

Introducción a la Programación Competitiva

Inicial

Matías Ramos

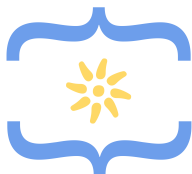
Universidad Tecnológica Nacional – Facultad Regional Santa Fe

Training Camp 2026



Gracias Sponsors!

Organiza



Diamond Sponsor



Facultad de
INFORMÁTICA



UNIVERSIDAD
NACIONAL
DE LA PLATA

Outline

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

como competidor



Porto 2019



Anarap (Egipto '24)



Fruta Fresca (México '24)



Fruta Fresca (Kazajistán '24)



Champan en Lata (Brasil '25)



Champan en Lata (Chile '26)

Outline

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

¿Qué es un problema?

Problema de ejemplo:

Juan tiene $0 \leq X \leq 90$ manzanas, y quiere repartirlas entre $0 \leq Y \leq 10$ amigos.

Suponiendo que quiere maximizar la cantidad de manzanas que reparte, y que todos reciban la misma cantidad, ¿cuántas manzanas recibiría cada uno?

Input:

2

15 2

17 5

Output:

7

5

Outline

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones**
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

¿Cuántas operaciones entran en tiempo?

Podemos dividir en tres secciones:

- $\leq 10^8$ **operaciones**: Siempre entran en 1 segundo.
- **Entre 10^8 y 10^9 operaciones**: Puede funcionar, dependerá principalmente de qué hace cada operación.
- $> 10^9$ **operaciones**: **PELIGRO**: casi siempre lleva a TIME LIMIT.

Notación Big O ($\mathcal{O}$)

¿Qué es y para qué sirve?

- Nos permite estimar de manera abstracta el **tiempo de ejecución** (o memoria) de un algoritmo en el **peor caso**.
- Expresa cómo crece el número de operaciones en función del tamaño de la entrada (N).
- **Regla fundamental:** Despreciamos constantes y términos de menor orden.
 - Ejemplo: $3N^2 + 5N + 10 \Rightarrow \mathcal{O}(N^2)$

Ejemplos intuitivos:

- $\mathcal{O}(1)$: Acceso directo en un vector.
- $\mathcal{O}(\log N)$: Búsqueda binaria (dividir el problema a la mitad).
- $\mathcal{O}(N)$: Recorrer un arreglo con un solo bucle.

¿Qué complejidad usar según los límites del problema?

Para un límite de tiempo de 1 segundo ($\approx 10^8$ operaciones):

Complejidad	Límite típico de N	Algoritmo / Ejemplo
$\mathcal{O}(1)$	10^{18}	Fórmulas matemáticas, acceso directo
$\mathcal{O}(\log N)$	10^{18}	Búsqueda binaria
$\mathcal{O}(N)$	10^8	Bucle simple, Two pointers
$\mathcal{O}(N \log N)$	10^6	Ordenamiento (std::sort), Segment Tree
$\mathcal{O}(N^2)$	5000	Bucles anidados, DP simple
$\mathcal{O}(N^3)$	500	Multiplicación de matrices
$\mathcal{O}(2^N)$	20	Backtracking, fuerza bruta
$\mathcal{O}(N!)$	10	Permutaciones de un arreglo

Los tipos de datos tienen sus límites también:

- `int` $\Rightarrow$ desde -2^{31} hasta $2^{31} - 1$ ($\approx \pm 2 \cdot 10^9$).
- `long long` $\Rightarrow$ desde -2^{63} hasta $2^{63} - 1$ ($\approx \pm 9 \cdot 10^{18}$).

Al superar esos límites, llegamos al famoso `OVERFLOW`.

¿Cómo evitar el overflow?

El overflow es un error **MUY COMÚN**.

El uso de **aritmética modular** (A.K.A. operaciones MOD) es fundamental:

- `%` (MOD): devuelve el resto de una división.
- ADD mod: $(a + b) \bmod m$
- SUB mod: $((a - b) \bmod m) + m \bmod m$
- MUL mod: $(a \cdot b) \bmod m$

Recomendaciones:

- Usar tipos de 64 bits: `long long` en C++.
- Es mejor usar MOD de más que de menos.

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles**
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

Operaciones comunes:

- `sort(v.begin(), v.end(), comparador);`

```
vector<int> v;  
sort(v.begin(), v.end());
```

- `lower_bound(v.begin(), v.end(), x);` (requiere orden)
- `set.lower_bound(x);` (asi se usa en sets)
- `upper_bound();`

Estructuras importantes

Hay algunas estructuras que son **muy importantes**:

- vector
- queue
- deque
- set
- map

- `vector<T>` en C++, con `push_back` y `pop_back`
- `ArrayList<T>` en Java, con `add` y `remove(list.size()-1)`
- `list` en Python, con `append` y `pop`
- Acceso con `v[pos]` o `v.get(pos)`
- Todas estas operaciones son **$O(1)$**

- `queue<T>` en C++, con `push`, `front`, `pop`
- `ArrayDeque<T>` en Java, con `add`, `getFirst`, `remove`
- `collections.deque` en Python, con `append`, `deque[0]`, `popleft`
- Se comporta como una cola (FIFO)
- Operaciones en **$O(1)$**

- `deque<T>` en C++, con `push_front`, `push_back`, `pop_front`, `pop_back`
- `ArrayDeque<T>` en Java, con `addFirst`, `addLast`, `removeFirst`, `removeLast`
- `collections.deque` en Python, con `appendleft`, `append`, `popleft`, `pop`
- Acceso en **$O(1)$**
- Todas las operaciones anteriores son también **$O(1)$**

- `set<T>` en C++
- `TreeSet<T>` en Java
- En Python no hay un equivalente ordenado nativo
- Mantiene elementos **únicos y ordenados**
- Permite insertar, borrar, consultar existencia, `lower_bound`, `upper_bound`
- Todas en **$O(\log N)$**
- Variantes:
 - `unordered_set` (C++) / `HashSet` (Java): $O(1)$ amortizado
 - `multiset`: permite elementos repetidos

- `map<K, V>` en C++
- `TreeMap<K, V>` en Java
- En Python: `collections.OrderedDict` (aproximado)
- Mantiene pares **clave** $\rightarrow$ **valor**
- Acceder, insertar, borrar en **$O(\log N)$**
- Variante: `unordered_map` (hash, $O(1)$ amortizado)

Outline

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum**
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

Prefix Sum

Dado un vector V , queremos poder calcular rápidamente la suma de los primeros X elementos, o de cualquier subarreglo.

Para eso construimos un **vector auxiliar** P de tamaño $n + 1$, donde:

$$P[i] = V[0] + V[1] + \dots + V[i - 1]$$

Ejemplo:

$$V = [2, 6, 2, 5, 10]$$

$$P = [0, 2, 8, 10, 15, 25]$$

Entonces, la suma del subarreglo $V[l .. r]$ se puede calcular en $\mathcal{O}(1)$ como:

$$P[r + 1] - P[l]$$

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida**
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

Entrada y salida en C++

Forma elegante de entrada/salida:

```
using namespace std;
int main(){
    int a,b;
    cin >> a >> b;
    cout << a << " " << b << '\n'; // << endl; (es mas lento)
}
```

Leer y escribir consume tiempo, por lo tanto hay que tener cuidado.

Recomendaciones en C++:

```
int main() {  
    ios::sync_with_stdio(false); // no mezclar con scanf/printf  
    cin.tie(nullptr); // no espera cout antes de leer  
    int a, b;  
    cin >> a >> b;  
    cout << a << " " << b << '\n';  
}
```

Outline

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial**
- 8 Consejos personales

El Rol Permitido: La IA como Tutor

Fuera de las competencias oficiales, usá la IA como un **"socio de depuración"** y no como un generador de soluciones.

A. Para comprender el problema (sin "spoilers"):

- **Traducción y aclaración:** Excelente para interpretar enunciados complejos en inglés técnico o reformular enunciados ambiguos.
- **Modo Socrático:** Pedí pistas, no la solución directa.
- **Prompt recomendado:**

Prompt Clave

"Explícame este problema de programación competitiva con otras palabras y dame una pista (hint) matemática o algorítmica, pero no me digas el algoritmo final ni me des código."

B. Para identificar errores (Debugging efectivo):

- **Buscar el contraejemplo:** Si tenés Wrong Answer (WA) en un testcase oculto, en vez de pedir "corregí mi código", pasale el enunciado y tu código e indicá:
 - *"¿Qué caso borde (edge case) o desbordamiento de enteros (integer overflow) se me está escapando en este enfoque?"*
- **Generación de Casos de Prueba (Stress Testing):**
 - Pedile un script para generar casos de prueba aleatorios o extremos (valores máximos, mínimos, arreglos vacíos).
 - Compará tu solución rápida que falla contra una lenta pero segura (fuerza bruta).

La Línea Roja: Lo que Prohíben las Plataformas

Es vital conocer las reglas actuales para evitar baneos innecesarios (ej. en Codeforces):

- **Prohibición de Lógica:** No se permite introducir el enunciado para que la IA devuelva el algoritmo exacto o el código a usar.
- **Prohibición de Refactorizar en Competencia:** No se puede copiar el código que falló en un testcase del sistema durante el contest y pedirle a la IA que lo arregle.
- **Uso Limitado de Copilot/Cursor:** Permitido en práctica únicamente para sintaxis repetitiva o boilerplate. Nunca para delegar la lógica central del problema.

La Analogía del Entrenador

Un buen entrenador te enseña a levantar la barra y te corrige la postura (pistas, explicaciones), pero **si levanta la barra por ti (te da el código funcional), tus músculos no crecen.**

Regla de Oro:

- Usá la IA para entender por qué falla un enfoque o desmenuzar enunciados en la fase de práctica.
- Nunca la uses para saltarte la frustración del diseño lógico: **la frustración es el proceso de aprendizaje.**
- En la compe presencial (como ICPC) no hay internet ni IA; necesitás desarrollar tu propia intuición algorítmica.

Outline

- 1 Presentacion
- 2 ¿Qué es un problema?
- 3 Operaciones
- 4 Estructuras útiles
- 5 Prefix Sum
- 6 Entrada y salida
- 7 Uso de Inteligencia Artificial
- 8 Consejos personales

Consejos personales

- Usar una tablita para anotar ideas por problema

A	B	C	D	E	F	G	H	I	J	K	L	M	N
			contar swaps en algoritmo de ordenamiento loco						la resp es siempre < 30 , y probablemente bastante menor inclusive, tiene pinta que sale con dp o bfs	grafo completo grande pintado en dos colores, te dan el coloreo de un color, contar una porqueria, no parece facil		feo, pero capaz no tan dificil. ESPERA PC	nano
		si pudiera preguntar el mas frecuente en $[0, N]$ - $[L, R]$ + $[L, R]$ para todo L, R ganaria		grafos		math		para mi es - si ves una monedita, aumentas la siguiente posicion con la que chocarias arriba, y con la que chocarias abajo, pero antes queda e viador de alguna de las dos. Si usa			tiene pinta de greedy, pero no es tan trivial como parece		fede
PC	Dado un grafo completo con pesos G , llevarlo a un grafo completo con pesos H , ejecutando una operacion la cantidad de veces que quieras. (no es necesario		imprimir la secuencia y ver algun patron???????						espera PC				juampi

- Aprovechar simulaciones para mejorar la dinámica
- Mantener una buena comunicación con el equipo

Consejos personales

- Tener un template útil y propio

```
#include <bits/stdc++.h>
#define forn(i,a,b) for(int i=(a);i<(b);i++)
#define forn(i,n) forn(i,0,n)
#define dforn(i,n) for(int i=n-1;i>=0;i--)
#define forall(it,v) for(auto it=v.begin();it!=v.end();it++)
```

- Acostumbrarse a usar un notebook
- Aprender a estimar tiempos, antes de resolver

- Asegurarse de entender bien el enunciado, comprobar viendo los ejemplos
- Intentar hacer al menos una demo minima de la solucion
- Leer editoriales → entender el **cómo**, no solo el código
- Usar IA como herramienta, no como solución (ver sección anterior para más detalle)

- **UPSOLVEAR**, esa es la clave

FedeNQ	B - In Case of an Invasion, Please...	C++20 (GCC 13-64)	Accepted	733 ms
FedeNQ	B - In Case of an Invasion, Please...	C++20 (GCC 13-64)	Wrong answer on test 19	718 ms
Fruta Fresca: marianoferesin, FedeNQ , Aristides	B - In Case of an Invasion, Please...	C++20 (GCC 13-64)	Wrong answer on test 10	2999 ms
Fruta Fresca: marianoferesin, FedeNQ , Aristides	B - In Case of an Invasion, Please...	C++20 (GCC 13-64)	Compilation error	0 ms

- No quedarse con un solo problema, leer otros.
- Intentar resolver casos mas simples, ver como mejorar la solucion.
- Si los problemas no salen, aprovechen a tomar algunos de los problemas propuestos en las clases.
- No tengan miedo de preguntar a los profes/organizadores, les podemos dar una ayudita.
- Disfrutar, tanto las competencias como los entrenamientos

Problemas Propuestos

Para practicar los temas de hoy (hacer clic en el título para abrir, seleccionar el texto para ver la pista):

- [Weird Algorithm:](#)
- [Missing Number:](#)
- [Increasing Array:](#)
- [Distinct Numbers:](#)
- [Static Range Sum Queries:](#)
- [Ferris Wheel:](#)
- [Concert Tickets:](#)

Pueden crearse una cuenta en cses.fi para mandar sus soluciones.

Gracias por la atención.

Suerte en la compe de hoy y en los próximos días.

Cualquier duda o consulta, tienen mi mail o me buscan por telegram
@somartam