1. Show that the following program computes the square root of a natural number.

```
method squareRoot (n: nat) returns (r: nat)
 ensures r * r <= n < (r + 1) * (r + 1)
{
r := 0;
var sqr := 1;
while sqr <= n {
 r := r + 1;
 sqr := sqr + 2 * r + 1; }
}
```

2. Show that the following program computes the product of two natural numbers.

```
method product (m: nat , n: nat ) returns ( res: nat )
  ensures res == m * n;
{
var m1: nat := m; res := 0;
while m1 != 0 {
 var n1: nat := n;
 while n1 != 0 {
  res := res + 1;
  n1 := n1 - 1; }
 m1 := m1 - 1; }
}
```

3. Show that the following programs compute the summation of the first n natural numbers. Suggestion: Consider functions that state that the value is $\frac{n(n+1)}{2}$.

   (a)
   ```
   method suma ( n: nat ) returns ( s: nat ) {
   s := 0;
   var i := 0;
   while i < n {
    i := i + 1;
    s := s + i;
    }}
   ```

   (b)
   ```
   method sumd ( n: nat ) returns ( s: nat ) {
   s := 0;
   var i: int := n;
   while i > 0 {
    s := s + i;
    i := i - 1;
    }}
   ```

4. Write a a method in Dafny that given an array of integers computes the minimal sum of consecutive values `minSum`. Write the assertions that ensure the correction of your program.

5. Show the correction of the following insertion method to sort an array of integers.

```
method insertionSort(a: array<int>)
  modifies a
  ensures isSorted(a, 0, a.Length)
  ensures multiset(a[..]) == multiset(old(a[..]))
{
    var i := 0;
    while i < a.Length
    {
        var j := i;
        while j > 0 && a[j-1] > a[j]
          {
            a[j-1], a[j] := a[j], a[j-1];
            j := j - 1;
        }
        i := i + 1;
    }
}
```

6. Prove the correction of the following program w.r.t. the specifications (pre and post conditions)
.

```
method findMaxFrequency(v: array<int>) returns (maxElt: int , count: int )
        requires v != null && v.Length >= 1;
    ensures countOcc (v,maxElt,v.Length ) == count;
    ensures forall j :: 0 <= j < v.Length ==> countOcc(v, v[j], v.Length) <= count;
{
  var i := 0;
  var freq := map [];
  while i < v.Length {
   if v[i] in freq {
    var prev_freq := freq[v[i]];
        freq := freq[v[i] := prev_freq + 1];
   } else {
     freq := freq [v[ i ]:= 1];
   }
   i := i + 1;
  }
 i := 0;
 maxElt := v[0];
 while i < v.Length {
        if freq[v[i]] > freq[maxElt] {
                maxElt := v[i]; }
        i := i + 1; }
count := freq[maxElt];
}
```

```
function countOcc(v: array<int >, elt: int , n: nat) : nat {
 if n == 0 then 0 else
  countOcc(v,elt,n-1) + if v[n-1] == elt then 1 else 0
}
```

7. Consider the problems of total correctness done before by hand and using **asserts/assumes** build the tableaux using Dafny.

8. Implement and verify the correctness of the usual sorting algorithms for an array of integers: `SelectSort` e `MergeSort`.