Actividad 2.2

Comparación a posteriori

Algoritmos y Estructuras de Datos

Tema 2: algoritmos no recursivos

Vamos a comparar “a posteriori” el tiempo que se tarda en ordenar un vector con un mal algoritmo y con un buen algoritmo, independientemente de lo bueno que sea el ordenador.

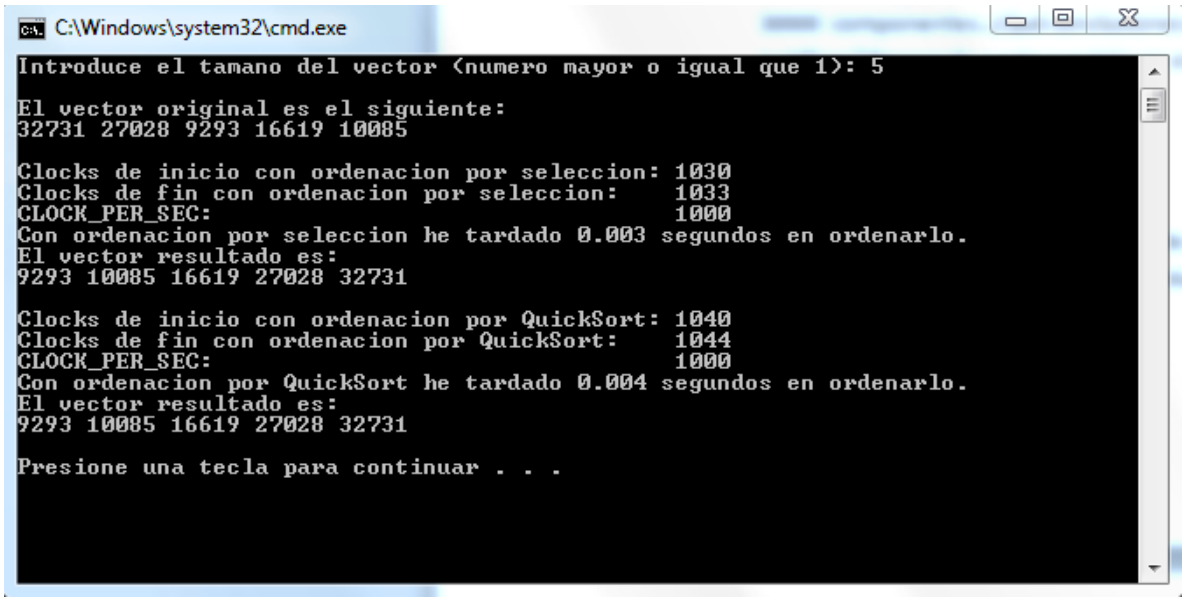
Para ello:

1. Idear un algoritmo para ordenar un array de número enteros. Se realizará una función en la cual el array de enteros se pasa como argumento (se pasa el puntero a su primer elemento), y la función lo ordena. O bien se creará una clase “VectorEnteros” que contenga como atributo un array de ints y que tenga métodos para ordenarse.
2. Programa una función (sin clase) que imprima por pantalla un array de enteros en el caso de que su número de componentes sea menor o igual que 10. En caso contrario imprimirá “<demasiados componentes para mostrar>”. Se le pasa la dirección de comienzo del vector y su número de componentes.
3. Programar un main() que haga lo siguiente:
    a. Se pide por teclado el número de componentes del array que se ordenará (al menos 1). Se imprime el array por pantalla. Se imprimen todos los números seguidos, separados por un espacio (no por un retorno de carro)
    b. Se ordena el array utilizando el algoritmo ideado. Se imprime por pantalla el array resultado y el tiempo empleado en ordenarlo.
    c. Se ordena el array utilizando el método "qsort()", que implementa el algoritmo "Quick Sort", el cual es tremendamente eficiente cuando el tamaño del array es muy grande. Se imprime por pantalla el array resultado y el tiempo empleado en ordenarlo.
4. Se harán dos pruebas: una con un array de 5 componentes, y otra con un array de 30000 componentes. ¿Qué conclusiones extraes de los resultados y cómo los justificas?

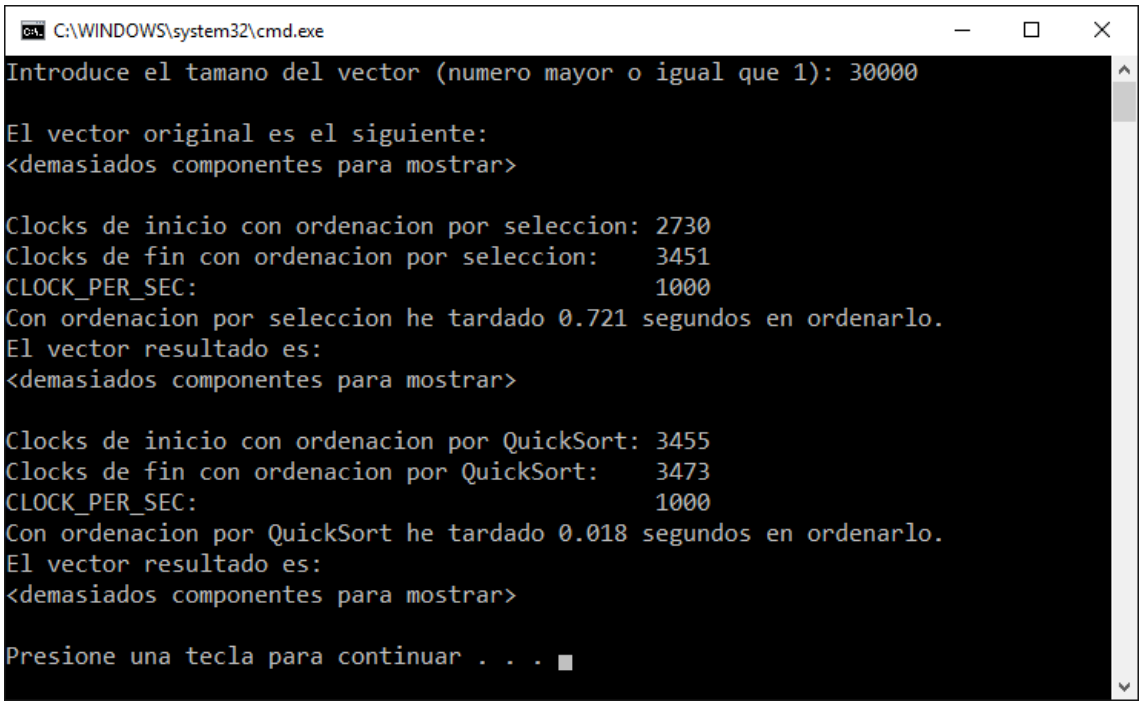
Tener en cuenta lo siguiente:

- En ningún momento de este curso podrás usar la clase predefinida de C++ llamada “vector”.
- Para medir el tiempo entre dos secciones de código, utiliza el método "clock()" de la biblioteca "ctime" (búscalo en www.cplusplus.com).
- Imprime el resultado en segundos, con suficientes decimales. Si es necesario, utiliza "printf()" y establece el número de decimales a imprimir. Busca los detalles de cómo usar printf() en www.cplusplus.com
- El método "qsort()" se encuentra en la biblioteca "cstdlib". Busca y comprende los detalles de cómo usarlo en www.cplusplus.com
- Para generar números aleatorios usar “srand()” (para establecer la semilla) y “rand()” (para generar el número aleatorio). Busca y comprende los detalles de cómo usarlos en www.cplusplus.com
- Ten en cuenta que el operador “/” en C/C++ hace la división entera si ambos operandos son de tipo “int”. En caso de que uno de ellos sea “float”, hace la división real. Tenlo en cuenta a la hora de calcular los segundos a partir de los clocks transcurridos.
- Como el tamaño del array no es constante (se define en tiempo de ejecución, pues lo introduce el usuario por teclado), es necesario usar malloc() para crear el array. Igualmente, habrá que liberar la memoria con free(). En su lugar, también se pueden utilizar “new” y “delete”, pero se recomienda utilizar malloc() y free() para entrenar con ellos.

La prueba con 5 componentes daría el siguiente resultado (este resultado puede variar debido a que el array es aleatorio):



Para la prueba con 30000 componentes, el resultado sería:


C:\WINDOWS\system32\cmd.exe
Introduce el tamano del vector (numero mayor o igual que 1): 30000
El vector original es el siguiente:
<demasiados componentes para mostrar>
Clocks de inicio con ordenacion por seleccion: 2730
Clocks de fin con ordenacion por seleccion: 3451
CLOCK_PER_SEC: 1000
Con ordenacion por seleccion he tardado 0.721 segundos en ordenarlo.
El vector resultado es:
<demasiados componentes para mostrar>
Clocks de inicio con ordenacion por QuickSort: 3455
Clocks de fin con ordenacion por QuickSort: 3473
CLOCK_PER_SEC: 1000
Con ordenacion por QuickSort he tardado 0.018 segundos en ordenarlo.
El vector resultado es:
<demasiados componentes para mostrar>
Presione una tecla para continuar . . .