Zijian Gao
26319560
2/11/2016
CS311 HW1

2-4.)

a. (1,4), (0,5), (2,3), (2,4), (3,4)

b. The array A[n..1] has the most inversions. It has $\sum_{i=1}^{n-1} i = \frac{1}{2}(n-1)n$ inversions. This is because every element at position i has n - i - 1 inversions because for every element at index j > i, a[j] < a[i], which is the definition of inversion. Or in other words, moving from the tail of the list to the head, the

c. If an input list A[1..n] has j inversions, the runtime of insertion sort is increased by j, resulting in a worst case run time of O(n^2), which would be the case where the input list is reversed resulting in a maximum number of inversions. This is because insertion sort moves each list element to its correct location by comparing with the element at the previous index position, starting with index 1. If the previous element, which has the lesser index position, has a greater value, the two values are swapped, increasing the runtime by the total amount of swaps that need to be done. This is the definition of an inversion. Each swap reduces the number of inversions by 1, and so j swaps must be done for j inversions, for a maximum of $\frac{1}{2}(n^2 - n)$ swaps, resulting in the runtime

$$T(n) = \frac{1}{2}(n^2 - n) + n = \frac{n^2 + n}{2} = O(n^2)$$

d.

```
Merge(A, p, q, r):
n_1 = q - p + 1
n_2 = r - q
let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
for i = 1 to n_1
    L[i] = A[p + i - 1]
for j = 1 to n_2
    R[j] = A[q + j]
L[n_1 + 1] = infinity
R[n_2 + 1] = infinity
i = 1
j = 1
inversions = 0
for k = p to r
    if L[i] <= R[j]
        A[k] = L[i]
        i = i + 1
    else
        inversions = inversions + r - i + 1
        j = j + 1
return inversions
```

```
count_inversions(A, p, r):
if p < r
    q = [(p + r) / 2]
    return count_inversions(A, p q) + count_inversions(a, q + 1, r) + Merge(a, p, q, r)
else
    return 0
```