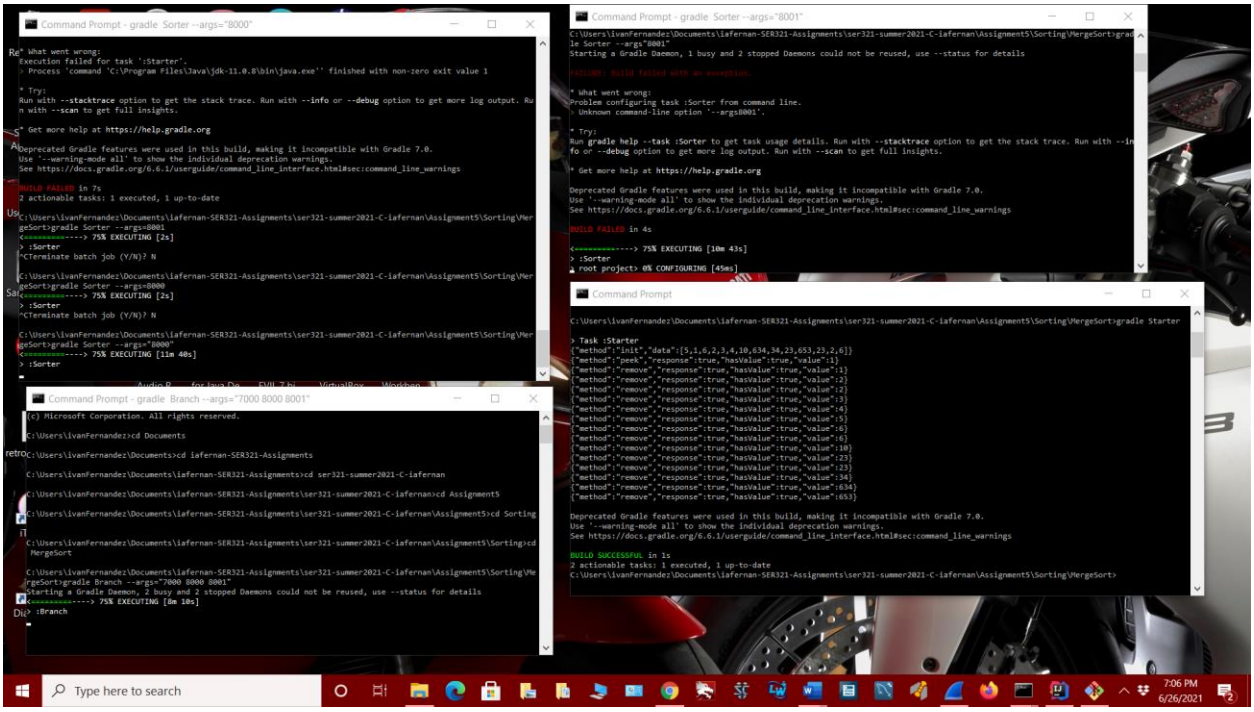
Ivan Fernandez

SER 321 – Summer C

6/25/2021

1 Activity: Distributed Algorithm: Sorting

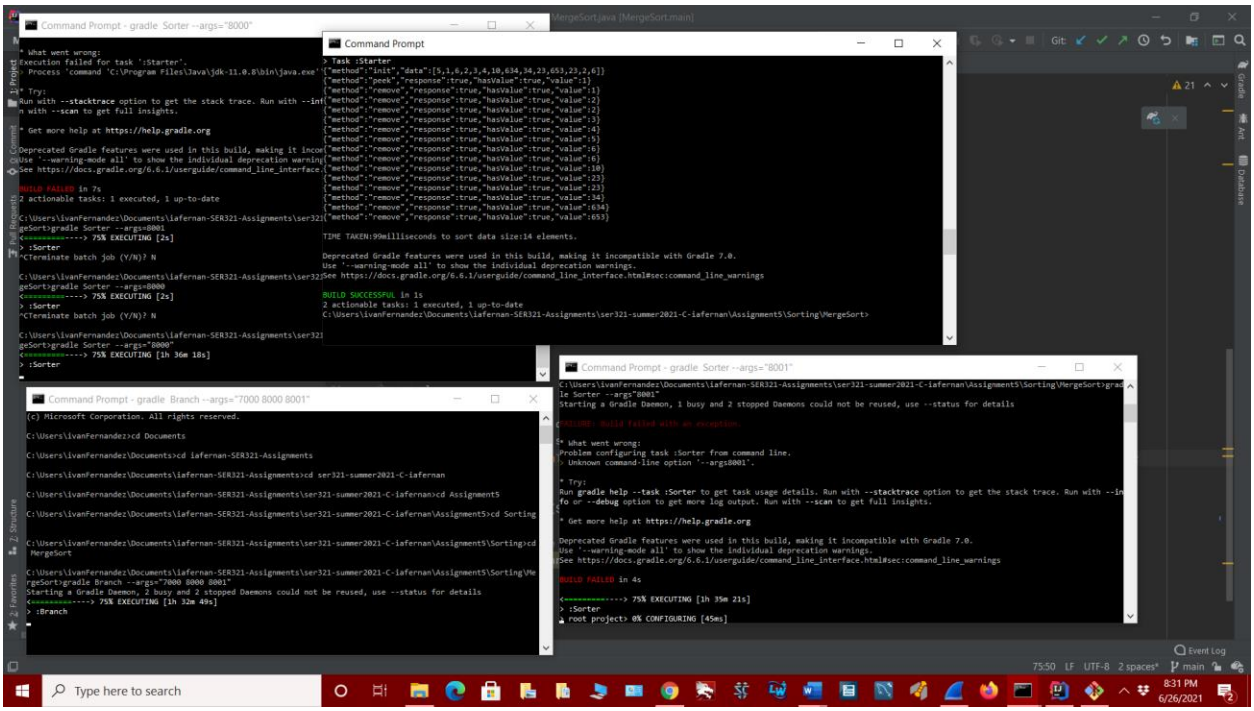
Task 1: Getting started



1. Explain the main structure of this code and the advantages and disadvantages of the setup of the distributed algorithm.

The code is built and structured to represent nodes (sorters) and branches that are used for the parallel distributed merge sort algorithm. The code is running by first creating the nodes, the lower parts of the hierarchy, then the branch or branches that are structured above the nodes that will distribute the array to be sorted to the nodes. The nodes are representative of separate servers on distinct ports that sort through one half of the array and each return the smallest value contained within their array. The branch then takes the smallest value from each node and begins to create a new resultant array with sorted values from smallest to greatest.

2. Run the code with different arrays to sort (different sizes, numbers etc.) and include code to measure the time (you can just enter start and end times). In your Readme describe your experiments and your analyzes of them. E.g. why is the result as it is? Does the distribution help? Why, why not? See this as setting up your own experiment and give me a good description and evaluation.



Test #	# of Branches	# of Sorters	Array Size	Time to sort in milliseconds
1	1	2	14	99
2	1	2	28	229
3	2	2 and 1 (tree)	14	191
4	2	2 and 1 (tree)	28	246
5	1	2	50	379
6	1	2	100	584
7	2	2 and 1 (tree)	50	384
8	2	2 and 1 (tree)	100	590

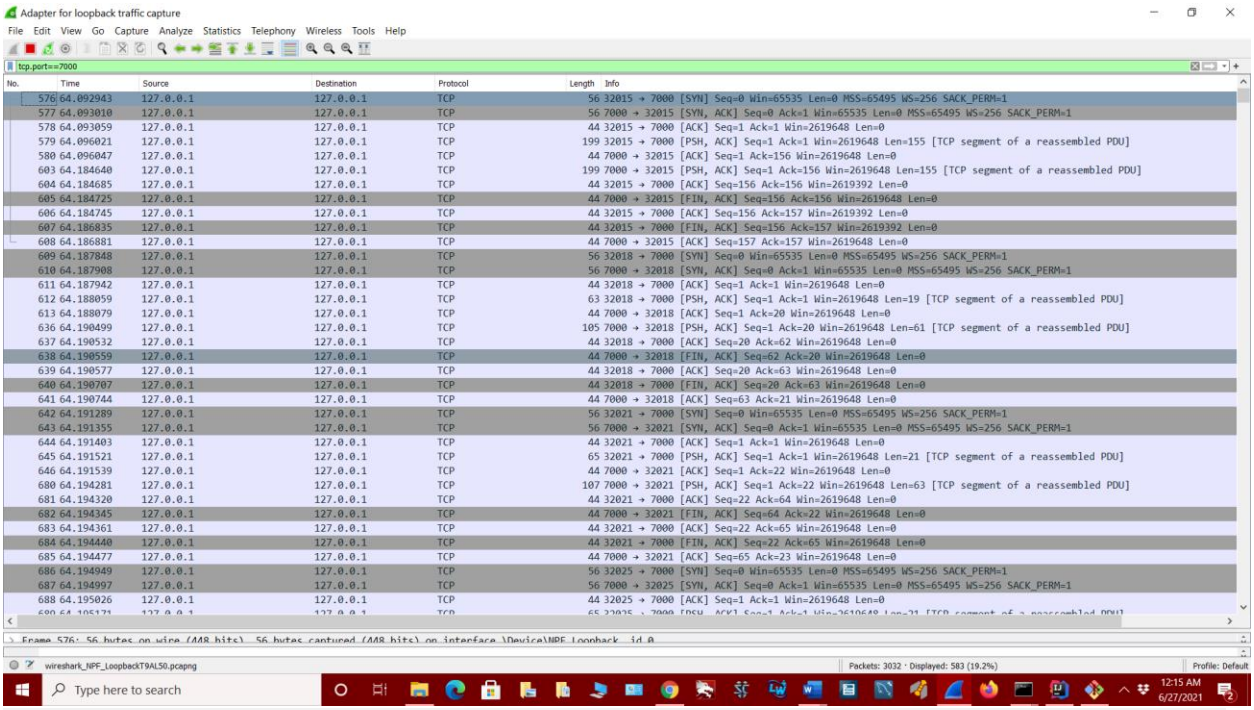
This experiment analyzed the sorting of varying sized arrays and nodes. When adding an additional branch and node, this branch was connected to an existing branch with connections to two nodes and their respective ports, thus creating a tree-like structure with two branches and three nodes total. When examining the results, there is an upward trend, as expected, of the time it takes to sort the array as the elements of the array increases. The time complexity of Merge Sort is O(n log n), which is a very efficient algorithm running on one system on its own. In conclusion, the difference in run time in a single system vs a distributed system for Merge Sort is negligible when comes to sorting small quantities of data. There would be seen greater differences when working with very large sets of data.

3. Experiment with the "tree" setup, what happens with more or less nodes when sorting the same array and different arrays? When does the distribution make things better? Does it ever make things faster? As in the previous step experiment and describe your experiment and your results in detail.

When sorting the same array using more nodes, there can be seen initially a more significant difference in time when dealing with smaller array sizes. The larger the array gets, there is a smaller difference in time when it comes to sorting the array. There seems to be no benefit to adding additional nodes when dealing with a small sized array, if anything it makes sorting take more time. When looking at the 100-element array, there is only a small difference in sorting time with two nodes vs. three. Again, a trend can

be seen and shows that adding additional nodes would be most suited for dealing with large amounts of data, otherwise there seems to be no benefit to it.

4. Explain the traffic that you see on Wireshark. How much traffic is generated with this setup, and do you see a way to reduce it?



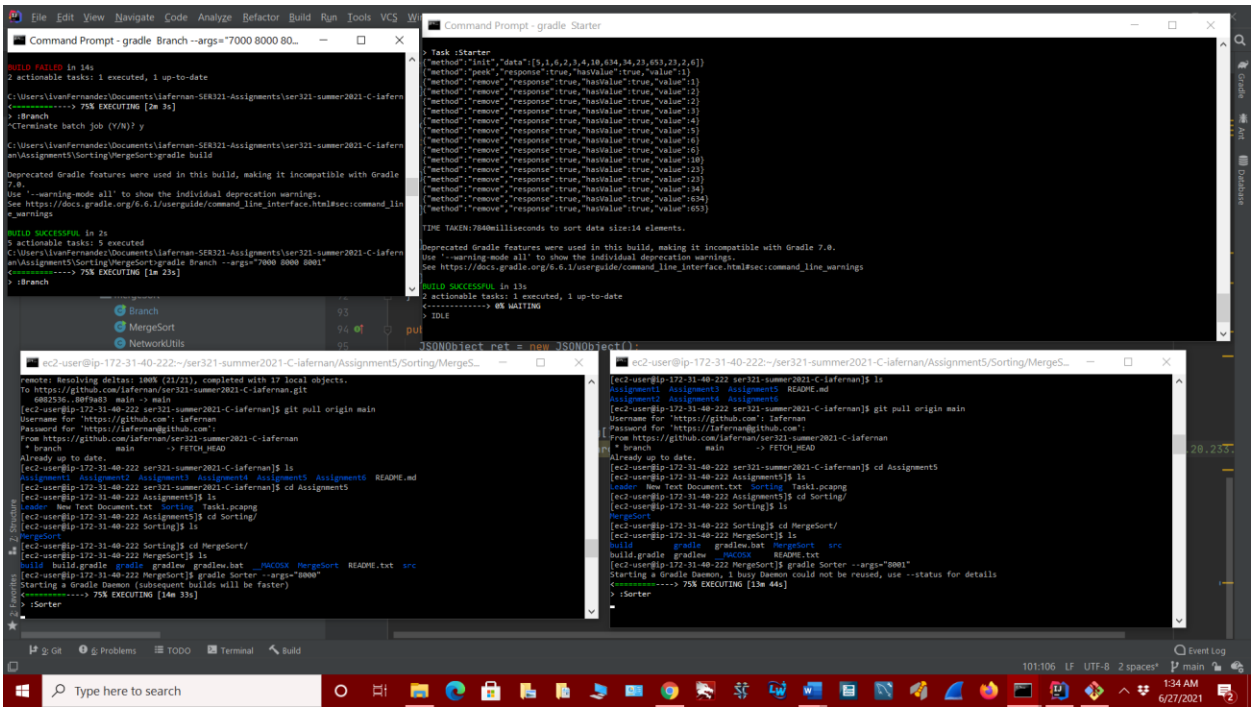
Tcp port 7000 is flooded with traffic once the program is ran with gradle Starter. I noticed that the larger the array size or amount of data, the more traffic there was on wireshark. The larger the data set, the more traffic can be seen between the branch and the nodes that it is communicating with, as the array is being split and sent to each node, then as the nodes send data back to the main branch.

Task 2: Running it outside of localhost

1. Do you expect changes in runtimes? Why, why not?

It is my prediction that runtimes may be different especially when dealing with smaller data sets, and that it will take longer to sort the data in the arrays. Since communications are occurring across to a remote network and not locally, there are more variables to consider that may increase the time in sorting the data sets. One such factor to consider may be how busy the remote server is at the time of communications.

2. Do you see a difference how long it takes to sort the arrays? Explain the differences (or why there are not differences)



I executed the program with an array of 14 elements, a local branch, and two remote AWS nodes. The amount of time in executing the distributed Merge Sort algorithm across the AWS network is considerably longer, in comparison to when it is executed locally. The amount of time that it took was 7840 milliseconds, when it only took 99 milliseconds locally.

Task 3: How to improve

1. Where is the most time lost, and could you make this more efficient?

The run time of the program is dependent on the speed of the nodes in the network. I believe that speed is a factor that varies from node to node, dependent on the hardware of the node, how many nodes and branches are involved, the amount of traffic handled by the host, how large the data set is, whether the node fails or not, and the speed of the network. Efficiency will be gained by testing out algorithms with variations of factors and determining whether it would be worth it to distribute it, like it has been done for this exercise.

2. Does it make sense to run the algorithm as a distributed algorithm? Why or why not?

I believe that this is strongly dependent on the efficiency of the algorithm itself. When looking at merge sort, it makes sense to distribute the algorithm when dealing with larger sets of data. This makes sense because of the time complexity of Merge Sort, it being linearithmic. As mentioned in the lectures, distributed programming is strongly suited for algorithms that are quadratic.