# Ayudantía 1

## ¿Que es programación competitiva?



La Programación Competitiva es un deporte mental, en el que los participantes se sumergen en la emoción de resolver problemas desafiantes con especificaciones concretas en un tiempo limitado.

# ¿De qué me sirve la programación competitiva?

La programación competitiva te ayuda a profundizar en temas avanzados de matemáticas y algoritmos, te brinda la habilidad de implementar soluciones complejas y entrena tu pensamiento lógico.



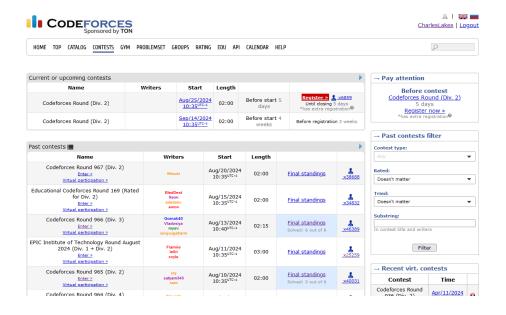
## ¿De qué me sirve en el futuro profesional?

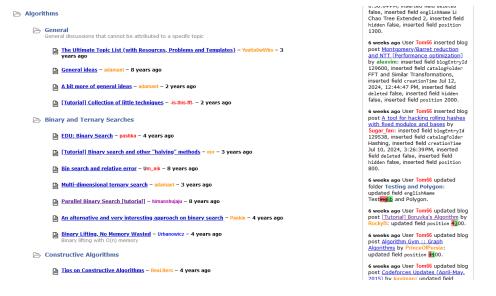
Muchas grandes empresas requieren superar pruebas técnicas, que a menudo incluyen la resolución e implementación de algoritmos.

Además, las empresas buscan personas capaces de crear e implementar soluciones a problemas complejos, por lo que esta habilidad es fundamental.

## ¿Dónde puedo aprender más sobre programación

La página más activa y con una amplia cantidad de problemas, recursos y tutoriales es Codeforces. En esta plataforma se realizan competencias con frecuencia, y la comunidad crea tutoriales para aprender temas avanzados.





# ¿Que competencias existen?





# ¿Que lenguajes de programación se usan?

### Top

- 1. C++
- 2. Python
- 3. Java

## ¿Por qué C++?

- Es un lenguaje rápido en comparación con Python u otros de alto nivel.
- La biblioteca estándar de C++ (STL) ofrece una amplia gama de funcionalidades y estructuras de datos.
- Permite programar soluciones eficientes y versátiles gracias a la STL.
- En mi experiencia con programación competitiva, nunca he necesitado usar punteros.

### Estructura básica de un programa

```
// Esta directiva incluye todas las bibliotecas estándar del lenguaje.
// Aunque no es óptima para desarrollar software complejo,
// en programación competitiva nos ahorra mucho tiempo.
#include <bits/stdc++.h>
using namespace std;
int main() {
    cout << "Hola mundo" << endl;</pre>
    return 0;
```

#### Estructuras de Datos - Vector

### ¿Cómo usar arreglos dinámicos sin punteros?

Podemos utilizar el contenedor vector , implementado en la STL de C++.

Se define de la siguiente manera:

vector<tipo\_de\_dato> nombre\_de\_variable;

#### Estructuras de Datos - Vector

```
vector<int> arr1; // Arreglo vacío de ints
vector<int> arr2(10); // Arreglo inicializado con 10 elementos
vector<int> arr3(8, -1); // Arreglo inicializado con 8 elementos, asignando -1 a todas las posiciones
vector<vector<string>> arr5; // Vector que almacena vectores de string
```

```
arr1.push_back(10); // Inserta 10 al final en O(1)
arr2.pop_back(); // Elimina el último elemento en O(1)
arr2[3] = 15; // Asigna 15 en la posición 3
arr2.clear(); // Elimina todos los elementos del vector
cout << arr3.size(); // Devuelve el tamaño del vector</pre>
```

#### Estructuras de Datos - Vector

```
// Recorrer el vector usando un bucle for tradicional
for(int i = 0; i < arr1.size(); i++){
    cout << arr1[i] << endl;
}
// Recorrer el vector usando un foreach
for(int numero : arr1){
    cout << numero << endl;
}</pre>
```

### Estructuras de Datos - Queue

Una queue (cola) es un contenedor que sigue el principio FIFO (First In, First Out).

Se define de la siguiente manera:

queue<tipo\_de\_dato> nombre\_de\_variable;

### Estructuras de Datos - Queue

```
queue<int> cola;
```

```
cola.push(20); // Inserta el valor 20 al final de la cola
cout << cola.front() << endl; // Imprime el valor en el frente de la cola
cola.pop(); // Elimina el elemento al frente de la cola
cout << cola.size() << endl; // Devuelve la cantidad de elementos en la cola
cout << (cola.empty() ? "Cola vacía" : "Cola no vacía") << endl; // Verifica si la cola está vacía</pre>
```

### Estructuras de Datos - Stack

Una stack (pila) es un contenedor que sigue el principio LIFO (Last In, First Out).

Se define de la siguiente manera:

stack<tipo\_de\_dato> nombre\_de\_variable;

#### Estructuras de Datos - Stack

```
stack<int> pila;
```

```
pila.push(30); // Inserta el valor 30 en la pila
cout << pila.top() << endl; // Imprime el valor en el tope de la pila
pila.pop(); // Elimina el elemento en el tope de la pila

cout << pila.size() << endl; // Devuelve la cantidad de elementos en la pila
cout << (pila.empty() ? "Pila vacía" : "Pila no vacía") << endl; // Verifica si la pila está vacía</pre>
```

### Estructuras de Datos - Set

Un set es un contenedor que almacena elementos únicos en orden específico.

Se define de la siguiente manera:

set<tipo\_de\_dato> nombre\_de\_variable;

#### Estructuras de Datos - Set

```
set<int> conjunto; // Orden de menor a mayor
set<int,greater<int>> alreves; // Orden de menor a mayor
```

```
conjunto.insert(15); // Inserta el valor 15 en el set
conjunto.insert(20); // Inserta el valor 20 en el set
conjunto.insert(10); // Inserta el valor 10 en el set
cout << conjunto.size() << endl; // Devuelve la cantidad de elementos en el set
cout << (conjunto.empty() ? "Set vacío" : "Set no vacío") << endl; // Verifica si el set está vacío
// Imprimir todos los elementos del set
for (int elemento : conjunto) {
    cout << elemento << " "; // Imprime cada elemento
}
cout << endl;
// Eliminar un elemento
conjunto.erase(20); // Elimina el elemento 20 del set</pre>
```

### Estructuras de Datos - Map

Un map es un contenedor que almacena pares clave-valor ordenados por las claves.

Se define de la siguiente manera:

map<tipo\_de\_clave, tipo\_de\_valor> nombre\_de\_variable;

### Estructuras de Datos - Map

```
map<string, int> mapa;
```

```
mapa["manzanas"] = 10; // Asigna el valor 10 a la clave "manzanas"
mapa["naranjas"] = 5; // Asigna el valor 5 a la clave "naranjas"

cout << mapa["manzanas"] << endl; // Imprime el valor asociado a la clave "manzanas"
mapa.erase("naranjas"); // Elimina el par clave-valor con la clave "naranjas"

cout << mapa.size() << endl; // Devuelve la cantidad de elementos en el map
cout << (mapa.empty() ? "Mapa vacío" : "Mapa no vacío") << endl; // Verifica si el map está vacío

// Iteración sobre el map usando un foreach
for (auto par : mapa) {
    cout << "Clave: " << par.first << ", Valor: " << par.second << endl; // Imprime cada clave y su valor asociado
}</pre>
```

### Otros Tipos de Datos - Pair

Un pair es una estructura que puede contener dos valores de diferentes tipos. Se usa comúnmente para almacenar pares de datos relacionados, como coordenadas o claves y valores.

Definición y uso:

```
pair<tipo_de_dato1, tipo_de_dato2> nombre_de_variable;
pair<int, string> par;

// Inicializar un par
par = make_pair(1, "uno");

// Acceso a elementos
cout << par.first << endl; // Imprime el primer elemento (1)
cout << par.second << endl; // Imprime el segundo elemento ("uno")</pre>
```

### Otros Tipos de Datos - Tuple

Una tuple es similar a un pair, pero puede contener múltiples valores de diferentes tipos. Esto es útil cuando necesitas agrupar más de dos valores.

Definición y uso:

```
#include <tuple>
tuple<tipo_de_dato1, tipo_de_dato2, tipo_de_dato3> nombre_de_variable;
tuple<int, string, double> tupla;

// Inicializar una tupla
tupla = make_tuple(1, "uno", 3.14);

// Acceso a elementos
cout << get<0>(tupla) << endl; // Imprime el primer elemento (1)
cout << get<1>(tupla) << endl; // Imprime el segundo elemento ("uno")
cout << get<2>(tupla) << endl; // Imprime el tercer elemento (3.14)</pre>
```

La función sort es utilizada para ordenar elementos en un contenedor. Esta función es parte de la librería  $\mbox{\ensuremath{\ensuremath{\ensuremath$ 

#### Ordenar en Orden Ascendente

```
vector<int> numeros = {5, 2, 9, 1, 5, 6};

// Ordenar el vector en orden ascendente
sort(numeros.begin(), numeros.end());

// Imprimir el vector ordenado
cout << "Vector ordenado: ";
for (int num : numeros) {
    cout << num << " ";
}
cout << endl;</pre>
```

#### Ordenar en Orden Descendente

Para ordenar en orden descendente, puedes utilizar la función greater<int>() como tercer argumento:

```
sort(numeros.begin(), numeros.end(), greater<int>());

cout << "Vector ordenado en orden descendente: ";
for (int num : numeros) {
    cout << num << " ";
}
cout << endl;</pre>
```

#### Ordenar con una Función de Comparación Personalizada

Puedes definir una función de comparación personalizada para ordenar elementos de acuerdo con un criterio específico. A continuación, se muestra un ejemplo de cómo ordenar un vector de pares de enteros según el segundo elemento de cada par:

```
#include <bits/stdc++.h>
using namespace std;
bool compararPorSegundo(pair<int, int> a, pair<int, int> b) {
    return a.second < b.second;</pre>
int main() {
    vector<pair<int, int>> pares = {{1, 4}, {2, 2}, {3, 3}};
    sort(pares.begin(), pares.end(), compararPorSegundo);
    cout << "Vector de pares ordenado por el segundo elemento: ";</pre>
    for (pair<int,int> par : pares) {
        cout << "(" << par.first << ", " << par.second << ") ";</pre>
    cout << endl;</pre>
    return 0;
```