Dafny coursework exercises

John Wickerson

Autumn term 2022

There are three tasks, each of which involve some programming, some verifying, and a few short questions. The tasks appear in roughly increasing order of difficulty, and each is worth 15 marks. Tasks labelled ($\star$) are expected to be straightforward. Tasks labelled ($\star\star$) should be manageable but may require quite a bit of thinking. Tasks labelled ($\star\star\star$) are highly challenging; it is not expected that many students will complete these.

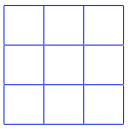
Marking principles. If you have completed a task, you will get full marks for it and it is not necessary to show your working. If you have not managed to complete a task, partial credit may be given if you can demonstrate your thought process. For instance, you might not be able to come up with *all* the invariants that are necessary to complete the verification, but perhaps you can confirm *some* invariants and express (in comments) some of the other invariants that you think are needed but haven't managed to verify.

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 welcome to show your working on incomplete tasks by decorating your file with `/*comments*/` or `//comments`. Some of the tasks contain questions that require short written answers; these answers can be provided as comments.

Plagiarism policy. You **are** allowed to consult the coursework tasks from previous years – the questions and model solutions for these are available. You **are** allowed to consult internet sources like Dafny tutorials. You **are** allowed to work together with the other student in your pair. Please **don't**

submit these programs as questions on Stack Overflow! And please **don't** share your answers to these tasks outside of your own pair. If you would like to share your answers to these tasks publicly, e.g. on a public GitHub repo, you are welcome to do so after the deadline, but please check with me first, because some students may still be working on the coursework with an extended deadline.

Task 1 (⋆) How many squares (of any size) are in the following 3x3 grid? [1 mark]



Here is a Dafny program that calculates how many squares are in an `n`x`n` grid.


```
method countsquares(n:nat) returns (result:nat)
  ensures result == /*TODO*/
{
  var i := 0;
  result := 0;
  while i < n
    i := i + 1;
    result := result + i * i;
  }
}
```


Replace `/*TODO*/` with an expression that captures the relationship between the input `n` and the calculated `result`.[4 marks] Add annotations to your program so that Dafny can verify that your postcondition always holds.[5 marks]

Here is another Dafny program that calculates the same result in a different way.

```
method countsquares2(n:nat) returns (result:nat)
  ensures result == /*TODO*/
{
  var i := n;
  result := 0;
  while i > 0 {
    result := result + i * i;
```

```
    i := i - 1;
  }
}
```

Replace `/*TODO*/` with the same postcondition as before, and add annotations so that Dafny can verify that it always holds.[4 marks]

Which is easier to verify, `countsquares` or `countsquares2`? Comment briefly on why that might be.[1 mark]

Task 2 (⋆⋆) Let us define the `sorted` predicate as follows:

```
predicate sorted(A:array<int>)
  reads A
{
  forall m,n ::
    0 <= m < n < A.Length ==> A[m] <= A[n]
}
```


Here is a Dafny implementation of binary search for a value `v` in a `sorted` array `A` of integers.


```
method binarysearch_between(A:array<int>, v:int,
  lo:int, hi:int) returns (result:bool)
{
  if lo == hi {
    return false;
  }
  var mid:int := (lo + hi) / 2;
  if v == A[mid] {
    return true;
  }
  if v < A[mid] {
    result := binarysearch_between(A,v,lo,mid);
  }
  if v > A[mid] {
    result := binarysearch_between(A,v,mid+1,hi);
  }
}

method binarysearch(A:array<int>, v:int)
  returns (result:bool)
{
  result := binarysearch_between(A,v,0,A.Length);
```

```
}
```

Provide a postcondition for the `binarysearch` method that says that `result` is **`true`** if and only if `v` is one of the elements in `A`.[3 marks] Add annotations so that Dafny can verify that this postcondition always holds.[5 marks]

Write a second implementation of binary search, called `binarysearch_iter`, that uses iteration (a **`while`** loop) instead of recursion.[3 marks] Prove that your iterative implementation satisfies the same postcondition as the recursive implementation.[4 marks]

Task 3 (★★★) Here is a Dafny implementation of the Quicksort algorithm published by C.A.R. Hoare in 1961:

```
method partition(A:array<int>, lo:int, hi:int)
  returns (pivot:int)
  requires 0 <= lo < hi <= A.Length
  ensures 0 <= lo <= pivot < hi
  ensures forall k ::
    (0 <= k < lo || hi <= k < A.Length) ==>
      old(A[k]) == A[k]
  modifies A
{
  pivot := lo;
  var i := lo+1;
  while i < hi
    invariant 0 <= lo <= pivot < i <= hi
    invariant forall k ::
      (0 <= k < lo || hi <= k < A.Length) ==>
        old(A[k]) == A[k]
    decreases hi - i
  {
    if A[i] < A[pivot] {
      var j := i-1;
      var tmp := A[i];
      A[i] := A[j];
      while pivot < j
        invariant forall k ::
          (0 <= k < lo || hi <= k < A.Length) ==>
            old(A[k]) == A[k]
        decreases j
      {
        A[j+1] := A[j];
```

```
30            j := j-1;
31          }
32          A[pivot+1] := A[pivot];
33          A[pivot] := tmp;
34          pivot := pivot+1;
35        }
36        i := i+1;
37      }
38    }
39
40    method quicksort_between(A:array<int>, lo:int,
41        hi:int)
42      requires 0 <= lo <= hi <= A.Length
43      ensures forall k ::
44        (0 <= k < lo || hi <= k < A.Length) ==>
45          old(A[k]) == A[k]
46      modifies A
47      decreases hi - lo
48    {
49      if lo >= hi { return; }
50      var pivot := partition(A, lo, hi);
51      quicksort_between(A, lo, pivot);
52      quicksort_between(A, pivot+1, hi);
53    }
54
55    method quicksort(A:array<int>)
56      modifies A
57    {
58      quicksort_between(A, 0, A.Length);
59    }
```


The code has been partially verified for you: enough annotations have been added to convince Dafny that (1) none of the array accesses are out-of-bounds, and (2) that the `quicksort_between` method only modifies the `A` array between indices `lo` and `hi`.

Explain briefly what the **`old`** keyword on line 7 means, and why it is needed.[2 marks]

Complete the verification by establishing `sorted(A)` as a postcondition for the `quicksort` method.[13 marks]