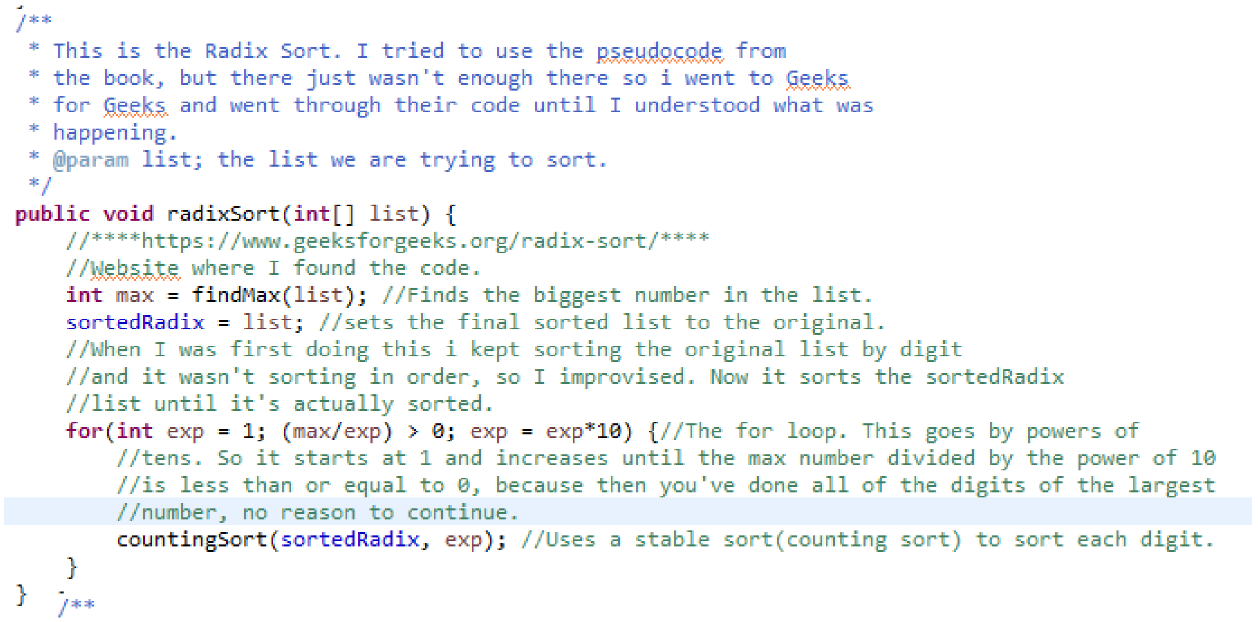
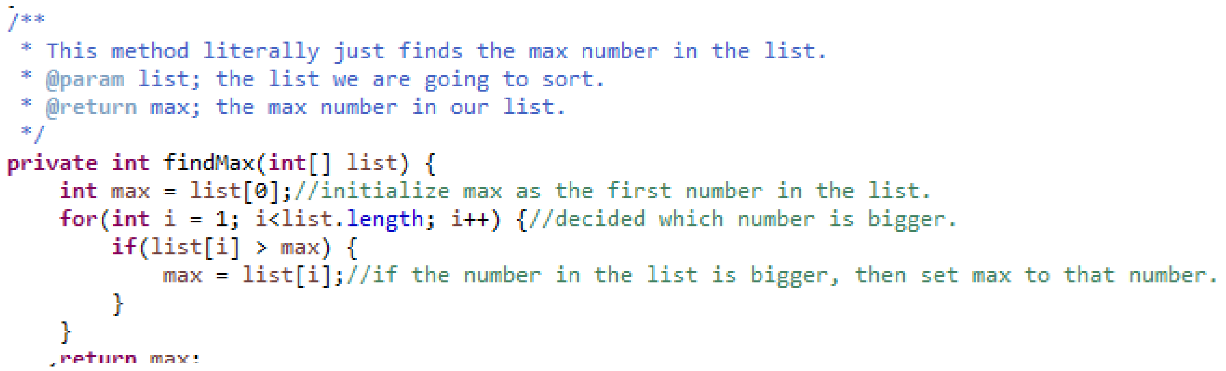
Tiffany Jansen
CS 361
Lab 2

# Lab Report

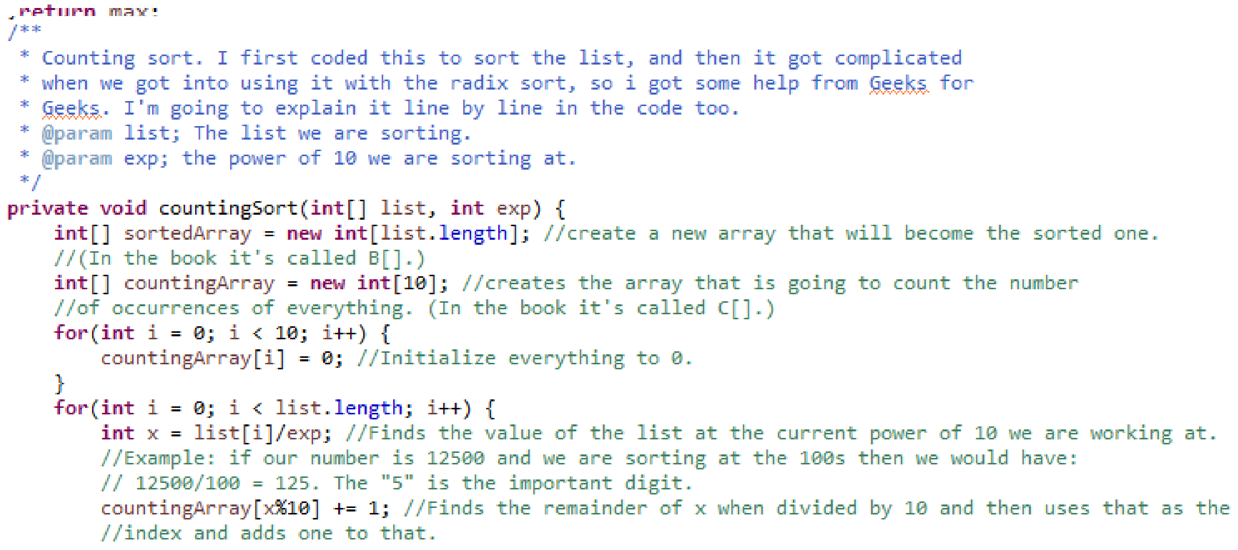
1. **Radix Sort Code:**



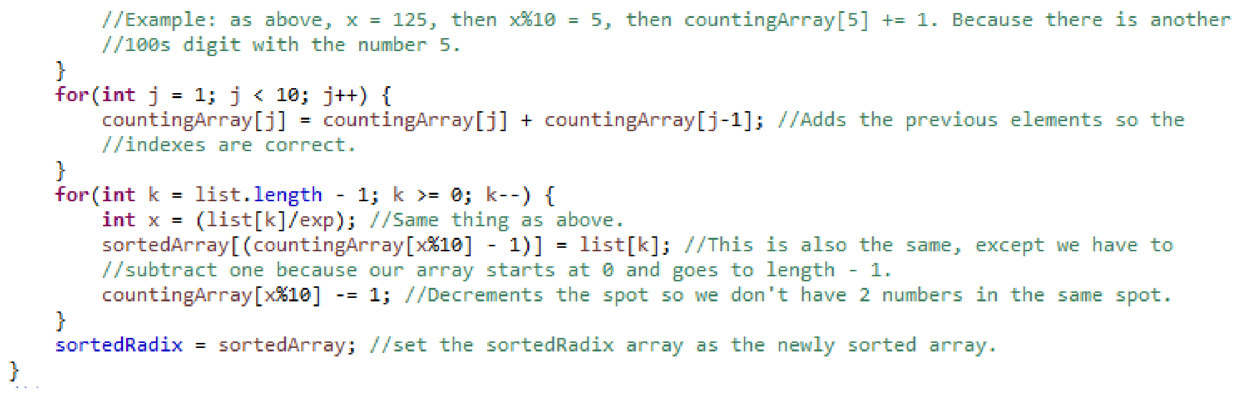
```
/**
 * This is the Radix Sort. I tried to use the pseudocode from
 * the book, but there just wasn't enough there so i went to Geeks
 * for Geeks and went through their code until I understood what was
 * happening.
 * @param list; the list we are trying to sort.
 */
public void radixSort(int[] list) {
    //****https://www.geeksforgeeks.org/radix-sort/****
    //Website where I found the code.
    int max = findMax(list); //Finds the biggest number in the list.
    sortedRadix = list; //sets the final sorted list to the original.
    //When I was first doing this i kept sorting the original list by digit
    //and it wasn't sorting in order, so I improvised. Now it sorts the sortedRadix
    //list until it's actually sorted.
    for(int exp = 1; (max/exp) > 0; exp = exp*10) {//The for loop. This goes by powers of
        //tens. So it starts at 1 and increases until the max number divided by the power of 10
        //is less than or equal to 0, because then you've done all of the digits of the largest
        //number, no reason to continue.
        countingSort(sortedRadix, exp); //Uses a stable sort(counting sort) to sort each digit.
    }
}
```



```
    /**
     * This method literally just finds the max number in the list.
     * @param list; the list we are going to sort.
     * @return max; the max number in our list.
     */
    private int findMax(int[] list) {
        int max = list[0];//initialize max as the first number in the list.
        for(int i = 1; i<list.length; i++) {//decided which number is bigger.
            if(list[i] > max) {
                max = list[i];//if the number in the list is bigger, then set max to that number.
            }
        }
        return max;
    }
```

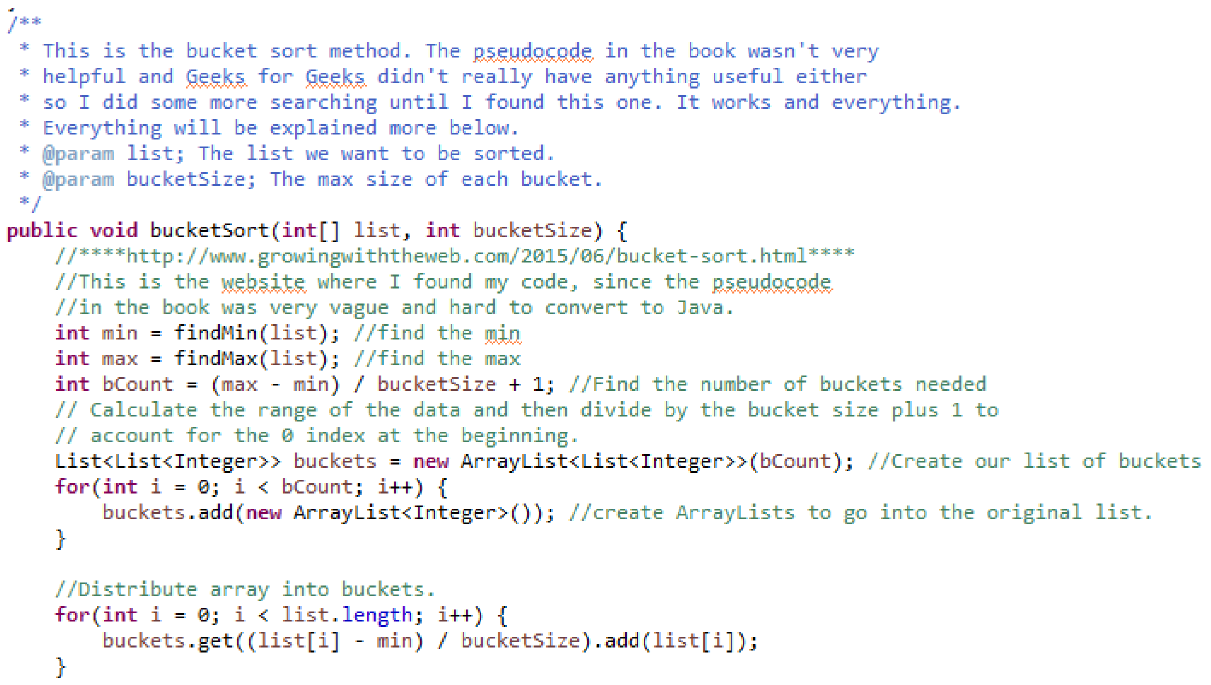


```
    /**
     * Counting sort. I first coded this to sort the list, and then it got complicated
     * when we got into using it with the radix sort, so i got some help from Geeks for
     * Geeks. I'm going to explain it line by line in the code too.
     * @param list; The list we are sorting.
     * @param exp; the power of 10 we are sorting at.
     */
    private void countingSort(int[] list, int exp) {
        int[] sortedArray = new int[list.length]; //create a new array that will become the sorted one.
        //(In the book it's called B[].)
        int[] countingArray = new int[10]; //creates the array that is going to count the number
        //of occurrences of everything. (In the book it's called C[].)
        for(int i = 0; i < 10; i++) {
            countingArray[i] = 0; //Initialize everything to 0.
        }
        for(int i = 0; i < list.length; i++) {
            int x = list[i]/exp; //Finds the value of the list at the current power of 10 we are working at.
            //Example: if our number is 12500 and we are sorting at the 100s then we would have:
            // 12500/100 = 125. The "5" is the important digit.
            countingArray[x%10] += 1; //Finds the remainder of x when divided by 10 and then uses that as the
            //index and adds one to that.
```

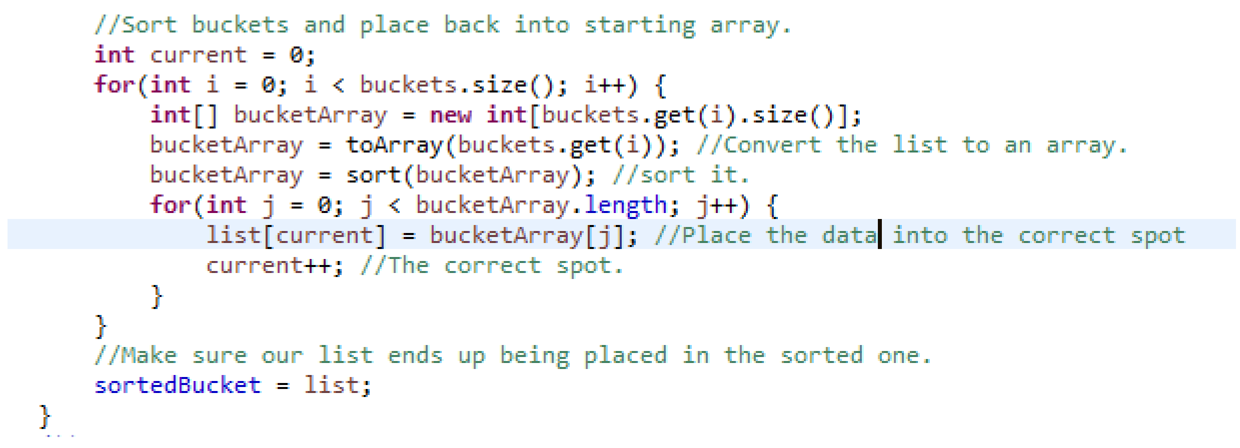


```java
            //Example: as above, x = 125, then x%10 = 5, then countingArray[5] += 1. Because there is another
            //100s digit with the number 5.
        }
        for(int j = 1; j < 10; j++) {
            countingArray[j] = countingArray[j] + countingArray[j-1]; //Adds the previous elements so the
            //indexes are correct.
        }
        for(int k = list.length - 1; k >= 0; k--) {
            int x = (list[k]/exp); //Same thing as above.
            sortedArray[(countingArray[x%10] - 1)] = list[k]; //This is also the same, except we have to
            //subtract one because our array starts at 0 and goes to length - 1.
            countingArray[x%10] -= 1; //Decrements the spot so we don't have 2 numbers in the same spot.
        }
        sortedRadix = sortedArray; //set the sortedRadix array as the newly sorted array.
    }
```

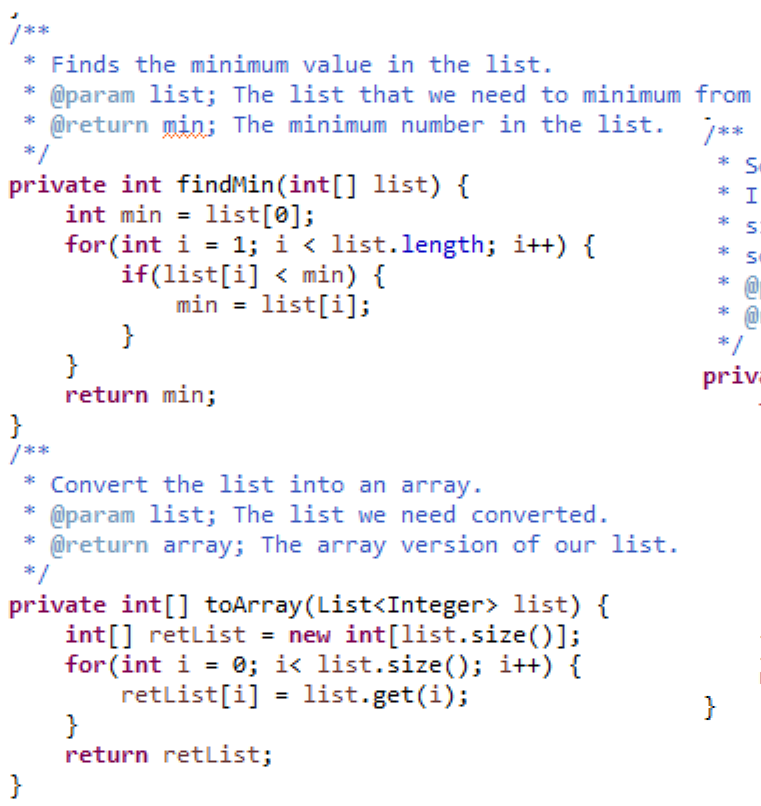
## 2. “Bin” (Bucket) Sort Code:



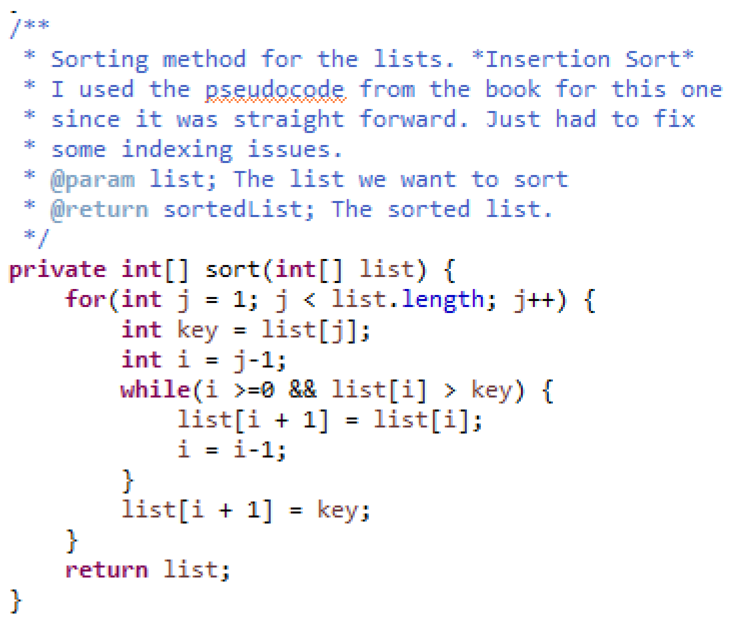
```java
    /**
     * This is the bucket sort method. The pseudocode in the book wasn't very
     * helpful and Geeks for Geeks didn't really have anything useful either
     * so I did some more searching until I found this one. It works and everything.
     * Everything will be explained more below.
     * @param list; The list we want to be sorted.
     * @param bucketSize; The max size of each bucket.
     */
    public void bucketSort(int[] list, int bucketSize) {
        //****http://www.growingwiththeweb.com/2015/06/bucket-sort.html****
        //This is the website where I found my code, since the pseudocode
        //in the book was very vague and hard to convert to Java.
        int min = findMin(list); //find the min
        int max = findMax(list); //find the max
        int bCount = (max - min) / bucketSize + 1; //Find the number of buckets needed
        // Calculate the range of the data and then divide by the bucket size plus 1 to
        // account for the 0 index at the beginning.
        List<List<Integer>> buckets = new ArrayList<List<Integer>>(bCount); //Create our list of buckets
        for(int i = 0; i < bCount; i++) {
            buckets.add(new ArrayList<Integer>()); //create ArrayLists to go into the original list.
        }

        //Distribute array into buckets.
        for(int i = 0; i < list.length; i++) {
            buckets.get((list[i] - min) / bucketSize).add(list[i]);
        }
```



```java
        //Sort buckets and place back into starting array.
        int current = 0;
        for(int i = 0; i < buckets.size(); i++) {
            int[] bucketArray = new int[buckets.get(i).size()];
            bucketArray = toArray(buckets.get(i)); //Convert the list to an array.
            bucketArray = sort(bucketArray); //sort it.
            for(int j = 0; j < bucketArray.length; j++) {
                list[current] = bucketArray[j]; //Place the data into the correct spot
                current++; //The correct spot.
            }
        }
        //Make sure our list ends up being placed in the sorted one.
        sortedBucket = list;
    }
```



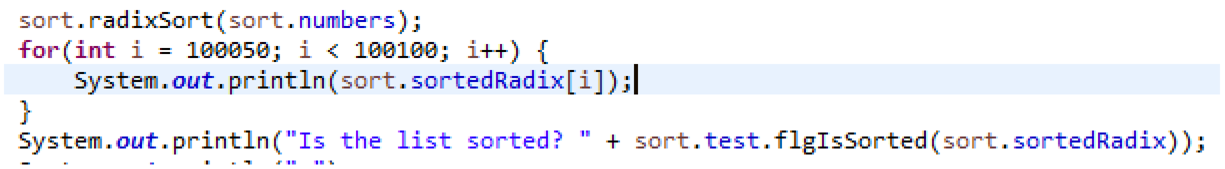
```
/**
 * Finds the minimum value in the list.
 * @param list; The list that we need to minimum from
 * @return min; The minimum number in the list.
 */
private int findMin(int[] list) {
    int min = list[0];
    for(int i = 1; i < list.length; i++) {
        if(list[i] < min) {
            min = list[i];
        }
    }
    return min;
}
/**
 * Convert the list into an array.
 * @param list; The list we need converted.
 * @return array; The array version of our list.
 */
private int[] toArray(List<Integer> list) {
    int[] retList = new int[list.size()];
    for(int i = 0; i< list.size(); i++) {
        retList[i] = list.get(i);
    }
    return retList;
}
```



```
/**
 * Sorting method for the lists. *Insertion Sort*
 * I used the pseudocode from the book for this one
 * since it was straight forward. Just had to fix
 * some indexing issues.
 * @param list; The list we want to sort
 * @return sortedList; The sorted list.
 */
private int[] sort(int[] list) {
    for(int j = 1; j < list.length; j++) {
        int key = list[j];
        int i = j-1;
        while(i >=0 && list[i] > key) {
            list[i + 1] = list[i];
            i = i-1;
        }
        list[i + 1] = key;
    }
    return list;
}
```

**3. Test Radix Sort and Bucket Sort:**

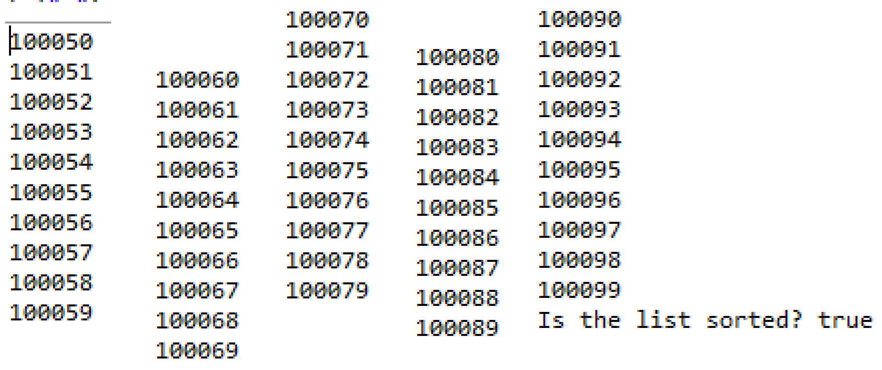
**Radix:**



```
sort.radixSort(sort.numbers);
for(int i = 100050; i < 100100; i++) {
    System.out.println(sort.sortedRadix[i]);
}
System.out.println("Is the list sorted? " + sort.test.flgIsSorted(sort.sortedRadix));
```

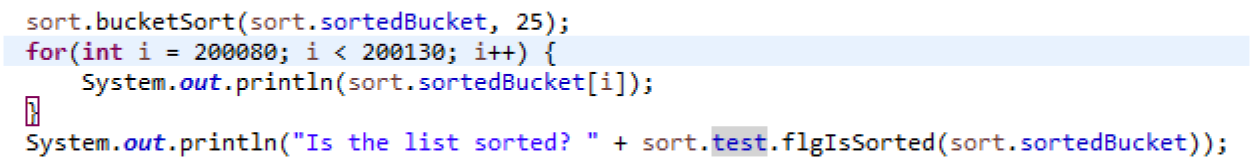
**Output:**

(for Radix)



```
100050
100051
100052
100053
100054
100055
100056
100057
100058
100059
100060
100061
100062
100063
100064
100065
100066
100067
100068
100069
100070
100071
100072
100073
100074
100075
100076
100077
100078
100079
100080
100081
100082
100083
100084
100085
100086
100087
100088
100089
100090
100091
100092
100093
100094
100095
100096
100097
100098
100099
Is the list sorted? true
```

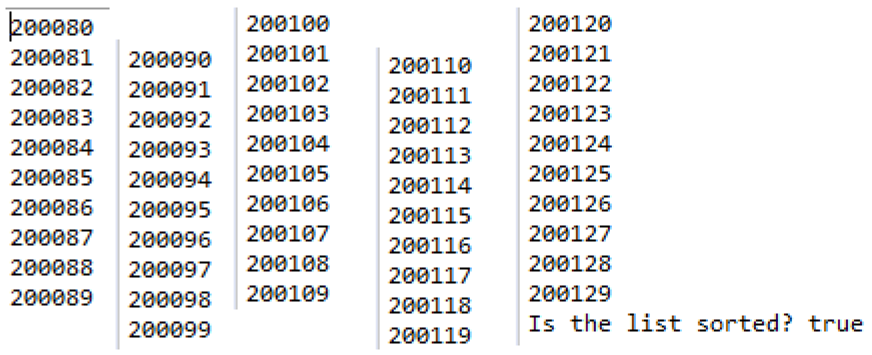
**Bucket:**



```
sort.bucketSort(sort.sortedBucket, 25);
for(int i = 200080; i < 200130; i++) {
    System.out.println(sort.sortedBucket[i]);
}
System.out.println("Is the list sorted? " + sort.test.flgIsSorted(sort.sortedBucket));
```

**Output:**

(For Bucket)



```
200080
200081
200082
200083
200084
200085
200086
200087
200088
200089
```

```
200090
200091
200092
200093
200094
200095
200096
200097
200098
200099
```

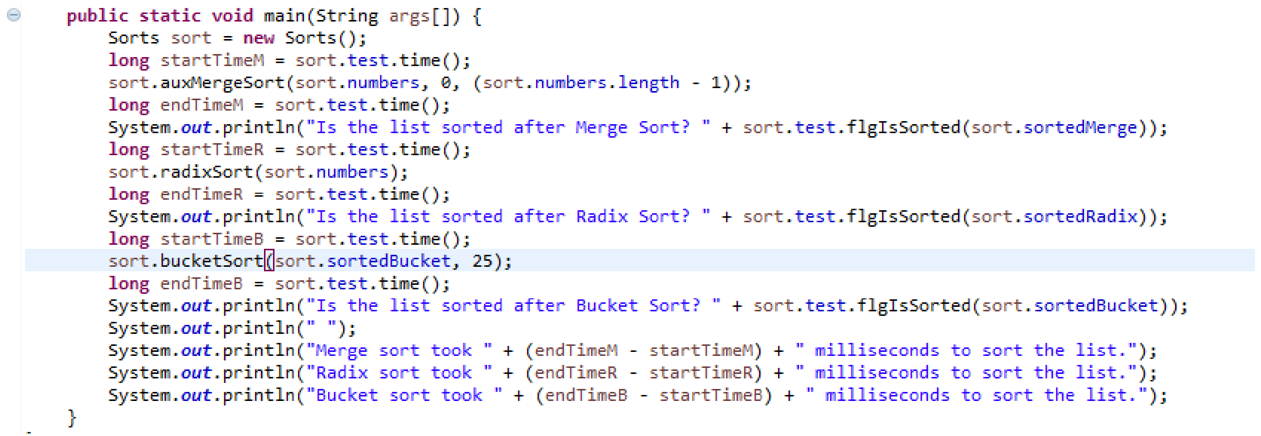
```
200100
200101
200102
200103
200104
200105
200106
200107
200108
200109
```

```
200110
200111
200112
200113
200114
200115
200116
200117
200118
200119
```

```
200120
200121
200122
200123
200124
200125
200126
200127
200128
200129
Is the list sorted? true
```

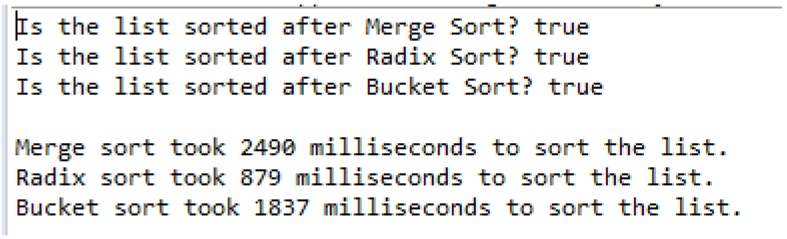
4. **Test the Times with Merge Sort:**

**Code:**



```java
public static void main(String args[]) {
    Sorts sort = new Sorts();
    long startTimeM = sort.test.time();
    sort.auxMergeSort(sort.numbers, 0, (sort.numbers.length - 1));
    long endTimeM = sort.test.time();
    System.out.println("Is the list sorted after Merge Sort? " + sort.test.flgIsSorted(sort.sortedMerge));
    long startTimeR = sort.test.time();
    sort.radixSort(sort.numbers);
    long endTimeR = sort.test.time();
    System.out.println("Is the list sorted after Radix Sort? " + sort.test.flgIsSorted(sort.sortedRadix));
    long startTimeB = sort.test.time();
    sort.bucketSort(sort.sortedBucket, 25);
    long endTimeB = sort.test.time();
    System.out.println("Is the list sorted after Bucket Sort? " + sort.test.flgIsSorted(sort.sortedBucket));
    System.out.println(" ");
    System.out.println("Merge sort took " + (endTimeM - startTimeM) + " milliseconds to sort the list.");
    System.out.println("Radix sort took " + (endTimeR - startTimeR) + " milliseconds to sort the list.");
    System.out.println("Bucket sort took " + (endTimeB - startTimeB) + " milliseconds to sort the list.");
}
```

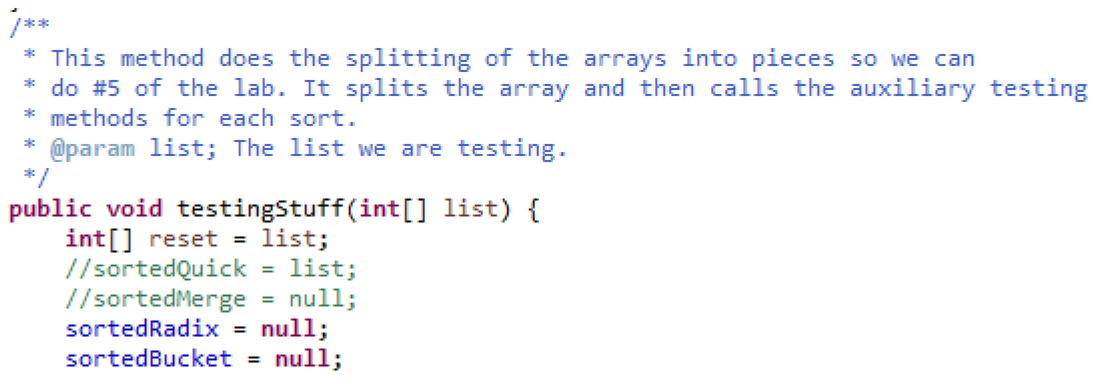
**Output:**



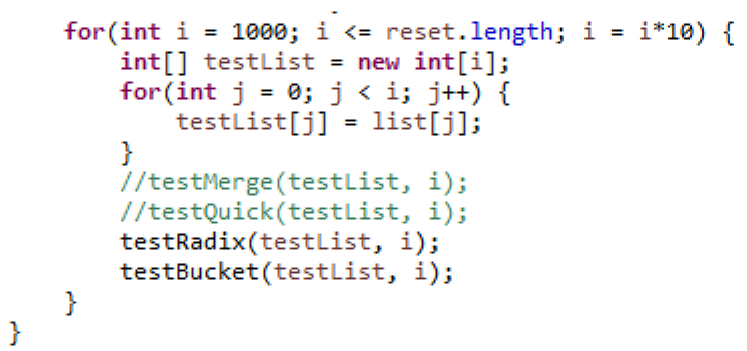
```
Is the list sorted after Merge Sort? true
Is the list sorted after Radix Sort? true
Is the list sorted after Bucket Sort? true

Merge sort took 2490 milliseconds to sort the list.
Radix sort took 879 milliseconds to sort the list.
Bucket sort took 1837 milliseconds to sort the list.
```

5. **Testing Stuff for Radix Sort and Bucket Sort**

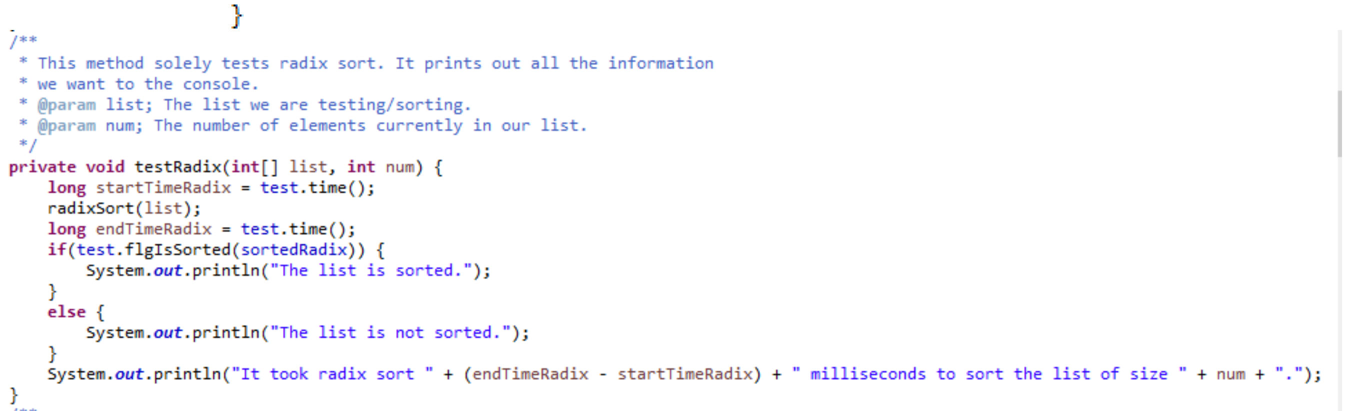
**Code:**



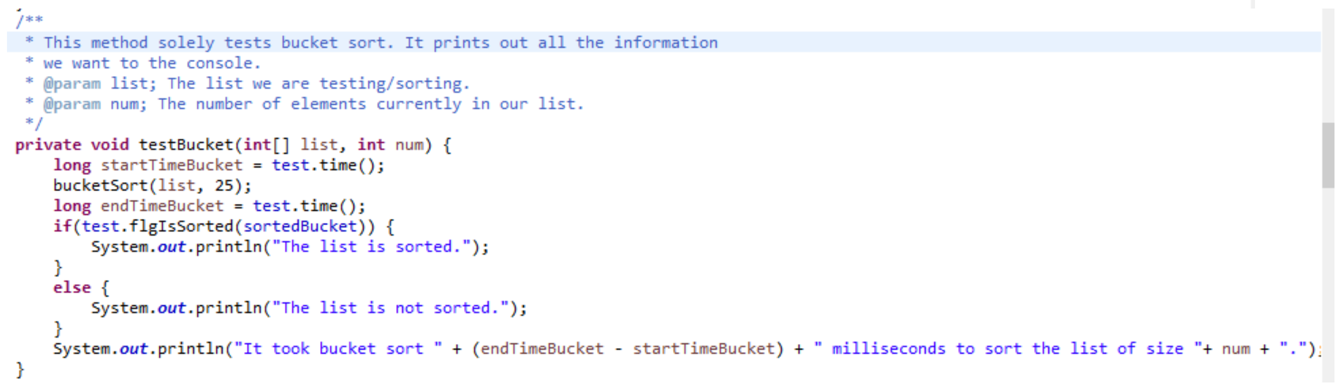
```java
/**
 * This method does the splitting of the arrays into pieces so we can
 * do #5 of the lab. It splits the array and then calls the auxiliary testing
 * methods for each sort.
 * @param list; The list we are testing.
 */
public void testingStuff(int[] list) {
    int[] reset = list;
    //sortedQuick = list;
    //sortedMerge = null;
    sortedRadix = null;
    sortedBucket = null;
```



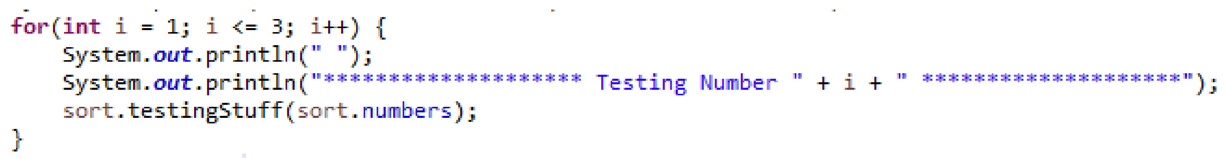
```java
        for(int i = 1000; i <= reset.length; i = i*10) {
            int[] testList = new int[i];
            for(int j = 0; j < i; j++) {
                testList[j] = list[j];
            }
            //testMerge(testList, i);
            //testQuick(testList, i);
            testRadix(testList, i);
            testBucket(testList, i);
        }
    }
```



```java
/**
 * This method solely tests radix sort. It prints out all the information
 * we want to the console.
 * @param list; The list we are testing/sorting.
 * @param num; The number of elements currently in our list.
 */
private void testRadix(int[] list, int num) {
    long startTimeRadix = test.time();
    radixSort(list);
    long endTimeRadix = test.time();
    if(test.flgIsSorted(sortedRadix)) {
        System.out.println("The list is sorted.");
    }
    else {
        System.out.println("The list is not sorted.");
    }
    System.out.println("It took radix sort " + (endTimeRadix - startTimeRadix) + " milliseconds to sort the list of size " + num + ".");
}
```

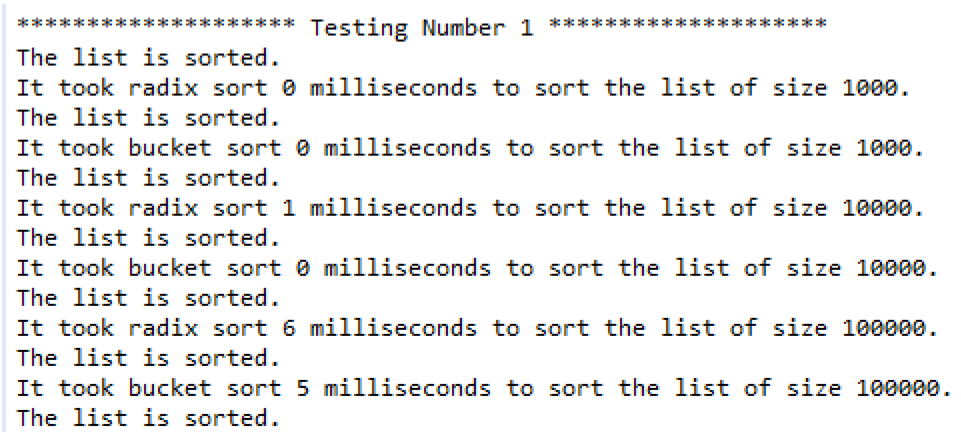


```java
/**
 * This method solely tests bucket sort. It prints out all the information
 * we want to the console.
 * @param list; The list we are testing/sorting.
 * @param num; The number of elements currently in our list.
 */
private void testBucket(int[] list, int num) {
    long startTimeBucket = test.time();
    bucketSort(list, 25);
    long endTimeBucket = test.time();
    if(test.flgIsSorted(sortedBucket)) {
        System.out.println("The list is sorted.");
    }
    else {
        System.out.println("The list is not sorted.");
    }
    System.out.println("It took bucket sort " + (endTimeBucket - startTimeBucket) + " milliseconds to sort the list of size "+ num + ".");
}
```

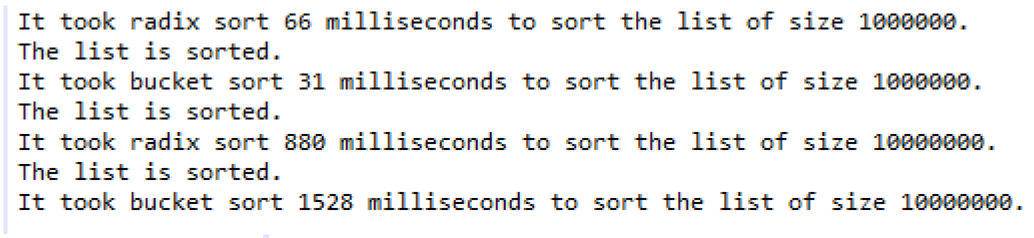


```java
for(int i = 1; i <= 3; i++) {
    System.out.println(" ");
    System.out.println("******************** Testing Number " + i + " ********************");
    sort.testingStuff(sort.numbers);
}
```

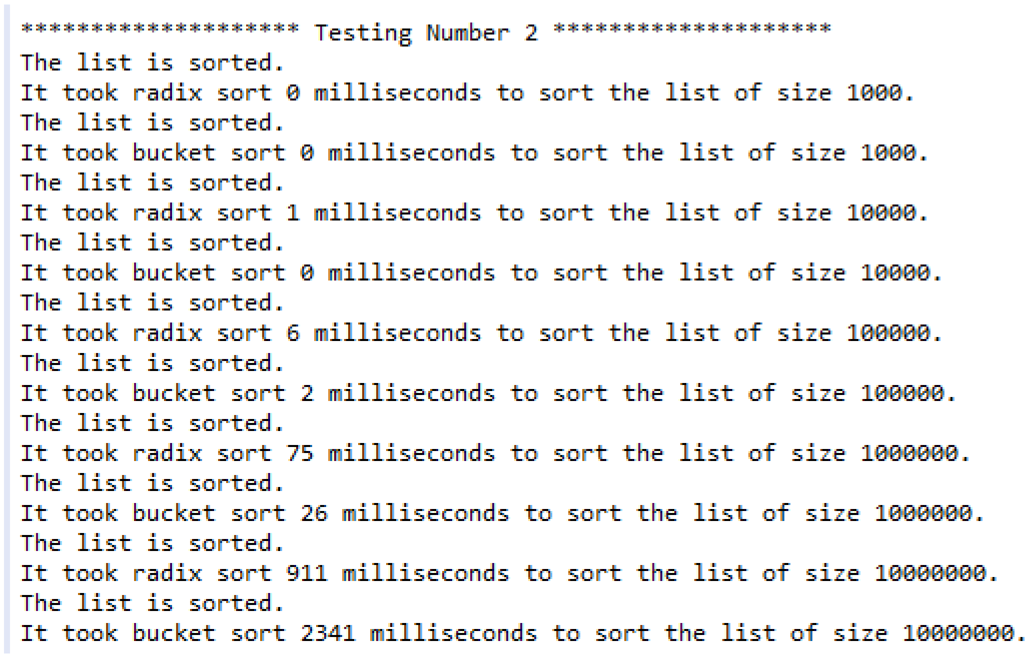
**Output:**



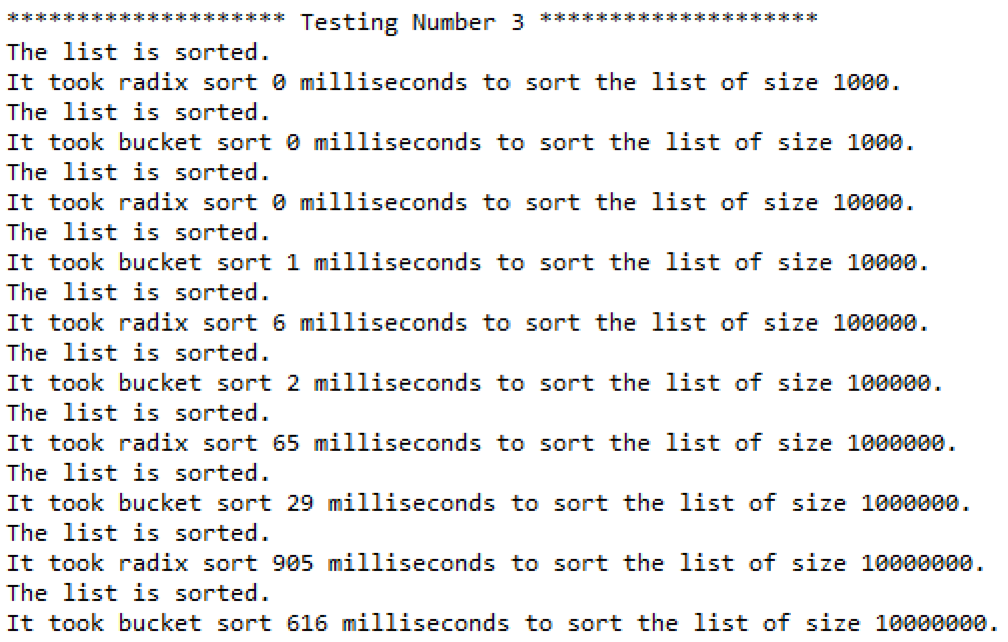
```
******************** Testing Number 1 ********************
The list is sorted.
It took radix sort 0 milliseconds to sort the list of size 1000.
The list is sorted.
It took bucket sort 0 milliseconds to sort the list of size 1000.
The list is sorted.
It took radix sort 1 milliseconds to sort the list of size 10000.
The list is sorted.
It took bucket sort 0 milliseconds to sort the list of size 10000.
The list is sorted.
It took radix sort 6 milliseconds to sort the list of size 100000.
The list is sorted.
It took bucket sort 5 milliseconds to sort the list of size 100000.
The list is sorted.
```



```
It took radix sort 66 milliseconds to sort the list of size 1000000.
The list is sorted.
It took bucket sort 31 milliseconds to sort the list of size 1000000.
The list is sorted.
It took radix sort 880 milliseconds to sort the list of size 10000000.
The list is sorted.
It took bucket sort 1528 milliseconds to sort the list of size 10000000.
```

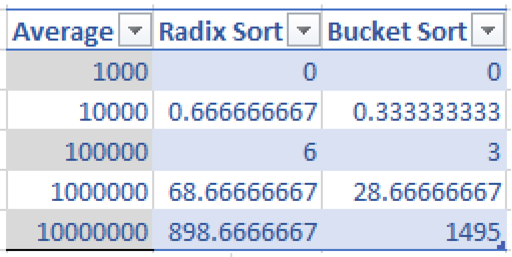


```
******************** Testing Number 2 ********************
The list is sorted.
It took radix sort 0 milliseconds to sort the list of size 1000.
The list is sorted.
It took bucket sort 0 milliseconds to sort the list of size 1000.
The list is sorted.
It took radix sort 1 milliseconds to sort the list of size 10000.
The list is sorted.
It took bucket sort 0 milliseconds to sort the list of size 10000.
The list is sorted.
It took radix sort 6 milliseconds to sort the list of size 100000.
The list is sorted.
It took bucket sort 2 milliseconds to sort the list of size 100000.
The list is sorted.
It took radix sort 75 milliseconds to sort the list of size 1000000.
The list is sorted.
It took bucket sort 26 milliseconds to sort the list of size 1000000.
The list is sorted.
It took radix sort 911 milliseconds to sort the list of size 10000000.
The list is sorted.
It took bucket sort 2341 milliseconds to sort the list of size 10000000.
```

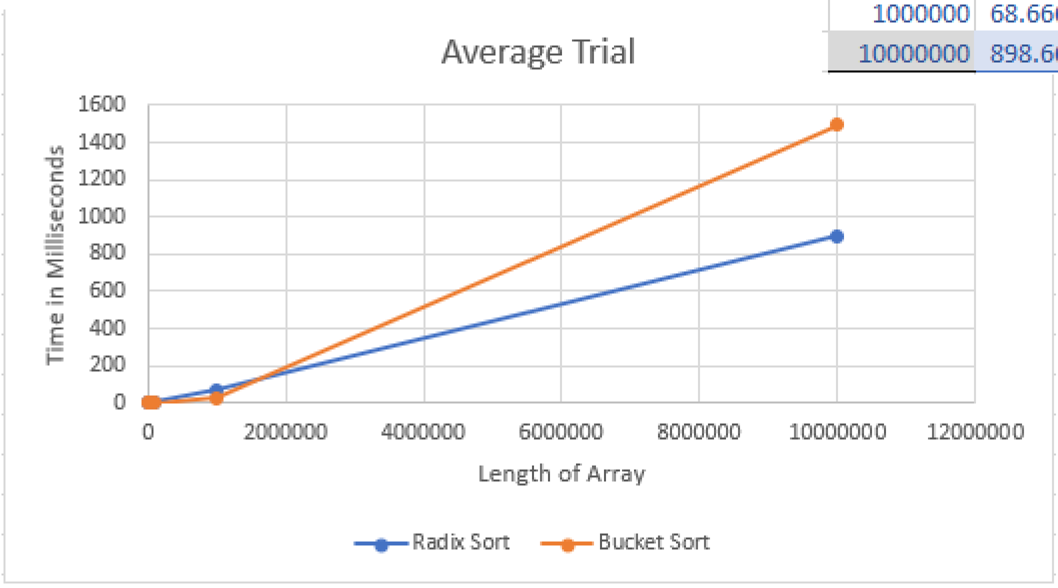


```
******************** Testing Number 3 ********************
The list is sorted.
It took radix sort 0 milliseconds to sort the list of size 1000.
The list is sorted.
It took bucket sort 0 milliseconds to sort the list of size 1000.
The list is sorted.
It took radix sort 0 milliseconds to sort the list of size 10000.
The list is sorted.
It took bucket sort 1 milliseconds to sort the list of size 10000.
The list is sorted.
It took radix sort 6 milliseconds to sort the list of size 100000.
The list is sorted.
It took bucket sort 2 milliseconds to sort the list of size 100000.
The list is sorted.
It took radix sort 65 milliseconds to sort the list of size 1000000.
The list is sorted.
It took bucket sort 29 milliseconds to sort the list of size 1000000.
The list is sorted.
It took radix sort 905 milliseconds to sort the list of size 10000000.
The list is sorted.
It took bucket sort 616 milliseconds to sort the list of size 10000000.
```

**Excel Graph and Table:**



| Average | Radix Sort | Bucket Sort |
|---|---|---|
| 1000 | 0 | 0 |
| 10000 | 0.666666667 | 0.333333333 |
| 100000 | 6 | 3 |
| 1000000 | 68.66666667 | 28.66666667 |
| 10000000 | 898.6666667 | 1495 |



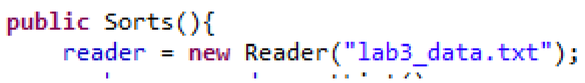
**Analysis of the Data:**

It seems like bucket sort does better than radix sort for smaller arrays, but as the arrays get longer bucket sort seems to lose.

6. **Finding the Max Ten:**
7. **Making sure it works (compared to Merge Sort):**

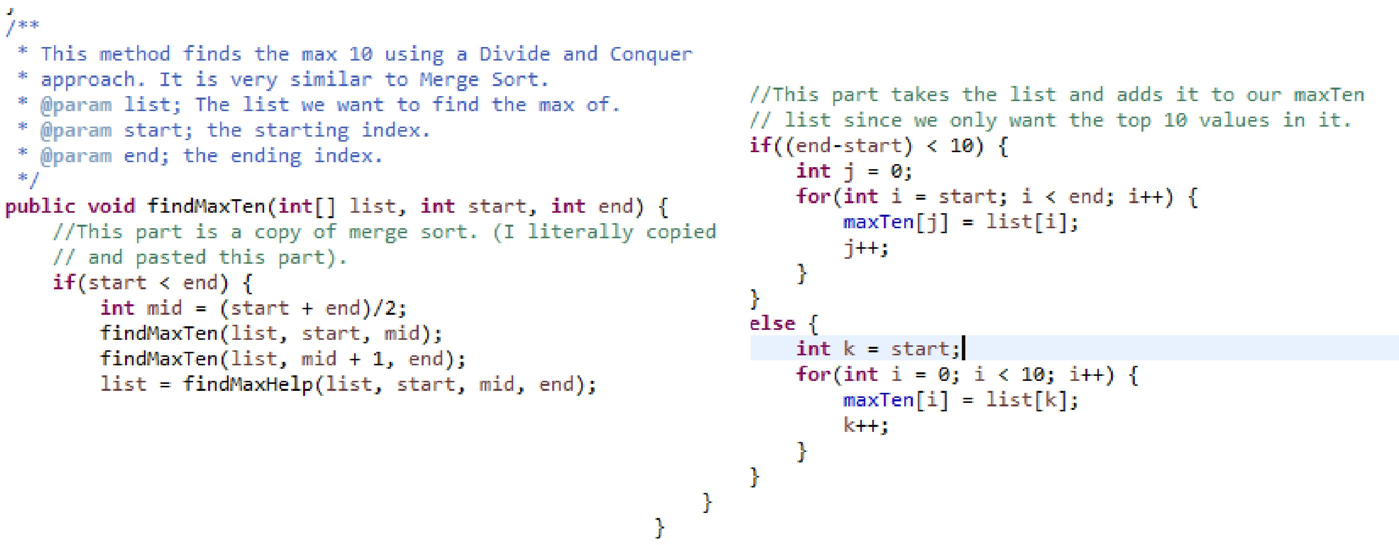
**Code:**

**Read new Data File:**



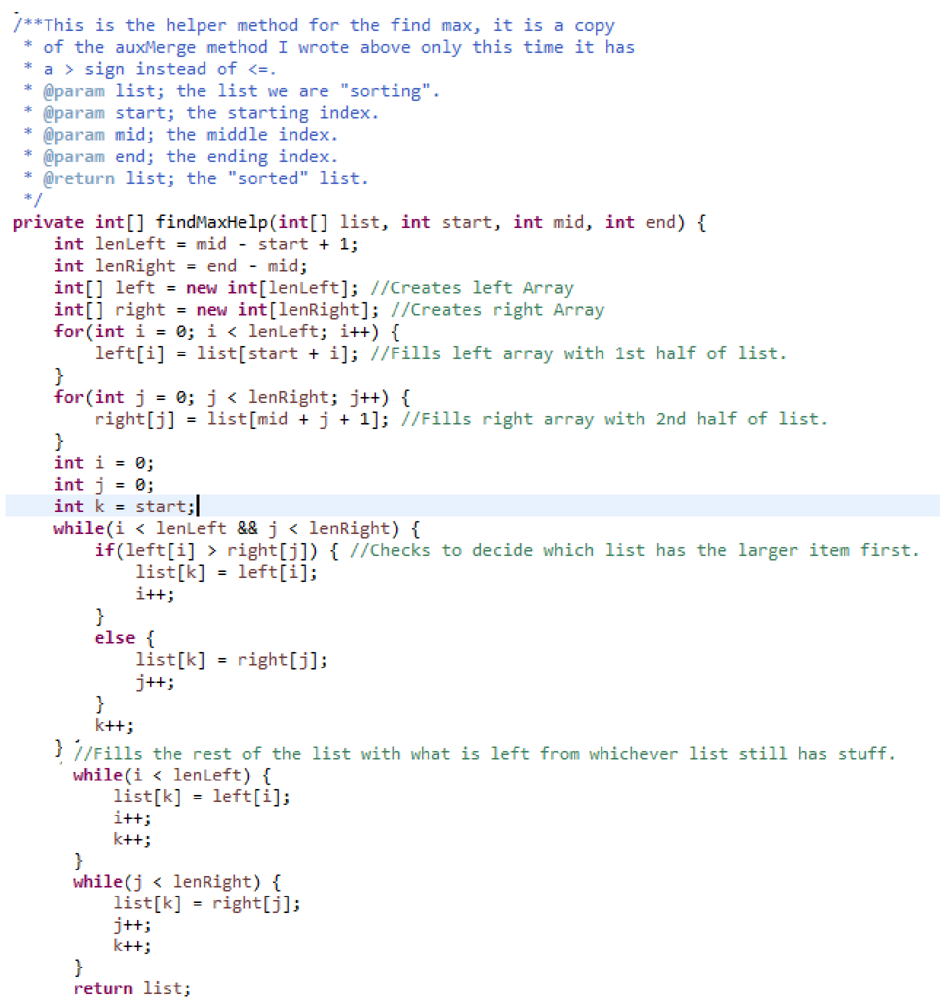
```java
public Sorts(){
    reader = new Reader("lab3_data.txt");
```

**Find Max Ten Method:**



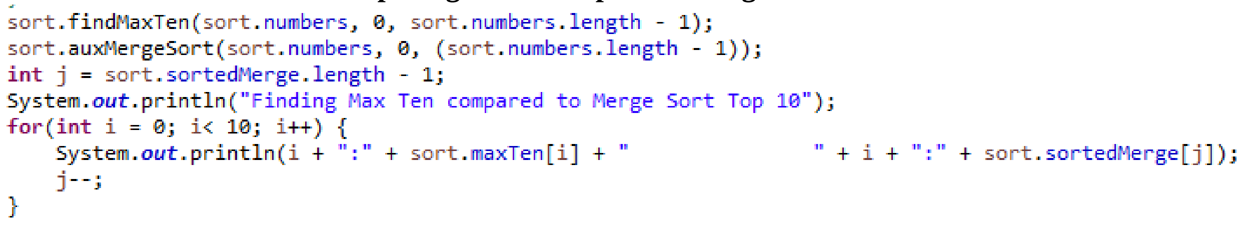
```java
/**
 * This method finds the max 10 using a Divide and Conquer
 * approach. It is very similar to Merge Sort.
 * @param list; The list we want to find the max of.
 * @param start; the starting index.
 * @param end; the ending index.
 */
public void findMaxTen(int[] list, int start, int end) {
    //This part is a copy of merge sort. (I literally copied
    // and pasted this part).
    if(start < end) {
        int mid = (start + end)/2;
        findMaxTen(list, start, mid);
        findMaxTen(list, mid + 1, end);
        list = findMaxHelp(list, start, mid, end);
```

```java
//This part takes the list and adds it to our maxTen
// list since we only want the top 10 values in it.
if((end-start) < 10) {
    int j = 0;
    for(int i = start; i < end; i++) {
        maxTen[j] = list[i];
        j++;
    }
}
else {
    int k = start;
    for(int i = 0; i < 10; i++) {
        maxTen[i] = list[k];
        k++;
    }
}
        }
    }
```



```java
/**This is the helper method for the find max, it is a copy
 * of the auxMerge method I wrote above only this time it has
 * a > sign instead of <=.
 * @param list; the list we are "sorting".
 * @param start; the starting index.
 * @param mid; the middle index.
 * @param end; the ending index.
 * @return list; the "sorted" list.
 */
private int[] findMaxHelp(int[] list, int start, int mid, int end) {
    int lenLeft = mid - start + 1;
    int lenRight = end - mid;
    int[] left = new int[lenLeft]; //Creates left Array
    int[] right = new int[lenRight]; //Creates right Array
    for(int i = 0; i < lenLeft; i++) {
        left[i] = list[start + i]; //Fills left array with 1st half of list.
    }
    for(int j = 0; j < lenRight; j++) {
        right[j] = list[mid + j + 1]; //Fills right array with 2nd half of list.
    }
    int i = 0;
    int j = 0;
    int k = start;
    while(i < lenLeft && j < lenRight) {
        if(left[i] > right[j]) { //Checks to decide which list has the larger item first.
            list[k] = left[i];
            i++;
        }
        else {
            list[k] = right[j];
            j++;
        }
        k++;
    } //Fills the rest of the list with what is left from whichever list still has stuff.
      while(i < lenLeft) {
          list[k] = left[i];
          i++;
          k++;
      }
      while(j < lenRight) {
          list[k] = right[j];
          j++;
          k++;
      }
      return list;
```

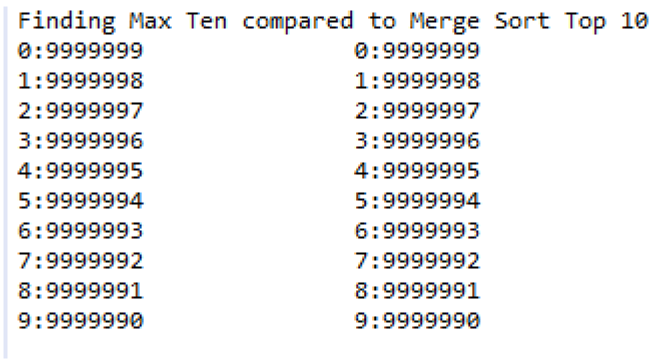
**Main Method for Comparing it to the top 10 of Merge Sort:**



```java
sort.findMaxTen(sort.numbers, 0, sort.numbers.length - 1);
sort.auxMergeSort(sort.numbers, 0, (sort.numbers.length - 1));
int j = sort.sortedMerge.length - 1;
System.out.println("Finding Max Ten compared to Merge Sort Top 10");
for(int i = 0; i< 10; i++) {
    System.out.println(i + ":" + sort.maxTen[i] + "                " + i + ":" + sort.sortedMerge[j]);
    j--;
}
```

**Output for entire list:**

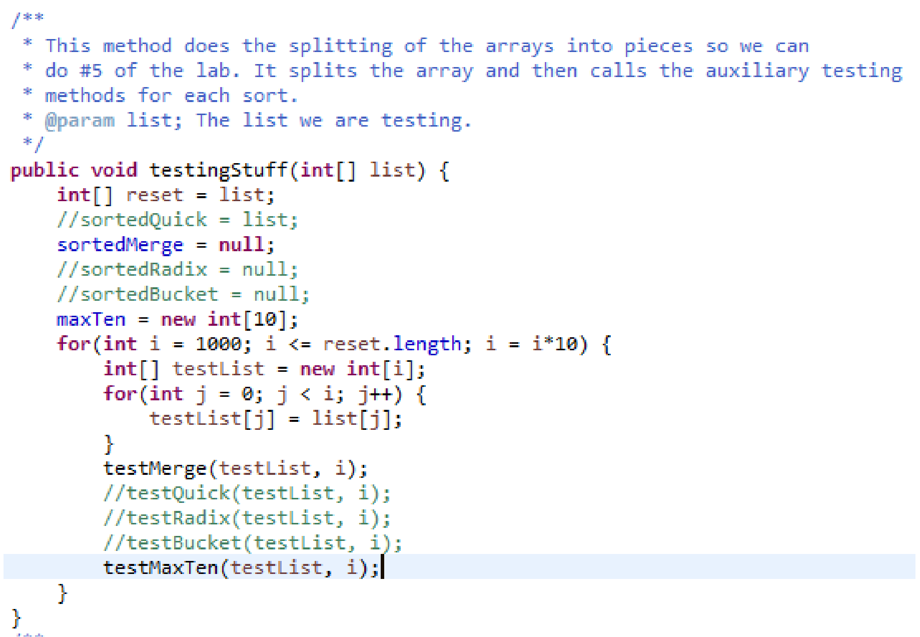
****(Not timed Yet)**



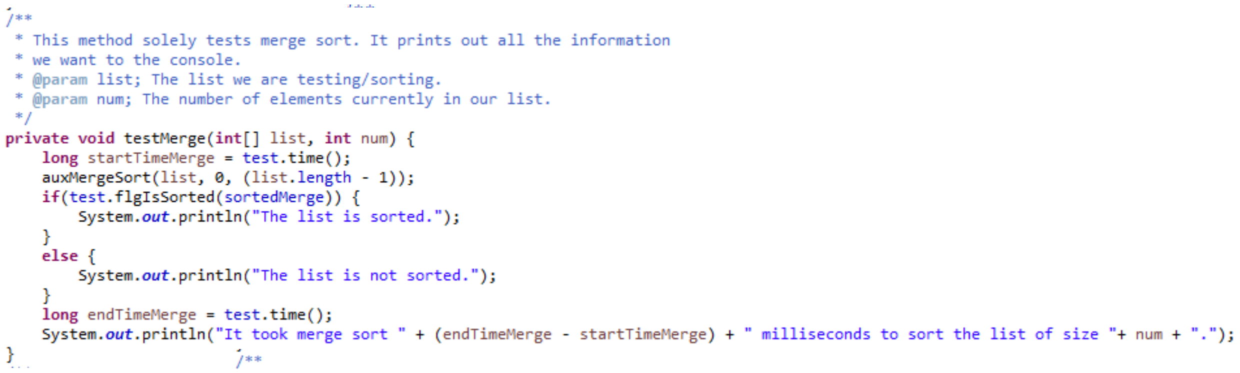
```
Finding Max Ten compared to Merge Sort Top 10
0:9999999           0:9999999
1:9999998           1:9999998
2:9999997           2:9999997
3:9999996           3:9999996
4:9999995           4:9999995
5:9999994           5:9999994
6:9999993           6:9999993
7:9999992           7:9999992
8:9999991           8:9999991
9:9999990           9:9999990
```

## 8. Test/Time the sort with finding the max.

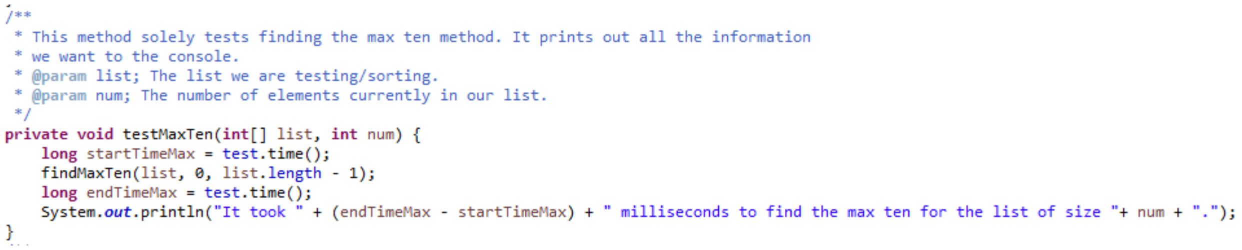
**Code:**



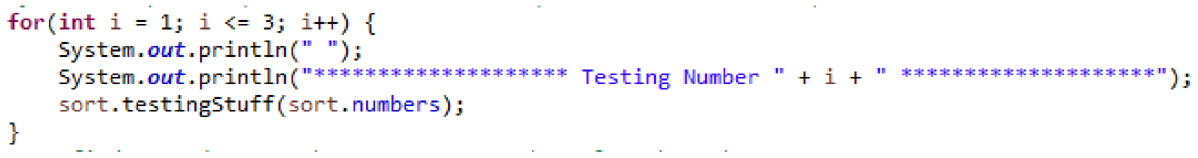
```java
/**
 * This method does the splitting of the arrays into pieces so we can
 * do #5 of the lab. It splits the array and then calls the auxiliary testing
 * methods for each sort.
 * @param list; The list we are testing.
 */
public void testingStuff(int[] list) {
    int[] reset = list;
    //sortedQuick = list;
    sortedMerge = null;
    //sortedRadix = null;
    //sortedBucket = null;
    maxTen = new int[10];
    for(int i = 1000; i <= reset.length; i = i*10) {
        int[] testList = new int[i];
        for(int j = 0; j < i; j++) {
            testList[j] = list[j];
        }
        testMerge(testList, i);
        //testQuick(testList, i);
        //testRadix(testList, i);
        //testBucket(testList, i);
        testMaxTen(testList, i);
    }
}
```



```java
/**
 * This method solely tests merge sort. It prints out all the information
 * we want to the console.
 * @param list; The list we are testing/sorting.
 * @param num; The number of elements currently in our list.
 */
private void testMerge(int[] list, int num) {
    long startTimeMerge = test.time();
    auxMergeSort(list, 0, (list.length - 1));
    if(test.flgIsSorted(sortedMerge)) {
        System.out.println("The list is sorted.");
    }
    else {
        System.out.println("The list is not sorted.");
    }
    long endTimeMerge = test.time();
    System.out.println("It took merge sort " + (endTimeMerge - startTimeMerge) + " milliseconds to sort the list of size "+ num + ".");
}
```

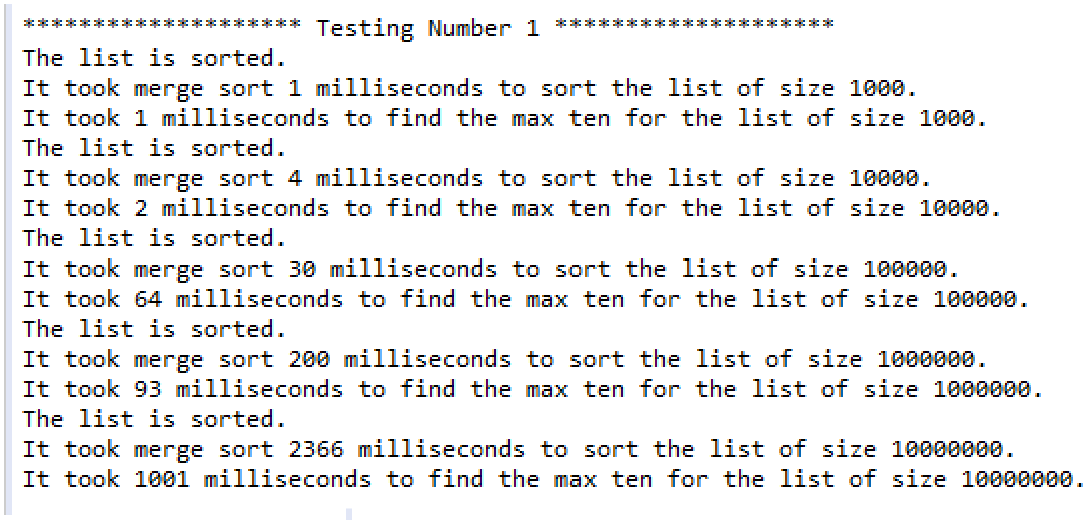


```java
/**
 * This method solely tests finding the max ten method. It prints out all the information
 * we want to the console.
 * @param list; The list we are testing/sorting.
 * @param num; The number of elements currently in our list.
 */
private void testMaxTen(int[] list, int num) {
    long startTimeMax = test.time();
    findMaxTen(list, 0, list.length - 1);
    long endTimeMax = test.time();
    System.out.println("It took " + (endTimeMax - startTimeMax) + " milliseconds to find the max ten for the list of size "+ num + ".");
}
```

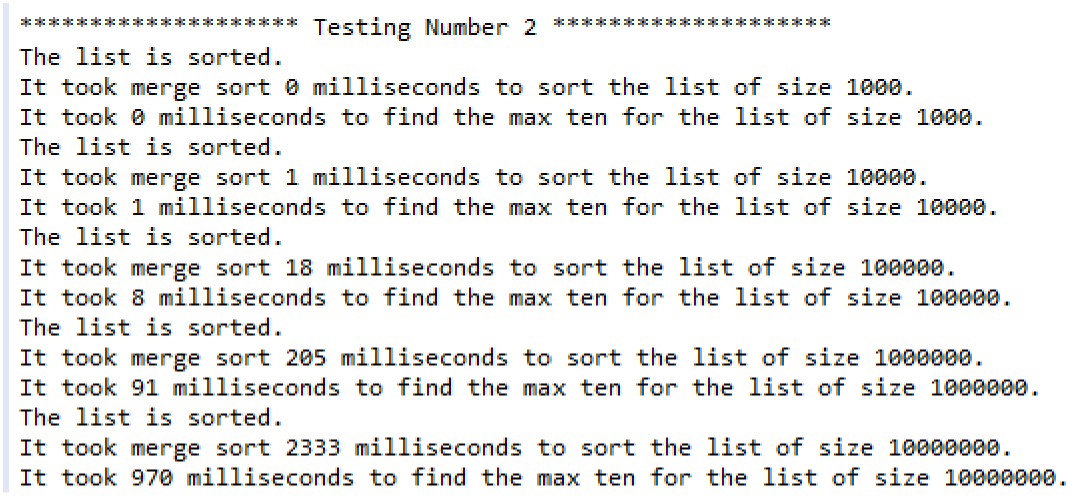


```java
for(int i = 1; i <= 3; i++) {
    System.out.println(" ");
    System.out.println("******************** Testing Number " + i + " ********************");
    sort.testingStuff(sort.numbers);
}
```

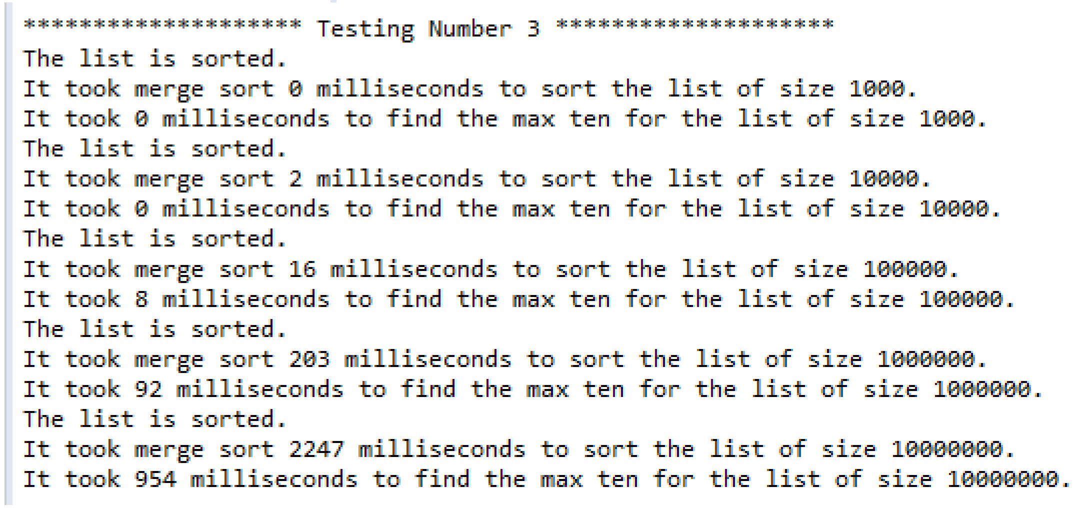
**Output:**



```
******************** Testing Number 1 ********************
The list is sorted.
It took merge sort 1 milliseconds to sort the list of size 1000.
It took 1 milliseconds to find the max ten for the list of size 1000.
The list is sorted.
It took merge sort 4 milliseconds to sort the list of size 10000.
It took 2 milliseconds to find the max ten for the list of size 10000.
The list is sorted.
It took merge sort 30 milliseconds to sort the list of size 100000.
It took 64 milliseconds to find the max ten for the list of size 100000.
The list is sorted.
It took merge sort 200 milliseconds to sort the list of size 1000000.
It took 93 milliseconds to find the max ten for the list of size 1000000.
The list is sorted.
It took merge sort 2366 milliseconds to sort the list of size 10000000.
It took 1001 milliseconds to find the max ten for the list of size 10000000.
```

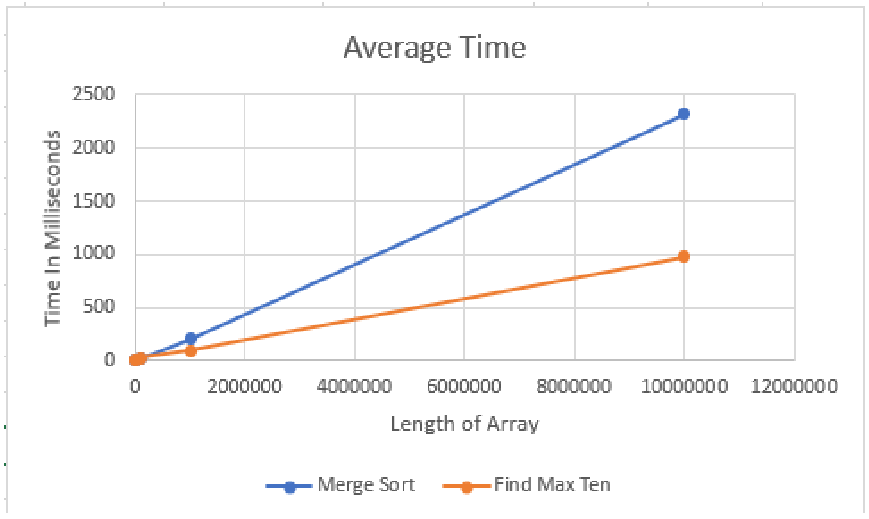


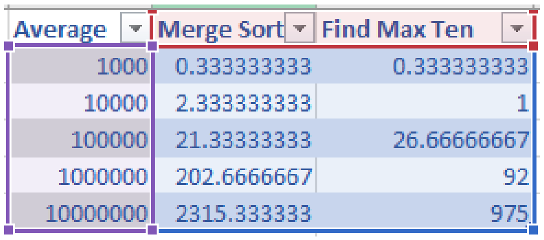
```
******************** Testing Number 2 ********************
The list is sorted.
It took merge sort 0 milliseconds to sort the list of size 1000.
It took 0 milliseconds to find the max ten for the list of size 1000.
The list is sorted.
It took merge sort 1 milliseconds to sort the list of size 10000.
It took 1 milliseconds to find the max ten for the list of size 10000.
The list is sorted.
It took merge sort 18 milliseconds to sort the list of size 100000.
It took 8 milliseconds to find the max ten for the list of size 100000.
The list is sorted.
It took merge sort 205 milliseconds to sort the list of size 1000000.
It took 91 milliseconds to find the max ten for the list of size 1000000.
The list is sorted.
It took merge sort 2333 milliseconds to sort the list of size 10000000.
It took 970 milliseconds to find the max ten for the list of size 10000000.
```



```
******************** Testing Number 3 ********************
The list is sorted.
It took merge sort 0 milliseconds to sort the list of size 1000.
It took 0 milliseconds to find the max ten for the list of size 1000.
The list is sorted.
It took merge sort 2 milliseconds to sort the list of size 10000.
It took 0 milliseconds to find the max ten for the list of size 10000.
The list is sorted.
It took merge sort 16 milliseconds to sort the list of size 100000.
It took 8 milliseconds to find the max ten for the list of size 100000.
The list is sorted.
It took merge sort 203 milliseconds to sort the list of size 1000000.
It took 92 milliseconds to find the max ten for the list of size 1000000.
The list is sorted.
It took merge sort 2247 milliseconds to sort the list of size 10000000.
It took 954 milliseconds to find the max ten for the list of size 10000000.
```

**Excel Table/Graph:**





| Average | Merge Sort | Find Max Ten |
|---|---|---|
| 1000 | 0.333333333 | 0.333333333 |
| 10000 | 2.333333333 | 1 |
| 100000 | 21.33333333 | 26.66666667 |
| 1000000 | 202.6666667 | 92 |
| 10000000 | 2315.333333 | 975 |

## 9. The Report Part:

First of all, I used a Divide-And-Conquer approach to solve the problem of finding the largest 10 numbers in the array because I thought it would be the easiest of the 3 methods that would work. I don't think a Greedy Algorithm would work unless we went through the ENTIRE list 10 times, but I wasn't sure if that counted so I changed it. I tried like 3 different ways, the first wasn't recursive and the second was to go through the entire list 10 times and find the max each time. The Divide-And-Conquer solution I came up with is VERY similar to Merge Sort where it splits the array down and then merges it back together, only this time we add an extra step of adding it to a list of length 10 with only the 10 biggest numbers in it. For some reason it was faster than just sorting it with merge sort, which I found very strange. I think this is because it was able to "ignore" the last part of the array, but I'm not sure. It seems like they should have been about the same time, but as the array got bigger the max ten method ran faster than merge sort even though they are basically the same thing. I figured it out all by myself and didn't use any website to help me. (If I had there would be comments in the code. 😊).