CSC410

Assignment 1

Due on 2023/09/28, at 11:59pm

Assignment Format and Guidelines on Submission

This assignment is worth 15% of the total course mark. Submit on Markus and follow these rules:

- This is a group assignment, designed for groups of 3 students. You can submit in smaller groups, but keep in mind that the workload is designed for 4 students. Use Piazza (student list) or Quercus to form your groups. Markus will be set to accept groups of up to size 4.
- There is a problem clearly marked for the graduate students enrolled in this course. If you are a group consisting of undergraduate students only, it does not count towards your assignment grade.
- Each group should submit six files named `rabbit.dfy`, `two-pointer-sum.dfy`, `gnome-sort.dfy`, and `reversal.dfy`, and in case of graduate students the latter is replaced with `array-swap-a.dfy`, `array-swap-b.dfy`, and `array-swap-c.dfy`. Note that Markus has been setup to accept exactly those files. All the required lemmas, helper functions, etc, should be included in the one file for each problem. Do not worry about not submitting the required files that do not apply to you (as a graduate or undergraduate student).
- Each file should contain the code as given in the handout. Changing the algorithm will result in zero marks for that question unless otherwise specified.
- Note that the marks associated to the problems are not proportional to their difficulty. For example, the first problem is very simple (warmup) problem, and are given a higher weight than it should have.
- Assume statements and declarations without bodies will result in zero marks for that entire question. These basically admit facts without proofs into your reasoning and are disallowed for that reason.
- Method signatures for each method should remain exactly as specified in the handout (and the accompanying code). Changing the method signature to something incompatible with the original will result in zero marks for that question.
- Make sure your final submitted proofs pass through the command line version of Dafny (and not just VSCode). VSCode can (on a rare occasion) be buggy in declaring things verified. If your proof requires specific command line options to be verified in reasonable time, make a note of them as a comment at the top of your submitted Dafny files. Our default setup is that we check your homework with Dafny 3.3.0 commandline with no options.

Note that your assignment will be automatically graded. Your function will be called from another function. If you mess with the signature, the call will fail and the autograder will give you a 0 mark.

The submission system will remain open for 12 hours after the deadline, but there is a penalty deduction formula set in Markus that deducts 4% for every hour of late submission up to 12 hours.

Word of Advice

Before you sit behind a Dafny terminal, make sure you have a detailed proof worked out on paper. If you do not have such a (detailed) complete proof, you cannot hack your way through a Dafny proof. You have already seen in class that even when we think our proofs are complete, some intermediate steps (deemed

trivial by us) require some more work to be certified in Dafny. If you have a complete proof, and are certain about its correctness, but cannot get it through to Dafny, then it most likely means that you are making a leap in reasoning somewhere that seems trivial to you, but not to the prover. We can assure you that this has nothing to do with a *feature* or a *command* that you do not know and have to dig up from a manual/tutorial. Everything you need to know to solve these has already been covered in class and in Dafny tutorials.

Problem 1 (30 points)

The file `reversal.dfy` includes a method that reverses an array in-place. A predicate is defined that formalizes the post condition, namely that the result is the reverse of the input. Prove that the postcondition as specified holds.

Graduate Students: You skip this problem, because a variation of it is already included in the grad specific problem at the end. So, part (a) of that problem replaces this problem for you.

Problem 2 (30 points)

The *two-pointers techniques* is a clever way of searching for a pair of elements in a sorted array. As an example of this, an implementation of *two-pointer sum* is given in the file `two-pointer-sum.dfy`. The method assumes a sorted array `a` as an input and returns a pair of elements whose sum adds to a given input value (`sum`). If such a pair does not exist, values `-1` are returned for both indices. The file includes the correctness specification for the method. Your task is to give the proof that the implementation satisfies it.

Note: the *"index out of range"* errors on line 29 are not a mistake. Once you enter the appropriate loop invariants, the errors will disappear.

Problem 3 (30 points)

Gnome sort is a simple sorting algorithm that can be viewed as a simplification of *insertion sort* which removes the need for nested loops. The file `gnome-sort.dfy` gives an implementation of this algorithm with a proof that it correctly sorts. As you can see, the proof is nearly trivial, because the algorithm is trivial. The tricky part of the correctness of this algorithm lies in proving that it terminates. Prove that it terminates.

Problem 4 (30 points)

A rabbit is located somewhere on a one-dimensional line, and it moves. It starts at location a. Every second, he moves b units to the right. Therefore, the location of the rabbit at time t is given by the expression $a + t * b$. But, the catch is that $a, b \in \mathbb{N}$ are constants that are unknown to us. We can only know check if the rabbit is at a particular location by querying that location, and we can only query one location at every second.

We can catch the rabbit as follows. The set $\mathbb{N} \times \mathbb{N}$ of pairs of natural numbers is countable [1]. Therefore, there exists a function $unpair : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ that covers all pairs of natural numbers. [2]

Our strategy is to check the location $x + t \times y$ at time t, where $(x, y) = unpair(t)$. There exists some t_0 such that $(a, b) = unpair(t_0)$. This means that at time t_0 we will check the location $a + t_0 * b$, where the rabbit shall be at time t_0 as well! Therefore, we are guaranteed to catch the rabbit.

Now, you take the high level argument above and the sketch of an implementation that is given to you in file `rabbit.dfy` and turn it into a complete formal argument in Dafny. Complete the implementation

[1] If you do not know what countable means, then look it up.

[2] Fun fact: this is part of the reasoning for why Dafny can use pairs of natural numbers as a termination measure.

of method *unpair* according to our strategy above, and prove that the while loop terminates (i.e. that we eventually catch the rabbit).

Hint: It is much easier to define unpair recursively, instead of defining it as a closed form function. It is also easier to formally reason about such definition in Dafny compared to the closed form.

Note: The purpose is for us to practice using a theorem prover, like Dafny, to do a formal proof that is not related to a program. The puzzle and its solution are effectively provided to you (and you may consult resources online for this as well). We are not asking you to find a solution for the puzzle. We are only asking you to use Dafny to formally prove that the provided solution for the puzzle is indeed correct. One of the main learning objectives of this first chapter of this course is to get you in a comfortable position to think about systems (computer programs or otherwise) in formal terms.

Graduate Problem (60 points)

The file `array-swap.dfy` includes a method `Reverse` that reverses a subarray of an array in-place. A predicate is defined that formalizes the postcondition.

It is a rather cute trick to use a reversal subroutine (like the one you just proved correct) to swap two consecutive array segments in place, without the need for auxiliary memory. This is problem that sometimes appears on interview questions during hiring for big companies like Google. The file includes a method `ArraySwap` that includes the (very simple and yet clever) implementation.

(a) (25 points) Prove that the postcondition as specified holds. This is the replacement of Problem 1 for the graduate students in the class.

(b) (5 points) To prove that `ArraySwap` is correct, you will need to enrich the postcondition of `Reverse`. This involves a simple addition, but the addition is essential. Add the additional postcondition and revise your proof from (a) so that the new specification of `Reverse` is proven correct. Note: the natural way of getting this done is to attempt to do (c) first, and discover the missing part during the process.

(c) (30 points) Prove `ArraySwap` satisfies its postcondition, that is, it correctly swaps the two consecutive array partitions.

File submission guidelines: For submitting parts (a,b), comment out the postcondition of `ArraySwap` so that your only prove goal becomes the postcondition of `Reverse`. For part (c), include the complete proof of everything.