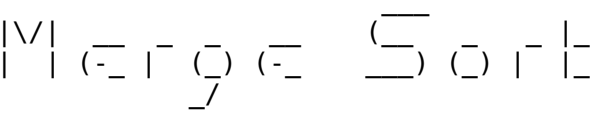
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬▬ 1. Introducción.
▬▬▬▬ 2. El Merge Sort.
▬▬▬▬ 3. Procedimiento de Merge.
▬▬▬▬ 4. Ejemplo del Merge Sort.
▬▬▬▬ 5. Otro Ejemplo del Merge Sort.
▬▬▬▬ 6. Análisis de la Eficiencia.
▬▬▬▬ 7. Usando el Teorema Maestro.
▬▬▬▬ 8. Otra Solución Para la Recurrencia.
▬▬▬▬ 9. Resumen.
▬▬▬▬ 10. Ejercicios.

▬▬▬▬ 1. Introducción.

En este capítulo veremos la técnica de “divide and conquer”, dividir y conquistar. Esta es la técnica muy utilizada en la solución de problemas.

En este capítulo se resolverá el problema del “merge sort”. Una vez implementado hablaremos sobre su tiempo de ejecución.

Por último es necesario mencionar que algunos algoritmos parecen naturalmente diseñados para programarse por medio de iteración, otros, parecen que son más fáciles de implementar por medio de recursión.

▬▬▬▬ 2. El Merge Sort.

El “merge sort” divide la lista de entrada en dos partes iguales. Realiza una llamada recursiva con cada una de estas partes y luego las vuelve a unir de manera que se mantenga el orden.

▬▬▬ 3. Procedimiento de Merge.

El procedimiento de “merge” es clave en el proceso, una vez que se tienen dos listas ordenadas se debe unir correctamente. A continuación se muestra su proceso.

merge([10,15,40] , [20, 25, 35, 60])

Se construye una nueva lista, conforme se va revisando el primer elemento de cada lista:

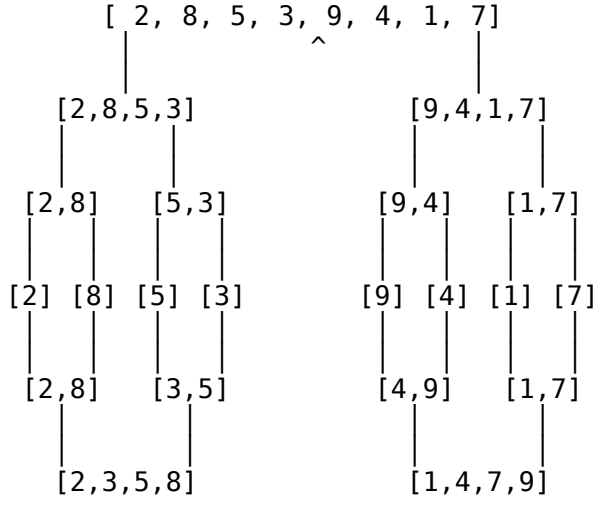
Listas del merge:		resultado:
[10,15,40]	[20, 25, 35, 60]	[]
[15,40]	[20, 25, 35, 60]	[10]
[40]	[20, 25, 35, 60]	[10, 15]
[40]	[25, 35, 60]	[10, 15, 20]
[40]	[35, 60]	[10, 15, 20, 25]
[40]	[60]	[10, 15, 20, 25, 35]
[]	[60]	[10, 15, 20, 25, 35, 40]
[]	[]	[10, 15, 20, 25, 35, 40, 60]

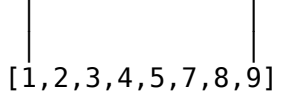
▬▬▬ 4. Ejemplo del Merge Sort.

Supongamos que se utiliza la siguiente lista de entrada:

lista: [2, 8, 5, 3, 9, 4, 1, 7];

Veamos cómo funciona el algoritmo.



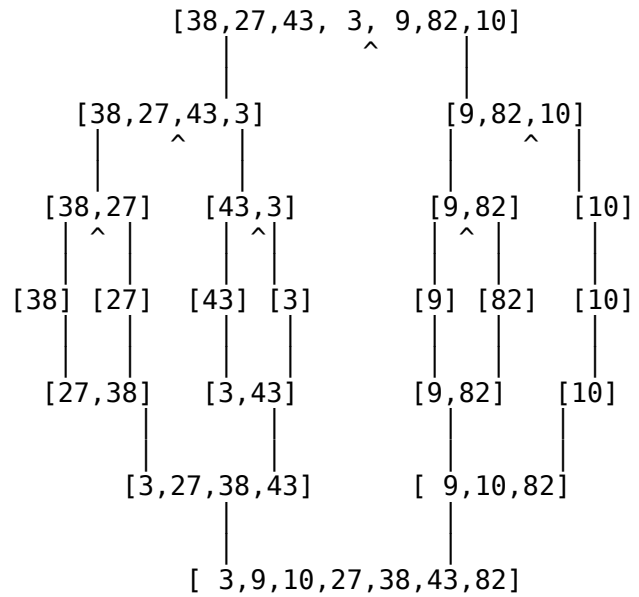


▬▬▬ 5. Otro Ejemplo del Merge Sort.

Supongamos que se utiliza la siguiente lista de entrada, veamos como funciona el algoritmo.

lista: [38, 27, 43, 3, 9, 82, 10];

A continuación se muestran todas las llamadas que se producen para llegar al resultado final:



▬▬▬ 6. Análisis de la Eficiencia.

●● La función de Merge.

La función de “merge” debe, en el peor de los casos recorrer toda la lista. Por lo tanto su tiempo es de O(n).

En general:
si la lista1 es de tamaño n1.
y la lista2 es de tamaño n2.

sea n = n1 + n2

Se deben recorrer todos los elementos lo que produce un tiempo:
O(n1 + n2)
O(n)

●● Partir la lista en dos.

Supondremos que para partir la lista en dos partes iguales se requiere algún tipo de tiempo lineal. de la forma de O(n)

●● El Merge Sort.

El “merge sort” tendrá entonces un tiempo de:

$T(1) = 1$

$T(n) = 2*T(n/2) + O(n)$

Al resolver la relación de recurrencia se obtiene:

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

▬▬▬ 7. Usando el Teorema Maestro.

Tenemos que el tiempo de ejecución está dado por:

$T(1) = 1$

$T(n) = 2*T(n/2) + O(n)$

Que puede describirse como

$T(n) = a*T(n/b) + f(n)$

$T(n) = 2*T(n/2) + n$

Tenemos:
a=2
b=2
d=1

Pues
f(n) = O(n) = O(n^1)

Tenemos:

a = 2

b^d = 2^1 = 2

Como a=b^d entonces:

O(n^d * $\log_2 n$)

= O(n^1 * $\log_2(n)$)

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

▬▬▬ 8. Otra Solución Para la Recurrencia.

En Wolfram tenemos:

T[n]== 2*T[n/2] + n, T[1]==1

Y se obtiene el resultado:

O(n * $\log_2(n)$) cuando $n \to \infty$

▬▬▬ 9. Resumen.

• El método de dividir y conquistar trata de reducir un problema en problemas más pequeños. Aquí lo hemos usado para resolver el problema de ordenamiento mediante el merge sort.

• Más adelante se estudiará el algoritmo de quick sort, que realiza un proceso muy simular al merge sort. Será útil analizar los parecidos y diferencias de ambos métodos.

▬▬▬ 10. Ejercicios.

●● Ejercicio 1.

Busque una implementación del “merge sort” de forma iterativa, sin recursión.
El proceso puede usar un stack para su implementación.
Cuál algoritmo le parece más sencillo?

●● Ejercicio 2.

Muestre cada uno de los pasos que realiza el algoritmo de “merge sort”, cuando se llama la siguiente función:

(%i) mergeSort([7,3,9,4,2,5,6,1,8]).

●● Ejercicio 3.

Muestre cada uno de los pasos que realiza el algoritmo de “merge sort”, cuando se llama la siguiente función:

```
(%i) mergeSort([3, 5, 2, 9, 8, 1, 6, 4, 7]).
```

●● Ejercicio 4.

Construya una lista con valores del 1 al 8, sin repeticiones.
Esta lista debe provocar que el "merge sort" realice su peor caso
al hacer 17 comparaciones.
(Sugerencia: Piense en el ordenamiento de la lista)

●● Ejercicio 5.

Dada una lista de la forma:

lista:[3,4,5,1,2,3,4,7,3,2,1]

Primero se buscan las partes que están naturalmente ordenadas:

[3,4,5,1,2,3,4,7,3,2,1]

[3,4,5] [1,2,3,4,7] [3] [2] [1]

Luego se combinan cada una de esta listas mediante un proceso de merge.

[3,4,5] + [1,2,3,4,7] [3] [2] [1]

[1,2,3,3,4,4,5,7] + [3] [2] [1]

[1,2,3,3,3,4,4,5,7] + [2] [1]

[1,2,2,3,3,3,4,4,5,7] + [1]

[1,1,2,2,3,3,3,4,4,5,7]

Este proceso de ordenamiento se llama “natural sort”.
Programe este procedimiento y encuentre el valor del peor caso.