```
selection_sort(A[0..n-1])
---------------------------------
k = 0
while k < n {          (*)
   i = index j in {k..n-1} for which A[j] is minimal
   swap A[k] and A[i]
   k = k + 1
}
```

Let A' be the original array passed on to the function

Loop invariant: A[0..k-1] contains the smallest k elements of A' in sorted order,
and A is a permutation of A'

Initialization: (k=0) A[0..-1] contains the smallest 0 elements of A' - trivially true,
A is also a permutation of A' (because A = A')

Maintenance: Assume invariant holds at point (*) in the program execution, show that it also holds after the loop body is executed

After the inner loop is executed, i contains an index relating to a smallest element in A[k..n-1]
Swapping it with A[k] results in sorted array A[0..k] containing the smallest k+1 values of A',
because A[0..k-1] was sorted and A[k..n-1] contained values bigger or equal to A[k-1].
Also, A was a permutation of A'. So, when swapping two elements, it still is. k is then incremented
and the loop invariant holds once more

Termination: The algorithm stops after exactly n iterations (counting) with k = n. Pluging k=n into the loop invariant shows A[0..n-1] contains the smallest elements of A' in sorted order Q.E.D.