INTRODUCCIÓN AL ANÁLISIS DE TIEMPO

La **figura 1** demuestra el concepto de *algoritmo.*

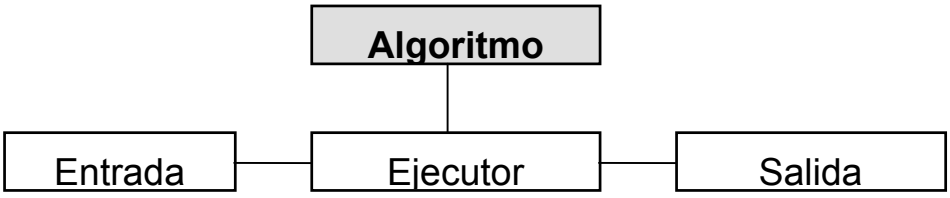

Figura 1: Concepto de Algoritmo

Un algoritmo es una abstracción de un programa y un programa es una implantación de un algoritmo.

El objetivo del análisis de tiempo es predecir cuánto tiempo toma un programa con una entrada específica para ser ejecutado sin tener que ejecutarlo. Problemas que se presentan:

- el tiempo de ejecución depende del computador,
- el tiempo de ejecución depende del sistema operativo y
- el tiempo de ejecución depende del compilador.

El tiempo de ejecución depende además de la entrada específica; demora más tiempo el ordenamiento de 1.000.000 que de 10 elementos. Si la entrada específica tiene la misma cantidad de elementos se pueden distinguir problemas simples y problemas no tan simples.

La cantidad de elementos de entrada ***n*** se denomina ***el tamaño*** del problema. Para calcular el tiempo que se ocupa para ejecutar un programa se asocia con el tamaño a través de la formula ***t = f(n)***, donde ***f*** es generalmente denominado la ***complejidad*** de tiempo.

El número ***n*** representa una medida para determinar la dimensión del problema a resolver. Por ejemplo, si se quiere ordenar una lista de elementos la medida para determinar la dimensión de este problema es la cantidad ***n*** de elementos que contiene la lista.

La comparación de dos algoritmos de búsqueda lleva a los siguientes valores:

- Algoritmo 1 - Búsqueda secuencial
- Algoritmo 2 - Búsqueda binaria.

Para el primer algoritmo se puede establecer que en el peor caso el tiempo para buscar un elemento es t = c * n. El tiempo depende de:

- c es una constante que representa las características del computador y otros factores que son independientes del algoritmo y
- n que es el tamaño de la lista.

Este tipo de formula no se utiliza en esta forma en la comparación de algoritmos porque es difícil (imposible) determinar el valor exacto de c. Se usa una fórmula que se denomina notación "O" mayúscula, lo que significa el ***orden de complejidad***:

$$t = O[f(n)].$$

Esto significa que el tiempo de ejecución es del orden O[f(n)]. Esto quiere decir que existen dos constantes M y n' de tal forma que si existe t = O[f(n)], entonces t < M*f(n) para todo n > n'.

Ejemplo: t(0) = 1, t(1) = 4, t(n) = $(n+1)^2$. Entonces: t = $O(n^2)$, si se asigna n' = 1 y M = 4. Así para $n \geq 1$ se obtiene $(n+1)^2 \leq 4\ n^2$.

El objetivo del análisis de tiempo de algoritmos es encontrar los algoritmos con el menor orden de complejidad posible que resuelven el mismo problema.

El algoritmo 1 de la búsqueda secuencial tiene claramente el orden de complejidad O(n).

Para el algoritmo 2 de la búsqueda binaria se obtiene lo siguiente: Después de cada comparación, el tamaño de la lista a ordenar se disminuye a la mitad, tomando los siguientes tamaños: 1, ½, ¼, 1/8, 1/16, etc. En el peor caso esta búsqueda termina si se llega a la lista vacía.

El número mayor de comparaciones es k, donde k es definido como el primer número entero que cumple la ecuación $2^k \geq n$. Esto se puede transformar en: $k \geq \log_2 n$ y así el orden de complejidad para la búsqueda binaria es $O(\log_2 n)$.

La comparación de los dos algoritmos de búsqueda muestra: Si el tamaño es n = 50.000, la búsqueda secuencial requiere en el peor caso 50.000 comparaciones, mientras la búsqueda binaria no más que $\log_2 50.000$, que es aproximadamente 16. Se observa una mejora de un factor de 3.125. Esta mejora se logra sólo si la lista se encuentra ordenada antes de aplicar la búsqueda binaria.

Un segundo ejemplo analiza dos algoritmos de ordenamiento:

- Algoritmo 3 - Ordenamiento por intercambio
- Algoritmo 4 - Ordenamiento por mezcla.

En el algoritmo 3 se busca el elemento mayor y este se intercambia con el primero de la lista. Después se avanza buscando el mayor de la lista sin considerar el primero y se intercambia el mayor encontrado con el nuevo primero. Este proceso se repite (n-1) veces.

Para encontrar el mayor elemento la primera vez se requiere analizar n elementos. Para el segundo mayor se requiere analizar (n-1), para el tercero (n-2), etc. La cantidad total es entonces:

$$n + (n-1) + (n-2) + ...+ 2 = \sum i \ (i=2,...,n)$$

$$½\ (n\ (n+1)) – 1 = ½\ n^2 + ½\ n – 1$$

Para problemas de grandes tamaños el término n^2 domina a los otros términos y es por eso que el orden de complejidad es $O(n^2)$.

Un concepto fundamental en el análisis del algoritmos consiste en que se gana mucho más en eficiencia tomando un mejor algoritmo que tratar de mejorar un algoritmo ineficiente.

Un mejor algoritmo para el ordenamiento es el algoritmo 4 *ordenamiento por mezcla (merge sort).* Se tienen **n** elementos en **m** grupos, es decir **n/m** elementos por cada grupo.

OOOO OOOO OOOO OOOO ..... OOOO

El tiempo de ordenamiento para cada grupo es el ya conocido orden $O(n^2/m^2)$ si se ordena a través del algoritmo ordenamiento por intercambio, y el tiempo total para los **m** grupos es m veces mayor: $O(n^2/m)$. El tiempo que se requiere para mezclar los grupos ordenados en un solo grupo ordenado es del orden O(mn). Esto se debe a que para cada uno de los **n** elementos hay que buscar en cada uno de los **m** grupos, cuál es el mayor de todos. Así el tiempo total es del orden $O(n^2/m+nm)$.

Ahora falta fijar el valor de **m**, lo que significa determinar en cuantos grupos hay que dejar los **n** elementos a ordenar. Las dos alternativas extremas son:

m = 1, así el tiempo total sería $O(n^2)$. Con esto no se gana nada.

m = n, así el tiempo total también es $O(n^2)$.

El valor óptimo es $m = n^{1/2}$, es decir la raíz de n. De esta manera se obtiene para el tiempo total tiene el orden de complejidad $O(n^{3/2})$.

La siguiente tabla da una impresión de la mejora del algoritmo ordenamiento por mezcla.

Tamaño n	$O(n^2)$	$O(n^{3/2})$	Mejora
10	100	32	3,1 veces
100	10.000	1.000	10 veces
1.000	1.000.000	32.000	31,3 veces
10.000	100.000.000	1.000.000	100 veces