Implementing and evaluating performance from merge sort algorithm
Report #03

By
Eduardo Castro

CS 303 Algorithms and Data Structures

September 17, 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 an merge sort algorithm for sorting all this numbers.

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

2. Program Design

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

The following steps were required to develop this program:

a) write the MergeSort class
b) test it for different sets of data and possibilities using JUnit
c) write the MergeSortTest 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) mergeSort (int[] A, int temp, int p, int r)
    a. Receives the array with all the numbers, a temporary array that at the beginning it is exactly like the array A and also receives the current beginning position and the last position. This informations can be different according to the recursive call that will be made.
b) merge(int[] A, int temp, int p, int q, int r)
    a. It's the method that actually do the merge, it receives the start point, the middle point and the last point of the array that needs to be worked(after separating the left to the right side, etc).
c) getsHowManyElementsOnTheInputFile(String inputAddressFile)
    a. Makes a search between all the elements from the text file to discover how many elements it has.
d) sortCheckAsc(int[] a)
    a. Just checks if the array is really sorted. Returns true if it is and false if it is not.

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, MergeSortTest, was used to test the MergeSortclass 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 TestSearch.java class catches this file and test all this data using the MergeSort algorithm. For evaluating purposes the TestSearch 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)
1	16	12551	5047	6882
2	32	13605	6871	7674
3	64	39767	22883	22873
4	128	87616	72827	72877
5	256	147916	313396	314333
6	512	421148	1248783	304419
7	1024	126093	1912295	151022
8	2048	271383	2785986	274391
9	4096	444822	28638266	473341
10	8192	704550	1137221	569352
11	2^4	2039	786	9519
12	2^5	2272	792	6859
13	2^6	6351	1374	21731
14	2^7	11059	3082	111558
15	2^8	15674	8714	167931
16	2^9	30860	29172	31191

17	2^10	93587	105771	63536
18	2^11	132862	422620	135035
19	2^12	435466	1679821	245818
20	2^13	582755	6508169	741876
21	2^14	1114154	26313129	1028820
22	2^15	2222611	105846646	2348747
23	2^16	4665448	415552749	4619049
24	2^17	10087389	1693814003	9437137
25	2^18	20471098	6803155383	20011695
26	2^19	43425139	27102843575	42149763
27	2^20	90666679	109702540923	88626826
28	2^21	192216106	441162640886	180633813
29	2^22	401116640	∞	387051274
30	2^23	830669870	∞	789643172
31	2^24	15059400	∞	14420230

5. Analysis and Conclusions

It's interesting how the two 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 Sort algorithm. For arrays big as 2^22 elements or more I left my computer trying to sort it for more than 10 minutes and it was stucked with nothing happening.

For really small arrays, like up to 64 elements, the Insertion Sort was twice faster than the Merge Sort and for bigger ones the difference seems to be smaller. For data structures with more than 512 elements the Merge Sort algorithms gets better, for bigger ones the Merge Sort gets much better like been more than 2000x faster than the Insertion Sort algorithm.

So the best option seems to be mixing both 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.

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).