Problema 1.1

Per il primo problema viene richiesto di utilizzare un approccio divide et impera, dove un modo alternativo di vedere il problema è in termini di inversioni. Considerando questi due caratteristiche la scelta dell'algoritmo di ordinamento è ricaduta su quello del **MergeSort,** che come da definizione risulta essere un algoritmo divide et impera. Inoltre, anche se non effettua un vero e proprio scambio, la traccia suggerisce di considerare come inversioni tutte le volte che i < j e $A_i > A_j$ per cui risulta essere un algoritmo che ben si adegua al nostro caso. Di conseguenza si conteggiano solo i valori che verificano questa condizione e questo viene effettuato con l'espressione: **count += m-i+1**, dove m-i+1 rappresenta il numero di elementi rimanenti nel primo sottovettore che sono maggiori dell'elemento corrente arr[j] nel secondo sottovettore.

Di seguito vengono riportati 3 casi di test alternativi a quelli riportati nell'esempio della traccia:

Sample Input

4
6
2
9
5
2
7
4
5
4
3
2
1
8
0

Sample Output

3
1
6

Di seguito viene riportata l'analisi di complessità:

Complessità nel caso peggiore: O(nlogn)
Complessità nel caso medio: O(nlogn)
Complessità nel caso migliore: O(nlogn)

La complessità tuttavia varia in base al numero di testcase poiché il MergeSort viene invocato più volte, in quel caso la complessità diventa O(N*nlogn) dove N grande è il numero di testcase e n il numero di numeri da ordinare.

Elaborato svolto da:
Marco Dell'Isola
Raffaele Cuzzaniti