How A Selection Sort Works

Suppose we have an array of numbers $A = [21, 34, 45, 10, 76, 67]$ which we "index" beginning at 0 (so $A[0] = 21$ and $A[5] = 67$). In additional suppose we have a function called PosOfSmallest() which returns the position of the smallest value in an array between positions $k$ and $n$.

```c
int PosOfSmallest(double A[], /*in*/ int k /*in*/, int n /*in*/) {
    int min = k;  // assume smallest is at position k
    for(int i = k+1; i <= n; i++)
        if (A[i] < A[min])        // something smaller?
            min = i;              // update current min
    return min;
}
```

For example, $\text{PosOfSmallest}(A, k = 1, n = 4)$ returns 3 (since $A[3] = 10$)

We can use the PosOfSmallest() function to sort the array $A = [21, 34, 45, 10, 76, 67]$

1. Call $\text{min} = \text{PosOfSmallest}(A, k = 0, n = 5)$. This returns 3 which we store in $\text{min}$

Then we exchange the values of $A[0]$ and $A[\text{min}=3]$:

$A = [10, 34, 45, 21, 76, 67]$

At this point the smallest item is in the correct position

2. Next call $\text{min} = \text{PosOfSmallest}(A, k = 1, n = 5)$. This returns 3 which we store in $\text{min}$.

Then we exchange the values of $A[1]$ and $A[\text{min}=3]$ so:

$A = [10, 21, 45, 34, 76, 67]$

The list is partially sorted - the two smallest items are in the correct positions

3. Call $\text{PosOfSmallest}(A, k=2, n = 5)$. This returns 3 which we store in $\text{min}$

Then we exchange the values of $A[2]$ and $A[\text{min}=3]$ so:

$A = [10, 21, 34, 45, 76, 67]$

4. Call $\text{PosOfSmallest}(A, k=3, n=5)$. This returns 3 which we store in $\text{min}$

Then we exchange the values of $A[3]$ and $A[\text{min}=3]$ so:

$A = [10, 21, 34, 45, 76, 67]$

Observe that this changes nothing

5. Finally call $\text{PosOfSmallest}(A, k=4, n=5)$. This returns 5 which we store in $\text{min}$.

Then we exchange the values of $A[4]$ with $A[\text{min}=5]$ so:

$A = [10, 21, 34, 45, 67, 76]$

Observe that we can stop after "size -1" iterations since after the 5th time, by default the largest value is in the last position!

Observe in the five calls to $\text{PosOfSmallest()}$ parameter $k$ increments from 0 to 4 while $n$ stays fixed at 5, the index of the last component, which is one less than the size of the array (6). If we embed this into a for loop, we sort the array

```c
for (int k = 0, k < size-1; k++)   // do size-1 times
    {min = PosOfSmallest(A, k, size-1);
    }
```