Assignment 1

Theodore S. Norvell

6892 Due 2016 Oct 18th.

Q0 [20] Divider:
Consider the following routine for computing quotients and remainders

```
// Repeated subtraction.
method divideWithRemainder(x : int, y : int) returns ( p : int, m : int )
requires y > 0
requires x >= 0
ensures p*y + m == x
ensures 0 <= m < y
{
    p, m := 0, x ;
    // Use the negation of m < y as the guard and the rest
    // of the postcondition as the invariant
    while( m >= y )
    invariant p*y + m == x && 0 <= m
    decreases m {
        var q := 1 ;
        p, m := p + q, m - q*y ;
    }
}
```

(a) [10] Modify the routine so that it doubles q as many times as possible before adding it to p. Ensure your routine verifies.

(b) [10] Further modify the routine so that it does no multiplication. (Use a tracking variable.) Ensure that the code still verifies.

(c) Bonus [5]. If all that doubling is moved ahead of the main loop, you can make an algorithm that has two loops. The first doubles q until $q \times y > x$ the second halves q, adds q to p, and deducts $q \times y$ from m whenever that wouldn't violate the invariant that $m \geq 0$. This turns out to be more efficient. It also turns out to be the Russian multiplication algorithm run backwards! See if you can implement and verify this algorithm.

Q1 [20] Partition.

Partition is a routine to pick a number q (where $p \leq q < r$) and rearranges the items in a nonempty segment $a[p, ..r]$ of an array into three segments $a[p, ..q]$, $[a(q)]$, and $a[q+1, ..r]$ (where $p \leq q < r$) so that all items in the first are less or equal to that in the second and the item in the item in the second is less or equal to all in the third. We can specify Partition in Dafny as follows

```
method partition( a : array<int>, p : int, r : int ) returns ( q : int )
requires a != null
requires 0 <= p < r <= a.Length
modifies a
ensures PermutationOf( a[..], old( a[..] ) )
ensures a[..p] == old(a[..p])
ensures a[r..] == old(a[r..])
ensures p <= q < r
ensures AllLessOrEqual( a[p..q], [a[q]] )
ensures AllLessOrEqual( [a[q]], a[q+1..r] )
```

where PermutationOf and AllLessOrEqual are defined by

```
predicate PermutationOf( s : seq<int>, t : seq<int> )
{
    multiset(s) == multiset(t)
}
predicate AllLessOrEqual( s : seq<int>, t : seq<int> )
{
    forall i,j :: 0 <= i < |s| && 0 <= j < |t| ==> s[i] <= t[j]
}
```

Partition should run in time roughly proportional to $r - p$. As a pragmatic consideration, we would like that the sizes of the two parts be roughly equal on average, given a random array. One way to do this is to pick an arbitrary item (e.g., the first or last or middle) to be $a[q]$ as a first step. Implement and verify Partition.

Advice: Draw a picture of your invariant.

Advice: I found that algorithms that only modify the array by swapping two elements to be the easiest for the verifier to check.

Bonus:[10] Implement a sorting algorithm other than selection sort. (I will post my selection sort code.).

Submit one Dafny file per question to the D2L dropbox.