Análisis de Algoritmos

tiempo de ejecución

B.S. Rodolfo Mercado Gonzales

Análisis de Algoritmos

- Las computadoras pueden ser muy rápidas, pero no infinitamente rápidas.
- La memoria puede ser barata, pero no es gratuita.
- El **tiempo de ejecución** y el **espacio usado de memoria** son recursos limitados.

Análisis de Algoritmos

“El análisis de un algoritmo busca predecir los recursos que el mismo requiere para poder ejecutarse”

Análisis de Algoritmos

- Para la solución de un problema buscamos el **algoritmo más eficiente**, por ende que minimice los recursos necesarios.

- Los recursos analizados son el tiempo de ejecución y el espacio de memoria usado.

Modelo RAM

- Para analizar definiremos una computadora modelo denominada RAM (Random Access Machine).
- Representa el comportamiento esencial de las computadoras.
- La instrucciones son ejecutadas una después de otra, sin concurrencia.
- Las operaciones simples o primitivas (+, -, * , / , =, if, return, …) y accesos a memoria son realizados en un paso o tiempo constante.

Modelo RAM

- Los bucles y las subrutinas están compuestas de varias operaciones primitivas
- Tener en cuenta que en cada iteración de un bucle se incrementa el valor de una variable.
- Las operaciones de asignación e igualdad depende de los tipos de datos en que se aplique (para datos primitivos se considera una operación).

Tiempo de Ejecución

El tiempo de ejecución de un algoritmo sobre una entrada en particular es el número de operaciones primitivas o pasos ejecutados

Tiempo de Ejecución

```
int i, sum;
sum = 0, i = 0;
while( i < n ){
    sum = sum + i;
    i = i + 1;
}
```

operaciones realizadas

2 asignaciones	*i, sum*
$n + 1$ comparaciones	*i < n*
n sumas	*sum + i*
n asignaciones	*sum =*
n incrementos	*i + 1*
n asignaciones	*i =*

$$T(n) = 5\,n + 3$$

Dificultades

- Calcular el número exacto de operaciones primitivas requiere que el algoritmo sea especificado detalladamente.
- Dos algoritmos pueden diferir en tiempo de ejecución solo por detalles de implementación.

Notación Big O

- La notación **Big O** ignora detalles que no impactan en la comparación de algoritmos.
- Para hallar la complejidad $\mathbf{O}(\boldsymbol{n})$ de un algoritmo sólo nos interesa el término de mayor orden de la función $\boldsymbol{T}(\boldsymbol{n})$.

Notación Big O

$$T(n) = 2n^2 + 100n + 6 = \boldsymbol{O(n^2)}$$

$$T(n) = 5 = \boldsymbol{O(1)}$$

$$T(n) = 3 * 2^n + 5 = \boldsymbol{O(2^n)}$$

$$T(n) = n + 6 = \boldsymbol{O(n)}$$

$$T(n) = 2\log n + 1 = \boldsymbol{O(log\ n)}$$

ahora ya podemos comparar algoritmos

Tiempo de Ejecución

En **C++**, aproximadamente $\mathbf{2 * 10^8}$ operaciones se ejecutan en 1 segundo.

```
clock_t ini = clock();
/*
    codigo
*/
clock_t fin = clock();
double tiempo = (double)(fin - ini) / CLOCKS_PER_SEC;
printf("tiempo : %.2f segundos", tiempo);
```

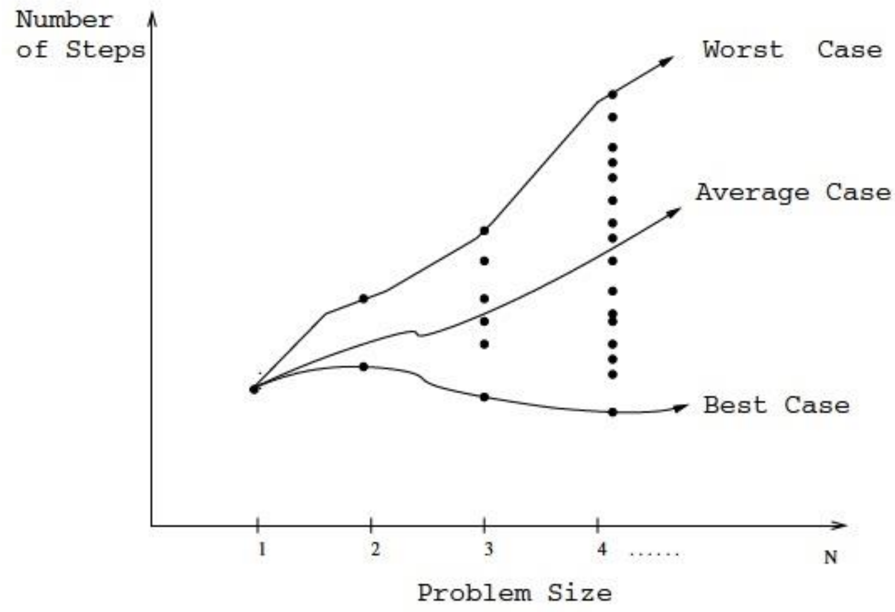
Análisis del tiempo de ejecución

Si mi algoritmo es exponencial, pero es rápido para entradas pequeñas entonces es eficiente?

Análisis del tiempo de ejecución

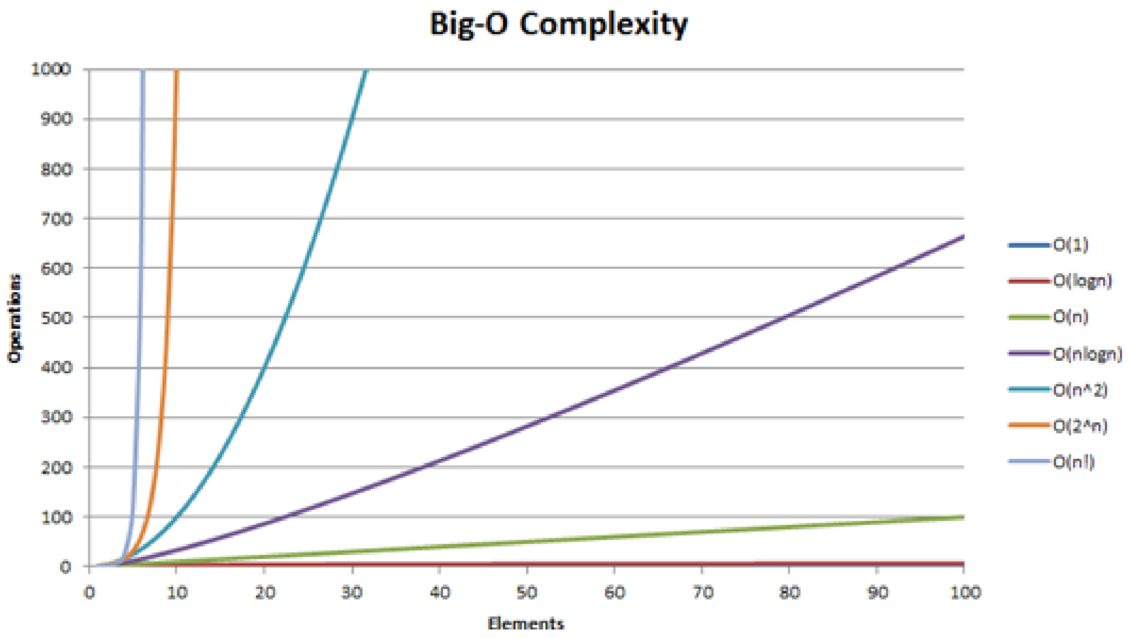
- Mejor caso : menor número de pasos para una entrada de tamaño n.
- Caso promedio : número de pasos promedio para una entrada de tamaño n
- **Peor caso** : máximo número de pasos para una entrada de tamaño n.

Análisis del tiempo de ejecución



debemos analizar el peor de lo casos.

Complejidad en Tiempo



Complejidad en Tiempo

Ahora podemos tener una idea del orden de complejidad que requeriría la solución a un problema dado una entrada de tamaño n.

Entrada	Posible solución
$n \leq 10$	$O(\, n!\,)$
$n \leq 22$	$O(\, 2^n\, n\,)$
$n \leq 50$	$O(\, n^4\,)$
$n \leq 1000$	$O(\, n^2\,)\, ,\, O(\, n^2 \log n)$
$n \leq 10^5$	$O(\, n \log n)$
$n \leq 10^9$	$O(\log n), O(\, 1)$

límites comunes en los concursos de programación.

Complejidad en Tiempo

Dado un número primo n ($n \leq 10^9$), se desea saber cuántos números enteros positivos menores o iguales a dicho número, son coprimos con n.

Solución Ingenua

Recorremos cada uno de los números del 1 a n y revisamos si es coprimo con n.

complejidad en tiempo: $\boldsymbol{O(n\ log n)}$

Complejidad en Tiempo

Solución Eficiente

Recordemos la función phi de Euler y que para un número primo n:

$$\varphi(n) = n - 1$$

complejidad en tiempo: $\boldsymbol{O(1)}$

Complejidad en Tiempo

Determinar si un número $\boldsymbol{n}$ $(\boldsymbol{n \leq 10^9})$ es primo.

Solución Ingenua

Tomando como un caso especial que el 1 no es primo, recorremos cada uno de los números del 2 a $n-1$ (posibles divisores), si encontramos que alguno es divisor de n entonces el número no es primo, caso contrario será primo.

complejidad en tiempo: $\boldsymbol{O(n)}$

Complejidad en Tiempo

Solución Eficiente

Teorema

Si n es un número compuesto, entonces n tiene al menos un divisor que es mayor que 1 y menor o igual a $\sqrt{n}$.

Complejidad en Tiempo

Solución Eficiente

Demostración

Sea $n = ab$; donde a, b son enteros y $1 < a \leq b < n$, entonces:

$\boldsymbol{a \leq \sqrt{n}}$, ya que si no caeríamos en una contradicción, porque tendríamos que $a, b > \sqrt{n}$ y por ende $ab > n$.

Complejidad en Tiempo

Solución Eficiente

Por ende, para saber si un número $n > 1$ es primo, sólo es necesario verificar que no tenga divisores en el rango $[2, \sqrt{n}]$.

Complejidad en tiempo: $\boldsymbol{O(\sqrt{n})}$

Complejidad en Tiempo

Ordenar un arreglo de n números.

Solución Ingenua

Utilicemos el algoritmo de "ordenamiento de burbuja".

Complejidad en tiempo: $\boldsymbol{O(n^2)}$

Complejidad en Tiempo

Solución Eficiente

Probemos la función **sort** que se encuentra en el encabezado **<algorithm>**.

```
//sea A un arreglo de enteros
sort( A, A + n );
```

complejidad en tiempo: $\boldsymbol{O(n\,log n)}$

Problemas

Codechef – Magic Pairs

Codechef – Chef Jumping

Codeforces – Game With Sticks

HackerRank – Strange Counter

Codeforces – Arpa's hard exam and Mehrdad's naive cheat

Codeforces – Mahmoud and a Triangle

Referencias

- Cormen, Thomas. Introduction to Algorithms.
- Skiena, Steven. The Algorithm Design Manual.
- Jiménez, Daniel. CS 1723 Data Structures. *https://www.cs.utexas.edu/users/djimenez/utsa/cs1723/lecture2.html*

¡ Good luck and have fun !