



# Competitive Programming Workshop: Introducción

Giovanny Alfonso Chávez Cenicerros

Universidad Autónoma de Chihuahua  
Facultad de Ingeniería

*[gchavezcenicerros@acm.org](mailto:gchavezcenicerros@acm.org)*

20 de marzo de 2019

# Contenido

## 1 Introducción

- ¿Qué es Programación Competitiva?
  - Suma de dos números

## 2 Implementando algoritmos

- Máximo producto de pares en una secuencia
- ¿Como saber si un algoritmo es mejor que otro?
  - Análisis asintótico
  - Notación Big- $O$
  - Crecimiento asintótico

## 3 Calentamiento algorítmico

- Máximo Común Divisor
- Números de Fibonacci

## 4 Referencias

# ¿Qué es Programación Competitiva?

- Es un deporte mental en el cual los participantes resuelven un conjunto de problemas bien especificados a través de programas de computadora.
- Los estudiantes universitarios, principalmente de ingeniería y carreras asociadas a ciencias de la computación, participan en las competencias organizadas por ACM ICPC.

## ¿Qué es Programación Competitiva?

Todos los algoritmos implementados serán sujetos a las siguientes limitaciones:

### Límite de tiempo

<b>C</b>	<b>C++</b>	<b>Java</b>	<b>Python</b>	<b>Haskell</b>	<b>Javascript</b>
1s	1s	1.5s	5s	2s	5s

### Límite de memoria

512 Mb.

¿Qué es Programación Competitiva?

## Comandos de compilación en C/C++

En Linux y MacOS, lo más probable es que tengas el compilador requerido. En Windows, puedes usar tu compilador favorito o instalar, por ejemplo, cygwin.

C (gcc 5.4.0). extensión: .c

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (g++ 5.4.0). extensión: .cc, .cpp

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

¿Qué es Programación Competitiva?

# Suma de dos números

## Descripción

Calcular la suma de dos números de un sólo dígito.

## Entrada

Dos enteros  $a$  y  $b$  en la misma línea (separados por un espacio).

## Salida

La suma de  $a$  y  $b$ .

## Restricciones

$0 \leq a, b \leq 9$ .

¿Qué es Programación Competitiva?

# Implementando el algoritmo en C++

```
1  #include <bits/stdc++.h>
2
3  int main( void )
4  {
5      unsigned int  a, b;
6      std::cin >> a >> b;
7      std::cout << a + b << std::endl;
8      return 0;
9  }
```

# Máximo producto de pares en una secuencia

	5	6	2	7	4
5		30	10	35	20
6	30		12	42	24
2	10	12		7	4
7	35	42	14		28
4	20	24	8	28	

## Descripción

Dada una secuencia de enteros no negativos  $a_1, \dots, a_n$ , calcular  $\max a_i \cdot a_j$  donde  $1 \leq i \neq j \leq n$ .

Tenga en cuenta que  $i$  y  $j$  deben ser diferentes, aunque puede ser el caso de que  $a_i = a_j$ .



# Máximo producto de pares en una secuencia

## Entrada

La primer línea contiene un entero  $n$ . La siguiente línea contiene  $n$  enteros no negativos  $a_1, \dots, a_n$  (separados por espacios).

## Salida

El producto por pares máximo.

# Máximo producto de pares en una secuencia

## Restricciones

$$2 \leq n \leq 2 \cdot 10^5$$

$$1 \leq a_1, \dots, a_n \leq 2 \cdot 10^5.$$

## Casos de prueba

Entrada	Salida
3	
1 2 3	6

# Construyendo el código

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  vector<unsigned long> M; // almacenar la secuencia
4
5  int main( void )
6  {
7      unsigned int  n;
8      unsigned long a;
9      cin >> n; // Capturar la primera linea
10     while(n--) // Capturar la segunda linea
11     {
12         cin >> a;
13         M.push_back(a);
14     }
15     // producto por pares máximo
16     cout << maxPairwise( ) << endl;
17     return 0;
18 }
```

## Implementando un algoritmo ingenuo:

```
1 unsigned long maxPairwise( void )
2 {
3     unsigned long m = 0;
4     for(unsigned int i = 0; i < M.size( ); ++i)
5         for(unsigned int j = 0; j < M.size( ); ++j)
6             if(i != j)
7                 if(M[i]*M[j] > 0)
8                     m = M[i]*M[j];
9     return m;
10 }
```

## Implementando un algoritmo optimizado:

```
1 unsigned long maxPairwiseFast( void )
2 {
3     unsigned int n = M.size( ) - 1;
4     sort(M.begin( ), M.end( ));
5     return M[n - 1]*M[n];
6 }
```

¿Como saber si un algoritmo es mejor que otro?

## ¿Como saber si un algoritmo es mejor que otro?

- El tiempo de ejecución de un algoritmo depende de cuánto tiempo le tome a una computadora ejecutar las líneas de código, y eso depende de la velocidad de la computadora, el lenguaje de programación y el compilador que traduce el programa, entre otros factores.
- Debemos enfocarnos en qué tan rápido crece un algoritmo respecto al tamaño de la entrada. A esto lo llamamos la tasa de crecimiento del tiempo de ejecución.

¿Como saber si un algoritmo es mejor que otro?

# Análisis asintótico

Para analizar el tiempo de ejecución de un algoritmo se emplean las siguientes notaciones:

## Notación Big- $O$

$$\exists k, n_0 > 0 : \forall n \geq n_0, T(n) \leq kf(n) \implies T(n) = O(f(n))$$

## Notación $\Omega$

$$\exists k, n_0 > 0 : \forall n \geq n_0, T(n) \geq kf(n) \implies T(n) = \Omega(f(n))$$

## Notación $\Theta$

$$\exists k, n_0 > 0 : \forall n \geq n_0, kf(n) \leq T(n) \leq kf(n) \implies T(n) = \Theta(f(n))$$

¿Como saber si un algoritmo es mejor que otro?

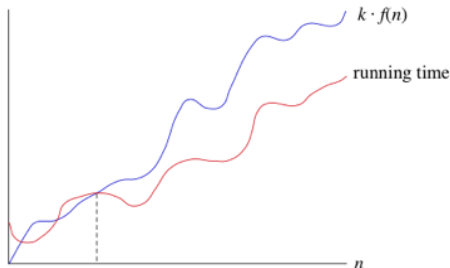
# Notación Big- $O$

- Usamos estas notaciones para acotar de manera asintótica el crecimiento de un tiempo de ejecución que esté dentro de factores constantes por arriba y por abajo.
- A veces queremos acotar solo por arriba, por ello usamos la notación Big- $O$  justo para estas ocasiones.

¿Como saber si un algoritmo es mejor que otro?

# Notación Big- $O$

Si un tiempo de ejecución  $T(n)$  es  $O(f(n))$ , entonces para  $n$  suficientemente grande,  $T(n)$  es a lo más  $kf(n)$  para alguna constante  $k$ .







¿Como saber si un algoritmo es mejor que otro?

# Crecimiento asintótico

Big- $O$	Nombre	Ejemplo
$O(1)$	constante	un número es par o impar
$O(\lg n)$	logarítmico	búsqueda binaria
$O(n)$	lineal	máximo de un arreglo desordenado
$O(n \lg n)$	<i>linealítmico</i>	ordenamiento con <i>mergesort/quicksort</i>
$O(n^2)$	cuadrático	operaciones con matrices
$O(n^k)$	polinomial	multiplicación de matrices
$O(2^n)$	exponencial	subconjuntos de un conjunto
$O(!n)$	factorial	permutaciones

# Máximo Común Divisor

$$\begin{aligned} & \text{GCD}(1344, 217) \\ &= \text{GCD}(217, 42) \\ &= \text{GCD}(42, 7) \\ &= \text{GCD}(7, 0) \\ &= 7 \end{aligned}$$

## Descripción

El máximo común divisor  $\text{GCD}(a, b)$  de dos enteros no negativos  $a$  y  $b$  es el mayor entero  $d$  que divide  $a$  y  $b$ . El objetivo es implementar el algoritmo euclidiano para el cálculo del máximo común divisor.

# Máximo Común Divisor

## Entrada

Dos enteros  $a$  y  $b$  en la misma línea (separados por un espacio).

## Salida

El  $GCD(a, b)$ .

# Máximo Común Divisor

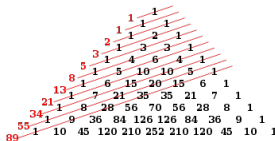
## Restricciones

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

## Casos de prueba

Entrada	Salida
18 35	1
28851538 1183019	17657

# Números de Fibonacci



En matemáticas, la sucesión o serie de Fibonacci hace referencia a la secuencia ordenada de números descrita por Leonardo de Pisa, matemático italiano del siglo XIII. La sucesión se define como  $F_0 = 0$ ,  $F_1 = 1$ , y  $F_i = F_{i-1} + F_{i-2}$  para  $i \geq 2$ . El objetivo es implementar un algoritmo eficiente para el cálculo de números de Fibonacci.

# Números de Fibonacci

Podríamos pensar en una implementación recursiva para calcular los números de Fibonacci:

```
1 unsigned int fibo( unsigned long n )
2 {
3     if(n == 0 || n == 1)
4         return n;
5     else
6         return fibo(n - 1) + fibo(n - 2);
7 }
```

Pero pronto veríamos que el cálculo de, por ejemplo,  $F_{40}$  ya lleva un tiempo considerable.

# Fibonacci simple

## Descripción

Dado un entero  $n$ , calcular el  $n$ -ésimo número de Fibonacci  $F_n$ .

## Entrada

Un entero  $n$ .

## Salida

El  $n$ -ésimo número de Fibonacci  $F_n$ .



# Fibonacci simple

## Restricciones

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

## Casos de prueba

Entrada	Salida
10	55
45	1134903170

# Algoritmo de Fibonacci optimizado

```
1  unsigned int fibo( unsigned long n )
2  {
3      unsigned long memo[n + 1];
4
5      memo[0] = 0;
6      memo[1] = 1;
7
8      for(int i = 2; i <= n; ++i)
9          memo[i] = (memo[i - 1] + memo[i - 2]);
10
11     return memo[n];
12 }
```

# Último dígito del $n$ -ésimo número de Fibonacci

## Descripción

Dado un entero  $n$ , encuentre el último dígito del  $n$ -ésimo número de Fibonacci  $F_n$  (es decir,  $F_n \bmod 10$ ).

## Entrada

Un entero  $n$ .

## Salida

$F_n \bmod 10$ .

## Restricciones

$0 \leq n \leq 10^7$ .

# Último dígito del n-ésimo número de Fibonacci

## Casos de prueba

Entrada	Salida
200	5

$$F_{200} = 280571172992510140037611932413038677189525.$$

Entrada	Salida
331	9

$$F_{331} = 668996615388005031531000081241745415306766 \\ 517246774551964595292186469.$$

