Comparison of Quick Sort and Merge Sort Report

If we explain the working logic of merge sort and quick sort, Merge Sort and Quick Sort are popular algorithms for sorting a set of elements. Both algorithms have their own advantages and disadvantages.

The Merge Sort algorithm starts by first splitting an array element in half. It then splits each sub-array into sub-sub-arrays by splitting each sub-array in half again. This process continues until the sub-arrays have no more elements. Each part of the array is now considered ordered.

For sorting, it concatenates each pair of arrays starting from the lower sub-arrays. Since both arrays are ordered, the concatenation is easily performed and creates a larger array. The process is completed by merging all sub-sub-arrays.

During the concatenation process, the elements of the two arrays are compared and the smaller one is placed first. This creates an ordered array, since both arrays are ordered. This concatenation operation is performed during the concatenation of sub-sub-arrays, resulting in an array.

The Quick Sort Algorithm selects an element, called the pivot, and positions the elements on the right side of this element to the right of the pivot and the elements on the left side to the left of the pivot. Thus, the pivot element is placed in the center of the array.

The same process is then repeated recursively to sort the elements on the left-hand side. Usually the first element on the left-hand side is selected as the pivot and the other elements are positioned to the left or right of the pivot. This is also done for the right-hand side elements.

The recursive process continues until each sub-array has no more elements. At this point, all elements in the array are sorted. During the sorting process, the elements are compared to the pivot and the smaller ones are positioned to the left of the pivot and the larger ones to the right.

The Merge Sort Algorithm is implemented in a tree structure, where at each level the sub-arrays are split into two. Since merging is performed at each level, the tree height is log N. Since N operations are performed at each level, the time complexity of Merge Sort is O(N log N). Therefore, Merge Sort can sort large-sized arrays quickly and efficiently.

Quick Sort can have different time complexity depending on the choice of pivot and the structure of the array. In the best case, when the choice of pivot is good and the array is almost sorted, the time complexity is O(N log N). However, in the worst case, when the pivot is always the smallest or largest element, the time complexity can be O(N^2). Quick Sort is also efficient in terms of memory usage. It uses less memory than other sorting algorithms and is preferred for sorting large arrays quickly.

Merge Sort is a more stable algorithm than Quick Sort, and even in the worst case the time complexity remains O(N log N). Furthermore, Merge Sort is a preferred algorithm, especially for disk sorting operations, because it minimizes data transfer by better partitioning and sorting data.

Quick Sort generally runs faster and uses less memory than Merge Sort. However, in the worst case the time complexity can be O(N^2) and the stability of the algorithm can be low. Also, Quick Sort may not be preferred in some applications due to the risk of getting a Stack Overflow error when sorting large arrays.

In conclusion, both algorithms have their advantages and disadvantages and should be chosen according to the scenario in which they will be used. Merge Sort is a more stable and reliable algorithm, while Quick Sort can run faster but requires careful choices.

The project I created to test the Merge Sort and Quick Sort algorithms under different conditions provides test data under a total of sixty different conditions. I calculated the data in nanoseconds and created the table below, but before moving on to the table, I want to talk about a bug and its solution.

While testing the project, quick sort was giving stack overflow error in sorted arrays. I solved this problem by adding the -Xss26m parameter to the arguments section in the run config in eclipse. This parameter allows the JVM to set the stack size of the thread. A stack is a section of memory used in a program and each thread has a stack. The quick sort algorithm was giving a stackoverflow error because it was using too many recursive calls. Since the "-Xss26m" parameter increases the stack size of the Java Virtual Machine, this problem was solved.

Below I have recorded the data in nanoseconds for 60 different cases.

	Equal Integers			**Random Integers**			**Increasing Integers**			**Decreasing Integers**		
	1.000	**10.000**	**100.000**	**1.000**	**10.000**	**100.000**	**1.000**	**10.000**	**100.000**	**1.000**	**10.000**	**100.000**
mergeSort twoPart	597k	1,528k	14,656k	614k	2,755k	20,604k	854k	2,200k	14,536k	842k	2,631k	16,823k
mergeSort threePart	802k	2,266k	13,350k	801k	3,395k	19,407k	814k	2,308k	13,471k	800k	3,002k	15,042k
quickSort firstElement	3,554k	18,235k	903,935k	559k	2,848k	13,305k	3,403k	16,975k	887,103k	3,648k	31,519k	3,887,355k
quickSort randomElement	3,925k	22,053k	857,296k	744k	3,475k	18,430k	1,097k	4,075k	16,850k	1,550k	4,766k	17,171k
quickSort medianElement	3,136k	29,405k	958,962k	975k	3,651k	17,133k	1,107k	3,758k	14,656k	1,115k	4,608k	14,888k

- If we compare Merge Sort two-part and three-part within themselves, three-part is more efficient as the data grows, but two-part gives results faster when the data is smaller. In general, it can be said that two-part is more commonly used.

- If we compare QuickSort within itself, the "first element" method works fine with random data, but it is very inefficient and takes a long time to produce results when there is a possibility of data being already sorted. Therefore, using the "median" or "random pivot" method in QuickSort is much more reliable. The median method may be the most logical method because it minimizes time loss even when the data is already sorted. Considering the worst-case scenario, using the median method will be more efficient.

- If we generally compare QuickSort and Merge Sort, Merge Sort is more efficient for large data, but QuickSort gives faster results for small data sizes.

Which options should be used for the real-life examples given:

- If we have a Turkish-English dictionary sorted alphabetically and we want to convert this dictionary to an English-Turkish dictionary, we have a large amount of data, so we can use merge sort with three parts to perform this sorting. The reason for using merge sort instead of quick sort is that it is more efficient for large amounts of data, and the reason for using three parts is that it provides more efficient results for very large data sets.

- If we want to sort people in a family tree based on their birth dates, the data will consist of small data sets within itself, so using quick sort would be faster and more efficient in terms of memory usage. As the order of the data retrieval from E-Devlet is unknown, using the median pivot would be the safest algorithm.