Parte 1 Identificación de Complejidad Algorítmica

Algoritmo 1 – obtenerElemento

Análisis: La función accede al primer elemento del arreglo usando un índice. Esta operación no depende del tamaño de la lista.

Complejidad temporal: O(1).

Justificación: Siempre ejecuta la misma cantidad de pasos, sin importar si la lista tiene 10, 1000 o un millón de elementos.

Algoritmo 2 – recorrerLista

Análisis: El bucle se ejecuta exactamente n veces. En cada iteración solo hace una operación sencilla (let x = i).

Complejidad temporal: O(n).

Justificación: El tiempo de ejecución crece de forma proporcional a n. Si duplicamos n, el tiempo de ejecución también se duplica aproximadamente.

Algoritmo 3 – recorrerMatriz

Análisis: Tiene dos bucles anidados. El bucle externo se ejecuta n veces, y en cada iteración ejecuta por completo el bucle interno también n veces.

Complejidad temporal: O(n^2).

Justificación: El total de operaciones es n * n = n^2. Si n se duplica, el tiempo de ejecución se multiplica por cuatro.

Parte 2 – Situación problema

Respuestas de reflexión

1. ¿Qué pasos seguiste para decidir la estrategia de resolución?
Primero analizamos qué operaciones necesitaba el sistema: verificar si un estudiante tiene un beneficio especial y calcular las combinaciones posibles de estudiantes en mesas de dos. Luego resolver cada operación de la forma más eficiente posible. Para la verificación del beneficio usamos una comparación directa de una propiedad booleana, y para las combinaciones aplicamos una fórmula matemática (n*(n-1)/2) que evita recorrer toda la lista. Con esto logramos que ambas operaciones fueran rápidas sin importar el tamaño de la lista.

2. ¿Cómo varía el tiempo de ejecución a medida que crece la lista?
El tiempo de ejecución se mantiene prácticamente constante en ambas operaciones, incluso cuando la lista de estudiantes aumenta de tamaño. La única parte que tarda más a medida que

la lista crece es la creación de los datos para hacer las pruebas, pero el algoritmo en sí se comporta en tiempo constante.

3. ¿Cuál crees que es la complejidad temporal de tu algoritmo? ¿Cómo lo justificas?
La verificación del beneficio especial tiene complejidad **O(1)** porque solo consulta un atributo de un objeto. El cálculo de combinaciones también es **O(1)** porque usa una fórmula matemática que depende únicamente del tamaño n de la lista, sin necesidad de recorrerla. Por eso ambos algoritmos son constantes y muy eficientes.

4. ¿Tu algoritmo sería eficiente si la lista tuviera millones de registros? Explica.
Sí, el algoritmo seguiría siendo eficiente con millones de registros, ya que las operaciones principales no dependen de recorrer la lista. Aunque el almacenamiento de tantos estudiantes consume memoria, las funciones que implementé siguen funcionando en tiempo constante, por lo que el desempeño no se vería afectado significativamente.