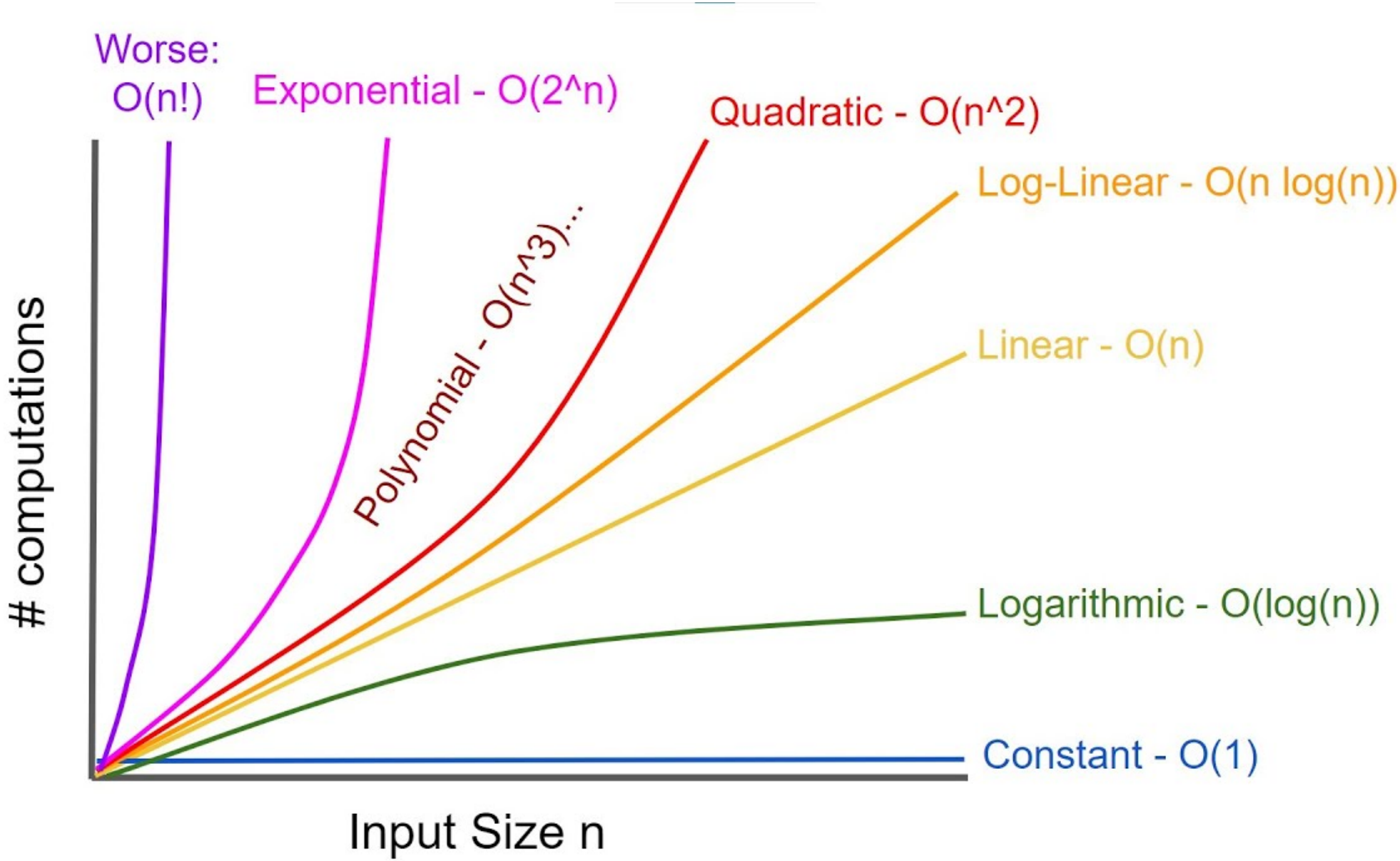
Complejidad computacional

PCFIM


Worse:
O(n!)
Exponential - O(2^n)
Quadratic - O(n^2)
Polynomial - O(n^3)...
Log-Linear - O(n log(n))
Linear - O(n)
Logarithmic - O(log(n))
Constant - O(1)
computations
Input Size n

Una forma de aproximar el tiempo que toma nuestro código en ejecutarse es contando la cantidad de operaciones *básicas* que realiza.

Se consideran operaciones básicas:

- Operaciones aritméticas (+, -, *, /, %, <, >, =)
- Operaciones lógicas (or, and, not, xor)
- Acceder al valor de un array
- Definir una nueva variable

¿Cuáles no son operaciones básicas?:

- Comparar strings, vectores
- Definir un vector
- Concatenar 2 strings

Consideramos que una computadora puede realizar hasta 10^8 operaciones básicas por segundo. Entonces, calcular el tiempo de ejecución sería:

$$\text{tiempo} = \frac{\#\ \text{operaciones}}{10^8}$$

Una forma más efectiva de medir esto, es con la complejidad asintótica del código.

$n = 1\,000\,000\,000$

$T(\text{operation}) = 1 \text{ second}$

$O(1)$ → 1 second

$O(\log n)$ → 20 seconds

$O(n)$ → 11 days

$O(n \log n)$ → 1 year

$O(n^2)$ → 32 000 years

$O(2^n)$ → Eternity

$O(n!)$ → ...

Complejidad asintótica

- El objetivo es resolver los problemas rápido
- Probar en el peor escenario posible
- Medir tiempos rápidamente
- Predecir el tiempo antes de implementarlo
- Hacer soluciones rápidas

```
for(int i=0; i<n; i++){ // O(n)
    for(int j=0; j<m; j++){ // O(m)
        ...
        doSomething();
        ...
    } // O(m)*O(doSomething)
}// O(n)*O(m)*O(doSomething)
```

```
void f(int x){
    int ans = 0; // O(1)
    for(int i=0; i<x; i++){
        ans += i; // O(1)
    } // O(x)
}// O(x) + O(1) = O(x)

for(int i=0; i<n; i++){
    f(10); // O(10)
}// O(10*n) = O(n)

for(int i=0; i<n; i++){
    f(i); // O(i)
} // Sum_{i=0}^{n} O(i) =
// O(1) + O(2) + ... + O(n-1) + O(n) = O(n^2)
```

Así, si $n = 10^5$:

```
void f(int x){
    int ans = 0; // O(1)
    for(int i=0; i<x; i++){
        ans += i; // O(1)
    } // O(x)
}// O(x) -> t = 10^5/10^8 = 0.001 s

for(int i=0; i<n; i++){
    f(10); // O(10)
}// O(10*n) = O(n) -> t = 10^5/10^8 = 0.001 s

for(int i=0; i<n; i++){
    f(i); // O(i)
} // Sum_{i=0}^{n} O(i) =
// O(1) + O(2) + ... + O(n-1) + O(n) = O(n^2)
// t = (10^5)^2/10^8 = 10^2 s
```

Un algoritmo tiene complejidad asintótica $O(f(n))$ si este no realiza más que $C \cdot f(n)$ operaciones en cualquier input, donde C es algún número constante.

Operaciones

Ligero	***Pesado***
+, -	%
*	Insertar elementos
Logicas	Input/Output

Planeando una solución

- Tu solución es lenta
- Primero, intenta mejorarla asintóticamente
- Reduce la complejidad en operaciones costosas
- Si aún tienes TLE, mejora tus constantes. Solo si estás cerca de las 10^8 operaciones
- Evita operaciones pesadas

Ejemplos

Substring problem

Dados dos strings s y t, verificar si s es un substring de t.

Input	*Output*
abac abacabad	Yes
cac abacabad	No
abab abacabab	Yes

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    string s, t; // O(1)
    cin >> s >> t; // O(|t| + |s|)
    int sz = s.size(); // O(1)
    for(int i=0; i<t.size()-sz+1; i++){ // O(|t|)
        bool test = true; // O(1)
        for(int j=0; j<sz; j++){ // O(|s|)
            if(s[j] != t[i+j]) test = false; // O(1)
        } // O(|s|)
        if(test){
            cout << "Yes";
            return 0;
        } // O(1)
    } // O(|t|*|s|)
    cout << "No";
    return 0;
}
```

Mathemagics

```
void f(int n){
    vector <bool> a(n, true);
    for(int i=2; i<n; i++){
        for(int j=i; j<n; j+=i) a[j] = false;
    }
}
```

```
void f(int n){
    vector <bool> a(n, true);
    for(int i=2; i<n; i++){
        if(a[i]){
            // cout << i << '\n';
            for(int j=2*i; j<n; j+=i) a[j] = false;
        }
    }
}
```

Little Pony and Sort by Shift

Dado un array de n elementos. Calcular el mínimo número de operaciones que se necesita para ordenarlo, siendo la operación permitida mover el último elemento al inicio de la secuencia.

Input

The first line contains an integer n $(2 \leq n \leq 10^5)$. The second line contains n integer numbers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq 10^5)$.

Output

If it's impossible to sort the sequence output -1. Otherwise output the minimum number of operations.

```cpp
#include <bits/stdc++.h>
using namespace std;

int n;
int getPos(int arr[]){
    int p = -1;
    for(int i=0; i<n-1; i++){
        if(arr[i] > arr[i+1]){
                if(p == -1) p = i;
                else return -1;
        }
    }
    if(p == -1) p = n-1;
    return p;
}

int solve(int arr[]){
    int p = getPos(arr);
    if(p == -1) return -1;
    if(p == n-1) return 0;
    if(arr[0] >= arr[n-1]) return n-p-1;
    else return -1;
}

int main(){
    cin >> n;
    int arr[n];
    for(int i=0; i<n; i++) cin >> arr[i];
    int ans = solve(arr);
    cout << ans;
    return 0;
}
```

The Monster

A monster is chasing after Rick and Morty on another planet. They're so frightened that sometimes they scream. More accurately, Rick screams at times b, $b + a$, $b + 2a$, $b + 3a$, $\ldots$ and Morty screams at times d, $d + c$, $d + 2c$, $d + 3c$, $\ldots$

The Monster will catch them if at any point they scream at the same time, so it wants to know when it will catch them (the first time they scream at the same time) or that they will never scream at the same time.

Input

The first line of input contains two integers a and b $(1 \leq a, b \leq 100)$.

The second line contains two integers c and d $(1 \leq c, d \leq 100)$.

Output

Print the first time Rick and Morty will scream at the same time, or -1 if they will never scream at the same time.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
        int a, b, c, d; cin >> a >> b >> c >> d;
        for(int i=0; i<=100; i++){
                int aux = d+c*i;
                int k = (aux-b)/a;
                if((aux-b)%a == 0 && k >= 0){
                        cout << aux << '\n';
                        return 0;
                }
        }
        cout << -1;
    return 0;
}
```