Logotipo

Descripción generada automáticamente **Escuela Politécnica Nacional**

**Facultad de Ingeniería de Sistemas**

**Matemáticas Computacionales y Teoría de la Computación**

Examen

**Autor:**

* Ruth Rueda

**Fecha de Entrega:** 06/03/2024

**Docente:** Ing. Mauricio Loachamín.

**2023-****B**

Contenido

[1. Introducción 1](#_Toc160572803)

[2. Objetivos 1](#_Toc160572804)

[3. Desarrollo del Programa 1](#_Toc160572805)

[4. Determinación de la complejidad 2](#_Toc160572806)

[5. Características del equipo 2](#_Toc160572807)

[6. Ejecución del programa 3](#_Toc160572808)

[7. Conclusiones 4](#_Toc160572809)

[8. Referencias 4](#_Toc160572810)

[Anexo I 1](#_Toc160572811)

# Introducción

El presente informe detalla el desarrollo de un programa en Java diseñado para calcular el tiempo de ejecución de las operaciones de suma y multiplicación de matrices de orden *n*x*n*. El programa permite analizar la complejidad temporal de un algoritmo mediante el ingreso de distintos valores de entrada. Además, se analiza la variación en el tiempo de ejecución considerando las especificaciones del hardware. A lo largo del informe, se proporciona una descripción completa de la implementación, se presentan resultados detallados de análisis de tiempo de ejecución y se aborda la complejidad algorítmica. Este documento permite comprender tanto la funcionalidad práctica del programa como su eficiencia y escalabilidad.

# Objetivos

* Analizar el tiempo de procesamiento en distintos ordenadores, ya que el tiempo en el cual se hace el proceso de sumar depende de las especificaciones del computador
* Analizar la complejidad algorítmica que permitirá entender cómo el rendimiento del programa se ve afectado según el tamaño de la matriz.

# Desarrollo del Programa

Se desarrolló un programa que permita ingresar al usuario el orden de la matriz para posteriormente generar dos matrices del mismo orden, donde los elementos de la matriz son generados de forma aleatoria en un intervalo del -100 al 100. El programa permite validar que el orden de la matriz sea un valor numérico entero mayor a cero.

Para determinar el tiempo de ejecución del programa se utilizó el método nanoTime( ) y luego se transforma el tiempo transcurrido a milisegundos.

|  |
| --- |
| // Se inicia el temporizador  long tiempoInicio = System.nanoTime();  //Aquí va el código para medir su tiempo de ejecución  // Finaliza el temporizador  long tiempoFinal = System.nanoTime();  // Calcula el tiempo transcurrido  long tiempoTranscurrido = tiempoFinal - tiempoInicio;  // Convierte el tiempo a segundos  double milisegundos = (double) tiempoTranscurrido / 1\_000\_000.0;  System.out.println("Tiempo de ejecución: " + milisegundos + " milisegundos"); |

El código para medir el tiempo de ejecución generalmente consiste en operaciones de inicio y finalización del temporizador que tienen un impacto mínimo en el rendimiento del algoritmo principal. Estas operaciones de medición del tiempo suelen ser de complejidad constante, lo que significa que su tiempo de ejecución no depende del tamaño de la entrada del algoritmo principal.

Para medir el tiempo de ejecución del programa se inicia el temporizador después que el usuario haya ingresado el orden de la matriz correctamente para determinar la complejidad del algoritmo que permite realizar la suma o la multiplicación de dos matrices.

El código desarrollado se encuentra en el Anexo 1.

# Determinación de la complejidad

Para determinar la complejidad del algoritmo, se examina los métodos:

**Generación de matrices aleatorias:** El método generarMatrizAleatoria recorre cada celda de la matriz y asigna un valor aleatorio. La complejidad de esta operación es lineal con respecto al tamaño de la matriz, es decir, , donde *n* es el orden de la matriz.

**Suma de matrices:** El método sumarMatrices recorre cada celda de las matrices de entrada y realiza una suma. Al igual que la generación de matrices, la complejidad de esta operación es lineal con respecto al tamaño de la matriz, es decir, , donde *n* es el orden de la matriz.

**Impresión de la matriz:** La función imprimirMatriz recorre cada celda de la matriz y las imprime. Al igual que las operaciones anteriores, la complejidad de esta operación es lineal con respecto al tamaño de la matriz, es decir, , donde *n* es el orden de la matriz.

**Multiplicación de matrices:** El método multiplicarMatrices() realiza la multiplicación de dos matrices cuadradas de orden *n*x*n*. En cada iteración del bucle más interno, se realizan dos operaciones básicas. Dado que hay iteraciones en total, el número total de operaciones básicas es . Por lo tanto, la complejidad temporal de este algoritmo de multiplicación de matrices es .

La complejidad temporal del algoritmo en su conjunto es , donde *n* es el orden de la matriz.

# Características del equipo

Procesador Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz 2.50GHz

RAM instalada: 16,0 GB

Sistema Operativo de 64 bits

# Ejecución del programa

Al ejecutar el programa se obtuvo los datos de la Tabla 1 y la gráfica de la Figura 1. que es una función cuadrática que al analizar el código era lo que esperábamos obtener.

Tabla 1. Tiempos de ejecución

|  |  |
| --- | --- |
| Orden de las matrices (n) | Tiempo [ms] |
| 10 | 19,9307 |
| 20 | 51,6238 |
| 30 | 90,8265 |
| 40 | 142,7335 |
| 50 | 206,7018 |
| 60 | 295,6089 |
| 70 | 400,2592 |
| 80 | 516,6074 |
| 90 | 644,4972 |
| 100 | 710,8939 |
| 200 | 2691,5203 |
| 300 | 5720,2971 |
| 400 | 10143,9265 |
| 500 | 15433,8769 |
| 600 | 22554,7286 |
| 700 | 30280,2091 |
| 800 | 39331,0801 |
| 900 | 50030,856 |
| 1000 | 61704,8691 |

Figura 1. Tiempo vs Orden de la Matriz

# Conclusiones

* Al incluir líneas de código para medir el tiempo de ejecución del algoritmo no se afecta a la complejidad algorítmica, solo se proporciona información adicional sobre el rendimiento real de un algoritmo en diferentes condiciones de entrada.
* Se comprobó que la función cuadrática fue obtenida de la suma de dos matrices.
* Se determinó que se puede mejorar el tiempo de ejecución del programa si no se imprimen las matrices 1 y 2 y solo se imprime la matriz resultante de la suma.

# Referencias

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introducción a algorítmos (3rd ed.). MIT Press.

[2] Kleinberg, J., & Tardos, É. (2006). Diseño de Algoritmos. Addison-Wesley.

# Anexo I

/\*\*

\* Copyright (C) 2024, Angel Pastaz, Ruth Rueda

\* (Grupo 11) Suma de Matrices

\* Version 1.0

\*/

import java.util.Scanner;

import java.util.Random;

public class SumaMatriz {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Suma de dos matrices: ");

int ordenMatriz = validarEntero(scanner, ">>Ingrese el orden de las matrices: ");

while(ordenMatriz <= 0){

System.out.println("El orden no debe ser negativo o cero.\n");

ordenMatriz = validarEntero(scanner, ">>Ingrese el orden de las matrices: ");

}

// Se inicia el temporizador

long tiempoInicio = System.nanoTime();

int[][] matriz1 = generarMatrizAleatoria(ordenMatriz);

int[][] matriz2 = generarMatrizAleatoria(ordenMatriz);

System.out.println("Matriz 1:");

imprimirMatriz(matriz1);

System.out.println("Matriz 2:");

imprimirMatriz(matriz2);

int[][] matrizSuma = sumarMatrices(matriz1, matriz2);

// Finaliza el temporizador

long tiempoFinal = System.nanoTime();

// Se imprime la matriz suma

System.out.println("Matriz suma:");

imprimirMatriz(matrizSuma);

// Calcula el tiempo transcurrido

long tiempoTranscurrido = tiempoFinal - tiempoInicio;

// Convierte el tiempo a segundos

double milisegundos = (double) tiempoTranscurrido / 1\_000\_000.0;

System.out.println("Tiempo de ejecución: " + milisegundos + " milisegundos");

scanner.close();

}

public static int validarEntero(Scanner scanner, String mensaje) {

int numero;

while (true) {

System.out.print(mensaje);

if (scanner.hasNextInt()) {

numero = scanner.nextInt();

break;

} else {

System.out.println("Error! Debe ingresar un número entero mayor a cero.\n");

scanner.next();

}

}

return numero;

}

public static int[][] generarMatrizAleatoria(int ordenMatriz) {

Random random = new Random();

int[][] matriz = new int[ordenMatriz][ordenMatriz];

for (int i = 0; i < ordenMatriz; i++) {

for (int j = 0; j < ordenMatriz; j++) {

matriz[i][j] = random.nextInt(100 + 100 + 1) -100;

}

}

return matriz;

}

public static int[][] sumarMatrices(int[][] matriz1, int[][] matriz2) {

int n = matriz1.length;

int[][] resultado = new int[n][n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

resultado[i][j] = matriz1[i][j] + matriz2[i][j];

}

}

return resultado;

}

public static void imprimirMatriz(int[][] matriz) {

int n = matriz.length;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

System.out.print(matriz[i][j] + "\t");

}

System.out.println();

}

}

}