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

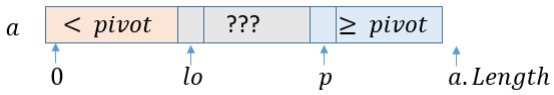
Exercise sheet 08

Deadline: June 17, 8:00 p.m.

Please submit only a Dafny-file **ex08_*your_name*.dfy**.

Problem 1 (5 points)**.** The algorithm *Quicksort*, for sorting an array a, needs an auxiliary method *partition*, which chooses some element $pivot \in a[..]$, then rearranges the elements of a, so that for some index p one has $a[p] = pivot$ and all elements of a which are smaller than $pivot$ are to the left of p and all elements larger or equal to the $pivot$ are to the right of p.
The final index p is returned. Of course, no element of a may be added or deleted.

The following program implements such a partitioning algorithm. It maintains the invariant depicted in the following sketch (It may in general be helpful to yourself if you draw such sketches of your intended specifications or invariants):



Your task is to add the described specifications as postconditions (*ensures*) and to add *invariant*s so that the program verifies in Dafny. (You will probably need several *ensures* and several *invariant*s.)

```
method partition(a:array<int>) returns (p:nat)
  modifies a
  requires a.Length > 0
  ensures   ....
{
  p := a.Length-1;
  var lo, pivot := 0, a[p];

  while lo < p
    invariant  ...
  {
    if a[lo] < pivot { lo := lo + 1; }
    else  { p := p - 1; a[lo],a[p] := a[p],a[lo]; }
  }
  a[p], a[a.Length-1] := a[a.Length-1],a[p];
}
```

Problem 2 (5 pts)**.** Let T be an arbitrary type, $a : array\langle T\rangle$ an array of elements of type T and $Prop : T \to bool$ a property which elements of type T may have or not have.

The following method assumes that there exists an element in a satisfying $Prop$. It will return the index i of such an element satisfying $Prop(a[i])$.

```
method FindWith<T>(a:array<T>,Prop:T->bool) returns (i:nat)
requires exists k | 0 <= k < a.Length :: Prop(a[k])
ensures
ensures Prop(a[i])
{
  i := 0;
  while !Prop(a[i])
    invariant
    invariant
    decreases
    {
      i := i+1;
    }
}
```

Complete this method, so that it verifies.

Problem 3 (6 pts)**.** Given a sorted array a, and given an integer sum, the method

```
method findSum(a:array<int>,sum:int) returns (lo:nat,hi:nat)
```

is supposed to find indices lo and hi such that $a[lo]+a[hi] == sum$. Your method should assume (**requires**) that

- the array a is sorted
- indices i, j with $a[i] + a[j] = sum$ do exist.

In order to specify that an array is sorted, it is easiest to work with the following definition:

```
ghost predicate sorted(a:array<int>)
  reads a
{ forall i,j | 0 ≤ i < j < a.Length :: a[i] ≤ a[j]}
```

Implement and verify $findSum$.

Hint: Start with indices $lo := 0$ and $hi := a.Length - 1$. Use a *while loop* to move the indices towards each other until the the desired indices are found. An invariant of the loop should state, in which part of the array the desired indices still remain.