

CS361 Algorithm Lab 2

What to do

1. Implement the Radix sort algorithm and use it to sort roughly 10,000,000 numbers. I am providing a new data file (on Moodle).
2. Implement the Bin sort algorithm and use it to sort roughly 10,000,000 numbers.
3. Make sure the results are sorted for 1 and 2. Show the screen dump indicate the sorting algorithms are actually sorting correctly.

```
381      /***** RADIX SORT *****/
382
383      int x = 1;
384      for(int y = 1000; y <= arr.length; y = y * 10){
385          long radixSortTime = System.nanoTime();
386          lab2RadixSort.radixsort(arr,y);
387          System.out.println("Radix sort ran " + x + ": " + (System.nanoTime()
388              x++;
389      }
390
391      if(lab2RadixSort.flagIsSorted(arr)){
392          System.out.println("The array was sorted using radix sort.");
393      }else{
394          System.out.println("It didn't work.");
395      }
396
397      /***** RADIX SORT *****/
398
399      /***** QUICK SORT *****/
```

```
Problems Javadoc Declaration Console
<terminated> CS361Labs [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (May 3, 2018, 5:27:30
Radix sort ran 1: 651328
Radix sort ran 2: 3029337
Radix sort ran 3: 16623786
Radix sort ran 4: 101228499
Radix sort ran 5: 1090385866
The array was sorted using radix sort.
```

As we can see the `flagIsSorted()` method is use in an if statement so that the statement “The array was sorted using radix sort.” Will print to the console only if the array is sorted. Otherwise the statement “It didn’t work.” Prints to the console. As we can see the proper statement is printed.

```
361      /***** BIN SORT *****/
362
363
364      int x = 1;
365      for(int y = 1000; y <= arr.length; y = y * 10){
366          long binSortTime = System.nanoTime();
367          lab2BinSort.binSort(arr,y);
368          System.out.println("Bin sort ran " + x + ": " + (System.nanoTime() - binSortTime));
369          x++;
370      }
371
372      if(lab2BinSort.flagIsSorted(arr)){
373          System.out.println("The array was sorted using bin sort.");
374      }else{
375          System.out.println("It didn't work.");
376      }
377
378
379      /***** BIN SORT *****/
```

```
Problems Javadoc Declaration Console
<terminated> CS361Labs [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (May 3, 2018, 5:35:33 AM)
Bin sort ran 1: 26292141
Bin sort ran 2: 22545399
Bin sort ran 3: 27129239
Bin sort ran 4: 46832581
Bin sort ran 5: 234057560
The array was sorted using bin sort.
```

As we can see the `flagIsSorted()` method is use in an if statement so that the statement “The array was sorted using bin sort.” Will print to the console only if the array is sorted. Otherwise the statement “It didn’t work.” Prints to the console. As we can see the proper statement is printed.

4. 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 sorted.

1	First Run	Bin sort	Radix sort	Quick sort
2	1 to 1000	32450497	637358	454231
3	1 to 10000	28636922	3018010	1775767
4	1 to 100000	34032187	15717967	12404311
5	1 to 1000000	53449699	103563842	94535488
6	1 to 10000000	182235238	1047301160	2090318656
7				
8	Second Run	Bin sort	Radix sort	Quick sort
9	1 to 1000	16402524	635470	635847
10	1 to 10000	22839157	3062565	1641725
11	1 to 100000	26219268	15649625	10107104
12	1 to 1000000	45718338	100030433	95174356
13	1 to 10000000	172285207	1044499882	2081354491
14				
15	Third Run	Bin sort	Radix sort	Quick sort
16	1 to 1000	26629699	656236	478774
17	1 to 10000	22783652	3065208	1725170
18	1 to 100000	26044825	13571416	9882821
19	1 to 1000000	44671681	101422577	94384078
20	1 to 10000000	173543309	1045383423	2073495885

```

401     int x = 1;
402     for(int y = 1000; y <= arr.length; y = y * 10){
403         long quickSortTime = System.nanoTime();
404         lab2QuickSort.auxQuickSort(arr, 0, y - 1);
405         System.out.println("Quick sort ran " + x + ": " + (System.na
406             x++);
407     }
408     if(lab2QuickSort.flagIsSorted(arr)){
409         System.out.println("The array was sorted using quick sort.")
410     }else{
411         System.out.println("It didn't work.");
412     }
413
414     /***** QUICK
415     /***** MERGE SC
416     /*
417     .

```

Problems @ Javadoc Declaration Console

<terminated> CS361Labs [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (May 3, 2018)

Quick sort ran 1: 592426

Quick sort ran 2: 2151082

Quick sort ran 3: 11799048

Quick sort ran 4: 94921754

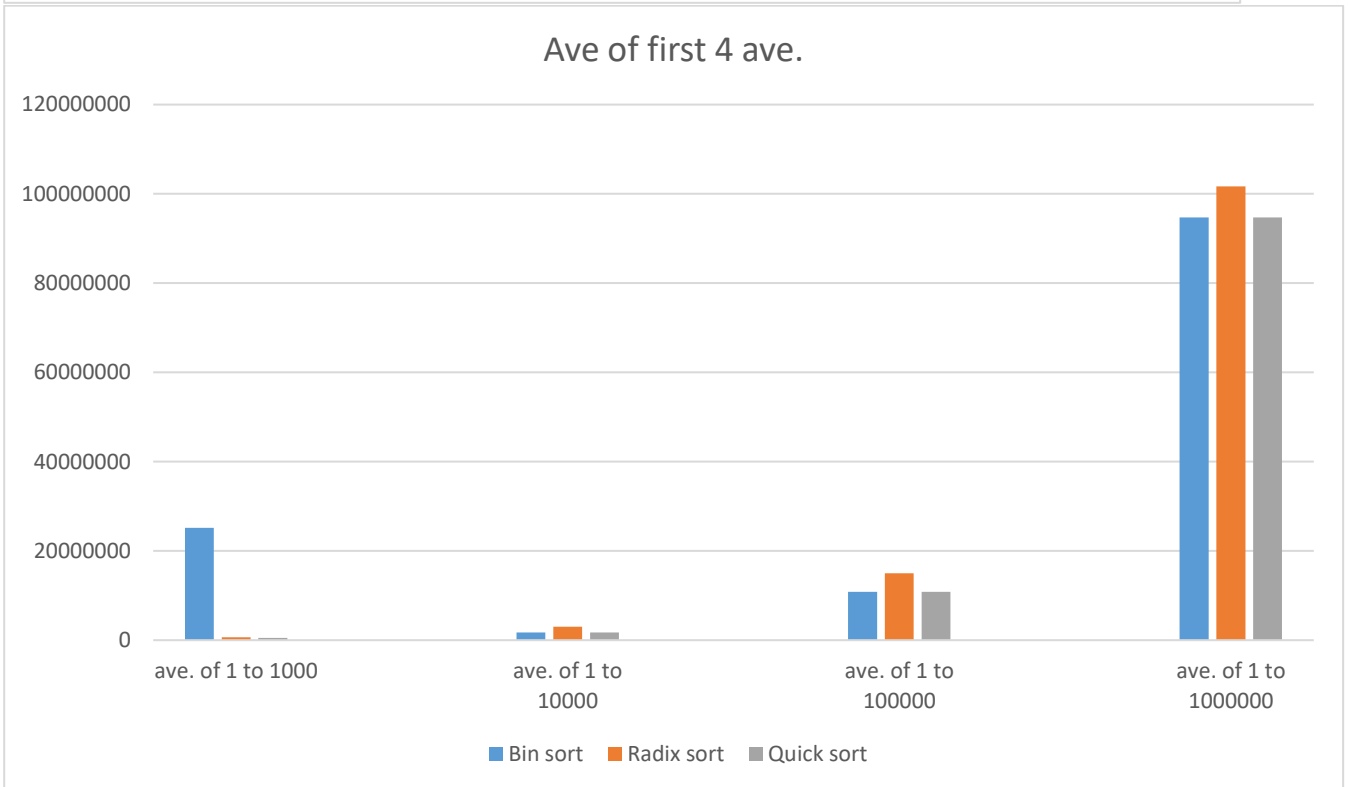
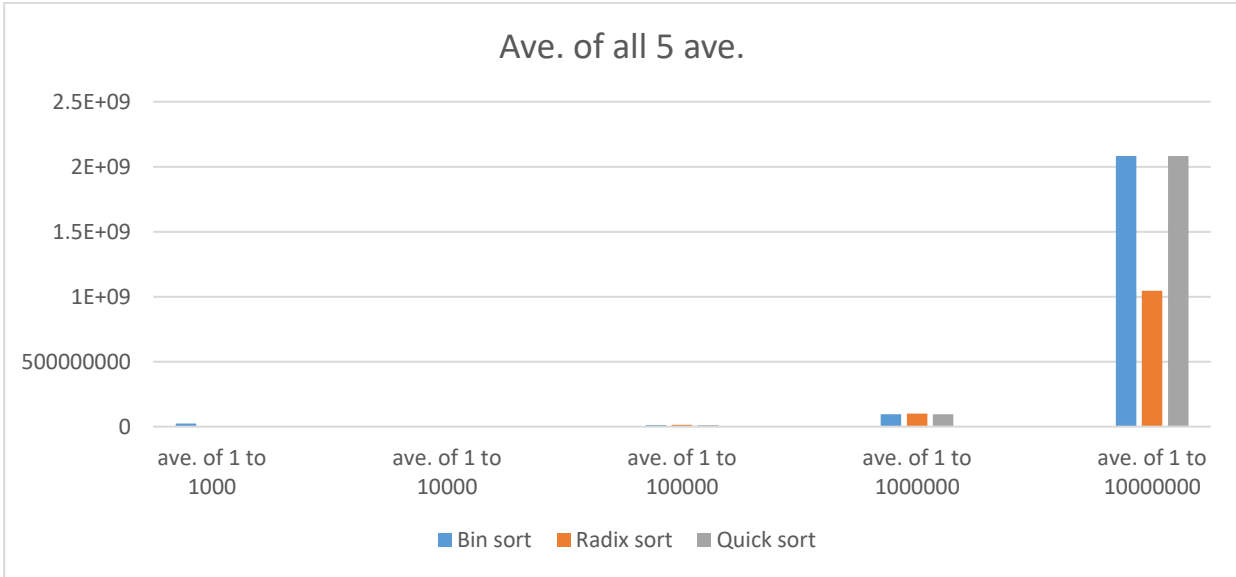
Quick sort ran 5: 2082709631

The array was sorted using quick sort.

As we can see the `flagIsSorted()` method is use in an if statement so that the statement “The array was sorted using quick sort.” Will print to the console only if the array is sorted. Otherwise the statement “It didn’t work.” Prints to the console. As we can see the proper statement is printed.

5. 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.

23				
24		Bin sort	Radix sort	Quick sort
25	ave. of 1 to 1000	25160906.67	643021.3333	522950.6667
26				
27	ave. of 1 to 10000	1714220.667	3048594.333	1714220.667
28				
29	ave. of 1 to 100000	10798078.67	14979669.33	10798078.67
30				
31	ave. of 1 to 1000000	94697974	101672284	94697974
32				
33	ave. of 1 to 10000000	2081723011	1045728155	2081723011
34				



6. 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, start with 1,000

and increases at 10x until it needs to read more than 10 million numbers. Output the execution time of your approach.

```
/**
 * This method uses recursion to find and sort the top ten values in the array
 * passed and stores them in the array topTen which is a field held in this class.
 *
 * @param ray The array that you want to find the top ten integers from.
 * @param y The starting index of the array that we want to look to.
 * @param n The ending index of the array that we want to look to.
 */
public void step6(int[] ray, int y, int n){

    int temp = getIndexOfMax(ray, 0, n);    // Get the max value's index from the array
    // in the array give the interval 0 to n and store the value in temp.
    topTen[y] = ray[temp];                // store the max values of the array and store it in the field array topTen.
    ray[temp] = ray[n];                    // Take where the max number is set it in to what the last value in the array is.
    ray[n] = topTen[y];                    // Now take the last value in the array to what the max value is.
    if(y < 9){                             // Only make the recursive call 10 times.
        step6(ray, y+1, n-1);              // Recursive call.
    }

}
```

Above is the recursive method that I used to sort the top ten integers in descending order. Below is a helper method for the recursive method.

```
/**
 * The method will parse through the array and find the index of the max
 * value found in the array.
 *
 * @param arrRec the array that will be passed.
 * @param y The starting of the index of the array that we are looking at.
 * @param n The ending of the index of the array that we are looking at.
 * @return The index of the max value found in the array passed.
 */
public int getIndexOfMax(int[] arrRec, int y, int n) {
    int indexOfMax = 0;
    for (int i = y; i < n; i++)
        if (arrRec[i] > arrRec[indexOfMax]) {
            indexOfMax = i;
        }
    return indexOfMax;
}
```

```
454      /*****Recursive Alg.*****/
455      int x = 1;
456      for(int y = 1000; y <= arr.length; y = y * 10){
457          long recursiveTime = System.nanoTime();
458          lab2Recursive.step6(arr, 0, y - 1);
459          System.out.println(x + ". The time it took :" + (System.nanoTime()-recursiveTime));
460          x++;
461          int w = 1;
462          for(int p:topTen){
463              System.out.println("The top ten " + w + "." + p);
464              w++;
465          }
466      }
467
468      /*****Recursive Alg.*****/
```

7. 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.

```

Problems @ Javadoc Declaration
<terminated> CS361Labs [Java Applic
0.) 9977352
1.) 9969305
2.) 9959412
3.) 9946907
4.) 9943402
5.) 9942826
6.) 9934933
7.) 9933970
8.) 9929185
9.) 9916077
Radix sort ran 1: 981712
0.) 9998346
1.) 9998094
2.) 9992947
3.) 9989207
4.) 9987617
5.) 9987497
6.) 9986825
7.) 9986124
8.) 9985819
9.) 9985600
Radix sort ran 2: 3253621
0.) 9999879
1.) 9999791
2.) 9999787
3.) 9999620
4.) 9999123
5.) 9999011
6.) 9998977
7.) 9998883
8.) 9998858
9.) 9998730
Radix sort ran 3: 13961836

```

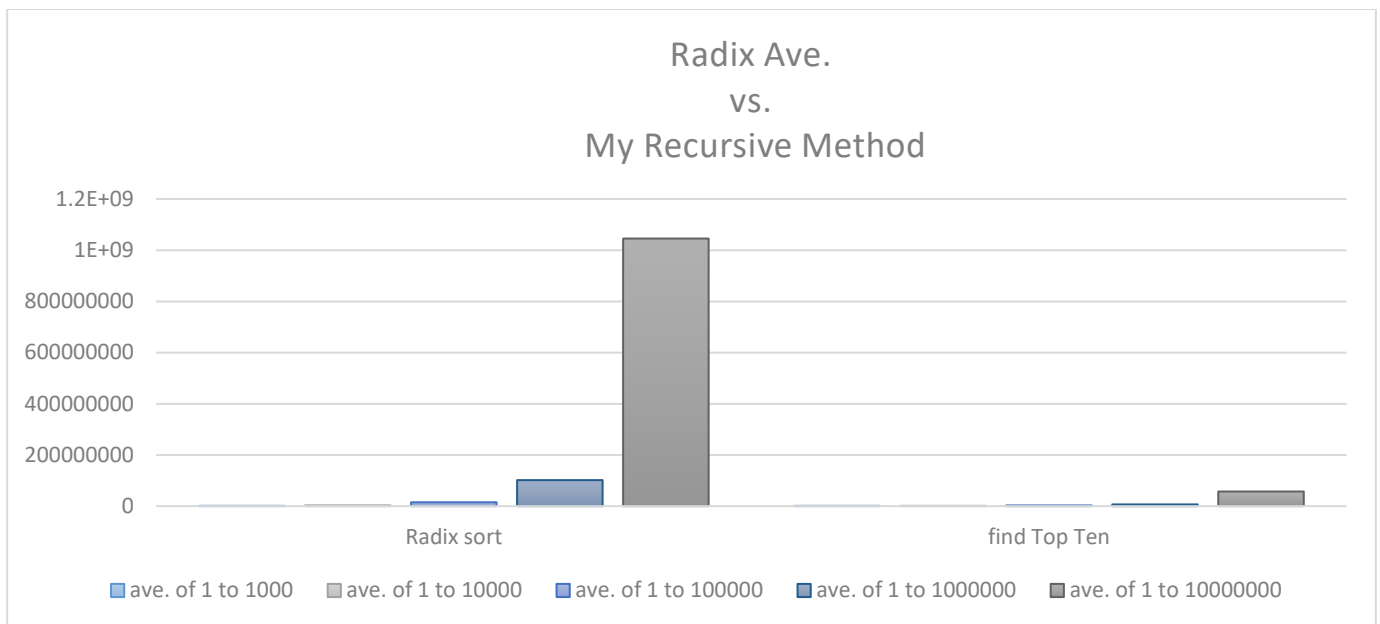
```

1.) The time it took :234856
The top ten 1.)9977352
The top ten 2.)9969305
The top ten 3.)9959412
The top ten 4.)9946907
The top ten 5.)9943402
The top ten 6.)9942826
The top ten 7.)9934933
The top ten 8.)9933970
The top ten 9.)9929185
The top ten 10.)9916077
2.) The time it took :1343435
The top ten 1.)9998346
The top ten 2.)9998094
The top ten 3.)9992947
The top ten 4.)9989207
The top ten 5.)9987617
The top ten 6.)9987497
The top ten 7.)9986825
The top ten 8.)9986124
The top ten 9.)9985819
The top ten 10.)9985600
3.) The time it took :3080311
The top ten 1.)9999879
The top ten 2.)9999791
The top ten 3.)9999787
The top ten 4.)9999620
The top ten 5.)9999123
The top ten 6.)9999011
The top ten 7.)9998977
The top ten 8.)9998883
The top ten 9.)9998858
The top ten 10.)9998730
4.) The time it took :6801755
The top ten 1.)9999994

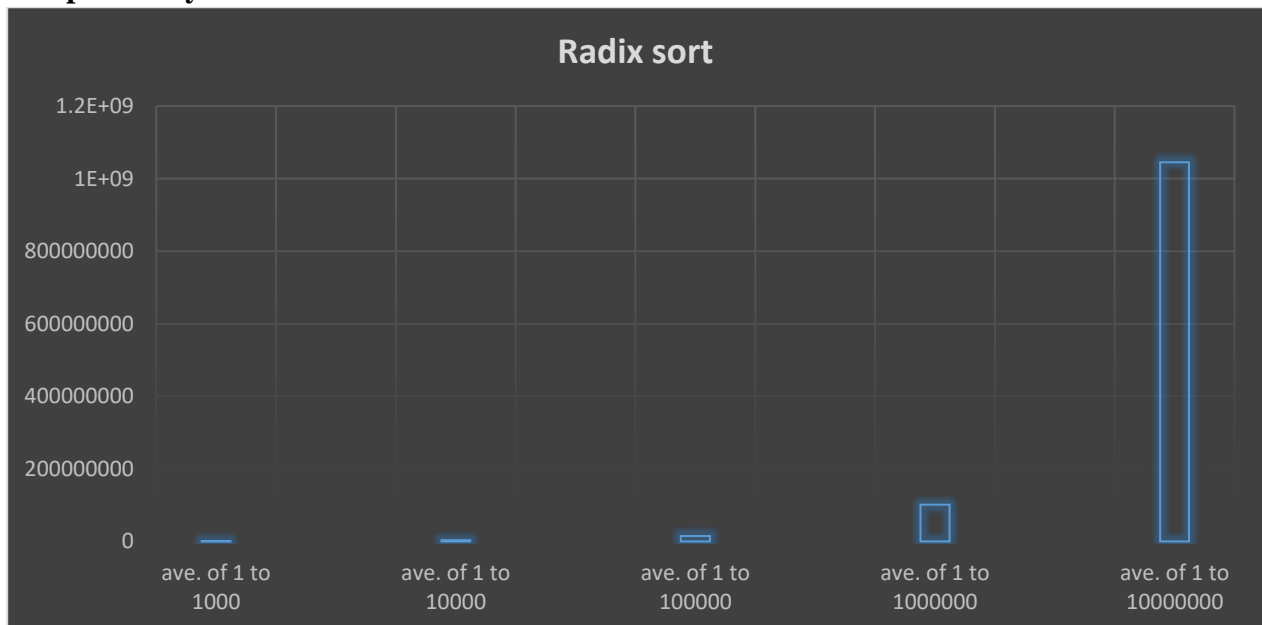
```

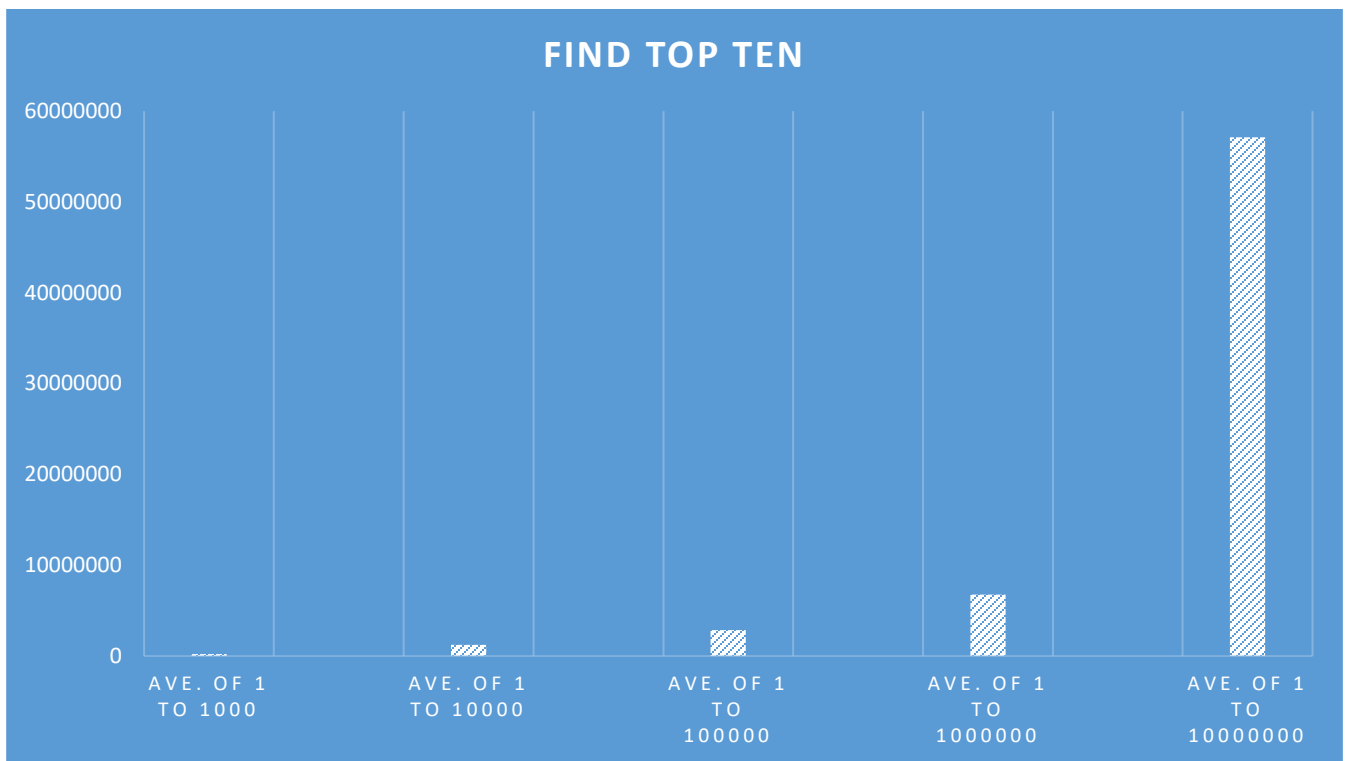
On the right is the print out of the recursive method that I wrote and executed to print out the top ten values in decreasing order. On the left is the same thing only radix sort is used to sort all of the integers in the array before printing the top ten values in decreasing order.

8. 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.



The above is them side by side. In order to have a better perspective I will show them independently.





9. 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?

To solve the problem, I went through many executions and approaches that ended up with stack over flow errors. Through all of them I was trying dynamic programming to store the values into an array (topTen) of size 10. I was then reminded of the pivot strategy that we used for partition method, so I thought that I could use that kind of methodology to solve this problem. Only worrying about the ten values that I care or have been tasked to find. I also employed divide-and-conquer as I separated the max value out of the array that I was parsing each time to my problem smaller each time.

My execution time appears to be some were above $n \log n$ time and below n^2 its also not n time nor constant time. So, I can conjecture that the time complexity is $n \log n$. It does not take long to find the top ten values.