Conclusiones del problema 2: Temperaturas DB.

La clase que implementamos cumple con la especificación lógica solicitada para el TAD Temperaturas_DB: debe almacenar mediciones de temperatura (temperatura float en ºC y fecha ingresada como "dd/mm/aaaa" y normalizada a datetime.date) y permitir consultas eficientes sobre rangos de fechas. Para garantizar eficiencia y crecimiento dinámico de los datos, la estructura interna elegida es un árbol AVL equilibrado que indexa por fecha.
Implementación y decisiones clave

- Estructura principal: árbol AVL donde cada nodo contiene clave = fecha (datetime.date) y valor = temperatura (float).
- Campos por nodo: altura, size (tamaño del subárbol) y temperatura. Estos campos se actualizan en cada inserción/borrado para mantener invariantes y permitir consultas en O(1) donde corresponde.
- Normalización de fechas: todas las entradas “dd/mm/aaaa” se convierten a objetos datetime.date mediante una función convertir_fecha que opera en tiempo constante.
- Operaciones implementadas: guardar_temperatura, devolver_temperatura, borrar_temperatura, devolver_temperaturas (rango ordenado), min_temp_rango, max_temp_rango, temp_extremos_rango, cantidad muestras.
- Para la lectura del archivo se utilizó el método open(), recorriendo cada línea para separar la fecha y la temperatura, y luego se llamó al método guardar_temperatura() para almacenar los datos en el árbol de la base de datos.

La siguiente tabla resume las complejidades Big-O de los métodos expuestos por Temperaturas_DB y su justificación breve:

Método	Orden de complejidad Big-O	Justificación
convertir_fecha	O(1)	Al no haber bucles ni llamadas recursivas, y solo convertir una cadena de texto fijo (de largo constante, ej. "15/03/2025") a un objeto datetime.date, la operación convertir fecha siempre va a tardar lo mismo, es decir es constante.
guardar_temperatura	O (log n)	Convierte la fecha en tiempo constante y luego inserta el dato en un árbol AVL. Esta estructura balanceada mantiene su altura proporcional a log n, por lo que la inserción requiere recorrer solo una

		rama del árbol.
devolver_temperatura	O (log n)	Convierte la fecha en tiempo constante y busca la clave en el árbol AVL. Al estar balanceado, la búsqueda implica recorrer a lo sumo una rama de longitud log n.
max_temp_rango	O(n)	Recorre todos los nodos del árbol para verificar cuáles están dentro del rango de fechas. En el peor caso (si el rango abarca casi todos los datos), visita los n nodos.
min_temp_rango	O(n)	Igual que max_temp_rango: recorre el árbol completo y compara los valores, con costo lineal respecto del número de nodos.
temp_extremos_rango	O(n)	Recorre el árbol una sola vez y calcula simultáneamente los valores mínimo y máximo. Aunque combina las operaciones de los métodos anteriores, sigue siendo un recorrido lineal .
borrar_temperatura	O (log n)	Convierte la fecha en tiempo constante y luego utiliza la función eliminar correspondiente en el árbol AVL. Al estar balanceado, el árbol solo recorre una rama de longitud log n
devolver_temperaturas	O(n)	Recorre todos los nodos del árbol para filtrar por rango y formatear las fechas. En el peor caso (si el rango abarca casi todo el árbol), eso implica visitar todos los nodos.
cantidad_muestras (si la mencionás)	O(1)	Devuelve un atributo ya almacenado (self.arbol.tamano), por lo que no realiza ningún recorrido.

Conclusiones

Se logró cumplir con las implementaciones solicitadas y superar los tests funcionales. Se verificó el orden de complejidad de cada método mediante análisis teórico y observación

experimental: las operaciones puntuales presentan un comportamiento O(log n), mientras que las consultas de rango se describen como O(log n + k), con peor caso O(n).

Como limitación, las mediciones experimentales pueden verse afectadas por ruido asociado a constantes de implementación o al entorno de ejecución; por ello, se corroboraron los resultados mediante inspección del código, analizando la presencia de bucles, recorridos y operaciones recursivas, confirmando así el orden de complejidad teórico.

Finalmente, para optimizar las consultas frecuentes de valores extremos (mínimo y máximo), podría considerarse mantener dichos agregados actualizados en cada subárbol del AVL, equilibrando el costo adicional en inserciones y eliminaciones con la mejora obtenida en las consultas.