1. Implemente en el lenguaje de su preferencia tres de los ejercicios 2.15 al 2.18 del libro de Dasgupta. Demuestre en su informe que sus algoritmos tienen la complejidad solicitada.
2. Implemente los algoritmos burbuja, mergesort y quicksort. Para todos los algoritmos, estudie sus tiempos de ejecución para instancias de tamaños variados de la manera que lo hicimos en la tarea anterior. Incluya en su informe la solución del problema 2.24 del libro de Dasgupta.

Los tiempos de ejecución se midieron con tamaños de instancias que son potencia de 10 (10,100, 1000, …) y en la medición del tiempo de cada instancia, se registraron 5 tiempos con los cuales se obtiene un tiempo promedio del tiempo de ejecución por cada instancia en los distintos algoritmos de ordenamiento (Ver Anexo Excel). A continuación, se muestra algunos de los tiempos registrados:

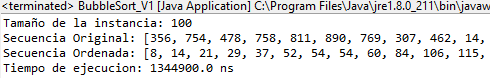
* 1. BubbleSort

Complejidad temporal: En cada iteración, se realiza n-1 comparaciones (siendo n el número de elementos de la secuencia) por lo que tenemos:

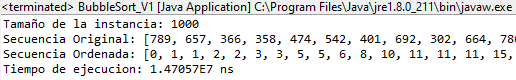
= (n-1) +(n-2) +…+1

Complejidad espacial: Como se trata de un algoritmo in situ, no hace uso de espacio auxiliar considerable por lo que su complejidad es O (1).

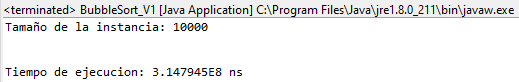
* + 1. Instancia tamaño 100



* + 1. Instancia tamaño 1000



* + 1. Instancia tamaño 10000



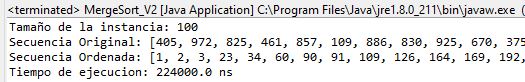
* 1. MergeSort

Complejidad temporal: Por teorema maestro, tenemos que la complejidad temporal para mergeSort es:

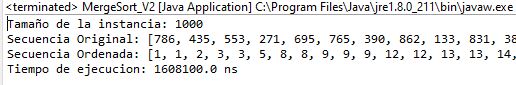
donde corresponde al llamado recursivo para los dos sub-arreglos y *O(n)* para merge.

Complejidad espacial: Como se trata de un algoritmo no *in situ,* estamos creado secuencias o arreglos temporales en cada llamado recursivo (una secuencia para la parte izquierda ‘l’ y otro para la derecha ‘r’) por lo que sería *O(l+r) = O(max(l,r)) = O(n).*

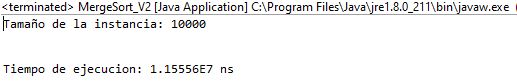
* + 1. Instancia tamaño 100



* + 1. Instancia tamaño 1000



* + 1. Instancia tamaño 10000



* 1. QuickSort

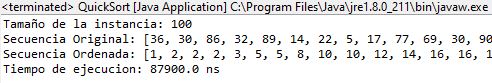
Complejidad temporal: Si el pivote resulta estar ubicado cerca del principio o final del arreglo, podemos considerar:

El mejor caso depende si el pivote es ubicado en la mitad de la secuencia o cerca de esta. Si es asi, su complejidad seria:

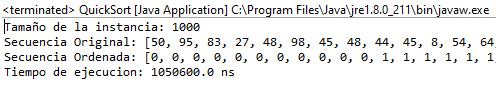
donde corresponde al llamado recursivo para los dos sub-arreglos y *O(n)* para partition.

Complejidad espacial: Debido a la llamada recursiva que realiza el algoritmo sobre sí mismo, el espacio requerido para ejecutarse es O (log n).

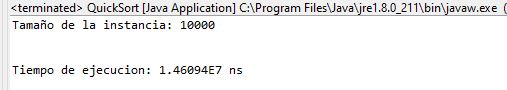
* + 1. Instancia tamaño 100



* + 1. Instancia tamaño 1000



* + 1. Instancia tamaño 10000



1. Implemente dos de los algoritmos de ordenamiento en tiempo O(n) vistos en clase. Mida su tiempo de ejecución con las mismas instancias usadas en el punto anterior. (Note que las instancias de ejemplo deben servir para todos los algoritmos).
   1. CountingSort

Complejidad temporal:

Complejidad espacial:

* 1. BucketSort

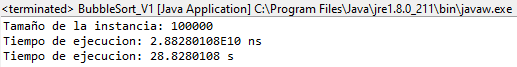
Complejidad temporal: Debido a que su complejidad temporal depende el algoritmo de ordenamiento utilizado sobre cada casillero, se puede decir que su complejidad es O(n2) la cual es dada por el algoritmo InsertionSort para el peor de los casos (un arreglo ordenado al revés).

Complejidad espacial: Como se trata de un algoritmo in situ, el espacio utilizado es O(n+k) donde *k* corresponde al número de casilleros y *n* es el número de elementos.

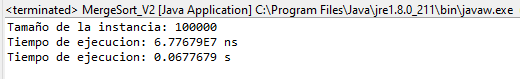
1. Para un tamaño de entrada grande, prediga el tiempo de ejecución de los cinco algoritmos. Mida y registre.

Como se describe en el numeral dos se registraron distintos tiempos para cada instancia de tamaño diferente (ver Anexo Excel) y estos tiempos se promediaron. Con estos tiempos, se realizó una gráfica en la cual realizamos un análisis de regresión y así poder obtener la ecuación de la gráfica para poder determinar el valor del tiempo para una instancia de tamaño mas grande (100000 para estos casos)

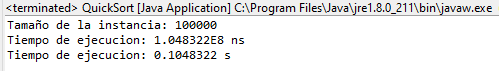
* 1. BubbleSort



* 1. MergeSort



* 1. QuickSort



* 1. CountingSort
  2. BucketSort



1. Conclusiones.