Análisis empírico y asintótico

Objetivos

- Realizar trazas de programas usando el depurador GDB y compara con trazas basadas en cout.
- Analizar el tiempo de ejecución empírico de diversos algoritmos utilizando gráficos.
- Analizar las tasas de crecimiento de la complejidad asintótica de diversos algoritmos utilizando gráficos.
- Implementar algoritmos recursivos en C++.
- Trabajar en grupo.

Enunciado

Actividades (Grupos de 2-3 alumnos)

Actividad 1: análisis asintótico

Usando como punto de partida la práctica anterior, realiza un análisis comparativo de los tiempos de ejecución empírico y asintótico del comportamiento del algoritmo para diversos valores del tamaño del problema 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 1 se muestra un ejemplo de un gráfico con los tiempos de ejecución empíricos para dos supuestos algoritmos llamados Algoritmo1 y Algoritmo2. 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 tener en cuenta cuáles son las características hardware del equipo físico (procesador, memoria RAM…) y el software (sistema operativo, lenguaje de programación, compilador...), en el que se realizan los experimentos.

En esta actividad el gráfico contendrá tres líneas, una para cada caso analizado del algoritmo.

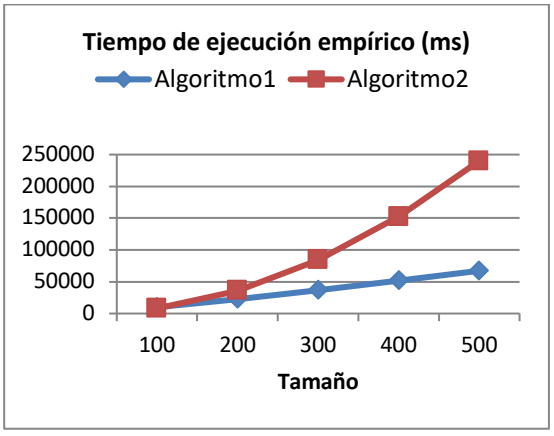


Figura 1: Tiempo de ejecución empírico.

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

En la Figura 2 se muestra, como ejemplo, un gráfico con las tasas de crecimiento de los algoritmos Algoritmo1 y Algoritmo2, suponiendo que sus órdenes asintóticos son O(logn) y $O(n^2)$, respectivamente.

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

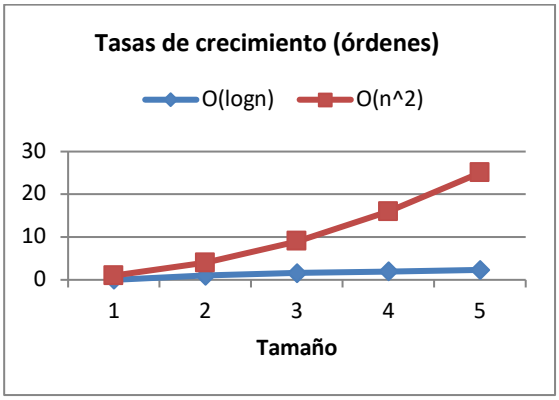


Figura 2: Tasas de crecimiento.

Actividad 2: recursividad y GDB (la pila de llamadas)

Dada la siguiente función recursiva que calcula la potencia de a^b:

```
función potencia2 (a:real, b:natural U{0}):real
   s:real
   si b = 0
      devolver 1
   si no
      s ← potencia2(a , b / 2)
      si espar(b)
         devolver s * s
      si no
         devolver a * s * s
      fsi
   fsi
función
```

a) Escribir la expresión matemática que representa la función del pseudocódigo anterior.

b) Implementar en C++ la función llamada **potencia2** correspondiente al pseudocódigo anterior y crear un programa que pida por teclado los valores de `a` y `b`, realice una llamada a la función e imprima el resultado de la operación.

c) A continuación, se muestra un ejemplo de una traza manual que contiene errores para un ejemplo concreto con a=2 y b=5. Ten en cuenta que al inicio de cada línea aparece el número de llamada recursiva: "1.-", "2.- "", 3.-, etc. y a continuación el nombre de la función junto con el valor de los parámetros a y b de cada llamada. Por otro lado, las llamadas recursivas que se realicen desde una función se imprimirán un nivel más adentro. Identifica y corrige los errores de la traza anterior.

Con errores:	Sin errores:
1.- potencia2 (2,5) 2.- potencia2 (2,3) 3.- potencia2 (2,2) 3.- potencia2 (1,2) 4.- potencia2 (1,1) 5.- potencia2 (1,0)	¿?¿?

c) Trazar el código: Utilizar el depurador GDB (ver Anexo I) para hacer un seguimiento de la pila de llamadas recursivas que se producen al ejecutar el programa para dicho ejemplo. implementa una nueva función llamada **potencia2_traza** que reproduzca la traza anterior mediante cout.

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:		practica3.zip
Contenido del archivo:		
	código fuente	contiene el código fuente y los nombres de los miembros del equipo. p3.cpp
	memoria	fichero PDF con el contenido solicitado en las actividades.

Todos los componentes del equipo entregarán el archivo practica3.zip en la tarea del campus virtual relativa a esta sesión.

Fecha fin de entrega: Domingo, 12 de marzo de 2023 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** (practica3.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).

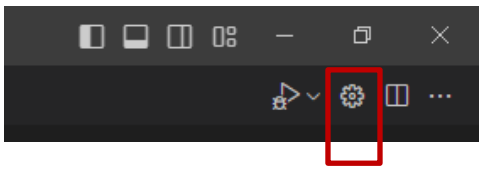
- **Opción C: 0 puntos**

- 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.

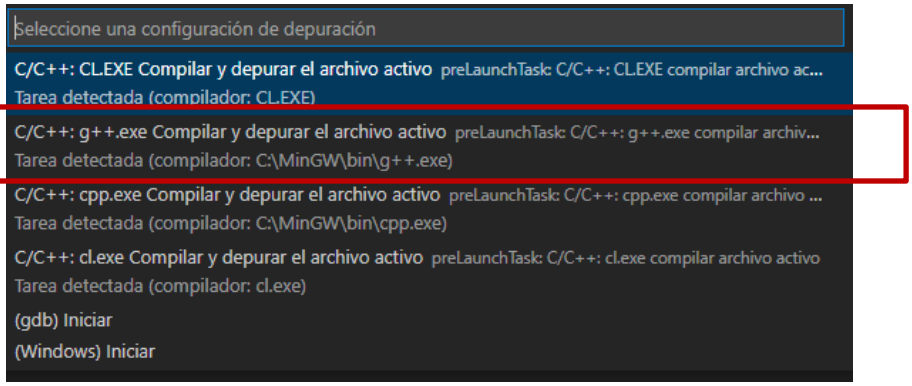
Anexo I: uso de GDB

Una de las características clave de Visual Studio Code (VS Code) es su gran soporte para la depuración. Uno depurador muy utilizado para el lenguaje C++ es GDB, y se puede utilizar directamente desde el propio IDE. A continuación, describimos algunos pasos a seguir para poder utilizar dicho depurador.

Lo primero que se debe hacer es crear un fichero de configuración. Esto se puede hacer de forma automática clicando en el engranaje que aparece en la esquina superior derecha de VS Code. Para que dicho engranaje aparezca, se debe tener abierta una carpeta y un fichero en formato ".cpp".



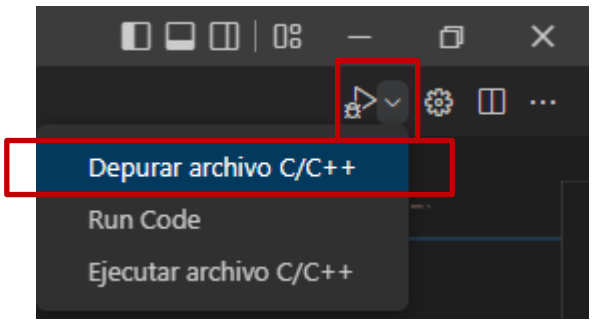
Una vez clicado el engranaje, se abrirá un desplegable en la parte central donde deberemos seleccionar el compilador de C++: g++.



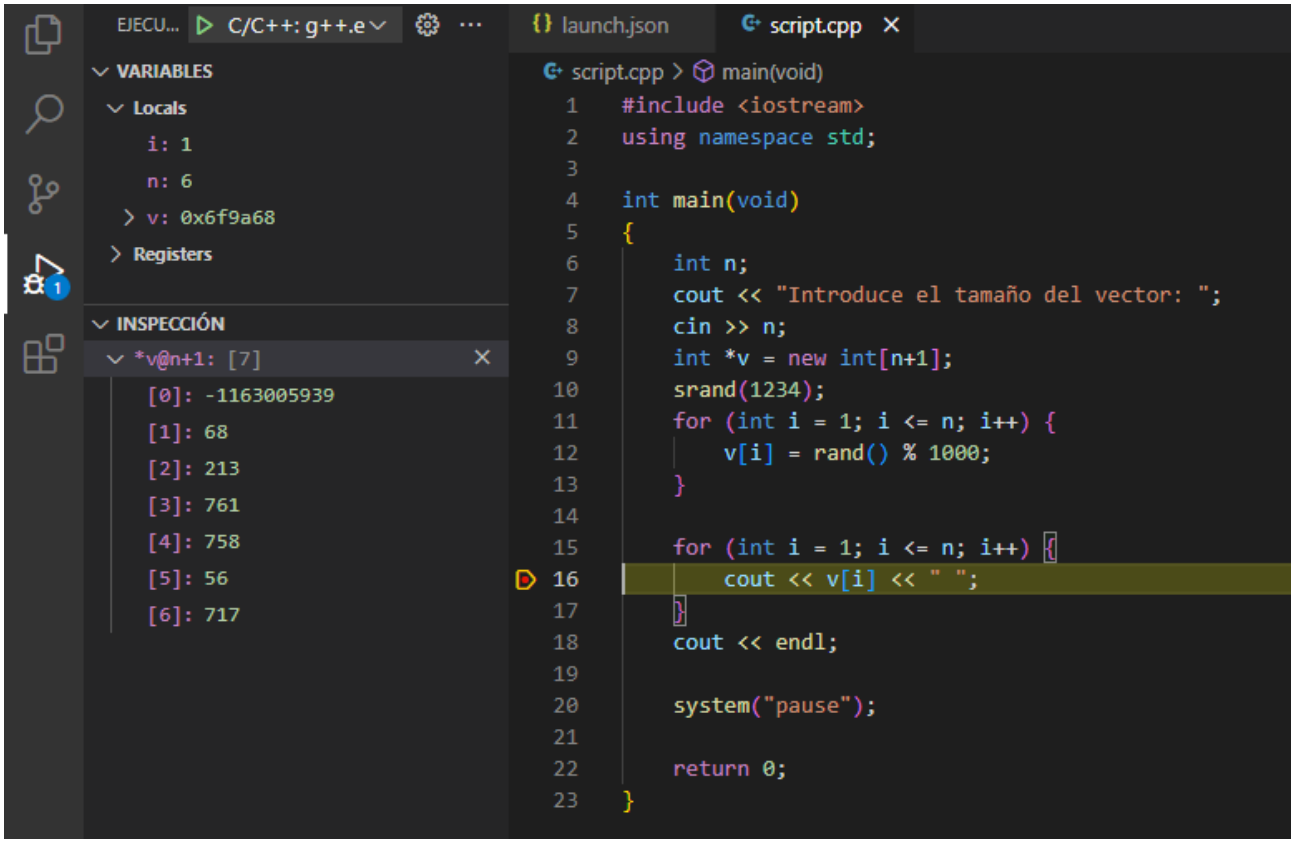
Tras ello, se creará de forma automática un fichero ***launch.json*** que será nuestro fichero de configuración. En este fichero debemos modificar el parámetro "*externalConsole*" y darle el valor "*true*".

```
{
    "configurations": [
        {
            "name": "C/C++: g++.exe Compilar y depurar el archivo activo",
            "type": "cppdbg",
            "request": "launch",
            "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${fileDirname}",
            "environment": [],
            "externalConsole": true,
            "MIMode": "gdb",
            "miDebuggerPath": "C:\\MinGW\\bin\\gdb.exe",
            "setupCommands": [
                {
                    "description": "Habilitar la impresión con sangría para gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                },
                {
                    "description": "Establecer tipo de desensamblado en Intel",
                    "text": "-gdb-set disassembly-flavor intel",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "C/C++: g++.exe compilar archivo activo"
        }
    ],
    "version": "2.0.0"
}
```

En este momento, ya estamos en condiciones de utilizar el depurador. El siguiente paso será activar puntos de interrupción. Los puntos de interrupción son el lugar de parada o pausa intencional en un programa, con el fin de depurarlo, y pueden activarse haciendo clic en el margen del editor o utilizando F9 en la línea actual. Por ejemplo, en la figura del final de esta página se aprecia un punto de interrupción (puntito rojo) en la lína 16. Una vez activados los puntos de interrupción, se puede lanzar nuestro programa en modo depuración haciendo clic en el botón "*play*" situado al lado del engranaje o utilizando la tecla F5.



Tras ello, se abrirá de forma automática el panel de "Ejecución y depuración", donde en el margen izquierdo aparecerán las variables del método que se esté ejecutando en ese momento. Además, para mostrar el **contenido de un vector**, en la sección de "inspección" se debe añadir una nueva variable cuyo valor se escribirá siguiendo el patrón: "* + nombre del vector + @ + la longitud del vector".



Una vez iniciada la depuración, se podrá continuar la ejecución del programa de diferentes modos. Por ejemplo, se puede ir línea a línea, continuar hasta el siguiente punto de interrupción… Además, cada vez que se avance, en la pestaña de las variables se resaltarán aquellos valores que han sufrido cambios desde el último paso.