Análisis de algoritmos

Complejidad en el tiempo y eficiencia

Eficiencia

- Es la propiedad mediante la cual un algoritmo debe alcanzar la solución al problema en el tiempo más corto posible y/o utilizando la cantidad más pequeña posible de recursos físicos.

- La eficiencia se puede medir a través del **análisis de algoritmos**.

- El formato de la eficiencia se puede expresar mediante una función:
  - **f(n) = eficiencia**.

Ejemplos

```
int n=5;
int arreglo[n] = {1,2,3,4,5};
int suma = 0;

for(int i=0; i<n; i++) {
    suma = suma + arreglo[i];
}
```

Para este caso **f(n) = n**, ya que la eficiencia es directamente proporcional al número de iteraciones.

Complejidad en el tiempo

- Es la cantidad de tiempo de computadora que necesita un algoritmo para ejecutarse.

- Utilizamos una función que llamaremos T(n), para representar el número de unidades de tiempo tomadas por un algoritmo para cualquier entrada de tamaño n.

- Si la función es T(n) = c * n, decimos que el tiempo de ejecución es lineal.

Ejemplo

```
int sumatoria(int arreglo[], int n) {
    int suma = 0;                    // tiempo t1

    for(int i=0; i<n; i++) {         // tiempo t2
        suma = suma + arreglo[i];    // tiempo t3
    }
    return suma;                     // tiempo t4
}
```

Tenemos que:

$T(n) = t1 + (n * t2) + (n * t3) + t4$

Ejemplo

```
int sumatoria(int arreglo[], int n) {
    int suma = 0;                    // tiempo t1

    for(int i=0; i<n; i++) {         // tiempo t2
        suma = suma + arreglo[i];    // tiempo t3
    }
    return suma;                     // tiempo t4
}
```

Tenemos que:

T(n) = t1 + (n * t2) + (n * t3) + t4

- Si T(n) es el tiempo de ejecución de un programa con entrada de tamaño n, será posible analizar T(n) como el número de sentencias ejecutadas por el programa.

Complejidad en el tiempo

Podemos realizar el análisis bajo tres posibles casos:

- **Peor caso**: es el peor tiempo que un algoritmo puede tener.

- **Mejor caso**: el mejor tiempo que un algoritmo puede tener.

- **Caso medio**: consiste en calcular T(n) como el tiempo medio de ejecución del programa sobre todas las posibles ejecuciones de entradas de tamaño n.

Notación O-Grande

- Se suele expresar O(n) y representa el “orden de”.
- La podemos leer como “O de n”.
- También conocida como “notación Big-O” ó “notación asintótica”.
- Expresa la complejidad, en términos de tendencia.

Notación O-Grande

Si $f(n) = n^2 - 2n + 3$ **entonces** $O(n) = n^2$.

O(n) expresa una aproximación de la relación existente entre el tamaño de un problema y la cantidad de proceso necesario para hacerlo.

Notación O-Grande

- **O(1)** = orden constante.
- **O(log n)** = orden logarítmico.
- **O(n)** = orden lineal.
- **O(n logn)** = orden logarítmico lineal.
- **O(n^2)** = orden cuadrático.
- **O(n^3)** = orden cúbico.
- **O(2^n)** = orden exponencial.
- **O(n!)** = orden factorial.