MERGE SORT

Edson Victor Lipa Urbina

Estrategia

Conceptualmente, el ordenamiento por mezcla funciona de la siguiente manera:

1)Si la longitud de la lista es 0 ó 1, entonces ya está ordenada. En otro caso:

2)Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.

3)Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.

4)Mezclar las dos sublistas en una sola lista ordenada.

Optimizacion

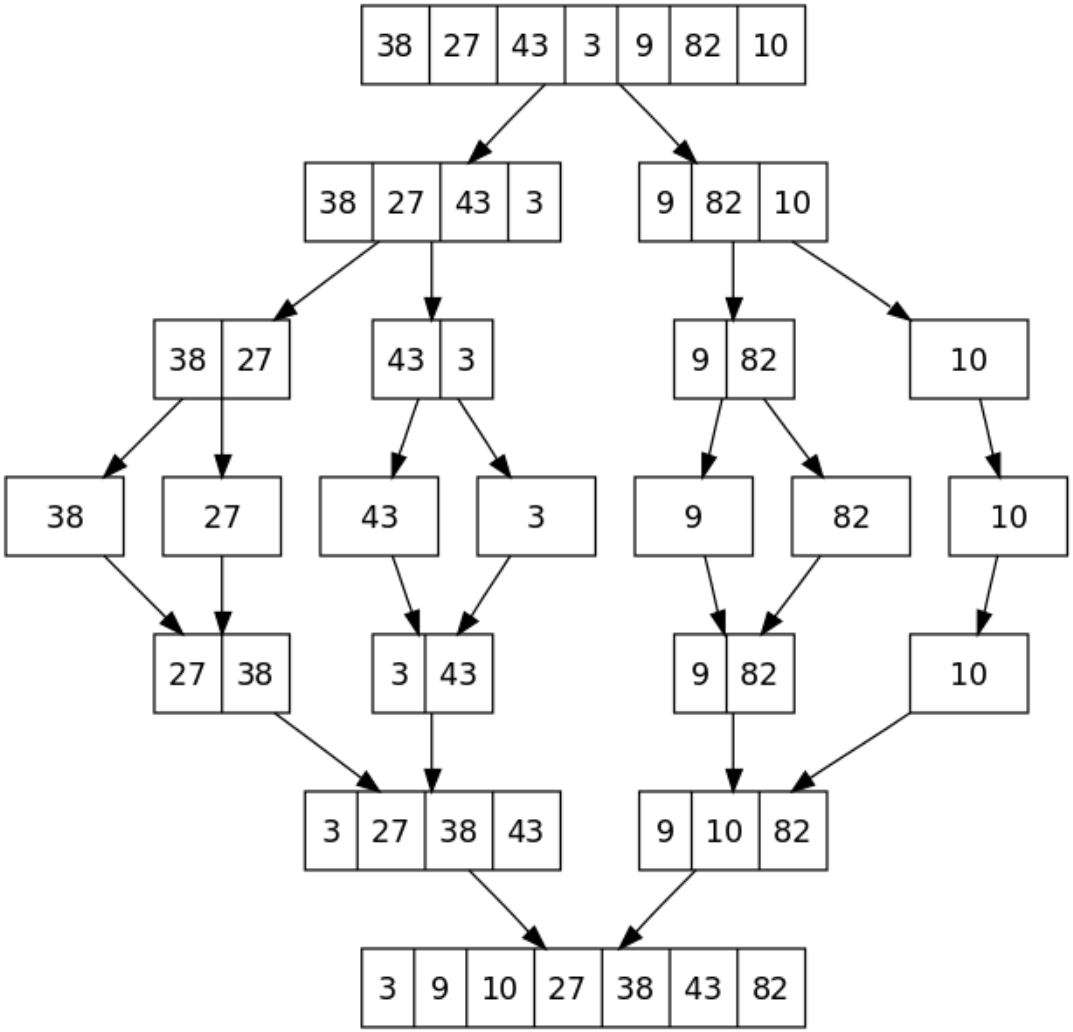
Por ejemplo, el algoritmo "tiled merge sort" deja de particionar subarrays cuando se han alcanzado subarrays de tamaño S, donde S es el número de elementos que caben en una única página en memoria.

Cada uno de esos subarrays se ordenan con un algoritmo de ordenación in-situ, para evitar intercambios en memoria

PSEUDOCODIGO

MergeSort(arr[], l, r)
If r > l

1. Encuentra el punto medio que divide al vector o al arreglo en 2:
   middle m = (l+r)/2
2. Llama mergeSort para la primera mitad:
   Call mergeSort(arr, l, m)
3. Llama mergeSort para la segunda mitad:
   Call mergeSort(arr, m+1, r)
4. llama a merge para unir y ordenar dos partes y asi ordenas los pasos 2 y 3 :
   Call merge(arr, l, m, r)


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Analisis

En resumen: Peor caso	O(*n* log *n*)
Mejor caso	O(*n* log *n*) typical, O(*n*) natural variant
Caso Promedio	O(*n* log *n*)
La complejidad del espacio en el peor de los casos	O(*n*) total, O(*n*) auxiliary

Comparaciones

- Aunque heapsort tiene los mismos límites de tiempo que merge sort, requiere sólo Θ(1) espacio auxiliar en lugar del Θ(n) de merge sort
- En el lado bueno, merge sort es un ordenamiento estable, paraleliza mejor, y es más eficiente manejando medios secuenciales de acceso lento(lista enlazada)
- El mal rendimiento de las listas enlazadas ante el acceso aleatorio hace que otros algoritmos (como quicksort) den un bajo rendimiento, y para otros (como heapsort) sea algo imposible

Analisis de tiempo