Algoritmos divide y vencerás
Quicksort y Mergesort

Patricia Aguado Labrador

October 2018

1 Introducción

Divide y Vencerás es uno de los paradigmas más importantes en el diseño de algoriumos. El método está basado en la resolución recursiva de un problema, dividiéndolo en dos o más subproblemas más pequeños del mismo tipo o similar y combinando las soluciones obtenidas para obtener la solución del problema original.

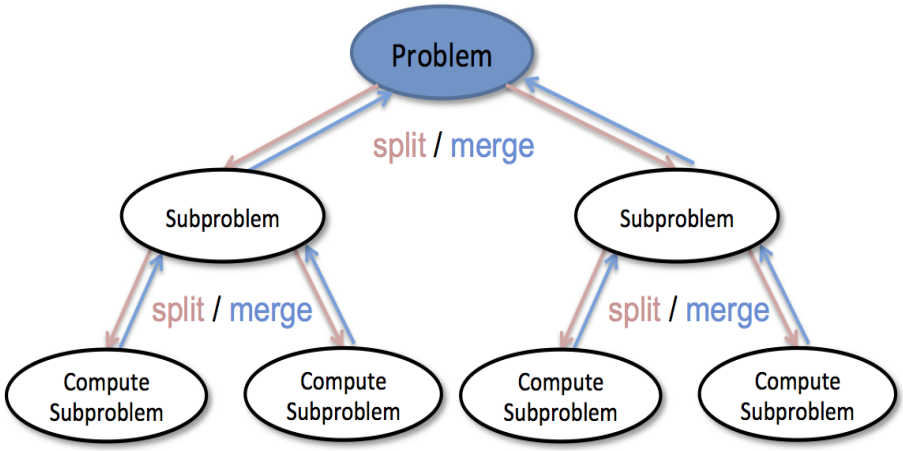
Esta técnica es la base de algoritmos como Karatsuba (multiplicación de números grandes), análisis sintácticos, transformada discreta de Fourier o algoritmos de ordenación en los que se centra este trabajo.

Para que el enfoque divide y vencerás merezca la pena son necesarias tres condiciones:

1. Se debe elegir con cuidado entre utilizar un subalgoritmo básico o realizar llamadas recursivas.
2. Tiene que ser posible descomponer en subcasos y recomponer las soluciones parciales de forma eficiente.
3. Los subcasos deben ser dentro de lo posible del mismo tamaño.

Desde el punto de vista de la eficiencia es muy importante que la división de los subproblemas sea lo más equilibrada posible, ya que conviene que la solución de cada uno de los subproblema tengan un coste similar. Por otro lado, también lo será conseguir que los subproblemas sean independientes sin que haya solapamiento entre ellos, ya que si existe una dependencia entre las soluciones el tiempo de ejecución será exponencial.

Estos algoritmos están naturalmente implementados como procesos recursivos, lo que provoca la herencia de las ventajas e inconvenientes de la recursividad. Por ejemplo, por un lado tendremos un diseño simple que nos permitirá una mayor facilidad de depuración y mantenimiento, y por contra tendremos un mayor tiempo de ejecución que con un diseño iterativo además de la complejidad espacial.



2 Algoritmos de ordenación

Son algoritmos que ponen elementos de una lista o un vector en secuencias dadas por una relación de orden, es decir, la salida ha de ser una permutación de la entrada que satisfaga la relación de orden dada.

2.1 Mergesort

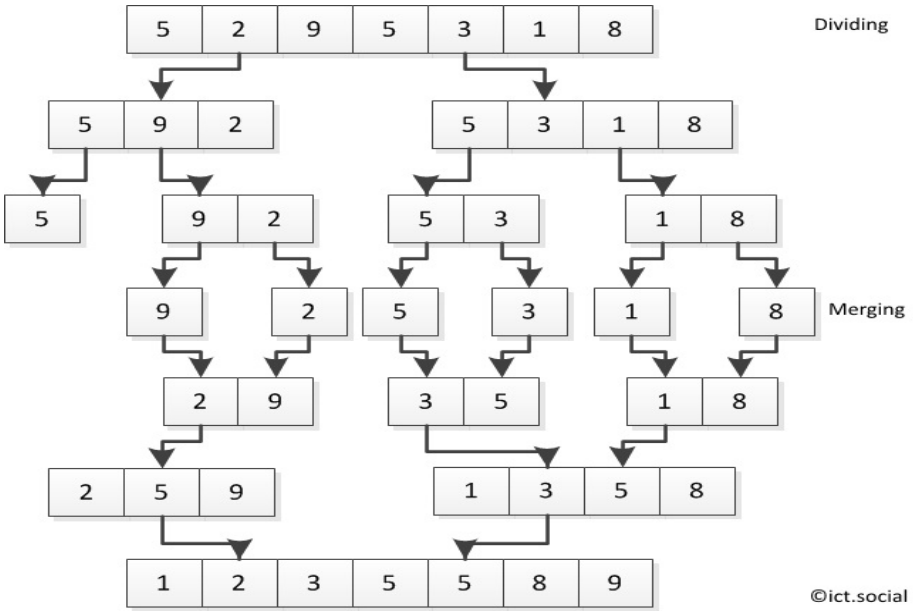
Es un algoritmo de ordenamiento por mezcla desarrollado en 1945 por John Von Neumann.
Características:

- Es un método universal, es decir, está basado en comparaciones.
- Fácilmente adaptable a acceso secuencial, pudiéndose aplicar a otras estructuras de datos.
- Estable, es decir, mantiene el orden original de los elementos con claves iguales.
- No adaptativo

El algoritmo en primer lugar comprueba el número de elementos de la lista, en el caso de estar formada por uno o ser lista vacía (caso base) ya está ordenada. En otro caso se dividirá la lista desordenada en dos sublistas de la mitad del tamaño, estas se ordenarán recursivamente aplicando mergesort y por último se mezclarán ambas sublistas en una sola.

En mi implementación la función mergesort es la que realiza la partición de la lista en dos sublistas de la mitad del tamaño, y es aplicada recursivamente a estas. Por otro lado, está la función merge que toma dos listas de las que

compara sus elementos y devuelve una lista auxiliar en la que están ordenados. Esta lista auxiliar es una de las desventajas del algoritmo.



2.2 Quicksort

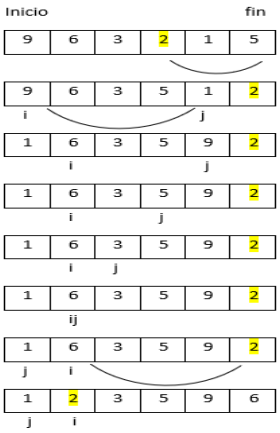
Es un algoritmo de ordenación rápida creado por Tony Hoare.
Características:

- Es un método universal, es decir, está basado en comparaciones.
- No estable,es decir, altera el orden original de los elementos con claves iguales, pudiendo provocar una pérdida de eficiencia.
- No adaptativo
- Trabaja sobre el propio vector

El algoritmo en primer lugar elige un elemento de la lista a ordenar, al que denominaremos pivote. Resituará los demás elementos a cada lado del pivote, dejando a un lado los menores y al otro los mayores. La lista queda dividida en dos sublistas ,una con los elementos de la derecha y otra con los de la izquierda del pivote (este ya ocupa su lugar en la lista ordenada). Se repite el proceso con las sublistas de forma recursiva hasta tener listas vacías o de un elemento (caso base).

La eficiencia de este algoritmo depende fundamentalmente de la elección del elemento que será denominado con el nombre de pivote, ya que este determina las sublistas. Hay determinadas formas de elegirlo, por ejemplo, elegir el primer

o último elemento de la lista como pivote no es una buena elección, ya que si la lista está medianamente ordenada, las sublistas tendrán tamaños muy diferentes. Por está razon, en mi implementación he decidido que sea el elemento que ocupa el lugar central de la lista (división entera del tamaño de la lista entre dos).



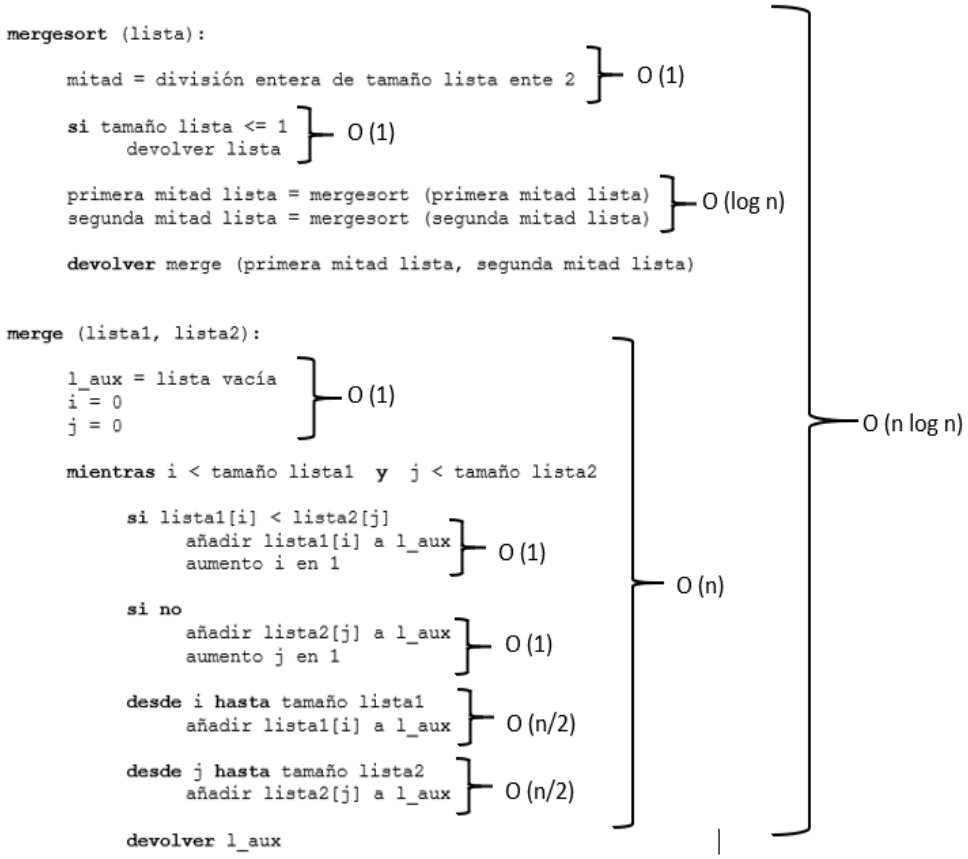
2.3 Comparación

En quicksort lo más costoso es la división en vectores de menor tamaño, ya que una vez ordenados estos la combinación es inmediata. Sin embargo, en mergesort la división en subvectores de menor tamaño consiste simplemente en tomar la mitad de los elementos, mientras que el proceso de mezcla de los subvectores con su debida ordenación es lo que supone el mayor coste del algoritmo.
Por otro lado, una de las desventajas de mergesort es que trabaja sobre un array auxiliar lo cual implica un uso de memoria y de trabajo extra consumido en las copias entre sublistas. Esto en quicksort no pasa ya que el ordenamiento se realiza sobre el propio vector.
El algoritmo quicksort es menos estable que el mergesort, ya que en el caso promedio es de los algoritmos más rápidos, pero su rendimiento puede resultar bastante malo para una lista desordenada concreta, pudiendo no comportarse mucho mejor que los algoritmos simples de ordenación.

ALGORITMO	TIEMPO	ESPACIO	ESTABLE
QUICKSORT	$O(n^2)$ [peor] O(n log n) [promedio]	O(n) [peor] O(log n) [promedio]	NO
MERGESORT	O(n log n)	O(n)	SI

3 Pseudocódigo

3.1 Mergesort



3.2 Quicksort

4 Análisis de coste

4.1 Mergesort

El algoritmo consiste en dividir el vector que queremos ordenar en dos partes, ordenar estas utilizando de nuevo el algoritmo y una vez ordenadas fusionarlas. Como caso base de la recursividad tenemos una un vector formado por un solo elemento. El paso de dividir tarda un tiempo constante independientemente del tamaño de la entrada, lo que sería O(1). El paso de mezclar las listas ordenadas será de O(n), las dos primeras llamadas recursivas tardarán una constante multiplicada por n/2, y asi sucesivamente dividiendo entre 2 el tiempo de la mezcla y multiplicando por 2 el número de subvectores en cada nivel de la recursión, hasta llegar a los casos base de la recursión (tendremos n al empezar con un vector de tamaño n) y tener un tiempo de costante por n, que será lo que tarden en conjunto.

Conclusión para cada tamaño de subproblema el algoritmo tardará un tiempo lineal O(n), entonces el tiempo total de mergesort será este tiempo multiplicado

por los niveles de recursividad que al tratarse de un arbol binario sera logaritmo en base dos de la cantidad de elementos más uno. De esta manera tenemos un tiempo total de n(log n +1) para el peor caso y el caso promedio, del cual podemos descartar el +1, quedandonos con un tiempo de ejecución de O(nlog n).
En cuanto al coste espacial tendremos un O(n), debido al vector auxiliar que necesitamos.

4.2 Quicksort

El peor caso del algoritmo quicksort en mi implementación será tener que ordenar una lista cuyos elementos sean crecienes hasta la mitad y decrecientes de la mitad hasta el final (por ejemplo: [5,6,7,8,4,3,2]) ya que al realizar la división en sublistas me van a quedar lo más desbalanceadas posibles al quedarme el pivote en un extremo de la lista como posición final. El tiempo empleado por el algoritmo para realizar la partición en la llamada inicial será de tiempo lineal n y para las subsiguientes llamadas recursivas sobre n-1 elementos será n-1 y así sucesivamente ya que la otra sublista estará vacía. De esta forma tenemos que la suma del tiempo que conlleva hacer las particiones en cada nivel de la recursión es de (n + (n-1)+...+2), que nos da una suma aritmética (de 1 a n, por eso le restamos 1) y lo que tenemos es que ((n+1)(n/2)-1) es el tiempo que tarda en total, que operando es (nˆ 2/2)+(n/2)-1 y que acotando por O grande tenemos un tiempo de O(nˆ 2) para el peor caso.

En el mejor de los casos la posición final del pivote me quedará en la mitad de la lista, permitiendo realizar la división en sublistas de la mitad del tamaño o de la mitad menos uno. En cualquiera de estos casos tendré como mucho n/2 elementos en las particiones y el árbol de tamaños y particiones que generará se parecerá al del algoritmo mergesort. Por este motivo tendremos un tiempo acotado por O(n log n).

En el caso promedio, el vector no quedará dividido en partes iguales a ambos lados, sino que cada sublista tendrá por tamaño una proporción del tamaño de la lista. De este modo tendremos una relación de recurrencia T(n,m)= T(m,.)+T(n-m-1,.)+f(n,m), donde m es el número de elementos menores que el pivote y f(n,m) son las operaciones no recursivas de la partición (n-1 comparaciones, m+2 intercambios), operando tenemos que T(n)= nlog n +O(n), siendo acotado por O(n log n).

References

[1] G.Brassard and P. Bratley, *Fundamentos de algoritmia*, Prentice Hall, Madrid, 1998.

[2] César Vaca Rodríguez, *Estructuras de datos y algoritmos. Tema 1, análisis de algoritmos*

[3] Jesús Bisbal Riera, *Manual de Algorítmica: Recursividad, complejidad y diseño de algoritmos*, Editorial UOC, Madrid, 2009.

[4] https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento.
(1)