Chapter 2: Algorithmic Warm Up

Our goal in this chapter is too show that a clever insight about a problem may make an algorithm for this problem a billion times faster! We will consider a number of programming challenges sharing the following property: a naive algorithm for each such challenge is catastrophically slow. Together, we will design algorithms for these problems that will be as simple as the naive solutions, but much more efficient.

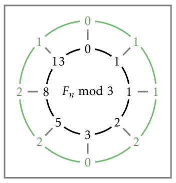
You may think that we are obsessed with the Fibonacci numbers because the first six challenges in this section represent questions about them. However, we start from seemingly similar Fibonacci problems because they gradually become more and more difficult — each new problem requires a development of a new idea. For example, after implementing the "Fibonacci Number" challenge you may be wondering what is so difficult about the "Last Digit of Fibonacci Number". However, after checking the constrains, you will learn that your implementation of the "Fibonacci Number" will take the eternity to solve the "Last Digit of Fibonacci Number", forcing you to invent a new idea!

$F_n = F_{n-1} + F_{n-2}$

Fibonacci Number
Compute the n-th Fibonacci number.
Input. An integer n.
Output. n-th Fibonacci number.

F_{100} =354224848179 261915075

Last Digit of Fibonacci Number
Compute the last digit of the n-th Fibonacci number.
Input. An integer n.
Output. The last digit of the n-th Fibonacci number.



Huge Fibonacci Number
Compute the n-th Fibonacci number modulo m.
Input. Integers n and m.
Output. n-th Fibonacci number modulo m.

1 + 1 + 2 + 3 + 5 + 8 = 20

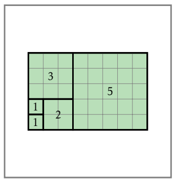
Last Digit of the Sum of Fibonacci Numbers

Compute the last digit of $F_0 + F_1 + \cdots + F_n$.
Input. An integer n.
Output. The last digit of $F_0 + F_1 + \cdots + F_n$.

2 + 3 + 5 + 8 + 13 = 31

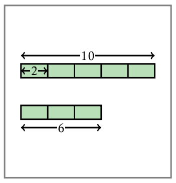
Last Digit of the Partial Sum of Fibonacci Numbers

Compute the last digit of $F_m + F_{m+1} + \cdots + F_n$.
Input. Integers $m \le n$.
Output. The last digit of $F_m + F_{m+1} + \cdots + F_n$.



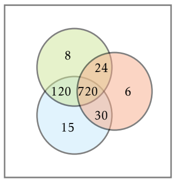
Last Digit of the Sum of Squares of Fibonacci Numbers

Compute the last digit of $F_0^2 + F_1^2 + \cdots + F_n^2$.
Input. An integer n.
Output. The last digit of $F_0^2 + F_1^2 + \cdots + F_n^2$.



Greatest Common Divisor

Compute the greatest common divisor of two positive integers.
Input. Two positive integers a and b.
Output. The greatest common divisor of a and b.



Least Common Multiple

Compute the least common multiple of two positive integers.
Input. Two positive integers a and b.
Output. The least common multiple of a and b.

2.1 Programming Challenges

2.1.1 Fibonacci Number

Fibonacci Number Problem
Compute the n-th Fibonacci number.

Input: An integer n.
Output: n-th Fibonacci number.

$$F_n = F_{n-1} + F_{n-2}$$

Fibonacci numbers are defined recursively:

$$F_n = \begin{cases} n & \text{if } n \text{ is 0 or 1} \\ F_{n-2} + F_{n-1} & \text{if } n \geq 2 \end{cases}$$

resulting in the following recursive algorithm:

```
Fibonacci($n$):
if $n \leq 1$:
   return $n$
else:
   return Fibonacci($n-2$) + Fibonacci($n-1$)
```

Input format. An integer n.

Output format. F_n.

Constraints. $0 \leq n \leq 45$.

Sample 1.

Input:

```
3
```

Output:

```
2
```

Sample 2.

Input:

```
10
```

Output:

```
55
```

Time and memory limits. When time/memory limits are not specified, we use the default values specified in Section 1.3.1.

2.1.2 Last Digit of Fibonacci Number

Last Digit of Fibonacci Number Problem
Compute the last digit of the n-th Fibonacci number.

> **Input:** An integer n.
> **Output:** The last digit of the n-th Fibonacci number.

$F_{100} = 354\,224\,848\,179\,261\,915\,075$

Input format. An integer n.

Output format. The last digit of F_n.

Constraints. $0 \leq n \leq 10^6$.

Sample 1.

Input:

```
3
```

Output:

```
2
```

$F_3 = 2$.

Sample 2.

Input:

```
139
```

Output:

```
1
```

$F_{139} = 50\,095\,301\,248\,058\,391\,139\,327\,916\,261$.

Sample 3.

Input:

```
91239
```

Output:

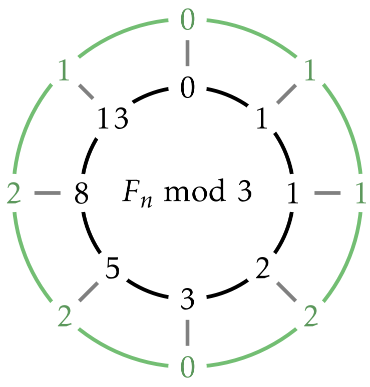
```
6
```

$F_{91\,239}$ will take more than ten pages to write down, but its last digit is 6.

2.1.3 Huge Fibonacci Number

Huge Fibonacci Number Problem
Compute the n-th Fibonacci number modulo m.

> **Input:** Integers n and m.
> **Output:** n-th Fibonacci number modulo m.



Input format. Integers n and m.

Output format. F_n mod m.

Constraints. $1 \le n \le 10^{14}$, $2 \le m \le 10^3$.

Sample 1.

Input:

```
1 239
```

Output:

```
1
```

F_1 mod $239 = 1$ mod $239 = 1$.

Sample 2.

Input:

```
115 1000
```

Output:

```
885
```

F_{115} mod $1000 = 483\,162\,952\,612\,010\,163\,284\,885$ mod $1000 = 885$.

Sample 3.

Input:

```
2816213588 239
```

Output:

```
151
```

$F_{2816213588}$ would require hundreds pages to write it down, but $F_{2816213588} \bmod 239 = 151$.

2.1.4 Last Digit of the Sum of Fibonacci Numbers

Last Digit of the Sum of Fibonacci Numbers Problem
Compute the last digit of $F_0 + F_1 + \cdots + F_n$.

> **Input:** An integer n.
> **Output:** The last digit of $F_0 + F_1 + \cdots + F_n$.

$1+1+2+3+5+8=20$

Input format. An integer n.

Output format. $(F_0 + F_1 + \cdots + F_n) \bmod 10$.

Constraints. $0 \le n \le 10^{14}$.

Sample 1.

Input:

```
3
```

Output:

```
4
```

$F_0 + F_1 + F_2 + F_3 = 0 + 1 + 1 + 2 = 4$.

Sample 2.

Input:

```
100
```

Output:

```
5
```

$F_0 + \cdots + F_{100} = 927\,372\,692\,193\,078\,999\,175$.

Hint. Since the brute force approach for this problem is too slow, try to come up with a formula for $F_0 + F_1 + F_2 + \cdots + F_n$. Play with small values of n to get an insight and use a solution for the previous problem afterward.

2.1.5 Last Digit of the Partial Sum of Fibonacci Numbers

Last Digit of the Partial Sum of Fibonacci Numbers Problem
Compute the last digit of $F_m + F_{m+1} + \cdots + F_n$.

> **Input:** Integers $m \le n$.
> **Output:** The last digit of $F_m + F_{m+1} + \cdots + F_n$.

$2 + 3 + 5 + 8 + 13 = 31$

Input format. Integers m and n.

Output format. $(F_m + F_{m+1} + \cdots + F_n) \bmod 10$.

Constraints. $0 \le m \le n \le 10^{14}$.

Sample 1.

Input:

```
3 7
```

Output:

```
1
```

$F_3 + F_4 + F_5 + F_6 + F_7 = 2 + 3 + 5 + 8 + 13 = 31$.

Sample 2.

Input:

```
10 10
```

Output:

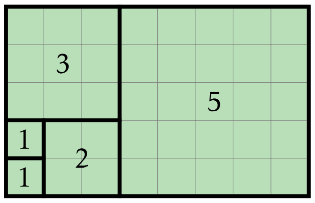
```
5
```

$F_{10} = 55$.

2.1.6 Last Digit of the Sum of Squares of Fibonacci Numbers

Last Digit of the Sum of Squares of Fibonacci Numbers Problem
Compute the last digit of $F_0^2 + F_1^2 + \cdots + F_n^2$.

> **Input:** An integer n.
> **Output:** The last digit of $F_0^2 + F_1^2 + \cdots + F_n^2$.



Hint. Since the brute force search algorithm for this problem is too slow (n may be as large as 10^{14}), we need to come up with a simple formula for $F_0^2 + F_1^2 + \cdots + F_n^2$. The figure above represents the sum $F_1^2 + F_2^2 + F_3^2 + F_4^2 + F_5^2$ as the area of a rectangle with vertical side $F_5 = 5$ and horizontal side $F_5 + F_4 = 3 + 5 = F_6$.

Input format. Integer n.

Output format. $F_0^2 + F_1^2 + \cdots + F_n^2 \bmod 10$.

Constraints. $0 \le n \le 10^{14}$.

Sample 1.

Input:

```
7
```

Output:

```
3
```

$F_0^2 + F_1^2 + \cdots + F_7^2 = 0 + 1 + 1 + 4 + 9 + 25 + 64 + 169 = 273$.

Sample 2.

Input:

```
73
```

Output:

```
1
```

$$F_0^2 + \cdots + F_{73}^2 = 1\,052\,478\,208\,141\,359\,608\,061\,842\,155\,201.$$

Sample 3.

Input:

```
1234567890
```

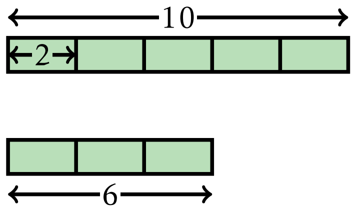
Output:

```
0
```

2.1.7 Greatest Common Divisor

Greatest Common Divisor Problem
Compute the greatest common divisor of two positive integers.

> **Input:** Two positive integers a and b.
> **Output:** The greatest common divisor of a and b.



The greatest common divisor $\text{GCD}(a,b)$ of two positive integers a and b is the largest integer d that divides both a and b. The solution of the Greatest Common Divisor Problem was first described (but not discovered!) by the Greek mathematician Euclid twenty three centuries ago. But the name of a mathematician who discovered this algorithm, a century before Euclid described it, remains unknown. Centuries later, Euclid's algorithm was re-discovered by Indian and Chinese astronomers. Now, the efficient algorithm for computing the greatest common divisor is an important ingredient of modern cryptographic algorithms.

Your goal is to implement Euclid's algorithm for computing GCD.

Input format. Integers a and b (separated by a space).

Output format. $\text{GCD}(a,b)$.

Constraints. $1 \le a, b \le 2 \cdot 10^9$.

Sample.

Input:

```
28851538 1183019
```

Output:

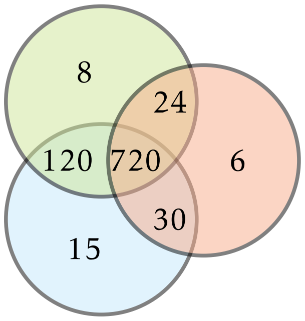
```
17657
```

$28851538 = 17657 \cdot 1634$, $1183019 = 17657 \cdot 67$.

2.1.8 Least Common Multiple

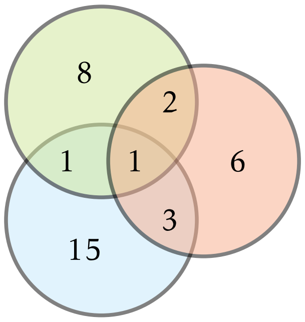
Least Common Multiple Problem
Compute the least common multiple of two positive integers.

> **Input:** Two positive integers a and b.
> **Output:** The least common multiple of a and b.



The least common multiple $\text{LCM}(a, b)$ of two positive integers a and b is the smallest integer m that is divisible by both a and b.

The figure above shows the LCM for each pair of numbers 6, 8, and 15 as well as the LCM for all three of them. The figure below shows the GCD for the same numbers.



Stop and Think. How $\text{LCM}(a, b)$ is related to $\text{GCD}(a, b)$?

Input format. Integers a and b (separated by a space).

Output format. $\text{LCM}(a, b)$.

Constraints. $1 \le a, b \le 10^7$.

Sample.

Input:

```
761457 614573
```

Output:

```
467970912861
```

2.2 Summary of Algorithmic Ideas

Now when you've implemented several algorithms, let's summarize and review the main ideas that we've discussed in this chapter.

What solution fits into a second? Modern computers perform about 10^8–10^9 basic operations per second. If you program contains a loop with n iterations and n can be as large as 10^{14}, it will run for a couple of days. In turn, this means that for a programming challenge where a naive solution takes as many steps, you need to come up with a different idea.

Working with large integers. If you need to compute the last digit of an integer m, then, in many cases, you can avoid computing m explicitly: when computing it, take every intermediate step modulo 10. This will ensure that all intermediate values are small enough so that they fit into integer types (in programming languages with integer overflow) and that arithmetic operations with them are fast.