Luis Fernando Villegas Mendoza - 421041345
Estructuras Discretas
Ciencias de la computación
Práctica 01

Pruebas de tiempo para práctica 01

findFirstAndLast		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
[1,4,2,1,6,2,9], 2	0	0
[4,2,7,5,4,3,7,2,5,3,4,1], 15	0	0
[3,2,1,4,2], 1	0	0

La diferencia de tiempo fue imperceptible en milisegundos, sin embargo, al analizar el algoritmo anterior y el que generé para la práctica y lo que particularmente hace que mejora la complejidad en tiempo es que en el ejemplo original en el peor de los casos que sería cuando el número no se encuentra en el arreglo el programa tendría que recorrer todos los elementos del arreglo dos veces, primero de izquierda a derecha, luego de derecha a izquierda.

En mi algoritmo lo recorrerá revisando simultáneamente de izquierda a derecha y de derecha a izquierda, salvando valores posibles en ambos lados, terminando en una cantidad de iteraciones igual a la mitad de la cantidad de los elementos, pudiendo ahorrarse la mitad o hasta tres cuartos del tiempo del original.

isSudokuValid		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
ejemplo2a	0	0
ejemplo2b	0	0

La diferencia de tiempo fue imperceptible en milisegundos, sin embargo, al analizar el algoritmo anterior y el que generé para la práctica y lo que particularmente hace que mejora la complejidad en tiempo es que en el ejemplo original debe hacer iteraciones por cada valor posible por cada fila por cada columna.

En mi algoritmo revisará cada una fila y columna a la vez, además de hacer uso de memoria y verificaciones que pueden descartar más prontamente un sudoku inválido más prontamente, como un valor fuera del rango o un valor duplicado. Incluso para ahorrar una iteración más, se añadió un contador de verificado, ya que si se revisan duplicados, solamente debería haber tantos casos correctos de aparición de número como cantidad de elementos de la fila o columna.

Luis Fernando Villegas Mendoza - 421041345
Estructuras Discretas
Ciencias de la computación
Práctica 01

rotateArray		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
[1,4,2,1,6,2,9], 2	0	0
[4,2,7,5,4,3,7,2,5,3,4,1], 15	0	0
[3,2,1,4,2], 1	0	0

La diferencia de tiempo fue imperceptible en milisegundos, sin embargo, para este caso realicé la prueba aumentando de manera enorme la cantidad de movimientos a realizar y pude corroborar la enorme diferencia, cuando el primero algoritmo realizaba el proceso en 209 ms mi algoritmo no tomaba ni 1 ms.

La diferencia radica principalmente en que el algoritmo de ejemplo tiene que realizar iteraciones de la cantidad de elementos del arreglo por la cantidad de movimientos, por lo cual tanto al aumentar el tamaño del arreglo o de los movimientos se estaría aumentando el tiempo de ejecución. Mi algoritmo limita la cantidad de movimientos a la cantidad de elementos existentes, debido a que cada vuelta completa es irrelevante, ya que posicionará a los elementos de la misma forma. La siguiente gran diferencia es que el ejemplo realiza un solo movimiento a la vez, mi algoritmo realiza el movimiento total en un solo cambio.