Sofía Albert 181767

Estructuras de Datos Avanzadas

Heaps

- Programe la estructura de datos "Heap" con todas sus operaciones usando arreglos
- Implemente el heap-sort (consiste solamente en insertar los n datos a ordenar y luego ejecutar n borraMin())
- Compare el desempeño con el Merge sort y el Trie sort y entregue un documento vía Github detallando sus experimentos y resultados.

Heaps

Los heaps o cúmulos son árboles binarios. Todos los niveles están llenos excepto, tal vez, el último. Todas las hojas están recargadas hacia la izquierda, por lo que no hay nodos con hijo derecho sin tener un hijo izquierdo.
Al programar la estructura heap usando arreglos, los hijos están en un arreglo de tipo T, y para acceder al hijo izquierdo, se accede al arreglo en la posición del padre * 2, o a la posición del padre *2 +1 para acceder al hijo derecho.
La estructura heaps tiene como operaciones insertar (que inserta un dato y re-acomoda el árbol si es así necesario), buscar mínimo (que regresa al dato menos que es, además, la raíz del heap) y eliminar mínimo (que re-acomoda el árbol para que siga cumpliendo con las características del heap).

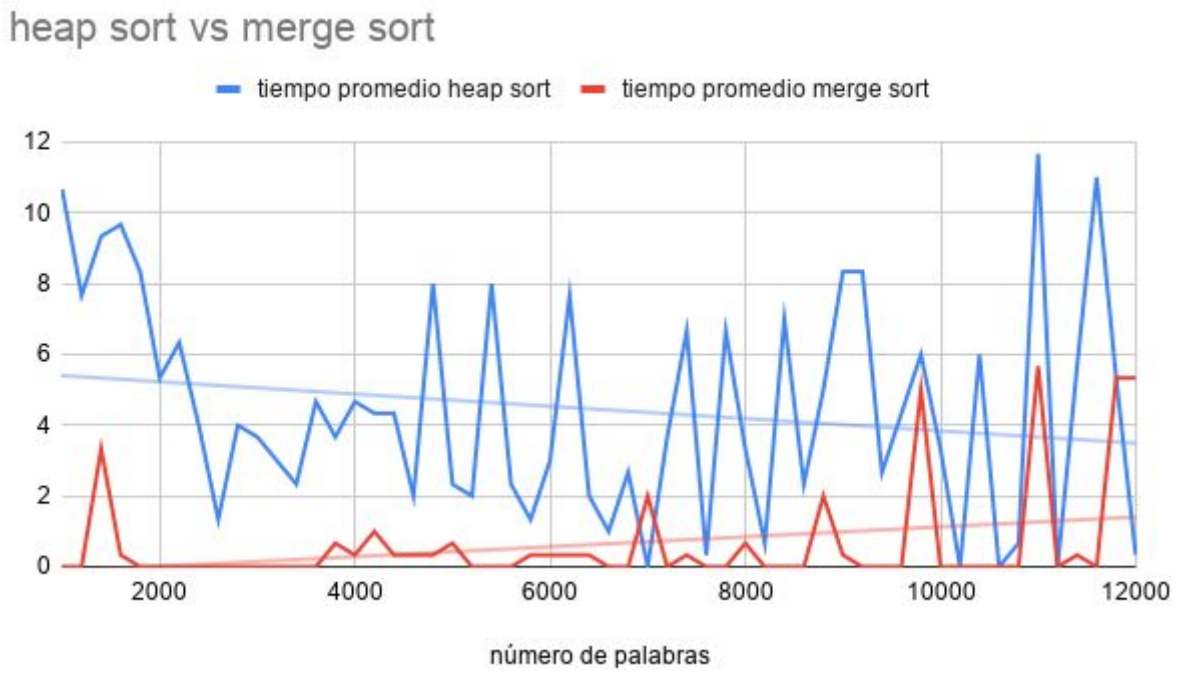
Experimento

Se condujo un experimento en NetBeans para comprar el desempeño de Merge Sort y el de Heap Sort. Se calculó la diferencia en milisegundos del tiempo que toma insertar y acomodar *n* palabras usando heap y usando merge sort. El merge sort es más rápido pues no es necesario insertar las palabras; parte de un arreglo dado. Aún así, resulta interesante comparar ambos desempeños.
Se probaron ambos algoritmos para diferente número de palabras: desde 200 hasta 12,000 en intervalos de 200 palabras.

Resultados

número de palabras	tiempo promedio heap sort	tiempo promedio merge sort
1000	10,66666667	0
1200	7,666666667	0
1400	9,333333333	3,333333333
1600	9,666666667	0,3333333333
1800	8,333333333	0
2000	5,333333333	0
2200	6,333333333	0
2400	4	0
2600	1,333333333	0
2800	4	0
3000	3,666666667	0
3200	3	0
3400	2,333333333	0
3600	4,666666667	0
3800	3,666666667	0,6666666667
4000	4,666666667	0,3333333333
4200	4,333333333	1
4400	4,333333333	0,3333333333
4600	2	0,3333333333
4800	8	0,3333333333
5000	2,333333333	0,6666666667
5200	2	0
5400	8	0
5600	2,333333333	0
5800	1,333333333	0,3333333333
6000	3	0,3333333333
6200	7,666666667	0,3333333333
6400	2	0,3333333333
6600	1	0
6800	2,666666667	0
7000	0	2
7200	3,666666667	0
7400	6,666666667	0,3333333333
7600	0,3333333333	0
7800	6,666666667	0

8000	3,333333333	0,6666666667
8200	0,6666666667	0
8400	7	0
8600	2,333333333	0
8800	5	2
9000	8,333333333	0,3333333333
9200	8,333333333	0
9400	2,666666667	0
9600	4,333333333	0
9800	6	5
10000	3,333333333	0
10200	0	0
10400	6	0
10600	0	0
10800	0,6666666667	0
11000	11,66666667	5,666666667
11200	0	0
11400	5,666666667	0,3333333333
11600	11	0
11800	5,333333333	5,333333333
12000	0,3333333333	5,333333333



Resultados

Podemos ver que el heap sort toma más tiempo que el merge sort, aunque la línea de tendencia del merge sort tiende hacia arriba, mientras que la del heap sort hacia abajo. Aún así, el heap sort no resulta tan eficiente pues los diferentes tiempos fluctúan mucho, no es un algoritmo constante, pero sí es un algoritmo funcional pues logra insertar y acomodar 12,000 palabras en menos de 12 milisegundos.