Dafny coursework exercises

John Wickerson

Autumn term 2019

Tasks are ordered in roughly increasing order of difficulty. Tasks labelled (⋆) are expected to be straightforward. Tasks labelled (⋆⋆) should be manageable but may require quite a bit of thinking, and it may be necessary to consult additional sources of information, such as an online Dafny tutorial or Stack Overflow. Tasks labelled (⋆⋆⋆) are challenging. It is not expected that many students will complete these, but partial credit will be given to partial answers.

Submission process. You are expected to produce a single Dafny source file called `YourName.dfy`. This file should contain your solutions to all of the tasks below that you have attempted. You are not expected to complete all tasks. You *are* expected, however, to provide detailed annotations throughout your file (in the form of `/*comments*/` or `//comments`) that demonstrate the extent to which you have understood the software verification process.

Task 1 (⋆) Write a predicate that determines whether an array of integers is sorted in ascending order. Here is a template:

```
predicate sorted(A:array<int>)
  reads A
{
  // ...
}
```

[[[Edit 4-Nov-2019: I have decided to give away the answer to this task, because it occurs to me that a wrong answer here could jeopardise the rest of the tasks, which is not good! So, here is the answer:

```
predicate sorted(A:array<int>)
  reads A
{
```

```
  forall m,n :: 0 <= m < n < A.Length ==> A[m] <= A[n]
}
```

]]]

Task 2 (⋆⋆) Here is an implementation of bubble sort.[1]

```
method bubble_sort(A:array<int>)
  ensures sorted(A)
  modifies A
{
  var i := 0;
  while i < A.Length {
    var j := 1;
    while j < A.Length - i {
      if A[j-1] > A[j] {
        A[j-1], A[j] := A[j], A[j-1];
      }
      j := j+1;
    }
    i := i+1;
  }
}
```

Instrument this code with enough loop invariants (and other assertions as you see fit) so that Dafny can prove that the postcondition is always met. *[Hint: you might find it helpful to define a predicate that determines whether a given* region *of an array is sorted.]*

You might find the following `Main` function helpful if you want to actually try *running* the code (by pressing `F5`).

```
method Main() {
  var A:array<int> := new int[7] [4,0,1,9,7,1,2];
  print "Before: ", A[0], A[1], A[2], A[3],
        A[4], A[5], A[6], "\n";
  bubble_sort(A);
  print "After:  ", A[0], A[1], A[2], A[3],
        A[4], A[5], A[6], "\n";
}
```

Task 3 (⋆⋆) Here is an implementation of selection sort.[2]

[1] https://en.wikipedia.org/wiki/Bubble_sort
[2] https://en.wikipedia.org/wiki/Selection_sort

```
method selection_sort(A:array<int>)
  ensures sorted(A)
  modifies A
{
  var i := 0;
  while i < A.Length {
    var k := i;
    var j := i+1;
    while j < A.Length {
      if A[k] > A[j] {
        k := j;
      }
      j := j+1;
    }
    A[k], A[i] := A[i], A[k];
    i := i + 1;
  }
}
```

Instrument this code with enough loop invariants (and other assertions as you see fit) so that Dafny can prove that the postcondition is always met.

Task 4 (★★★) Here is an implementation of insertion sort.[3]

```
method insertion_sort(A:array<int>)
 ensures sorted(A)
 modifies A
{
 var i := 0;
 while i < A.Length {
   var j := i;
   var tmp := A[j];
   while 1 <= j && tmp < A[j-1] {
     A[j] := A[j-1];
     j := j-1;
   }
   A[j] := tmp;
   i := i+1;
 }
}
```

[3] https://en.wikipedia.org/wiki/Insertion_sort

Instrument this code with enough loop invariants (and other assertions as you see fit) so that Dafny can prove that the postcondition is always met.

Task 5 (⋆⋆⋆) Here is an implementation of Shellsort.[4]

```
method shellsort(A:array<int>)
  modifies A
  ensures sorted(A)
{
  var stride := A.Length / 2;
  while 0 < stride {
    var i := 0;
    while i < A.Length {
      var j := i;
      var tmp := A[j];
      while stride <= j && tmp < A[j-stride] {
        A[j] := A[j-stride];
        j := j-stride;
      }
      A[j] := tmp;
      i := i+1;
    }
    stride := stride / 2;
  }
}
```

Instrument this code with enough loop invariants (and other assertions as you see fit) so that Dafny can prove that the postcondition is always met. *[Hint: you may find it helpful to note the similarities between this algorithm and insertion sort.]*

Task 6 (⋆⋆) Here is an implementation of what I have christened 'JohnSort'.

```
method john_sort(A:array<int>)
  modifies A
  ensures sorted(A)
{
  var i := 0;
  while i < A.Length {
    A[i] := 42;
    i := i + 1;
```

[4] https://en.wikipedia.org/wiki/Shellsort

```
    }
}
```

Is it possible to prove that the postcondition is always met? What are the implications of that?