

# 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:

```
1 predicate sorted(A:array<int>)  
2   reads A  
3   {  
4     // ...  
5   }
```

*[[[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:*

```
1 predicate sorted(A:array<int>)  
2   reads A  
3   {
```

```

4   forall m,n :: 0 <= m < n < A.Length ==> A[m] <= A[n]
5 }

```

///

**Task 2 (☆☆)** Here is an implementation of bubble sort.<sup>1</sup>

```

1 method bubble_sort(A:array<int>)
2   ensures sorted(A)
3   modifies A
4 {
5   var i := 0;
6   while i < A.Length {
7     var j := 1;
8     while j < A.Length - i {
9       if A[j-1] > A[j] {
10        A[j-1], A[j] := A[j], A[j-1];
11      }
12      j := j+1;
13    }
14    i := i+1;
15  }
16 }

```

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).

```

1 method Main() {
2   var A:array<int> := new int[7] [4,0,1,9,7,1,2];
3   print "Before: ", A[0], A[1], A[2], A[3],
4         A[4], A[5], A[6], "\n";
5   bubble_sort(A);
6   print "After:  ", A[0], A[1], A[2], A[3],
7         A[4], A[5], A[6], "\n";
8 }

```

**Task 3 (☆☆)** Here is an implementation of selection sort.<sup>2</sup>

<sup>1</sup>[https://en.wikipedia.org/wiki/Bubble\\_sort](https://en.wikipedia.org/wiki/Bubble_sort)

<sup>2</sup>[https://en.wikipedia.org/wiki/Selection\\_sort](https://en.wikipedia.org/wiki/Selection_sort)

```

1 method selection_sort (A:array<int>)
2   ensures sorted(A)
3   modifies A
4   {
5     var i := 0;
6     while i < A.Length {
7       var k := i;
8       var j := i+1;
9       while j < A.Length {
10        if A[k] > A[j] {
11          k := j;
12        }
13        j := j+1;
14      }
15      A[k], A[i] := A[i], A[k];
16      i := i + 1;
17    }
18  }

```

Assignment Project Exam Help

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.<sup>3</sup>

```

1 method insertion_sort (A:array<int>)
2   ensures sorted(A)
3   modifies A
4   {
5     var i := 0;
6     while i < A.Length {
7       var j := i;
8       var tmp := A[j];
9       while 1 <= j && tmp < A[j-1] {
10        A[j] := A[j-1];
11        j := j-1;
12      }
13      A[j] := tmp;
14      i := i+1;
15    }
16  }

```

<sup>3</sup>[https://en.wikipedia.org/wiki/Insertion\\_sort](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.<sup>4</sup>

```
1 method shellsort (A:array<int>)
2   modifies A
3   ensures sorted(A)
4 {
5   var stride := A.Length / 2;
6   while 0 < stride {
7     var i := 0;
8     while i < A.Length {
9       var j := i;
10      var tmp := A[j];
11      while stride <= j && tmp < A[j-stride] {
12        A[j] := A[j-stride];
13        j := j-stride;
14      }
15      A[j] := tmp;
16      i := i+1;
17    }
18    stride := stride / 2;
19  }
20 }
```

Assignment Project Exam Help  
<https://powcoder.com>  
Add WeChat powcoder

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’.

```
1 method john_sort (A:array<int>)
2   modifies A
3   ensures sorted(A)
4 {
5   var i := 0;
6   while i < A.Length {
7     A[i] := 42;
8     i := i + 1;
```

---

<sup>4</sup><https://en.wikipedia.org/wiki/Shellsort>

```
9   }  
10 }
```

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

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder