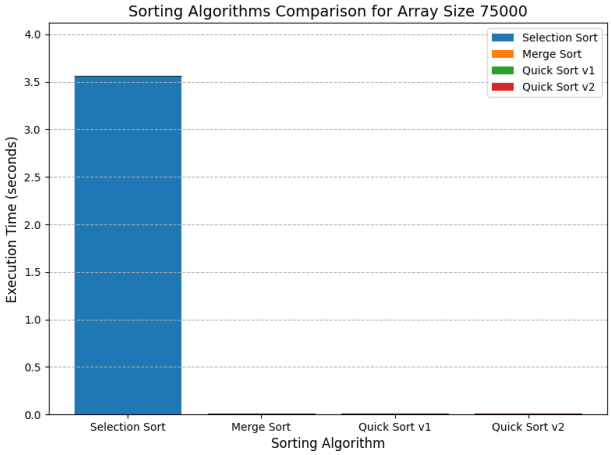

Figure 8: Comparison of Selection Sort, Merge Sort, and the two versions of Quick Sort when sorting an array of size 75000.

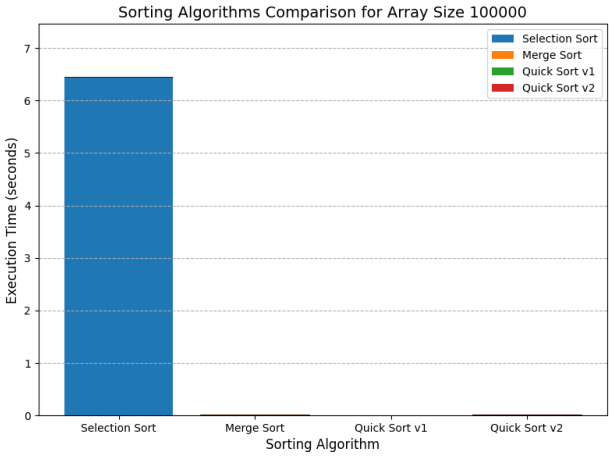

Figure 9: Comparison of Selection Sort, Merge Sort, and the two versions of Quick Sort when sorting an array of size 100000.

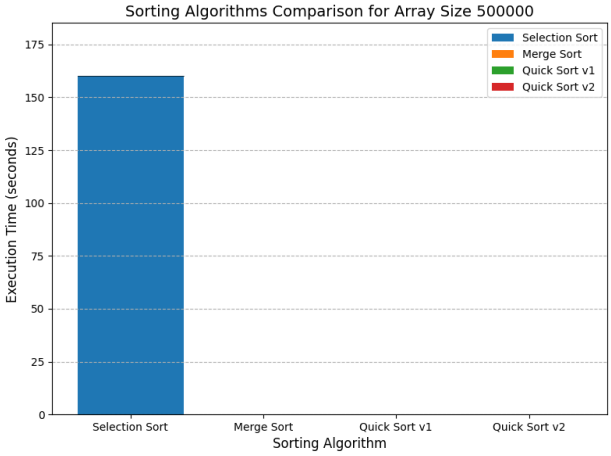

Figure 10: Comparison of Selection Sort, Merge Sort, and the two versions of Quick Sort when sorting an array of size 500000.

The graphs above compare the efficiency of sorting algorithms when the input array size is fixed. In Figure 5, the graph shows that Selection Sort is the least efficient algorithm. This is because Selection Sort has a time complexity of $O(n^2)$.

The graph also shows that the two versions of Quick Sort are the most efficient algorithms. This is because Quick Sort has a time complexity of $O(n \log n)$. Additionally, the graph demonstrates that the second version of Quick Sort is more efficient than the first version. The higher constant factor in the second version, due to the pivot swap with the last element of the array, makes the first version more efficient.

The graph further illustrates that Merge Sort is more efficient than Selection Sort, but less efficient than the two versions of Quick Sort. This is because Merge Sort has a time complexity of $O(n \log n)$, which is more efficient than Selection Sort, but less efficient than Quick Sort.