Implementing and evaluating performance from quick sort algorithm
Report #05

By
Eduardo Castro

CS 303 Algorithms and Data Structures

October 1, 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 quick 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 sort algorithm and also with another approach from quick sort algorithm, this time using the median of three.

2. Program Design

This program required only three classes: one for the traditional Quick Sort, another one for the Quick Sort using the median of three and one driver class for testing both.

The following steps were required to develop this program:

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

The following methods were defined within the QuickSort, QuickSortWithMedianOfThree and QuickSortTest classes:

a) quickSort(int[] arrayOfElements, firstElement, lastElement)
    a. Receives the array with all the numbers and make the sorting.
a) quickSort(int[] arrayOfElements, firstElement, lastElement)
    a. Receives the array with all the numbers and make the sorting. In case of the data structure that will be used be smaller than the CUTOFF then we call InsertionSort instead of QuickSort algorithm.
b) partition(int[] arrayOfElements, firstElement, lastElement)
    a. Method that makes the partition between the elements
c) swapElements(int[] arrayOfElements, firstElement, secondElement)
    a. Method that swaps two elements between each other
d) median3(int[] arrayOfElements, int firstElement, int i, int lastElement)
    a. Method that will return the position that holds the median value
e) insertionSort(int[] arrayOfElements, firstElement, lastElement)
    a. Method that sorts an array using insertionSort algorithm
f) getHowManyElementsOnTheInputsFile(String inputAddressFile)
    a. Method that returns the number of elements into an input file
g) sortCheckAsc(int[] a)
    a. Method that returns true if the data structure is sorted and false if it isn't
h) main()

a. Makes all the work, prepares the data structures, call the methods, etc.

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, QuickSortTest, was used to test the QuickSort 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 QuickSortTest class catches this file and test all this data using the QuickSort algorithm with two different approaches: the traditional one and with the median of three one. For evaluating purposes the QuickSortTest 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 #	Input Size	Elapsed Time (Merge)	Elapsed Time (Insertion)	Elapsed Time (Mixed)	Elapsed Time (Heap)	Elapsed Time (Quick)	Elapsed Time (Quick +Median of 3)
1	16	12551	5047	6882	28654	11993	15036
2	32	13605	6871	7674	49486	10297	11290
3	64	39767	22883	22873	306805	46837	42831
4	128	87616	72827	72877	54403	47319	48312
5	256	147916	313396	314333	86173	106413	69439
6	512	421148	1248783	304419	200143	232651	157906
7	1024	126093	1912295	151022	366786	492234	342837
8	2048	271383	2785986	274391	422122	263159	209035
9	4096	444822	28638266	473341	636303	262775	497264

10	8192	704550	1137221	569352	1217646	562435	1413036
11	2^4	2039	786	9519	3023	1348	1988
12	2^5	2272	792	6859	4552	2105	3407
13	2^6	6351	1374	21731	8086	4528	7046
14	2^7	11059	3082	111558	15684	8520	13869
15	2^8	15674	8714	167931	31714	16775	30478
16	2^9	30860	29172	31191	96334	37378	41541
17	2^10	93587	105771	63536	201179	76898	92467
18	2^11	132862	422620	135035	423065	115068	211623
19	2^12	435466	1679821	245818	678192	241898	781864
20	2^13	582755	6508169	741876	1726389	506084	542385
21	2^14	1114154	26313129	1028820	2575729	1321898	1416274
22	2^15	2222611	105846646	2348747	4530959	3245557	2479869
23	2^16	4665448	415552749	4619049	8373864	5970731	5070290
24	2^17	10087389	1693814003	9437137	15300203	12326387	10684651
25	2^18	20471098	6803155383	20011695	35224655	23836616	22547796
26	2^19	43425139	27102843575	42149763	70104522	51022723	45873886
27	2^20	90666679	0970254092	88626826	153413722	109280940	97598560
28	2^21	192216106	4116264088	180633813	371670054	233732097	204154704
29	2^22	401116640	∞	387051274	869719347	474394699	473591243
30	2^23	830669870	∞	789643172	1897862395	1013110592	921557977
31	2^24	15059400	∞	14420230	22405197	16600561	16614338

5. Analysis and Conclusions

It’s interesting how the QuickSort is much faster than any other of the algorithms with exceptions of some cases involving Merge and Insertion Sort. It’s kind of understandable why some programmers says that they tend to use QuickSort most of the times as default.

The median of three resulted in smaller times in some cases but I confess that I didn’t catch it’s purposes at all, the “gaining time” is not that much at all and in most of the cases it returned a bigger time.

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).