Universidad Nacional Autónoma de México
Facultad de Ciencias

Estructura de Datos 2022-1

Práctica 01:
Complejidad Computacional

Rodríguez García Ulises
318042202
Uribe García Zurisadai
318223197

Octubre 01, 2021

Tablas con los resultados obtenidos a partir de la comparación del algoritmo propuesto y el algoritmo optimizado.

mergeSortedArray(int[], int, int[],int)		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
ArrayA1.txt, 500, ArrayA2.txt, 700	5	0
ArrayB1.txt, 2000, ArrayB2.txt, 3500	2	1
ArrayC1.txt, 4000, ArrayC2.txt, 4000	30	1
ArrayD1.txt, 7000, ArrayD2.txt, 8000	19	1
ArrayE1.txt, 15000, ArrayE2.txt, 19000	98	2
ArrayF1.txt, 30000, ArrayAF.txt, 25000	356	1

El algoritmo que diseñamos mejora la complejidad, ya que en lugar de usar dos for con uno de ellos anidado para hacer el ordenamiento, nosotros usamos solo uno con el cual comparamos ambos arreglos asignamos uno por uno a un nuevo arreglo creado a partir de los dos anteriores.

isValidBoard(int[][])		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
BoardA.txt	81	0
BoardB.txt	106	2
BoardC.txt	224 634	47
BoardD.txt	8013	0
BoardE.txt	246 922	166
BoardF.txt	*No cargó*	227

Para hacer que este método tuviera complejidad $O(n^2)$, creamos un arreglo temporal y usamos dos for anidados en donde fuimos contando la cantidad de veces que se repetía cada número en filas y columnas, usamos otro for para recorrer el arreglo temporal y observar las veces que este se repite.

rotateArray(int[], int)		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
ArrayA1.txt, 500	5	0
ArrayB1.txt, 1000	22	0
ArrayC1.txt, 2000	33	1
ArrayD1.txt, 3000	60	1
ArrayE1.txt, 10000	290	2
ArrayAF.txt, 20000	831	1

Para optimizar el último algoritmo en vez de usar dos for anidados para rotar el arreglo, usamos solo uno con el cual primero se copia el arreglo en uno auxiliar y después asigna el valor al arreglo inicial en la posición rotada.