**Sorting Algorithms Report**

*Three sorting algorithms were implemented in this project, namely: Selection Sort, Merge Sort, and Quick Sort.*

**Selection Sort**

This sorting algorithm starts with the first element in the unsorted list and scans the rest of the list to find a smaller element. After a smaller element is found, the first element is replaced with the smallest element found. After the replacement, the algorithm comes to the second element and uses the same principle to replace the smaller element. This process continues until the list reaches (n - 1), after which the list is sorted. The static void method selectionSort(ArrayList<Integer> list) performs sorting after the elements are sorted. The size of the list is created by a for-loop in which random numbers are generated and the user determines the required size.

List sizes for testing: 1K, 10K, 50K, 100K.

The number of operations for each size:

1K - 500492 operations

10K - 50004987 operations

50K - 1250024987 operations

100K - 5000049041 operations

Operations were counted by comparisons and exchanges.

Chart, line chart

Description automatically generated

**Merge Sort**

In this algorithm, an unsorted list is continuously divided into two halves until only one element remains. This procedure is done through recursion, which stops after only one element remains in the list. After the division, a function is also called recursively that will join the halves and sort them as well. The sorting function is repeated until the last two halves are joined and sorted. The mergeSort(...) function splits the list until there is only one element left, and the merge(..) function merges them again and sorts them until the last two halves are joined and sorted together. The size of the list is created by a for-loop in which random numbers are generated and the user determines the required size.

List sizes for testing: 1K, 10K, 50K, 100K.

The number of operations for each size:

1K – 18648 operations

10K - 254014 operations

50K - 1502860 operations

100K – 3204689 operations

Operations were counted by comparisons and exchanges.

Chart, line chart, scatter chart

Description automatically generated

**Quick Sort**

This algorithm starts by creating a pivot and splitting the list around that element.

The pivot can be any element in the list. The list is constantly divided into two lists where the elements on the left side are smaller than the pivot while the elements on the right side are larger. In the quickSort(..) function, the index of the element smaller than the pivot and the index of the element larger than the pivot are found, while in the swap(..) function they are swapped. If the index for finding the smaller element is the same as the index for finding the larger element, the larger element is replaced with the pivot. . The size of the list is created by a for-loop in which random numbers are generated and the user determines the required size.

List sizes for testing: 1K, 10K, 50K, 100K.

The number of operations for each size:

1K – 13620 operations

10K - 187374 operations

50K - 1151951 operations

100K – 2589531 operations

Operations were counted by comparisons and exchanges.

Chart, line chart

Description automatically generated

**CONCLUSION:**

The selection sort algorithm has the fastest growth rate therefore it is the slowest of the three algorithms. It has O(n^2) average time complexity. Both merge and quick sort algorithms have O(n\*log n) average time complexity but it can be concluded from the data that quick sort is faster than merge sort, therefore is the fastest algorithm of all three.