Sorting, greedy y búsqueda binaria

Sorting (ordenamiento)

Para ordenar un vector podemos usar la función `std::sort`

Para ordenar un vector podemos usar la función `std::sort`

```
vector <int> a = {5, 2, 6, 2, 1, 8};
sort(a.begin(), a.end());
// a queda como {1, 2, 2, 5, 6, 8}
```

Sorting

Para ordenar un vector podemos usar la función `std::sort`

```
vector <int> a = {5, 2, 6, 2, 1, 8};
sort(a.begin(), a.end());
// a queda como {1, 2, 2, 5, 6, 8}
```

Complejidad temporal $O(n \log n)$ si $|a| = n$

Para ordenar un vector podemos usar la función `std::sort`

```
vector <int> a = {5, 2, 6, 2, 1, 8};
sort(a.begin(), a.end());
// a queda como {1, 2, 2, 5, 6, 8}
```

Complejidad temporal $O(n \log n)$ si $|a| = n$

¿qué son `a.begin()` y `a.end()`?

Función de comparación

Podemos cambiar la **función de comparación** para ordenar de la forma que queramos. Por ejemplo, para ordenar de mayor a menor:

```
bool es_mayor(int a, int b){
  // La función de comparación debe retornar true
  // cuando "a" va antes que "b" en nuestro ordenamiento
  return a > b;
}
```

```
sort(a.begin(), a.end(), es_mayor);
```

Función de comparación

La librería estándar trae `greater<T>` que funciona como `es_mayor`

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

Función de comparación

La librería estándar trae `greater<T>` que funciona como `es_mayor`

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

También podemos usar funciones anónimas o lambdas:

```
sort(a.begin(), a.end(), [](int a, int b){
  return a > b;
});
```

Algoritmos greedy

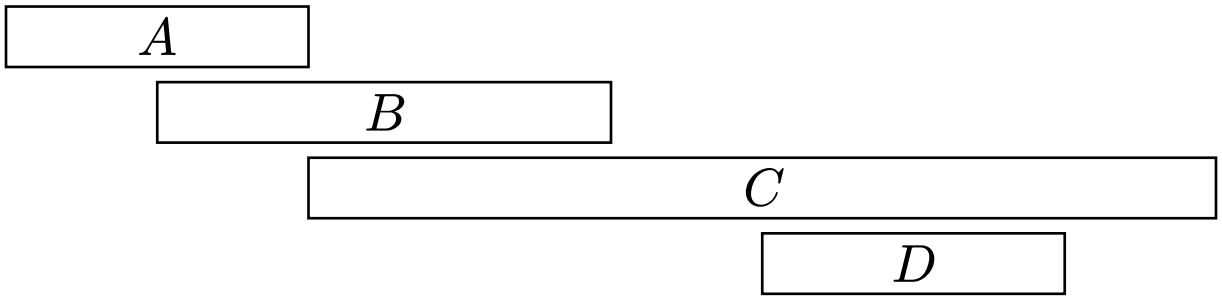
Programando eventos

Dadas n películas en un cine con sus tiempos de inicio y fin, encuentra un horario que permita ver la mayor cantidad de películas:

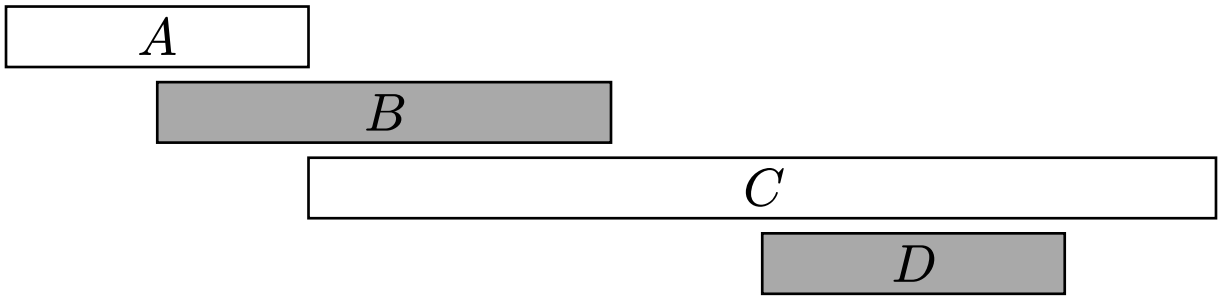
- Solo se puede ver una a la vez.
- No se puede ver una película parcialmente.

película	tiempo inicio	tiempo fin
A	1	3
B	2	5
C	3	9
D	6	8

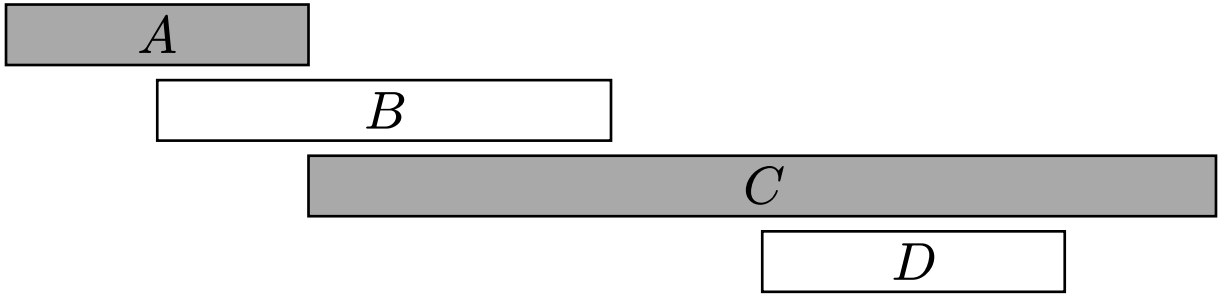
Solución al ejemplo



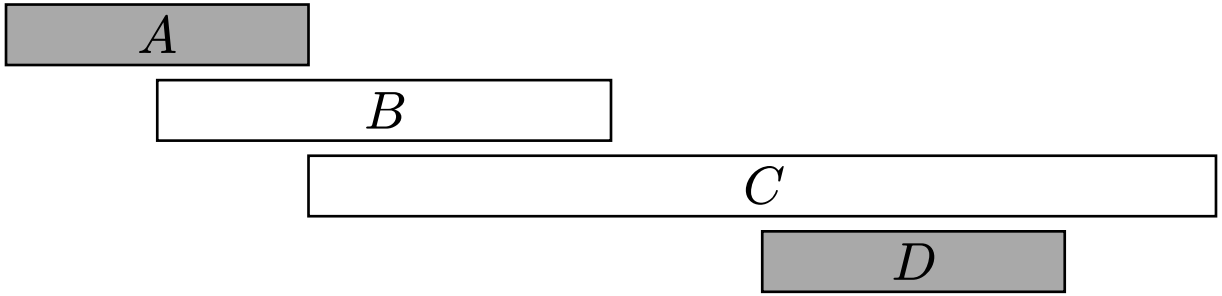
Solución al ejemplo



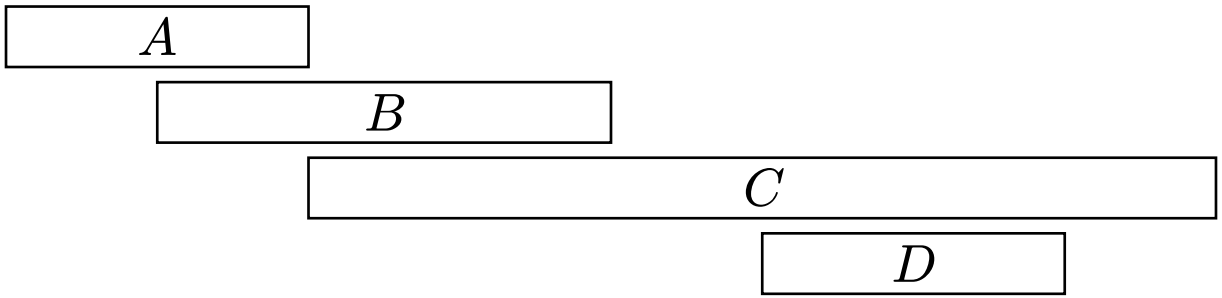
Solución al ejemplo



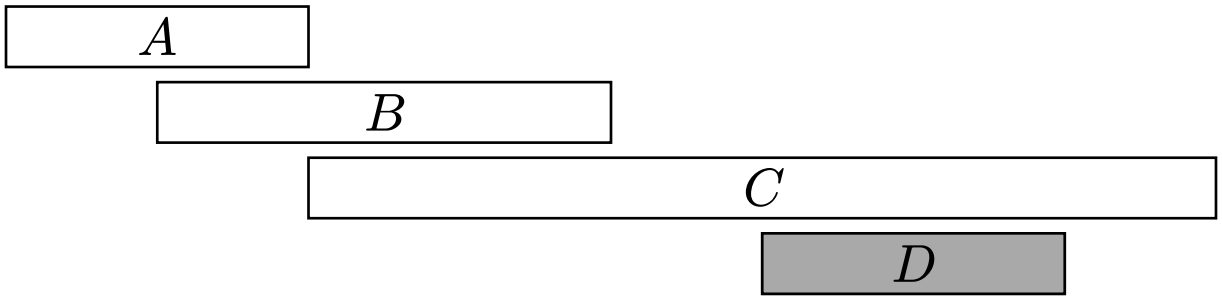
Solución al ejemplo



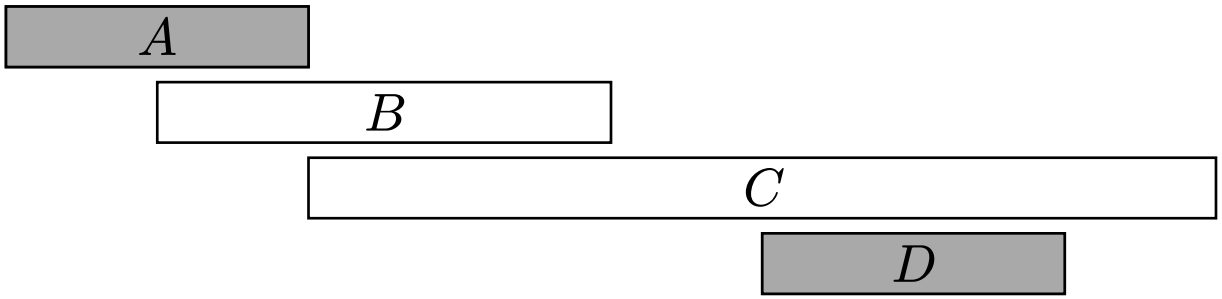
Idea greedy: los escogemos del más pequeño al más grande



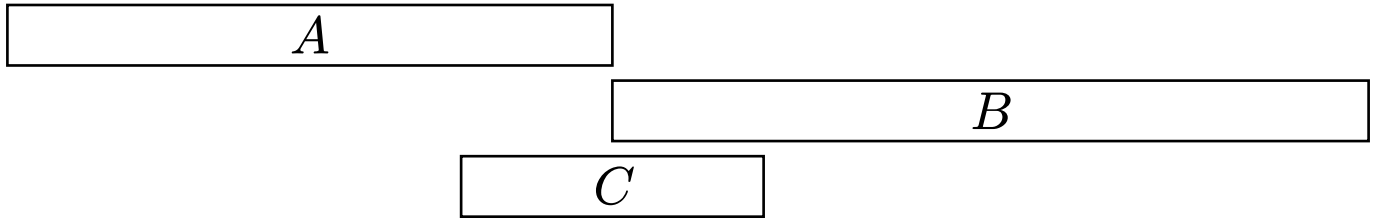
Idea greedy: los escogemos del más pequeño al más grande



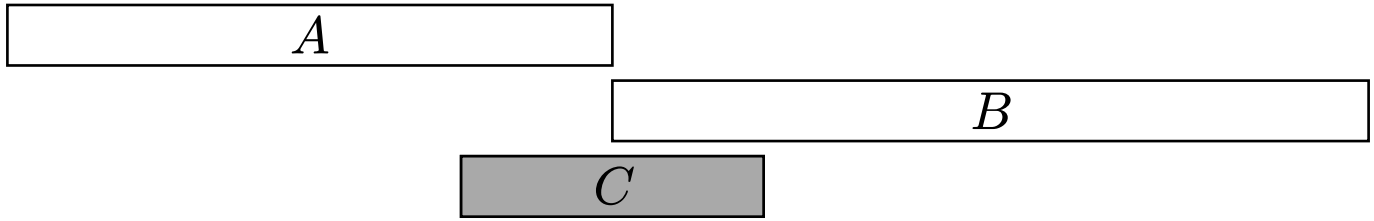
Idea greedy: los escogemos del más pequeño al más grande



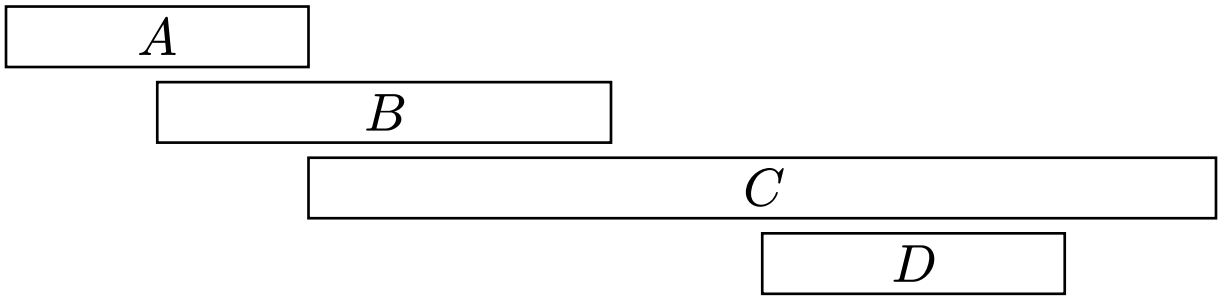
No funciona 😔



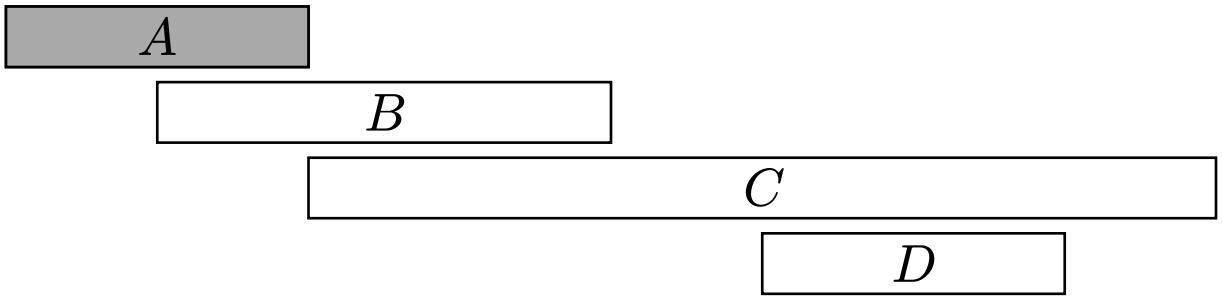
No funciona 😔



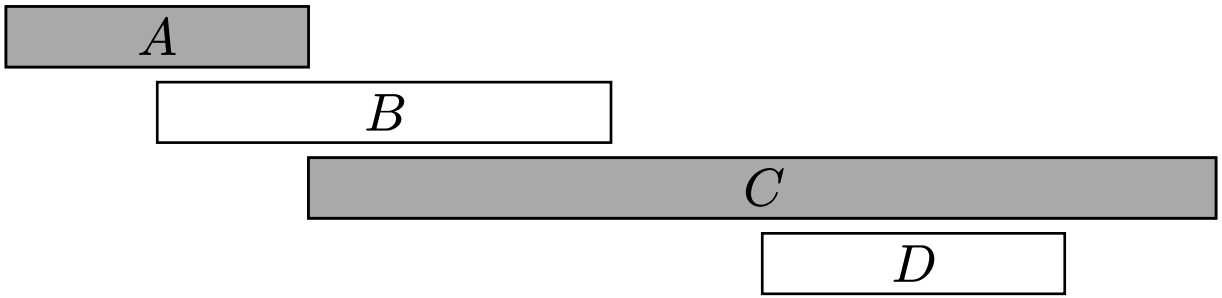
Idea greedy 2: escogemos en orden de cuál **comienza** primero



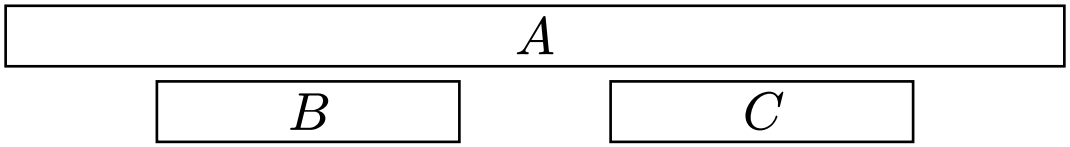
Idea greedy 2: escogemos en orden de cuál **comienza** primero



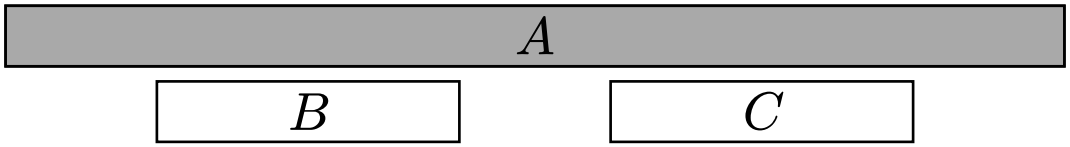
Idea greedy 2: escogemos en orden de cuál **comienza** primero



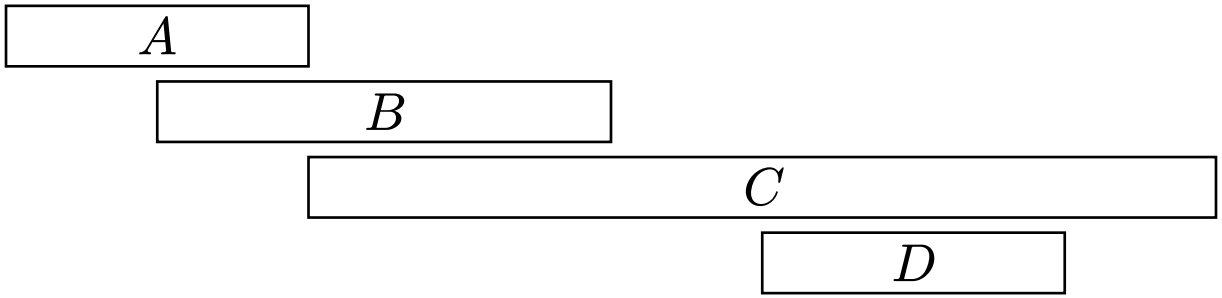
Tampoco funciona 😔 😔



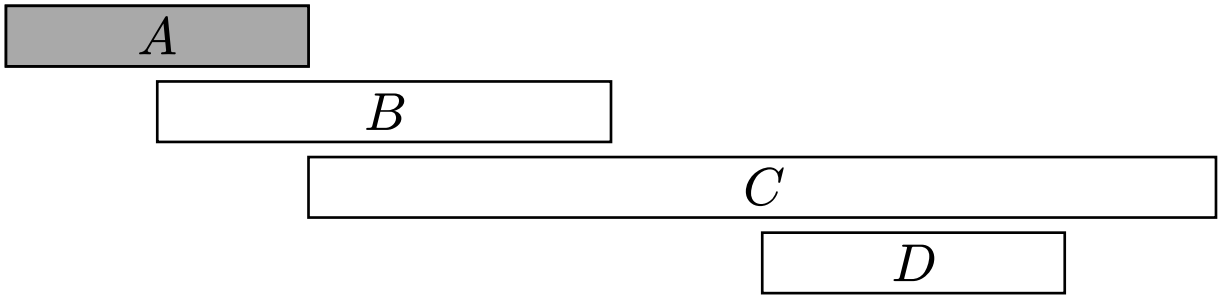
Tampoco funciona 😔 😔



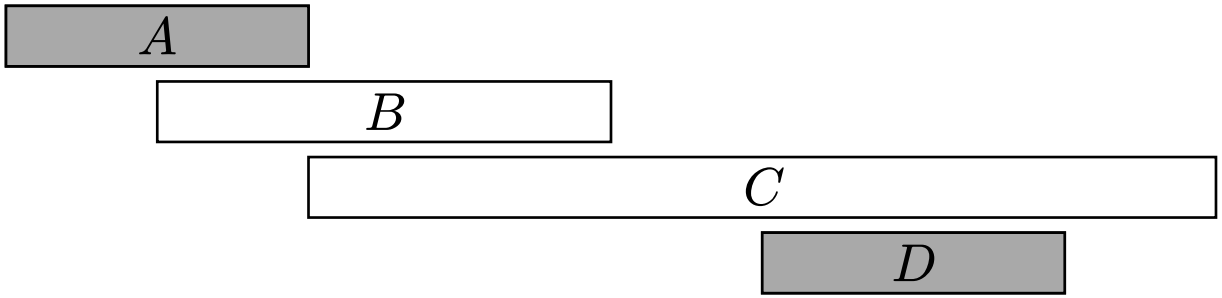
Idea greedy 3: escogemos en orden de cuál **termina** primero



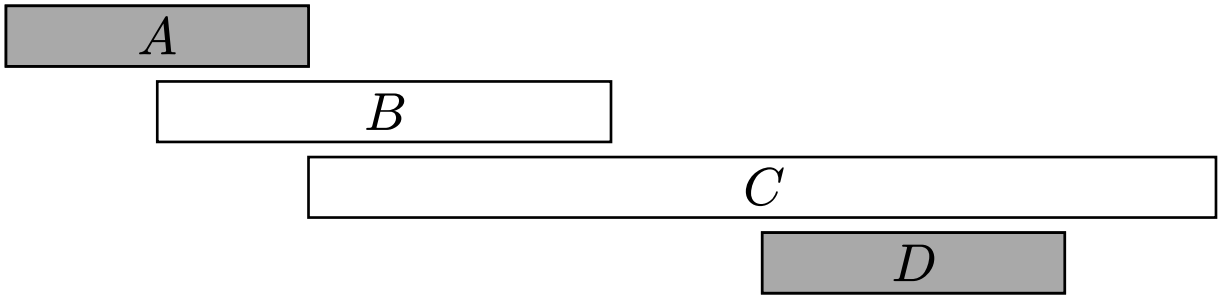
Idea greedy 3: escogemos en orden de cuál **termina** primero



Idea greedy 3: escogemos en orden de cuál **termina** primero



Idea greedy 3: escogemos en orden de cuál **termina** primero



Funciona **siempre**

Un algoritmo greedy es uno que…

Un algoritmo greedy es uno que…

- Toma siempre la “mejor” decisión **local**, con la esperanza de construir una solución óptima **global**.

Algoritmos greedy

Un algoritmo greedy es uno que…

- Toma siempre la “mejor” decisión **local**, con la esperanza de construir una solución óptima **global**.
- No es un algoritmo específico, si no una **estrategia** o **forma de pensar.**

Algoritmos greedy

Un algoritmo greedy es uno que…

- Toma siempre la “mejor” decisión **local**, con la esperanza de construir una solución óptima **global**.
- No es un algoritmo específico, si no una **estrategia** o **forma de pensar.**
- Son **peligrosos**, porque es fácil convencerse de que funciona cuando no.

Algoritmos greedy

Un algoritmo greedy es uno que…

- Toma siempre la “mejor” decisión **local**, con la esperanza de construir una solución óptima **global**.
- No es un algoritmo específico, si no una **estrategia** o **forma de pensar.**
- Son **peligrosos**, porque es fácil convencerse de que funciona cuando no.
- Experiencia, intuición, jugársela

Dragones y caballeros

Tenemos n dragones y m caballeros. Los dragones tienen fuerzas $a_1, a_2, ..., a_n$ y los caballeros tienen fuerzas $b_1, b_2, ..., b_m$. Un caballero puede pelear con un solo dragón y le gana si tiene igual o más fuerza.

¿Escogiendo las peleas, es posible derrotar a todos los dragones?

Dragones y caballeros

Tenemos n dragones y m caballeros. Los dragones tienen fuerzas $a_1, a_2, ..., a_n$ y los caballeros tienen fuerzas $b_1, b_2, ..., b_m$. Un caballero puede pelear con un solo dragón y le gana si tiene igual o más fuerza.

¿Escogiendo las peleas, es posible derrotar a todos los dragones?

Entrada

```
2 3
5 4
7 8 4
```

Salida

```
YES
```

Dragones y caballeros

Tenemos n dragones y m caballeros. Los dragones tienen fuerzas $a_1, a_2, ..., a_n$ y los caballeros tienen fuerzas $b_1, b_2, ..., b_m$. Un caballero puede pelear con un solo dragón y le gana si tiene igual o más fuerza.

¿Escogiendo las peleas, es posible derrotar a todos los dragones?

Entrada

```
2 1
5 5
10
```

Salida

```
NO
```

Bit flips

Tenemos dos secuencias a y b de n bits. Queremos convertir a en b usando la siguiente operación:

- Escoger un substring de a e invertir todos los bits en él.

¿Cuál es la menor cantidad de operaciones para convertir a en b?

Bit flips

Tenemos dos secuencias a y b de n bits. Queremos convertir a en b usando la siguiente operación:

- Escoger un substring de a e invertir todos los bits en él.

¿Cuál es la menor cantidad de operaciones para convertir a en b?

Entrada

```
7
1000100
0011100
```

Salida

```
2
```

Búsqueda binaria

Buscar un número en un arreglo ordenado

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15

El algoritmo

- Inicializamos nuestro espacio de búsqueda $[l, r]$.

El algoritmo

- Inicializamos nuestro espacio de búsqueda $[l, r]$.
- En cada iteración, consultamos el elemento al medio del espacio de búsqueda $\left\lfloor \frac{l+r}{2} \right\rfloor$ y descartamos una de las mitades, achicando nuestro espacio de búsqueda.

El algoritmo

- Inicializamos nuestro espacio de búsqueda $[l, r]$.
- En cada iteración, consultamos el elemento al medio del espacio de búsqueda $\left\lfloor \frac{l+r}{2} \right\rfloor$ y descartamos una de las mitades, achicando nuestro espacio de búsqueda.
- Cantidad de iteraciones hasta que nuestro espacio de búsqueda tenga tamaño 1:

El algoritmo

- Inicializamos nuestro espacio de búsqueda $[l, r]$.
- En cada iteración, consultamos el elemento al medio del espacio de búsqueda $\left\lfloor \frac{l+r}{2} \right\rfloor$ y descartamos una de las mitades, achicando nuestro espacio de búsqueda.
- Cantidad de iteraciones hasta que nuestro espacio de búsqueda tenga tamaño 1:

$$O(\log n)$$

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Generalización

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

FFTTTTTTTTTTT

Generalización

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

FFTTTTTTTTTTT es monótona

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

FFTTTTTTTTTTT es monótona

FFFFFFFFFFFFFFFF

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

FFTTTTTTTTTTT es monótona

FFFFFFFFFFFFFFFF es monótona

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

FFTTTTTTTTTTT es monótona

FFFFFFFFFFFFFFFF es monótona

FFFFTTTTTFFFFFFF

La búsqueda binaria no sirve solo para buscar un número en un arreglo.

Para generalizarla, necesitamos dos conceptos:

Función binaria: Es una función f que retorna true o false.

Función monótona: Una función binaria es *monótona* si toma un valor hasta cierto punto en donde cambia al otro valor y se mantiene en ese otro valor.

TTTTTTTTTTFFFFFFFF es monótona

FFTTTTTTTTTTT es monótona

FFFFFFFFFFFFFFFF es monótona

FFFFTTTTTFFFFFFF **no** es monótona

Aplicando la generalización

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15

Aplicando la generalización

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15

Función binaria:

Aplicando la generalización

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15

Función binaria: $f(x) = (x \geq 2)$

Aplicando la generalización

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15

Función binaria: $f(x) = (x \geq 2)$

¿Es monótona?

Aplicando la generalización

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15

Función binaria: $f(x) = (x \geq 2)$

¿Es monótona? Sí, porque el arreglo está ordenado

Aplicando la generalización

Buscamos el 2 en este arreglo **ordenado**

-5	0	1	2	3	3	5	5	7	9	10	15
F	F	F	V	V	V	V	V	V	V	V	V

Función binaria: $f(x) = (x \geq 2)$

¿Es monótona? Sí, porque el arreglo está ordenado

Implementación

Raíz cuadrada

Dado un entero n calcula

$$\lfloor \sqrt{n} \rfloor$$

Raíz cuadrada

Dado un entero n calcula

$$\lfloor \sqrt{n} \rfloor$$

Entrada	Salida
9	3

Raíz cuadrada

Dado un entero n calcula

$$\lfloor \sqrt{n} \rfloor$$

Entrada	Salida
15	3

Suma de cuadrados

Dado un entero c ($0 \leq c \leq 2^{31} - 1$) di si existen dos enteros a y b tal que $a^2 + b^2 = c$.

Suma de cuadrados

Dado un entero c ($0 \leq c \leq 2^{31} - 1$) di si existen dos enteros a y b tal que $a^2 + b^2 = c$.

Entrada	Salida
5	YES

Suma de cuadrados

Dado un entero c ($0 \leq c \leq 2^{31} - 1$) di si existen dos enteros a y b tal que $a^2 + b^2 = c$.

Entrada	Salida
3	NO

Fábrica de figuritas

Tienes n impresoras 3D, la i-ésima de ellas tarda a_i segundos en imprimir una figurita. ($1 \leq n \leq 2 \cdot 10^5$, $1 \leq a_i \leq 10^9$)

¿Cuánto es el tiempo mínimo requerido para imprimir k figuritas? ($1 \leq k \leq 10^9$)

Fábrica de figuritas

Tienes n impresoras 3D, la i-ésima de ellas tarda a_i segundos en imprimir una figurita. ($1 \leq n \leq 2 \cdot 10^5$, $1 \leq a_i \leq 10^9$)

¿Cuánto es el tiempo mínimo requerido para imprimir k figuritas? ($1 \leq k \leq 10^9$)

Entrada

```
n k
a1 a2 a3
```

Salida

```
tiempo_minimo
```

Fábrica de figuritas

Tienes n impresoras 3D, la i-ésima de ellas tarda a_i segundos en imprimir una figurita. ($1 \leq n \leq 2 \cdot 10^5$, $1 \leq a_i \leq 10^9$)

¿Cuánto es el tiempo mínimo requerido para imprimir k figuritas? ($1 \leq k \leq 10^9$)

Entrada

```
3 7
3 2 5
```

Salida

```
8
```