Análisis empírico

Objetivos

- Implementar pseudocódigo de un algoritmo en C++.
- Obtener y analizar el tiempo de ejecución empírico de algoritmos implementados en C++.
- Implementar y analizar diferentes formas de trazar la ejecución de un algoritmo.
- Trabajar en grupo.

Enunciado

El siguiente algoritmo ordena ascendentemente los elementos de un vector V de tamaño n:

```
función Selección (V:&entero[n])
   i,j,posmin:entero
   para i←1 hasta n-1 hacer
      posmin ← i
      para j←i+1 hasta n hacer
         si Vj<Vposmin
            posmin ← j
         fsi
      fpara
      Intercambiar(Vi,Vposmin)
   fpara
ffunción

función Intercambiar(x:&entero, y:&entero)
   temp:entero
   temp ← x
   x ← y
   y ← temp
ffunción
```

Actividades (Equipos de 2 o 3 alumnos)

Actividad 1. Parte 1: ordenar vector cualquiera con datos aleatorios

1.- Implementa un programa para obtener el tiempo de ejecución empírico del algoritmo que realiza la ordenación de un vector de números enteros en un caso cualquiera. El programa tendrá definidas la función main y las funciones Selección e Intercambiar.

Los pasos que sigue el programa son:

- **Paso 1.** Generar un vector del tamaño que el usuario introduzca por teclado e inicializarlo con números aleatorios.

  Para generar los números aleatorios se cogerá como semilla un valor fijo definido en una constante del programa. Para establecer una semilla se utiliza la función **srand**(semilla), donde semilla es la variable que contiene el valor de la semilla a partir de la cual se generan los números aleatorios. Esta función solamente hay que llamarla una vez. A continuación, se deberá utilizar la función **rand**() para generar los números aleatorios; cada vez que se llame a esta función se generará un nuevo número aleatorio. Ambas funciones srand y rand se encuentran en la **librería stdlib.h**.

- **Paso 2:** utilizando el vector cualquiera generado, mostrar sus 10 primeras posiciones antes de ordenarlo, invocar el algoritmo de ordenación y mostrar el tiempo de ejecución empírico en milisegundos que ha tardado (consultar el **Anexo I**). Por último, volver a mostrar las 10 primeras posiciones del vector una vez ordenado (que ahora deberán estar ordenadas de menor a mayor).

Actividad 1. Parte 2: ordenar un vector en su caso mejor y peor

Extender el código implementado en la función main de la parte 1 para medir el tiempo de ejecución al ordenar: (1) un vector que represente el caso mejor, y (2) un vector que represente el caso peor.

Actividad 2: trazando el funcionamiento del algoritmo/programa

Trabajar en la traza del funcionamiento del algoritmo y del programa para un caso concreto de entrada. Definir una configuración de entrada del problema que te permita analizar el comportamiento de algoritmo desde un punto de vista práctico. Elaborar una memoria en la que se represente gráficamente la configuración del problema elegida (valor de n, contenido del vector…) y completarla con las siguientes trazas:

1) **Traza manual:** se deberá realizar a mano una traza del algoritmo en la que se observe como se van intercambiando los valores del vector elegido como ejemplo, y como se va modificando su contenido durante todo el proceso de ordenación.
2) **Traza basada en la salida del programa**: incluir en el código del algoritmo las sentencias necesarias para replicar la traza del punto 1 usando funciones que permitan escribir el contenido de las diferentes variables del algoritmo en la consola (ej., cout, printf…). Adjuntar en la memoria la captura de pantalla de la salida del programa incluyendo esta segunda traza.
3) **Traza utilizando un depurador:** por último, explorar las posibilidades de depuración ofrecidas por el IDE o entorno de desarrollo que se esté utilizando. Trazar el programa con el depurador y adjuntar una captura de pantalla en la que se observe el estado de las variables del programa en el momento en el que se produce el primer intercambio en el vector.

Responde en la memoria a las siguientes preguntas: ¿crees que realizar trazas es una labor útil? ¿conocías o habías utilizado los tres tipos de trazas comentados? ¿las utilizarás en el futuro?

Modo de entrega

La práctica se realizará en equipos de **dos o tres alumnos** y se entregarán los siguientes ficheros con los nombres que se indican.

Archivo comprimido:	practica2.zip
Contenido del archivo:	
código fuente	contiene el código fuente y los nombres de los miembros del equipo. p2.cpp
memoria	fichero PDF con el contenido solicitado en la actividad 2.

Todos los componentes del equipo entregarán el archivo practica2.zip en la tarea del campus virtual relativa a esta sesión.

Fecha fin de entrega: Domingo, 27 de febrero de 2022 a las 23:59.

Evaluación

La calificación de la actividad es de 0,1 puntos (0,05 por cada actividad).

A continuación, se indica el sistema de evaluación:

- **Opción A: La práctica se entrega durante la sesión de prácticas.**

Cada alumno/-a del equipo **entregará la práctica** (practica2.zip) **en la tarea del campus virtual** de la asignatura. Antes de subir la tarea a la web se debe recibir el visto bueno del profesorado y todos los miembros del equipo deben estar presentes y explicar cualquier aspecto que se solicite. Se evaluará cogiendo al azar la práctica de uno de los miembros del equipo, de forma que dicha práctica será la que se corrija.

Si hay algún miembro del equipo que no entrega la práctica en la tarea, no responde correctamente a las preguntas realizadas por el profesorado o no está presente…, no se le calificará según esta opción y puede optar a la calificación según la opción B.

- **Opción B: La práctica se entrega en horario posterior a la sesión de prácticas.**

A esta opción optarán los **equipos y miembros de equipos que no cumplen los requisitos para ser evaluados según la opción A**. Cada miembro del equipo debe entregar la tarea antes de la fecha de fin de entrega y todas las actividades deberán ser correctas. Se calificará cogiendo al azar la práctica de uno de los miembros del equipo, de forma que dicha práctica será la que se corrija. El programa debe seguir las especificaciones que se dan.

La calificación máxima que se puede obtener bajo esta Opción B es de 0,05 puntos (0,025 cada actividad).

- **<u>Opción C:</u> 0 puntos**

- El alumno/a:
  - No entrega la práctica en la tarea de la web de la asignatura.
  - No asiste a la sesión de prácticas cumpliendo con las condiciones establecidas.
- El programa no funciona correctamente y no realiza lo que se pide.
- Se detecta copia con otras prácticas. La nota será un 0 en esta práctica para todas las prácticas implicadas, aun cuando la práctica haya sido valorada previamente de forma positiva por parte del profesorado.

Anexo: Medida del tiempo de ejecución empírico

Para obtener el tiempo de ejecución empírico se pueden utilizar diferentes funciones. En esta práctica utilizamos la función clock() incluida en la librería time.h. El prototipo de la función es: clock_t clock(void).

Esta función devuelve un valor **aproximado** del tiempo transcurrido en **pulsos del reloj** del sistema desde la última vez que se llamó a la función. Si hay algún error, la función devuelve el valor -1.

Para transformar este valor a segundos se divide el valor resultante por una constante llamada CLOCKS_PER_SEC. Esta constante generalmente es igual a 1000 pero puede cambiar según sea el sistema operativo y el compilador que utilices. Puedes imprimir por pantalla esta constante para saber su valor.

NOTAS:

- La precisión es aproximadamente de 10 milisegundos. Para esta práctica consideraremos adecuada esta precisión. Para obtener un valor más aproximado al real se puede ejecutar el programa muchas veces y obtener el promedio de los tiempos resultantes.
- Existen otras funciones para obtener el tiempo de ejecución de una parte del código de un programa como por ejemplo gettimeofday, que funciona bien en Linux y que en Windows tiene una precisión similar a clock().

El siguiente programa se muestra cómo calcular el tiempo de ejecución en milisegundos que tarda una función llamada Calcular.

```
#include <iostream>
#include <time.h>
using namespace std;

void Calcular(void)
{
   …          // código de la función Calcular
}

int main(void)
{
   clock_t tinicio, tfin;
   double tiempo;
   int resultado;

   tinicio = clock();                              // almacenamos el instante actual
   resultado = Calcular();                         // ejecutamos la función
   tfin = clock();                                 // almacenamos el instante actual

   tiempo = (double)(tfin-tinicio) / CLOCKS_PER_SEC * 1000; // resultado en milisegundos

   cout << "El tiempo de ejecucion en ms es " << tiempo << endl;

   return 0;
}
```