Question 1:

A = [31, 41, 59, 26, 41, 58]
(Note: The sorted part will be colored red and arrays are counted from 1)

1st iteration: i = 2,
A = [31, 41, 59, 26, 41, 58]
Number of iterations executed: 0
More precisely: only one comparison operation (1 > 0 && 31 > 41), which returns False and breaks the **while** loop.

2nd iteration: i = 3,
A = [31, 41, 59, 26, 41, 58]
Number of iterations executed: 0
More precisely: (2 > 0 && 41 > 59) => False, breaks **while** loop

3rd iteration: i = 4,
A = [26, 31, 41, 59, 41, 58]
Number of iterations executed: 3
More precisely: (3 > 0 && 59 > 26) => True
(2 > 0 && 41 > 26) => True
(1 > 0 && 31 > 26) => True
(0 > 0 && A[0] > 26) => False, breaks **while** loop

4th iteration: i = 5,
A = [26, 31, 41, 41, 59, 58]
Number of iterations executed: 1
More precisely: (4 > 0 && 59 > 41) => True
(3 > 0 && 41 > 41) => False, breaks **while** loop

5th iteration: i = 6,
A = [26, 31, 41, 41, 58, 59]
Number of iterations executed: 1
More precisely: (5 > 0 && 59 > 58) => True
(4 > 0 && 41 > 58) => False, breaks **while** loop

Question 2:

A = [4, 3, 2, 1]
1st iteration: A = [3, 4, 2, 1], number of basic operations: 2
2nd iteration: A = [2. 3. 4. 1], number of basic operations: 3
3rd iteration: A = [1, 2, 3, 4], number of basic operations: 4
=> Total: 2 + 3 + 4 = 9 basic operations.

B = [1, 2, 3, 4]
1st iteration: A = [1, 2, 3, 4], number of basic operations: 1
2nd iteration: A = [1, 2, 3, 4], number of basic operations: 1
3rd iteration: A = [1, 2, 3, 4], number of basic operations: 1
=> Total: 1 + 1 + 1 = 3 basic operations.

C = [3, 1, 4, 2]
1st iteration: A = [1, 3, 4, 2], number of basic operations: 2
2nd iteration: A = [1, 3, 4, 2], number of basic operations: 1
3rd iteration: A = [1, 2, 3, 4], number of basic operations: 3
=> Total: 2 + 1 + 3 = 6 basic operations.

The detailed solution for each operation should be similar to that of question 1. Note that the number of basic operations (the number of comparison operations in the **while** clause) is different from the number of iterations (the number of times the **while** loop iterates successfully).

Question 3:

```
InsertionSort(A[1..n])
for (i = 2; i <= n; i++)
    v = A[i];
    j = i - 1;
    while (j > 0 & A[i] < v)
        A[j + 1] = A[j];
        j--;
    A[j + 1] = v;
```

We simply alter the comparison criterion in the **while** loop so that the larger numbers are “pulled” towards the head of the array.

Question 4:

Elementary operation: (i <= n-1 & ascending).
Size of the input: n (The length of the input array).

Worst-case input: L is already sorted in ascending order.
Number of elementary operations performed: n
In particular: (i <= n-1 & ascending) returns True for n-1 times, and returns False when i = n, which will then break the **while** loop.

Best-case input: L is already sorted but in descending order.
Number of elementary operations performed: 2
In particular: The comparison operations (1 <= n-1 & ascending) returns True. Then ascending will be set to False since the array is in descending order and i will be incremented to 2. After that (2 <= n-1 & ascending) returns False and break the **while** loop.