SJSU CS-149-05 Operating System Final Project Report

Instructor: Jahan Ghofraniha

**Student Name: Chenyu Yang & Eric Van**

**Table Of Contents**

**1. Introduction:**..........................................................................................................................*2*

**2. Problem statement:**...............................................................................................................*3*

**3. Software Architecture:**.........................................................................................................*4*

**4. Implementation (GitHub Link):**.........................................................................................*5*

*4.a: The Basic Function Part (“MergeSort.java”and “QucikSort.java”)................................5*

*4.b:The RecursiveTask Part (“MergeSortTask.java”and “QucikSortTask.java”)....................8*

*4.c: Final project main part(“FinalProjectTest.java”)............................................................10*

**5. Test & results (this should include unit test and integrated test results):**......................*13*

*5.a: The Unit Test Part File (“MergeSortTest.java”and “QucikSortTest.java”).....................18*

**6. Name of each member who works on each section:**........................................................*19*

**7. Conclusions:**........................................................................................................................*20*

**8. References (the material should be fully referenced with links to the reference source code or content).**.....................................................................................................................*21*

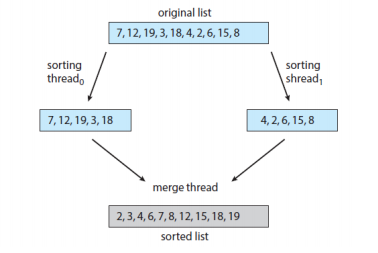
**9. Appendix:**...........................................................................................................................*22*

**(Hint: should include any extra information related to the project including the source code for your project.)**

*Our Final Project GitHub Link address:*[*https://github.com/ChenHCY/CS-149-Fianl-Project*](https://github.com/ChenHCY/CS-149-Fianl-Project)

**1. Introduction:**

This project is a multithreaded sorting program that can compare the used time of merge sort and quicksort when the level of the array list changes bigger and bigger. We create a list of random integers divided into two smaller lists of equal size and two separate threads. It can sort each sublist using a sorting algorithm such as quicksort and mergesort. It is then merged by a third thread which would be a merging thread. (like this graph shown).

****

We used Java’s fork-join parallelism to Implement the preceding project. The project will be created in two different versions. Each version will implement a different divide-and-conquer sorting algorithm. In this case, we used quicksort and mergesort. We created two files *“MergeSort.java”* and *“QucikSort.java”*  that can Implement this requirement.

From the basics learned, the quicksort implementation will be using the quick sort algorithm to sort the right and left arrays based on the position of the pivot. Merge sort algorithms will divide the list into two evenly sized arrays. When the list is under a certain threshold value, it should directly apply a simple algorithm. In this case, we are using bubble sort. A class is created that extends RecursiveAction. And Follow the project requirement, It's a non-result-bearing ForkJoinTask. *(“MergeSortTask.java” and “QucikSortTask.java”)*

In the main function test code, we used the for-loop to check the Merge sort and Quicksort output result correct or not. And in the Unite test and integration test, we create 2 files: *“MergeSortTest.java”* and *“QucikSortTest.java”* to do the Test case that when the level size of the array list changed bigger and bigger, compare the used time of merge sort and quicksort through the Junit Test on Java editor.

**2. Problem Statement:**

In terms of code projects, we generally use multithreading for two different purposes:

One is to subdivide the program into several modules with relatively independent functions to prevent one of the functional modules from blocking and causing the entire program to freeze, such as Java GUI programs.

The other is to improve operating efficiency, such as multiple cores running at the same time, or in a single core, when a thread performs IO operations, another thread can execute at the same time. Now, we will continue to introduce 2 hands reason for multithreading in the sort.

2a. The advance for MergeSort:

In fact, multithreading is very suitable for merging and sorting. Because merge sort is a typical application of recursive thinking. We can assume the quick sort algorithm is an improvement of the bubble sort algorithm. So it can run independently after splitting, and finally merge the results. For the sub-arrays that are split into, they are independent of each other, and resources are not shared, so it is very safe to use multiple threads for merging and sorting. [(1. the website link for we search used)](https://www.geeksforgeeks.org/quick-sort-using-multi-threading/)

2b. The advance for QuickSort:

When we use multithreaded in the quick sort, the multithreaded can allow concurrent execution of two or more parts of a program. It means we can be the maximum utilization of the CPU. Each part of these is called a thread. So, we can see the threads are lightweight processes within this project process.

So these reasons are why we need to use multithreaded programming in the merge sort and quicksort. It can save time, and much safe, and maximum used utilization of the CPU.

**3. Software Architecture**

****

-We created the project with a Main function called “FinalProjectTest”.

-In this project, we created four main classes which are the QuickSortTask, MergeSortTask, quickSort, and mergeSort. In this part, we used *“java.util.concurrent.RecursiveAction”* to do multith-readed it.

-We also created two other classes to test it with JUnit called QuickSortTest and MergeSortTest.

**4. Implementation (GitHub Link):**

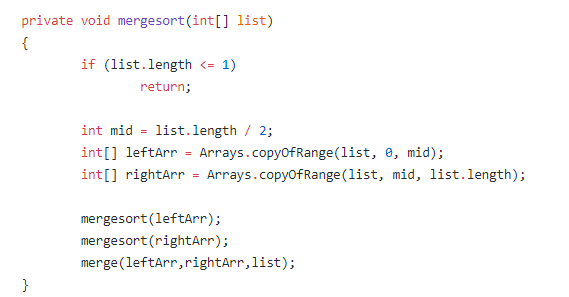
**Our Project GitHub Link address:** [*https://github.com/ChenHCY/CS-149-Fianl-Project*](https://github.com/ChenHCY/CS-149-Fianl-Project)

***4.a: The Basic Function Part*** *(“MergeSort.java”and “QucikSort.java”)*

4.a.1:  *MergeSort.java:*

In this file we created 2 function, one is merge function, the other one is mergesort function

Function: The mergesort function takes in the list of arrays and divides the list by two. It then creates two different arrays called leftArr and rightArr separates the array. It later calls the merge function.



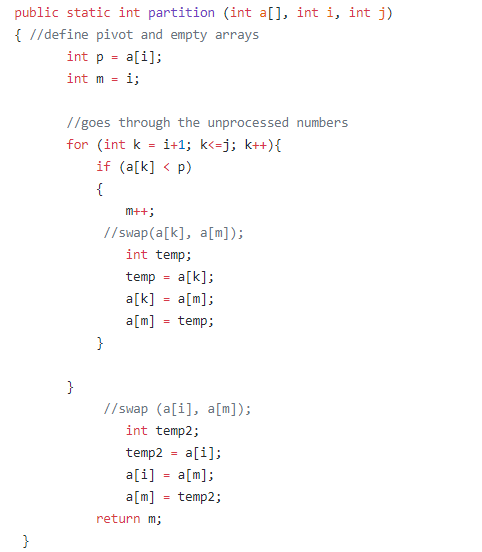
Function: The merge function takes in the array list and checks to see if an integer is larger than the next integer in the list. If so, it would swap it and move on. It compares the two arrays. We also used the do-while loop in there. It can support our logic and ideas to achieve it.



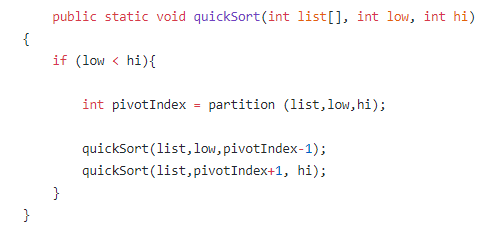
4.a.2 *QuickSort.java:*

In this file we created 2 function, one is quickSort function, the other one is partition function

In this file, the partition function makes an array and defines the pivot. It goes through the numbers to check if the next number is larger than the pivot. It puts the integers in a temp array and swaps when going through the unprocessed numbers.



The function quickSort takes in the list array [] sorts the list through quicksort when the condition is met.

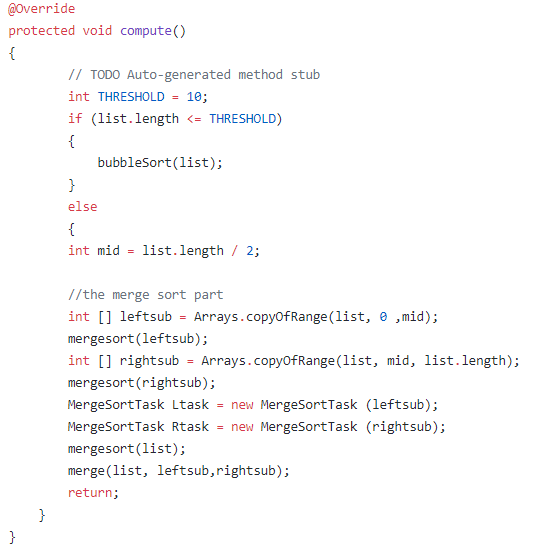


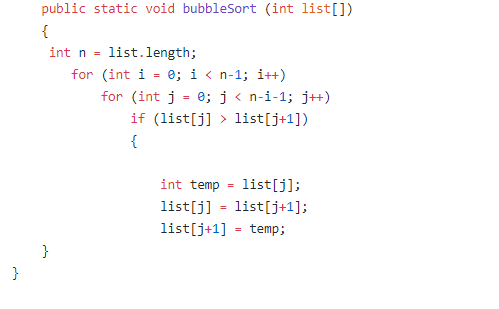
*4.b:The RecursiveTask Part (“MergeSortTask.java”and “QucikSortTask.java”)*

4.b.1:  *MergeSortTask.java:*

From our basic kown, ForkJoinPool is the implementation class of ExecutorService, so it is a special thread pool. ForkJoinPool provides the following two commonly used constructors. In this project, we use ForkJoinPool to start two threads to perform this comparison merge sort, and quicksort use time, and print requirements.

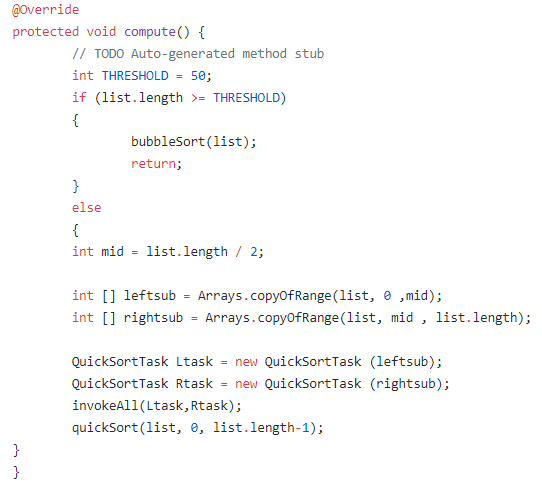
Here is where the multithreading is done for quicksort. We took the MergeSort.java code and pasted it into this file for the mergesort algorithm. MergeSortTask extends RecursiveAction. Under void compute() we have a threshold with an if and else statement. If the list is under a specific threshold, the list would get bubblesorted. Else, the list will be divided into two different arrays and mergesorted. And this class is created that extends RecursiveAction. And Follow the project requirement, It's a non-result-bearing ForkJoinTask.



**

4.b.2:  *QuickSortTask.java:*

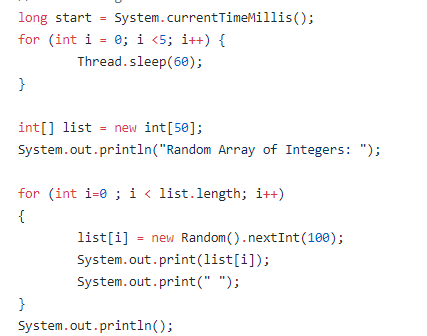
In this file, the quicksort implementation is done. It works pretty much exactly like the one above, MergeSortTask.java but instead of the mergesort step if the threshold is exceeded, it does a quicksort algorithm instead.



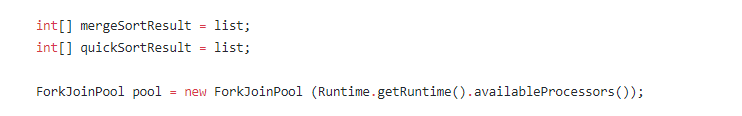
*4.c: Final project main part(“FinalProjectTest.java”)*

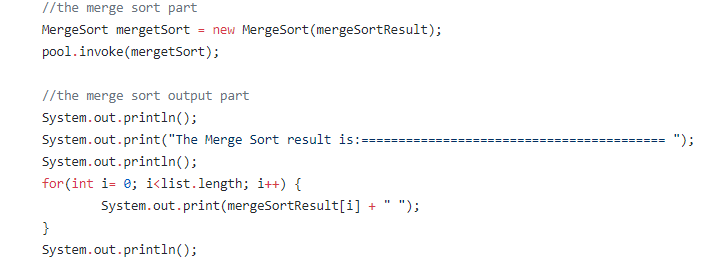
4.c *“FinalProjectTest.java”:*

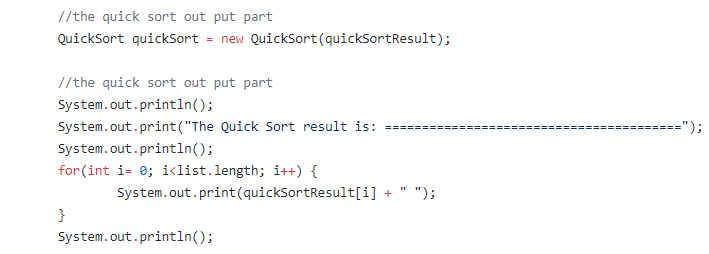
In this part, we created a *“FinalProjectTest.java”,*we used the for-loop to create a list that can have random numbers , and Definition long start = System.currentTimeMillis(); and also use for-loop to go through Thread.sleep(60); It can calculate the sort Execution Time and print out compile in the Step 5.



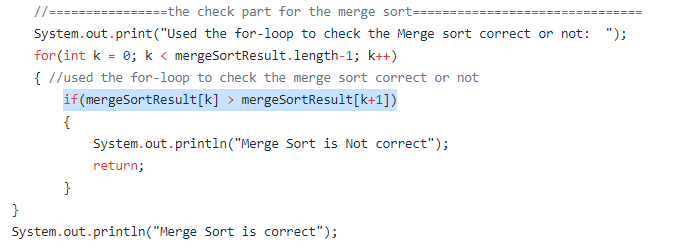
Second, we create the int[] mergeSortResult = list; and int[] quickSortResult = list; to move forward into the merge sort function part and quick sort function part that can get the **Two Sort Result**. And used ForkJoinPool pool for the RecursiveTask (RecursiveAction).

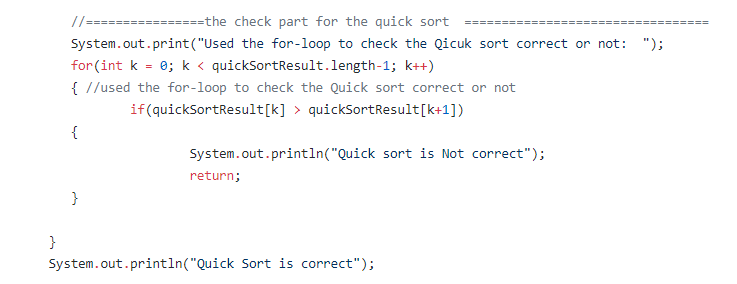






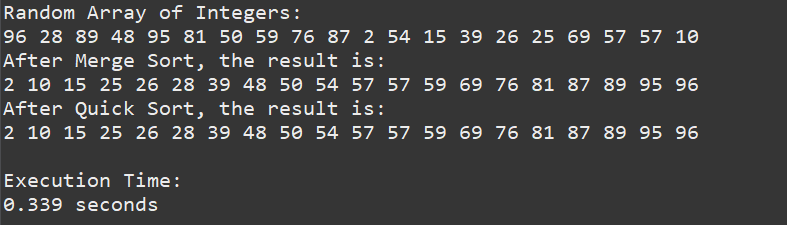
The third part is the Check part function by using For-Loop to check whether the result of Merge Sort and Quicksort is correct or not. We set up the for loop if-statement condition to be if(mergeSortResult[k] > mergeSortResult[k+1]), It means when the number is larger the next number, it will print out **“The result is Not correct”**.



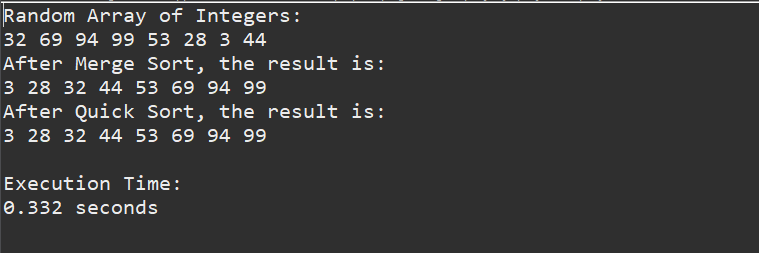


**5. Test & results**

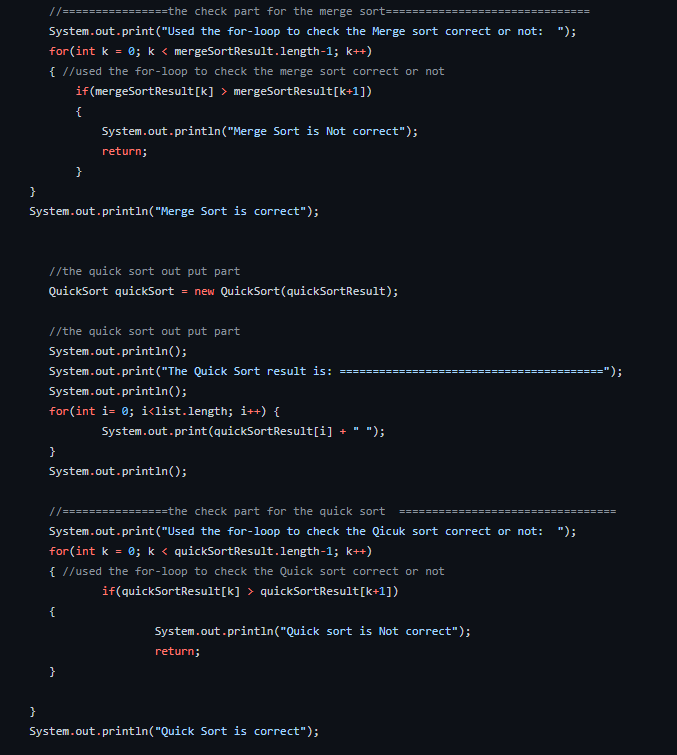
This is the result when the list array size is 20 with the threshold as only 10.

****

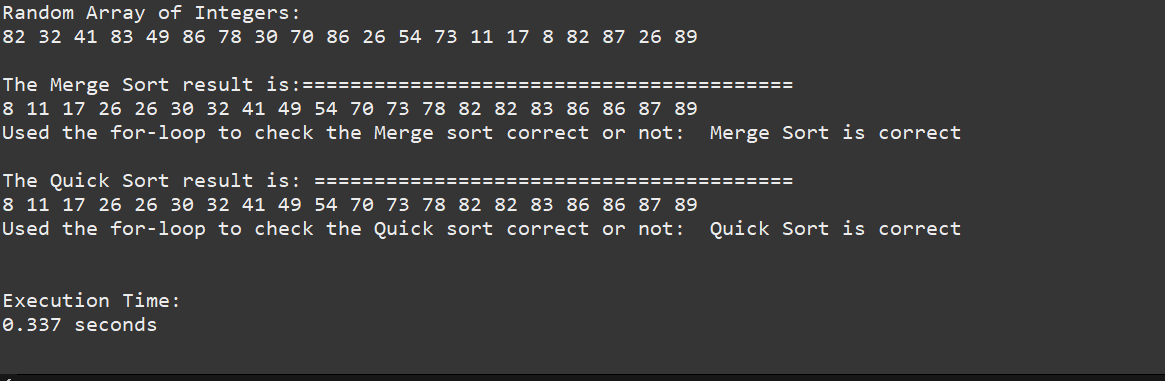
This is the result when the list array size is under the threshold of 10.



In the code below, we used a print statement to check if the sorting algorithm is really working through quicksort and mergesort.

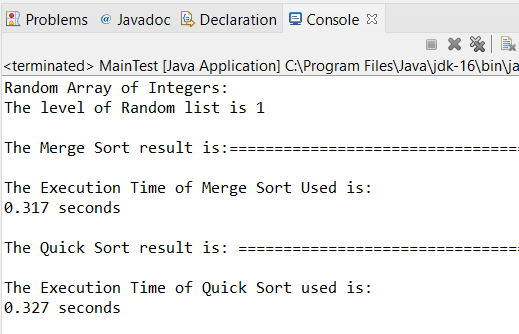


This is the result after running the program with list[12].

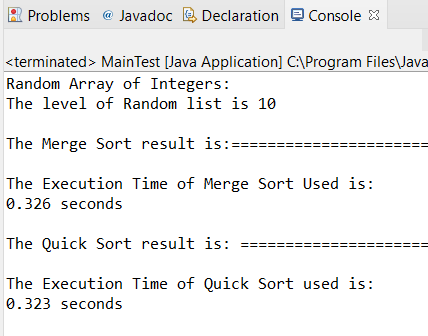


5.a: Compare the Execution time with different array size in the Merge sort and Quicksort:

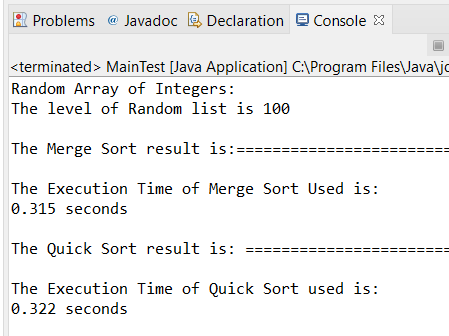
**List[1]**:



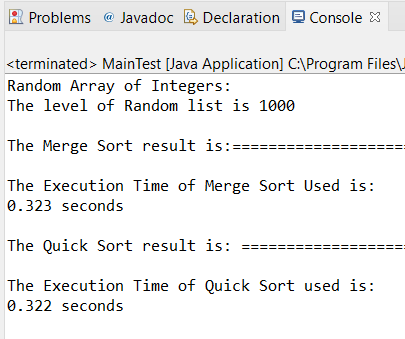
**List[10]:**

****

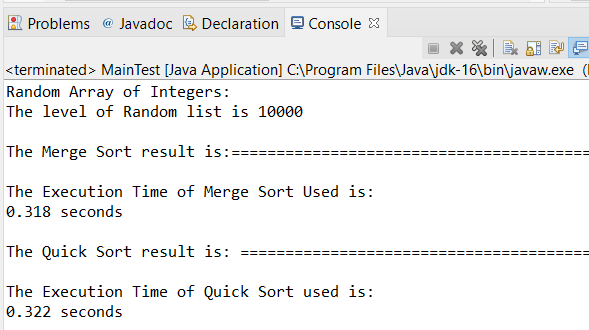
**List[100]:**

****

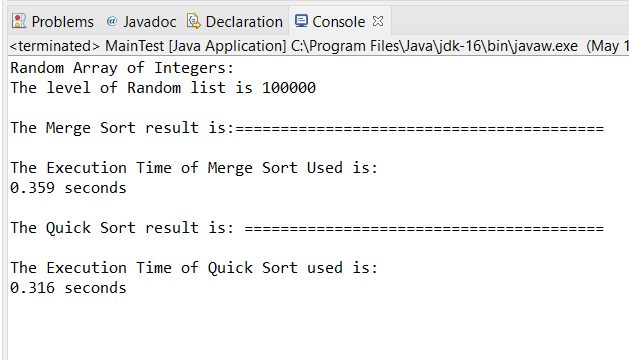
**List[1000]:**

****

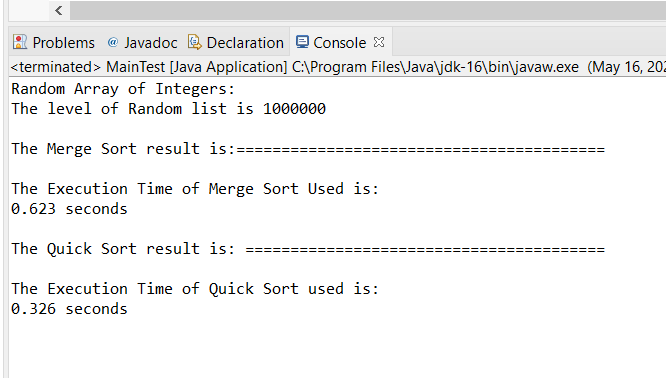
**list [10000]:**

****

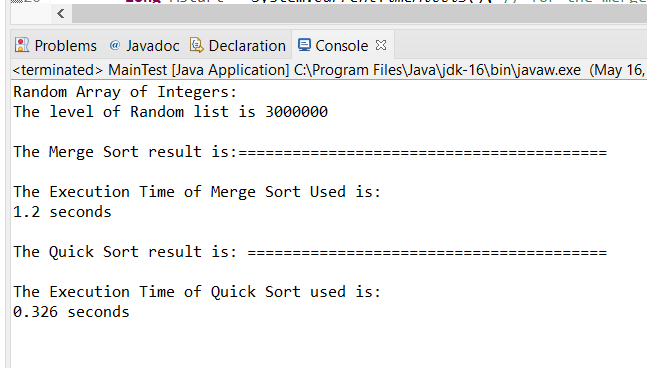
**list[100000]:**

****

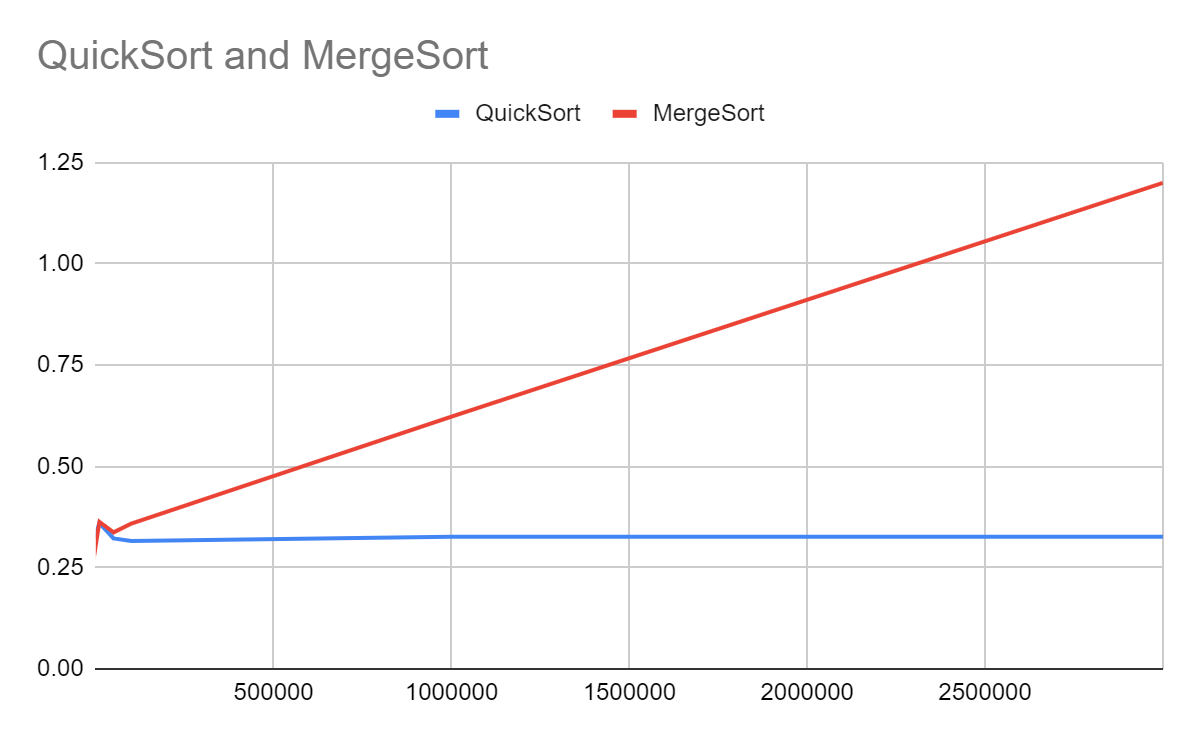
**List[1,000,000]:**

****

**List[3,000,000]:**

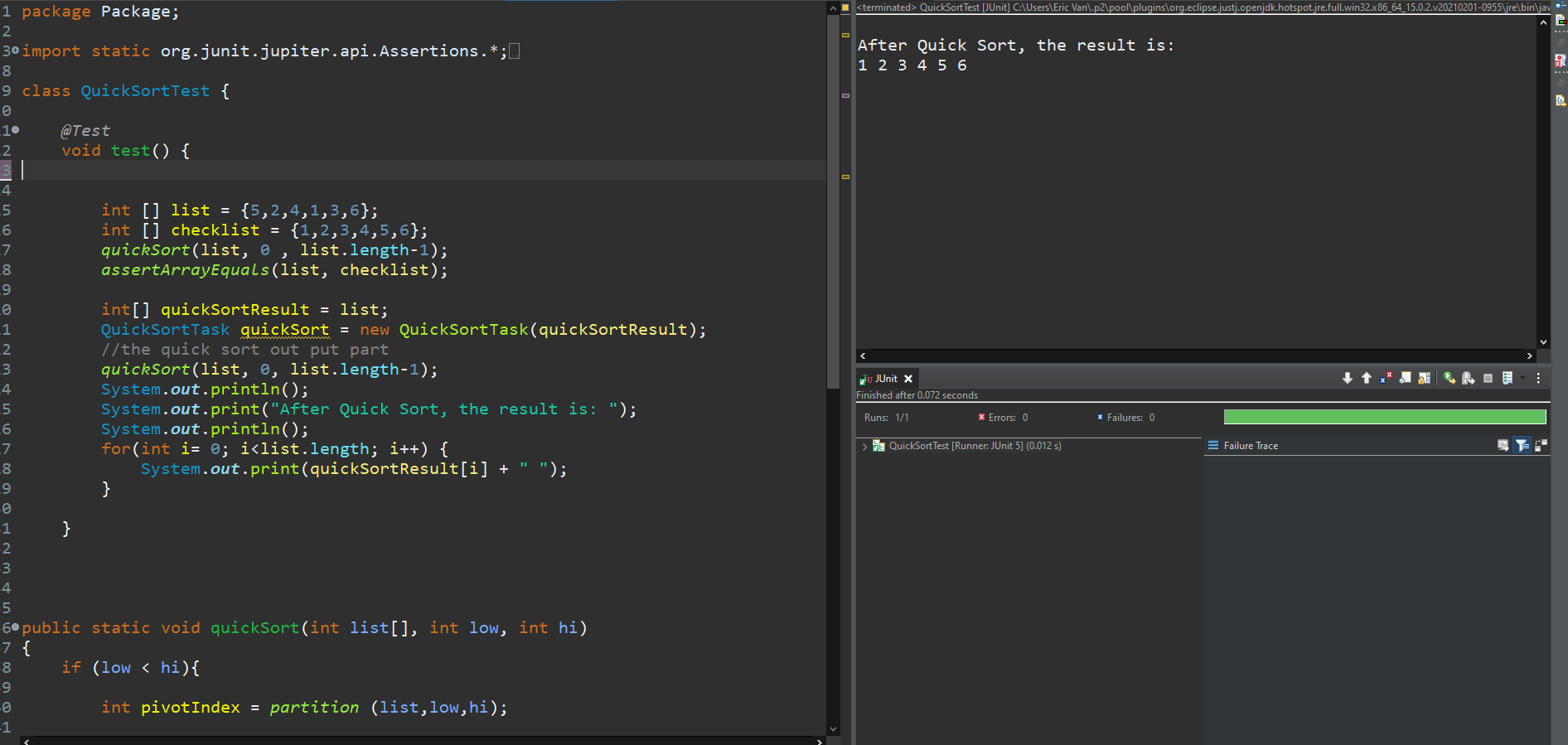
****

**And We did a table for through all of the our project output result:** All of the our project result can support followed the level of arraylist

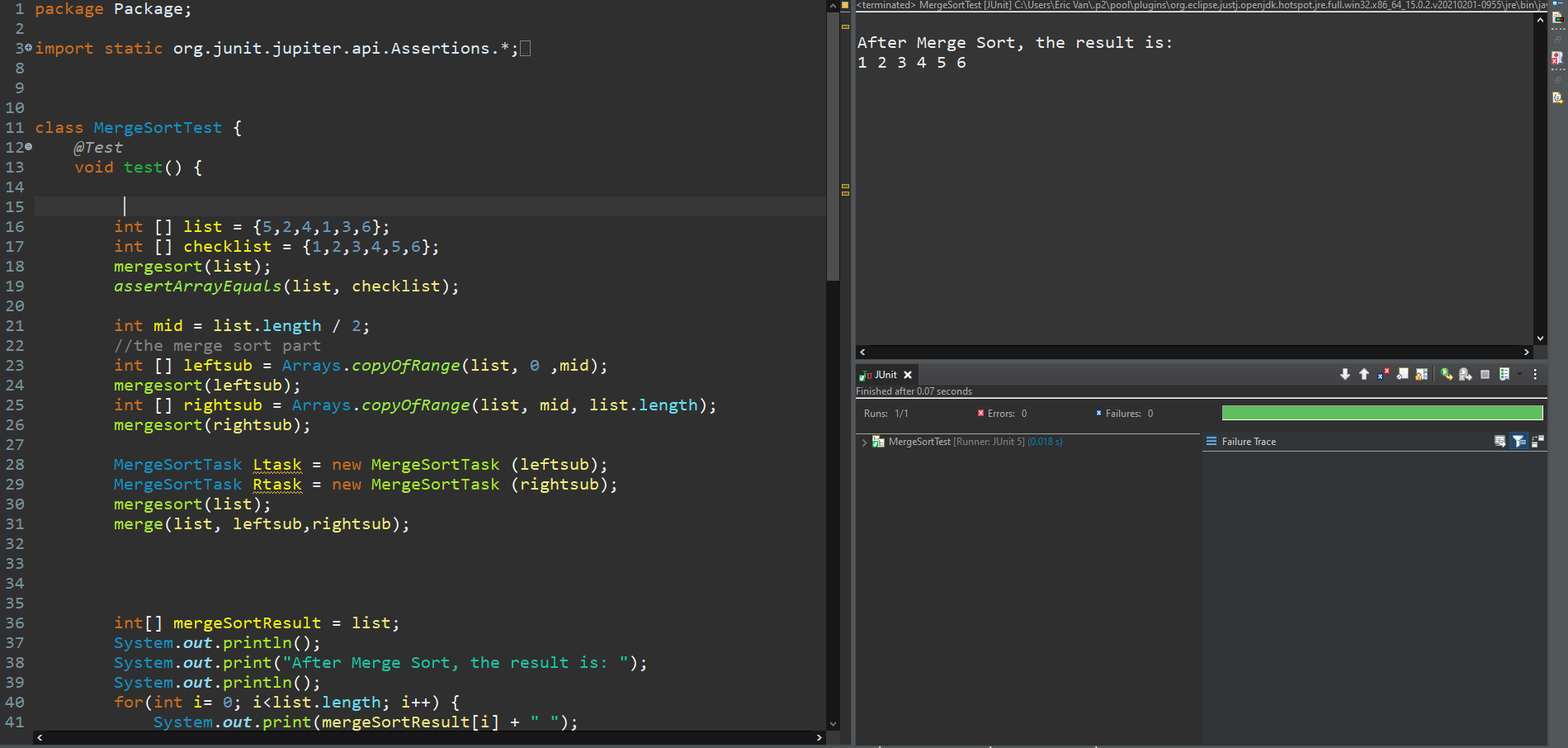


*5.a: The Unit Test Part File (“MergeSortTest.java”and “QucikSortTest.java”)*

5.a.1: JUnit testing for QuickSort: *(MergeSortTest.java)*



5.a.2: JUnit testing for MergeSort: *(QucikSortTest.java)*



//Run the for mulithread to see the time, the graph for the merge sort and quick sort

**try when the level of array list on 20000, 100000 and 200000**

**6. Name of each member who works on each section:**

Member 1. Eric Van:

1.QuickSort code

2.BubbleSort code

3.RecursiveAction

4. JUnit Test

Member2: Chenyu Yang:

1. MergeSort code

2.The Check part on the Main Test Function (Result)

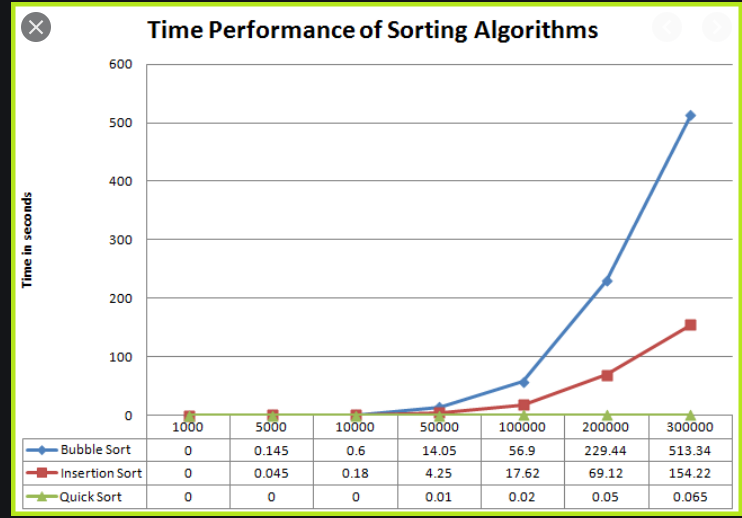
3. Time execution time code

4. The part of check the Execution time

Both:

We both did the report part together on each one through the google doc.

**7. Conclusion**



From this project, we have followed this graph to compare the time of merge sort or quicksort. And from this graph, we discussed and think that multithreading does affect the performance of the sorting algorithms. As the name suggests, quicksort is the winner in terms of speed compared to mergesort..

When we run the program, the levels of the array list are big, much bigger ...the time of sort list used will be much less than the time of merge sort list used. Like we said, "quicksort is the winner in terms of speed compared to mergesort".

From Our output result (We put our *Screenshot of the Output Result* on *Step 5 Test & Result*):

1.When the level of the array list was 20000, the merge sort started to use a little more time than before. It means the merge sort starts to be slow. But Quicksort did not change anything and always kept its fast speed on doing the sorting number.

2.when the level of the array list was 100000 and 200000, the amplitude of the merge sort used time becomes much bigger than before, which means that merge sort becomes much slower. But the quick sort still maintains a very fast speed.

3. From our project output result, we also get to follow the levels of the array list to be big, the amplitude of the merge sort used time also will become much bigger and much bigger.

Finally, we can summarize all of them. Our output result from the final project can prove this graph and our assumption was correct.

And the most important thing we learned in this Final Project is we know that following the array list be bigger, the time used of quicksort was linearly changed, but the time used of merge sort was not linearly changed. As the level of the array list becomes larger and larger, the amplitude of the time that Merge sort used will become larger.

**8. References (the link address for what we used in the report) :**

*1.(Used multi-thread for the Quick Sort):* [*https://www.geeksforgeeks.org/quick-sort-using-multi-threading/*](https://www.geeksforgeeks.org/quick-sort-using-multi-threading/)

*2.(The Java Fork Join Pool):*

[*https://java-8-tips.readthedocs.io/en/stable/forkjoin.html#:~:text=ForkJoinPool%20creation,be%20used%20by%20the%20pool*](https://java-8-tips.readthedocs.io/en/stable/forkjoin.html#:~:text=ForkJoinPool%20creation,be%20used%20by%20the%20pool)

*3.(JAVA Fork Join pool):*

[*https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html*](https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html)

*4. (RecursiveAction class in JAVA):* [*https://www.geeksforgeeks.org/java-util-concurrent-recursiveaction-class-in-java-with-examples/*](https://www.geeksforgeeks.org/java-util-concurrent-recursiveaction-class-in-java-with-examples/)

***9. Appendix***

**Our Final Project GitHub Link address:** [*https://github.com/ChenHCY/CS-149-Fianl-Project*](https://github.com/ChenHCY/CS-149-Fianl-Project)