Exercises NMCGJ:
Reasoning About Algorithms

A so-called *invariant* of a loop is a relation between the variables and parameters in a program. This relation should hold before and after each iteration, so it remains "invariant". Knowing its invariant(s) is essential for understanding the effect of a loop and to prove its correctness.

A proof of correctness based on invariant(s) requires the following steps:

1. Specify and prove the invariant(s):
   1.1. Prove that the invariant(s) hold before the loop.
   1.2. Prove that the invariant(s) hold at the end of the loop iteration, assuming it is true before the iteration.
   1.3. By induction, the invariant(s) hold after the loop.
2. Prove the correct result using the invariant(s).
3. Prove that the loop terminates in a finite number of steps.

This procedure can be followed for each loop, also for nested loops.

Exercise 1

The next algorithm computes $n!$ for a given positive integer n. Prove its correctness.

```
int i = n, fac = 1;
while (i > 1) {
   fac = fac * i;
   i--;
}
```

Exercise 2

The next algorithm computes $\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$. Prove its correctness.

```
int i = 0, binom = 1;
while (i < k) {
   binom = binom * (n - i) / (i + 1);
   i++;
}
```

Exercise 3

Write an algorithm that computes the greatest common divisor of two positive integer numbers x and y, based on the formula of Euclides: if $x>y>0$ then $gcd(x, y) = gcd(x - y, y)$. Prove the correctness of the algorithm.

Exercise 4

The next algorithm converts a positive integer *n* into its binary form (using an array). Prove its correctness.

```
int i = 0, m = n;
int[] bits = new int[32];
while (m > 0) {
   bits[i] = m % 2;
   m = m / 2;
   i++;
}
```

Exercise 5: Bubble sort

Bubble sort is a simple sorting algorithm that works roughly speaking by "bubbling": in each step, two neighboring elements are compared, and they are swapped if they are in the wrong order. In this way, the largest elements will be "bubbled" to the right.

```
int i = list.length;
while (i > 0) {
   int j = 0;
   while (j < i-1) {
      if (list[j+1] < list[j]) {
         int temp = list[j+1];
         list[j+1] = list[j];
         list[j] = temp;
      }
      j++;
   }
   i--;
}
```

The algorithm consists of an outer loop that runs over the array *list* from right to left. Everything at the right of the current position is already sorted. Everything at the left of the current position is not yet sorted, but each of these elements is smaller than any element on the right.

Prove the correctness of the algorithm. It could be helpful to make first a picture to see what is really happening...