Cálculo de complejidades

Órdenes de complejidades

Órdenes de complejidad

- Las funciones de complejidad algorítmica más habituales en las cuales el único factor del que dependen es el tamaño de la muestra de entrada *n*, ordenadas de mayor a menor eficiencia son:

$O(1)$	Orden constante
$O(\log n)$	Orden logarítmico
$O(n)$	Orden lineal
$O(n \log n)$	Orden cuasi-lineal
$O(n^2)$	Orden cuadrático
$O(n^3)$	Orden cúbico
$O(n^a)$	Orden polinómico
$O(2^n)$	Orden exponencial
$O(n!)$	Orden factorial

- **O(1):** Complejidad constante. Cuando las instrucciones se ejecutan una vez.

- **O(*log n*):** Complejidad logarítmica. Esta suele aparecer en determinados algoritmos con iteración o recursión no estructural, ejemplo la búsqueda binaria.

- **O(*n*):** Complejidad lineal. Es una complejidad buena y también muy usual. Aparece en la evaluación de bucles simples siempre que la complejidad de las instrucciones interiores sea constante.

- **O($n \log n$):** Complejidad cuasi-lineal. Se encuentra en algoritmos de tipo divide y vencerás como por ejemplo en el método de ordenación Quicksort y se considera una buena complejidad.
  Si *n* se duplica, el tiempo de ejecución es ligeramente mayor del doble.

- **O(n^2):** Complejidad cuadrática. Aparece en bucles o ciclos doblemente anidados. Si n se duplica, el tiempo de ejecución aumenta cuatro veces.

- **O(n^3):** Complejidad cúbica. Suele darse en bucles con triple anidación. Si *n* se duplica, el tiempo de ejecución se multiplica por ocho. Para un valor grande de *n* empieza a crecer dramáticamente.

- **$O(n^a)$:** Complejidad polinómica ($a > 3$).
  Si a crece, la complejidad del programa es bastante mala.

- **$O(2^n)$:** Complejidad exponencial. No suelen ser muy útiles en la práctica por el elevadísimo tiempo de ejecución. Se dan en subprogramas recursivos que contengan dos o más llamadas internas.

Ejemplo - Mensajes

```
#define N 10
int main()
{
        int c[N] = {10,7,8,6,9,9,8,10,8,7};
        int i;

        for (i=0; i<5; i++)
        {
                printf("paso %d", i);
        }

        return 0;
}
```

Complejidad constante
O(1)

Ejemplo - Promedio

```
#define N 10
int main()
{
        int c[N] = {10,7,8,6,9,9,8,10,8,7};
        int i;
        float suma = 0, promedio = 0;

        for (i=0; i<N; i++)
        {
                suma = suma+c[i];
        }
        promedio = suma/N;
        printf("El promedio es %.2f \n",promedio);

        return 0;
}
```

Complejidad lineal
O(n)

Ejemplo – Suma de matrices

```
void sumaMatrices(int a[][N], int b[][N], int N)
{
        int c[N][N];
        int i, j;
         for (i = 0; i < N; i++)
        {
                for (j = 0; j < N; j++)
                {
                        c[i][j] = a[i][j] + b[i][j];
                        printf("%i\t",c[i][j]);
                }
                printf("\n");
        }
}
```

Complejidad cuadrática
$O(n^2)$

Análisis de casos

Algoritmo de inserción

Ordenamiento por inserción

Uno de los métodos de ordenamiento más sencillos es el ordenamiento por inserción directa. Se tiene una secuencia de números x_1, x_2,..., x_n. Los números se recorren de izquierda a derecha y se escribe x_i a la izquierda de x_{i-1} si x_i es menor que x_{i-1}. En otras palabras, desplazamos a x_i de manera continua hacia la izquierda hasta que los números a su izquierda sean menores que o iguales a él.

Algoritmo 2-1 □ Ordenamiento por inserción directa

Input: $x_1, x_2, \ldots, x_n$.

Output: La secuencia ordenada de $x_1, x_2, \ldots, x_n$.

```
For j := 2 to n do
Begin
    i := j − 1
    x := x_j
    While x < x_i and i > 0 do
    Begin
        x_{i+1} := x_i
        i := i − 1
    End
    x_{i+1} := x
End
```

Considere la secuencia de entrada 7, 5, 1, 4, 3, 2, 6. El ordenamiento por inserción directa produce la secuencia ordenada siguiente:

7

5, 7

1, 5, 7

1, 4, 5, 7

1, 3, 4, 5, 7

1, 2, 3, 4, 5, 7

1, 2, 3, 4, 5, 6, 7.

En nuestro análisis, como medida de la complejidad temporal del algoritmo se usará el número de intercambios de datos $x := x_j$, $x_{i+1} := x_i$ y $x_{i+1} := x$. En este algoritmo hay dos ciclos: uno exterior (**for**) y otro interior (**while**). Para el ciclo exterior siempre se ejecutan dos operaciones de intercambio de datos; a saber, $x := x_j$ y $x_{i+1} := x$. Debido a que el ciclo interior puede o no ser ejecutado, el número de intercambios de datos realizados para x_i en el ciclo interior se denotará por d_i. Así, evidentemente, el número total de movimientos de datos para el ordenamiento por inserción directa es

$$X = \sum_{i=2}^{n} (2 + d_i)$$

$$= 2(n-1) + \sum_{i=2}^{n} d_i$$

Esto ocurre cuando los datos de entrada ya están ordenados.

Mejor caso: $\sum_{i=2}^{n} d_i = 0$, $X = 2(n-1) = O(n)$

Fórmulas

$$\sum_{i=1}^{n} (a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$

$$\sum_{i=1}^{n} c = nc$$

Peor caso: El peor caso ocurre cuando los datos de entrada están ordenados a la inversa. En este caso,

$$\begin{aligned} d_2 &= 1 \\ d_3 &= 2 \\ &\vdots \\ d_n &= n-1. \end{aligned}$$

De este modo,

$$\sum_{i=2}^{n} d_i = \frac{n}{2}(n-1)$$

$$X = 2(n-1) + \frac{n}{2}(n-1) = \frac{1}{2}(n-1)(n+4) = O(n^2).$$

Caso promedio: Cuando se está considerando x_i, ya se han ordenado $(i - 1)$ datos. Si x_i es el más grande de todos los i números, entonces el ciclo interior no se ejecuta y dentro de este ciclo interior no hay en absoluto ningún movimiento de datos. Si x_i es el segundo más grande de todos los i números, habrá un intercambio de datos, y así sucesivamente. La probabilidad de que x_i sea el más grande es $1/i$. Esta también es la probabilidad de que x_i sea el j-ésimo más grande para $1 \le j \le i$. En consecuencia, el promedio $(2 + d_i)$ es

$$\frac{2}{i} + \frac{3}{i} + \cdots + \frac{i+1}{i} = \sum_{j=1}^{i} \frac{(j+1)}{i}$$

$$= \frac{i+3}{2}.$$

La complejidad temporal media para el ordenamiento por inserción directa es

$$\sum_{i=2}^{n} \frac{i+3}{2} = \frac{1}{2}\left(\sum_{i=2}^{n} i + \sum_{i=2}^{n} 3\right)$$

$$= \frac{1}{4}(n-1)(n+8) = O(n^2).$$

Fórmulas

$$\sum_{i=1}^{n} ca_i = c \sum_{i=1}^{n} a_i$$

En resumen, la complejidad temporal del ordenamiento por inserción directa para cada caso es:

Mejor caso: $2(n-1) = O(n)$.

Caso promedio: $\frac{1}{4}(n+8)(n-1) = O(n^2)$.

Peor caso: $\frac{1}{2}(n-1)(n+4) = O(n^2)$.