Lab session 6: Imperative Programming

This is the last lab of a series of weekly programming labs. You are required to solve these programming exercises and submit your solutions to the automatic assessment system *Themis*. The system is 24/7 online. Note that the weekly deadlines are very strict (they are given for the CET time zone)! If you are a few seconds late, Themis will not accept your submission (which means you failed the exercise). Therefore, you are strongly advised to start working on the weekly labs as soon as they appear. Do not postpone making the labs!
Note: You are not allowed to use the math.h library. You are free to use any constructs that were taught in this course.

Problem 0: Time Complexity

In this problem the specification constant `N` is a positive integer (i.e. `N>0`). Determine for each of the following program fragments the *sharpest upper limit* for the number of calculation steps that the fragment performs in terms of `N`. For a fragment that needs `N` steps, the correct answer is therefore $O(N)$ and not $O(N^2)$ as $O(N)$ is the sharpest upper limit. Argue in two to three sentences which option is correct per program fragment. Submit a single pdf.

1. ```
   int s = 0;
   for (int i = 1; i * i < N; i++) {
     s = s + i;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

2. ```
   int s = 0, i = 1, j = N;
   while (i < j) {
     s = s + i;
     i++;
   }
   for (int i = 0; i < s; i++) {
     j--;
   }
   ```
   (a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

3. ```
   int i, j, s = 0;
   for (i=0; i < N; i++) {
     for (j = i*i; j > 0; j = j/2) {
       s++;
     }
   }
   ```
   ```

(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

4.
```
int s = 0, i = N;
while (i) {
  i--;
  s++;
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

5.
```
int i = 0, j = N;
  while (j - i > 1) {
    if (i%2 == 0) {
      i = (i + j)/2;
    } else {
      j = (i + j)/2;
    }
  }
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

6.
```
int j;
for (int i = 1; i < 5; i++) {
  j = i;
  while (j < N / j) {
    j++;
  }
}
```
(a) $O(\log N)$ (b) $O(\sqrt{N})$ (c) $O(N)$ (d) $O(N \log N)$ (e) $O(N^2)$

Problem 1: Missing Numbers

In this exercise you are given a sequence with n numbers which optionally contains duplicates. The range of numbers is specified by the smallest and largest number in the array. Consider the sequence `2,2,1,7,5`. The smallest number is `1` and the largest number is `7` defining the range `[1,7]`. In this range, the values `3,4,6` are missing and are to be printed on the output.

Write a program that reads from the input the sequence. The end of the sequence is indicated with the terminating value `-1`. On the output you should print the number of missing numbers. Your solution **needs** to dynamically resize an array since the number of integers in the sequence is initially unknown. Points will be subtracted if your program does not support dynamic memory reallocation.

Example 1:

input:
```
2,2,1,7,5,-1
```
output:
```
3
```

Example 2:

input:
```
8,9,1,2,5,6,7,0,-1
```
output:
```
2
```

Example 3:

input:
```
1,3,3,7,8,4,-1
```
output:
```
3
```

Problem 2: Partners

In this exercise you are asked to find suitable partners for a student. Each student gets a numerical score which is used for finding a partner. The difference between the scores of two students gives an indication how well they can work together. If both students have the same score, it is considered a perfect fit. If the scores are far apart, then we consider it a poor fit. In this exercise we want to find how many suitable partners exists for a student.

Consider the following input: `5:2,2,1,7,5`. In this sequence we can see that there are a total of five students in which two students are a perfect fit (they both have the score `2`). In this case we are lucky, but it can occur that two students do not share the same score. To allow for some margin, we consider an additional input m which denotes how far apart the differences may be. With `m = 1`, a student who has a score of `2` can also pair with the student who has a score of `1` resulting in two possible partners.

Write a program that reads from the input two lines. The first line specifies the number of students n together with their scores. The second line holds two integers: the score s of a student and the margin value m. On the output you should print how many suitable partners exist for the student with score s considering a margin m, keeping in mind that students cannot partner with themselves.

Example 1:

input:
```
5:2,2,1,7,5
2 1
```
output:
```
2
```

Example 2:

input:
```
8:2,2,1,7,5,9,3,5
1 4
```
output:
```
5
```

Example 3:

input:
```
4:2,2,2,1
1 0
```
output:
```
0
```

Problem 3: Minimum Steps

Consider the following iterative process. Given two positive integers n, and g (where $0 < n \leq g$), we try to reach g starting from n, where we are allowed to iteratively use the operations `triple` ($n \to 3 \times n$), `double` ($n \to 2 \times n$), and `increment` ($n \to n + 1$). For example, let $n = 1$, and $g = 42$. We can reach 42 in 5 steps in the following way:

$$1 \to_{+1} 2 \to_{\times 3} 6 \to_{+1} 7 \to_{\times 3} 21 \to_{\times 2} 42$$

Clearly, several other possibilities exist (e.g. $1 \to_{+1} 2 \to_{+1} 3 \to_{\times 2} 6 \to_{+1} 7 \to_{\times 3} 21 \to_{\times 2} 42$). However, there is no way to reach 42 (starting in 1) with fewer than 5 steps.

Write a program that reads from the input integers n and g (where $0 < n \leq g \leq 10^8$). Your program should output the minimal number of steps that are needed to reach g starting from n.

Example 1:
input:
```
1 42
```
output:
```
5
```

Example 2:
input:
```
1 10
```
output:
```
3
```

Example 3:
input:
```
10 42
```
output:
```
3
```

Problem 4: Playing Draughts

In a game of draughts (or checkers), the goal is to capture all the pieces of an opponent. In this version you want to check how many pieces can be captured in one move. A piece can be captured if a diagonally adjacent square contains an opponent's piece and there is no piece directly beyond it on that diagonal line. After capturing a piece, it might be possible to directly capture another one if again an opponent's piece is within reach. An example of a configuration in which brown can capture four pieces in two possible ways is depicted in the figure below.

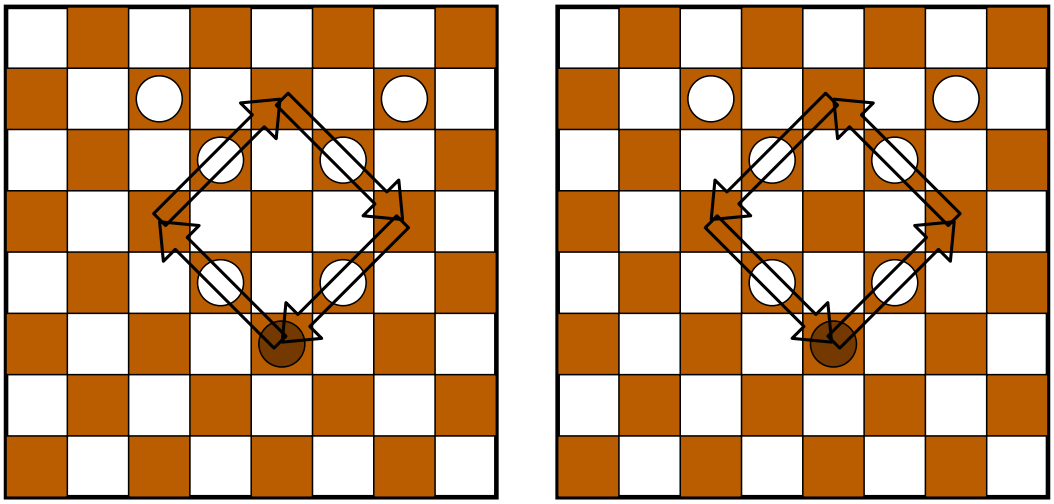

Figure 1: Capturing four white pieces with a brown piece.

Write a program that reads from the following from the input: first one line holding three numbers `x`, `y`, and `m` representing the `(x, y)`-position of the brown piece and the number `m` of white pieces. Then m lines follow holding the `(x, y)`-positions of the white pieces. Print on the output the maximum number of white pieces brown can capture in one turn. Note that the first test case refers to the board configuration depicted in Figure 1. You may assume that the board is `8 x 8`.

Example 1:

input:

```
4 5 6
2 1
6 1
3 2
5 2
3 4
5 4
```

output:

```
4
```

Example 2:

input:

```
0 7 3
1 6
1 4
0 3
```

output:

```
1
```

Example 3:

input:

```
1 6 5
0 1
0 3
2 3
4 3
2 5
```

output:

```
2
```