Análisis del orden de complejidad Big-O

MÉTODO	COMPLEJIDAD	EXPLICACIÓN
guardar_temperatura	O(log n)	Insertar un nuevo nodo en el árbol AVL toma tiempo logarítmico porque el árbol está balanceado (no se vuelve “largo y flaco”).
devolver_temperatura	O(log n)	Buscar una fecha recorre el árbol desde la raíz hasta una hoja, en altura logarítmica.
borrar_temperatura	O(log n)	Eliminar un nodo (una fecha) también tiene costo logarítmico, ya que incluye rebalancear el árbol.
max_temp_rango	O(k + log n)	Se recorren sólo los nodos dentro del rango de fechas (k = cantidad de nodos en ese rango).
min_temp_rango	O(k + log n)	Se recorren sólo los nodos dentro del rango de fechas (k = cantidad de nodos en ese rango).
temp_extremos_rango	O(k + log n)	Llama internamente a los dos anteriores
devolver_temperaturas	O(k + log n)	Igual que arriba, recorre el árbol parcialmente.
cantidad_muestras	O(1)	solo devuelve el valor de 1 variable
cargar_desde_archivo	O(m log n)	inserta m registros uno por uno, cada inserción cuesta O(log n)

Explicación de la solución:

El programa implementa una base de datos de temperaturas utilizando un árbol AVL como estructura principal. Cada nodo del árbol almacena una fecha y su temperatura asociada, permitiendo realizar operaciones de inserción, búsqueda, eliminación y consulta de rangos de manera eficiente.

El código main realiza dos pruebas:

1. Carga manual de datos: inserta cuatro temperaturas y muestra las consultas básicas.

2. Carga desde archivo: lee 120 muestras desde un archivo de texto (muestras.txt) y realiza consultas sobre un rango de fechas determinado.

Conclusiones:

1. Al implementar un árbol AVL aseguramos que las operaciones se realicen de manera más rápida y eficiente.
2. El sistema demostró funcionar correctamente tanto con una pequeña cantidad de datos como con un archivo de 120 registros, mostrando resultados consistentes y ordenados por fecha.