Conor Roberts #1056167

1. Q1
   1.1. Brute force
      1.1.1. **Design:** Approaching this problem logically, and inefficiently, the most simple method of brute forcing is to check for all inversions of each index of our data array. To accomplish this, I will construct a nested for loop. The outer layer will keep track of the element from which I will be comparing others and the inner layer will be the element to compare that to. The inner layer will iterate *n* times for each iteration of the outer layer. Thus, my algorithm will have an efficiency class of $\theta \in n^2$
      1.1.2. Pseudocode

```
ALGORITHM bruteInversionCount
//Computes the number of inversions of an array of integers A[0,n-1]
i<-n-1
inversions<-0
while i>=0
    j<-i+1
    while j<n
        if A[i]>A[j]
            inversions<-inversions+1
        j<-j+1
    i<-i-1
```

   1.2. Divide and conquer: My algorithm go brr
      1.2.1. This algorithm is $\theta \in nlogn$
      1.2.2. Pseudocode

```
ALGORITHM divideInversionCount
// Performs mergesort but also counts the amount of inversions
// Takes in an array of integers A[0..n-1]
// I'm not going to use the floor thing because you know that I didn't need it in C

if n<=1
    end

copy A[0..(n/2)-1] to B[0..(n/2)-1]
copy A[(n/2)..n-1] to C[0..(n/2)-1]
divideInversionCount(B)
divideInversionCount(C)
merge(A,B,C)

ALGORITHM (A[0..An-1],B[0..Bn-1],C[0..Cn-1])
a<-0, b<-0, k<-0
while b<Bn and c<Cn do
    if B[b] <= C[c]
        A[a]<-B[b]
        i<-i+1
    else
        A[a]=C[c]
        b<-b+1
        add up inversions here
    a<-a+1

if B=Bn
    copy C[c..Cn-1] to A[a..An-1]
else
    copy B[b..Bn-1] to A[a..An-1]
```

2. Q2
    2.1. Brute force
        2.1.1. Efficiency class of $\theta \in n^3$

    2.2. Divide and conquer
        2.2.1. Efficiency $\theta \in (nlogn)$