Análisis empírico y asintótico

Objetivos

- Implementar pseudocódigo de un algoritmo en C++ siguiendo las reglas de transformación de la práctica 1.
- Obtener el tiempo de ejecución empírico de algoritmos implementados en C++.
- Analizar el tiempo de ejecución empírico de un algoritmo utilizando una representación gráfica.
- Analizar las tasas de crecimiento de la complejidad asintótica de un algoritmo utilizando una representación gráfica.
- Trabajar en grupos de 2 o 3 miembros.

Enunciado

El siguiente algoritmo ordena ascendentemente los elementos de un vector V de tamaño n, considerando vectores desde la posición 1 hasta la posición n.

```
función Inserción (V:&entero[n])
   i,j,x:entero
   para i←2 hasta n hacer
      x ← Vi
      j ← i-1
      mientras j>0 y Vj>x hacer
         Vj+1 ← Vj
         j ← j-1
      fmientras
      Vj+1 ← x
    fpara
ffunción
```

Actividades

Actividad 1:

Implementa un programa para obtener el tiempo de ejecución empírico del algoritmo para realizar la ordenación de un vector de números enteros en un caso cualquiera, en el caso mejor y en el caso peor utilizando el algoritmo anterior. Aplica las reglas de transformación de pseudocódigo a implementación en C++ para codificar la función Inserción.

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. Este vector será el vector correspondiente al caso cualquiera.

  Para generar los números aleatorios se cogerá como semilla un valor que se pida por teclado. Para establecer esta semilla se utiliza la función `srand(semilla)`, donde semilla es una 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. Una vez llamada a la función `srand` para establecer la semilla inicial a partir de la cual se generan los números aleatorios utiliza la función `rand()` para generar los números aleatorios. Cada vez que se llama a esta función se genera un nuevo número aleatorio. Las funciones `srand` y `rand` se encuentran en la librería `stdlib.h`.

  En el Anexo 1 se muestra un ejemplo de uso de las funciones `srand` y `rand`.

- **Paso 2:** Aplicar el algoritmo de ordenación para el caso cualquiera generado en el paso anterior, para el caso mejor y para el caso peor. En cada caso se mostrarán cuáles son los valores del vector a ordenar,

el vector ordenado tras aplicar el algoritmo Para simplificar la salida por pantalla de los vectores, solamente se muestran los valores comprendidos entre las posiciones que el usuario introduce por teclado (ambas posiciones incluidas).

Además de imprimir los vectores también se imprimirá el tiempo de ejecución empírico en milisegundos que el algoritmo tarda en realizar la ordenación para cada uno de los casos. En el Anexo 2 se muestra información sobre cómo calcular el tiempo de ejecución empírico.

Ejemplo de ejecución:

En la Figura 1 se muestra la salida del programa tomando como semilla el valor 20, para un vector de 5000 elementos y mostrando por pantalla los elementos del vector que ocupan las posiciones desde el valor 1 hasta el 10. Todos estos valores se piden por teclado. Los tiempos de ejecución y valores aleatorios mostrados dependerán de la máquina en la que se ejecute el programa, compilador…

```
Semilla inicial para generar numeros aleatorios: 20
Introduce tamaño del vector (n): 5000
Posiciones inicial y final del vector para mostrar
Inicial: 1
Final: 10

ALGORITMO DE INSERCION

CASO CUALQUIERA
---------------
Vector a ordenar: 103 26079 18073 24951 18538 24795 5078 6508 13002 5955
Vector ordenado:  7 9 14 23 29 29 33 33 37 37
Tiempo de ejecucion (ms): 21

CASO MEJOR
---------------
Vector a ordenar: 7 9 14 23 29 29 33 33 37 37
Vector ordenado:  7 9 14 23 29 29 33 33 37 37
Tiempo de ejecucion (ms): 0

CASO PEOR
---------------
Vector a ordenar: 5000 4999 4998 4997 4996 4995 4994 4993 4992 4991
Vector ordenado:  1 2 3 4 5 6 7 8 9 10
Tiempo de ejecucion (ms): 32

Presione una tecla para continuar . . .
```

Figura 1: Ejemplo de ejecución.

Actividad 2:

2.- Realiza un análisis comparativo de los tiempos de ejecución empírico y asintótico del comportamiento del algoritmo para diversos valores del tamaño del problema (n) y muestra los resultados en un gráfico tal y como se indica a continuación. Escribe las conclusiones que obtienes de los resultados empíricos y asintóticos.

Caso empírico: Representa los valores de los tres casos: caso cualquiera, caso mejor y caso peor.

En la Figura 2 se muestra un ejemplo genérico de un gráfico con los tiempos de ejecución empíricos para los tres posibles casos de un algoritmo. En el eje horizontal se indica el tamaño del problema y en el vertical el tiempo de ejecución en milisegundos.

Como el tiempo de ejecución depende de factores externos hay que especificar cuáles son las características hardware del equipo físico (procesador, memoria RAM…) y el software (sistema operativo, lenguaje de programación, compilador...).

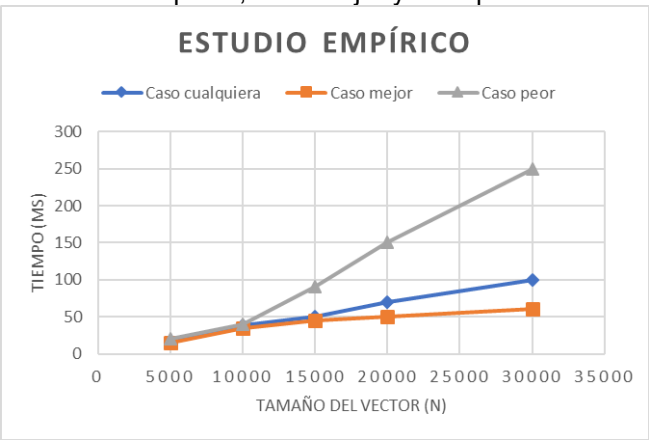

Figura 2: Tiempo de ejecución empírico.

Caso asintótico: Representa las complejidades asintóticas de los casos mejor y peor.

En la Figura 3 se muestra un gráfico con las tasas de crecimiento de un algoritmo suponiendo que los casos peor y mejor tienen una tasa de crecimiento asintótica logn y n^2, respectivamente.

En el eje horizontal se indica el tamaño y en el vertical el valor del orden.

Figura 3: Tasas de crecimiento.

Anexo 1: Ejemplo de uso de las funciones srand y rand

El siguiente código muestra por pantalla la generación de n números aleatorios tomando utilizando una semilla inicial. Los valores de la semilla inicial (semilla) y el número total de elementos a generar (n) se piden por teclado.

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

using namespace std;

int main()
{
     int semilla,n,i;

     cout << "Introduce semilla: ";
     cin >> semilla;

     cout << "Introduce total de elementos a generar: ";
     cin >> n;

     srand(semilla);
     for (i=1;i<=n;i++)
        cout << rand() << " ";
     cout << endl;

     system("pause");
     return 0;
}
```

A continuación se muestra la salida por pantalla del programa para una semilla con valor 10 y un total de n=5 elementos.

```
Introduce semilla: 10
Introduce total de elementos a generar: 5
71 16899 3272 13694 13697
Presione una tecla para continuar . . .
```

Anexo 2: 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 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;

   tinicio = clock();                  // almacenamos el instante actual
   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;
}
```