Actividad 5.2

QuickSort básico

Algoritmos y Estructuras de Datos

Tema 5: ordenación

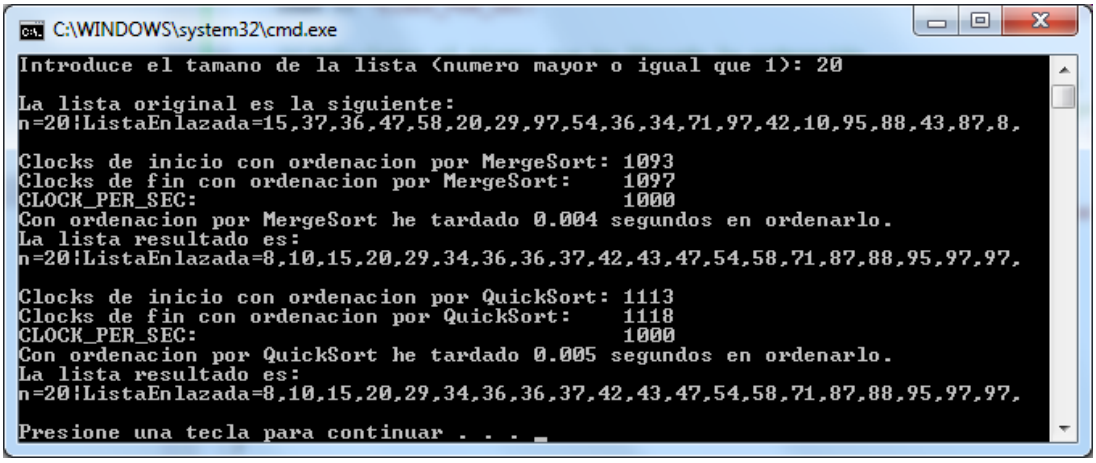
Amplía la actividad 5.1, añadiendo una primera versión sencilla que implemente la ordenación por QuickSort. Ahora tendremos dos métodos públicos para ordenar:

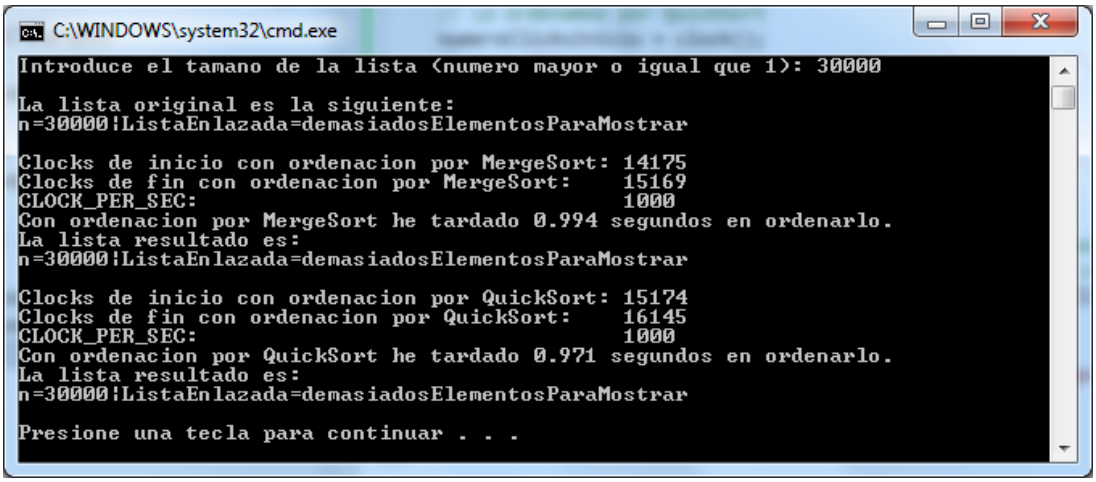
- ordenarPorMergeSort(), que será el método "ordenar()" que teníamos en 5.1, pero renombrado al nuevo nombre "ordenarPorMergeSort()". Llamará a su correspondiente método privado recursivo llamado "ordenarPorMergeSortRecursivo()".
- ordenarPorQuickSort(). Lo tenemos que hacer. Llamará a su correspondiente método privado recursivo llamado "ordenarPorQuickSortRecursivo()".

Ten en cuenta lo siguiente:

- Tienes que hacer que la elección del pivote sea O(1). En esta primera versión de QuickSort, elige como pivote el último elemento de la lista. Como es doblemente enlazada y circular, acceder al último elemento nos cuesta un tiempo de O(1).
- A la hora de hacer la partición, ten muy en cuenta que ninguna de las dos sublistas debe quedar vacía. De hecho, pon la precondición de que el tamaño de la lista sea mayor o igual que 1 en ordenarPorQuickSortRecursivo(). Para conseguirlo, vete repartiendo todos los elementos de la lista (empezando por el primero y avanzando hacia delante) y, cuando llegues al último (al pivote), mételo en la sublista que en ese momento tenga menos elementos. Así nos aseguramos de que, en el peor caso, al menos una de las sublistas tenga un elemento (el pivote) y no cero elementos.
- El reparto de los elementos entre las dos sublistas debe tener una complejidad espacial de O(1). Para conseguirlo, cada vez que insertes un elemento en una de las sublistas, tienes que eliminarlo de la lista original.
- Cuidado cuando juntes las dos sublistas al final del algoritmo. Este proceso, en esta primera versión, debe tardar un tiempo de O(n) como máximo. La complejidad espacial debe ser O(1), y, para ello, cada vez que pongamos un nodo en la lista resultado, lo quitaremos de la sublista origen.
- Ten cuidado cuando crees cada sublista en cada paso recursivo, para que no sea destruida antes de tiempo.

El main comparará el tiempo que tardamos en ordenar la misma lista enlazada de tamaño introducido por teclado y componentes aleatorios con nuestros MergeSort y QuickSort. ¿Qué conclusiones se pueden extraer?


C:\WINDOWS\system32\cmd.exe
Introduce el tamano de la lista (numero mayor o igual que 1): 20
La lista original es la siguiente:
n=20¦ListaEnlazada=15,37,36,47,58,20,29,97,54,36,34,71,97,42,10,95,88,43,87,8,
Clocks de inicio con ordenacion por MergeSort: 1093
Clocks de fin con ordenacion por MergeSort: 1097
CLOCK_PER_SEC: 1000
Con ordenacion por MergeSort he tardado 0.004 segundos en ordenarlo.
La lista resultado es:
n=20¦ListaEnlazada=8,10,15,20,29,34,36,36,37,42,43,47,54,58,71,87,88,95,97,97,
Clocks de inicio con ordenacion por QuickSort: 1113
Clocks de fin con ordenacion por QuickSort: 1118
CLOCK_PER_SEC: 1000
Con ordenacion por QuickSort he tardado 0.005 segundos en ordenarlo.
La lista resultado es:
n=20¦ListaEnlazada=8,10,15,20,29,34,36,36,37,42,43,47,54,58,71,87,88,95,97,97,
Presione una tecla para continuar . . .


C:\WINDOWS\system32\cmd.exe
Introduce el tamano de la lista (numero mayor o igual que 1): 30000
La lista original es la siguiente:
n=30000¦ListaEnlazada=demasiadosElementosParaMostrar
Clocks de inicio con ordenacion por MergeSort: 14175
Clocks de fin con ordenacion por MergeSort: 15169
CLOCK_PER_SEC: 1000
Con ordenacion por MergeSort he tardado 0.994 segundos en ordenarlo.
La lista resultado es:
n=30000¦ListaEnlazada=demasiadosElementosParaMostrar
Clocks de inicio con ordenacion por QuickSort: 15174
Clocks de fin con ordenacion por QuickSort: 16145
CLOCK_PER_SEC: 1000
Con ordenacion por QuickSort he tardado 0.971 segundos en ordenarlo.
La lista resultado es:
n=30000¦ListaEnlazada=demasiadosElementosParaMostrar
Presione una tecla para continuar . . .