mergeSortedArray(int[], int, int[], int)		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
ArrayA1.txt, 500, ArrayA2.txt, 700	8 milisegundos	0 milisegundos
ArrayB1.txt, 2000, ArrayB2.txt, 3500	50 milisegundos	1 milisegundos
ArrayC1.txt, 4000, ArrayC2.txt, 4000	218 milisegundos	1 milisegundos
ArrayD1.txt, 7000, ArrayD2.txt, 8000	234 milisegundos	2 milisegundos
ArrayE1.txt, 15000, ArrayE2.txt, 19000	727 milisegundos	6 milisegundos
ArrayF1.txt, 30000, ArrayF2.txt, 25000	2678 milisegundos	4 milisegundos

Al usar este metodo se reduce la complejidad y esta se vuelve lineal O(n+m) pues en lugar de primero copiar cada arreglo por separado y luego recorrer el areglo result dos veces para hacer las comparaciones, ahora se van recorriendo los dos al mismo tiempo, y mientras se copian al arreglo result vamos comparando aprovechando el hecho de que los dos arreglos originales ya estan ordenados.

isValidBoard(int[][])		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
BoardA.txt	46 milisegundos	2 milisegundos
BoardB.txt	61 milisegundos	4 milisegundos
BoardC.txt	80631 milisegundos	158 milisegundos
BoardD.txt	1472 milisegundos	143 milisegundos
BoardE.txt	1210337 milisegundos	853 milisegundos
BoardF.txt	2594062 milisegundos	905 milisegundos

El algoritmo 2 mejora la complejidad ya que tiene una complejidad de O(n^2) y el anterior O(n^3) , pues este algoritmo depende de dos n anidadas , pues lo que hace este método es crear 3 arreglos unidimensionales; para las verticales , horizontales y un arreglo de tamaño de las horizontales y verticales para ir actualizando los contadores de las variables , el anterior algoritmo ocupaba un for de mas pues utilizaba un for innecesario para comparar los elementos.

rotateArray(int[], int)		
Entradas	Milisegundos algoritmo 1	Milisegundos algoritmo 2
ArrayA1.txt, 500	0 milisegundos	0 milisegundos
ArrayB1.txt, 1000	0 milisegundos	2 milisegundos
ArrayC1.txt, 2000	2 milisegundos	1 milisegundo
ArrayD1.txt, 3000	4 milisegundos	1 milisegundo
ArrayE1.txt, 10000	0 milisegundos	3 milisegundos
ArrayF1.txt, 20000	2 milisegundos	16 milisegundos

El segundo algoritmo es mejor , el primero tiene complejidad O(n*m) y este O(n) pues la m era otro for para la cantidad de desplazamientos, en cambio en este método se utiliza únicamente un for que depende de n y nos apoyamos del concepto de modulo para ir haciendo las alteraciones correspondientes.