Ashlyn Cooper
CPSC 3120 HW 2

Question 1

a) If it is the same number from left to right, it is in the form *abccba*. If the number is a multiple of 9, then the sum of its digits should also be a multiple of 9 or be 9. When we remove the first and last digits, we get some number in the form *bccb.* This new number has only a prime factor of 11, which means that *bccb* is some power of 11.
    a. If we enumerate through the powers of 11, $11^3 = 1331$. This is the only number that is 4 digits and satisfies the condition that the number is in the form *bccb*.
    b. To find the remaining two digits, we know that the two digits follow the rule that a + 1 + 3 + 3 + 1 + a = 9 or a + 1 + 3 + 3 + 1 + a= a multiple. They must also be the same number to maintain the form *abccba*.
        i. First we will try to find a number that sums to 9:
            1. a + 1 + 3 + 3 + 1 + a = 9
            2. 2a + 8 = 9
            3. A = ½, which is not possible in this problem.
        ii. So we move on to find if it adds to the next multiple of 9, which is 18.
            1. a + 1 + 3 + 3 + 1 + a = 18
            2. 2a + 8 = 18
            3. 2a = 10
            4. a = 5
            5. This satisfies all of our set conditions
    c. Answer: Hannah's 6-digit secret number is 513315.

b) There are 6 ways that 15 can be written as the sum of 4 counting numbers. Enumerated those ways are:
    a. 1 + 2 + 3 + 9
    b. 1 + 2 + 4 + 8
    c. 1 + 2 + 5 + 7
    d. 1 + 3 + 4 + 7
    e. 1 + 3 + 5 + 6
    f. 2 + 3 + 4 + 6

Question 2

For the first method we will do a simple-minded approach. In this approach we will traverse through the array from start to finish and for each element count the number of elements smaller than the current number up to the current index. We will then sum up the count of inversion for each index to get the answer. The **complexity analysis is O(n^2)** as we will use two nested loops going through the array from start to finish.

The pseudocode is as follows:

```
let count = 0
let size = size of the array

For elements in array index i=0 to size-1{
     For all other elements in array from index j=i+1 to size {
          If element i > element j, increment count
     }
}

Return count
```

For a more efficient algorithm, we will use an enhanced merge sort method. We will divide the array into two equal halves at each step. We will then use a merge function that will count the number of inversions when the two halves of the array are merged. We will create two indices in this merge function, i and j, with i being in the first half and j being in the second half. If array[i] is > array[j], then there are a number of inversions equal to mid-1. We will then recursively continue to divide the array into halves and find the answer by summing the number of inversions in the first half, second half, and the number when we merge the two. The base case will be when there in only 1 element in the given half. The **complexity of this algorithm is O(n log n)** because we are using a divide and conquer method.

The pseudocode is as follows:

For the algorithm to work we will need to use 2 functions. The first function is a recursive merge sort function that sorts the input array and returns the number of inversions in the array.

The second function merges two sorted arrays and returns the inversion count in those arrays.

```
mergeSort{

     let count = 0;
     declare mid
     if (right item . left item){
          divide the array in 2 parts, set mid = (right+left)/2

          increment count and call mergeSort on the left half
          increment count and call mergeSort on the right half

          increment count and call merge on the two parts of the
          array
     }

     Return count
}
```

```
Merge{

     Count = 0
     Set indices for each subarray
     i is the right left subarray, j is the midpoint, k is the
index for the merged subarrary (same as left to start)

     while i <= midpoint and j <= right subarray {
          if value at i <= value at j {
               set a temp array index k++ = i++
          {
          Otherwise{
               set a temp array index k++ = j++

               count = count + (middle point - 1)
          }
     }

     Copy remaining elements from left subarray to temp array

     Copy remaining elements from right subarray to temp array

     Copy the merged elements in temp to the original array

     Return count (the number of inversions)

}
```