a. (2, 1)(3, 1)(8, 6)(8, 1)(6, 1)

b. The array ⟨n, $n - 1$, . . . 2, 1⟩ has the most inversions.It has $n(n - 1)/2$ of them.

c. The running time of insertion-sort is Θ(n+i), where i is the number of inversions.

d. We need to create two procedures:

1. MERGE-COUNT, which accepts the same arguments as MERGE.
2. MERGE-SORT-COUNT, which accepts the same arguments as MERGE-SORT.

The procedures are defined as follows.

MERGE-COUNT(A,p,q,r)

```
n_L = q − p + 1
n_R = r − q
let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
for i = 0 to n_L − 1
    L[i] = A[p + i]
for j = 0 to n_R − 1
    R[j] = A[q + j + i]
i = 0
j = 0
k = p
inversions = 0 // Added part
while i < n_L and j < n_R
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
        inversions = inversions + n_L − i // Added part
    k = k + 1
```

```
while i < n_L
    i = i + 1
    k = k + 1
while j < n_R
    j = j + 1
    k = k + 1
return inversions // Added part
```

MERGE-SORT-COUNT(*A, p, r*)

```
if p ≥ r
    return 0
q = ⌊(p + r)/2⌋
inversions = 0
inversions = inversions+ MERGE-SORT-COUNT(A, p, q)
inversions = inversions+ MERGE-SORT-COUNT(A, q+1, r) // Counting inversions within the two parts.
inversions = inversions+ MERGE-COUNT(A, p, q, r) // Counting inversions between them.
return inversions
```

We then simply make a copy *C* of the array we want to sort, and run MERGE-SORT-COUNT(*C*, 1, *n*) to get our answer.