ALGORÍTMICA Y PROGRAMACIÓN I

EFICIENCIA

"eficiencia es la capacidad de disponer de alguien o algo para conseguir el cumplimiento adecuado de una función"

¿ CUÁL ES MÁS EFICIENTE ?

(MÍNIMO)

- Método 1:

```
m := a;
if b < m then m := b;
if c < m then m := c;
```

- Método 2:

```
if a <= b
  then if a <= c then m := a
                 else m := c
  else if b <= c then m := b
                else m := c
```

- Método 3:

```
if (a <= b) and (a <=c) then m := a;
if (b <= a) and (b <=c) then m := b;
if (c <= a) and (c <=b) then m := c;
```

- Método 4:

```
if (a <= b) and (a <= c) then m := a
  else if b <= c then m := b
              else m := c;
```

EFICIENCIA

Un algoritmo es eficiente si hace uso adecuado de los recursos donde se ejecuta.

Recursos

- Tiempo de CPU.
- Memoria utilizada.

¿Por qué preocuparse por la eficiencia si podemos conseguir un equipo más potente?

Entonces, ¿qué y cómo medimos?

COMPLEJIDAD COMPUTACIONAL

Indica el **esfuerzo** que hay que realizar para aplicar un algoritmo y lo **costoso que éste resulta**.

Puede medirse en términos:

- Espaciales: cantidad de memoria que un algoritmo consume o utiliza durante su ejecución → Complejidad espacial
- **Temporales**: tiempo que necesita el algoritmo para ejecutarse → Complejidad temporal
- Otros: utilización de CPU, utilización de periféricos, tiempo y costo de implementación.

ANÁLISIS SEGÚN SU EJECUCIÓN

Nos enfocamos en la complejidad temporal por ser el recurso más critico.

El tiempo de ejecución lo podemos medir

- Empíricamente – a posteriori
- Teóricamente – a priori

ANÁLISIS EMPÍRICO

Se basa en la selección de diferentes juegos de datos para luego utilizarlos con el algoritmo y medir los tiempos de respuesta.

Ventaja:

- Permite realizar un evaluación experimental de los recursos consumidos.

Desventajas:

- Se debe implementar el algoritmo.
- El juego de datos puede no ser adecuado para recorrer todas las posibilidades
- La máquina que se utiliza tiene gran impacto en el tiempo de ejecución

ANÁLISIS TEÓRICO (ASINTÓTICO)

Busca obtener una medida independizándose de los datos y de la máquina en que se ejecutará comparando la cantidad de operaciones (asignaciones, comparaciones, llamas) que se realizaran.

Ventajas

- Es independiente de los datos de entrada
- Es independiente de las condiciones de ejecución.
- Requiere solo una especificación de alto nivel (pseudocodigo)

Desventajas:

- Requiere de otros conocimientos (matemáticos).

TIEMPO DE EJECUCIÓN DE UN ALGORITMO

El tiempo de ejecución de un algoritmo T(n) se dice que es de orden f(n) cuando existe una función matemática f(n) que acota a T(n).

$T(n) = O(f(n))$ si $\exists$ constantes k y m tales que:

$T(n) \leq k.f(n)$ para $n \geq m$

Ejemplo:

$$\left.\begin{array}{l} \blacksquare\ T_1 = O(n^2) \\ \blacksquare\ T_2 = O(2n) \end{array}\right\}\ T_2 \text{ es más eficiente para } n \geq 3$$

BÚSQUEDA SECUENCIAL

Sólo nos interesan las **comparaciones** dado que las asignaciones no toman un valor significativo (en términos de cantidad).

Como máximo se realizan n comparaciones

$$T(n) = O(n)$$

Si el arreglo está ordenado podríamos pensar en una cota de n/2, pero la cota máxima sigue siendo n, por lo tanto, el orden seguirá siendo de O(n)

BÚSQUEDA BINARIA

A medida que avanzamos en la búsqueda del elemento, la cantidad de elementos del contenedor se va reduciendo a la mitad en cada iteración, por lo tanto podemos reducir esto en la siguiente expresión.

$$\frac{n}{2};\frac{n}{4};\frac{n}{8};\ldots=\frac{n}{2^1};\frac{n}{2^2};\frac{n}{2^3}\;;\ldots;\frac{n}{2^k}$$

El proceso de divisiones termina cuando la cantidad de elemento a buscar es 1;

El valor de K me da la cantidad máxima de comparaciones

BÚSQUEDA BINARIA

Como $\frac{n}{2^k}$ <= 1 entonces n <= 2^k

Si aplicamos logaritmos en ambos miembros:

- $Log_2(n) <= k.log_2(2)$ ➔ $Log_2(n) <= k$

Por cada división se realiza una comparación, por lo tanto el numero máximo de comparaciones es: **$Log_2(n)$**

Entonces el orden de la Búsqueda Binaria = $O(Log_2(n))$ + el orden de lo que implique tener el contenedor ordenado.

BÚSQUEDA BINARIA

N	$Log_2(n)$
1	0
2	1
4	2
8	3
16	4
128	7
512	9
1.024	10
100.000	~16

ORDENAMIENTO: MÉTODO DE SELECCIÓN

El número de comparaciones en cada pasada , en un contenedor de N elementos, es:

PASADAS	COMPARACIONES
1	n - 1
2	n - 2
3	n - 3
..	..
N	1

ORDENAMIENTO: MÉTODO DE SELECCIÓN (I)

Si la cantidad de comparaciones es:

(n-1) + (n-2) + … 2 + 1 = (por progresión aritmética) = $\sum_{i=1}^{n-1} i = \frac{1}{2}(n^2 - n)$

El orden es $O(n^2)$, aunque haya que sumarle los 3 intercambios de cada pasada 3(n-1) igual mantiene el orden.

ORDENAMIENTO: MÉTODO DE SELECCIÓN (2)

Cant comparaciones = $n-1 + n-2 + \ldots + 2 + 1$

Cant comparaciones = $1 + 2 + \ldots + n - 2 + n-1$

Sumo ambas expresiones:

2Cant comparaciones = $n + n + n + \ldots + n$

2Cant comparaciones = $n(1 + 1 + 1 + \ldots + 1) = n(n-1)$

Cant de comparaciones = $(n^2 - n) / 2$

El orden es $O(n^2)$, aunque haya que sumarle los 3 intercambios de cada pasada 3(n-1) igual mantiene el orden.

ORDENAMIENTO: MÉTODO DE BURBUJA

Con un razonamiento similar obtenemos que la cantidad de comparaciones es $(n^2 - n)/2$ → el orden es $O(n^2)$, a esto hay que sumarle los tres intercambios de cada pasada (3(n-1)), lo que no modifica el la cota de n^2

Si usamos la burbuja Mejorada entonces podemos tener una cota inferior y una superior.

(n-1) <= comp <= (n(n-1))/2.

(n-1) si el vector está ordenado

ORDENAMIENTO: MÉTODO DE INSERCIÓN

Si aplicamos el mismo razonamiento que el de la burbuja mejorada obtenemos:

```
(n-1) <= comp <= (n(n-1))/2.
(n-1) si el vector está ordenado
```

ALGUNOS VALORES PARA PENSAR…

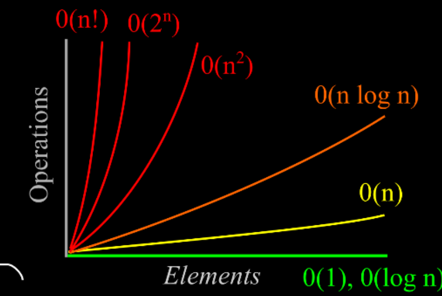
N	$Log_2(n)$	n^2
1	0	1
2	1	4
4	2	16
8	3	64
16	4	256
…	…	…
50.000	~15	2.500.000.000
100.000	~16	10.000.000.000

Legend

TIME Complexity VS. SPACE Complexity

Good Fair Bad

Good Fair Bad



<BIG-O-CHEATSHEET>

</>

DATA STRUCTURE Operations

DATA Structure	TIME Complexity Average: Access	Average: Search	Average: Insertion	Average: Deletion	Worst: Access	Worst: Search	Worst: Insertion	Worst: Deletion	SPACE Complexity Worst
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

ARRAY SORTING Algorithms

ARRAY Algorithms	TIME Complexity Best	Average	Worst	SPACE Complexity Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(1)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Tree Sort	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Shell Sort	O(n log(n))	O(n log(n)^2)	O(n log(n)^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Counting Sort	O(n+k)	O(n+k)	O(n+k)	O(k)
Cubesort	O(n)	O(n log(n))	O(n log(n))	O(n)

BIBLIOGRAFÍA

- Algoritmos, Datos y Programas. De Giusti - Madoz y otros.
- Tecnicaas de diseño de Algoritmos. Guerequeta - Vallecillo
- Estrcturas de datos. Cairo – Guardati
- https://www.bigocheatsheet.com/