Ching-Yuan Chang u0914005

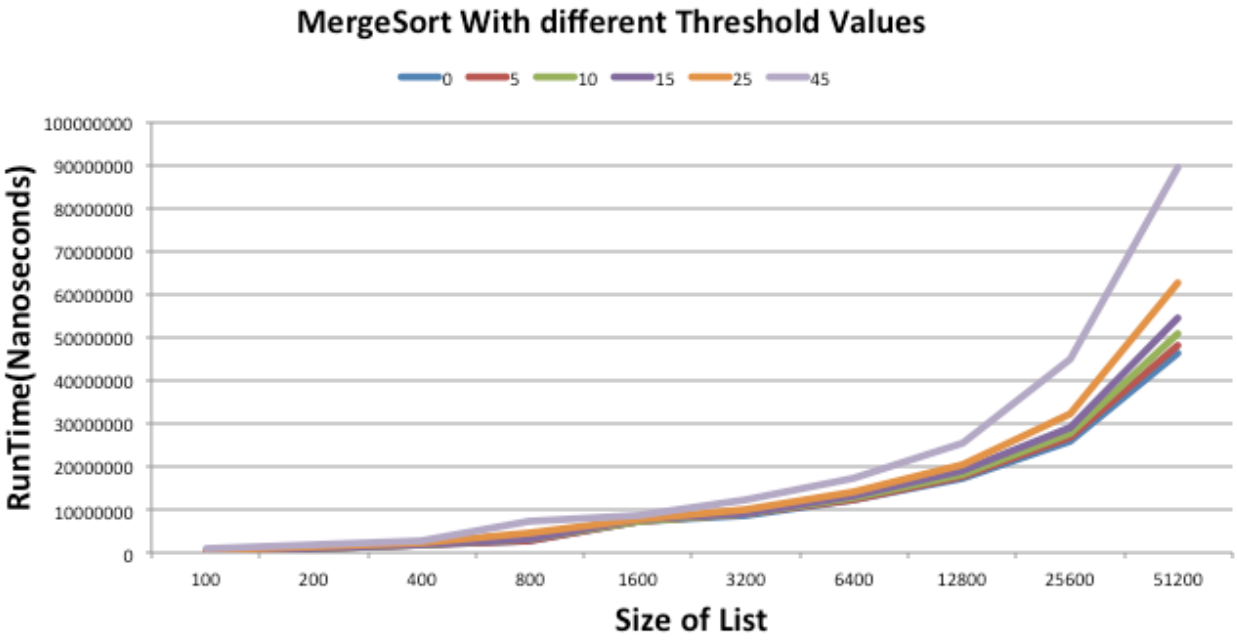
Assignment 05 Analysis Document

1. My partner is Min Kim and I will be the one submitting the source code.

2. Min is a great partner, she knows what she is talking about, and she have better ideas on choosing the pivot point then I do. I do plan on working with her again.

3. I think the great part about having a partner is that when I am not sure about something, I can just ask my partner about it. And when I have a question or our understanding of the question is different, we can just discuss it and find the right answer. The cons of the pair programming is that we need to find time that works for both of us, so that we can sit down and do the project together, and it is really hard to meet up together during ta hours.

4.

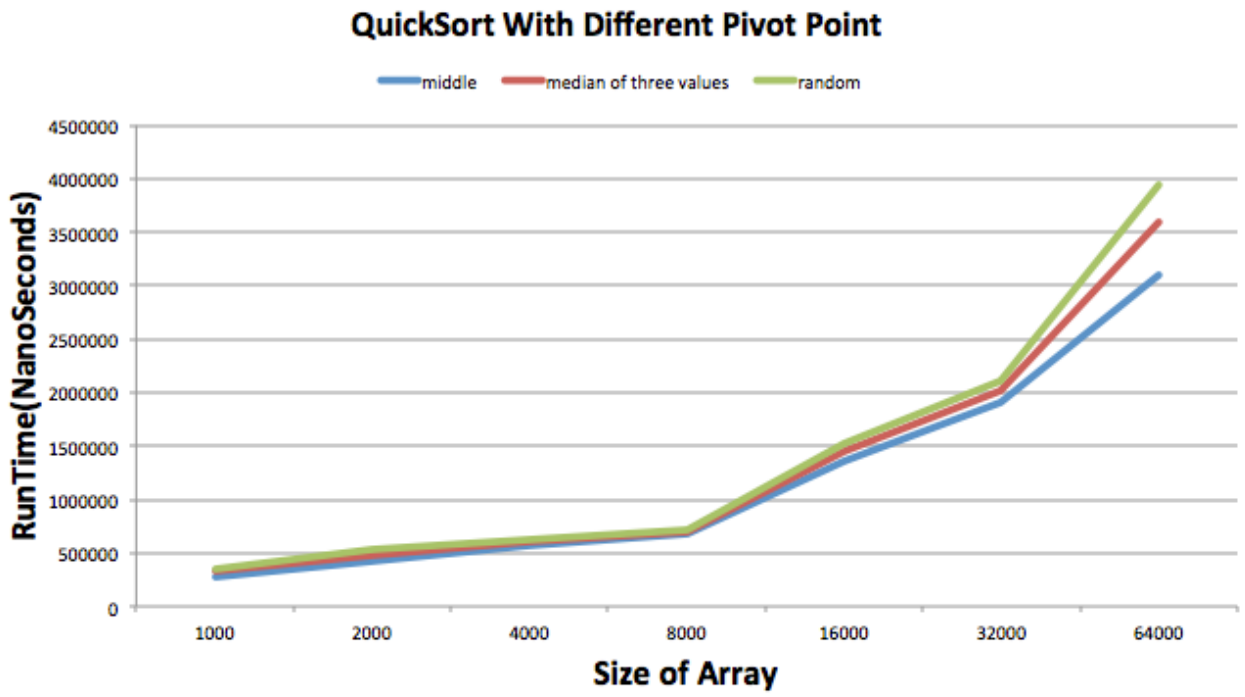


For this experiment we set the threshold to 0, 5, 10, 15, 25, and 45. We use our getAverageArray method to create a array of size from 100 and doubles it until 51200. we made sure that we are using the same array for all the different thresholds by making an separate array and copy the original array to it every time the threshold changes. As you can see from our result, the best threshold value for our merge sort is 0 because as you can see form the graph above, threshold value 45 takes the longest time to run, and when the value is 0(which is the blue line), it takes the shortest time to run which means that our merge sort runs the fastest when it doesn't even switch to insertions sort.

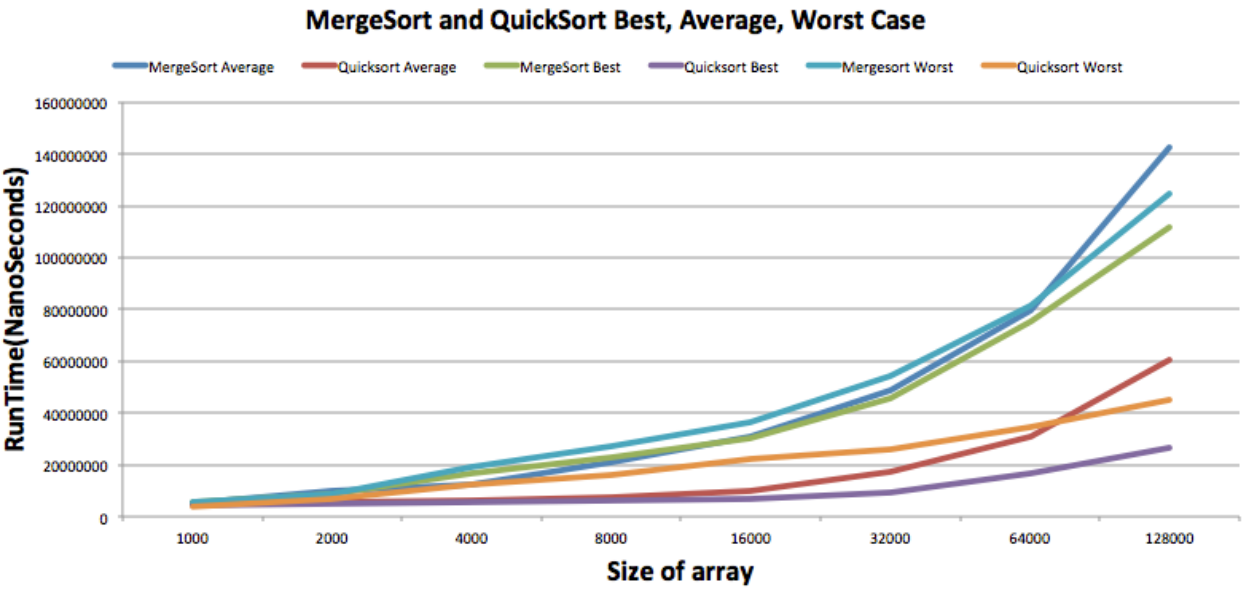
5.

for this experiment, we have three cases that can generate different set of pivots, we made a method that will always make the pivot the middle index of the array that needs to be compared. Another case is that we will find the median of three values and then swap the item to the middle so we can use that item as our pivot, lastly we have a random number generator that generates a random number from 0 to size-1 every time we generate our pivot point. For our arrays, we generate an array of the average case and then we have an temp array that we will use to sort, after every sort we empty the array and then add the already generated array into the temp array again and sort it with a different pivot generate method. For the difference size of array, we started from 1000 and doubles it until the size reaches 6400. As you can see from the graph below, the pivot generate method that generates the best pivot point is the one

that always get the pivot at the middle. These pivot points didn't really make a huge difference in running time but the at the end when the array size is 6400, the difference started to show.



6.



For this experiment we run the merge sort and quick sort with an array list that is in order, random order, and in reverse order. Again we have a temp array for our random order list so every time the array that is going to be sorted will be the same. This time we also start the array size from 1000 and doubles it until it goes up to 128000. as you can see from the graph above, the big-o complexity for the best case for quick sort is O(n logn) because at the largest array size the time is slightly going up and if the graph keeps going, then I think the complexity will be O(n log n) this is the same for all the cases in quick sort and the best and worst case for merge sort. For the average case of merge sort, I think the big-o complexity is O(n^2) because

the run time at the largest list size it is starting to curve up more then the best and worst case. So I think as the list gets bigger, the runtime will increase more and more and curve up to create a O(n^2) looking curve. As you can see from the graph above, all of the quick sort curve are below the merge sort curves, which means that quick sort runs faster then merge sort.

7. The actual running time did not exhibit the growth rate that I was expected, because the run time of quick sort should be O(n log n) for best and average, and for worst case we are expecting O(n^2) but our method did a lot better then we expected and give us the run time of O(n log n), which is better the run time of our quick sort method for worst case. For merge sort, we were expecting to see O(n log n) for all the case, but only the worst and the best case turn out to be O(n log n), and our average case turned out to be O(n^2).
8. we approximately spend 10 to 15 hours on this assignment.