CSC 260.002 Programming Assignment #6
Due Date: Thursday, October 20
NOTE: assignment #7 will be due shortly after #6

In this program you will experiment with arrays. Specifically, you will fill an array with random int values, compute the trimmed mean and mode and then manipulate the array by removing any duplicate values. You will then do the same thing to a sorted version of the array. The idea here is to see how to implement some of these operations on an unsorted array versus a sorted array. Below are the specifics for how to implement the assignment.

main: Declare two arrays, unsorted and sorted, instantiating sorted to an int[30]. Declare two int variables numberUnsorted, numberSorted. main will call a method to fill the unsorted array with random values. This method should declare and instantiate a Random number generator, instantiate an int array of 30 elements, fill the array with random values in the range of 1 to 30, and return the array. You will call this from main using notation like `unsorted = fillArray();` main will then copy all 30 elements of unsorted into sorted (use a for loop assigning each `sorted[i] = unsorted[i];` Next, main will sort the sorted array using `Arrays.sort(sorted);` Note that the Arrays class is part of the java.util package. The unsorted array remains unsorted.

Next, main will call methods to obtain the trimmed mean of the unsorted array, of the sorted array, the mode of the unsorted array, and of the sorted array. Note that you should receive the same answers for trimmed mean and for mode between the two arrays. The reason you are doing this on both arrays is to see how the logic differs between doing some computations on an unsorted versus a sorted array. How to compute these values is described below. These methods should returns that you will then output as in `System.out.println("The mode is " + mode(unsorted));` You will pass the unsorted array to the two methods computing values in an unsorted fashion and the sorted array to the two methods computing values in a sorted fashion. Note that the trimmed median will be a double while the mode is an int. When outputting a double, remember that you might want to format it so your output should use printf instead of println.

main will call two additional methods to remove all elements found in the arrays. Again, there will be two methods, one for the unsorted array and one for the sorted array. These methods will directly manipulate the array passed as a parameter (recall that passing an array is handled by passing a reference so that the method can directly manipulate the original array, not a copy). But as these methods will be removing elements of the array, both methods need to return the number of elements left in the array when done as unsorted.length will not change but the number of elements that you will want to output will change. Use notation like this:

```
numberUnsorted = removeDuplicatesUnsorted(unsorted);
```

The last thing main will do is output the final versions of both arrays with the duplicates removed. This is why you need numberUnsorted and numberSorted (to control these for loops).

The trimmed mean is the average of the values having discarded the minimum and maximum. In order to compute this for the unsorted array, you will need to compute the sum of the values, locate the minimum and maximum and subtract these from the sum and then divide the sum by 28 (having discarded the min and max values). You may search for the minimum and maximum in the

trimmed mean method or in separate methods, your choice. For the sorted array, the trimmed mean is computed by summing up the values of sorted[1] through sorted[28] and dividing by 28 (because sorted[0] will be the min and sorted[29] will be the max).

The mode is the *most commonly occurring* element. It is easier to determine this in the sorted array and you should be able to use a single for loop. Iterate down the array comparing sorted[i] to sorted[i+1]. If they equal, you have another occurrences of the current value. If not, then you are shifting from one number to the next. If the number of occurrences of the latest value > number of occurrences of the previous max, then you found a value occurring more often. Let's step through how to do this for this array: 1 3 3 4 4 4 5 5 8 10 10. You start with sorted[0] as the most occurring with a max occurrence of 1 and a current occurrence of 1. i = 0. Compare sorted[0] == sorted[1]. We have a mismatch, so we are ready to move on to the next number. Our most common number is 1 which occurred 1 time. Reset current to 1 to account for sorted[1]. Compare sorted[1] to sorted[2]. We have a match so current is 2, increment i to 2. sorted[2] != sorted[3], so we are done looking at the 3's. But since current > max, we have a new most common, sorted[2]. So most common is now 3 which occurred current times (2). Now we increment i to 3 and continue. You might try to work through the logic on paper before writing the code.

To find the most common element in an unsorted array, you need to look at each element, unsorted[i], and iterate down the rest of the array to count the number of times it appears. Thus, you will need two nested for loops. The code will look something like following.

```
for(int i=0;i<30;i++)
{
        current = 1;     // we match a[i]
        for(int j=i+1;j<30;j++)
                if(a[i]==a[j]) current++;
        if(current > max) {…}
}
```

The code in the … is the code you need to remember a[i] and max as the most commonly occurring number to this point and how many times it appeared.

The removeDuplicatesUnsorted method will similarly have two nested loops. For each duplicate you find, rather than counting it, you want to remove it.

```
for(i=0;i<number;i++)
      for(j=i+1;j<number;j++)
            if(a[i]==a[j]) {
                  remove a[j]
                  number--
            }
```

To remove a[j], you will use a for loop that will shift all elements of the array j+1 through number-1 down one position to the left. For instance, array[j] = array[j+1], array[j+1] = array[j-2], etc You can implement the remove code in this method or as a separate method having been passed the array and j.

The removeDuplicatesSorted method can work the same but notice that removing a duplicate will always be side by side with the current value. You can implement this far more efficiently using a single for loop with a nested while loop as follows where n is initialized to 30.

```
for(int i=0;i<n-1;i++)
        while(a[i]==a[i+1])
        {
                remove a[i+1]
                n--;
        }
```

The same remove code will work here as you have for the removeDuplicatesUnsorted. Note though that the above logic has a logical error. The outer for loop iterates up to n-1. But n can be decremented inside the code. In order to ensure this works accurately, you have to make sure that i continues to be < n in the while loop as you decrement n.

The removeDuplicatesUnsorted and removeDuplicatesSorted methods need to return the new value of n because both methods will be removing elements in the array which you will need to output the new arrays. Note that the array will actually still have 30 elements in it, but by shifting items down, we wind up pushing values to the right. For instance, if our array was 5 8 3 5 9 3 6, removing the second 5 actually leaves the array as 5 8 3 9 3 6 6 (we shifted the 9 one to the left, we shifted the second 3 one to the left, we shifted the 6 one to the left leaving two 6's). Now we remove the second 3 and we wind up with 5 8 3 9 6 6 6. The value for number was original 7 but is now 5, so our for loops will only visit the first 5 values, 5 8 3 9 6, leaving the last two 6s from being visited by a for loop. The returned new number of elements in the arrays is used to control the last for loops to output the lists of values.

Hand in your program and one sample output. An example output is shown below (using two columns to save space).

The trimmed mean for the unsorted array is 16.54
The trimmed mean for the sorted array is 16.54
The mode for the unsorted array is 19
The mode for the sorted array is 17

Unsorted list with duplicates removed	Sorted list with duplicates removed
19	2
4	3
3	4
15	6
25	7
29	8
17	9
2	12
12	15
9	16
20	17
28	18
8	19
18	20
16	21
6	24
24	25
21	28
7	29
30	30