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.**

```
        def countSort(arrayToSort)
                x=0;
                count = []
                output = []
                for (i = 0; i < arrayToSort.length; i++) {
                        # counter is going to represent the number of items small than A[i]
                        counter = 0;
                        for (j = 0; j < arrayToSort.length; j++) {
                                x++;
                                # so the right A[i] is the element we are using as a benchmark. on
the left A[j] is the element we are comparing A[i] against to see if it's less than A[i]
                                if (A[j] < A[i]):
                                        counter = counter + 1;
                        }
                        count[i] = counter;

                        # the index which is count[i] is where A[i] is put in the output array
                        output[count[i]] = A[i];
                }
```

2. **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 for the above problem is O(n^2). We have two nested loops: one outer loop to go through every element of the input array and one inner loop to check how many elements in the input array are smaller than the focused element in the outer loop.

Random, ordered and reverse ordered data will also give a time complexity of O(n^2). I think regardless of order (random, ordered, reverse), we still need two loops: one outer loop to go through every element in the input array and one inner loop to find how many elements in the array are smaller than the element from the outer loop.

To grader: Is that not right? Does order count?

3. **How many comparisons are necessary to find the largest and smallest of a set of n distinct elements? Do not assume the answer must involve sorting. It could but does not need to do so. Try to be as efficient as you can.**

For Grader: This was pretty tricky for me so if there's a better way, please lmk in the grading.

Here are two intuitive ways to think about this:

1. I think one intuitive way to think about this is as follows:
For every 2 elements in the array of numbers, we need to do these 3 steps:
1. Pick 2 elements(a, b), compare them. (say a > b)
2. Update min by comparing (min, b)
3. Update max by comparing (max, a)

That's 3 steps for every 2 elements so N*3/2

2. Here's another way that's a bit more fleshed out:
Say we have three elements a,b,c. If we make two comparisons to infer that

$a < b$ and $a < c$

we have excluded two elements from the potential minimums (b and c), but only one from the potential maximums (a). So it takes 2 comparisons to reduce the potential minimums and maximums by <u>at least</u> one.

So given this, let's split the input array into pairs and compare each pair. So if we have an input array of [1,2,3,4,5,6,7,8] we now have [[1,2], [3,4], [5,6], [7,8]]. After n/2 comparisons, we have reduced the potential minimums and potential maximums to n/2 each. The potenetial minimums are now in one array, the potential maximums in another. So we have one array of [1,3,5,7] and another array of [2,4,6,8]. Furthermore, those two sets are disjoint so now we have two problems, one minimum and one maximum, each of size n/2. So,

n/2 = number of comparisons once the input array is split into pairs.
n/2-1 = number of comparisons to determine the min and max in each of the two split up arrays. We have two arrays (one of potential mins and another of potential maxes) remember.

The total number of comparisons is:

$n/2 + 2(n/2 - 1) = n/2 + n - 2 =$ **<u>$3n/2 - 2$</u>**