Input: { 1, 4, 7, 2, 5, 3 }

Output: { 1, 2, 3, 4, 5, 7 }

```
Bubble Sort --

Initial :   1  4  7  2  5  3

Pass 1:    1  4  2  5  3  7     (swaps = 3)

Pass 2:    1  2  4  3  5  7     (swaps = 2)

Pass 3:    1  2  3  4  5  7     (swaps = 1)

Pass 4:    1  2  3  4  5  7     (swaps = 0)
```

```
BubbleSort( int[] A, int n )
  Input: An array A containing n >= 1 integers
  Output: The sorted version of the array A

for i = 1 to (n-1)
{
     for j = 0 to (n-2)
          if A[j] > A[j+1]
          {
                 // swap A[j] with A[j+1]
                 tmp <- A[j]
                 A[j] <- A[j+1]
                 A[j+1] <- tmp
           }
}
return A
```

Complexity (best and worst case):

(c*(n-1)) * (n-1)   = c*(n-1)^2
                      = O(n^2)

c = 1 + 3 + 2 + 3 + 2 + 2 = 13

Input: {cat, mat, bat, ant}

Output: {ant, bat, cat, mat}

Input: { 7, 5, 4, 3, 2, 1 }

Output: { 1, 2, 3, 4, 5, 7 }

```
Bubble Sort --

Initial :   7  5  4  3  2  1

Pass 1:    5  4  3  2  1  7     (swaps = 5)

Pass 2:    4  3  2  1  5  7     (swaps = 4)

Pass 3:    3  2  1  4  5  7     (swaps = 3)

Pass 4:    2  1  3  4  5  7     (swaps = 2)

Pass 5:    1  2  3  4  5  7     (swaps = 1)
```

```
BubbleSortOptimized( int[] A, int n )
  Input: An array A containing n >= 1 integers
  Output: The sorted version of the array A

for i = 1 to (n-1)
{
     swaps = 0
     for j = 0 to (n-1-i)
          if A[j] > A[j+1]
          {
                 // swap A[j] with A[j+1]
                 tmp <- A[j]
                 A[j] <- A[j+1]
                 A[j+1] <- tmp
                 swaps <- swaps + 1
           }
      if swaps == 0:
          break
}
return A
```

Worst case complexity:

c * [(n-1) + (n-2) + (n-3) + ... + 1]
= c * [(n-1)*n/2]
= O(n^2)

Best case complexity:

c*(n-1)
= O(n)