Diseño y análisis de algoritmos

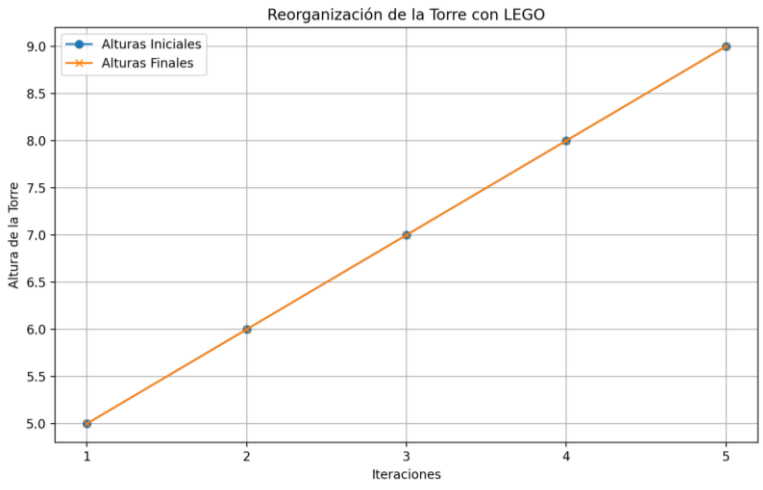
Proyecto 1

Catalina Velez – 202123920

1. Explicación de la solución:

El algoritmo comienza inicializando una instancia de la clase SolucionProyecto1 con la torre dada como entrada. Esto incluye almacenar la configuración de la torre, su tamaño, un contador de movimientos y una lista para registrar inversiones. El algoritmo itera sobre cada bloque de la torre, comenzando desde el segundo bloque hasta el penúltimo. Durante cada iteración, compara la altura del bloque actual con la del bloque anterior. Si la altura del bloque actual es mayor que la del bloque anterior, el algoritmo considera ajustar los bloques adyacentes para igualar las alturas y reducir la cantidad de movimientos necesarios. Se aplican diferentes condiciones para realizar los ajustes, donde se intenta minimizar la diferencia de altura entre los bloques adyacentes, redistribuyendo los bloques si es necesario. Si no es posible igualar las alturas de los bloques adyacentes sin aumentar la altura del bloque actual, se ajustan los bloques adyacentes para reducir la diferencia de altura al máximo posible.

Una vez determinados los ajustes necesarios, el algoritmo los ejecuta. Esto implica redistribuir los bloques de manera que se cumplan las condiciones establecidas, actualizando la configuración de la torre y registrando los movimientos realizados. Finalmente, cuando es completada la iteración sobre toda la torre, el algoritmo finaliza y devuelve la configuración final de la torre, así como el número total de movimientos realizados.



La gráfica generada muestra la evolución de la altura de una torre de bloques de lego a lo largo de varias iteraciones del algoritmo implementado. Aquí hay una explicación detallada de cada componente de la gráfica. El eje X representa el número de iteraciones del algoritmo. Cada punto en el eje X corresponde a una iteración del algoritmo, comenzando desde la primera iteración y avanzando hacia la última. Por otro lado, el eje Y representa la altura de la torre de bloques de lego. Cada punto en el eje Y representa la altura de la torre en una iteración específica del algoritmo. La altura se mide en el número de bloques de lego apilados verticalmente.

2. **Análisis de complejidades espacial y temporal:**

    a. Complejidad espacial:

El espacio utilizado por la instancia de la clase es proporcional al tamaño de la torre, ya que se almacena una lista para representar la torre y otra lista para registrar inversiones. Por lo tanto, la complejidad espacial es O(n), donde n es el tamaño de la torre. No hay estructuras de datos adicionales que aumenten el uso de memoria de manera significativa, por lo que la complejidad espacial se mantiene en O(n).

    b. Complejidad temporal:

La complejidad temporal del algoritmo está determinada principalmente por el bucle que itera sobre cada bloque de la torre. Dentro de este bucle, se realizan operaciones como comparaciones de altura, cálculos de diferencia de altura, ajustes de bloques y actualizaciones de contadores. Estas operaciones son de tiempo constante O(1), ya que no dependen del tamaño de la torre. Sin embargo, el bucle puede iterar hasta n-1 veces, donde n es el tamaño de la torre. En el peor caso, el algoritmo puede requerir una cantidad significativa de iteraciones para reorganizar completamente la torre. Dicho esto, la complejidad temporal del algoritmo está en el orden de O(n), donde n es el tamaño de la torre.

3. **Conclusión:**

A pesar de que el algoritmo implementado no es basado en una solución de programación dinámica, pues no vemos la estructura típica de un algoritmo de programación dinámica, como la definición de subproblemas, la recurrencia y la memorización de resultados. La solución implementada se basa en iterar sobre la torre y realizar ajustes en función de ciertas condiciones específicas, donde se solucionan los casos establecidos por el enunciado del proyecto efectivamente y en con una complejidad temporal y espacial buena de O(n)