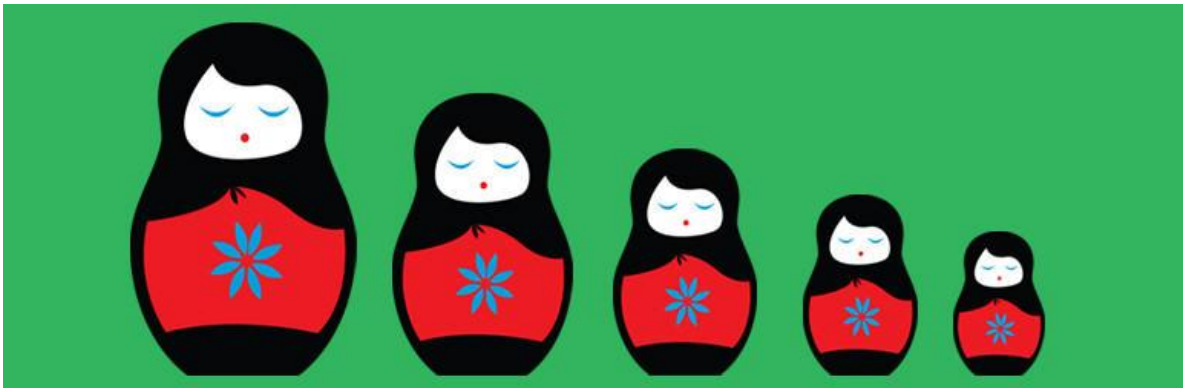
INSTITUTO POLITÉCNICO NACIONAL

Escuela Superior de Cómputo
(ESCOM)

Práctica 3: recursividad



Análisis y diseño de Algoritmos

3CV5

Prof. Andrés García Floriano

Alumno: Solis Bustos Daniel

Introducción.

En el presente trabajo se realiza el análisis e implementación de versiones **iterativas y recursivas** de distintos algoritmos fundamentales en la programación: la suma de los dígitos de un número, la inversión de una cadena, la búsqueda binaria y la potenciación. Estos algoritmos permiten comprender la diferencia entre resolver un problema mediante estructuras repetitivas y mediante llamadas recursivas.

El objetivo principal de esta práctica es **comparar el desempeño** de ambas soluciones, midiendo su **tiempo de ejecución** y su **consumo de memoria**, así como analizar el comportamiento de la **pila de llamadas** en los algoritmos recursivos. Para ello, se utilizó el lenguaje de programación **C**, debido a su cercanía con el manejo de memoria y su eficiencia.

Este análisis permite reforzar conceptos importantes como la complejidad computacional, el uso de la recursión y el impacto que tiene en los recursos del sistema, conocimientos esenciales en la formación de un ingeniero en sistemas computacionales.

Desarrollo.

Para el desarrollo de la práctica se implementaron dos versiones de cada algoritmo: una **iterativa**, basada en ciclos, y otra **recursiva**, basada en llamadas a la misma función hasta alcanzar un caso base. Todas las implementaciones se realizaron en el lenguaje C, utilizando arreglos dinámicos y estáticos según el caso.

En la **suma de dígitos**, el número se manejó como una cadena de caracteres para poder recorrer cada dígito de manera sencilla. La versión iterativa utiliza un ciclo for, mientras que la versión recursiva procesa un dígito por llamada hasta llegar al final de la cadena. En este caso, la solución recursiva genera múltiples llamadas en la pila, una por cada dígito del número.

Para la **inversión de una cadena**, se intercambiaron los caracteres desde los extremos hacia el centro. La versión iterativa emplea un ciclo while, mientras que la versión recursiva intercambia los caracteres y se llama a sí misma reduciendo el rango de la cadena en cada llamada. Este algoritmo permite visualizar claramente cómo se va construyendo y liberando la pila de llamadas recursivas.

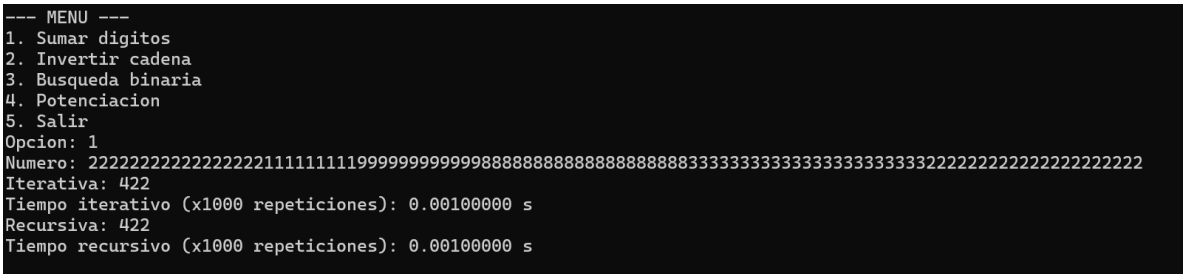
En la **búsqueda binaria**, se utilizó un arreglo ordenado de gran tamaño para resaltar la eficiencia del algoritmo. La versión iterativa ajusta los índices mediante un ciclo, mientras que la versión recursiva divide el problema en subarreglos más pequeños en cada llamada. Ambas versiones presentan una complejidad de **O(log n)**, aunque la versión recursiva requiere memoria adicional debido a la pila de llamadas.

Finalmente, en la **potenciación**, se implementó el método de exponenciación rápida tanto de forma iterativa como recursiva. Este método reduce considerablemente el número de operaciones necesarias al dividir el exponente entre dos en cada paso. La versión recursiva muestra claramente el proceso de división del problema y el uso de la pila para almacenar resultados parciales.

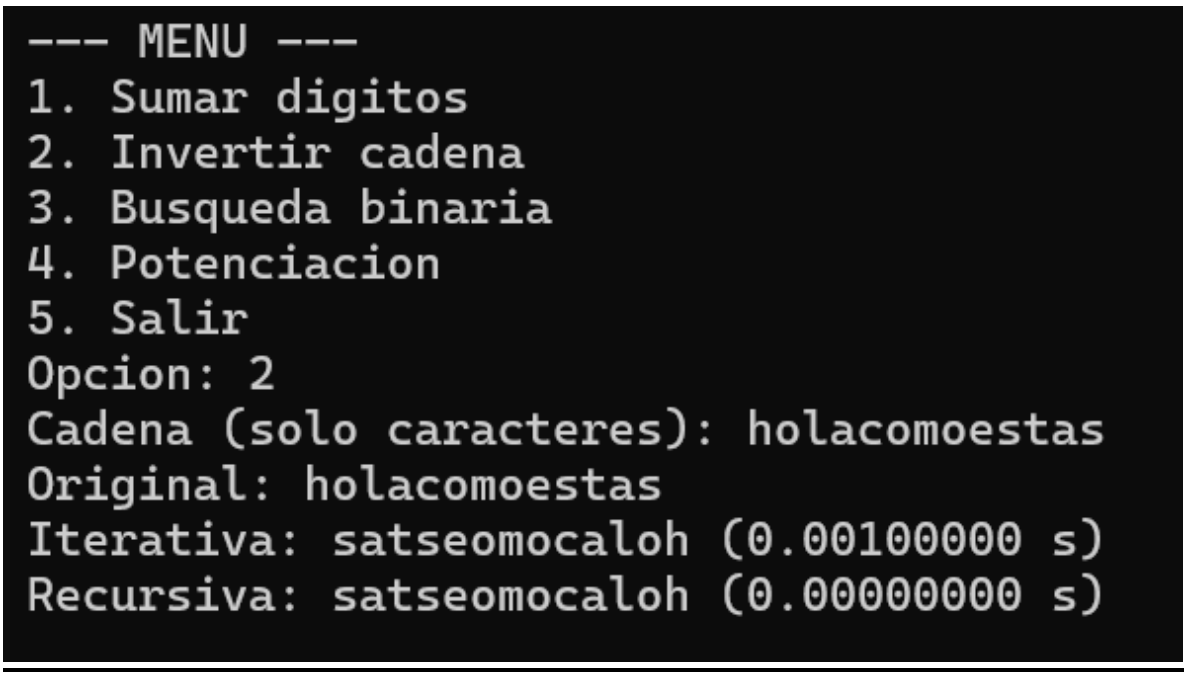
Para medir el **tiempo de ejecución**, cada algoritmo se ejecutó múltiples veces y se utilizó la función clock() de la librería <time.h>. El **consumo de memoria** se analizó de forma teórica, considerando que las versiones recursivas requieren memoria adicional para cada llamada en la pila, mientras que las versiones iterativas mantienen un uso de memoria constante.

Evidencias

Suma de dígitos.



Invertir cadenas.



Búsqueda binaria.

```
--- MENU ---
1. Sumar digitos
2. Invertir cadena
3. Busqueda binaria
4. Potenciacion
5. Salir
Opcion: 3
Numero a buscar (0..9999999): 120940
Iterativa -> Pos: 120940 (0.00000000 s)
Recursiva -> Pos: 120940 (0.00100000 s)
```

Potenciación.

```
--- MENU ---
1. Sumar digitos
2. Invertir cadena
3. Busqueda binaria
4. Potenciacion
5. Salir
Opcion: 4
Base: 9
Exponente: 9999999999999999
Iterativa: 662963356
Tiempo iterativo (1000 repeticiones): 0.00100000 s
Recursiva: 662963356
Tiempo recursivo (1000 repeticiones): 0.00300000 s
```

Conclusión.

A partir de los resultados obtenidos, se concluye que tanto las soluciones iterativas como las recursivas permiten resolver correctamente los problemas planteados; sin embargo, presentan diferencias importantes en cuanto a **eficiencia y uso de memoria**.

Las versiones **iterativas** tienden a ser más eficientes en tiempo y consumo de memoria, ya que no generan llamadas adicionales en la pila. Por otro lado, las versiones **recursivas** ofrecen una solución más clara y elegante en algunos casos, pero implican un mayor consumo de memoria debido a la pila de llamadas, lo cual puede afectar el rendimiento en problemas de gran tamaño.

La búsqueda binaria y la potenciación demostraron que la recursión puede ser muy eficiente cuando el problema se divide adecuadamente, manteniendo una complejidad baja. No obstante, es importante considerar el contexto y las limitaciones del sistema al elegir entre una solución iterativa o recursiva.

En conclusión, esta práctica permitió reforzar el entendimiento del funcionamiento interno de los algoritmos, la recursión y la gestión de memoria, aportando una base sólida para el análisis y diseño de soluciones eficientes en el desarrollo de software.