# Dafny coursework exercises

John Wickerson

Autumn term 2025

There are four tasks, involving some programming and some verifying. The tasks appear in roughly increasing order of difficulty, and each is worth 25 marks. Tasks labelled (⋆) are expected to be straightforward. Tasks labelled (⋆⋆) should be manageable but may require quite a bit of thinking. Tasks labelled (⋆⋆⋆) are highly challenging; it is not expected that many students will complete these.

**Marking principles.** If you have completed a 'verification' task, you will get full marks for it and it is not necessary to show your working. When marking the 'programming' parts, I will take into account how well-written your code is.

If you have not managed to complete a task in full, 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.

Unless instructed otherwise, if you are provided with a method or a function, you should not change the computation that it does. You may freely add ghost code or assertions because these do not affect the computation that the code does. Also, unless instructed otherwise, you should not add extra preconditions to code that you are given, because extra preconditions make code less useful, and a proportionate penalty will be imposed.

**Submission process.** You are expected to produce a single Dafny source file called `Surname1Surname2.dfy`, where `Surname1` and `Surname2` are the surnames of the two students in the pair. This file should contain

[0]Revision history: first version, 15 October 2025.

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`.

**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. You **are** allowed to ask questions on Stack Overflow etc., but make your questions generic (e.g. “Why can’t Dafny prove 2+2<5?”); please **don’t** ask for solutions to these specific tasks! Please **don’t** share your answers to these tasks outside of your own pair, but please **do** help each other with general Dafny tips and tricks – after all, when one student helps another student, *both* learn!

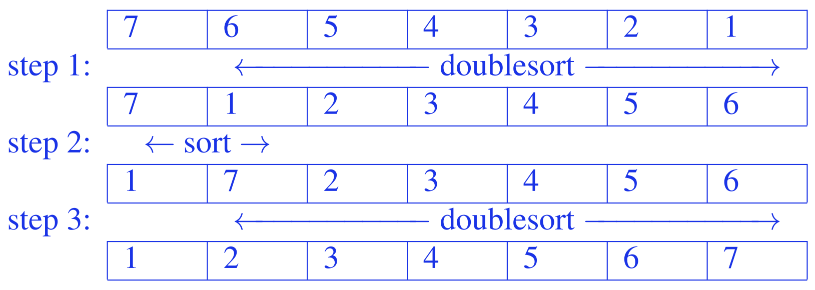
**Task 1** (⋆) Implement a method in Dafny called `sort_pair(A,i,j)`. The three parameters are an array `A` of integers, and two indexes into that array, with `i<j`. The method should swap the `i`th and `j`th elements of `A` if `A[i]` is greater than `A[j]`. The method should not modify any elements other than `A[i]` and `A[j]`.

Capture the specification above by giving one or more postconditions to `sort_pair`. Verify that they all hold. You may add any reasonable preconditions on `A`, `i`, and `j`.

**Task 2** (⋆⋆) I have invented a new sorting algorithm, and your job is to verify it. The algorithm is called “doublesort”. It is not particularly efficient.

Doublesort consists of three steps. The first step is to recursively use doublesort to sort the entire array except the first element. The second step is to swap the first and second elements of the array if they are in the wrong order. The third step is the same as the first step: to recursively use doublesort to sort the entire array except the first element.

Here’s a picture, in case it is helpful.



Doublesort can be implemented in Dafny as follows. We first define a helper method that applies doublesort to a given suffix of the array.

```
method doublesort_from(A:array<int>, lo:int)
{
  if lo + 1 < A.Length {
    doublesort_from(A, lo + 1);
    if A[lo + 1] < A[lo] {
      A[lo], A[lo + 1] := A[lo + 1], A[lo];
    }
    doublesort_from(A, lo + 1);
  }
}
```

That is, `doublesort_from(A, lo)` runs doublesort on all but the first `lo` elements of `A`. We can then define the entry point to doublesort which simply asks the helper method to doublesort the entire array.

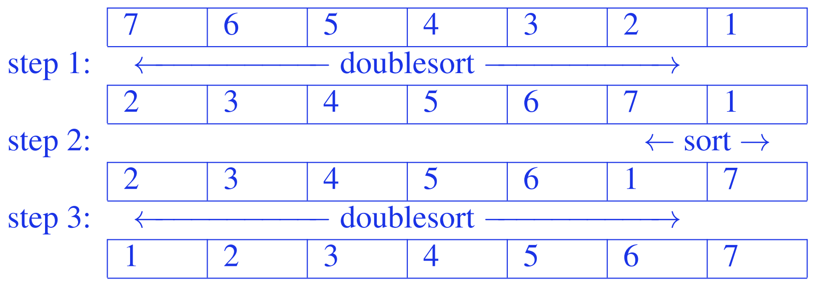
```
method doublesort(A:array<int>)
  ensures sorted(A)
{
  doublesort_from(A, 0);
}
```

Your task is to verify that `doublesort(A)` has postcondition `sorted(A)`, where `sorted` is defined in the usual way.

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

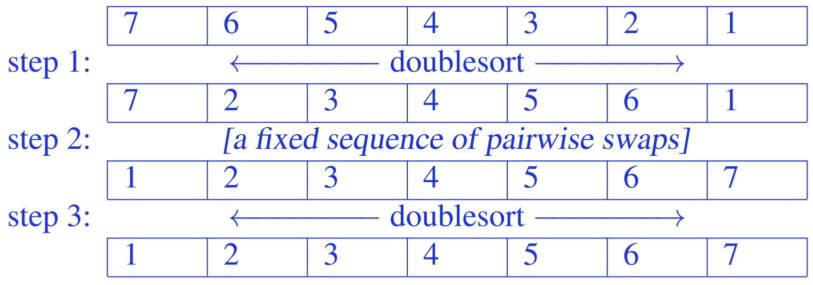
**Task 3** (⋆⋆) Implement an alternative version of doublesort that works from the opposite end of the array. That is, the first and third steps should sort the entire array except the *last* element, and the second step should swap the *last and penultimate* elements if they are out of order. Prove that your alternative version of doublesort has postcondition `sorted(A)`.

Here's a picture, in case it is helpful.



**Task 4** (⋆⋆⋆) Implement a third version of doublesort that works from *both* ends of the array. That is, the first and third steps should sort the entire array except the *first and last* elements, and the second step should carry out a fixed sequence of pairwise swaps involving the elements at the ends of the array (precise sequence to be determined by you). Prove that your third version of doublesort has postcondition `sorted(A)`.

Here's a picture, in case it is helpful.


7 6 5 4 3 2 1
step 1:
doublesort
7 2 3 4 5 6 1
step 2:
[a fixed sequence of pairwise swaps]
1 2 3 4 5 6 7
step 3:
doublesort
1 2 3 4 5 6 7