CS 6110 — Assignment 6
Gavin Gray u1040250

Note: All Dafny files are included in the submitted zip file. The code files are also inserted after each question, with annotations of the form ' `// (X)` *' where 'X' corresponds to the enumerated discussion number in the student response.*

1. (**25 points**) Verify this Dafny program[1], introducing assertions as hinted by comments `//a-` below. You must also complete missing pieces of code, if any, as hinted by `//c-` below. Other comments appear as `//` and may be left alone.

   Student Response:

```
method maxarr(a: array?<int>) returns (max:int)
  requires a != null // (1)
  ensures forall i :: 0 <= i < a.Length ==> max >= a[i] // (2)
{
  max := 0; // (3)
  if (a.Length == 0) { return ;}
  var i := 0;
  while (i < a.Length)
    decreases a.Length - i // (4)
    invariant 0 <= i <= a.Length // (5)
    invariant forall j :: 0 <= j < i ==> max >= a[j] // (6)
  {
    if (a[i] > max)
      { max := a[i]; } // (7)
    i := i + 1;
  }
}
```

   1. This assertion states that 'a' must be a non-null array. This is useful because there does not exist a proper value for an array which is null.
   2. In this postcondition we are asserting that 'max' is indeed greater than (or equivalent to) every argument in the array.
   3. As per the method description, on a size 0 array 'a', the method should return 0. This is okay because the postcondition still holds, max is greater than every element in 'a'.

[1] By this, I mean that you must get a message of the following form: "*Dafny program verifier finished with X verified, 0 errors.*"

4. This decreases clause tells dafny that on each loop iteration, 'i' is getting closer to the end of the array.
5. Here we are specifying that 'i' is always staying within the array bounds.
6. This loop invariant says that the value currently in 'max' is the max of the "array so far". This helps dafny see that we are making incremental progress towards the final postcondition.

2. (**10 points**) This problem modifies the previous program to make one more "ensures" (over and above `max >= a[i]` for all `i`) that is important to have for this program to be deemed correct. Your task is to discover this ensures clause and insert it at the place where I place a hint. Then verify the program with the modified ensures clause. Now, also recall that when one strengthens the "ensures" clause, one may also have to strengthen the loop invariant (in this program, one must strengthen).

   Student Response:

```
method maxarr(a: array?<int>) returns (max: int)
  requires a != null // (1)
  ensures forall i :: 0 <= i < a.Length ==> max >= a[i] // (2)
  ensures (a.Length == 0) || (exists i :: (0 <= i < a.Length) && max == a[i]) // (8)
{
  max := 0; // (3)
  if (a.Length == 0) { return; }
  var i := 1; // (10)
  max := a[0]; // (11)
  while (i < a.Length)
    decreases a.Length - i // (4)
    invariant 0 <= i <= a.Length // (5)
    invariant forall j :: 0 <= j < i ==> max >= a[j] // (6)
    invariant exists j :: 0 <= j < i && max == a[j] // (9)
  {
    if (a[i] > max)
      { max := a[i]; } // (7)
    i := i + 1;
  }
}
```

   For the parts that are the same as *Q1* I will omit their respective annotations below.

8. This postcondition is saying that the array 'a' was either of length 0, or the maximum that was returned is in-fact an element of 'a'.

9. This loop invariant is helping the stronger postcondition. This is saying that for every iteration the value of max is an element of the "seen so far" part of the array.
10. For my loop-invariant to work I had to change the starting point of the iteration at index 1. I believe this was necessary to help the loop invariant because when 'i' was equal to 0, max wasn't equal to an array element.
11. Because we are now starting at $i := 1$ we also start with $max := a[0]$. This way, the value of 'max' is always an element of 'a'.

3. (**35 points**) Please update the Google doc of your project. A Github is optional.[2] If you have made code changes on this Github, let me know that, as well. But the Google doc is what I'll be grading. **I believe in you interacting with me frequently and making steady progress.** Also, as much as what you do, **good documentation counts for most of your points.** Good documentation is pithy, bulleted, easy-to-grade, and cuts to the chase. During the last week, you'll get the time to add more prose. Aim to have dated entries and also declare your plans for what you'll achieve in the coming week. I require all projects to meet me every week, even for 5 minutes.

   Student Response: See the GoogleDoc for updates.

[2]Depends on your project; I should have made this clear earlier.

4. (**30 points**) Verify this Dafny program, introducing assertions as hinted by comments `//a-` below. You must also complete missing pieces of code, if any, as hinted by `//c-` below. Other comments appear as `//` and may be left alone.

Student Response:

```
function recsum(a: array?<int>,Max:nat) : int
  reads a
  requires a != null
  requires 0 <= Max <= a.Length
  decreases Max
{
  if a.Length == 0 then 0
  else if 0 == Max then 0
  else a[Max - 1] + recsum(a, Max-1) // (1)
}
method sumarr(a: array?<int>) returns (sum:int)
  requires a != null
  ensures sum == recsum(a,a.Length)
{
  sum := 0;
  if (a.Length==0) {return;}
  var i : int := 0;
  while (i < a.Length)
  invariant 0 <= i <= a.Length // (2)
  invariant sum == recsum(a, i) // (3)
  decreases a.Length - i // (4)
  {
    sum := sum + a[i]; i:=i+1;
  }
}
```

1. This piece of code will recursively sum the rest of the array, adding in the value for the current element at 'Max - 1'. Note that we have to use 'Max-1' instead of 'Max' because this index starts at 'a.Length' which is one position out of bounds.
2. This invariant is stating that 'i' is constrained to the array bounds.
3. This crucial invariant helps Dafny see that we are building up 'sum' to match the final postcondition, that is, at each step the sum is equal to the sum of the "array so far".
4. This may not be needed but I find it customary to always include a decreases clause.