Actividades adicionales

Algoritmos y Estructuras de Datos

Tema 5: ordenación

1º Grado en Ingeniería en Desarrollo de Contenidos Digitales
© Profesor Dr. Carlos Grima Izquierdo (www.carlosgrima.com)
U-tad (www.u-tad.com)

Por cada apartado evaluable de programación, el 50% de la nota será la funcionalidad, y el 50% restante será la buena programación. Como en las actividades realizadas en clase, sólo se valorará la buena programación si el apartado cumple completamente su funcionalidad. Si no compila o no se cumple totalmente la funcionalidad o da error en tiempo de ejecución, se evaluará con 0 el apartado en cuestión. También se pondrá 0 en el apartado en cuestión si éste viola memoria (dé o no dé error en tiempo de ejecución), si no es portable a otros compiladores o plataformas, o si no cumple con todos los requisitos del enunciado (número y tipo de parámetros, la complejidad que tiene que cumplir, dónde tiene que estar el método, etc.). Los distintos aspectos de la buena programación se evaluarán con distinta puntuación según su dificultad, a criterio del profesor.

Actividad adicional 5.1

Programa el método de ordenación de números enteros por Inserción para una lista contigua. Haz una versión para ordenar de menor a mayor, y otra para ordenar de mayor a menor. Cada una de las versiones no devolverá nada y recibirá dos parámetros: un puntero al comienzo del array y su número de componentes. Ponlo todo en una biblioteca llamada "ordenacion". Pruébalo con un main que pida el número de componentes, genere un array aleatorio de ese tamaño (cada componente en el rango [-23,+30]), lo muestre por pantalla, lo ordene de menor a mayor, muestre el resultado por pantalla (los números separados por espacio), ordene de mayor a menor el array original, y muestre el resultado por pantalla.

Actividad adicional 5.2

Repite la actividad anterior, pero con el método de ordenación por Burbuja. La burbuja se mueve de izquierda a derecha.

Actividad adicional 5.3

Repite la actividad anterior, pero ahora la burbuja se mueve de derecha a izquierda.

Actividad adicional 5.4

Programa el algoritmo MergeSort para listas contiguas. Haz que ordene de menor a mayor (como es habitual), pero también de mayor a menor. Para ello, añade lo siguiente a la clase ListaContigua (actividad 4.2):

Dos métodos públicos, sin parámetros y sin retorno, llamados "ordenarAscendentePorMergeSort()" y "ordenarDescendentePorMergeSort()". Cada uno de ellos ordenará de menor a mayor y de mayor a menor respectivamente.

Los anteriores métodos llamarán respectivamente a los métodos privados recursivos "ordenarAscendentePorMergeSortRecursivo()" y "ordenarDescedentePorMergeSortRecursivo()". No devolverán nada, y ambos recibirán un puntero a la lista que se quiere ordenar (un objeto de la clase ListaContigua).

Pruébalo con un main() similar al de la actividad adicional 5.1. Pero ahora, en vez de crear un array aleatorio, se creará un objeto de la clase ListaContigua (actividad 4.2) con componentes aleatorios.

Actividad adicional 5.5

Repite la actividad anterior, pero ahora los algoritmos estarán fuera de toda clase (no existe en este programa la clase ListaContigua, ni tampoco hay que crearla). Por lo tanto, ordenarán directamente un array de números enteros. Los algoritmos estarán en una biblioteca llamada "ordenación". Ten en cuenta que la partición deberá hacerse en O(1) tanto en tiempo como en espacio. La combinación será O(n) en tiempo y espacio. Pruébalo con un main() similar al de la actividad adicional 5.1.

Actividad adicional 5.6

Repite la actividad anterior, pero ahora con el algoritmo QuickSort. La obtención del pivote adecuado debe tener un tiempo de O(1), teniendo en cuenta que es muy probable que la lista a ordenar ya esté ordenada o casi ordenada. El reparto según el pivote debe tener un espacio adicional de O(1). La combinación debe ser O(1) tanto en tiempo como en memoria adicional.

Actividad adicional 5.7

Programa el algoritmo de "ordenación por rango" para una lista enlazada, cuyos componentes son enteros entre -28 y +11. Para ello, usa la clase ListaEnlazadaSimple (actividad 4.3) y añádela el método público que ordene según este algoritmo de ordenación. El método no recibirá y devolverá nada. El método tiene que tener un tiempo O(n) y un espacio O(1). Pruébalo con un main en donde se pida el tamaño de la lista por teclado, genere una ListaEnlazadaSimple de ese tamaño (cada componente estará comprendido entre +11 y -28), la imprima, la ordene, e imprima el resultado.