Comparacion de Algoritmos de Ordenamiento

Bruno S. Lopez Fernandez
Estructura de datos avanzadas

February 22, 2018

Introduccion Se realizo esta tarea utilizando los metodos de ordenamiento vistos en clase, para cada uno de los siguientes incisos se entrega las graficas requeridas y se anexan conclusiones y comentarios.

Instrucciones Se pidio que se realizaran los 5 metodos de ordenamiengo vistos en clase:

a Selection Sort

b Insertion Sort

c Bubble Sort

d Merge Sort

e Quick Sort

Para cada uno de estos metodos se correrian los algoritmos de ordenamiento con listas de tamaos distintos, con una seleccion de mas de 5 tamaos.
El objetivo es aprender empiricamente las diferencias de estos algoritmos, sobre todo en terminos de comparaciones realizadas y tiempo transcurrido.

Procedimiento Primero se programaron los algoritmos de ordenamiento en python, en un archivo llamado *Sorting.py*. Despues de esto se creo la clase *Movie.py* para crear objetos de clase movie y despues poder hacer una lista de estos objetos a partir del archivo *"movies_titles2.txt"* que tenia entradas de 17770 peliculas, en las que se dividia por id, nombre y ao de la pelicula.
Se decidio tomar como factor de comparacion entre las peliculas al elemento id, pues era de tipo entero y facilitaria un poco el trabajo. Una vez teniendo esto se creo la clase *main.py* con la cual se llamaba a los algoritmos de ordenamiento y a los objetos de clase Movie, en esta se leia el archivo .txt y se definio una funcion en la cual se realizaran los cinco metodos de ordenamiento y se graficaran los tiempos que utilizaban y la cantidad de comparaciones realizadas. A su vez, esta funcion fue llamada tres veces distintas, primero para un arreglo previamente ordenado, luego para un arreglo ordenado de manera inversa y por ultimo para un arreglo ordenado de manera aleatoria.

Resultados A continuacion se muestran las graficas que se obtuvieron:

Datos Ordenados

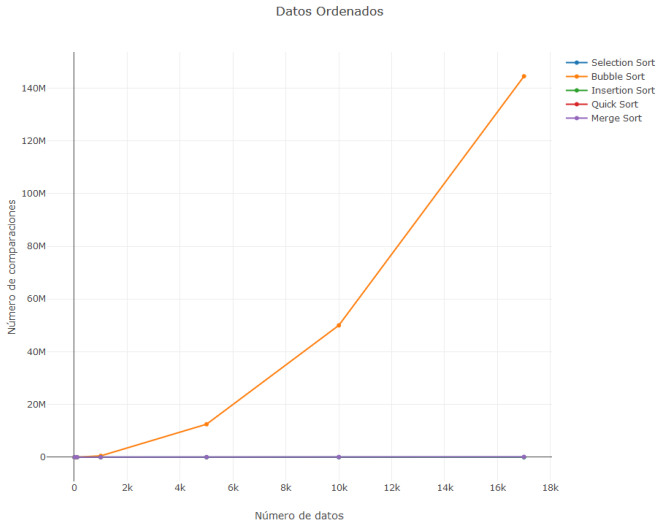

Figure 1: Comparacion entre algoritmos con respecto al numero de comparaciones que estos realizaron cuando los datos se analizaron de manera ordenada.

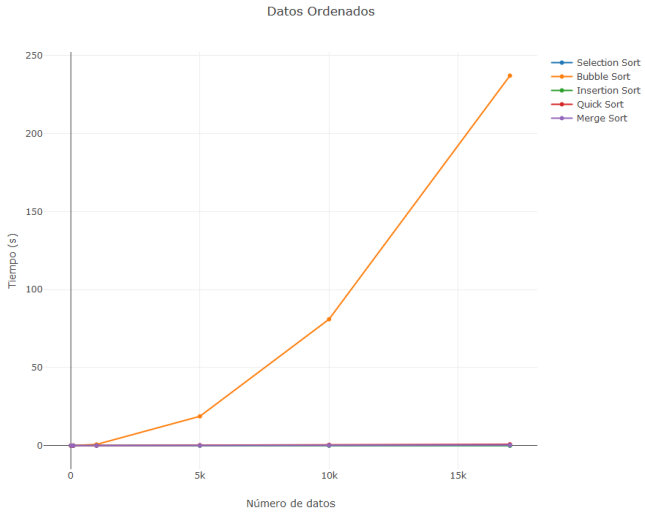

Figure 2: Comparacion entre algoritmos con respecto al tiempo de procesamiento requerido por cada uno de ellos cuando los datos se analizaron de manera ordenada.

Orden Inverso

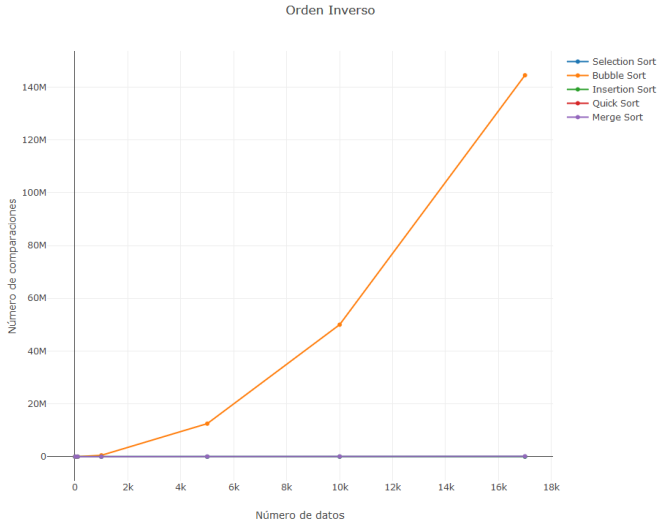

Figure 3: Comparacion entre algoritmos con respecto al numero de comparaciones que estos realizaron cuando los datos se analizaron siendo recibidos con un orden inverso.

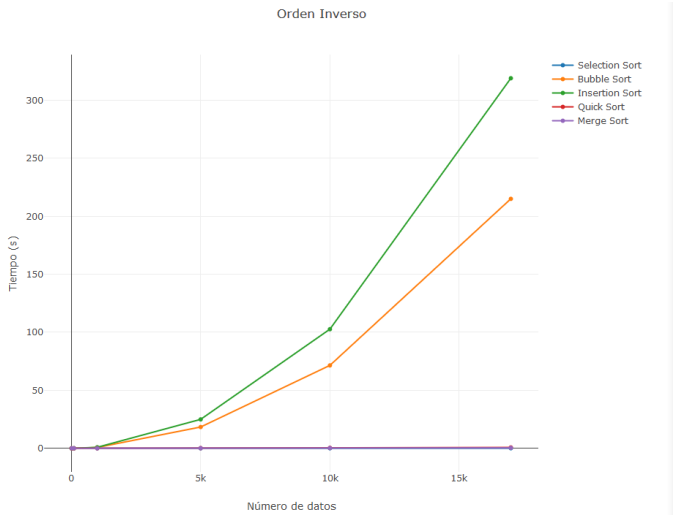

Figure 4: Comparacion entre algoritmos con respecto al tiempo de procesamiento requerido por cada uno de ellos cuando los datos se analizaron siendo recibidos con un orden inverso.

Orden Aleatorio

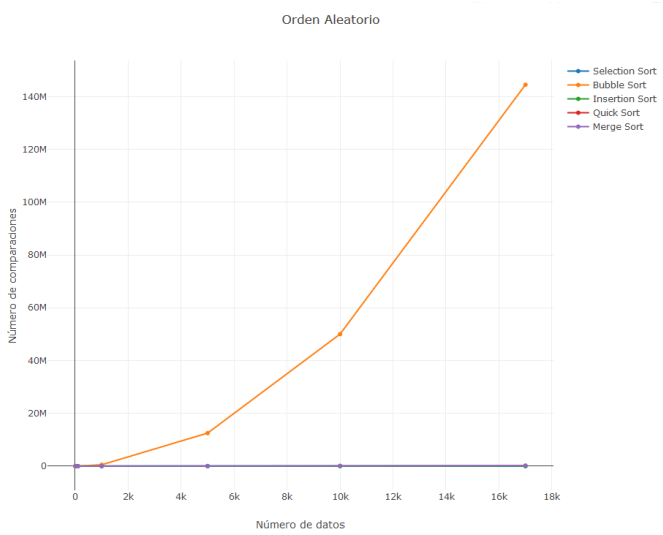

Figure 5: Comparacion entre algoritmos con respecto al numero de comparaciones que estos realizaron cuando los datos se analizaron siendo recibidos con un orden aleatorio.

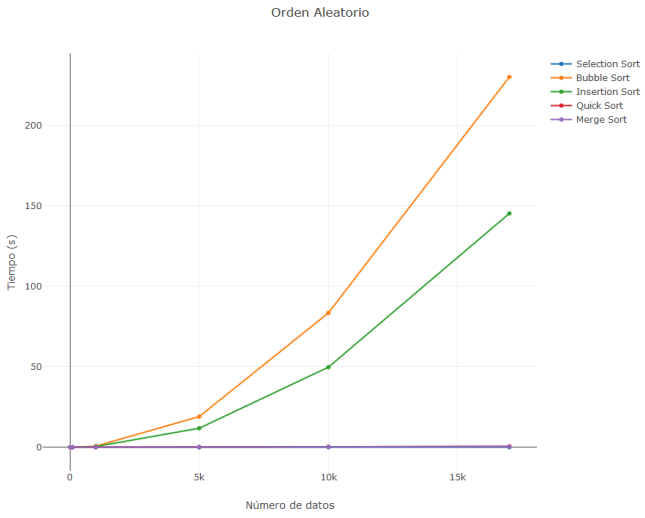

Figure 6: Comparacion entre algoritmos con respecto al tiempo de procesamiento requerido por cada uno de ellos cuando los datos se analizaron siendo recibidos con un orden aleatorio.

Conclusiones Despues de correr cada no de los algoritmos para los distintos valores que se escogieron, que fueron (10, 100, 1000, 5000, 10000, 17000), se pueden observar de manera visual los distintos comportamientos de cada una de los algoritmos, lo interesante es que varian con respecto al orden de los datos, asi como con respecto a la cantidad de datos que se estan analizando. A pesar de que parece imperceptible la diferencia entre los algorimos como el Merge Sort y el Quick Sort, si lo hay y es recomendable tomarla en cuenta, debido a que el Quick Sort es mas eficiente en cuanto a tiempo y cantidad de comparaciones realizadas. Lo que claramente se puede ver es que el peor algoritmo en cuanto a estos dos rubros es el Bubble Sort, seguido por el Insertion Sort. Esta fue una tarea que requirio de bastante tiempo, no solo para programar, pero tambien para correr los algoritmos, aun asi, gracias a lo visto en clase, donde se discutieron y se programaron todos los algoritmos (aunque fuera en pseudocodigo o en Java), se podria decir que no fue tan complicada. Considero que una de las grandes complicaciones fue aprender la estructura de Python y sobretodo como graficar los valores encontrados.