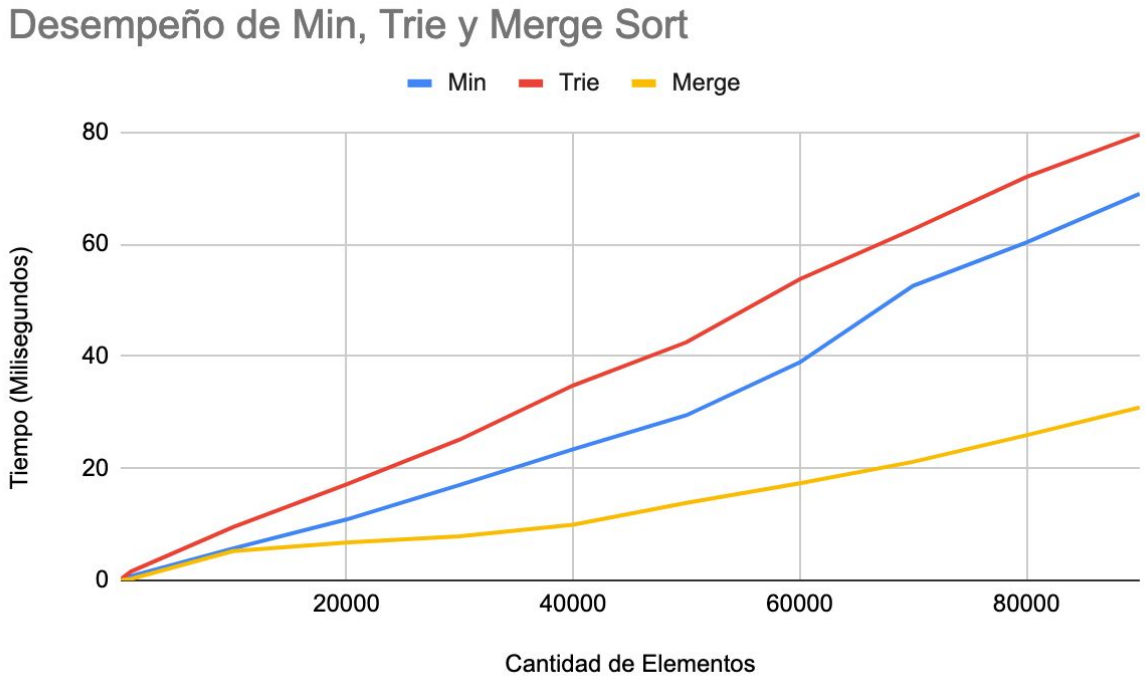
Jorge Edgar Rodríguez Ortiz Loyola 181334

Estructuras de Datos Avanzadas

Tarea Heaps

La tarea consistió en programar la clase Heap, precisamente un Min Heap para poder ordenar de menor a mayor (aunque se podría guardar al revés y ya se tienen ordenados los elementos de mayor a menor) implementando las operaciones de eliminar el mínimo, insertar, imprimir obtener los hijos de los nodos, saber si es hoja o no y ordenamiento; este último consistiendo en eliminar el mínimo hasta que quede vacío el Heap. Una vez que se tiene programada la clase Heap, se realizó un experimento para probar su desempeño. Se leyeron 90 mil palabras de un archivo de texto, se guardaron en un arreglo (pasando todo a minúsculas e ignorando palabras con caracteres especiales para que fueran compatibles con nuestra lista de símbolos de Trie, otro algoritmo con el cual lo comparamos) y se insertaron distintas longitudes de este arreglo en un Trie, en un nuevo arreglo auxiliar (para el Mergesort) y en el Heap. A continuación se ordenó el arreglo auxiliar con Merge Sort (previamente programado en clase) , se mandó a llamar al método Ordenamiento Lexicográfico en Trie y el método HeapSort de MinHeap . Se tomó el promedio del tiempo de los tres procesos después de correr cada uno 100 veces, los resultados son los siguientes:

	Tiempo (Milisegundos)		
# Elementos	Min	Trie	Merge
100	0,13	0,23	0,03
1000	0,77	1,58	0,25
10000	5,7	9,5	5,22
20000	10,86	17,14	6,75
30000	17,03	25,13	7,89
40000	23,4	34,79	9,93
50000	29,47	42,5	13,82
60000	38,9	53,76	17,34
70000	52,58	62,67	21,16
80000	60,31	72,02	25,91
90000	69,03	79,56	30,86



Los resultados indican que el algoritmo de Merge Sort es el más eficiente, a continuaciòn esta el ordenamiento utilizando el Min Heap y por último, aunque no mucho más atrás, el ordenamiento lexicográfico de los Tries.

Una ventaja, aparte de la eficiencia, que tiene el ordenamiento por Heaps sobre el ordenamiento Lexicográfico es que no es posible utilizar un Trie como sustituto a los algoritmos de ordenación previamente vistos, pues solo puede ordenar objetos compuestos. Tal es el caso de un String, compuesto por caracteres, en los que para realizar un CompareTo hay que irse fijando en cada uno de los subelementos del String.

En esta tarea aprendí que medir el tiempo de algoritmos requiere de mayor atención de la que le había estado prestando hasta el momento. Mediciones individuales presentan discrepancias a menudo. Por esto hacer las mediciones muchas veces y sacar un promedio arrojó resultados más confiables y sin saltos como lo habían hecho mis tareas pasadas. Otro punto importante es que cada computadora arroja tiempos distintos, por lo que cuando empecé a medir los tiempos en distintas computadoras (por intentar hacer parte de la tarea en mi casa y otra parte en el ITAM) obtenía resultados incongruentes. Tuve que hacer todas las mediciones en una sola computadora para arreglar esto.