Implementing and evaluating performance from heap sort algorithm
Report #04

By
Eduardo Castro

CS 303 Algorithms and Data Structures

September 24, 2014

1. Problem Specification

The goal of this assignment was to write a program that will read a text file with a lot of different numbers, catch it and implement a heap sort algorithm for sorting all this numbers.

After the implementation we need to evaluate the results and compare the performance from heap sort algorithm for different sizes of data structures and compare it with the performance from insertion and merge sort algorithm.

2. Program Design

This program required only two classes: one for the Heap Sort and one driver class for testing it.

The following steps were required to develop this program:

a) write the Heap class
b) test it for different sets of data and possibilities using JUnit
c) write the HeapSortTest class
d) test it for different sets of data and possibilities using JUnit
f) use the driver class to display and measure the results/performance

The following methods were defined within the LinearSearch, BinarySearch and TestSearch classes:

a) heapSort (int[] A)
    a. Receives the array with all the numbers and make the sorting.
b) buildMaxHeap(int[] A)
    a. Call the maxHeapify method for the first part of the array
c) maxHeapify(int[] A, int i)
    a. Discover which one is bigger
d) swap(int[] A, int firstElement, int secondElement)
    a. Make a swap between first and second elements
e) left(int i)
    a. Just returns 2*i + 1
f) swap(int i)
    a. Just returns 2*i + 2

The Scanner class provided by Java was used to read in the necessary values within the provided driver program. The println method of the System.out object was used to display the inputs and results for the provided driver program.

3. Testing Plan

The provided driver program, HeapSortTest, was used to test the HeapSort class as required by the assignment. The following test cases were used for this assignment.

The inputs used came from texts files given for the assignment that comes with the data structure s with all the elements, the sizes varies between 16 and 2^24 elements.

The HeapSortTest class catches this file and test all this data using the HeapSort algorithm. For evaluating purposes the HeapSortTest also catches the execution time from each search. For comparison purposes we also need to test these data structures using the InsertionSort algorithm.

4. Test Cases

The test cases are shown in the table below:

Test Case Number	Input Quantity	Elapsed Time(Merge)	Elapsed Time(Insertion)	Elapsed Time(Mixed)	Elapsed Time(Heap Sort)
1	16	12551	5047	6882	28654
2	32	13605	6871	7674	49486
3	64	39767	22883	22873	306805
4	128	87616	72827	72877	54403
5	256	147916	313396	314333	86173
6	512	421148	1248783	304419	200143
7	1024	126093	1912295	151022	366786
8	2048	271383	2785986	274391	422122
9	4096	444822	28638266	473341	636303
10	8192	704550	1137221	569352	1217646
11	2^4	2039	786	9519	3023
12	2^5	2272	792	6859	4552
13	2^6	6351	1374	21731	8086
14	2^7	11059	3082	111558	15684
15	2^8	15674	8714	167931	31714
16	2^9	30860	29172	31191	96334
17	2^10	93587	105771	63536	201179
18	2^11	132862	422620	135035	423065

19	2^12	435466	1679821	245818	678192
20	2^13	582755	6508169	741876	1726389
21	2^14	1114154	26313129	1028820	2575729
22	2^15	2222611	105846646	2348747	4530959
23	2^16	4665448	415552749	4619049	8373864
24	2^17	10087389	1693814003	9437137	15300203
25	2^18	20471098	6803155383	20011695	35224655
26	2^19	43425139	27102843575	42149763	70104522
27	2^20	90666679	109702540923	88626826	153413722
28	2^21	192216106	441162640886	180633813	371670054
29	2^22	401116640	∞	387051274	869719347
30	2^23	830669870	∞	789643172	1897862395
31	2^24	15059400	∞	14420230	22405197

5. Analysis and Conclusions

It's interesting how the three different algorithms for sorting works differently according to the size of the input array. For small arrays the Insertion Sort algorithms seems to be much better and faster but for bigger ones it is really slow when compared to the Merge or Heap Sort algorithms. For arrays 4 to 15 the Heap Sort seemed to be much more fast.

So the best option seems to be, again, mixing the three algorithms into one program, when the data structure is small enough we call one sorting algorithm and when it's bigger enough we call another. Doing this we were capable of getting mixed results that were more "optimised". Seems to be the best to be done.

Relating to the asymptotic upper bound from the functions from the lab notes I found that for the function 'a' is O(n) and for the function 'b' is O(n log n).

6. References

The parameters used was from the homework assignment provided in class and I used as reference for learning how to manipulate text files in Java one post from Stack Overflow(http://bit.ly/1gS00Ne) and another post for learning how to measure time(http://bit.ly/Wck04t).