Análisis de los algoritmos

Un algoritmo es una secuencia de pasos lógica para encontrar la solución de un problema.

El análisis de algoritmos consiste en predecir la cantidad de recursos (tiempo, memoria, comunicación) que un algoritmo requerirá para cualquier entrada.

- Esto nos permite comparar los costos relativos de dos o más algoritmos para resolver el mismo problema.
- El análisis de algoritmos también les da una herramienta a los diseñadores de algoritmos para estimar si una solución propuesta es probable que satisfaga las restricciones de recursos de un problema.
- El concepto de razón de crecimiento, es la razón a la cual el costo de un algoritmo crece conforme el tamaño de la entrada crece.

Todo algoritmo debe contar con las siguientes características: preciso, definido y finito. Por Preciso, entenderemos que cada paso del algoritmo tiene una relación con el anterior y el siguiente; un algoritmo es Definido, cuando se ejecuta más de una vez con los mismos datos y el resultado es el mismo; y Finito, indica que el algoritmo cuenta con una serie de pasos definidos o que tiene un fin.

Hablando de estructuras de datos podemos decir que los algoritmos según su función se dividen en:

- Algoritmos de ordenamiento
- Algoritmos de búsqueda.

Un algoritmo de ordenamiento, es el que pone los elementos de una lista o vector en una secuencia (ascendente o descendente) diferente a la entrada, es decir, el resultado de salida debe ser una permutación (reordenamiento) de la entrada que satisfaga la relación de orden requerida.

Un algoritmo de búsqueda, es aquel que está diseñado para encontrar la solución de un problema boleano de existencia o no de un elemento en particular dentro de un conjunto finito de elementos (estructura de datos), es decir al finalizar el algoritmo este debe decir si el elemento en cuestión existe o no en ese conjunto, además, en caso de existir, el algoritmo podría proporcionar la localización del elemento dentro del conjunto.

Concepto de complejidad de algoritmos.

La mayoría de los problemas que se plantean en la actualidad se pueden resolver con algoritmos que difieren en su eficiencia. Dicha diferencia puede ser irrelevante cuando el número de datos es pequeño pero cuando la cantidad de datos es mayor la diferencia crece. Ejemplo: Suma de 4 y 10 primero números naturales.

1+2+3+4 = 10	3	3	4*(4+1)/2 = 10
	tiempo		
1+2+3+4+5+6+7+8+9+10 = 55	9	3	10*(10+1)/2 = 55

La complejidad de un algoritmo o complejidad computacional, estudia los recursos y esfuerzos requeridos durante el cálculo para resolver un problema los cuales se dividen en: tiempo de ejecución y espacio en memoria. El factor tiempo, por lo general es más importante que el factor espacio, pero existen algoritmos que ofrecen el peor de los casos en un menor tiempo que el mejor de los casos, lo cual no es la mejor de las soluciones.

El factor tiempo de ejecución de un algoritmo depende de la cantidad de datos que se quieren procesar, una forma de apreciar esto es con las cuatro curvas que se presentan en las gráficas de la figura 1.1.

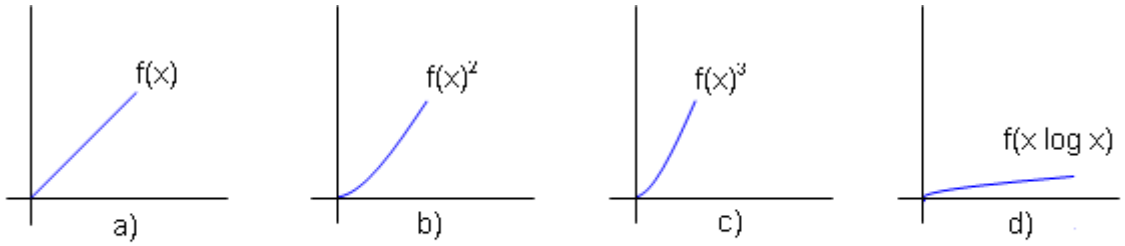

Figura 1.1 Funciones típicas para el análisis de un algoritmo.

Como se puede apreciar en las gráficas, entre mayor se al número de datos mayor tiempo se aplica en las gráficas a), b) y c), lo cual no ocurre con la gráfica d), por lo tanto podemos deducir que una función que se acerque más al eje de las x es más constante y eficiente en el manejo de grandes cantidades de datos.

Complejidad.

Tiempo de ejecución de un algoritmo.

El tiempo de ejecución de un algoritmo, se refiere a la suma de los tiempos en los que el programa tarda en ejecutar una a una todas sus instrucciones, tomando en cuenta que cada instrucción requiere una unidad de tiempo, dicho tiempo se puede calcular en función de n (el número de datos), lo que se denomina T(n)

Si hacemos un análisis de forma directa al programa para determinar el tiempo de ejecución del mismo, debemos definir el conjunto de operaciones primitivas, que son independientes del lenguaje de programación que se use. Algunas de las funciones primitivas son las siguientes:

- Asignación de un valor a una variable.
- Llamada a un método.
- Ejecución de una operación aritmética.
- Comparar dos números.
- Poner índices a un arreglo.
- Seguir una referencia de objeto.
- Retorno de un método.

El tiempo de Ejecución de un programa se mide en función de *N*, lo que designaremos como *T(N).*

Esta función se puede calcular físicamente ejecutando el programa acompañados de un reloj, o calcularse directamente sobre el código, contando las instrucciones a ser ejecutadas y multiplicando por el tiempo requerido por cada instrucción. Así, un trozo sencillo de código como:

S1;

for(x = 0; x < N; x++)

S2;

Demanda: *T(N)* = t1 + t2 * N

Donde t1 es el tiempo que lleva ejecutar la serie S1 de sentencias, y t2 es el que lleva la serie S2.

Habitualmente todos los algoritmos contienen alguna sentencia condicional o selectiva, haciendo que las sentencias ejecutadas dependan de la condición lógica, esto hace que aparezca más de un valor para *T(N),* es por ello que debemos hablar de un rango de valores:

Tmin(N) ≤ T(N) ≤ Tmax(N)

- Estos extremos son llamados "*el peor caso*" y "*el mejor caso*" y entre ambos se puede hallar "*el caso promedio*" o el más frecuente, siendo este el más difícil de estudiar; nos centraremos en el "*el peor caso*" por ser de fácil cálculo y se acerca a "*el caso promedio*", brindándonos una medida pesimista pero fiable.
- Toda función *T(N)* encierra referencias al parámetro *N,* y a una serie de constantes *Ti* dependientes de factores externos al algoritmo. Se tratará de analizar los algoritmos dándoles autonomía frente a estos factores externos, buscando estimaciones generales ampliamente válidas, a pesar de ser demostraciones teóricas.

Complejidad en espacio.

La complejidad de espacio, se refiere a la memoria que utiliza un programa para su ejecución; es decir el espacio de memoria que ocupan todas las variables propias del programa. Dicha memoria se divide en Memoria estática y Memoria dinámica. Es la memoria que utiliza un programa para su ejecución; es decir el espacio de memoria que ocupan todas las variables propias del algoritmo.

Para calcular la memoria estática, se suman la cantidad de memoria que ocupa cada una de las variables declaradas en el programa.

Tomando en cuenta los tipos de datos primitivos del lenguaje de programación java podemos determinar el espacio que requiere cada una de las variables de un programa, de acuerdo a lo siguiente:

Tipo de dato primitivo	Tamaño en bits	Tamaño en Bytes
byte	8	1
char	16	2
short	16	2

int	32	4
float	32	4
long	64	8
double	64	8

El cálculo de la memoria dinámica, no es tan simple ya que depende de cada ejecución del programa o algoritmo y el tipo de estructuras dinámicas que se estén utilizando.

Selección de un algoritmo.

Una de las características primordiales en la selección de un algoritmo es que este sea sencillo de entender, calcular, codificar y depurar, así mismo que utilice eficientemente los recursos de la computadora y se ejecute con la mayor rapidez posible con un eficaz uso de memoria dinámica y estática.

Ejemplo: algoritmo de búsqueda en arboles.

Función búsqueda _arboles.

Devuelve una solución o fallo.

Inicializa un árbol de búsqueda con estado inicial.

Bucle hacer

Si no hay candidatos para expandir.

Entonces devolver fallo.

En otro caso escoger nodo para expandir.

Si el nodo es el objetivo.

Entonces devolver solución.

En otro caso expandir nodo.

M = profundidad máxima del árbol (puede ser infinita)

D = profundidad de la mejor solución (menor costo)

B = factor de ramificación (número máximo de sucesiones) = 10

Depth	Nodes	Time	Memory
0	1	1 milisecond	100 bytes

2	111	.1 second	11 Kb
4	11111	11 second	1 Mb
6	10 ^ 6	18 minutos	111 Mb

Eficiencia de los algoritmos.

La Eficiencia de un algoritmo es comprobar que dicho análisis debe ser empírico y teórico, en cualquiera de los diferentes métodos de los algoritmos de ordenamiento, los cuales son:

- Ø Bubble sort
- Ø Selection sort
- Ø Insertion sort
- Ø Shell sort
- Ø Ha sort
- Ø Merge sort
- Ø Quick sort

¿Por qué estudiar la eficiencia de los algoritmos? Porque nos sirve para establecer la frontera entre lo factible y lo imposible.

- Ejemplo: Algoritmos de ordenación Observación El tiempo de ejecución depende del tamaño del conjunto de datos. Objetivo Parametrizar el tiempo de ejecución en función del tamaño del conjunto de datos, intentando buscar una cota superior que nos sirva de garantía.

Tipos de Análisis

- Peor de los Casos: Se corresponde con el peor tiempo. T(n) es el tiempo máximo sobre las entradas.
- Mejor Caso: Límite inferior en el tiempo. T(n) es el menor tiempo de todas las posibles entradas
- Caso Promedio: Es el tiempo medio esperado sobre todas las posibles entradas de tamaño n. Se considera una distribución de probabilidad sobre las entradas.
- Análisis Probabilístico: Es el tiempo de ejecución esperado para una entrada aleatoria. Se expresa tanto el tiempo de ejecución y la probabilidad de obtenerlo
  - Análisis Amortizado: El tiempo que se obtiene para un conjunto de ejecuciones, dividido por el número de ejecuciones.

También para seleccionar correctamente el mejor algoritmo es necesario realizar estas preguntas:

¿Qué grado de orden tendrá la información que vas a manejar?

Si la información va a estar casi ordenada y no quieres complicarte, un algoritmo sencillo como el ordenamiento burbuja será suficiente. Si por el contrario los datos van a estar muy desordenados, un algoritmo poderoso como Quicksort puede ser el más indicado. Y si no puedes hacer una presunción sobre el grado de orden de la información, lo mejor será elegir un algoritmo que se comporte de manera similar en cualquiera de estos dos casos extremos.

¿Qué cantidad de datos vas a manipular?

Si la cantidad es pequeña, no es necesario utilizar un algoritmo complejo, y es preferible uno de fácil implementación. Una cantidad muy grande puede hacer prohibitivo utilizar un algoritmo que requiera de mucha memoria adicional.

¿Qué tipo de datos quieres ordenar?

Algunos algoritmos sólo funcionan con un tipo específico de datos (enteros, enteros positivos, etc.) y otros son generales, es decir, aplicables a cualquier tipo de dato.

¿Qué tamaño tienen los registros de tu lista?

Algunos algoritmos realizan múltiples intercambios (burbuja, inserción). Si los registros son de gran tamaño estos intercambios son más lentos.

Bibliografías:

https://sites.google.com/site/estdatjiq/home/unidad-vii

http://rhomarycristobal.blogspot.com/2011/12/septima-unidad-analisis-de-algoritmos.html