INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

YOSSHUA ELI CISNEROS VILLASANA

CLAVE UNICA: 179889

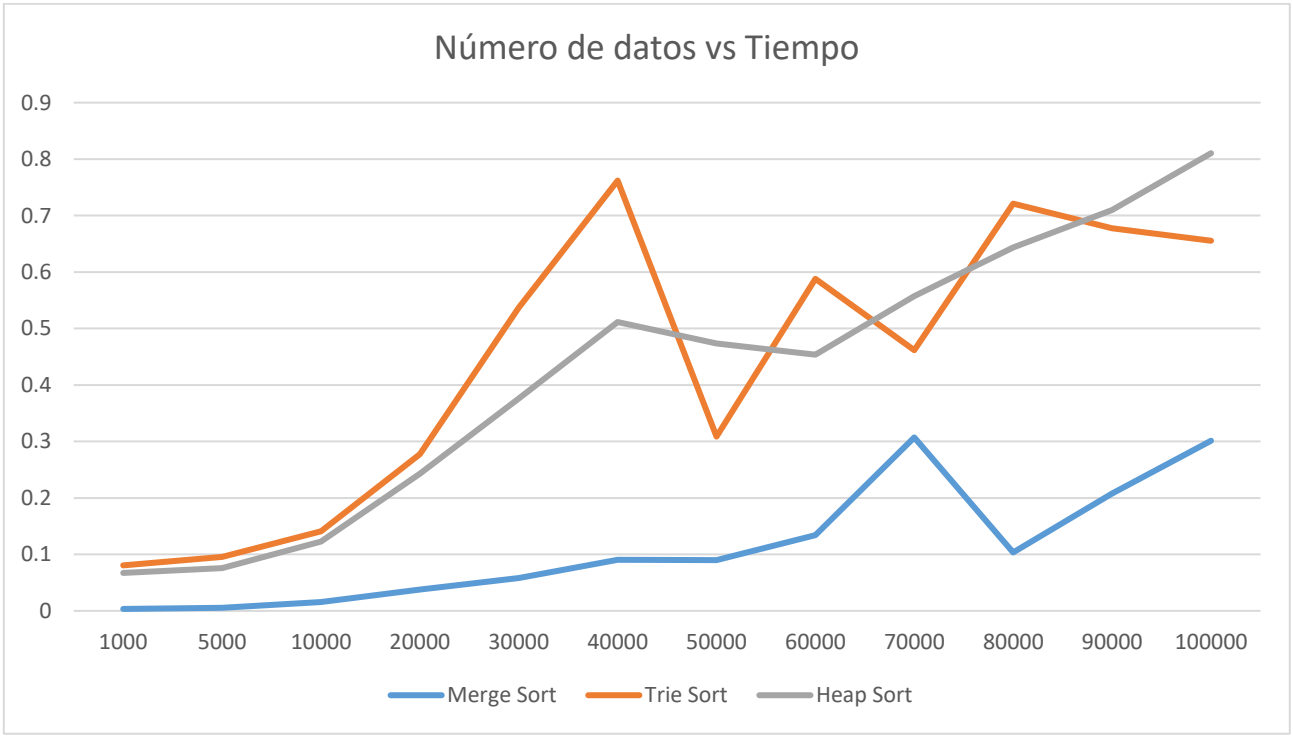
HEAP – TAREA EXTRA

PROFESOR: FERNANDO SPONDA DARLINGTON

MATERIA: ESTRUCTURAS DE DATOS AVANZADAS

Número de datos	Tiempo en segundos de Merge Sort	Tiempo en segundos de Trie	Tiempo en segundos Heap Sort
1000	0.0034434	0.0805792	0.0672254
5000	0.0056415	0.0956627	0.0758514
10000	0.015568	0.1408163	0.1226638
20000	0.0379145	0.2777323	0.2428046
30000	0.0579592	0.5369931	0.3761304
40000	0.0907125	0.7619792	0.5115882
50000	0.0900892	0.3088597	0.4733685
60000	0.1341295	0.5877904	0.4539124
70000	0.3071746	0.4617292	0.5572064
80000	0.1032839	0.7212812	0.6434315
90000	0.2076511	0.6775727	0.7097691
100000	0.3011337	0.655169	0.8104415

Tabla 1. Comparación del tiempo de ordenamiento



Gráfica 1. Comparación de tiempo respecto al número de datos

Análisis

Como era predecible, el Merge Sort se posicionó en primer lugar respecto del Trie Sort y el Heap Sort. En un principio yo pensé que el Heap Sort iba a tardar menos tiempo que los demás algoritmos de ordenación. Sin embargo, debido a la implementación el costo de tiempo aumenta cuando tenemos que reacomodar el Heap después de eliminar un elemento, es decir, se agrega O log n a la complejidad del algoritmo. En un principio parecería que tenemos acceso constante a los elementos al momento de eliminarlos, pero no es así. En cuanto al uso de memoria, probablemente el Heap se posiciona como el más ahorrador, pues no utiliza memoria extra. Sin embargo, es sabido que siempre que se utiliza recursión es más rápido que hacerlo de manera iterativa, pues utiliza herramientas internas de la computadora, lo cual lo hace más eficiente.

En la gráfica 1, observamos como el Heap Sort le gana al Trie Sort en la mayoría de los casos, probablemente con más datos ganaría el Trie Sort, pues el árbol que guarda los datos estaría lleno y sería más fácil el acceder a todos los datos. Pero solo en un caso en el que sean muchísimos datos, ya que, de no ser así, el Trie Sort sigue ocupando mucha memoria por la cantidad de espacio que reserva por cada carácter que se añade a un nodo.