International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:2, No:6, 2008

# Analysis of Modified Heap Sort Algorithm on Different Environment

Vandana Sharma, Parvinder S. Sandhu, Satwinder Singh, and Baljit Saini

***Abstract*—**In field of Computer Science and Mathematics, sorting algorithm is an algorithm that puts elements of a list in a certain order i.e. ascending or descending. Sorting is perhaps the most widely studied problem in computer science and is frequently used as a benchmark of a system's performance. This paper presented the comparative performance study of four sorting algorithms on different platform. For each machine, it is found that the algorithm depends upon the number of elements to be sorted. In addition, as expected, results show that the relative performance of the algorithms differed on the various machines. So, algorithm performance is dependent on data size and there exists impact of hardware also.

***Keywords*—**Algorithm, Analysis, Complexity, Sorting.

## I. Introduction

SORTING algorithms are classified by several criteria such as Computational complexity where worst, average and best number of comparisons for several typical test cases in terms of the size of the list are computed. Stability is based on memory usage and use of other computer resources. The difference between worst case and average behavior, behaviors on practically important data sets. The data sets could be completely sorted, inversely sorted and almost sorted. There many algorithms are available for sorting. Such case requires comparison of algorithms to implement sorting on that data structure so that better one is chosen. The analysis of an algorithm is based on time complexity and space complexity. The amount of memory needed by program to run to completion is referred as space complexity. The amount of time needed by an algorithm to run to completion is referred as time complexity. For an algorithm time complexity depends upon the size of input.

In this paper, a comparative performance evaluation of improved heap sort algorithm is done with three traditional sorting algorithms: heap sort, quick sort, and merge sort. In order to study the interaction between the algorithms and the platform, all the algorithms have been implemented on different platforms for a range of integer data items.

Parvinder S. Sandhu is Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 India (phone: +91-98555-32004; e-mail: parvinder.sandhu@gmail.com).

Vandana Sharma is Sr. Lecturer with Computer Science & Engineering Department, Chitkara Institute of Engineering & Technology, Raj Pura (Punjab)-India.

Satwinder Singh is Lecturer (Computer Science & Engineering Department), Baba Banda Singh Bahadur Engg. College, Fategarh Sahib (Punjab)-India.

Baljit Saini is Lecturer (Computer Science & Engineering Department), Sant Baba Bhag Singh Institute of Engg. & Technology, Jalandhar (Punjab)-India

## II. Sorting Algorithms

According to Knuth theoretical lower bound for general sorting algorithms is [1]:

$$\log(n!) = n\log n - n\log e + \theta(\log n)$$
$$\approx n \log n - 1.442695n$$

For the worst-case numbers of comparisons, this lower bound makes sorting by merging, sorting by insertion and binary search very efficient.

Merge Sort follow divide and conquer approach. Merge Sort performs at most nlogn – n+1 key comparisons and requires O(n) extra space.

Quick Sort is also a divide and conquer algorithm that is most often implemented using recursion [2]. Quick Sort has worst-case running time of $O(n^2)$ but is typically O(nlogn) and in practice one of the fastest of the comparison based sorting algorithms. Hoare proposed CLEVER-QUICKSORT in worst case still it has $\Theta(n^2)$ comparisons and in average case number of comparisons are reduced to 1.188nlogn-2.255n [3].

HEAPSORT needs 2nlogn comparisons [4-5]. Heap Sort is a divide and conquer algorithm that first orders keys in a binary heap and then reorders the heap into sorted order [6]. Heap Sort is an optimal comparison sort, achieving O(nlogn) performance for any input ordering. It is relatively easy to implement as an in-place and non-recursive sort. In 1990 in proceedings of MFCS90, Wegner proved that Heap sort needs 2nlogn comparisons and upper bound for comparisons in Bottom-up-heap sort of 1.5nlogn [7].

In [8] worst case number of comparisons of the algorithm is about 1.5nlogn-0.4n. A variant of Heap sort proposed by Carlson, needs nlog n + (n log log n) comparisons [9]. Wegner showed that McDiarmid and Reed's variant of Bottom-up-heap sort needs nlogn+1.1 n comparisons [10].

A new variant of Heap Sort is modified heap sort. Basic idea of new algorithm is similar to classical Heap sort algorithm but it builds heap in another way. This new algorithm requires nlogn-0.788928n comparisons for worst-case and nlogn-n comparisons in average case [11] if it uses Gonnet and Munro's [12] fastest algorithm for building heaps. This algorithm uses only one comparison at each node. With one comparison it can be decided which child of node

contains larger element. This child is directly promoted to its parent position in this way algorithm walks down the path until a leaf is reached.

The McDiarmid and Reed's Variant of BOTTOM-UP-HEAPSORT algorithm [13] uses, on average about 1.52n comparisons to build a heap.

WEAK HEAPSORT proposed by Dutton [14] uses less than nlogn+0.086013n comparisons. RELAXED-WEAK-HEAPSORT is a WEAK HEAPSORT [15] variant which consumes at most O(nlogn) extra bits and executes exactly nk-$2^k$+1 comparisons in best worst and average cases if k=[log n].

The MacDiarmid and Reed's variant of BOTTOM –UP-HEAP SORT algorithm [13,16-19] uses, on average, about 1.52n comparisons to build a heap. Reinhard [20] shows that MERGESORT can be designed in place with nlogn –1.3n + O (logn) comparisons in worst case, but practical purpose algorithm is too slow.

## III. Performance Study

In the previous section, complexities of heap sort, quick sort, merge sort and improved heap sort algorithm are discussed. A detailed study to assess the performance of improved heap sort algorithm with respect to the heap, quick and merge sort algorithms on different platforms is conducted. The performance metric in all the experiments is the total execution time taken by the sorting operation. These test beds are:

- Test Bed I: Celeron 2.5 GHz, 512 MB RAM, 40GB HDD, Windows XP Professional with service Pack 2, Microsoft Visual C++ compiler
- Test Bed II: AMD 2800+, 512 MB RAM, 40GB HDD, Windows XP Professional with service Pack 2, Microsoft Visual C++ compiler
- Test Bed III: Pentium 4, 2.4 GHz, 512MB RAM, 80 GB HDD, Windows XP Professional with Service Pack 2, Microsoft Visual C++ compiler
- Test Bed IV: AMD 64 bit, 1.8GHz, 512 MB RAM, 80 GB HDD, Windows XP Service Pack 2, Microsoft Visual C++ compiler
- Test Bed V: Pentium, 1.6 GHz, 1GB RAM, 60 GB HDD, Windows XP Professional Service Pack2, Microsoft Visual C++ compiler

Results of Teat Bed II are shown in Table I. It shows the execution times of all the four algorithms for number of data items ranging from 1K to 100K. In Fig. 1, it is observed that modified heap sort algorithm takes less time than other sorting algorithms for data items 100K .

Results of Test Bed III are shown in Table II. It represents the execution times of all the four algorithms for no. of data items ranging from10 to 100K similar as in category I.

Fig. 2 shows that modified heap sort beats all other sorting algorithms in question for higher no of data items.

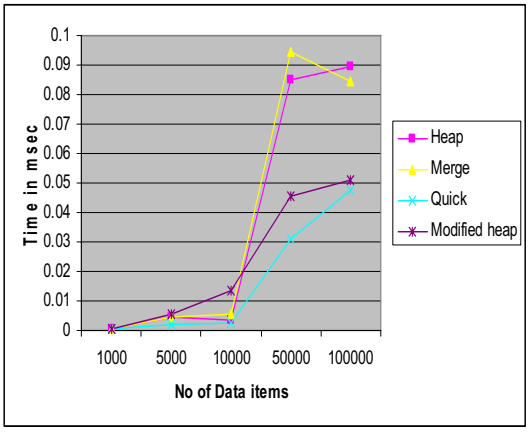


Fig. 1 Sorting Algorithms Performace Category-I

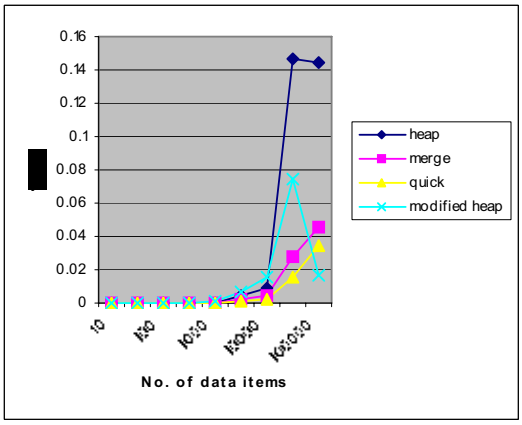


Fig. 2 Sorting Algorithms Performance Category-II

TABLE I
AVERAGE SORTING TIME (IN MSEC) OF ALGORITHMS ON RANDOM DATA AVERAGED 50 RUNS (TEST BED II)

| Sorting Algorithm | Number of data item | | | | |
|---|---|---|---|---|---|
| | 1000 | 5000 | 10000 | 50000 | 100000 |
| Heap | 0.0003 | 0.0045 | 0.0033 | 0.0848 | 0.0895 |
| Merge | 0.0003 | 0.0045 | 0.0054 | 0.094 | 0.0842 |
| Quick | 0.0003 | 0.0021 | 0.002 | 0.0311 | 0.0473 |
| Modified Heap | 0.0003 | 0.0057 | 0.013 | 0.0454 | 0.0511 |

TABLE II
AVERAGE SORTING TIME (IN MS) OF ALGORITHMS ON RANDOM DATA AVERAGED 50 RUNS (TEST BED III)

| Sorting Algorithm | Number of data item | | | | |
|---|---|---|---|---|---|
| | **500** | **1000** | **5000** | **10000** | **50000** |
| **Heap** | 0 | 0.0003 | 0.0048 | 0.0092 | 0.14628 |
| **Merge** | 0 | 0 | 0.0027 | 0.0042 | 0.0273 |
| **Quick** | 0 | 0.0003 | 0.0006 | 0.0021 | 0.01548 |
| **Modified Heap** | 0 | 0.0015 | 0.0066 | 0.0154 | 0.07428 |

## IV. CONCLUSION

This paper presented the comparative performance study of four sorting algorithms on different platform. For each machine, it is found that the algorithm depends upon the number of elements to be sorted. In addition, as expected, results show that the relative performance of the algorithms differed on the various machines. In category I for large number of data items performance of improved heap sort is better than heap sort. But for small number of data items performance of modified heap sort is similar to other algorithms. But in Category II platform, in case of data size 1K, modified heap sort algorithm performs exceptionally well. It outperforms all other algorithms.

## REFERENCES

[1] Knuth D E. The Art of Programming-Sorting and Searching. 2nd edition Addison Wesley.
[2] Thomas H, Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms, 2nd edition, MIT Press, Cambridge, May 2001, Chap. 7.
[3] Hoare C A R. Quicksort. Computer Journal, 5(1):10-15.
[4] Floyd R W. Algorithm 245: Treesort 3. Communications of ACM, 1964, 7(4): 701.
[5] Williams J W J. Algorithm 232: HEAPSORT. Communications of ACM, 1964, 7(4): 347-348.
[6] Cormen et al. Introduction to Algorithms, Chap. 6.
[7] I.Wegner: BOTTOM-UP-HEAPSORT beating on average QUICKSORT (if n is not very small). Proceedings of the MFCS90, LNCS 452: 516-522, 1990
[8] Wegner I. The Worst Case Complexity of McDiarmid and Reed's Variant of BOTTOM-UP HEAP SORT. Information and computation, 1992, 97(1): 86-96.
[9] S.Carlsson: A variant of HEAPSORT with almost optimal number of comparisons. Information Processing Letters, 24: 247-250, 1987.
[10] I.Wegner: The worst case complexity of Mc diarmid and Reed's variant of BOTTOM-UP-HEAP SORT. Proceedings of the STACS91, LNCS 480: 137-147, 1991.
[11] Xio Dong Wang, Ying Jie Wu. An improved heap sort algorithm with nlogn –0.788928n comparisons in worst case. Journal of Computer Science and Technology. 22(6): 898-903.
[12] Gonnet G H, Munro J I. Heaps on Heaps. SIAM Journal on Computing, 1986, 15(6): 964-971.
[13] McDiarmid C J H, Reed B A. Building Heaps Fast. Journal of Algorithms, 1989, 10(3): 352~365.
[14] Dutton R D. Weak Heap Sort. BIT, 1993, 33(3): 372-381.
[15] Edelkamp A S, Stiegeler P. Implementing HEAPSORT with nlogn - .0n and QUICKSORT with nlogn+0.2n comparisons. ACM journal Of Experimental Algorithmics (JEA), 2002, 7(1): 1-20.
[16] Cantone D, Cincotti G. QuickHeapsort: an efficient mix of classical sorting algorithms. Theoretical Computer Science 2002, 285(1): 25-42.
[17] Carlsson S, Chen J. The Complexity of Heaps. The Third Annual ACM SIAM symposium on Discrete Algorithms, SIAM, Philandelphia, PA, October 1992, pp 393-402.
[18] Ding Y, Weiss M A. Best Case Lower Bounds for Heap Sort. Computing, 1992, 49(1): 1-9.
[19] Z LiBruce A. REEd: Heap Building Bounds. LNCS, 2005, 3608(1): 14-23.
[20] Reinhard K. Sorting in Place with a Worst Case Complexity of nlogn-1.3n + O(logn) comparisons and nlogn+O(1) transports. LNCS, 1992, 650(6): 489-499.