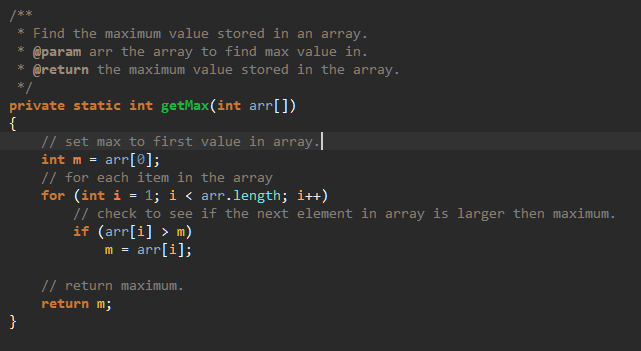
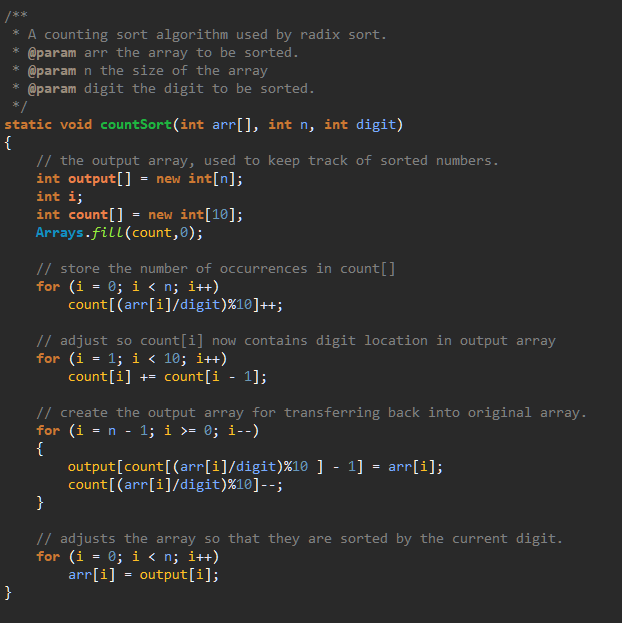
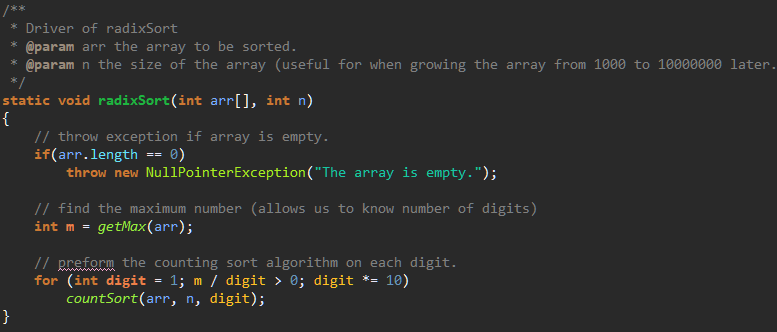
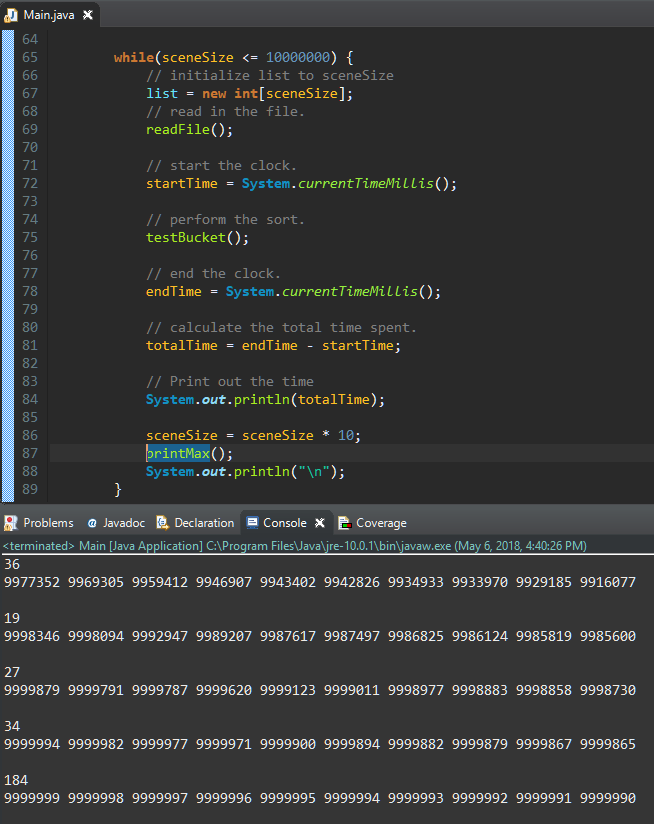
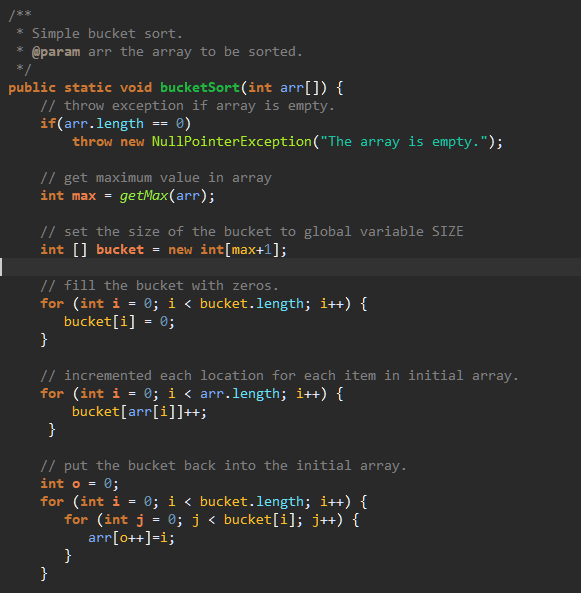
# Lab 2

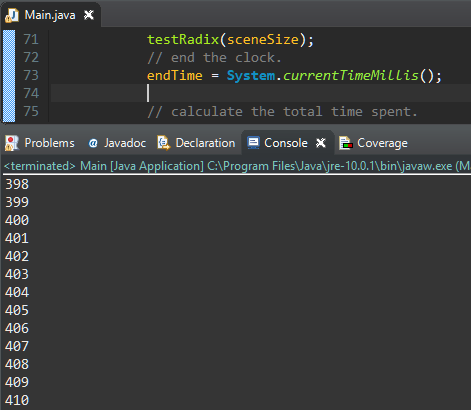
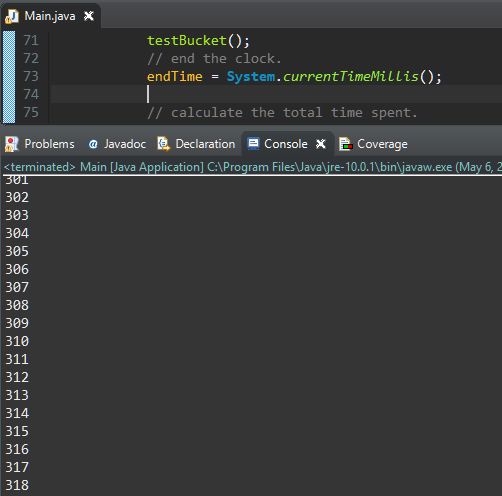
**Implement the Radix sort algorithm and use it to sort roughly 10,000,000 numbers.**

The book was pretty thin on sudocode on this one and I had a hard time figuring out how to get the digits to work and to actually use the counting sort to produce a sorted list in the end. Mainly just used [geekforgeek](https://www.geeksforgeeks.org/radix-sort/) as my reference on this one. I also place the getMax() method here since I use it here, but I also use it in the bucket sort as well. 

**Implement the Bin sort algorithm and use it to sort roughly 10,000,000 numbers.**

Struggled with getting this to work, I did not want to have to use the SIZE global variable which was the only way I could get it to work until I made a get max loop to find the largest item stored in the sorted array. I most likely added some O time by doing this… used a combination of the book, along with [sanfoundy.com](https://www.sanfoundry.com/java-program-implement-bucket-sort/), [javacodex.com](https://www.javacodex.com/Sorting/Bucket-Sort), and [growingwiththeweb.com](http://www.growingwiththeweb.com/2015/06/bucket-sort.html). For proof that it works see next question.

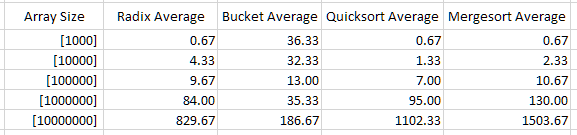
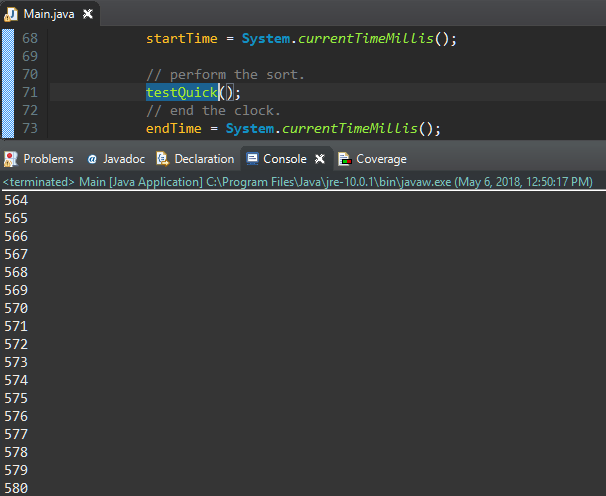
**Make sure the results are actually sorted for 1 and 2. Show the screen dump indicate the sorting algorithms are actually sorting correctly.**



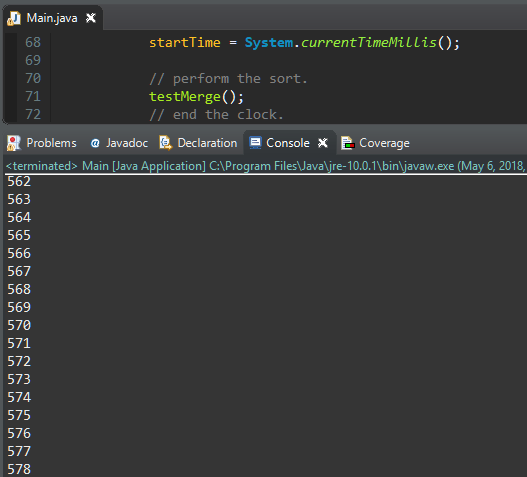
Proof that Bucket Sort works correctly.

Proof that Radix Sort works correctly.

**Show the execution time comparison with your either quick sort or merge sort. Also make sure the result of your quick sort or merge sort is actually sorted.**

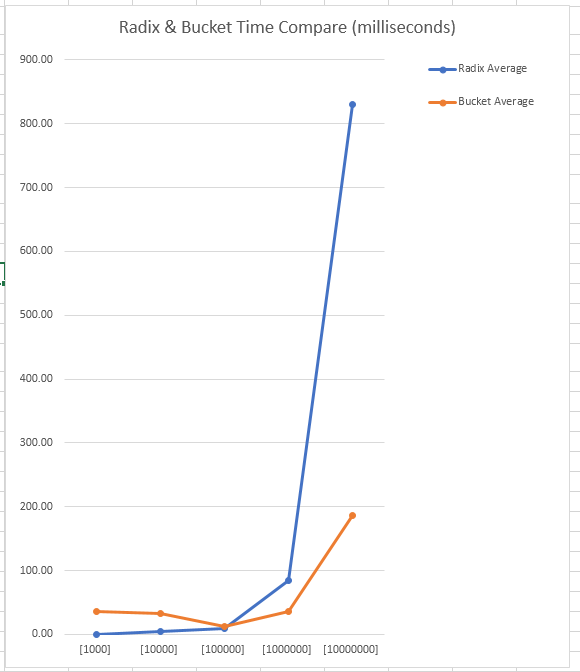


Proof that QuickSort works correctly



Proof that MergeSort works correctly.

**Run your code for 1~3 three times, record the execution time in milliseconds for each run on each size, enter the milliseconds reading into an Excel spreadsheet, calculate the average execution time in milliseconds and provide your results in a table and/or as a line chart.**



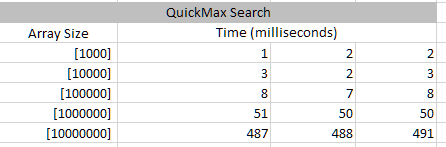
**Use your Lab 1 read method to from my data file. Then write recursive algorithm to list the largest 10 elements of the data you read, and listing them in decreasing order as the output. Again, starts with 1,000 and increases at 10x until it needs to read more than 10 million numbers. Output the execution time of your approach.**

The scenario method below is what allows me to increase the size of the array starting at 1k and going to 10mil. I used this scenario method for all time tests by replacing the area with the “// preform the sort” comment and calling the sort I wanted to test.

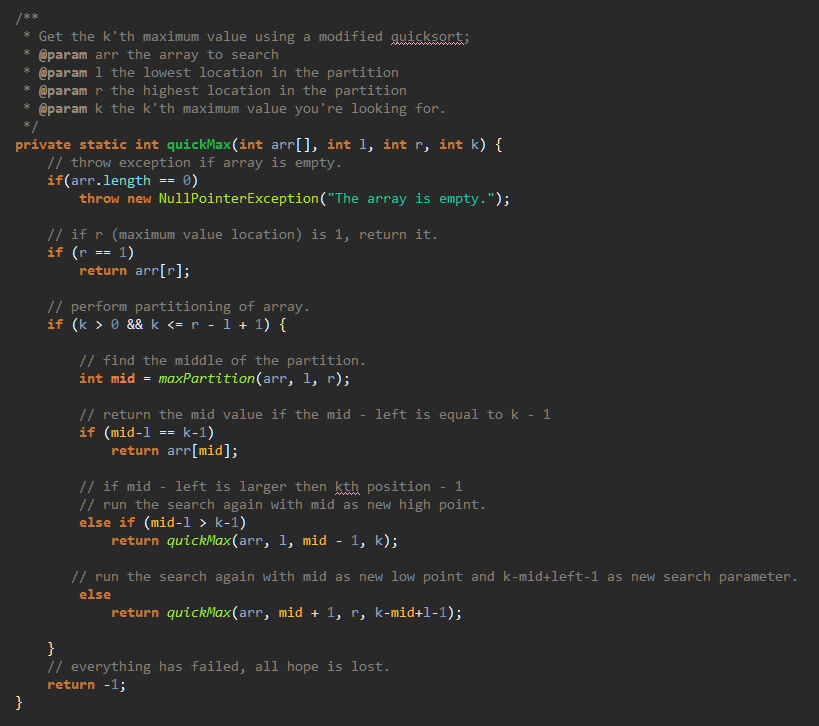
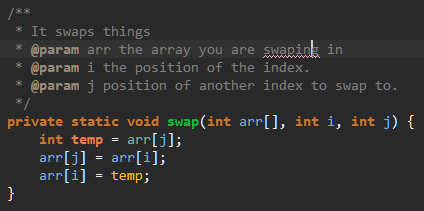
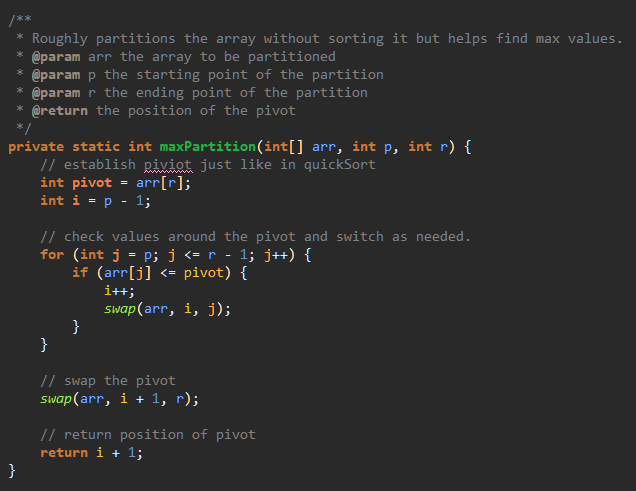
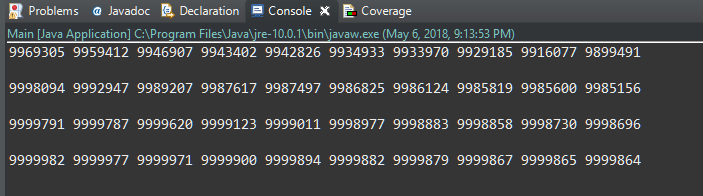


This was a nightmare and took me three attempts to find something that would work. However, I was able to get something working using [oppansource.com](http://oppansource.com/merge-sort-quick-sort-and-finding-kth-largest-element-in-an-unsorted-array/). My initial goal was to go through the array, finding the maximum and storing it in a separate array. Then go through it again and grab the next lowest number as long as it was not already stored in the maxList array. The end result is a modified quicksort that does not fully sort and instead stops at the point where pivot itself is k’th smallest element

Time Totals:

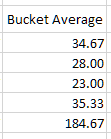
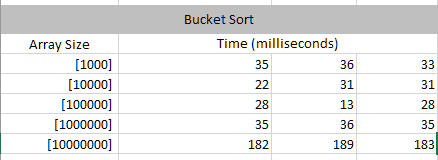


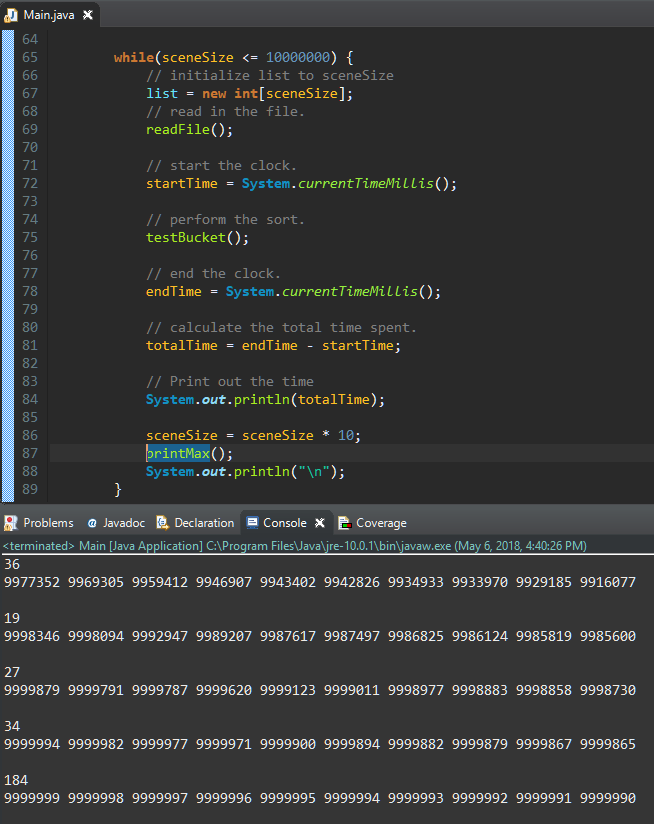
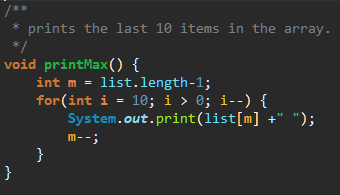
Code:

****

**Test your result by calling one of your sorting algorithm to sort the data first and display largest numbers in decreasing order as the output. Output the execution time of your approach.**

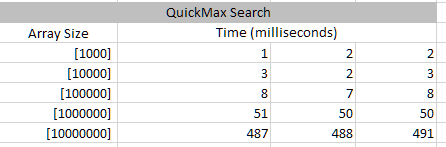
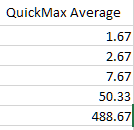
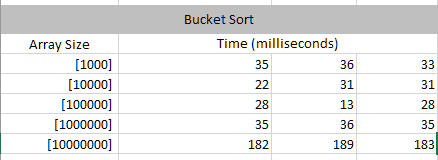
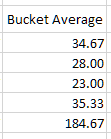
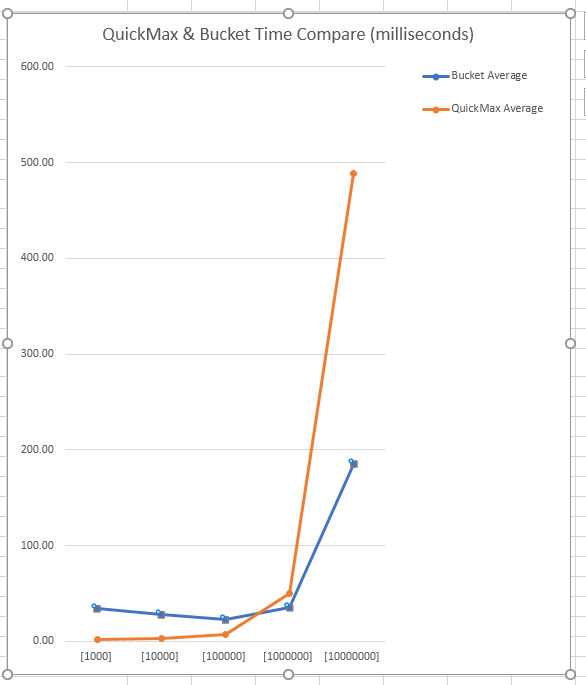
Simple and straightforward. Ran the standard time test but added the print max method that grabs the list.length-1 and then using a for loop works its way down 10 slots from the maximum position in the list.





**Run your code for part 6 and 7 three times, record the execution time in milliseconds for each run on each size, enter the milliseconds reading into an Excel spreadsheet, calculate the average execution time in milliseconds for each run on each size and display your results in both a table and as a line chart.**

I can only post the information from the bucket sorted list since I was not able to get my recursive version working.



**Write a half to one page report to explain your execution time observation and discuss the problem solving approach you applied for step 6. Is it DP, greedy algorithm, or divide-and-conquer?**

This was a nightmare. My goal was to go through the array, finding the maximum and storing it in a separate array. Then go through it again and grab the next lowest number as long as it was not already stored in the maxList array. My plan was to make it recursive once I was able to figure out how to accomplish this in a non-recursive way. Eventually I found a website that helped explain how this could be done using a modified quicksort algorithm and I was able to implement that into my program. It runs faster than quicksort in general which makes since because it is not doing nearly as much work. However my bucket sort algorithm which I tested it against was much faster at sorting the whole array and outputting the final 10 items in it.

**First attempt:**I tried using modulo to filter through and keep track of the last 10 max values. The problem I ran into here was that if the max value did not change from the previous time through it just overwrote all the value in the array.

**Attempt two:**Here I tried to loop through the max list and then when picking a number for each slot find the maximum value as long as it was not equal to the previous maximum value. This ended up getting me all the same number as well or all zeros.