***Presentado por : Tomas Quintero, Juan Esteban Ruiz:***

Algoritmos :

1. Método addTask():

- Complejidad Temporal:

- La función realiza operaciones de inserción en tres estructuras de datos: `hashTable`, `priorityQueue`, y `deadlineHeap`. La inserción en estructuras de datos bien implementadas, como un hash table y un heap binario, es generalmente O(1) en promedio.

- La operación switch tiene una complejidad constante O(1).

- La complejidad temporal es dominada por las operaciones de inserción, y dado que son O(1) en promedio, la complejidad total es O(1).

- Complejidad Espacial:

- La función crea un objeto `TaskReminder` y lo almacena en tres estructuras de datos. Sin embargo, solo se almacenan referencias a objetos, no los objetos en sí. Entonces, la complejidad espacial es O(1).

- Las estructuras de datos (`hashTable`, `priorityQueue`, `deadlineHeap`, `actionStack`) ocupan una cantidad constante de espacio adicional, independientemente del número de tareas. La complejidad espacial es O(1).

2. Método `showTasksByPriority`:

- Complejidad Temporal:

- La función utiliza un bucle `for` para iterar sobre `priorityQueue`. La operación de iteración es O(N), donde N es el número de tareas.

- Dentro del bucle, las operaciones de concatenación de cadenas y la impresión en consola son O(1) por cada iteración.

- La complejidad temporal total es dominada por el bucle de iteración, siendo O(N).

- Complejidad Espacial:

- La función utiliza un StringBuilder (`msg`) para construir el mensaje, pero su tamaño no depende de N, por lo que es O(1).

- Las variables locales y otras estructuras de datos tienen una complejidad constante.

- La complejidad espacial es O(1).

Análisis y conclusiones:

- La función demuestra una eficiencia temporal destacada al realizar operaciones de inserción en tres estructuras de datos clave: `hashTable`, `priorityQueue`, y `deadlineHeap`. La inserción en estas estructuras es O(1) en promedio, gracias a implementaciones eficientes.

- El uso de un switch con constantes contribuye a una complejidad constante O(1).

- La complejidad temporal total está dominada por las operaciones de inserción, y dado que son O(1) en promedio, la complejidad total es O(1).

- A nivel espacial, la función maneja eficientemente la creación de un nuevo objeto `TaskReminder`. Al almacenar referencias en lugar de duplicar datos, la complejidad espacial es O(1).

- Las estructuras de datos adicionales (`hashTable`, `priorityQueue`, `deadlineHeap`, `actionStack`) mantienen una complejidad espacial constante, independientemente del número de tareas. La complejidad espacial total es O(1).

2. Método `showTasksByPriority`:

- La eficiencia temporal se mantiene sólida a pesar de la operación de iteración sobre `priorityQueue`. El bucle `for` introduce una complejidad O(N), donde N es el número de tareas.

- Operaciones dentro del bucle, como la concatenación de cadenas e impresión en consola, son O(1) por cada iteración.

- La complejidad temporal total está dominada por el bucle de iteración, siendo O(N).

- Complejidad Espacial:

- La función construye el mensaje utilizando un StringBuilder (`msg`), pero su tamaño no depende de N, lo que lleva a una complejidad espacial constante, O(1).

- Variables locales y otras estructuras de datos tienen una complejidad espacial constante.

- La complejidad espacial total es O(1).

En resumen, la clase Controller presenta algoritmos eficientes y bien diseñados. Los análisis de complejidad respaldan un rendimiento constante y escalable, crucial para un sistema de gestión de tareas robusto.