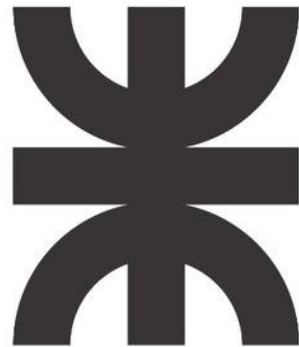


**Universidad Tecnológica
Nacional**

Facultad Regional Tucumán



**Algoritmos y Estructuras de
Datos**

Trabajo Integrador – Monografía

Integrantes:

Comisión: 1k02

- Arteaga Lucas
- Giacobbe Franco
- Delgado Jorge Luis
- Jiménez Fabricio

Enunciados propuestos

1) ¿Qué es el Análisis Algorítmico?

→ El análisis de algoritmos es una parte importante de la Teoría de complejidad computacional, que provee estimaciones teóricas para los recursos que necesita cualquier algoritmo que resuelva un problema computacional dado. Estas estimaciones resultan ser bastante útiles en la búsqueda de algoritmos eficientes. A la hora de realizar un análisis teórico de algoritmos es común calcular su complejidad en un sentido asintótico, es decir, para un tamaño de entrada suficientemente grande. La cota superior asintótica, y las notaciones omega (cota inferior) y theta (caso promedio) se usan con esa finalidad. Por ejemplo, la búsqueda binaria decimos que se ejecuta en una cantidad de pasos proporcional a un logaritmo, en $O(\log(n))$, coloquialmente "en tiempo logarítmico". Normalmente las estimaciones asintóticas se utilizan porque diferentes implementaciones del mismo algoritmo no tienen por qué tener la misma eficiencia. No obstante, la eficiencia de dos implementaciones "razonables" cualesquiera de un algoritmo dado están relacionadas por una constante multiplicativa llamada constante oculta. La medida exacta (no asintótica) de la eficiencia a veces puede ser computada, pero para ello suele hacer falta aceptar supuestos acerca de la implementación concreta del algoritmo, llamada modelo de computación. Un modelo de computación puede definirse en términos de un ordenador abstracto, como la Máquina de Turing, y/o postulando que ciertas operaciones se ejecutan en una unidad de tiempo. Por ejemplo, si al conjunto ordenado al que aplicamos una búsqueda binaria tiene 'n' elementos, y podemos garantizar que una única búsqueda binaria puede realizarse en un tiempo unitario, entonces se requieren como mucho $\log_2 N + 1$ unidades de tiempo para devolver una respuesta. Las medidas exactas de eficiencia son útiles para quienes verdaderamente implementan y usan algoritmos, porque tienen más precisión y así les permite saber cuánto tiempo pueden suponer que tomará la ejecución. Para algunas personas, como los desarrolladores de videojuegos, una constante oculta puede significar la diferencia entre éxito y fracaso. Las estimaciones de tiempo dependen de cómo definamos un paso. Para que el análisis tenga sentido, debemos garantizar que el tiempo requerido para realizar un paso se encuentre acotado superiormente por una constante. Hay que mantenerse precavido en este terreno; por ejemplo, algunos análisis cuentan con que la suma de dos números se hace en un paso. Este supuesto puede no estar garantizado en ciertos contextos. Si por ejemplo los números involucrados en la computación pueden ser arbitrariamente grandes, dejamos de poder asumir que la adición requiere un tiempo constante (usando papel y lápiz, compara el tiempo que necesitas para sumar dos enteros de 2 dígitos cada uno y el necesario para hacerlo con dos enteros, pero de 1000 dígitos cada uno).

2) Definir Orden de un Algoritmo.

→ El orden mide cuan rápidamente aumenta el tiempo de ejecución de un algoritmo cuando aumenten los datos de entrada. Es decir, que, si para una lista de 100 elementos el algoritmo tarda x segundos, para una lista de 1000 elementos (10 veces más grande) tardará 10 veces más.

3) Analizar los siguientes métodos de ordenamiento:

a. Intercambio o burbuja mejorada

Como ya sabemos mediante el método burbuja, dado un arreglo de n números, se requiere de $n-1$ pasos para dejar el arreglo ordenado. Se puede observar que en el primer paso el primer elemento mayor queda en la primera posición mayor (última si es que estamos ordenando de menor a mayor); en el segundo paso el segundo elemento mayor queda en la segunda posición mayor (penúltima); y así sucesivamente. Por esta razón el número de comparaciones, debería irse reduciendo en uno, en cada paso. Se puede observar, además, que en muchos casos se consigue tener ordenado el arreglo, en un número menor de pasos a $n-1$, por lo cual el resto de los pasos serían innecesarios. Considerando estos dos hechos se podría mejorar el método burbuja eliminando los pasos innecesarios y reduciendo las comparaciones con cada paso. Una manera sencilla de hacer esto sería detectando mediante algún registro si se han efectuado cambios o no, y reduciendo el número de comparaciones en cada paso.

b. Inserción o método de la baraja

El método de inserción directa es el que generalmente utilizan los jugadores de cartas cuando ordenan éstas, de ahí que también se conozca con el nombre de método de la baraja.

DESCRIPCIÓN.

El algoritmo de ordenación por el método de inserción directa es un algoritmo relativamente sencillo y se comporta razonablemente bien en gran cantidad de situaciones. Completa la triplete de los algoritmos de ordenación más básicos y de orden de complejidad cuadrático, junto con Selection Sort y Bubble Sort. Se basa en intentar construir una lista ordenada en el interior del array a ordenar. De estos tres algoritmos es el que mejor resultado da a efectos prácticos. Realiza una cantidad de comparaciones bastante equilibrada con respecto a los intercambios, y tiene un par de características que lo hacen aventajar a los otros dos en la mayor parte de las situaciones. Este algoritmo se basa en hacer comparaciones, así que para que realice su trabajo de ordenación son imprescindibles dos cosas: un array o estructura similar de elementos comparables y un criterio claro de comparación, tal que dados dos elementos nos diga si están en orden o no. En cada iteración del ciclo externo los elementos 0 a i forman una lista ordenada.

ANÁLISIS DEL ALGORITMO.

- + Estabilidad: Este algoritmo nunca intercambia registros con claves iguales. Por lo tanto, es estable.
- + Requerimientos de Memoria: Una variable adicional para realizar los intercambios.
- + Tiempo de Ejecución: Para una lista de n elementos el ciclo externo se ejecuta $n-1$ veces. El ciclo interno se ejecuta como máximo una vez en la primera iteración, 2 veces en la segunda, 3 veces en la tercera, etc.

Ventajas:

- + Fácil implementación.
- + Requerimientos mínimos de memoria.

Desventajas:

- + Lento.
- + Realiza numerosas comparaciones.

Este también es un algoritmo lento, pero puede ser de utilidad para listas que están ordenadas o semi-ordenadas, porque en ese caso realiza muy pocos desplazamientos.

c. Selección o método sencillo

Este algoritmo consiste en encontrar el menor de todos los elementos del arreglo o vector e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenarlo todo. Este algoritmo mejora ligeramente el algoritmo de la burbuja. En el caso de tener que ordenar un vector de enteros, esta mejora no es muy sustancial, pero cuando hay que ordenar un vector de estructuras más complejas, la operación de intercambiar los elementos sería más costosa en este caso.

Descripción del algoritmo

- * Buscar el mínimo elemento de la lista
- * Intercambiarlo con el primero
- * Buscar el siguiente mínimo en el resto de la lista
- * Intercambiarlo con el segundo

Y en general:

- * Buscar el mínimo elemento entre una posición i y el final de la lista
- * Intercambiar el mínimo con el elemento de la posición i

Ventajas del algoritmo

- * Es fácil su implementación.
- * No requiere memoria adicional.
- * Realiza pocos intercambios.
- * Tiene un rendimiento constante, pues existe poca diferencia entre el peor y el mejor caso.

Desventajas del algoritmo

- * Es lento y poco eficiente cuando se usa en listas grandes o medianas.
- * Realiza numerosas comparaciones.

d. Rápido o QuickSort

La técnica Quicksort: declara las funciones `qs(recursivo)` y `QuickSort` para ordenar una secuencia de números, imprime el arreglo separado por comas.

¿Como funciona?

Se tiene un array de n elementos, tomamos un valor del array como pivote (usualmente el primero), separamos los elementos menores a este pivote a la izquierda y los mayores a la derecha, es decir, dividimos el array en 2 subarrays.

Con estos subarrays se repite el mismo proceso de forma recursiva hasta que estos tengan más de 1 elemento.

Ejemplos de cómo funciona:

En el siguiente ejemplo se marcan el pivote y los índices i y j con las letras p , i y j respectivamente.

1. Comenzamos con la lista completa. El elemento pivote será el 4:

5 - 3 - 7 - 6 - 2 - 1 - 4
p

2. Comparamos con el 5 por la izquierda y el 1 por la derecha.

5 - 3 - 7 - 6 - 2 - 1 - 4
i j p

3. 5 es mayor que 4 y 1 es menor. Intercambiamos:

1 - 3 - 7 - 6 - 2 - 5 - 4

i j p

4. Avanzamos por la izquierda y la derecha:

1 - 3 - 7 - 6 - 2 - 5 - 4
i j p

5. 3 es menor que 4: avanzamos por la izquierda. 2 es menor que 4: nos mantenemos ahí.

1 - 3 - 7 - 6 - 2 - 5 - 4
i j p

6. 7 es mayor que 4 y 2 es menor: intercambiamos.

1 - 3 - 2 - 6 - 7 - 5 - 4
i j p

7. Avanzamos por ambos lados:

1 - 3 - 2 - 6 - 7 - 5 - 4
i j p

8. En este momento termina el ciclo principal, porque los índices se cruzaron. Ahora intercambiamos lista[i] con lista[p] (pasos 16-18):

1 - 3 - 2 - 4 - 7 - 5 - 6

p

9. Aplicamos recursivamente a la sublista de la izquierda (índices 0 - 2). Tenemos lo siguiente:

1 - 3 - 2

10. 1 es menor que 2: avanzamos por la izquierda. 3 es mayor: avanzamos por la derecha. Como se intercambiaron los índices termina el ciclo. Se intercambia lista[i] con lista[p]:

1 - 2 - 3

11. El mismo procedimiento se aplicará a la otra sublista. Al finalizar y unir todas las sublistas queda la lista inicial ordenada en forma ascendente.

1 - 2 - 3 - 4 - 5 - 6 - 7

e. Por Mezcla o MergeSort

El algoritmo de ordenamiento por mezcla (merge sort en inglés) es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Es de complejidad $O(n \log n)$. Fue desarrollado en 1945 por John Von Neumann. Conceptualmente, el ordenamiento por mezcla funciona de la siguiente manera:

- Si la longitud de la lista es 0 o 1, entonces ya está ordenada.

En otro caso:

- Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño, luego seguir dividiendo hasta obtener vectores unitarios.
- Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.
- Mezclar todas las sublistas en una sola lista ordenada.



El ordenamiento por mezcla incorpora dos ideas principales para mejorar su tiempo de ejecución:

- Una lista pequeña necesitará menos pasos para ordenarse que una lista grande. Se necesitan menos pasos para construir una lista ordenada a partir de dos listas también ordenadas, que a partir de dos listas desordenadas. Por ejemplo, sólo será necesario entrelazar cada lista una vez que están ordenadas.

4) Diferencia entre los diferentes métodos.

Algoritmos de ordenamiento	QuickSort	MergeSort	Selección	Intercambio	Inserción
Características principales	Utiliza un pivote y ordena los elementos según él.	Divide el vector en subvectores unitarios para posteriormente unirlos de a partes y ordenados.	encuentra el menor de todos los elementos del vector y lo intercambia con el que está en la primera posición.	Recorre el arreglo intercambiando los elementos adyacentes que estén desordenados, se recorre el vector tantas veces hasta que ya no hayan cambios que realizar.	Toma uno por uno de los elementos y avanza hacia su posición con respecto a los anteriormente ordenados hasta recorrer todo el vector.
Ventajas	No requiere memoria adicional y es de rápida ejecución.	Método de ordenamiento estable mientras la función de mezcla sea implementada correctamente. Es eficiente para vectores grandes ya que lo divide y ataca el problema por partes.	Es fácil su implementación, realiza pocos intercambios y no requiere de memoria adicional.	Es eficaz, sencillo y el código que realiza el ordenamiento es reducido.	Es fácil su implementación y sus requerimientos con respecto a la memoria son mínimos.
Desventajas	Trabaja con recursividad y su implementación es complicada.	Su principal desventaja radica en que está definido recursivamente y su implementación no recursiva emplea una pila, por lo que requiere un espacio adicional de memoria para almacenarla.	Es lento y poco eficiente cuando se lo utiliza en listas medianas o grandes. Además, realiza numerosas comparaciones.	Consume bastante tiempo de computadora y requiere de muchas lecturas/escrituras en memoria	Es bastante lento y realiza numerosas comparaciones

Fuentes consultadas:

Cómo ANALIZAR tus ALGORITMOS (en Ingeniería Informática)  
<https://www.youtube.com/watch?v=IZgOEC0NIbw>

Libro Introducción a los algoritmos (Introduction to Algorithms) de [Thomas H. Cormen](#), [Charles E. Leiserson](#), [Ronald L. Rivest](#) y [Clifford Stein](#).

<https://jona83.wordpress.com/unidad-1/algoritmos-de-ordenamiento/>

<https://es.slideshare.net/geoXnet/ordenamiento-en-c>

https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento

<https://es.wikipedia.org/wiki/Quicksort>

https://es.wikipedia.org/wiki/Ordenamiento_por_mezcla

https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n

https://www.youtube.com/watch?v=HVa2_UtXkCI&t=361s

<https://www.youtube.com/watch?v=IYNyL0HuWSg&t=465s>

<https://www.youtube.com/watch?v=BMSqO2doDTQ&t=723s>

<https://www.youtube.com/watch?v=kOgzXagXpTg&t=261s>

<http://www.bufoLand.cl/tc/burbuja2.php>