Royal University of Phnom Penh

Faculty of Engineering

Department : Information Technology Engineering

Instructor : Kor Sochea

Course : Operating System

Name : Nath Sovanroth

Year : 3 (Generation 7th)

Semester : 1

Table of Contents

I. Introduction

In everyday life, sorting is important, and it is also a basic computer science process that arranges data in a specific order. It is used to arrange the data in order for it to be searched, analyzed, and manipulated more easily. Sorting algorithms are used to arrange data in ascending or descending order according to predefined criteria. Sorting algorithms can be used to organize data in a variety of ways, including by size, alphabetically, numerically, or chronologically. Sorting algorithms can also be used to arrange data in a specific order, such as by priority or importance. Sorting algorithms are used in a variety of applications, including databases, search engines, data analysis, other job fields such as accounting, and more.

II. Objective

The objective of this sorting is to arrange data in a specific numerical order based on user input. Also, when we use the multiple threading option, we will have a faster sorting time, which means we will use more threads of our computer's CPU to accomplish this.

III. Algorithm

a. Merge Sort Algorithm

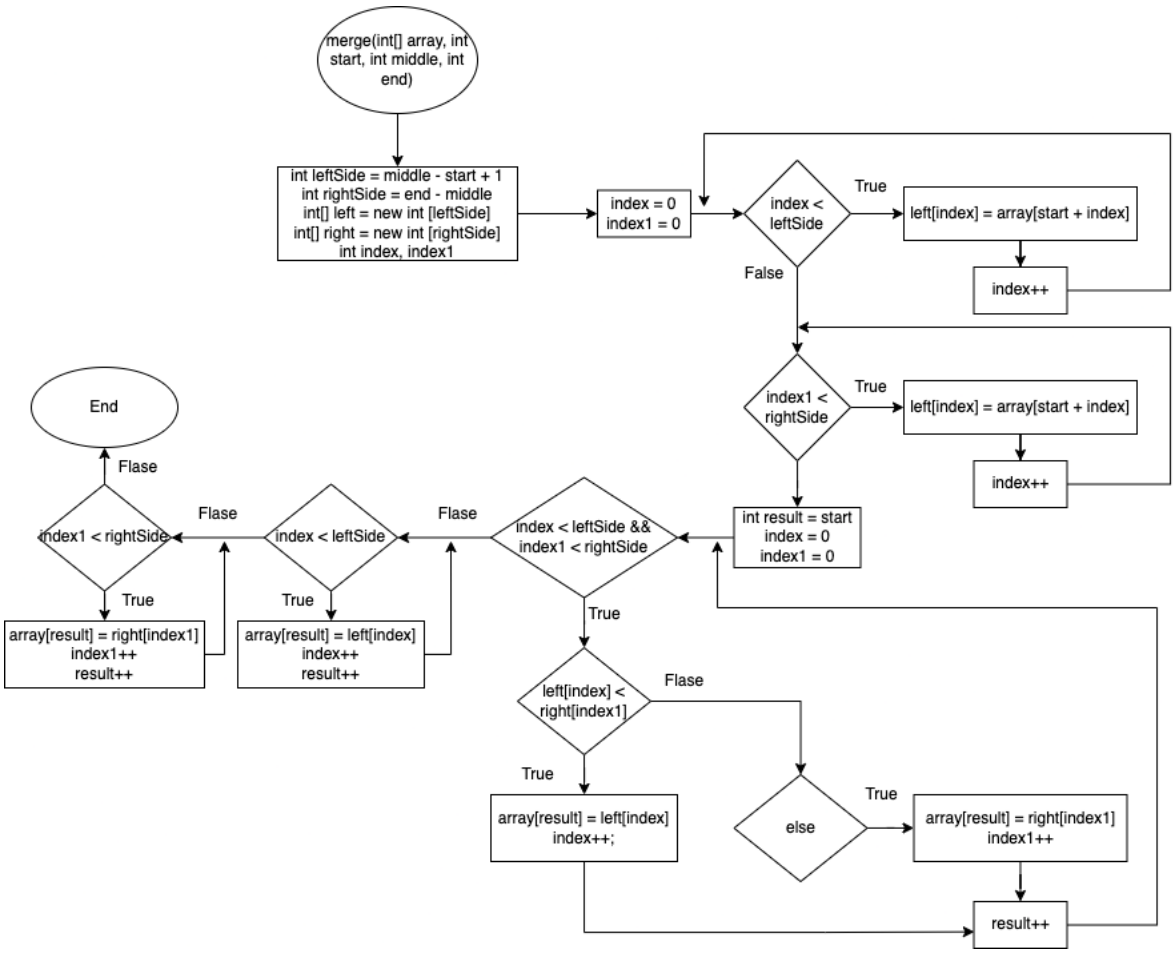
The "merge search" algorithm will be used in the source code. Merge sort is one of Java's most adaptable sorting algorithms. It sorts array elements using the divide-and-conquer strategy. It is also a stable sort, which means that it does not change the order of the original elements in an array in relation to each other. The underlying strategy divides the array into smaller segments until segments of only two (or one) elements are obtained. These segments' elements are now sorted, and the segments are merged to form larger segments. This process is repeated until the entire array has been sorted.

This algorithm has two main parts:

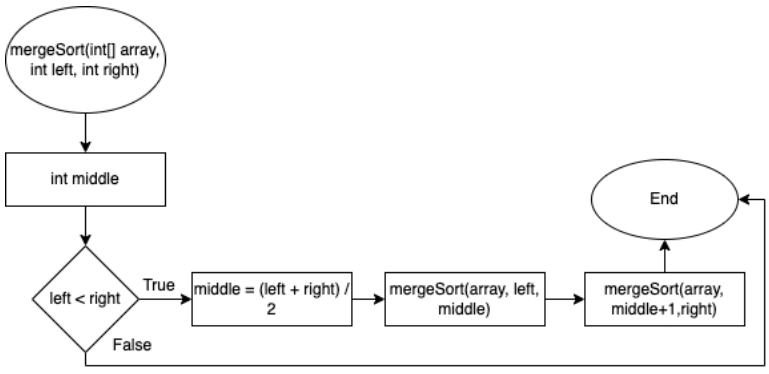
- mergSort(): This function computes the middle index for the subarray and then divides it in half. The first half runs from index left to middle, while the second half runs from index middle+1 to right. After partitioning, this function automatically calls the merge() function to sort the subarray being handled by the mergeSort() call.
- merg(): This function requires four parameters: the array, the starting index (left), the middle index (middle), and the ending index (right) (right). Once received, merge() function divides the subarray into two subarrays, one on the left and one on the right. The left subarray

runs from index left to middle, while the right subarray runs from index middle+1 to right. This function then compares and merges the two subarrays to produce the sorted subarray.

1. Flowchart for merg() Method



2. Algorithm for mergSort() Method



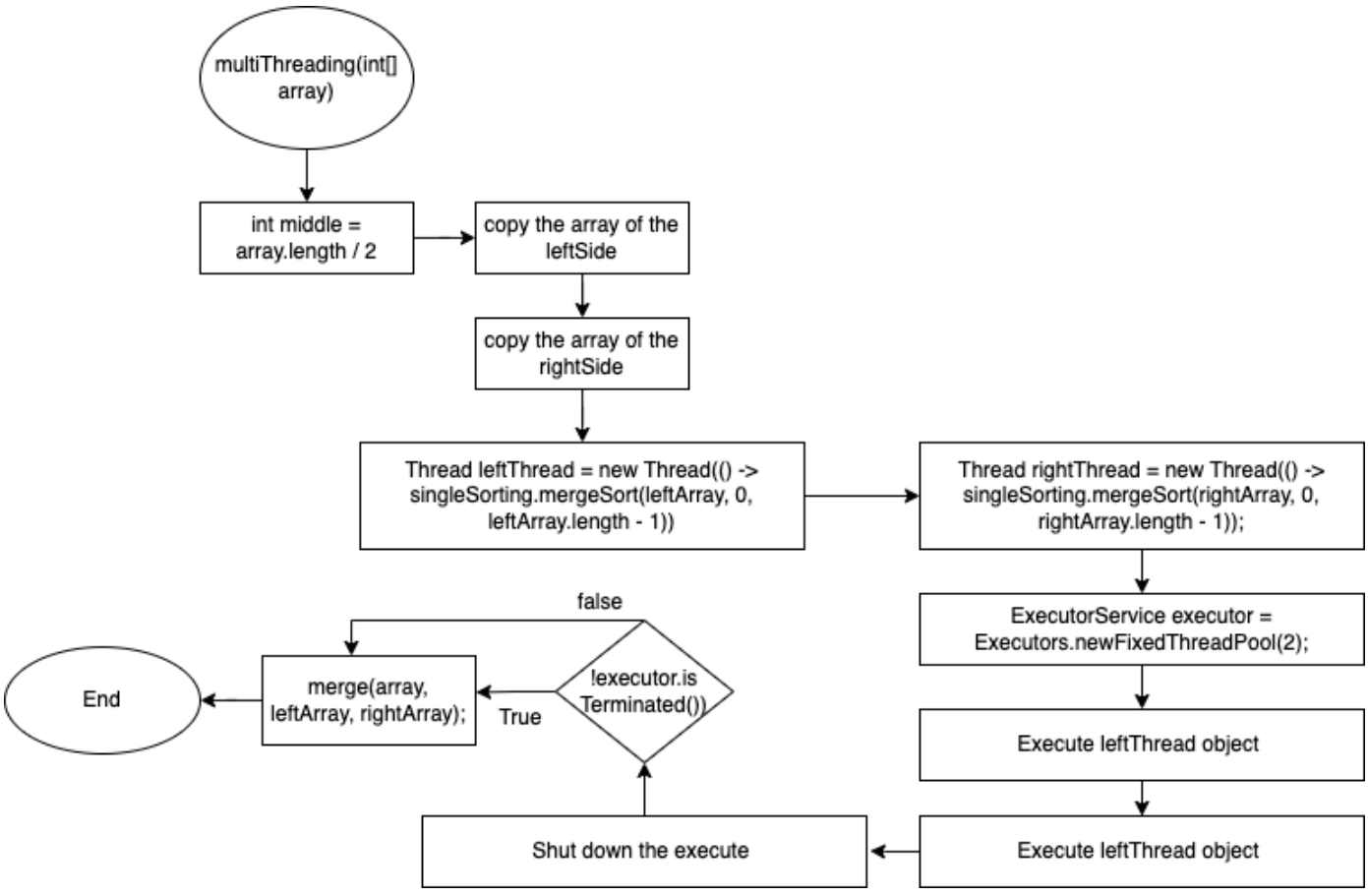
b. Single Threading Algorithm

As the algorithm predicted, I simply call the function to a method I've prepared and named singleThreading. The process will then run as a single thread for this single thread function.

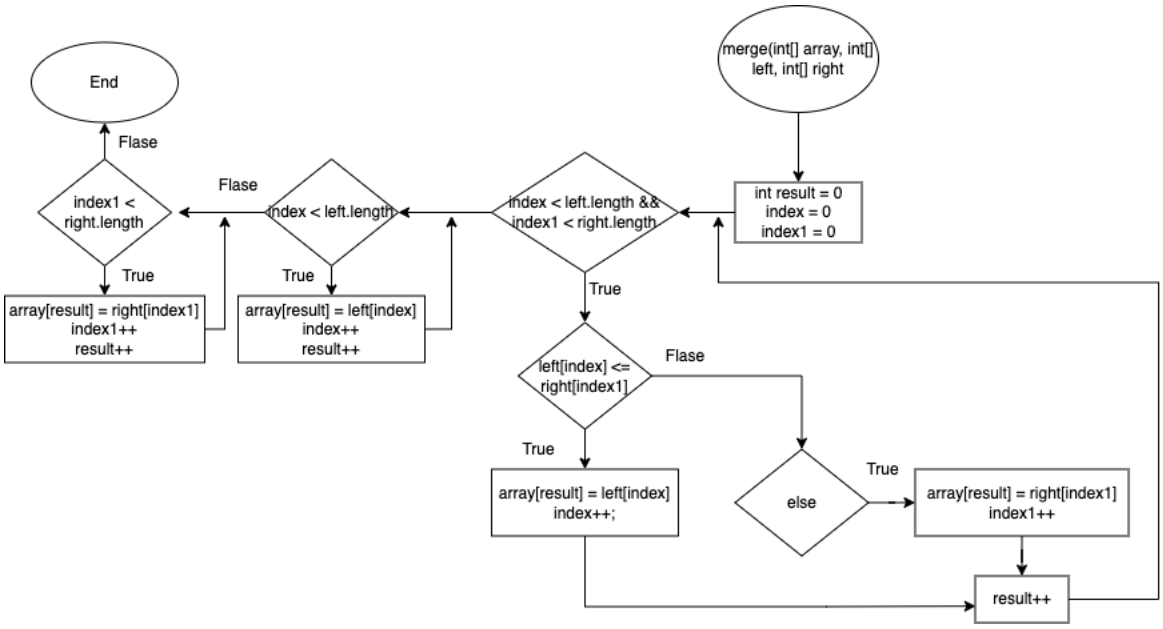
c. Multi-Threading Algorithm

This may be a bit complicated to compile, but for the algorithm that I have planned, I will divide the array into two subarrays, then create two thread objects, one for the left array and another for the right array, using the mergeSort method that I just demonstrated above. After that, I will use the ExecutorService, which is a Java library, to create a pool of two threads to execute the two subarray objects. After that, I will shutdown the executor. After that, I'll add a check to the executor in a loop to notify it that it has been terminated. At the very end, I will merge and sort the left side array and the right side array together to get a final merging sort.

1. Algorithm for multiThreading()



2. Algorithm for merge() method in multi thread



IV. Result

After I finish the code in this section, I will test it and record the results with the chart to make it easier to read.

M1 Pro Chip		Excercute Time in Milisecond	
		Single Thread	Multi-Thread
Amount of Array	10	0.023875 ms	0.013958 ms
	100	0.127917ms	0.042458 ms
	1000	0.25175 ms	0.175542 ms
	10000	2.457375 ms	1.393875 ms
	100000	20.004375 ms	12.688042 ms
	1000000	132.172708 ms	75.903875 ms
	10000000	1380.674708 ms	725.251875 ms
	100000000	15088.151291 ms	7749.078708 ms

V. Conclusion

The chart shows how strong the task is when done with multiple threads and how powerful it is when accomplished with a single thread. With this, I believe the user who frequently uses sort to sort data will be pleased to use sort when it works with multiple threads. Finally, I'm hoping that this will be useful to me one day when sorting through a large amount of data, allowing me to free up more time to work on something else.

VI. Reference

https://favtutor.com/blogs/sorting-algorithms-java#:~:text=Merge%20sort%20is%20one%20of,an%20array%20concerning%20each%20other.

https://drive.google.com/file/d/1NprOAv2-IEJy_McG7U4dHI3k8VJeT3cg/view?usp=sharing