Program Verification Prof. Dr. H. Peter Gumm
Summer 2023

Exercise sheet 07

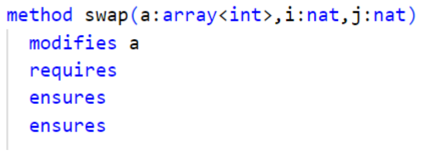
Deadline: June 10, 8:00 p.m.

Please submit only a Dafny-file **ex07_*your_name*.dfy**.

The combination of the following problems combined yield a complete verification of the selectionsort algorithm.

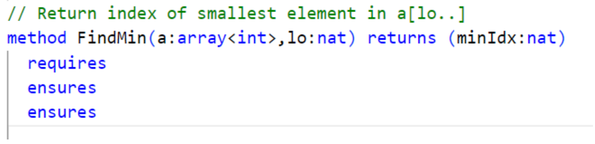
Problem 1 (6 points)**.** You are supposed to complete the following Method headers with appropriate *ensures/requires* specifications. You are not required to implement the methods.

(a). Method $swap(a, i, j)$ is supposed to swap in array a the elements at indices i and j.



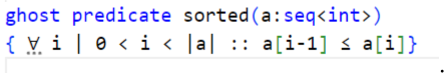
```
method swap(a:array<int>,i:nat,j:nat)
  modifies a
  requires
  ensures
  ensures
```

(b). Method $FindMin$ shall return the index of the smallest element in section $a[lo..]$ of integer array a.



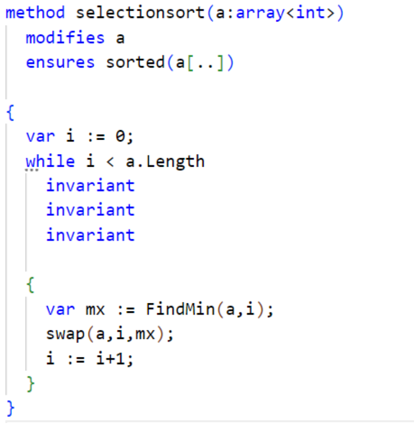
```
// Return index of smallest element in a[lo..]
method FindMin(a:array<int>,lo:nat) returns (minIdx:nat)
  requires
  ensures
  ensures
```

Problem 2 (3 pts)**.** Recall that from an array a you can obtain the sequence of elements in a as $a[..]$. Therefore, to check whether an array is sorted, it is enough to check whether $a[..]$ is sorted. The *ghost predicate sorted* below defines when a sequence of integers is sorted.



```
ghost predicate sorted(a:seq<int>)
{ ∀ i | 0 < i < |a| :: a[i-1] ≤ a[i]}
```

Below you find an implementation of the sorting algorithm *selectionsort*. Add appropriate *invariants* so that the algorithm verifies. Notice, that it uses the methods *swap* and *FindMin* from the previous problem, but it needs only their specifications, not their implementations.



```
method selectionsort(a:array<int>)
  modifies a
  ensures sorted(a[..])

{
  var i := 0;
  while i < a.Length
    invariant
    invariant
    invariant

  {
    var mx := FindMin(a,i);
    swap(a,i,mx);
    i := i+1;
  }
}
```

Problem 3 (3 pts)**.** In the previous problem we only specified that *selection sort* should modify the array so that it becomes sorted. In order to make sure that the sorted array contains the same elements as the original one, we can use the *multiset* data structure. A multiset is like a list, where the order of elements does not matter, at the same time it is like a set, but the elements can have multiple occurrences in the multiset. Thus, e.g., $multiset\{1, 2, 5, 2, 1, 4, 3, 3, 1\} = multiset\{1, 1, 1, 2, 2, 3, 3, 4, 5\}$, but $multiset\{1, 1, 2\} \neq multiset\{1, 2\}$.

Add $ensures\, multiset(a[..]) = multiset(old(a[..]))$ to the specification of *selectionsort* and augment this and the other involved methods, so that all verify in Dafny.

Problem 4 (4 pts)**.** Implement $FindMin$, so that its specification (Problem 1,(b)) verifies.