

# Introducción

---

Miguel Ortiz

**Programación competitiva para ICPC**

Octubre 2023

- Ingeniería Informática en la Universidad Mayor de San Simón
- Experiencia en competencias ICPC desde 2016
- Clasificado a la final mundial ICPC 2022
- Experiencia dando clases de programación competitiva a estudiantes de colegio y universidad
- [github.com/MickyOR](https://github.com/MickyOR) ← aquí pueden encontrar las diapositivas
- [t.me/MickyOr](https://t.me/MickyOr)

# ¿Qué es la programación competitiva?

- Es un deporte mental
  - Deporte: *Actividad física, usualmente competitiva y organizada, que tiene la meta de usar, mantener o mejorar las capacidades y habilidades físicas.*
- Se resuelven problemas lógicos
- Diseño de algoritmos
- Trabajo en equipo (ICPC)



# Ventajas

- Se practica programación
- Se mejora el razonamiento algorítmico
- Cierra la brecha entre la teoría y la practica
- Más oportunidades de trabajo
  - Entrevistas de trabajo
  - Pasantías en empresas de software

Dado un problema, queremos:

- Resolverlo eficientemente, usando algoritmos y estructuras de datos
- Convertir la solución a un programa
- Hacerlo lo más rápido posible (bajo presión)
- Hacerlo correctamente (sin bugs)

# ¿Cómo lo haremos?

- Estudiando categorías comunes de problemas
- Aprendiendo algoritmos y estructuras de datos
- Aprendiendo otros conceptos importantes para resolver problemas
- Practicando resolver problemas
- Practicando más
- ¡Practicando aún más!

# Requisitos para el curso

- Variables
- Lectura/escritura de datos por consola
- Bucles (`for`, `while`)
- Condicionales (`if/else`)
- Operadores lógicos
- Operadores aritméticos
- Funciones
  - Recursividad (solo para un par de temas, no es 100% necesario)
- Arreglos

# Programa

No. de clase	Fecha	Temas/actividad
1	02.10	Introducción, análisis de complejidad, estructuras de datos no lineales ya implementadas
2	04.10	Grafos
3	06.10	Búsqueda binaria
4	09.10	Matemáticas
5	11.10	Programación dinámica
6	13.10	Estructuras de datos para consultas en rangos

Horario: 20:30 - 22:30



- Clases teóricas:
  - Resolución de problemas del tema anterior
  - Explicación de temas
- Competencias:
  - Duran 46 horas, desde el final de una clase hasta el inicio de la siguiente
  - Problemas relacionados con el último tema
  - Pueden participar individualmente o en equipos
  - Se puede participar de forma individual

# Introducción a competencias

---

- Problemas típicos en competencias de programación
- Usualmente consisten de:
  - Descripción del problema
  - Descripción del input
  - Descripción del output
  - Ejemplo de un caso de prueba (input y output)
  - El tiempo límite en segundos
  - La memoria límite en MB o kB
- Te dicen que escribas un programa que resuelva el problema para todos los inputs válidos
- El programa no debe exceder los límites de tiempo o memoria

# Ejemplo de un problema

## Descripcion del problema

Escriba un programa que multiplique pares de enteros.

## Input

El input empieza con una línea que contiene un entero  $T$ , donde  $1 \leq T \leq 100$ , denotando el número de casos de prueba. Luego  $T$  líneas siguen, cada una conteniendo un caso de prueba. Cada caso de prueba consiste de dos enteros  $A, B$ , donde  $-2^{20} \leq A, B \leq 2^{20}$ , separados por un espacio.

## Output

Por cada caso de prueba, imprima una línea conteniendo el valor de  $A \times B$ .

## Ejemplo de un problema

Input de ejemplo	Output de ejemplo
4 3 4 13 0 1 8 100 100	12 0 8 10000

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- ¿La solución es correcta?

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- ¿La solución es correcta?
- ¿Y si  $A = B = 2^{20}$ ?



# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- ¿La solución es correcta?
- ¿Y si  $A = B = 2^{20}$ ? El output es 0...

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        int A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- ¿La solución es correcta? ¡No!
- ¿Y si  $A = B = 2^{20}$ ? El output es 0...

## Ejemplo de solución

- Cuando  $A = B = 2^{20}$ , la respuesta debería ser  $2^{40}$

## Ejemplo de solución

- Cuando  $A = B = 2^{20}$ , la respuesta debería ser  $2^{40}$
- Muy grande para un entero de 32 bits, overflow

## Ejemplo de solución

- Cuando  $A = B = 2^{20}$ , la respuesta debería ser  $2^{40}$
- Muy grande para un entero de 32 bits, overflow
- Un entero de 64 debería ser suficiente

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        long long A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        long long A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- ¿La solución es correcta?

# Ejemplo de solución

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cin >> T;

    for (int t = 0; t < T; t++) {

        long long A, B;
        cin >> A >> B;

        cout << A * B << endl;
    }

    return 0;
}
```

- ¿La solución es correcta? ¡Si!



# Errores comunes

```
#include <iostream>
using namespace std;

int main() {
    int T;
    cout << "Ingrese el numero de casos de prueba: ";
    cin >> T;

    for (int t = 0; t < T; t++) {

        long long A, B;
        cout << "Ingrese dos enteros: ";
        cin >> A >> B;

        // Verificar limites  $-2^{20} \leq A, B \leq 2^{20}$ 
        if (-(1<<20) <= A && A <= (1<<20) &&
            -(1<<20) <= B && B <= (1<<20)) {

            cout << "La respuesta es: ";
            cout << A * B << endl;

        }

    }

    return 0;
}
```

# Jueces virtuales

- De los jueces más populares:
  - Codeforces
  - AtCoder
  - Kattis
  - Online Judge (ex UVa)
- Se envían las soluciones a los jueces virtuales y se obtiene feedback de inmediato
- Se puede enviar en cualquier lenguaje soportado:
  - C
  - C++
  - Java
  - Python 2
  - Python 3
  - C#
  - y otros

# Veredictos

- El feedback sobre las soluciones es limitado
- Usualmente se recibe alguno de los siguientes veredictos:
  - Accepted
  - Wrong Answer
  - Compile Error
  - Run Time Error
  - Time Limit Exceeded
  - Memory Limit Exceeded
- No se revela cuáles son los casos de prueba que se usan para probar la solución

## Tips para competencias

---

# Escribir más rápido

- Buenos competidores tienen soluciones simples; no tienen que escribir tanto, pero es importante escribir rápido
- A veces es inevitable tener que escribir mucho código
- Typing.com tiene un curso de mecanografía gratuito:
  - **<https://www.typing.com/student/lessons>**
- TypeRacer es una forma divertida y efectiva de practicar:
  - **<https://play.typeracer.com/>**
- Monkytype es una alternativa con más opciones de configuración:
  - **<https://monkeytype.com/>**

# Análisis de algoritmos

- Al resolver un problema, nuestra solución debe ser lo suficientemente rápida y no puede usar demasiada memoria
- También queremos que nuestra solución sea lo más simple posible
- Debemos analizar el algoritmo para determinar si una solución se ejecutará dentro del tiempo límite
- Regla general:  $10^8$  operaciones por segundo

# Análisis de algoritmos

- Al resolver un problema, nuestra solución debe ser lo suficientemente rápida y no puede usar demasiada memoria
- También queremos que nuestra solución sea lo más simple posible
- Debemos analizar el algoritmo para determinar si una solución se ejecutará dentro del tiempo límite
- Regla general:  $10^8$  operaciones por segundo
- Queremos ordenar  $n \leq 10^6$  enteros, y tenemos 3 segundos.
  - ¿Podemos usar un simple bubble sort?  $O(n^2)$
  - ¿Qué hay de un merge sort?  $O(n \log n)$

# Análisis de algoritmos

- Al resolver un problema, nuestra solución debe ser lo suficientemente rápida y no puede usar demasiada memoria
- También queremos que nuestra solución sea lo más simple posible
- Debemos analizar el algoritmo para determinar si una solución se ejecutará dentro del tiempo límite
- Regla general:  $10^8$  operaciones por segundo
- Queremos ordenar  $n \leq 10^6$  enteros, y tenemos 3 segundos.
  - ¿Podemos usar un simple bubble sort?  $O(n^2)$
  - ¿Qué hay de un merge sort?  $O(n \log n)$
- Queremos ordenar  $n \leq 10^3$  enteros, y tenemos 3 segundos.
  - ¿Podemos usar un simple bubble sort?  $O(n^2)$



# Análisis de algoritmos

- Al resolver un problema, nuestra solución debe ser lo suficientemente rápida y no puede usar demasiada memoria
- También queremos que nuestra solución sea lo más simple posible
- Debemos analizar el algoritmo para determinar si una solución se ejecutará dentro del tiempo límite
- Regla general:  $10^8$  operaciones por segundo
- Queremos ordenar  $n \leq 10^6$  enteros, y tenemos 3 segundos.
  - ¿Podemos usar un simple bubble sort?  $O(n^2)$
  - ¿Qué hay de un merge sort?  $O(n \log n)$
- Queremos ordenar  $n \leq 10^3$  enteros, y tenemos 3 segundos.
  - ¿Podemos usar un simple bubble sort?  $O(n^2)$
- Siempre ir por la solución más simple que no sobrepase el tiempo límite

- A veces no estás seguro si tu solución es correcta
- ¡Intenta *demostrar* su correctitud!
- Incluso si no logras demostrarlo o desmentirlo, probablemente tendrás una mejor comprensión del problema
- Libro gratuito sobre demostraciones: [Book of Proof](#)

# Aprende tu lenguaje de programación

- Lo *ideal* es conocer el lenguaje de programación como la palma de tu mano
- Esto incluye sus librerías:
  - C++ — Standard Template Library
  - Java — Class Library
- Si el lenguaje ya lo tiene implementado, no es necesario hacerlo de nuevo

# Prueba tu solución

- Verificar que la solución es correcta y que corre dentro del tiempo límite
- O ver que no es correcta aunque no sepas por qué
- **Trata de romper tu solución** encontrando un contraejemplo (una entrada para la cual tu solución da una salida incorrecta, o tarda demasiado en calcular una respuesta)
- Trata con casos borde, input grande, etc.

# Práctica y más práctica

- La habilidad de resolver problemas viene con la práctica
- Muchas veces se resuelven problemas encontrando patrones (el problema es similar a otro que ya resolviste)
- Muchos jueces online tienen problemas de competencias pasadas
- Algunos de estos jueces organizan competencias frecuentemente
- Codeforces, AtCoder, Codechef, LeetCode, TopCoder, etc.

# Formato de competencias

- Contests duran 2 días
- Participación individual o en equipo
- Necesitan una cuenta en [vjudge.net](https://vjudge.net)