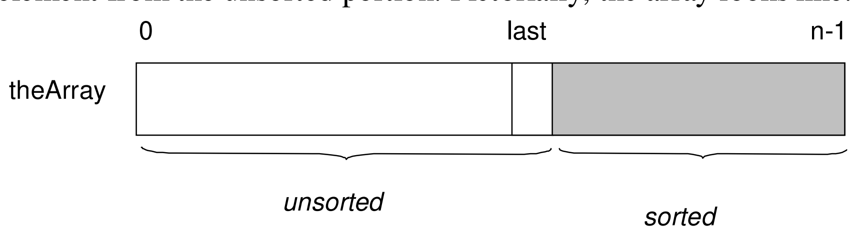
```
int indexOfLargest(Comparable[] theArray, int size);
// pre:  0 ≤ size ≤ theArray.length, theArray is non-null and every
//       element of the array is non-null
// post: returns result such that 0 ≤ result ≤ size-1 and
//       theArray[i] <= theArray[result] for all i such that 0 ≤ i ≤ size-1

void selectionSort(Comparable[] theArray, int n) {
// pre:  0 ≤ n ≤ theArray.length, theArray is non-null and every
//       element of the array is non-null
// post: values in theArray[0..n-1] are in ascending order

   for (int last = n-1;                          // initialization
      // for-inv: -1 ≤ last ≤ n-1 < theArray.length and
      // theArray and all of its elements are non-null
      // theArray[i] ≤ theArray[i+1] ∀ last+1 ≤ i ≤ n-2 and
      // theArray[i] ≤ theArray[j] ∀ 0 ≤ i ≤ last, last+1 ≤ j ≤ n-1

         last >= 1; last-- /* body */) {

      int largest = indexOfLargest(theArray, last+1);
      // found-largest: 1 ≤ last ≤ n-1 < theArray.length and
      // theArray and all of its elements are non-null
      // theArray[i] ≤ theArray[i+1] ∀ last+1 ≤ i ≤ n-2 and
      // theArray[i] ≤ theArray[j] ∀ 0 ≤ i ≤ last, last+1 ≤ j ≤ n-1 and
      // theArray[i] ≤ theArray[largest] where 0 ≤ largest ≤ last and
      //                                 ∀ 0 ≤ i ≤ last,

      Comparable temp = theArray[largest];      // body
      theArray[largest] = theArray[last];       // body
      theArray[last] = temp;                    // body
   }
   // term: theArray[i] ≤ theArray[i+1] ∀ 0 ≤ i ≤ n-2
}
```

The for-loop invariant, in English, essentially states that the portion of the array from index `last+1` to index `n-1` is in ascending order, and that every element in the sorted portion is greater than (or equal) to every element from the unsorted portion. Pictorially, the array looks like:



Note that it also true that the portion of the array starting at index `last` through to `n-1` is in sorted order. We could adjust the assertions to reflect this but it does not gain us anything in the proof of correctness largely because we do not know any relationship between the object at index `last` and the elements to the left of this object.

We now complete the steps of a formal verification that `selectionSort` is correct under the assumption that `indexOflargest` is correct.

1. *precondition & initialization ⇒ for-inv*

   From the precondition we know that `0 ≤ n ≤ theArray.length`. From the initialization we know that `last = n-1`. Together, this tells us that `-1 ≤ last ≤ n-1 < theArray.length`. `last = n-1` also implies that the remainder of the for loop invariant is vacuously true. That is, since there is no known sorted section, we can say anything we want about this nonexistent subarray. Since `theArray` does not change, it and all its elements are still non-null when we first encounter the for loop.

   ∴ precondition & initialization ⇒ for-inv

2. *for-inv & loop entry ⇒ indexOfLargest precondition*

   `indexOfLargest` is called with `size = last + 1`. Therefore, the loop invariant tells us both that `theArray` and all its elements are non-null, and that `0 ≤ size ≤ theArray.length`. That is, the precondition of `indexOfLargest` is true. We should also note that `1 ≤ last` since we have entered the loop.

   ∴ for-inv & loop entry ⇒ indexOfLargest precondition

3. *indexOfLargest postcondition ⇒ found-largest*

   `largest` is assigned the `result` of `indexOfLargest` and we still have that `size = last + 1`. So direct substitution into the postcondition of `indexOflargest` gives us that `0 ≤ largest ≤ last` and `theArray[i] <= theArray[largest]` for all `i` such that `0 ≤ i ≤ last`. The postcondition implies that `indexOfLargest` does not change the elements of `theArray`. The method cannot change `last` or `theArray` itself. This implies that the remainder of the `found-largest` assertion must be true.

   ∴ indexOfLargest postcondition ⇒ found-largest

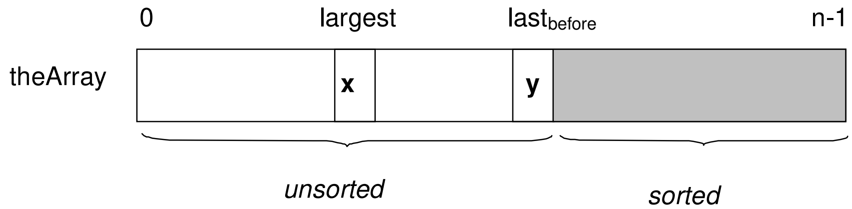
4. *found-largest + body ⇒ for-inv*

   Since the body of the loop does not alter the elements of `theArray` or `theArray` itself, we know that `-1 ≤ last ≤ n-1 < theArray.length` and `theArray` and all of its elements are non-null.

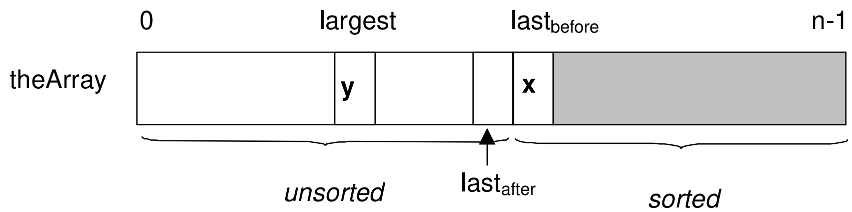
   From `found-largest` we also know that $1 \le last_{before} \le n-1$
   and $theArray[i] \le theArray[i+1]\ \forall\ last_{before}+1 \le i \le n-2$
   and $theArray[i] \le theArray[j]\ \forall\ 0 \le i \le last_{before}$ and $last_{before}+1 \le j \le n-1$
   and $theArray[i] \le theArray[largest]\ \forall\ 0 \le i \le last_{before}$
   and $0 \le largest \le last_{before}$

   The body swaps $theArray[last_{before}]$ with `theArray[largest]` and sets $last_{after} = last_{before} - 1$. So $1 \le last_{before} \le n-1$ implies $-1 \le last_{after} \le n-2$.

Therefore



is transformed into



The element we have swapped to the front of the sorted part must be smaller than or equal to all elements in the sorted part, otherwise it would have been picked earlier. Similarly, we already know it was the largest element in the unsorted part, so all elements in the unsorted part must be less than or equal to it, so

$\texttt{theArray[i]} \le \texttt{theArray[last}_{\texttt{after}}\texttt{]}\ \forall\ 0 \le \texttt{i} \le \texttt{last}_{\texttt{after}}$ and
$\texttt{theArray[i]} \le \texttt{theArray[j]}\ \forall\ 0 \le \texttt{i} \le \texttt{last}_{\texttt{after}}$ and $\texttt{last}_{\texttt{after}}+1 \le \texttt{j} \le \texttt{n}-1$

$\therefore$ found-largest + body $\Rightarrow$ for-inv

5. *for-inv & loop exit $\Rightarrow$ term*

From `for-inv` we know that $-1 \le \texttt{last} \le \texttt{n}-1$ and if the loop exits, we know that $\texttt{last} < 1$. Hence either $\texttt{last} = -1$ or $\texttt{last} = 0$. Substituting into for-inv therefore reveals that either

$\texttt{theArray[i]} \le \texttt{theArray[i+1]}\ \forall\ 0 \le \texttt{i} \le \texttt{n}-2$ and
$\texttt{theArray[i]} \le \texttt{theArray[j]}\ \forall\ 0 \le \texttt{i} \le -1$ and $1 \le \texttt{j} \le \texttt{n}-1$.

or

$\texttt{theArray[i]} \le \texttt{theArray[i+1]}\ \forall\ 1 \le \texttt{i} \le \texttt{n}-2$ and
$\texttt{theArray[i]} \le \texttt{theArray[j]}\ \forall\ 0 \le \texttt{i} \le 0$ and $1 \le \texttt{j} \le \texttt{n}-1$.

In either case, term is true.

$\therefore$ for-inv & loop exit $\Rightarrow$ term

6. *term $\Rightarrow$ `selectionSort` postcondition*

This is trivially true.

7. *Bounding function for for-inv*

Let `f(n) = last+1` be the bounding function. Show that is non-negative, integer, and decreasing.

Since `last` is an integer initially set to `n−1` ($0 \le n \le$ `theArray.length`), and it is decremented by the integer 1 for each loop iteration, the bounding function is integer-valued.

Since we have now proved that the loop invariant is true every time it is reached, we know that `f(n)` is non-negative during execution of the loop.

Since `f(n)` starts at `n` and has one subtracted on every loop iteration, the function is decreasing (alternately, `f(n)` is represented by the graph

$$f(x) = -x \ \forall\ 0 \le x \le n = \texttt{theArray.length},\ x \in \Re$$

whose derivate is negative implying `f(n)` is decreasing).