Exercises NMCGJ:
Reasoning About Algorithms

Possible proofs for the correctness of the algorithms in the exercises.

Exercise 1

```
// precondition: n >= 1

// invariant: n! == i! * fac && i >= 1

int i = n, fac = 1;
// initialization: n! == n! * 1 && n >= 1
// => OK

while (i > 1) {
   // n! == i! * fac && i > 1
   fac = fac * i;
   // n! == (i-1)! * fac && i > 1
   i--;
   // n! == i! * fac && i >= 1
}
// n! == i! * fac && i >= 1 && i <= 1
// => n! == i! * fac && i == 1

// result: fac == n!

// termination: follows from
//   1) in each step: i is decreasing by 1
//   2) lower bound: i >= 1 (by invariant)
```

Exercise 2

```
// precondition: n >= k >= 0

// invariant: binom == ( n choose i ) && i <= k

int i = 0, binom = 1;
// initialization: binom == ( n choose 0) == 1 && 0 <= k
// => OK

while (i < k) {
   // binom == ( n choose i ) && i < k
   binom = binom * (n - i) / (i + 1);
   // binom == ( n choose i ) * (n - i) / (i + 1)
   //       == n! / (i! * (i + 1)) * (n - i) / (n - i)!
   //       == n! / (i + 1)! / (n - i - 1)!
   //       == (n choose (i + 1))
```

```
   // && i < k
   i++;
   // binom == ( n choose i ) && i <= k
}
// binom == ( n choose i ) && i <= k && i >= k
// => binom == ( n choose i ) && i == k

// result: binom == ( n choose k )

// termination: follows from
//   1) in each step: i is increasing by 1
//   2) upper bound: i <= k (by invariant)
```

Exercise 3

```
int a = x, b = y;
while (a != b) {
   if (a > b)
      a = a - b;
   else
      b = b - a;
}
int gcd = a;
```

```
// precondition: x > 0, y > 0

// invariant: gcd(x,y) == gcd(a,b) && a > 0 && b > 0

int a = x, b = y;
// initialization: gcd(x,y) == gcd(x,y) && x > 0 && y > 0
// => OK

while (a != b) {
   // gcd(x,y) == gcd(a,b) && a > 0 && b > 0 && a != b
   if (a > b)
      a = a - b;
      // gcd(x,y) == gcd(a+b,b) == gcd(a,b) by Euclides
      // && a > 0 (because of if condition) && b > 0
   else // implicitly: if (a < b)
        // no equality because of while condition
      b = b - a;
      // gcd(x,y) == gcd(a,b+a) == gcd(a,b) by Euclides
      // && a > 0 && b > 0 (because of implicit if condition)
}
// gcd(x,y) == gcd(a,b) && a == b > 0

// result: gcd(x,y) == a [ or gcd(x,y) == b ]

// termination: follows from
//   1) in each step: either a or b are decreasing
//   2) lower bound: a > 0 and b > 0 (by invariant)
```

Exercise 4

```
// precondition: n >= 0

// let b be the integer represented by array bits
// so b = sum_(0<=k<32) bits[k] * 2^k

// invariant: n == b + m * 2^i && m >= 0

int i = 0, m = n;
int[] bits = new int[32];
// initialization: n == 0 + n * 1 && n >= 0
// => OK

while (m > 0) {
   // n == b + m * 2^i && m > 0
   bits[i] = m % 2;
   m = m / 2;
   // if m is even:
   //    bits[i] = 0
   //    m = m / 2
   //    => n == b + (2*m) * 2^i == b + m * 2^(i+1)
   //       && m > 0
   // if m is odd:
   //    bits[i] = 1
   //    m = (m-1) / 2
   //    => n == b - 2^i + (2*m+1) * 2^i == b + m * 2^(i+1)
   //       && m >= 0
   // after both cases:
   // n == b + m * 2^(i+1) && m >= 0
   i++;
   // n == b + m * 2^i && m >= 0
}
// n == b + m * 2^i && m >= 0 && m <= 0
// => n == b + m * 2^i && m == 0

// result: n == b (=> bits has the right binary representation)

// termination: follows from
//   1) in each step: m is decreasing
//   2) lower bound: m >= 0 (by invariant)
```

Illustration of invariant: choose $n = 28$. At the end of the loop iteration i, we have

i	2^i	m	b	bits
1	2	14	0	{0, 0, 0, 0, 0, 0, ...}
2	4	7	0	{0, 0, 0, 0, 0, 0, ...}
3	8	3	4	{0, 0, 1, 0, 0, 0, ...}
4	16	1	12	{0, 0, 1, 1, 0, 0, ...}
5	32	0	28	{0, 0, 1, 1, 1, 0, ...}

Exercise 5: Bubble sort

Proof of inner loop:

```
// precondition: i > 0

// invariant: list[j] >= list[0..j] && j < i

int j = 0;
// initialization: list[0] >= list[0] && 0 < i => OK

while (j < i-1) {
   // list[j] >= list[0..j] && j < i-1
   if (list[j+1] < list[j]) {
      // list[j] >= list[0..j] && list[j] > list[j+1] && j < i-1
      int temp = list[j+1];
      list[j+1] = list[j];
      list[j] = temp;
      // list[j+1] >= list[0..j-1] && list[j+1] > list[j]
      // && j < i-1
   }
   // else
      // list[j+1] >= list[j] >= list[0..j] && j < i-1
   // after both cases:
   // => list[j+1] >= list[0..j+1] && j < i-1
   j++;
   // list[j] >= list[0..j] && j < i
}
// list[j] >= list[0..j] && j < i && j >= i-1
// => list[j] >= list[0..j] && j == i-1

// result: list[i-1] >= list[0..i-1]

// termination: follows from
//   1) in each step: j is increasing by 1
//   2) upper bound: j <= i-1 (by invariant)
```

Proof of outer loop:

```
// let n = list.length

// invariant: sorted(list[i..n-1])? && list[i] >= list[0..i]
              && i >= 0

int i = list.length;
// initialization: OK (as list[n] does not exist or
//                 it could be initialized as list[n] = infinity)

while (i > 0) {
   // sorted(list[i..n-1])? && list[i] >= list[0..i] && i > 0

   // inner loop:
   // precondition: i > 0 => OK
```

```
    // result: list[i-1] >= list[0..i-1]

    // => sorted(list[i..n-1])?
    //    && list[i] >= list[i-1] >= list[0..i-1] && i > 0
    // => sorted(list[i-1..n-1])?
    //    && list[i-1] >= list[0..i-1] && i > 0
    i--;
    // sorted(list[i..n-1])? && list[i] >= list[0..i] && i >= 0
}
// sorted(list[i..n-1])? && i == 0

// result: sorted(list[0..n-1])

// termination: follows from
//   1) in each step: i is decreasing by 1
//   2) lower bound: i >= 0 (by invariant)
```

Illustration of invariant: choose *list* = {24, 14, 10, 30, 45, 2}. At the end of the outer loop iteration *i*, we have

i	list
5	{14, 10, 24, 30, 2, 45}
4	{10, 14, 24, 2, 30, 45}
3	{10, 14, 2, 24, 30, 45}
2	{10, 2, 14, 24, 30, 45}
1	{2, 10, 14, 24, 30, 45}
0	{2, 10, 14, 24, 30, 45}