

EjercicioPosteriori

Estructuras de Datos

Tema 1: tipos abstractos de datos y algoritmia

1º Grado en Ingeniería de la Computación

© Profesor Dr. Carlos Grima Izquierdo (www.carlosgrima.com)

URJC (www.urjc.es)

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:

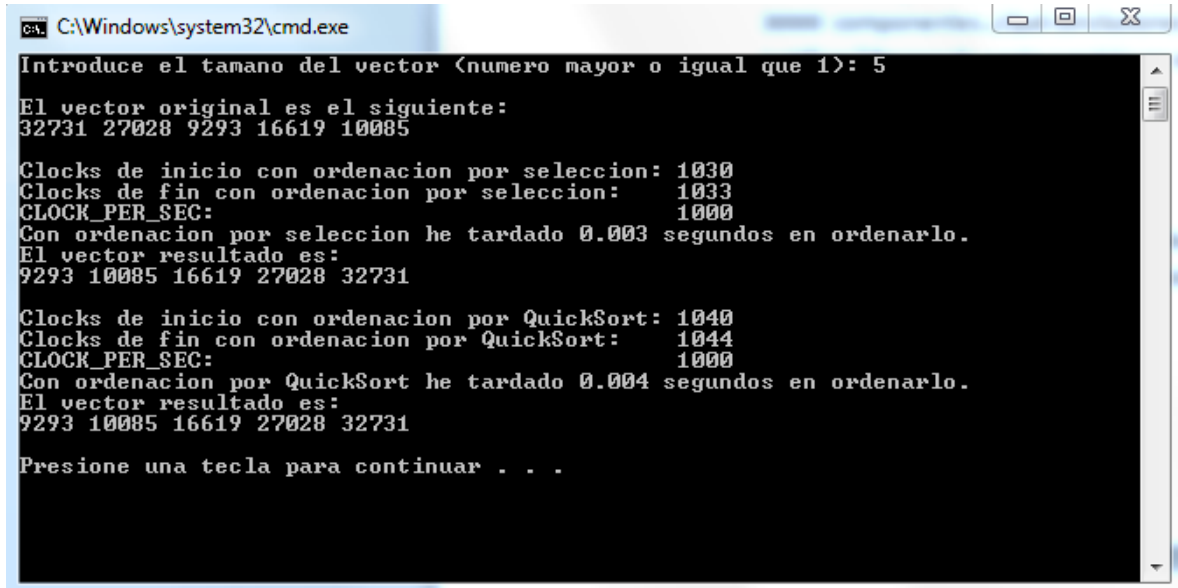
1. Programar el algoritmo de ordenación por Selección para ordenar un array de números enteros, basándote en la animación que aparece al principio de https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n. Se realizará una función (sin clase) en la cual el array de enteros se pasa como argumento (se pasa el puntero a su primer elemento y su número de elementos "n"), y la función lo ordena.
2. Para facilitar el código del algoritmo de ordenación por selección, se puede programar una función auxiliar que calcule el mínimo de una porción de un vector. Esta función tendría cuatro parámetros: el puntero al principio del vector, su tamaño ("n"), la posición de inicio desde donde buscar, y la posición de fin hasta donde buscar. Devolverá la posición (en el vector) del mínimo encontrado.
3. 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>”. Al igual que a la función de ordenar, se le pasa la dirección de comienzo del vector y su número de componentes.
4. 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 genera un array de números aleatorios. 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 de ordenación por selección. 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.

5. 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:

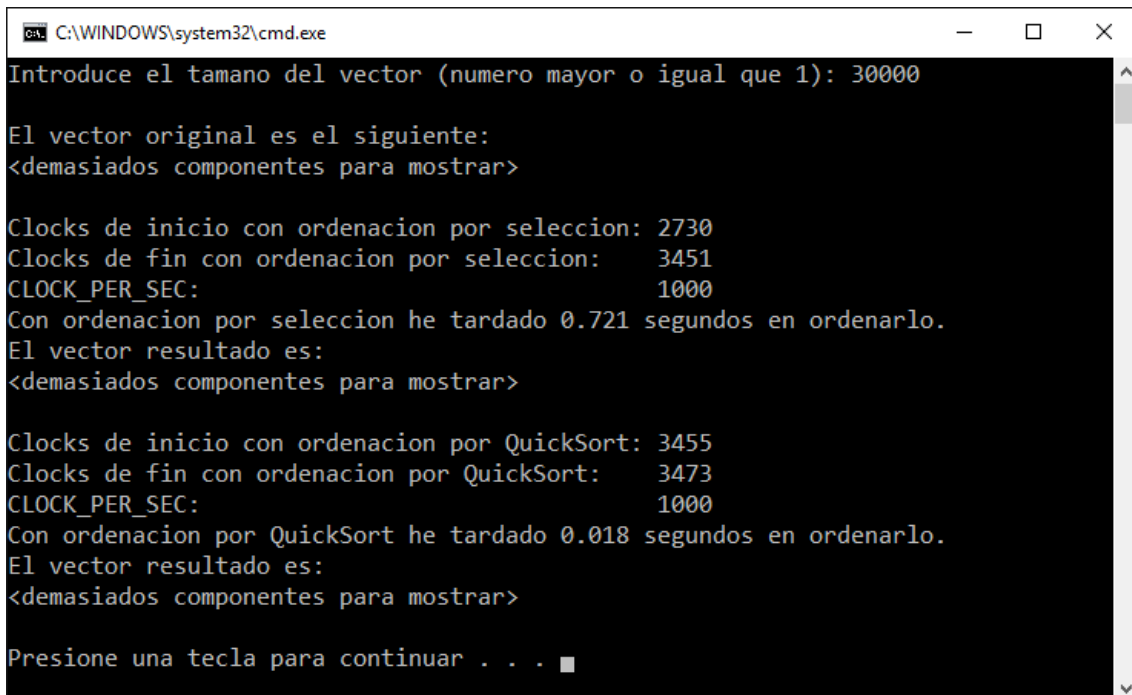
- En ningún momento de este curso podrás usar la clase predefinida de C++ llamada "vector".
- Para medir el tiempo entre dos secciones de código, utiliza el método "clock()" de la biblioteca "ctime" (búscalo en www.cplusplus.com).
- Imprime el resultado en segundos, con suficientes decimales. Si es necesario, utiliza "printf()" y establece el número de decimales a imprimir. Busca los detalles de cómo usar printf() en www.cplusplus.com
- Para obtener el número de segundos con sus decimales, tendrás que dividir los ticks que se tarda entre el número de ticks por segundo. 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.
- El método "qsort()" se encuentra en la biblioteca "cstdlib". Busca y comprende los detalles de cómo usarlo en www.cplusplus.com
- El método "qsort()" necesita, como uno de sus parámetros, el puntero a una función que compare dos componentes cualesquiera del vector a ordenar. Tienes todos los detalles y un ejemplo en la entrada de "qsort()" en www.cplusplus.com Si se quiere ordenar un array al revés (de mayor a menor), este método de comparación tendrá que devolver resultados inversos: positivo cuando $\text{operando1} < \text{operando2}$ y viceversa.
- 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 Es muy importante saber que lo ideal es llamar srand() sólo una vez en todo el programa. Por lo tanto suele ponerse al principio del main.
- 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.
- Para un mayor orden y legibilidad, vamos a dividir el proyecto en 7 archivos:
 - main.cpp, que contendrá la función main
 - impresionVectores.h e impresionVectores.cpp, que contendrá la declaración y la definición de la función para imprimir el vector
 - ordenacionQuickSort.h y ordenacionQuickSort.cpp, que contendrá la declaración y definición de la función para comparar dos números enteros (función necesaria para usar qsort()).
 - ordenacionSeleccion.h y ordenacionSeleccion.cpp, que contendrá la declaración y definición de las dos funciones necesarias para ordenar por Selección (la de buscar mínimo y la de la ordenación por selección propiamente dicha)

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



```
C:\Windows\system32\cmd.exe
Introduce el tamaño del vector (numero mayor o igual que 1): 5
El vector original es el siguiente:
32731 27028 9293 16619 10085
Clocks de inicio con ordenacion por seleccion: 1030
Clocks de fin con ordenacion por seleccion: 1033
CLOCK_PER_SEC: 1000
Con ordenacion por seleccion he tardado 0.003 segundos en ordenarlo.
El vector resultado es:
9293 10085 16619 27028 32731
Clocks de inicio con ordenacion por QuickSort: 1040
Clocks de fin con ordenacion por QuickSort: 1044
CLOCK_PER_SEC: 1000
Con ordenacion por QuickSort he tardado 0.004 segundos en ordenarlo.
El vector resultado es:
9293 10085 16619 27028 32731
Presione una tecla para continuar . . .
```

Para la prueba con 30000 componentes, el resultado sería:



```
C:\WINDOWS\system32\cmd.exe
Introduce el tamaño 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 . . . ■
```