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

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

Inversions:
(4, 2)
(4, 3)
(7, 5)
(7, 2)
(7, 3)
(5, 2)
(5, 3)

Selection Sort --

```
Initial :  1  4  7  5  2  3
Pass 1:    1  4  3  5  2  7
Pass 2:    1  4  3  2  5  7
Pass 3:    1  2  3  4  5  7
Pass 4:    1  2  3  4  5  7
Pass 5:    1  2  3  4  5  7
```

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

Complexity (best and worst case):

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

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

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

Selection Sort --

```
Initial :  7  5  4  3  2  1
Pass 1:    1  5  4  3  2  7
Pass 2:    1  2  4  3  5  7
Pass 3:    1  2  3  4  5  7
Pass 4:    1  2  3  4  5  7
Pass 5:    1  2  3  4  5  7
```

```
SelectionSortOptimized( 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)
{
     inversions = 0
     for j = 0 to (n-1-i)
         if A[j] > A[j+1]
                inversions <- inversions + 1
      if inversions == 0:
         break

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

Worst case complexity = O(n^2)
Best case complexity = O(n)