CS 325
Group Assignment 3
Ujjval Kumaria, Vijay Vardhan Tadimeti, Aashwin Vats

1. Pseudocode for each of the two methods.

Pseudocode for brute force approach $\Theta(n^2)$.

```
FUNCTION bruteforce_count(A):
      initialize count to 0
      for i in range(0,len(A)):
        for j in range(i,len(A)):
           IF A[i]>A[j]:
              increment count
         RETURN count
```

Pseudocode for divide & conquer approach $\Theta(nlogn)$.

```
FUNCTION mergecount_inversion(lst):
      IF len(lst) <= 1:
          RETURN lst, 0
      middle <- int( len(lst) / 2 )
      left, a <- mergecount_inversion(lst[:middle])
      right, b <- mergecount_inversion(lst[middle:])
      res, c <- merge(left, right)
      RETURN res, (a + b + c)

FUNCTION merge(left, right):
      res <- []
      count <- 0
      i, j <- 0, 0
      left_len <- len(left)
      WHILE i < left_len AND j < len(right):
            IF left[i] <= right[j]:
                  res.append(left[i])
                  i += 1
            ELSE:
                  res.append(right[j])
                  count += left_len - i
```

```
            j += 1
        res += left[i:]
        res += right[j:]
    RETURN res, count
```

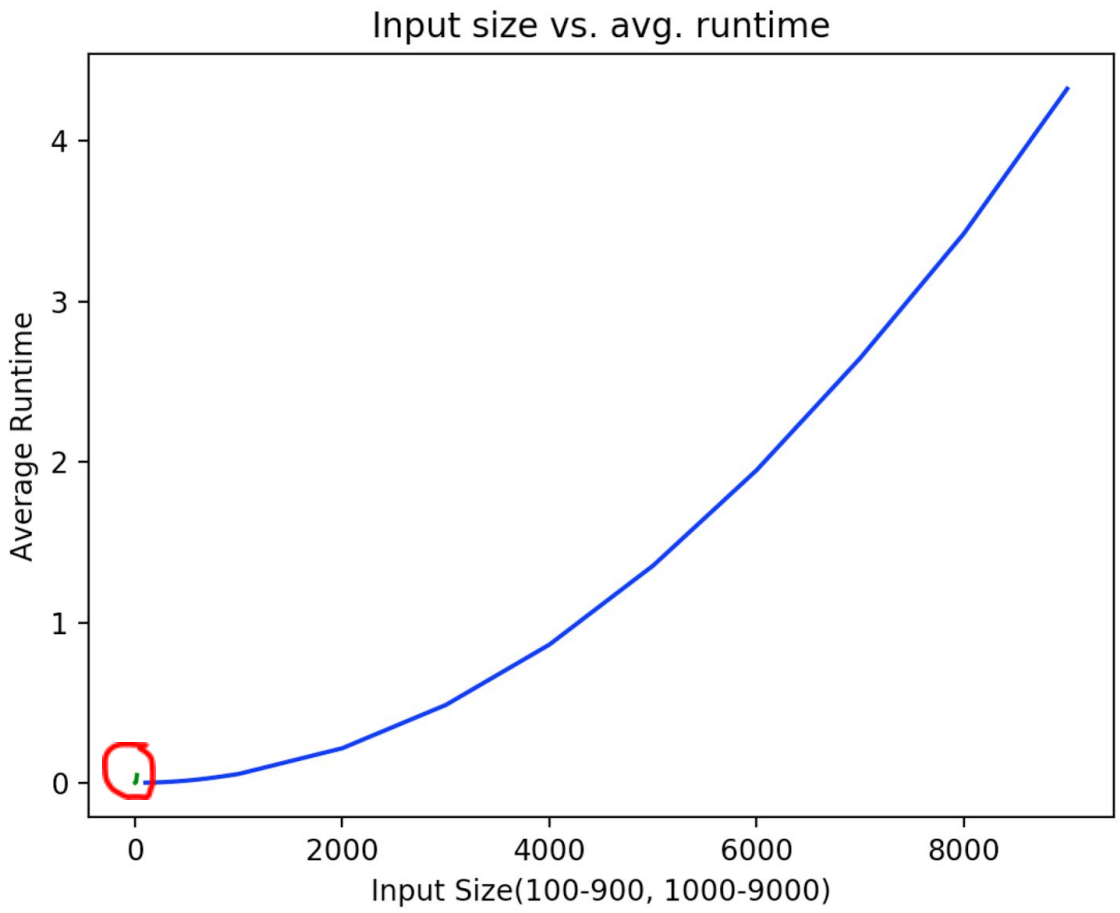
2. Recurrence relation for divide-and-conquer approach:

Let's assume that n is a power of 2, i.e. n = 2k for some k
Now, running time can be analysed by using a recurrence relation. Each "divide" step yields two sub-problems of size n/2. The "merge" part clearly takes O(n) time as it is simply a while loop as you can see above. Hence, following will be the recurrence relation for this approach:
$T(n) = 2T(n/2) + \Theta(n)$
And the solution to this recurrence would be $\Theta(n\log n)$.

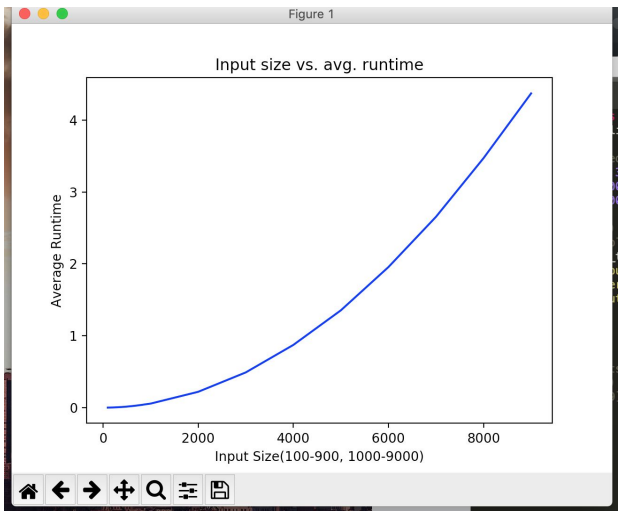
3. Refer python file: In the main function, the first two function calls will get results for this part of the question.

4. Refer python file: In the main function, the third function call will get results for this part of the question.
The python script will generate a single plot to display the input size versus average runtime both these approaches took. The runtime of brute-force for these input sizes was observed between 0-4 seconds whereas the avg. runtime of divide and conquer approach was observed between 0-0.04 seconds. Hence, plotting both in a single plot resulted in a much scaled down version of divide and conquer approach(the green dot highlighted in red circle) as compared to the brute-force approach. Brute-force is much slower than the divide-and-conquer.
Brute force = Theta(n^2), whereas divide-and-conquer = Theta(nlogn).

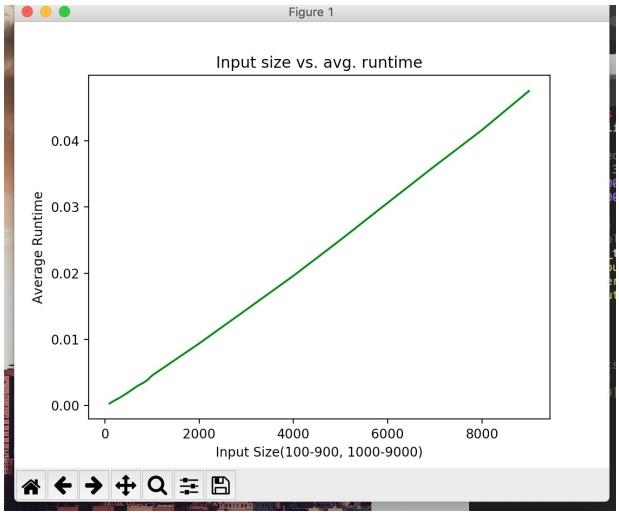
Following is the graph that the python script will generate:



We also plotted these graphs separately to compare them and following are the results:



Brute-force approach



Divide & Conquer approach