COMPARATIVA DE ALGORITMOS DE ORDENACIÓN

BUBBLESORT Y QUICKSORT

Resumen del documento

Explicación y desglose de los conocidos algoritmos de Ordenación Bubble Sort y Quicksort, atendiendo a los puntos más importantes de cada uno y llevando a cabo posteriormente el análisis detallado de ambos.

Autores

- Iván García Campos (alu0100693737@ull.edu.es).
- Javier Alberto Martín (alu0100836400@ull.edu.es).
- Eduardo Escobar Alberto (alu0100825985@ull.edu.es).

Contexto del documento

CENTRO	Universidad de La Laguna
ASIGNATURA	Diseño y Análisis de Algoritmos
PROFESOR	Eduardo Manuel Segredo González
FECHA DE ENTREGA	Lunes, 20 de marzo de 2017

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

Un algoritmo de ordenamiento es aquel que pone elementos de una lista o vector en una relación de orden. Normalmente, esta relación de orden es numérica o lexicográfica.

El problema del ordenamiento ha atraído gran cantidad de investigación, probablemente debido a la complejidad de resolverlo eficientemente. Aunque muchos puedan considerarlo un problema resuelto, nuevos y útiles algoritmos de ordenamiento se siguen inventando hasta el día de hoy, como el ordenamiento de biblioteca publicado por primera vez en el 2004.

En este trabajo, se pretende el análisis de dos algoritmos de ordenamiento ampliamente conocidos, Bubble Sort y Quicksort.

BUBBLE SORT

Bubble Sort es un algoritmo de ordenamiento sencillo, enmarcado en la lista de cuadráticos. Es un método de ordenación por intercambio, es decir, se recorre sucesivamente la secuencia intercambiando pares de elementos consecutivos desordenados, y luego, se van comparando los pares consecutivos de elementos desde el final hacia el principio. Al proceso desde la comparación de los dos últimos hasta los dos primeros se le llama **pasada**.

La secuencia se suele representar verticalmente y se denomina método de la burbuja porque parece que hay un elemento que sube como una burbuja. Hay que destacar que es necesario revisar varias veces toda la lista hasta que no exista la posibilidad de realizar más intercambios, debido a que ya está ordenada.

Este algoritmo también es conocido como el método del intercambio directo, ya que solo usa comparaciones para operar con los elementos. Esto hace que, aunque el ordenamiento de la burbuja sea uno de los algoritmos más sencillos de implementar, presenta una eficiencia superior a muchos otros algoritmos de ordenación como el de inserción o el de ordenación rápida (Quicksort).

```
procedimiento BubbleSort(a0, a1, a2, ..., a(n - 1))
    para i <- 1 hasta n hacer
        para j <- 0 hasta n - i hacer
            si a(j) > a(j + 1) entonces
                aux      <- a(j)
                a(j)     <- a(j + 1)
                a(j + 1) <- aux
            fin si
        fin para
    fin para
fin procedimiento
```

Ilustración 2: Pseudocódigo del algoritmo Bubble Sort

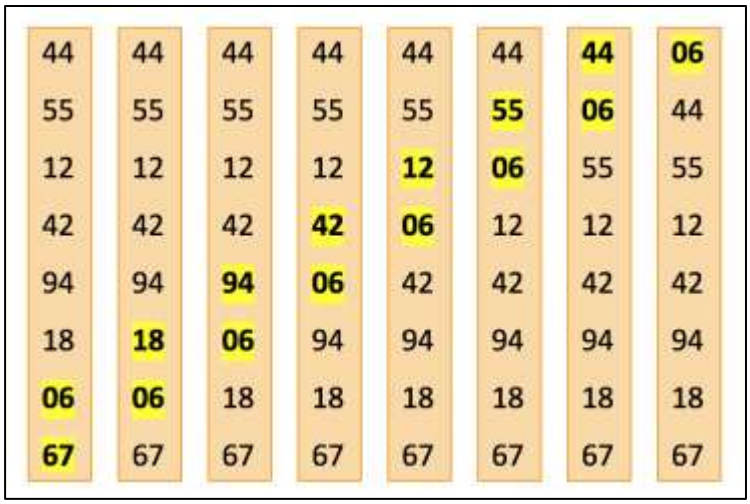


Ilustración 1: Ejemplo de pasada, en el que el elemento burbuja es el 6.

Como puede observarse en el pseudocódigo, el algoritmo de la burbuja siempre tiene el mismo número de comparaciones:

$$c(n) = \frac{n^2 - n}{2}$$

ESTRATEGIA DIVIDE Y VENCERÁS

El término divide y vencerás hace referencia a la técnica algorítmica basada en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que estos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente.

Podemos observar que esta estrategia está basada en tres pasos diferenciados:

1. Dividir el problema en varios sub-problemas del mismo tipo, pero más pequeños.
2. Resolver los sub-problemas de manera recursiva. Si el sub-problema es suficientemente pequeño, proporcionar la solución trivial.
3. Combinar las soluciones de los sub-problemas para obtener la solución del problema original.

```
procedimiento DivideVenceras (p: problema)
    Dividir (p, p1, p2, ..., pk)
    para i := 1, 2, 3, ..., k
        si := Resolver (pi)
    solucion := Combinar (s1, s2, ..., sk)
fin procedimiento
```

Ilustración 3: Ejemplo de pseudocódigo generalizado para la técnica divide y vencerás.

QUICKSORT

El algoritmo de ordenamiento rápido (Quicksort), basado en la **técnica divide y vencerás**, fue desarrollado por Charles Antony Richard Hoare en 1960. Es la técnica de ordenamiento más rápida conocida. El algoritmo original es **recursivo**, aunque existen versiones mejoradas que utilizan elementos iterativos para consumir menos recursos.

El algoritmo trabaja de la siguiente forma:

1. Elegir un elemento de la lista de elementos a ordenar, al que llamaremos **pivote**.
2. Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
3. La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
4. Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido. Más adelante se estudiará la complejidad, pero cabe destacar inicialmente, que se trata de un algoritmo logarítmico, por lo que, en **el caso promedio, el orden es $O\ (nlog\ n)$.** No es extraño, pues, que la mayoría de optimizaciones que se aplican al algoritmo se centren en la elección del **pivote**.

```
procedimiento Quicksort(sec, ini, fin)
    i = ini
    f = fin
    p = sec[(i + f) / 2]
    mientras (i < f)
        mientras (sec[i] < p] i++
        mientras (sec[f] > p) f--
    fin mientras
    si (i < f)
        x      = sec[i]
        sec[i] = sec[f]
        sec[f] = x
        i++
        f--
    fin si
    si (ini < f) Quicksort(sec, ini, f)
    si (i < fin) Qsort(sec, i, fin)
fin procedimiento
```

Ilustración 5: Pseudocódigo del algoritmo Quicksort

A continuación, podemos ver una parte de la traza del algoritmo, en el que se detalla una de las **pasadas** de la recursividad, observando que se establece inicialmente un elemento pivote sobre el que ordenar el resto, quedando al final dos subsecuencias.

ANÁLISIS DE LA COMPLEJIDAD

El análisis de la complejidad computacional determina la cantidad de recursos requeridos por un algoritmo en particular para resolver un problema.

BubbleSort:

La implementación del algoritmo BubbleSort consta de dos bucles anidados en términos de n. Se realizan siempre el mismo número de comparaciones:

$$c(n) = \frac{n^2 - n}{2}$$

La complejidad computacional es de orden $O(n^2)$ y puede verse fácilmente en el pseudocódigo.

QuickSort:

El algoritmo QuickSort consta de un bucle en términos de n y una recursividad basada en la técnica divide y vencerás que usa 2 llamadas recursivas de tamaño n/2.

Partiendo de la función de recurrencia para el algoritmo QuickSort:

$$T(n) \leq 2T(n/2) + O(n)$$

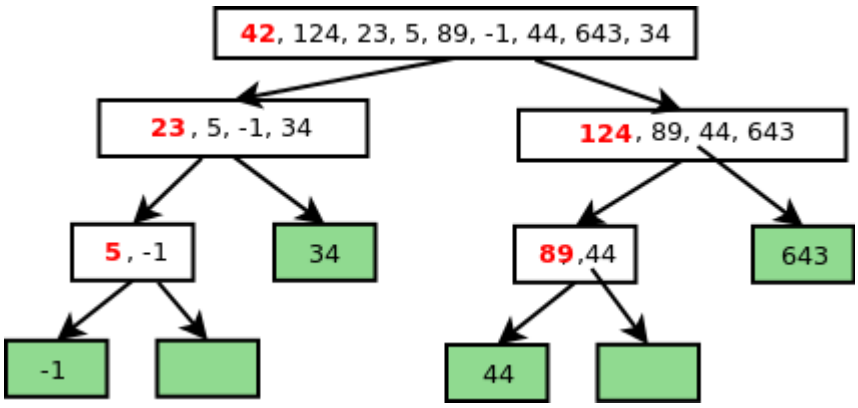
Aplicamos el Teorema maestro para poder saber la complejidad asintótica:

$$T(n) = \begin{cases} O(n^d log(n)) & si\ a = b^d \\ O(n^d) & si\ a < b^d \\ O(n^{log_b(a)}) & si\ a > b^d \end{cases}$$

Estamos ante el primer caso: $a = b^d$ por lo que:

$$O(n) = n \log_2(n)$$

Un posible ejemplo de árbol de recurrencia para el algoritmo QuickSort es:



BUBBLE SORT VS. QUICKSORT

Si bien **Bubble Sort** es uno de los métodos de ordenamiento más sencillos de implementar, **está considerado como ineficiente**, ya que para que una lista esté completamente ordenada es necesario repetir el procedimiento más de una vez, y en el peor de los casos, cuando todos los elementos están en desorden, este método tendrá una complejidad de $O(n^2)$. Por otro lado, **el método Quicksort es considerado el más eficiente y rápido método de ordenamiento**. pues en el mejor de los casos, cuando el pivote se encuentre en medio de la lista a ordenar, este método tendrá una complejidad de O(nlog n) y en el peor, cuando el pivote se encuentre en uno de los extremos de la lista su complejidad será de $O(n^2)$.

En conclusión, se podría decir que Quick Sort **nunca será menos eficiente que Bubble Sort**, pues la mayoría de los ordenamientos se realizan con listas casi ordenadas, pero aun así sigue dependiendo de la posición del pivote.

En una lista ordenada, Bubble Sort, pese a no realizar ningún intercambio, igualmente **compara todos los elementos del conjunto**, por lo que su complejidad sigue siendo cuadrática. Existen implementaciones mejoradas de Bubble Sort, en que el algoritmo recuerda el último lugar donde fue hecho algún intercambio, realizando menos comparaciones.

Para conjuntos de gran tamaño, la diferencia en el rendimiento de ambos algoritmos puede ser enorme, con una clara ventaja para Quicksort. Para listas pequeñas, la diferencia es menor. Sin embargo, para una implementación de Quicksort en que el pivote sea elegido efectivamente al azar, Quicksort sigue siendo mejor que Bubble Sort.

De todas formas, para arreglos pequeños existen algoritmos con mejor desempeño, como es el caso de Insertion Sort, por lo que Quicksort generalmente es implementado de tal forma que cuando queden arreglos pequeños a ordenar, estos últimos sean ordenados mediante Insertion Sort, esto ayuda también a disminuir la cantidad de llamadas recursivas que realiza Quicksort.

Destacar finalmente, que de los algoritmos de ordenamiento de orden cuadrático (Bubble Sort, Insertion Sort, Selection Sort), Bubble Sort es quien tiene el peor desempeño.

CÓDIGO ELABORADO

Para dar soporte a todo lo detallado en este informe, se ha desarrollado el código funcional de cada uno de los algoritmos. En el enlace que se deja a continuación, nos encontramos un repositorio GitHub, en el que podemos acceder a él.

https://github.com/alu0100693737/DAA_LI_4_Comparativa-de-algoritmos-de-ordenacion--Bubble-Sort-y-Quicksort

Además, para el entendimiento del código desarrollado, se aporta la documentación necesaria.

https://alu0100693737.github.io/DAA_LI_4_Comparativa-de-algoritmos-de-ordenacion--Bubble-Sort-y-Quicksort/source/html/

El programa ordena un vector del tamaño deseado mediante la técnica Quicksort y Bubble Sort.

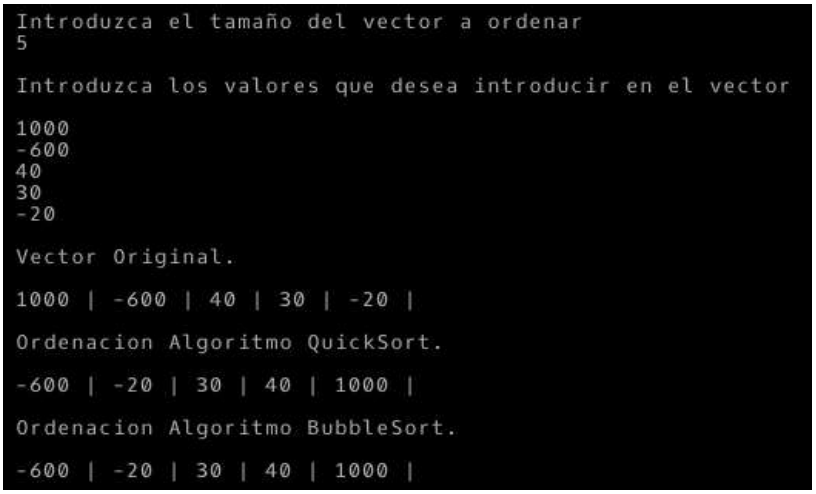
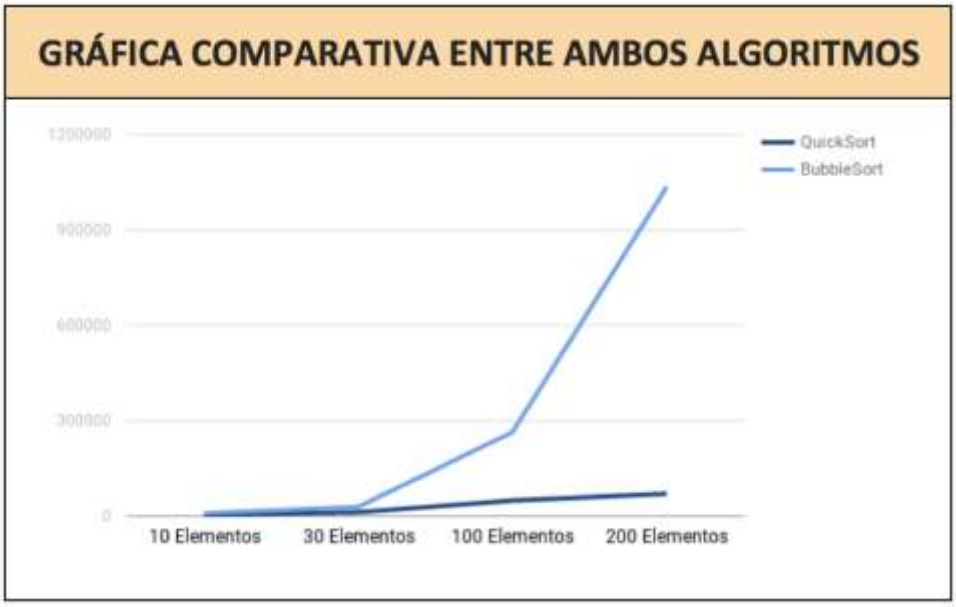


Ilustración 6: Ejemplo de ejecución del programa desarrollado

EJECUCIONES EXPERIMENTALES

BUBBLE SORT			
10 ELEMENTOS	**30 ELEMENTOS**	**100 ELEMENTOS**	**200 ELEMENTOS**
13507 ns	42346 ns	274230 ns	1034653 ns
7960 ns	29604 ns	234730 ns	1047150 ns
8745 ns	30892 ns	306412 ns	1030688 ns
8425 ns	14674 ns	254151 ns	1029692 ns
8542 ns	30211 ns	258133 ns	1032093 ns
8922 ns	29089 ns	262169 ns	1043884 ns

QUICKSORT			
10 ELEMENTOS	**30 ELEMENTOS**	**100 ELEMENTOS**	**200 ELEMENTOS**
4697 ns	*14926 ns*	*52304 ns*	*78969 ns*
3885 ns	*12867 ns*	*47378 ns*	*67495 ns*
3684 ns	*12243 ns*	*50945 ns*	*68986 ns*
3731 ns	*11647 ns*	*47984 ns*	*68874 ns*
3583 ns	*11745 ns*	*47698 ns*	*69355 ns*
3765 ns	*12453 ns*	*49766 ns*	*68497 ns*



CONCLUSIONES

1. Queda demostrada la complejidad de los algoritmos de ordenación Bubble Sort y Quicksort.

2. El algoritmo Bubble Sort tiene una complejidad computacional del orden **$O(n^2)$** siendo un algoritmo bastante ineficiente, es decir, es demasiado lento, ya que realiza numerosas comparaciones y sobretodo numerosos intercambios. Sin embargo, es un algoritmo bastante sencillo de implementar y no requiere memoria adicional.

3. En el caso del ordenamiento rápido Quicksort, tiene una complejidad de **O(nlog n)** siendo un algoritmo eficiente. Sin embargo, este algoritmo utiliza la técnica divide y vencerás como método para conseguir esta complejidad teniendo una implementación algo más compleja.

REFERENCIAS

- Wikipedia. **Cota superior asintótica**.

https://es.wikipedia.org/wiki/Cota_superior_asint%C3%B3tica

- Wikipedia. **Library Sort.**

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

- Wikipedia. **Quicksort.**

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

- Campus Virtual ULL. Apuntes de la asignatura.
- Campus Virtual ULL. Apuntes AEDA.
- Mis Algoritmos. Ordenamiento Rápido Quicksort.

http://mis-algoritmos.com/ordenamiento-rapido-quicksort