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] 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

```java
/**
 * A counting sort algorithm used by radix sort.
 * @param arr the array to be sorted.
 * @param n the size of the array
 * @param digit the digit to be sorted.
 */
static void countSort(int arr[], int n, int digit)
{
    // the output array, used to keep track of sorted numbers.
    int output[] = new int[n];
    int i;
    int count[] = new int[10];
    Arrays.fill(count,0);

    // store the number of occurrences in count[]
    for (i = 0; i < n; i++)
        count[(arr[i]/digit)%10]++;

    // adjust so count[i] now contains digit location in output array
    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    // create the output array for transferring back into original array.
    for (i = n - 1; i >= 0; i--)
    {
        output[count[(arr[i]/digit)%10 ] - 1] = arr[i];
        count[(arr[i]/digit)%10]--;
    }

    // adjusts the array so that they are sorted by the current digit.
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}
```

```java
/**
 * Find the maximum value stored in an array.
 * @param arr the array to find max value in.
 * @return the maximum value stored in the array.
 */
private static int getMax(int arr[])
{
    // set max to first value in array.
    int m = arr[0];
    // for each item in the array
    for (int i = 1; i < arr.length; i++)
        // check to see if the next element in array is larger then maximum.
        if (arr[i] > m)
            m = arr[i];

    // return maximum.
    return m;
}
```

```java
/**
 * Driver of radixSort
 * @param arr the array to be sorted.
 * @param n the size of the array (useful for when growing the array from 1000 to 10000000 later.
 */
static void radixSort(int arr[], int n)
{
    // throw exception if array is empty.
    if(arr.length == 0)
        throw new NullPointerException("The array is empty.");

    // find the maximum number (allows us to know number of digits)
    int m = getMax(arr);

    // preform the counting sort algorithm on each digit.
    for (int digit = 1; m / digit > 0; digit *= 10)
        countSort(arr, n, digit);
}
```

bucket sort as well.

Main.java

```
64
65          while(sceneSize <= 10000000) {
66              // initialize list to sceneSize
67              list = new int[sceneSize];
68              // read in the file.
69              readFile();
70
71              // start the clock.
72              startTime = System.currentTimeMillis();
73
74              // perform the sort.
75              testBucket();
76
77              // end the clock.
78              endTime = System.currentTimeMillis();
79
80              // calculate the total time spent.
81              totalTime = endTime - startTime;
82
83              // Print out the time
84              System.out.println(totalTime);
85
86              sceneSize = sceneSize * 10;
87              printMax();
88              System.out.println("\n");
89          }
```

Problems | Javadoc | Declaration | Console | Coverage

<terminated> Main [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (May 6, 2018, 4:40:26 PM)

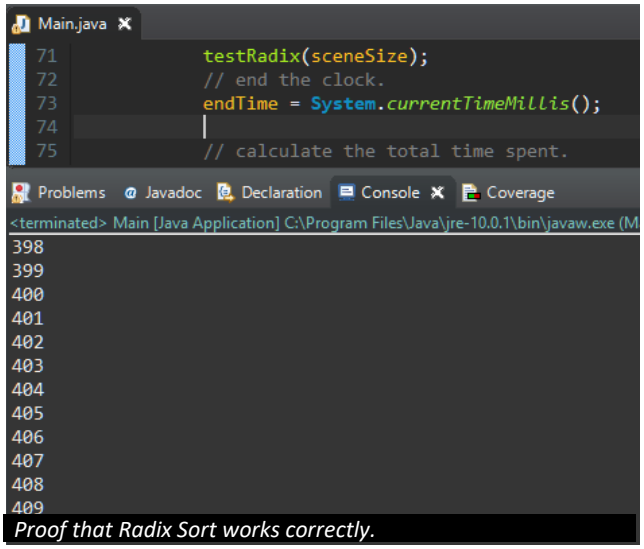
```
36
9977352 9969305 9959412 9946907 9943402 9942826 9934933 9933970 9929185 9916077

19
9998346 9998094 9992947 9989207 9987617 9987497 9986825 9986124 9985819 9985600

27
9999879 9999791 9999787 9999620 9999123 9999011 9998977 9998883 9998858 9998730

34
9999994 9999982 9999977 9999971 9999900 9999894 9999882 9999879 9999867 9999865

184
9999999 9999998 9999997 9999996 9999995 9999994 9999993 9999992 9999991 9999990
```

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

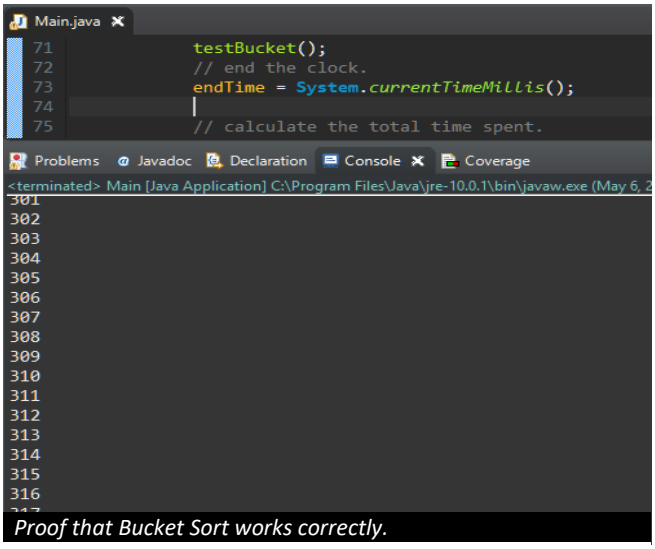
Struggled with getting this to work, I did not want to have to use the SIZE global variable which was the only way I could get it to work until I made a get max loop to find the largest item stored in the sorted array. I most likely added some O time by doing this… used a combination of the book, along with sanfoundy.com, javacodex.com, and growingwiththeweb.com. For proof that it works see next question.

```java
/**
 * Simple bucket sort.
 * @param arr the array to be sorted.
 */
public static void bucketSort(int arr[]) {
    // throw exception if array is empty.
    if(arr.length == 0)
        throw new NullPointerException("The array is empty.");

    // get maximum value in array
    int max = getMax(arr);

    // set the size of the bucket to global variable SIZE
    int [] bucket = new int[max+1];

    // fill the bucket with zeros.
    for (int i = 0; i < bucket.length; i++) {
       bucket[i] = 0;
    }

    // incremented each location for each item in initial array.
    for (int i = 0; i < arr.length; i++) {
        bucket[arr[i]]++;
     }

    // put the bucket back into the initial array.
    int o = 0;
    for (int i = 0; i < bucket.length; i++) {
       for (int j = 0; j < bucket[i]; j++) {
          arr[o++]=i;
       }
    }
```

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



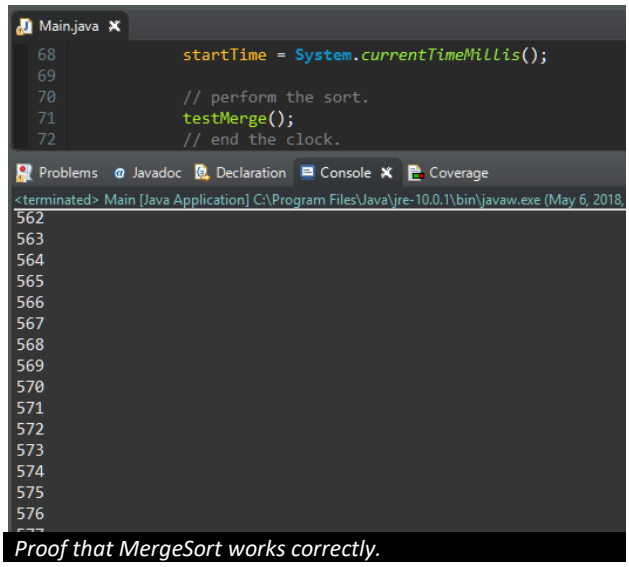
Proof that Radix Sort works correctly.



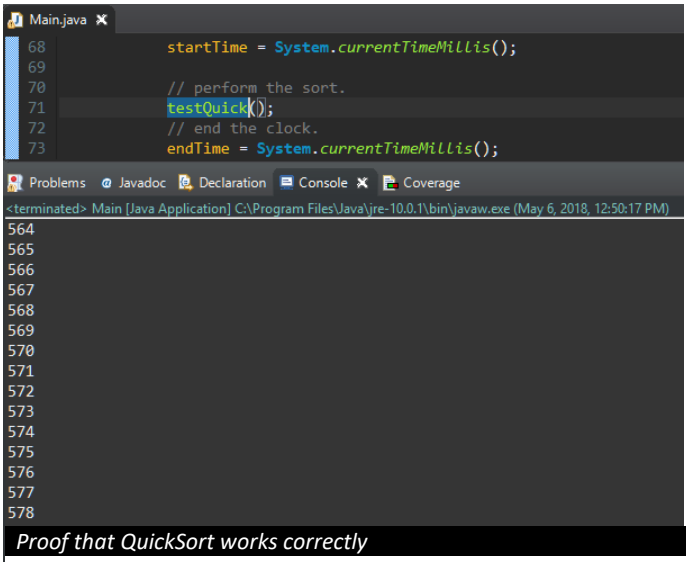
Proof that Bucket Sort works correctly.

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.

Array Size	Radix Average	Bucket Average	Quicksort Average	Mergesort Average
[1000]	0.67	36.33	0.67	0.67
[10000]	4.33	32.33	1.33	2.33
[100000]	9.67	13.00	7.00	10.67
[1000000]	84.00	35.33	95.00	130.00
[10000000]	829.67	186.67	1102.33	1503.67

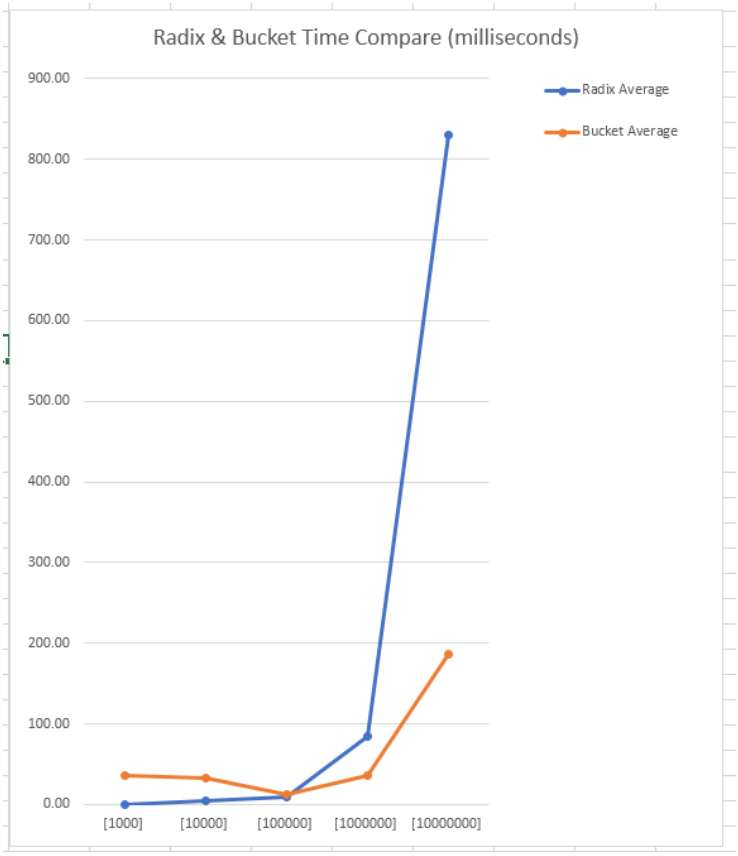


Proof that MergeSort works correctly.



Proof that QuickSort 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 method is for testing the speed of the sorts,
 * replace the test"X" method call to test each sort.
 * @throws FileNotFoundException needed for the readFile() method.
 */
public void scenario() throws FileNotFoundException {
    int sceneSize = 1000;
    long startTime, endTime, totalTime = 0;


    while(sceneSize <= 10000000) {
        // initialize list to sceneSize
        list = new int[sceneSize];
        // read in the file.
        readFile();

        // start the clock.
        startTime = System.currentTimeMillis();

        // perform the sort.
        testMerge();

        // end the clock.
        endTime = System.currentTimeMillis();

        // calculate the total time spent.
        totalTime = endTime - startTime;

        // Print out the time
        System.out.println(totalTime);

        sceneSize = sceneSize * 10;
        //printMax();
        //System.out.println("\n");
    }

}
```

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](). 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:

QuickMax Search			
Array Size	Time (milliseconds)		
[1000]	1	2	2
[10000]	3	2	3
[100000]	8	7	8
[1000000]	51	50	50
[10000000]	487	488	491

Code:

```java
/**
 * Get the k'th maximum value using a modified quicksort;
 * @param arr the array to search
 * @param l the lowest location in the partition
 * @param r the highest location in the partition
 * @param k the k'th maximum value you're looking for.
 */
private static int quickMax(int arr[], int l, int r, int k) {
    // throw exception if array is empty.
    if(arr.length == 0)
        throw new NullPointerException("The array is empty.");

    // if r (maximum value location) is 1, return it.
    if (r == 1)
        return arr[r];

    // perform partitioning of array.
    if (k > 0 && k <= r - l + 1) {

        // find the middle of the partition.
        int mid = maxPartition(arr, l, r);

        // return the mid value if the mid - left is equal to k - 1
        if (mid-l == k-1)
            return arr[mid];

        // if mid - left is larger then kth position - 1
        // run the search again with mid as new high point.
        else if (mid-l > k-1)
            return quickMax(arr, l, mid - 1, k);

       // run the search again with mid as new low point and k-mid+left-1 as new search parameter.
        else
            return quickMax(arr, mid + 1, r, k-mid+l-1);

    }
    // everything has failed, all hope is lost.
    return -1;
}
```

```java
/**
 * It swaps things
 * @param arr the array you are swaping in
 * @param i the position of the index.
 * @param j position of another index to swap to.
 */
private static void swap(int arr[], int i, int j) {
    int temp = arr[j];
    arr[j] = arr[i];
    arr[i] = temp;
}
```

```java
/**
 * Roughly partitions the array without sorting it but helps find max values.
 * @param arr the array to be partitioned
 * @param p the starting point of the partition
 * @param r the ending point of the partition
 * @return the position of the pivot
 */
private static int maxPartition(int[] arr, int p, int r) {
    // establish piviot just like in quickSort
    int pivot = arr[r];
    int i = p - 1;

    // check values around the pivot and switch as needed.
    for (int j = p; j <= r - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr, i, j);
        }
    }

    // swap the pivot
    swap(arr, i + 1, r);

    // return position of pivot
    return i + 1;
}
```

Problems | Javadoc | Declaration | Console | Coverage

Main [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (May 6, 2018, 9:13:53 PM)

```
9969305 9959412 9946907 9943402 9942826 9934933 9933970 9929185 9916077 9899491

9998094 9992947 9989207 9987617 9987497 9986825 9986124 9985819 9985600 9985156

9999791 9999787 9999620 9999123 9999011 9998977 9998883 9998858 9998730 9998696

9999982 9999977 9999971 9999900 9999894 9999882 9999879 9999867 9999865 9999864
```

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.

Bucket Sort				Bucket Average
Array Size	Time (milliseconds)			
[1000]	35	36	33	34.67
[10000]	22	31	31	28.00
[100000]	28	13	28	23.00
[1000000]	35	36	35	35.33
[10000000]	182	189	183	184.67

Main.java

```
64
65          while(sceneSize <= 10000000) {
66              // initialize list to sceneSize
67              list = new int[sceneSize];
68              // read in the file.
69              readFile();
70
71              // start the clock.
72              startTime = System.currentTimeMillis();
73
74              // perform the sort.
75              testBucket();
76
77              // end the clock.
78              endTime = System.currentTimeMillis();
79
80              // calculate the total time spent.
81              totalTime = endTime - startTime;
82
83              // Print out the time
84              System.out.println(totalTime);
85
86              sceneSize = sceneSize * 10;
87              printMax();
88              System.out.println("\n");
89          }
```

Problems Javadoc Declaration Console Coverage

<terminated> Main [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (May 6, 2018, 4:40:26 PM)

```
36
9977352 9969305 9959412 9946907 9943402 9942826 9934933 9933970 9929185 9916077

19
9998346 9998094 9992947 9989207 9987617 9987497 9986825 9986124 9985819 9985600

27
9999879 9999791 9999787 9999620 9999123 9999011 9998977 9998883 9998858 9998730

34
9999994 9999982 9999977 9999971 9999900 9999894 9999882 9999879 9999867 9999865

184
9999999 9999998 9999997 9999996 9999995 9999994 9999993 9999992 9999991 9999990
```

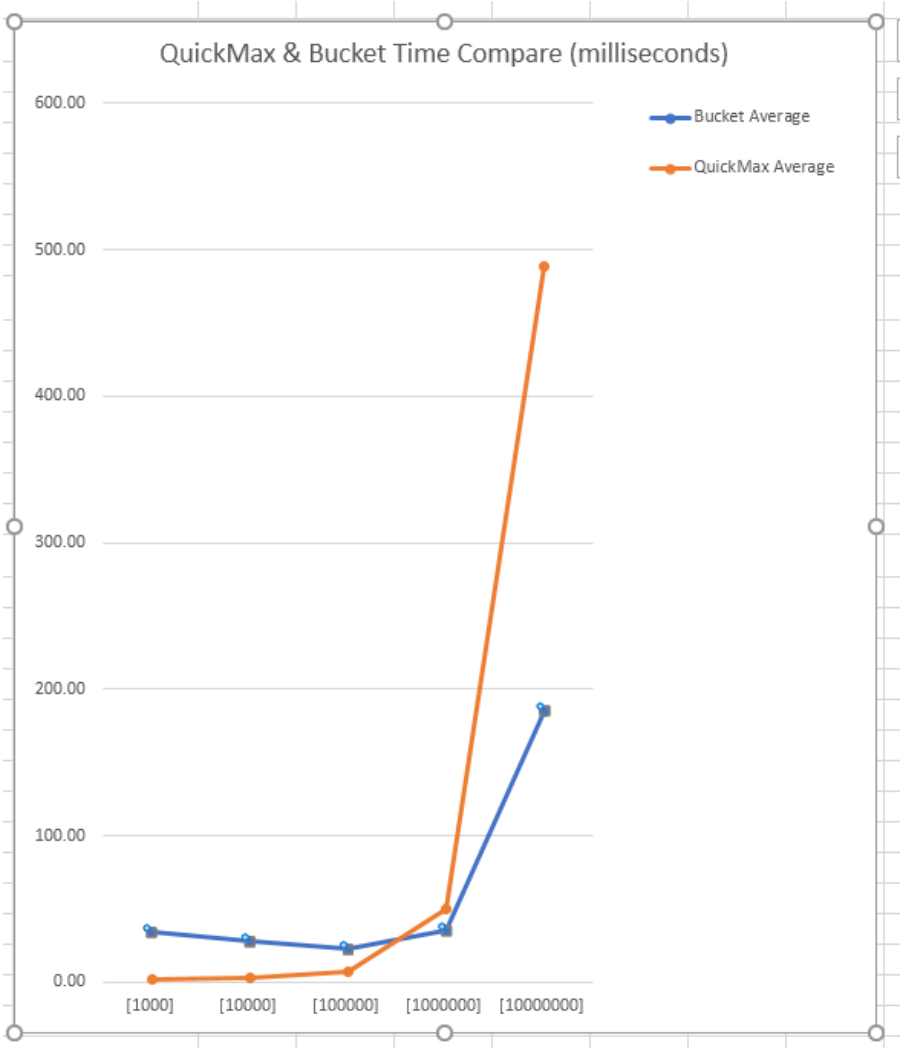
```
/**
 * prints the last 10 items in the array.
 */
void printMax() {
    int m = list.length-1;
    for(int i = 10; i > 0; i--) {
        System.out.print(list[m] +" ");
        m--;
    }
}
```

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.

QuickMax Search				
Array Size	Time (milliseconds)			QuickMax Average
[1000]	1	2	2	1.67
[10000]	3	2	3	2.67
[100000]	8	7	8	7.67
[1000000]	51	50	50	50.33
[10000000]	487	488	491	488.67

Bucket Sort				
Array Size	Time (milliseconds)			Bucket Average
[1000]	35	36	33	34.67
[10000]	22	31	31	28.00
[100000]	28	13	28	23.00
[1000000]	35	36	35	35.33
[10000000]	182	189	183	184.67



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.