MÉTODOS DE ORDENAMIENTO

Estructura de Datos Avanzados

18 DE SEMPTIEMBRE DEL 2019

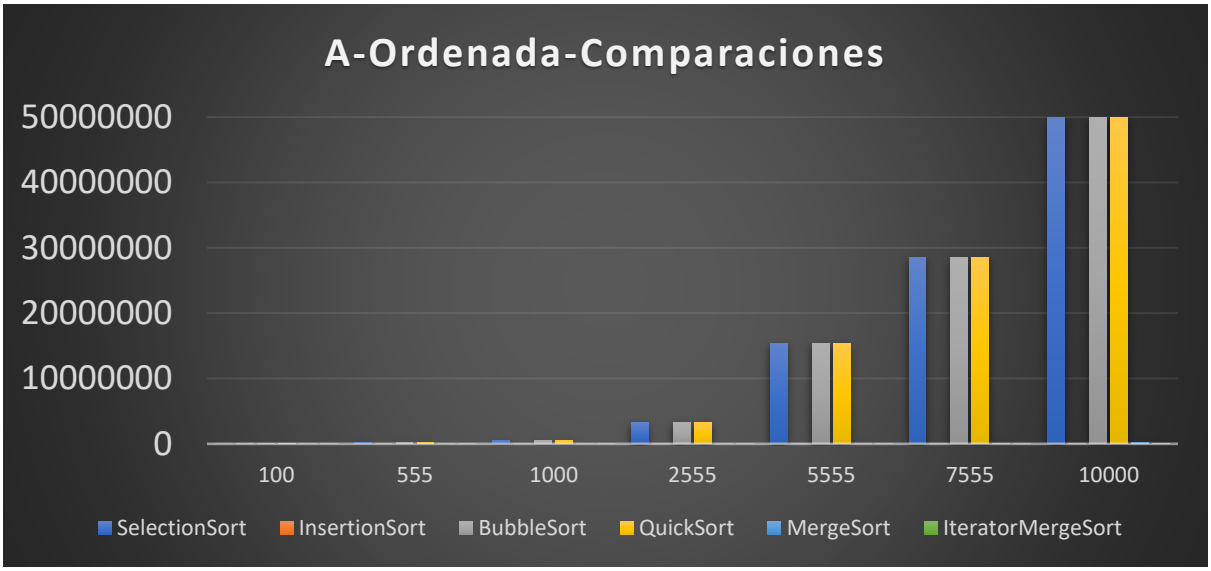
INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO
Diana Espinosa Ruiz

Introducción

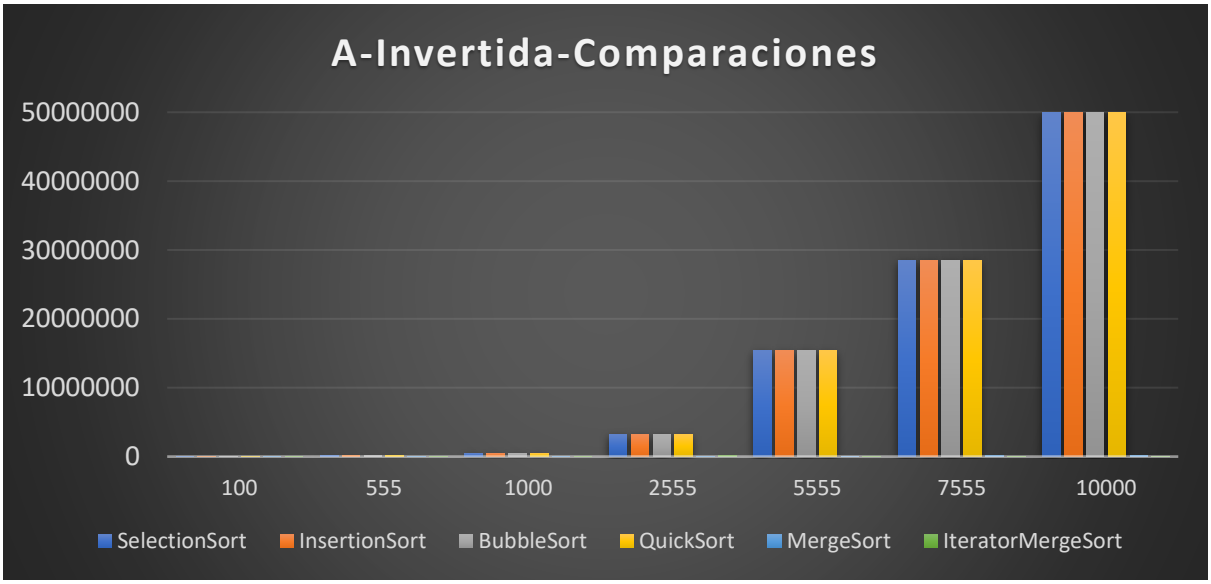
La tarea consistió en leer el archivo busines10k.json y ordenar la información como se pidiera. Para ordenar la información se utilizaron los siguientes métodos de ordenamiento: selection sort, insertion sort, bubble sort, quick sort, merge sort y iterator merge sort.

Las n que elegí para determinar de cuantos elementos de tipo business iban a ser mis arreglos son: 100,555,1000,2555,5555,7555,10000.

a) Ordenar un arreglo ordenado.



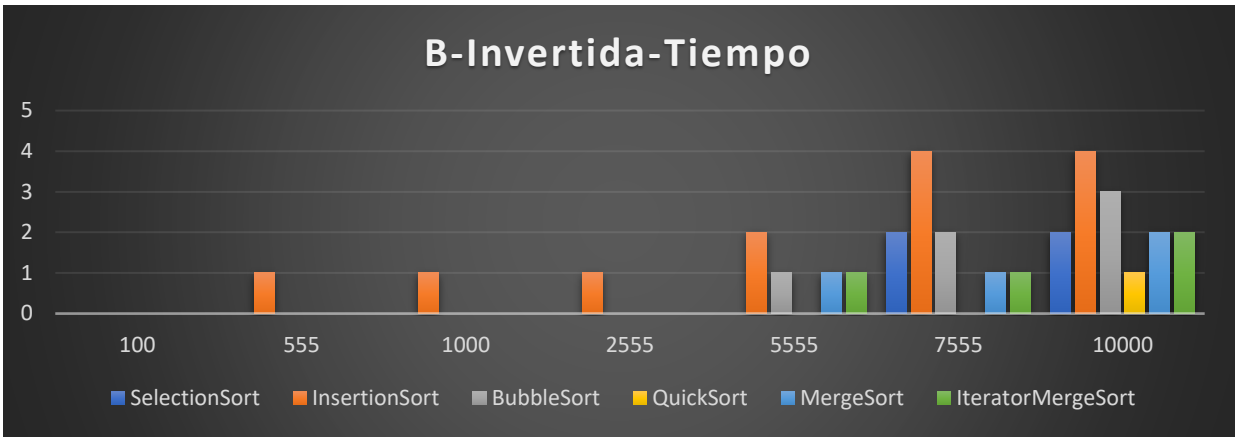
La gráfica A-Ordenada-Comparaciones muestra la cantidad de comparaciones que tuvo que hacer cada método para ordenar el arreglo ordenado. Nos podemos dar cuenta que Selection sort y Bubble sort son los menos eficientes al realizar esta función, pues son de n^2. Al mismo tiempo podemos darnos cuenta que IteratorMergeSort y Merge sort son de los más eficientes porque es de $n*\log_2(n)$; no



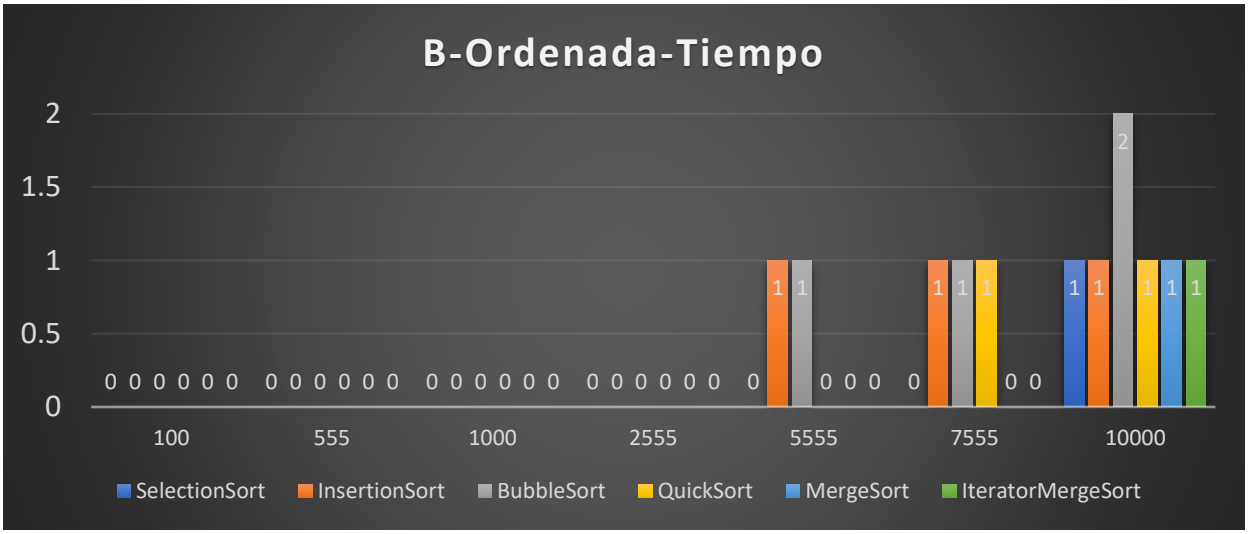
obstante, hay que resaltar que cuando el arreglo está ordenado Insertion sort es el método que hace menos comparaciones.

En la gráfica B-Ordenada-Tiempo podemos ver que con valores de n menores a 2555 es indiferente que método se utiliza ya que todos se resuelven en menos de 1 segundo. Pero si nos vamos a 10,000 elementos, Bubble sort se vuelve muy ineficiente ya que mientas los demás hacen un aproximado de 1 segundo, Bubble sort se tarda un aproximado de 2 segundos. Observando que entre mayores sean las cantidades, menos eficientes se van volviendo los métodos (por el momento considero estado ordenado).

b) Ordenar un arreglo invertido.



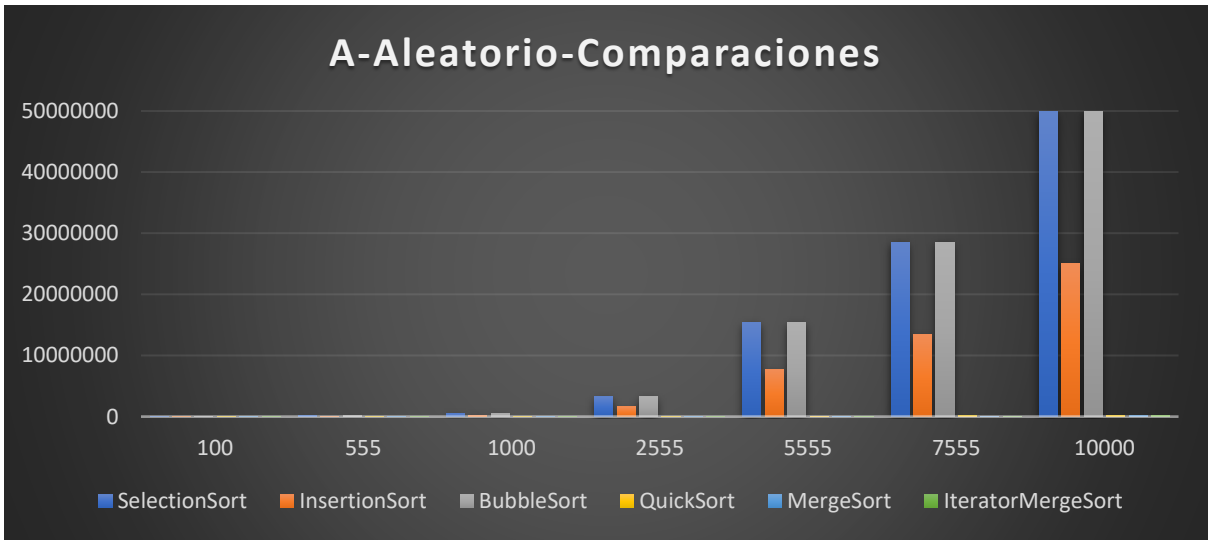
Cuando se habla de un arreglo invertido se habla del peor de los casos, porque el arreglo esta ordenado de mayor a menor en vez de menor a mayor y hay que voltearlo totalmente. Podemos darnos cuenta con la gráfica A-Invertida-Comparaciones que la mayoría de los métodos tardan n^2 en lograr invertir el arreglo. Podemos resaltar que Merge sort e IteratorMerge sort al igual que si estuvieron ordenados tienen O(n log n).



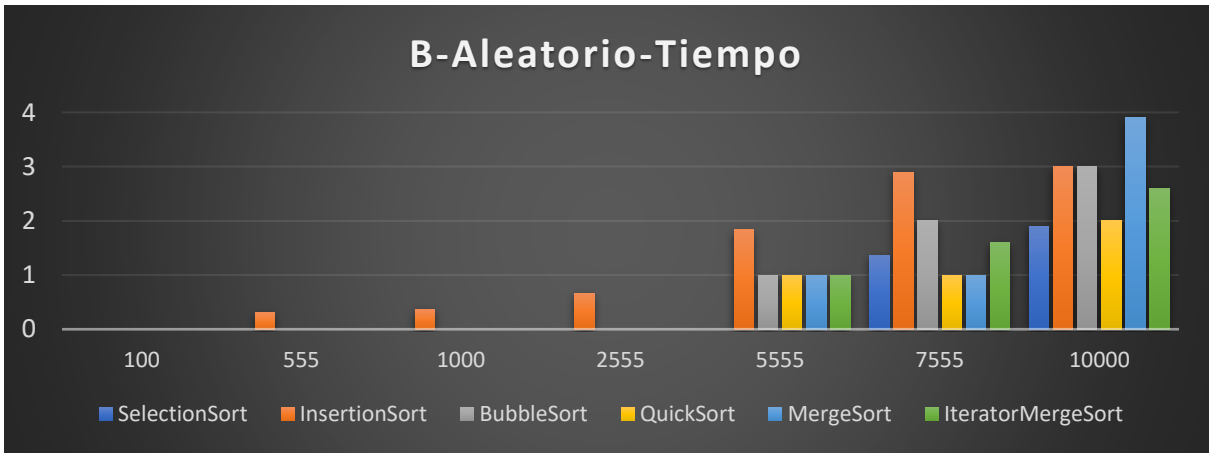
Con la gráfica B-Invertida-Tiempo podemos darnos cuenta que Insertion sort es un método que desde cantidades relativamente pequeñas empieza a tardan considerablemente más que los otros métodos en ordenar un arreglo invertido. Aquí es necesario resaltar que Quick sort es el método que menos tiempo tarda en ordenar el arreglo, por lo que es muy eficiente en cuento al tiempo.

c) Ordenar 30 arreglos aleatorios y de ahí sacar el promedio.

Para poder tener una mejor perspectiva de como funciona el método en casos no triviales (ordenado e invertido) lo mejor fue hacer 30 casos de cada n y de cada método para sacar un promedio de las comparaciones y el tiempo y tener una mejor aproximación de su eficiencia.



En la gráfica A-Aleatorio-Comparaciones podemos darnos cuenta que Quick sort, Merge sort y IteratorMerge sort son los mejores métodos para ordenar cuando el arreglo está desordenado. Sus comparaciones son considerablemente menos a las otras. Selection sort, Bubble sort e Insertion sort son de $O(n^2)$ mientras que Quick sort, Merge sort e IteratorMerge sort son de $O(n*\log_2(n))$ mostrando que, el menos en comparaciones, son las más eficientes



Con la gráfica B-Aleatorio-Tiempo podemos observar que de nuevo Insertion sort resulta ser la menos eficiente en cuanto al tiempo. Además, si observamos con cuidado podemos ir observando que al ir creciendo n el tiempo que se tarda en realizar cada método va variando y va resaltando poco a poco que el Quick sort es el más rápido de todos.

Se puede observar entonces que el método tanto más eficiente en comparaciones como en tiempo es el Quick sort, siguiéndole Merge sort e IteratorMerge sort y como menos eficientes estarían Bubble sort, Insertion sort y Selection sort.

Tablas de comparaciones y tiempo

Ordenada-Comparaciones						
	SelectionSort	InsertionSort	BubbleSort	QuickSort	MergeSort	IteratorMergeSort
100	4950	99	4950	4950	672	474
555	153735	554	153181	153735	5081	3510
1000	499500	999	498501	499500	9976	6053
2555	3262735	2554	3260181	3262735	29119	18164
5555	15426235	5554	15420681	15426235	69578	42394
7555	28535235	7554	28527681	28535235	97578	57547
10000	49995000	9999	49985001	49995000	133616	81717

Ordenada-Tiempo						
	SelectionSort	InsertionSort	BubbleSort	QuickSort	MergeSort	IteratorMergeSort
100	0	0	0	0	0	0
555	0	0	0	0	0	0
1000	0	0	0	0	0	0
2555	0	0	0	0	0	0
5555	0	1	1	0	0	0
7555	0	1	1	1	0	0
10000	1	1	2	1	1	1

Invertida-Comparaciones						
	SelectionSort	InsertionSort	BubbleSort	QuickSort	MergeSort	IteratorMergeSort
100	4950	4950	4950	4950	672	418
555	153735	153735	153735	153735	5081	3014
1000	499500	499500	499500	499500	9976	5933
2555	3262735	3262735	3262735	3262735	29119	95764
5555	15426235	15426235	15426235	15426235	69578	38987
7555	28535235	28535235	28535235	28535235	97578	55385
10000	49995000	49995000	49995000	49995000	133616	74613

Invertida-Tiempo						
	SelectionSort	InsertionSort	BubbleSort	QuickSort	MergeSort	IteratorMergeSort
100	0	0	0	0	0	0
555	0	1	0	0	0	0
1000	0	1	0	0	0	0
2555	0	1	0	0	0	0
5555	0	2	1	0	1	1
7555	2	4	2	0	1	1
10000	2	4	3	1	2	2

Aleatoria-Comparaciones						
	SelectionSort	InsertionSort	BubbleSort	QuickSort	MergeSort	IteratorMergeSort
100	4950	2559.133333	4950	665.2	672	658.3333333
555	153735	77878.1	153735	5456	5081	5246.6
1000	499500	243912.4	499500	11151.1	9976	9717.266667
2555	3262735	1640758.2	3262735	32870.83333	29119	28965.13333
5555	15426235	7699489.6	15426235	80851.4	69578	68804.16667
7555	28535235	13439626.53	28535235	112772.8	97578	95770.96667
10000	49995000	24998443.13	49995000	150735.0667	133616	173648.2333

Aleatoria-Tiempo						
	SelectionSort	InsertionSort	BubbleSort	QuickSort	MergeSort	IteratorMergeSort
100	0	0	0	0	0	0
555	0	0.3	0	0	0	0
1000	0	0.36666667	0	0	0	0
2555	0	0.66666667	0	0	0	0
5555	0	1.83333333	1	1	1	1
7555	1.36666667	2.9	2	1	1	1.6
10000	1.9	3	3	2	3.9	2.6