EFICIENCIA/COMPLEJIDAD ALGORÍTMICA

MEDIDA DE LOS RECURSOS QUE EMPLEA UN ALGORITMO EN SU ALMACENAMIENTO Y EJECUCIÓN.

>	UTILIZACIÓN DE RECURSOS
<	EFICIENCIA
>	COMPLEJIDAD

EL ANÁLISIS DE LA COMPLEJIDAD DE UN ALGORITMO PUEDE LLEVARSE A CABO DE DOS FORMAS:

- TEÓRICA
- EMPÍRICA

ANÁLISIS TEÓRICO

VENTAJAS DEL ENFOQUE TEÓRICO

- INDEPENDENCIA DEL LENGUAJE DE PROGRAMACIÓN UTILIZADO.
- INDEPENDENCIA DEL ORDENADOR EN EL QUE SE EJECUTA.
- INDEPENDENCIA DE LA PERICIA DEL PROGRAMADOR.

COMPLEJIDAD TEMPORAL

TIEMPO NECESARIO PARA EJECUTAR UN ALGORITMO.

COMPLEJIDAD ESPACIAL

CANTIDAD DE MEMORIA NECESARIA PARA EJECUTAR UN ALGORITMO.

HABITUALMENTE SE HA DE LLEGAR A UN COMPROMISO ENTRE LA COMPLEJIDAD TEMPORAL Y LA COMPLEJIDAD ESPACIAL

LA MEDIDA DE LA COMPLEJIDAD HA DE SER INDEPENDIENTE DE

- EL ORDENADOR CON EL QUE SE TRABAJE.
- EL LENGUAJE DE PROGRAMACIÓN.
- EL COMPILADOR.
- CUALQUIER OTRO ELEMENTO, HARDWARE O SOFTWARE, QUE PUEDA INFLUIR EN EL ANÁLISIS.

COMPLEJIDAD TEMPORAL

ENTRADA (n)

Número de componentes sobre las que se ejecuta un algoritmo.

T (n)

Complejidad en tiempo para una entrada de tamaño n.

PRINCIPIO DE INVARIANZA

Dado un algoritmo y dos implementaciones I1 e I2 que tardan T1(n) y T2(n) respectivamente, existe una constante real C>0 y un número natural n0 tales que para todo n >= n0 se verifica que

$$T1(n) \le C \, T2(n)$$

Esto es, dos implementaciones distintas del mismo algoritmo sólo difieren en cuanto a eficiencia en un factor constante para valores de la entrada suficientemente grandes.

COMPLEJIDAD TEMPORAL

ESTUDIO TEÓRICO

es necesario seleccionar los datos (n) que más influyen en el coste de un algoritmo. .

TAMAÑO DE LOS DATOS DE ENTRADA

-Cuando el tamaño de los datos es pequeño **NO** habrá diferencias significativas en el uso de distintos algoritmos.

- Cuando el tamaño de los datos es grande, los costes de los diferentes algoritmos **SI** pueden variar de manera significativa.

Los distintos algoritmos que resuelven un mismo problema pueden tener grandes diferencias en su tiempo de ejecución, a veces, de órdenes de magnitud (interesa calcular, de forma aproximada, el orden de magnitud que tiene el tiempo de ejecución de cada algoritmo).

COMPLEJIDAD TEMPORAL

ORDEN DE MAGNITUD

El ORDEN (logarítmico, lineal, cuadrático, exponencial, etc, ..) de la función T(n) (que mide la complejidad temporal de un algoritmo) es el que expresa el COMPORTAMIENTO DOMINANTE CUANDO EL TAMAÑO DE LOS DATOS ES GRANDE.

COMPORTAMIENTO ASINTÓTICO

COMPORTAMIENTO PARA VALORES DE LA ENTRADA SUFICIENTEMENTE GRANDES

COMPLEJIDAD TEMPORAL

OBJETIVO DEL ESTUDIO DE LA COMPLEJIDAD ALGORÍTMICA

DETERMINAR EL COMPORTAMIENTO ASINTÓTICO DEL COSTE.

DIMENSIONES

Θ	ORDEN EXACTO DE LA FUNCIÓN.
O	COTA SUPERIOR.
Ω	COTA INFERIOR.

COMPLEJIDAD TEMPORAL

COTA SUPERIOR (O)

POSIBILITA CALCULAR EL LÍMITE O COTA SUPERIOR DEL TIEMPO DE EJECUCIÓN DE UN ALGORITMO.

PROPIEDADES DE LA COTA SUPERIOR

O(f(n)) = k O(f(n)), k es una constante

O(f(n)+g(n)) = max (O(f(n)), O(G(n)))

O(f(n)) + O(g(n)) = O(f(n)+g(n))

O(f(n)) * O(g(n)) = O(f(n) * g(n))

O(O(f(n))) = O(f(n))

COMPLEJIDAD TEMPORAL

FUNCIONES DE COMPLEJIDAD TEMPORAL

- O(1) → Complejidad constante
- O(log n) → Complejidad logarítmica
- O(n) → Complejidad lineal
- O(n*log n)
- O(n**2) → Complejidad cuadrática
- O(n**3) → Complejidad cúbica
- O(n**k) → Complejidad polinómica (k>3)
- O(2**n) → Complejidad exponencial
- O(n!) → Complejidad factorial

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEJIDAD

Para calcular la complejidad de un algoritmo es necesario analizar la complejidad de las operaciones que lo componen.
Se va a calcular la complejidad temporal en el caso peor.

Instrucciones simples: se ejecutan una sola vez y tienen un coste constante O(1):

- Expresiones aritméticas con datos de tamaño constante.
- Comparaciones de datos simples.
- Asignación de datos simples.
- Lectura de datos simples.
- Escritura de datos simples.

Instrucciones compuestas: se aplica la regla de la suma, es decir:

$$T1,2(n) = T1(n) + T2(n) = \max(T1(n),T2(n))$$

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEIDAD

Instrucciones de selección (IF/CASE):

if condición I1; else I2;

La condición se evalua siempre, por lo que hay que añadir su coste Tcond(n) al peor de los dos posibles casos:

Tif(n)= Tcond(n) + Max(TI1(n),TI2(n))

```
switch (expresión) {
    case valor1:        instrucciones_1; break;
    case valor2:        instrucciones_3; break;
    ..................
case valork:        instrucciones_k; break;
};
```

Tswitch(n)= Texp(n) + Max(TI1(n),…,TIk(n))

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEIDAD

Instrucciones iterativas (FOR):

```
FOR k  in 1..n {
        I
};
```

El coste es el producto del número de iteraciones por el coste de las instrucciones del cuerpo del bucle:

Tfor(n)= multiplicacion(TI(n))

Con WHILE y DO se calcula de forma análoga.

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEIDAD

Subprogramas:

- **El tratamiento es diferente según se trate de subprogramas recursivos o no recursivos.**

- **Subprogramas no recursivos: el coste se calcula en función de las instrucciones que lo integran.**

- **Subprogramas recursivos: habrá que diferenciar los casos base de los recurrentes.**

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEJIDAD (EJEMPLOS)

- for (int i= 0; i < K; i++) {O(1) } → K*O(1) = O(1)

- for (int i= 0; i < n; i++) {O(1) } → n * O(1) = O(n)

- for (int i= 0; i < n; i++) {
            for (int j= 0; j < n; j++) {
                        O(1)
            }
  } → n * n * O(1) = O(n2)

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEJIDAD (EJEMPLOS)

```
c= 1;
while (c < N){
        O(1);
        c= 2**c;
 }
```

El valor incial de c es 1, siendo 2k al cabo de "k" iteraciones. El número de iteraciones es tal que

2k >= N => k= (log2 (N))**

[el entero inmediato superior] y, por tanto, la complejidad del bucle es O(log n).

COMPLEJIDAD TEMPORAL

CÁLCULO DE LA COMPLEJIDAD (EJEMPLOS)

```
for (int i= 0; i < N; i++) {
        c= i;
        while (c > 0) {
                O(1)
                c= 2**c;
        }
}
```

Tenemos un bucle interno de orden O(log n) que se ejecuta N veces, luego el conjunto es de orden O(n log n)

ALGORITMO BÚSQUEDA LINEAL	$O(n)$
ALGORITMO BÚSQUEDA BINARIA O DICOTÓMICA	$O(\log n)$
ALGORITMO DE INSERCIÓN DIRECTA	$O(n^2)$
ALGORITMO DE SELECCIÓN DIRECTA	$O(n^2)$
ALGORITMO DE INTERCAMBIO (BURBUJA)	$O(n^2)$
ALGORITMO SACUDIDA	$O(n^2)$
ALGORITMO SHELL (INCREMENTOS DECRECIENTES)	$O(n^{1.25})$ / $O(n^2)$
ALGORITMO QUICKSORT	$O(n \log n)$ / $O(n^2)$