Práctica opcional 1

Objetivo

Estudiar de manera práctica la complejidad asintótica de los distintos algoritmos de ordenación vistos en clase.

Fecha de entrega: hasta el 20 de enero.

Forma de entrega: a través del campus virtual.

Lenguaje de programación: C ó C++.

Grupos: se puede realizar de manera individual o en grupos de 2.

Calificación: hasta 0,5 puntos para cada componente del grupo. Esta nota se sumará a la obtenida en los controles, por participar en clase, etc., no pudiendo superar el total el tercio correspondiente a la evaluación continua.

Descripción

Crea un módulo *ordenación* (ficheros *ordenacion.h* y *ordenacion.cpp*) que contenga los distintos algoritmos de ordenación vistos en clase: ***selección***, ***inserción***, ***burbuja***, ***ordenación rápida*** (*quicksort*) y ***ordenación por mezcla*** (*mergesort*). Las cabeceras de las funciones deben ir en el fichero *ordenacion.h* y su implementación en el fichero *ordenacion.cpp.* Comenta tanto las cabeceras de las funciones como su implementación de manera adecuada (quiero comprobar que entiendes cómo funcionan los algoritmos).

Crea un módulo *aux* (ficheros *aux.h* y *aux.cpp*) que contenga funciones para: (1) rellenar un array con números aleatorios, (2) copiar el contenido de un array en otro, (3) comprobar si dos arrays son iguales y (4) comprobar si un array está ordenado.

Crea un módulo *test* (ficheros *test.h* y *test.cpp*) que contenga las funciones necesarias para comprobar que los algoritmos de ordenación funcionan correctamente. Para ello, genera un array de números aleatorios, haz 4 copias del array, ordena cada copia usando un algoritmo de ordenación distinto y comprueba que todos los arrays resultantes son iguales y están bien ordenados.

Crea un módulo *tiempos* que contenga las funciones necesarias para medir los tiempos de ordenación. Para cada uno de los algoritmos debes medir el tiempo que tarda en ordenar arrays con 5.000, 10.000, 15.000, …, hasta 50.000 elementos. Para evitar ruido al calcular los tiempos, repite cada operación varias veces y calcula los tiempos medios. Idealmente, el número de repeticiones debería depender del tamaño del array, de manera que los arrays pequeños se ordenen más veces y los grandes menos (de esa manera conseguirás que el experimento no tarde demasiado en ejecutarse). Para cada algoritmo de ordenación y longitud de array, imprime por consola los tiempos medios obtenidos.

Es importante que todos los algoritmos trabajen con los mismos datos de entrada para que la comparación sea justa. Antes de invocar a cada algoritmo de ordenación debes hacer una copia del array de números aleatorios original; si no lo haces, la segunda vez que invocas a un algoritmo de ordenación le estarás pasando un array ya ordenado.

Finalmente, desde el fichero *main.cpp* invoca a las funciones de test y cálculo de tiempos.

Una vez obtenidos los datos debes generar las gráficas correspondientes. Para ello puedes utilizar alguna librería auxiliar (como *gnuplot*) o copiar los datos a un fichero *Excel* y usar la utilidad de gráficas de este programa. Debes generar al menos 3 gráficas: una con todos los algoritmos de ordenación, otra sólo con los algoritmos $O(n^2)$ y otra sólo con los algoritmos $O(n \log n)$.

Material a entregar

- Código fuente.
- Gráficas de tiempos (fichero de imagen o fichero Excel).
- Memoria breve (2-3 páginas) que contenga al menos:
  - Los nombres de los integrantes del grupo.
  - Instrucciones para compilar el código y dependencias si las hubiera (librerías, …).
  - Método empleado para obtener los tiempos y generar las gráficas.
  - Gráficas de tiempos obtenidas.
  - Análisis de las gráficas (importante para la nota): ¿son los resultados esperados? ¿por qué algunos algoritmos son más lentos que otros? ¿hay diferencias incluso entre los que tienen la misma complejidad asintótica?
  - Comentarios adicionales sobre la realización de la práctica (opcional).

Pistas

Generación de números aleatorios

Para generar números aleatorios primero debes inicializar la semilla del generador usando *srand*. Esta función sólo se debe ejecutar una vez al principio de tu programa. Posteriormente puedes obtener números aleatorios invocando a la función *rand* tantas veces como sea necesario. El siguiente código se proporciona a modo de ejemplo.

```
#include<stdlib.h>
#include<time.h>

int main () {

   // Establecer una semilla para el generador de números aleatorios
   srand(time(NULL));

   // Obtener un número entero aleatorio
   int n1 = rand();
```

```
   // Obtener un número entero aleatorio entre 0 y 999
   int n2 = rand() % 1000;

   return 0;
}
```

Medición de tiempos

Para medir tiempos con suficiente precisión usaremos la función estándar *clock*. Obtendremos el número de *ticks* antes y después de la operación, y dividiremos la diferencia entre el número de *ticks* por segundo (constante *CLOCKS_PER_SEC* predefinida en cada sistema). El siguiente código se proporciona a modo de ejemplo.

```
#include<stdlib.h>
#include<time.h>

void operacion() {
   ...
}

int main () {

   // Obtener el tiempo antes de la operación
   clock_t tIni = clock();

   // Realizar la operación que queremos medir
   operacion();

   // Obtener el tiempo tras finalizar la operación
   clock_t tFin = clock();

   // Calcular el tiempo transcurrido en segundos
   float tiempo = ((float) (tFin - tIni)) / CLOCKS_PER_SEC;

   return 0;
}
```