

A competitive programmer’s Handbook



Welcome to the ultimate journey into the world of Competitive Programming and Data Structures. Whether you're a beginner or looking to sharpen your skills, this course is designed to equip you with the tools, techniques, and mindset to solve complex problems efficiently.

Let's dive into logic, algorithms, and beyond!

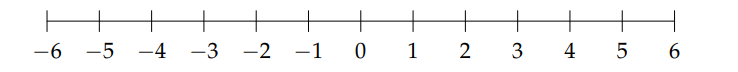
# 2 Modular Arithmetic

When first learning division, we often encounter the idea of remainders. For instance, dividing 7 by 3 gives a quotient of 2 with a remainder of 1. More generally, when dividing a number by another number , we obtain a quotient and a remainder , satisfying the equation:

where 0 ≤ r < n. This concept forms the foundation of modular arithmetic.

2.1.2 Intuition

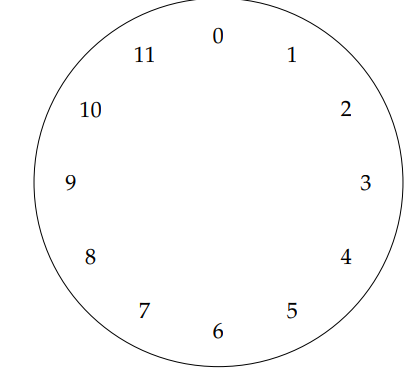
Imagine a number line.



As we’re taught in school, we can count on this number line. Starting at 0, we can count up 1, 2, 3, 4, . . . or down −1, −2, −3, −4, . . . .

But now imagine taking a piece of this number line and wrapping it around in a circle.





Think of modular arithmetic like a clock with **modulus 12**—after reaching 11, we loop back to 0. Counting forward or backward follows the same rule. In this system, adding or subtracting **12 hours** brings us back to the same position, making it a useful model for integer arithmetic.

Question: If a 12-hour clock starts at 0, where will the hour hand be after 2,025 hours?

2.1.2 Modular Operation

residue

Modulus

This operation finds the remainder when is divided by , which is always in the range . The key to efficient modular arithmetic is understanding how the basic operations of addition, subtraction, and multiplication work over a given modulus:

* Addition— What is (*x* + *y*) mod *n*? We can simplify this to

to avoid adding big numbers. Eg,. How much small change will I have if given $123.45 by my mother and $94.67 by my father?

Convert to cents:

* 123.45→ **12,345 cents**
* 94.67 → **9,467 cents**

(12*,* 345 mod 100) + (9*,* 467 mod 100) = (45 + 67) mod 100 = 12 mod 100

After receiving the money, your small change (cents left) will be **12 cents**.

* Subtraction— Subtraction is just addition with negative values. How much small change will I have after spending $52.53 from $218.12?

(12 mod 100) *−* (53 mod 100) = *−*41 mod 100 = 59 mod 100

After spending, you have 59 cents left.

* Multiplication — Since multiplication is just repeated addition,

Convert **$17.28** to **1,728 cents** and multiply by **2,143 hours**. Instead of direct multiplication, apply modular reduction:

Multiply the reduced values:

( 28 x 43 ) mod 100 = (1204) mod 100 = 4

Final small change: **$0.04** (or) 4 cents

* Division in modular arithmetic is more complex than addition, subtraction, and multiplication. Most problems you encounter won’t require direct modular division, so we’ll skip it for now. However, if you're a 3rd-year student studying Cryptography, you might already know how tedious this topic can be!

2.1.3 Finding Last Digit

What is the last digit of ? What we really want to know is . By doing repeated squaring, and taking the remainder mod 10 at each step we make progress very quickly:

1. Base Step :

1. Squaring Progressively :

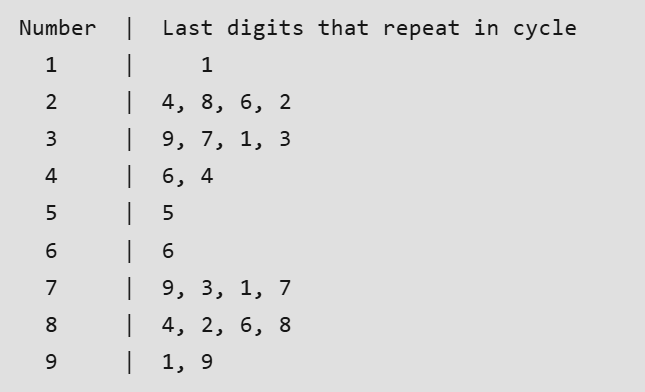
1. Final Computation :

Thus, the last digit of 6.

How to find the last digit of any large integer ?

For large exponentiations like computing the full number is impractical. However, the last digit follows a repeating cycle, allowing us to determine it efficiently without full computation

There occurs a repeating pattern of last digit for each . Which is shown in this table :



Observing the Pattern

The table in the image shows how the last digits of numbers repeat in cycles:

* **Example: Base 2** → Cycle: **2, 4, 8, 6**

Each base has a cycle of length **at most 4**.

Algorithm :

Since large values of are impractical to compute directly, we use modular arithmetic:

1. Extract the last digit of (since only this affects the result).
2. Compute to determine the effective exponent:
   * If , use exponent **4** (full cycle length).
   * Otherwise, use as the exponent.
3. Use the table to find the last digit in the cycle for the given base and exponent.

**Example Calculation**

Find the last digit of :

1. Compute , so we use exponent **4**.
2. The cycle for 7 is **(7, 9, 3, 1)**.
   * The 4th number is **1**.
3. So, the last digit of is 1.

Code Template :

C++

#include <bits/stdc++.h>

using namespace std;

// Function to find b % a

int Modulo(int a, char b[]) {

int mod = 0;

for (int i = 0; i < strlen(b); i++)

mod = (mod \* 10 + b[i] - '0') % a;

return mod;

}

// Function to find last digit of a^b

int LastDigit(char a[], char b[]) {

int len\_a = strlen(a), len\_b = strlen(b);

if (len\_a == 1 && len\_b == 1 && b[0] == '0' && a[0] == '0') return 1;

if (len\_b == 1 && b[0] == '0') return 1;

if (len\_a == 1 && a[0] == '0') return 0;

int exp = (Modulo(4, b) == 0) ? 4 : Modulo(4, b);

int res = pow(a[len\_a - 1] - '0', exp);

return res % 10;

}

Python

# Function to find b % a

def modulo(a, b):

mod = 0

for digit in b:

mod = (mod \* 10 + int(digit)) % a

return mod

# Function to find last digit of a^b

def last\_digit(a, b):

if a == "0" and b == "0":

return 1

if b == "0":

return 1

if a == "0":

return 0

exp = 4 if modulo(4, b) == 0 else modulo(4, b)

res = (int(a[-1]) \*\* exp) % 10

return res

Practice Problems :

1. [Find the last digit of .](https://codeforces.com/problemset/problem/742/A)
2. [Determine the final position after moving **b** steps around a circular sequence of **n** elements, starting from position **a**.](https://codeforces.com/problemset/problem/659/A)
3. [Find a^b .](https://leetcode.com/problems/super-pow/description/)
4. [Minimum coins.](https://www.codechef.com/problems/MINCOINS)

2.1.3 Fast Exponentiation

Code :

long long modExp(long long base, long long exp, long long m) {

if (exp == 0) return 1;

long long result = 1;

while (exp > 0) {

// If the current bit of exp is 1, multiply result by base

if (exp % 2 != 0)

result = (result \* base) % m;

// Square the base and reduce modulo m

base = (base \* base) % m;

// Move to the next bit of exp

exp = exp / 2;

}

return result;

}

Explanation :

1. **Binary Representation of n**: The loop processes each bit of n.
2. **Odd Exponent**: If the current bit of n is 1 (odd), multiply the result (res) by a and take modulo.
3. **Square and Reduce**: Square the base (a) and reduce it modulo mod in every iteration.
4. **Shift Exponent**: Divide n by 2 (right shift) to move to the next bit.

Practice Problems :

1. [Power of 2.](https://leetcode.com/problems/power-of-two/description/)
2. [Double modular exponentiation.](https://leetcode.com/problems/double-modular-exponentiation/description/)
3. [Modular exponentiation with description.](https://www.hackerrank.com/contests/yhxh9bxejzv/challenges/cs5800hw1modularexponentation)
4. [Another variant of modular exponentiation.](https://codeforces.com/problemset/problem/913/A)