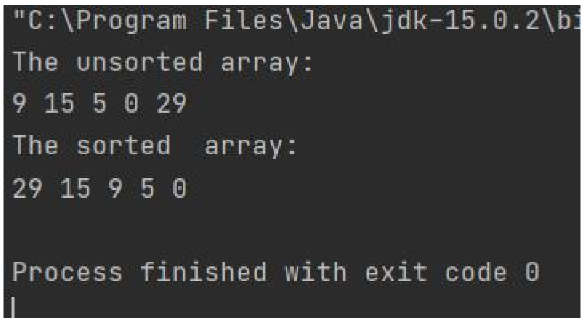
Quiz 12

Name: Zian Wang

Program:

The program will sort the array in the non-increasing order.



```
"C:\Program Files\Java\jdk-15.0.2\bi
The unsorted array:
9 15 5 0 29
The sorted  array:
29 15 9 5 0

Process finished with exit code 0
|
```

Time complexity analysis:

Bubble sort:
We assume there are n elements in the unsorted array. This algorithm's basic operation is the comparison and the swap of two adjacent elements. The swap operation is consisted by 3 operations: temp = S[t-1], S[t-1] = S[t], S[t] = temp.

Firstly we analyze the Worst-Case time complexity. When the unsorted array is in the reversed order of what we want (non-increasing if we want non-decreasing, non-decreasing if we want non-increasing), the comparison and operation should be done the maximum times. The comparison is in a for-loop which is also in a for-loop, the outside for-loop loops n times because the array is in the reversed order, and each time's loop makes only one element in the right place, so loop n times makes all the elements in the right place, the inside for-loop always loops n-1 times. Therefore, in the worst case, the comparison is done n*(n-1) times. For the swap operation, it is in the if chunk which is inside the two for-loop we talked before. We consider the inside for-loop, the array is in the reversed order when the first loop goes, so the swap will be done n-1 times to move the current first element to the last place, then when the second loop goes, the array is in the reversed order except the last element, so the swap will be done n-2 times to move the current first element to the second last place, then the for-loop goes and goes by the control of the outside for-loop just like this: each time the swap is done one times less. Finally the last time for-loop do not do the swap operation, and the outside for-loop is also done, so in the worst case, the swap is done (n-1) + (n-2) +... + 0 = n*(n-1) /2 times, so the assignment operation is done 3*n*(n-1) /2 times. So the Worst-Case time complexity of Bubble Sort is O(n*(n-1) + 3*n*(n-1) /2) = O(n^2).

Then we analyze the Average-Case time complexity of Bubble sort. We assume that every element has the same probability to be in any place, so every element has 1/n probability to be in any place. We also know that every element has only one right place, so every element has 1/n probability to move (n-1) distance, and the same 1/n probability to move (n-2) distance, also the same 1/n probability to move (n-3) distance,... , also the same 1/n probability to move 1 place, also the same 1/n probability just in the right place. Therefore, for one element, the swap operation has the same 1/n probability to be done from 0 time to (n-1) times, so the average time that the swap operation is done for one element is 0 + 1/n + 2/n +... + (n-1)/n = (n-1)*n /2n. This time is for one element, if we want to sort all n elements, the swap operation has to be done n* (0 + 1/n + 2/n +... + (n-1)/n) = (n-1)*n /2 times in average. Therefore, if we only consider the swap operation, the Bubble Sort's Average-time complexity is O((n-1)*n /2) = O(n^2), and its Worst-Time complexity is also O(n^2), so there is no need for us to consider the comparison operation because the Average-Time complexity is always smaller than the Worst-Time complexity. In other words, even if we add the time cost by comparison, its Average-Time complexity can not be larger than its Worst-Case time complexity: O(n^2). Therefore, the Average-Time complexity of Bubble Sort is also O(n^2).

Insertion Sort:

This analysis is the same as the analysis in the text book. Because the comparison of S[j] with x is the basic operation, and the worst case is also the array is in the reversed order of what we want. Therefore, in the worst case, the while-loop loops (i-1) times in each time of the outside for-loop, and the for-loop controls the value of i, and it will loop (n-1) times, so the comparison will be done 1 + 2 +.. + (n-1) = (n-1)*n/2 times. Therefore, the Worst-Case time complexity of Insertion sort is O(n^2).

In the average case, we also assume that every element has the same probability to stay in any place. Therefore, in the inner while-loop, the comparison is done (i+1) /2 - 1 /i times, and the outside for-loop loops (n-1) times, also controls the value of i. Therefore, the comparison is done (3/2 - 1/2) + (4/2 - 1/3) +... + ((n+1)/2 - 1/n) = (n+4)(n-1) /4 - ln(n) = n^2/4. Therefore, the Average-Case time complexity is also O(n^2).

Exchange Sort:

The exchange sort is just like Bubble sort, because Bubble Sort is kind of the Exchange Sort, it has 2 nested for-loop and 1 if chunk inside the 2 for-loop. The basic operation is swap operation. The worst case is still the array is in the reversed order of what we want, so the outside for-loop needs to loop n times. The first loop of the inside for-loop, we have to swap the first element to the right place, so the element has to be swapped (n-1) times, and the second loop of the inside for-loop, we have to exchange the element (n-2) times, so eventually, when the outside for-loop is done, the swap operation is done (n-1) + (n-2) +... + 1 times, just like the Bubble Sort, and because 1 time's swap operation is consisted by 3 assignment operation, the Worst-Case time complexity of Exchange Sort is O(3*(n-1)*n /2) = W(3*n*(n-1) /2) = O(n^2).

In the average case, we still assume that every element has the same probability to stay in any place. Therefore, the Average-Case time complexity is also like the case of the Bubble Sort, it is A(3*n*(n-1) /4) = O(n^2).

Selection Sort:
The selection sort has one for-loop and another for-loop inside the first one. Also a if chunk in the inside for-loop. The worst case is still the array is in the reversed order of what we want, and the basic operation is comparison and assignment. Therefore, in the worst case, the comparison is still done by (n-1) + (n-2) +... + 1 = (n-1)*n /2 times. The swap operation is done for each element, so the assignment is done by 3*n = 3n times. Therefore, the Worst-Case time complexity of Selection Sort is O((n-1)*n /2 + 3n) = O(n^2).

For the average case, we still assume that every element has the same probability to stay in any place. Therefore, the comparison is done (n-1)*n /4 times and the assignment is done 3n times, so the Average -Case time complexity is O((n-1)*n /4 + 3n) = O(n^2).