**Assignment 9 – Sorting**

*Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.*

1. **Let’s sort using a method not discussed in class. Suppose you have *n* data values in in array *A*. Declare an array called *Count*. Look at the value in *A[i]*. Count the number of items in *A* that are smaller than the value in *A[i]*. Assign that result to *count[i]*. Declare an output array *Output*. Assign *Output[count[i]] = A[i]*. Think about what the size of *Output* needs to be. Is it *n* or something else? Write a method to sort based on this strategy.**

Class Sorting{

//Assuming A is already provided somehow

Integer count[] = new Integer[A.length()];

For(i=0;i<A.length();i++){

Int tempValue = A[i];

Int smallerThan = 0;

For(j=0;j<A.length();j++){

If(A[j] < tempValue){

smallerThan++; //Duplicates and the i=j would not be counted

}

}

Count[i] = smallerThan;

}

Int Output[] = new Int[A.length()];

Static class insertIntoOutput(int[] output, int I, int value){

If(output[i] == null){

Output[i] = value;

} else {

insertIntoOutput(output, i+1, value)

//if there is a value there, that means that it is a duplicate. We then try the next value until there is an empty spot. There will be no infringement upon other spots, as the next highest value with have I+ number of duplicates. If there are three duplicates, they will fill (i,i+1,i+2) and the none duplicate value will be in i+3.

}

Return output;

}

For(i=0;i< A.length(); i++){

insertIntoOutput(Output, count[i], A[i]);

//Calling this will insert into Output the index of the value at count[i] the value of A[i]

}

}

The value of Output should be the same as the original input value, since this is a sort, and there are no insertions or deletions. Every A[i] will have a corresponding count[i] and a place in Output.

1. **Analyze the cost of the sort you wrote in the previous problem. What is the impact of random, ordered, or reverse ordered data?**

The cost of the initial counting and comparison is n^2, or it could be reduced to n(n-1). This is because each value has to be compared with each other value to find the count of items less than it. There might be ways to reduce this a bit further, but the overall cost should be O(n^2).

The cost of placing the values into the correct positon is usually a linear cost. For each A[i], there is only one corresponding value where it could go. However, the worst case scenario is that everything is a duplicate. This means that the count[] would be 0 for every index. Based on the way that I have written it, This means that the recursion will be run once the first time, twice the second time, n times the nth time to place the duplicate. This will become an operation of (n(n+1))/2. So worst case, this formula would be n^2-n+1/2\*n^2+1/2\*n = 3/2\*n^2-1/2\*n, or simply O(n^2).

This particular implementation is not sensitive to the order of the data. Each comparison will be made regardless of the data, as no optimization techniques are in place. This means that no matter if it is in reverse order or ordered or random, the number of comparisons will be the same. However, if it is known beforehand what sort of data will be used, we could modify the program to stop a comparison once it reaches a certain point. But since we do not, we have to program it in a way in which order would not matter.

1. **How many comparisons are necessary to find the largest and smallest of a set of n distinct elements?**  
   The best comparison method would take 3/2\*n number of comparisons.

This is done by comparing i and i+1 as a pair. The smaller of the pair is compared against a min value that is stored, and the larger of the value is compared against a max value that is stored. Min and max are initialized to inf and -inf, respectively. If the number of pairs is odd, then the first value is stored as the min and the max, and the first comparison is made against that. This means that we loop through the value n/2 times. Each time we loop through, we make three comparisons: one for A[i] < A[i+1], then one for A[i] < min, and one for A[i+1] > max. (Assuming that A[i] < A[i+1] returns true).