CSC410

Assignment 5

Due on Friday March 15, 2012 at 10pm

Assignment Format and Guidelines on Submission

You submit your assignment as a PDF file. In some cases you are asked to also submit a text file that successfully runs through Dafny verifier. Since these examples are all small, you can check your work by using Microsft's online interface for Dafny at `http://rise4fun.com/Dafny`. Doing this will give you the confidence that your work is correct, and makes our job of checking it easier. Do not forget to submit the text files (appropriately titled) separately through Markus. We will cut and paste your test into a Dafny window and check if it goes through.

Problem 1

Below, you can see a function that sorts an array. This a simpler sorting routine called *insertion sort*. The goal is to verify partial correctness of this implementation. Naturally, we expect a sorting routine to return a *sorted* array. In case you are wondering, you do not have to verify that the resulting array and the original one contain the same set of elements.

```
method insertSort(a : array<int>)
{
  var i := 1;
  while (i < a.Length)
  {
    var j := i;
    var value := a[i];
    a[i] := a[i-1];
    while (j > 0 && a[j-1] > value)
    {
      a[j] := a[j-1];
      j := j - 1;
    }
    a[j] := value;
    i := i + 1;
  }
}
```

(a) (6 points) What are the appropriate precondition and postcondition for this function?

(b) (8 points) What are the (inductive) loop invariants? Note that there are two while loops in this example.

(c) (5 points) Enumerate all basic paths in this function.

(d) (5 points) What is the verification condition for the basic path that starts and ends with the inner loop's invariant? Show your work and make sure that your final answer does not contain any instances of the *wp* function.

(e) (4 points) Submit a text file containing the code with your pre/post-conditions and loop invariants that successfully runs through Dafny.

Problem 2

Consider the following variation of the *insertion sort* implementation of problem 1.

```
method insertSort(a : array<int>)
{
  var i := 1;
  while (i < a.Length)
  {
    var j := i;
    var value := a[i];
    while (j > 0 && a[j-1] > value)
    {
      a[j] := a[j-1];
      j := j - 1;
    }
    a[j] := value;
    i := i + 1;
  }
}
```

answer parts (a), (b), and (e) from problem 1 for this variation again (each part is worth the same number of points as it does in problem 1).

Problem 3

The following elegant sort algorithm gets its name from the *Three Stooges* slapstick routine in which each stooge hits the other two.

Hint: this is a bit tricky. Try to elaborate on "what" we know about the contents of each 1/3rd array after each recursive call, with respect to the content of the other two 1/3rd arrays. Think of it as a counting argument.

```
method stoogeSort(a : array<int>, left : int, right : int)
{
  if (a[left] > a[right]) {
   // swap a[left] and a[right]
    var tmp := a[left];
    a[left] := a[right];
    a[right] := tmp;
  }
  if (left + 1 >= right)
     return;
  k := (right - left + 1) div 3;
  stoogeSort(a,left, right - k); // First two-thirds
  stoogeSort(a,left + k, right); // Last two-thirds
  stoogeSort(a,left, right - k); // First two-thirds again
}
```

(a) (6 points) What are the appropriate precondition and postcondition for this function?

(b) (5 pionts) What is the verification condition for the basic path that starts at the method's precondition and ends at the first recursive call? Show your work and make sure that your final answer does not contain any instances of the *wp* function.

(c) (3 points) Submit a text file containing the code with your pre/post-conditions that successfully runs through Dafny.

Problem 4

Remember the binary search example that you saw in the tutorial:

```
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    int m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BinarySearch(a, m + 1, u, e);
    else return BinarySearch(a, ℓ, m − 1, e);
  }
}
```

and the the fact that a tool like Dafny uses a tool like Z3 as the backend to check verification conditions. The problem with the use of "integer division" (div in the above pseudo code) is that it is not really part of the standard theory of linear arithmetic (there is not integer division operation in linear arithmetic). We want to find a fix for this:

(a) (2 points) enumerate the basic paths that include the use of "div".

(b) (4 points) Show how these basic paths can be altered so that they only use standard linear arithmetic (Hint: use an additional assume statement). How does this change affect the verification conditions?