Complejidad, Arreglos, Matrices y Trucos

Club de Algoritmia CUCEI - Club de Básicos

Tipos de datos

Tipo	Rango	En palabras	Bytes
bool	0 o 1	true o false	1
char	-127 a 127	cien	1
short	-3e4 a 3e4	10 mil	2
int	-2e9 a 2e9	mil millones	4
long long	-9e18 a 9e18	millón de billones	8
float	1e-38 a 3e38	de casi nada a un montón	4
double	2e-308 a 1e308	lo mismo pero mucho más	8
long double	3e-4932 a 1e4932	la neta ni se cuanto es eso	16

¡Cuidado con los decimales!

Los números decimales no pueden ser representados correctamente por binarios.

```
double x = 0.3*3+0.1;
printf("%.20f\n", x); // 0.99999999999999988898
```

Para comparar la igualdad de dos decimales a y b se utiliza:

```
if (abs(a-b) < 1e-9) {
    // a and b are equal
}
```

Ciclos

```cpp
for (int i = 0; i < n; ++i) {
  // Código
}

for (int i = 1; i <= n; ++i) {
   // Código
}

for (int i = n; i > 0; --i) {
   // Código
}

for (int i = n - 1; i >= 0; --i) {
   // Código
}
```

```cpp
int i = 0;
while (i < n) {
   // Código
   ++i;
}

int i = 0;
do {
   // Código
   ++i;
} while (i < n);

while (n--) {
   // Código
}
```

Funciones

En lugar de hacer esto

```
int main() {
   double a, b, c, x;
   a = 9;
   b = 12;
   c = 25;
   x = (-b + sqrt(b*b - 4*a*c))/(2*a);

   a = 54;
   b = 21;
   c = 77;
   x = (-b + sqrt(b*b - 4*a*c))/(2*a);
}
```

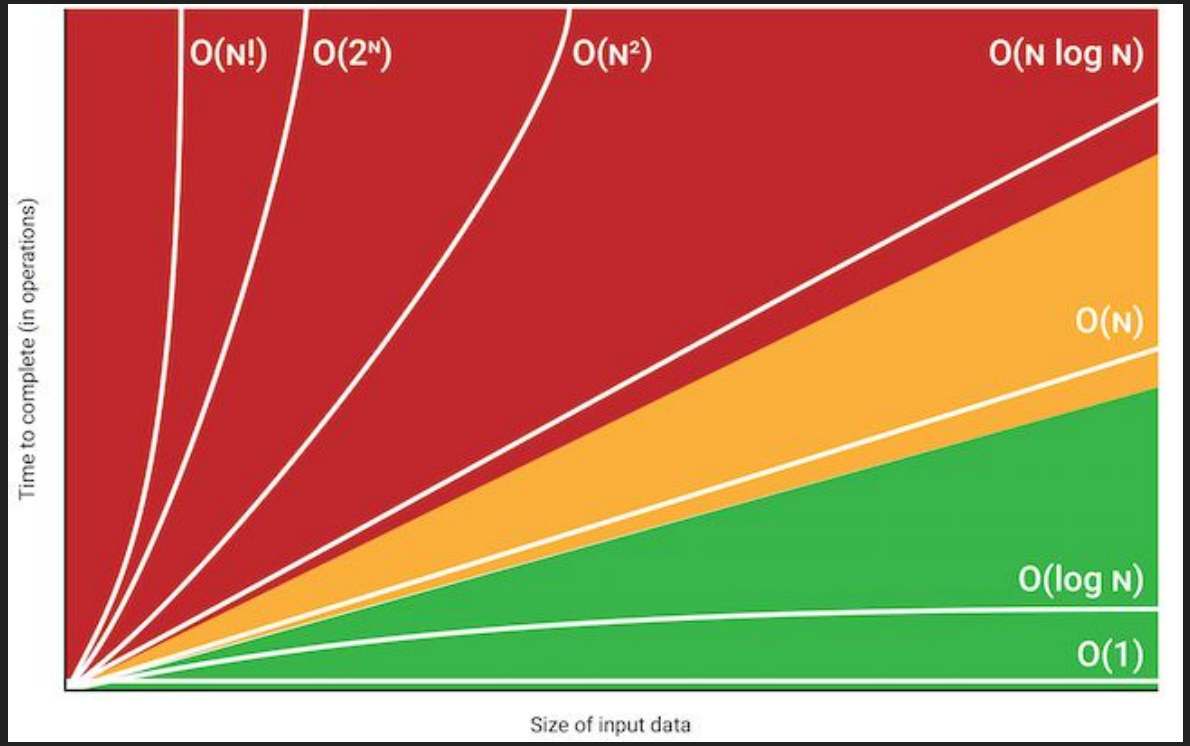
Hacer esto

```
double chicharronera(double a, double b, double c) {
   return (-b + sqrt(b*b - 4*a*c))/(2*a);
}

int main() {
   double x;
   x = chicharronera(9, 12, 25);
   x = chicharronera(54, 21, 77);
}
```

Complejidad

¿Cómo sé si un algoritmo es mejor que otro?

Para qué se usa



Medir el crecimiento del tiempo de ejecución en relación al tamaño del input (n).

Problema: Jorge y el concurso

https://omegaup.com/arena/problem/Jorge-y-el-concurso/#problems

Ejemplos de complejidad

```
for (int i = 0; i < n; ++i)
    {
        //O(n)
    }

for (int i = 0; i < n; ++i)
    {
        for (int j = 1; j <= n; ++j)
        {
            // O(n^2)
        }
    }
```

```
for (int i = 0; i < n; i += 2) {
   // O(n)
}

for (int i = 0; i < n; i++) {
  for (int j = i; j < n; ++j) {
      // O(n^2)
   }
}

   for (int i = 0; i < n; i++) {
       for (int j = n; j > 0; j--) {
           for (int k = 0; k < n * 2; k += 3)
{
               // O(n^3)
           }
       }
   }
}
```

Ejemplos de complejidad

```
while (n /= 2) {
   // O(log n)
}

for (int i = 0; i < n; ++i) {
   int j = n;
   while (j >= 0) {
       // O(n log n)
       j /= 2;
   }
}

for (int i = 0; i < n; i /= 3) {
   // O(log3 n)
}
```

```
for (int i = 0; i < 10000; i += 35) {
   // O(1)
}

for (int i = 0; i < A; ++i)
    {
        for (int j = 0; j < B; ++j)
        {
      // O(AB) o O(A^2) si ambos
      tienen el mismo límite de
      entrada
        }
    }
{
      for (int i = 0; i < A; ++i) { }
      for (int j = 0; j < B; ++j) { }
      // O(A+B) o O(A) si ambos tienen
      el mismo límite de entrada
}
```

Big O, Theta y Omega

Podemos definir Big O como el peor caso posible.

$$O(n)$$

Omega sería el mejor caso posible.

$$\Omega(n)$$

Theta sería el caso promedio.

$$\Theta(n)$$

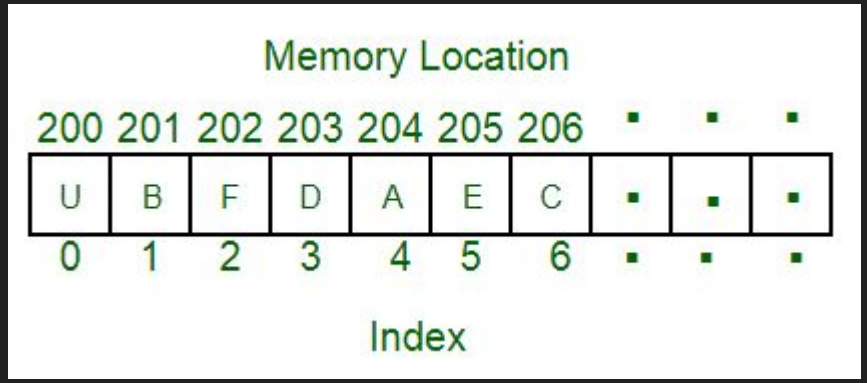
Normalmente $O(n) > \Omega(n)$, pero puede ser que $O(n) = \Omega(n) = \Theta(n)$

Arreglos y matrices

Contenedores estáticos

Cómo se almacenan en memoria

Los arreglos se almacenan en memoria de manera secuencial, permiten tener acceso a sus elementos en orden constante O(1).



Cómo iterar (Recorrer)

Forma convencional

```
int n = 10;
int a[n] = {3, 6, 8, 1, 5, 1, 6, 7, 8}; // Arreglo

for (int i = 0; i < n; ++i) {
   a[i] += 2; // A cada elemento el sumo 2
}

for (int i = 0; i < n; ++i) {
   cout << a[i] << '\n'; // Muestro contenido
}

// A = {5, 8, 10, 3, 7, 3, 8, 9, 10}
```

Forma rápida

```
int n = 100;
int a[n] = {3, 6, 8, 1, 5, 1, 6, 7, 8}; // Arreglo

// La palabra “auto” deduce el tipo de dato
// El “&” es para poder modificar los elementos
for (auto &x : a) {
   x += 2; // A cada elemento el sumo 2
}

// No necesito “&” porque no voy a modificar x
for (auto x : a) {
   cout << x << '\n';
}
// A = {5, 8, 10, 3, 7, 3, 8, 9, 10}
```

Copiar elementos de un arreglo a otro

```cpp
int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    int otherArr[10];

    // Copea uno por uno de arr a otherArr
    for (int i = 0; i < 10; ++i)
    {
        otherArr[i] = arr[i];
    }

    cout << "otherArray (For Loop):" << endl;
    for (auto e : otherArr)
        cout << e << ",";

    cout << endl;
```

```cpp
int arr[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    int otherArr2[10];

    // Copea uno por uno de arr a otherArr
    // Mucho más rápido de escribir y más eficiente
    copy(arr, arr + 10, otherArr2);

    cout << "otherArray2 (Copy):" << endl;
    for (auto e : otherArr2)
        cout << e << ",";

    cout << endl;
```

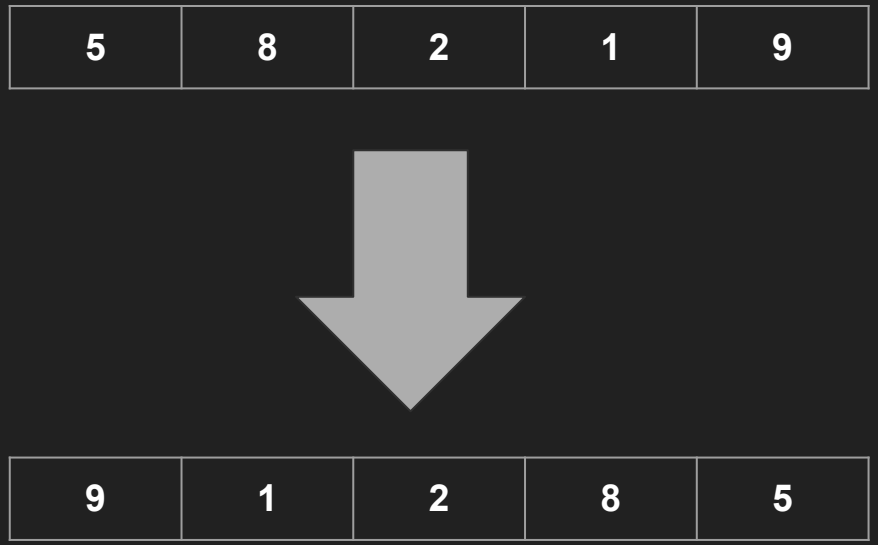
Arreglos dentro o fuera del main?

Los arreglos declarados dentro de funciones incluida la función main son alojados en el stack, mientras que los arreglos declarados fuera de las funciones, es decir, en el scope global son alojados en el heap que es mucho más grande que el stack, además de que tiene la capacidad de crecer.

En castellano: Fuera de funciones puedes guardar más elementos :D!

Problema: Revertir un arreglo

Se te proporciona un número **N**, que es la cantidad de elementos que contiene tu arreglo, después se te proporcionan **N** números los cuales forman parte del arreglo. Tu misión es imprimir el arreglo en orden opuesto a como se te fue dado.



Solución propuesta: Revertir un arreglo

Input:
5
5 8 2 1 9

Output:
9 1 2 8 5

```
int n;
cin >> n;
int arr[n];
for (int i = 0; i < n; ++i)
        cin >> arr[i];

for (int i = n - 1; i >= 0; --i)
        cout << arr[i] << " ";
```

Problema: Querys de actualización y consulta

Tienes **n** aldeas, cada i-th aldea tiene una *economía local* **EL**. Sin embargo, la *economía regional* **ER** de una aldea se ve determinada por la suma de las economías locales adyacentes más la suya.

Tu sistema recibe dos tipos de instrucciones:
- 0 para actualizar la **EL** de la aldea **a**.
- 1 para consultar la **ER** de la aldea **a**.

Tu sistema recibe **n** aldeas y **m** instrucciones.
Después **n** números de la **EL** de cada aldea.
Seguido de **m** líneas con ya sea:

0 a EL o **1 a**

Ejemplo:

Input:

```
5 4
10 2 6 9 1
1 3
0 4 4
1 3
1 1
```

Output:

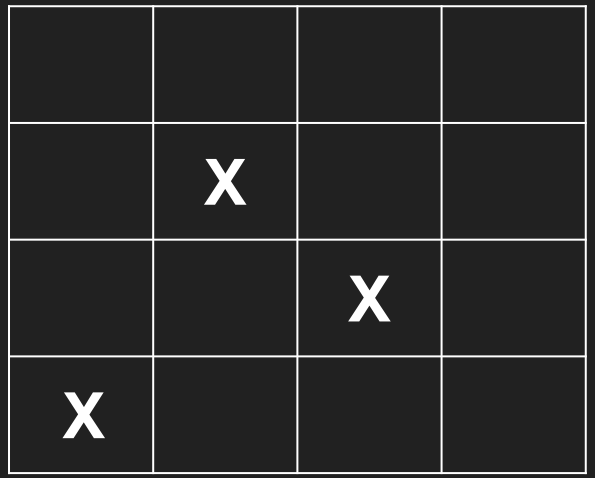
```
16
19
18
```

Solución propuesta: Querys de actualización y consulta

```cpp
int main() {
    int n, m;
    cin >> n >> m;

    int arr[n];
    for (int i = 0; i < n; ++i)
        cin >> arr[i];

    int instrucion, aldea;
    while (m--) {
        cin >> instrucion >> aldea;

        if (instrucion)  // instruccion == 1
            cout << arr[aldea-1] + arr[aldea] + arr[aldea+1] << '\n';

        else             // instruccion == 0
            cin >> arr[aldea];
    }
}
```

Problema: Buscaminas

Eres el encargado de arreglar buscaminas, un desarrollador un poco torpe dañó los contadores de bombas. Dada una matriz N * N, deberás determinar cuántas bombas tocan una casilla, las bombas serán representadas con una **X.** Tu respuesta será imprimir la matriz con los contadores de bombas en perfecto estado.



Solución propuesta: Buscaminas

Input:
4 3
4 1
2 2
3 3

https://pastebin.com/giM2NvJQ

Output:
1110
1X21
23X1
X211

Problema: El tronco más largo

Tienes una cámara full-hd 4k y le tomas una foto a unos troncos flotando en un río.

Como eres experto en IA de reconocimiento de imágenes, quieres encontrar el tamaño del tronco más largo.

Entrada:
n y m de la resolución en pixeles de la foto.
Una matriz de n x m de 0 y 1 donde el 0 es agua y el 1 es el tronco.

Todos los troncos son verticales de anchura 1.
Solo hay un tronco por columna.

Ejemplo:

Input:

```
5 8
00000000
01001010
01000010
01000010
00000010
```

Output:

```
4
```

Solución propuesta 1: El tronco más largo

```cpp
int main() {
    int n, m;
    cin >> n >> m;

    char arr[n][m];
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> arr[i][j];
    int maxi = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (arr[i][j] == '1') {
                int tam = 0;
                for (int k = i; k < n && arr[k][j] == '1'; ++k) {
                    arr[k][j] = '0';
                    tam++;
                }
                maxi = max(tam, maxi);
            }
        }
    }
    cout << maxi << '\n';
}
```

Solución propuesta 2: El tronco más largo

```cpp
int main() {
    int n, m;
    cin >> n >> m;

    char arr[n][m];
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            cin >> arr[i][j];

    int maxi = 0;
    for (int j = 0; j < m; ++j) {
        int tam = 0;
        for (int i = 0; i < n; ++i)
            if (arr[i][j] == '1')
                tam++;
        maxi = max(maxi, tam);
    }
    cout << maxi << '\n';
}
```

Funciones que deberías conocer

```
// Funciones matemáticas básicas

double a = 4, b = 7, c = 0;

c = max(a, b); // c = 7
c = min(a, b); // c = 4
c = sqrt(a);   // c = 2
c = pow(a, b); // c = 4^7 = 16,384
// Nota: Para elevar al cuadrado es más eficiente a*a que pow(a, 2);

c = abs(a - b); // c = 3
c = round(5.8); // c = 6
```

```
// Funciones de arreglos

int n = 10;
int arr1[n] = {0, 1, 3, 0, 4, 5, 7, 3, 5, 4};
int arr2[n] = {5, 6, 8, 1, 2, 0, 2, 3, 1, 0};
copy(arr1, arr1 + n, arr2); // Copia arr1 en arr2
// arr2 =  {5, 6, 8, 1, 2, 0, 2, 3, 1, 0}

int arr3[n] = {9, 9, 9, 9, 9, 9, 9, 9, 9, 9};
copy(arr1 + 2, arr1 + 5, arr3); // Copia solo desde arr1[2] hasta arr1[4]
// arr3 = {3, 0, 4, 9, 9, 9, 9, 9, 9, 9}

fill(arr1, arr1 + n, 0); // Cambia todos los valores de arr1 a 0
// arr1 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Plantilla genérica

```cpp
#include <bits/stdc++.h> // Todas las librerías en una
using namespace std;     // Ahorra escribir std::cin, std::cout, etc

// Mis #define, constantes o macros
#define ENDL '\n'

// Mis funciones
void funcion() {

}

int main() {
    // Vuelve más rápido cin y cout pero debo usar '\n' en lugar de endl
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    // Código fregón
    cout << "A partir de aquí programas" << ENDL;
    return 0;
}
```