Exercise 5.1.

a) For method M1 below, we notice that the postcondition is equivalent to r * 1 == x * y. Based on this, which loop design technique will result in the stated invariant?

```
method M1(x: int, y: int) returns (r: nat)
    requires 0 <= x && 0 <= y
    ensures r == x * y
{
    r := x;
    var n := y;
    while n != 1
        invariant r * n == x * y
    ...
}
```

3/

b) For method M2 below, which loop design technique will result in the stated invariant?

```
method M2(a: array<int>, y: int) returns (r: nat)
    ensures 0 <= r <= a.Length
    ensures forall i :: 0 <= i < r ==> a[i] != y
    ensures r == a.Length || 0 <= a[r]
{
    r := 0;
    while (r != a.Length && 0 > a[r])
        invariant 0 <= r <= a.Length
        invariant forall i :: 0 <= i < r ==> a[i] != y
    ...
}
```

1/

Exercise 5.2.

The following function computes 2^n.

```
function Power(n: nat): nat {
  if n == 0 then 1 else 2 * Power(n-1)
}
```

Below is the ComputePower method from the lecture without the loop body.

```
method ComputePower(n: nat) returns (p: nat)
  ensures p == Power(n)
{
  p := 1;
  var i := 0;
  while i < n
    invariant 0 <= i <= n
    invariant p == Power(i)
}
```

Explain why the specification and/or invariant will not be satisfied when you

a) change p := 1 to p := 2? breaks the invariant
b) change p := 1 to p := 2 and change the invariant to p == Power(i + 1)? breaks the postcondition
c) change p := 1 to p := 2 and change i := 0 to i := 1? breaks the postcondition, miss some iteration

Exercise 5.3.
The method LargestIndex determines the largest index r in an integer array a where a[r] == r. It requires that such an index exists.

```
method LargestIndex(a: array<int>) returns (r: int)
    requires exists i :: 0 <= i < a.Length && a[i]== i
    ensures 0 <= r < a.Length
    ensures a[r] == r
    ensures forall i :: 0 <= i < a.Length && a[i] == i ==> i <= r
```

a) Derive a suitable loop specification (i.e., guard and loop invariants) for the method. For each invariant you must state either the loop design techniques used, or the reason the invariant was added, e.g., to place bounds on variables appearing in another invariant.

b) Provide code to initialise all variables appearing in your loop specification from (a).

Exercise 5.4.
Derive a method

```
method Cube(n: nat) returns (c: nat)
        ensures c == n * n * n
```

with a loop that iterates n times and only does addition (no multiplication). Use the *wishing* method from lectures to assist you with the derivation.

```
        invariant 0<=n<=a.Length
        invariant 0<=r<=a.Length
invariant forall i:: 0<=i<n && a[i] == i => i <=r
```