Integrantes:

Joshue Avecillas,

Diego Samaniego

Ciclo:

Segundo Ciclo

Carrera:

Computación

Actividad:

Teoría Complejidad

INFORME RESUMIDO – Teoría de la Complejidad

1. Introducción

La teoría de la complejidad analiza cuántos recursos necesita un algoritmo, principalmente **tiempo de ejecución** y **memoria**. Su objetivo es evaluar qué tan eficiente es un algoritmo y cómo se comporta cuando crece el tamaño de los datos. Esta práctica introduce los conceptos básicos y los ejemplifica con código en Java.

2. Investigación Teórica

2.1 Teoría de la Complejidad

Estudia el rendimiento de los algoritmos según el uso de recursos. Permite comparar soluciones y decidir cuál es más eficiente.

2.2 Eficiencia de algoritmos

- **Coste temporal:** número de operaciones que realiza un algoritmo.
- **Coste espacial:** cantidad de memoria utilizada.

2.3 Factores que afectan el tiempo

- **Propios:** lógica, ciclos, recursión, estructuras de datos.
- **Circunstanciales:** hardware, compilador, sistema operativo.
- **Análisis teórico:** estudio matemático sin ejecutar el algoritmo.
- **Análisis experimental:** pruebas reales midiendo tiempos.

2.4 Notación Big O

Define cómo crece el tiempo de ejecución respecto al tamaño de entrada (*n*).
Incluye:

- **Mejor caso (Ω)**
- **Peor caso (O)**
- **Caso promedio (Θ)**

Ejemplos típicos: O(1), O(n), O(n^2), O(log n), O(n log n).

3. Ejemplos de Complejidad en Java

Cada clase representa una complejidad:

- **O(1):** operaciones constantes, no dependen del tamaño de la entrada.
- **O(n):** un ciclo que recorre los datos.
- **O(n^2):** dos ciclos anidados.
- **O(log n):** divide la entrada a la mitad en cada paso.
- **O(n log n):** combinación de un ciclo lineal con uno logarítmico.

Cada clase contiene un método ejemplo() que muestra su comportamiento.

4. Conclusiones

- La complejidad permite predecir el rendimiento de un algoritmo.
- Las más costosas entre las vistas son $O(n^2)$ porque escalan muy rápido.
- Analizar código y compararlo con la teoría ayuda a entender el impacto del tamaño de entrada.
- Big O es esencial para elegir algoritmos eficientes.

5. Resultados Obtenidos

Se comprendieron los conceptos principales de teoría de la complejidad y se aplicaron en ejemplos prácticos en Java, conectando teoría y práctica.