Journal of Computer Science 5 (7): 476-478, 2009
ISSN 1549-3636
© 2009 Science Publications

Comparative Performance Study of Improved Heap Sort Algorithm on Different Hardware

[1]Vandana Sharma, [2]Satwinder Singh and [3]K.S. Kahlon
[1]Department of Computer Science and Engineering,
Chitkara Institute of Engineering and Technology, Jansla, Punjab, India
[2]Department of Computer Science and Engineering and Information Technology,
Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, Punjab, India
[3]Department of Computer Science and Engineering,
Guru Nanak Dev University, Amritsar, Punjab, India

Abstract: Problem statement: Several efficient algorithms were developed to cope with the popular task of sorting. Improved heap sort is a new variant of heap sort. Basic idea of new algorithm is similar to classical Heap sort algorithm but it builds heap in another way. The improved heap sort algorithm requires nlogn-0.788928n comparisons for worst case and nlogn-n comparisons in average case. This algorithm uses only one comparison at each node. Hardware has impact on performance of an algorithm. Since improved heap sort is a new algorithm, its performance on different hardware is required to be measured. **Approach:** In this comparative study the mathematical results of improved heap sort were verified experimentally on different hardware. To have some experimental data to sustain this comparison five representative hardware were chosen and code was executed and execution time was noted to verify and analyze the performance. **Results:** Hardware impact was shown on the performance of improved heap sort algorithm. Performance of algorithm varied for different datasets also. **Conclusion:** The Improved Heap sort algorithm performance was found better as compared to traditional heap sort on different hardware, but on certain hardware it was found best.

Key words: Complexity, performance of algorithms, sorting

INTRODUCTION

As Knuth describes in[1] theoretical lower bound for general sorting algorithms is:

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

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

Cormen[2] describes Heap Sort is a divide and conquer algorithm that first orders keys in a binary heap and then reorders the heap into sorted order.

Heap sort was originally proposed by William in[3]. A heap of size n is an array a[1..n] containing n elements satisfying the following conditions (1) Each component of the array stores exactly one element; (2) The array represents a binary tree completely filled on all levels except possibly at the lowest, which is filled from the left up to appoint; (3) The root of the tree is a[1]; (4) for a node I in the binary tree, a[i] is its key parent(i) = $\lfloor i/2 \rfloor$ is its parent and 2i and 2i+1 are its children, if they exist; (5) The heap property is for all $2\leq i\leq n$, a{parent(i)]$\geq$ a[i].Thus the largest element in a heap is always at the root of the heap. There are two phases of in any heap sort algorithm. First, the input array is transformed into heap. Secondly element at the root is exchanged with the last element of the heap and the heap is rearranged to build an new heap with one fewer element. This is the most important phase and repeated (n-2 times) until the input array is entirely sorted. Algorithm of William[3] uses nlogn + O(n) comparisons in the worst case to build a heap on n elements and more than 2nlogn comparisons in the worst case to sort the elements.

Floyd[4] improved William's algorithm. His algorithm uses 2n comparisons in the worst case to build a heap. The sorting phase requires at most 2nlogn comparisons. The average case is hardly better than the worst case.

Wegner[5] proposed new variant of heap sort Bottom up heap sort which works like a heap but it rearranges the remaining heap in different way.

Carlson in[6] proposed a variant of Heap sort needs nlogn + (nlogn) comparisons.

Corresponding Author: Vandana Sharma, Department of Computer Science and Engineering, Chitkara Institute of Engineering and Technology, Jansla, Punjab, India

McDiarmid and Reed proposed[7] a new variant of bottom up heap to reduce number of comparisons. This algorithm uses 2 |_(n-1)/2 _| additional bits, 2 bits per internal nodes for storing three values u(unknown),l (left) and r(right). It has been shown that the algorithm uses (n+1) log n + 1.086072 n key comparisons in the worst case. In[7] algorithm for only heap creation phase was presented. Complete algorithm is found in[8]. Wegner in[8] showed that McDiarmid and Reed's variant of Bottom-up-heap sort needs nlogn +1.1 n comparisons. Wegner showed that worst case number of comparisons of the algorithm is about 1.5nlogn-0.4n. In the average case although in worst case.

Bojesen *et al*.[9] studied behavior of three methods for constructing a binary heap on different architecture and compilers. The methods considered were proposed by Williams, in which elements are repeatedly inserted into a single heap; the improvement by Floyd, in which small heaps are repeatedly merged to bigger heaps ; and another method proposed by[10] in which a heap is built layer wise. In[9] they showed that with careful memory tuning and code tuning performance of heap construction could be improved by a factor of two or three.

Wang and Wu in[11] presented a new variant of Heap Sort which is improved heap sort. Basic idea of this new algorithm is similar to classical Heap sort algorithm but it builds heap in another way. Basic idea is to use only one comparison at each node. In this algorithm shift walks down a path in the heap until a leaf is reached. The request of placing the element in the root immediately to its destination is relaxed. This new algorithm requires nlogn-0.788928n comparisons for worst-case and nlogn-n comparisons in average case which is only about 0.4n more than necessary. If it uses Gonnet and Munro’s[12] fastest algorithm for building heaps. It beats on average even the clever variant of Quick sort, if n is not very small. In it is shown that there is effect of platform on performance of algorithm. Performance of this new variant of heap sort was required to be measured on different platforms. So the research was carried out to check the behavior of the algorithm on different platforms in comparison to other traditional algorithms.

MATERIALS AND METHODS

In[11] improvement of complexity was shown mathematically and this was verified on five different hardware test beds. The experiments were conducted on following Test beds:

Test bed I: Celeron 2.5 GHz, 512 MB RAM, 40 GB HDD, Windows XP Professional with service Pack 2, Microsoft Visual C++ compiler.

Test bed II: AMD 2800+, 512 MB RAM, 40 GB HDD, Windows XP Professional with service Pack 2, Microsoft Visual C++ compiler.

Test bed III: Pentium 4, 2.4 GHz, 512 MB RAM, 80 GB HDD, Windows XP Professional with Service Pack 2, Microsoft Visual C++ compiler.

Test bed IV: AMD 64 bit, 1.8 GHz, 512 MB RAM, 80 GB HDD, Windows XP Service Pack 2, Microsoft Visual C++ compiler.

Test bed V: Pentium, 1.6 GHz, 1 GB RAM, 60 GB HDD, Windows XP Professional Service Pack 2, Microsoft Visual C++ compiler.

For the experiments randomly generated integer numbers have been used. Data files were used to obtain results. To study the performance of the algorithms data sets with 5-100 K items were used and code was executed 50 times and average execution time in ms was recorded for average case. Execution time for every dataset on each test bed was compared and analyzed.

RESULTS AND DISCUSSION

Results of all five Test beds are shown in Table 1. It shows the execution times of improved heap sort algorithm for no. of data items ranging from 5-100 K on all five Test beds. It is observed that as the size of the data increases the performance of the Improved heap sort algorithm improves dramatically as shown in Fig. 1.

In Fig. 1 it is observed that improved heap sort shows better performance on Test bed II. This result was further compared with the results of traditional heap sort on Test bed II as shown in Table 2.

Comparison of performance of traditional heap sort and improved heap sort is shown in Fig. 2 which clearly shows that improved heap sort performed better than the traditional heap sort algorithm.

Table 1: Average sorting time (ms) of improved heap sort algorithm on random data averaged 50 runs

No. of data items	5000	10000	50000	100000
Test bed I	0.00420	0.00822	0.06974	0.15444
Test bed II	0.00420	0.00900	0.04774	0.04870
Test bed III	0.00530	0.01248	0.05942	0.11612
Test bed IV	0.00570	0.01350	0.04540	0.05112
Test bed V	0.00464	0.01050	0.06178	0.13174

Table 2: Average sorting time (ms) of improved heap sort algorithm and traditional Heap sort on Test bed II

No. of data items	5000	10000	50000	100000
Improved Heap sort	0.0042	0.009	0.04774	0.04870
Traditional Heap sort	0.0045	0.009	0.04174	0.08648

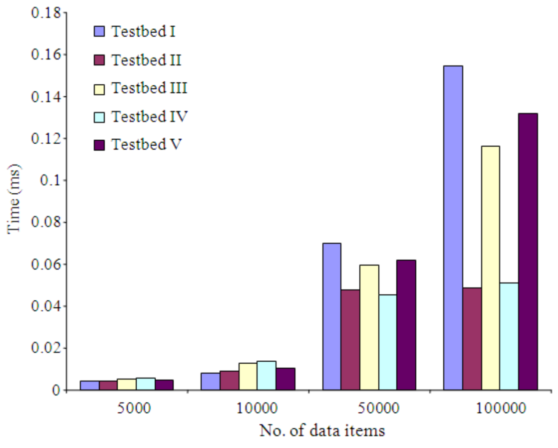

Fig. 1: Comparison of improved heap sort

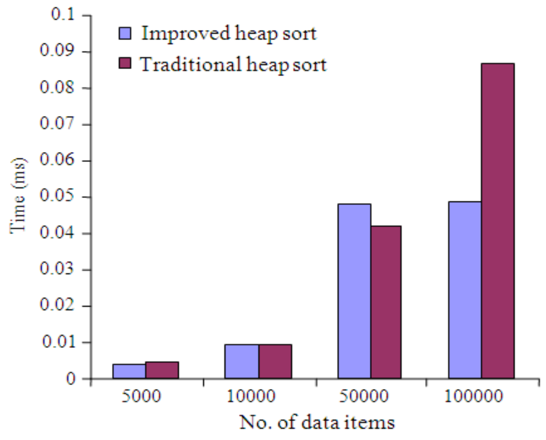

Fig. 2: Comparison of improved heap sort and traditional heap sort

CONCLUSION

It was verified that improved heap sort showed better performance on all Test beds. On Test bed I and Test bed V improved heap sort took less time for small dataset 5 and 10 K than other Test beds. For higher datasets of 50 and 100 K it took more time on Test bed I and V than on any other Test beds. Performance of improved heap sort was best on Test bed II for all datasets in comparison with other Test beds. However it's performance on Test bed II was comparable with performance on Test bed IV. Performance of improved heap sort on Test bed III for all datasets was consistent for all datasets. So it is recommended that improved heap sort is most suitable for the hardware configuration mentioned in Test bed II. It takes more time to sort larger datasets on Test bed I and V. In applications where algorithm performance needs to be consistent for all data ranges improved heap sort is suitable for Test bed III. In design and analysis of algorithm effect of caching can be taken in to account. By including memory system performance an algorithm analysis may lead to more correct results. For future work cache performance of improved heap can be studied which may lead more optimization.

REFERENCES

1. Knuth, D.E., 1988. The Art of programming-Sorting and Searching. 2nd Edn., Addison Wesley, ISBN: 020103803X, pp: 780.
2. Cormen, T.H. *et al*. 2001. Introduction to Algorithms. 2nd Edn., ISBN: 0262032937, pp: 1180.
3. Williams, J.W.J., 1964. ACM algorithm 232: Heap sort. Commun. ACM., 7: 347-348.
4. Floyd, R.W., 1964. ACM algorithm 245: Tree sort 3. Commun. ACM., 7: 701.
5. Wegner, I., 1990. Bottom-up-heap sort a new variant of heap sort beating on average quick sort (if n is not very small). Proceedings of the Mathematical Foundations of Computer Science 1990, Aug. 27-31, Springer-Verlag, London, UK., pp: 516-522. http://portal.acm.org/citation.cfm?id=663561
6. Carlsson, S., 1987. A variant of Heap sort with almost optimal number of comparisons. Inform. Process. Lett., 24: 247-250. http://portal.acm.org/citation.cfm?id=30995
7. McDiarmid, C.J.H. and B.A. Reed, 1989. Building heaps fast. J. Algorithms, 10: 352-365. April 1993
8. Wegner, I., 1991. The worst case complexity of Mc diarmid and Reed's variant of bottom-up-heap sort is less than nlogn+1.1n. Proceedings of the 8th annual symposium on Theoretical aspects of computer science, (ASTSCS'91), Springer-Verlag, Hamburg, Germany, pp: 137-147. http://portal.acm.org/citation.cfm?id=112114
9. Bojesen, J., J. Katajainen and M. Spork, 2000. Performance engineering case study: Heap construction. Lecture Notes Comput. Sci., 1668: 301-315. http://cat.inist.fr/?aModele=afficheN&cpsidt=1827233
10. Fadel, R., K.V. Jackobsen, J. Katajainen and J. Teuhola, 1999. Heaps and heap sorty on secondary storage. Theor. Comput. Sci., 220: 345-362. http://www.diku.dk/hjemmesider/ansatte/jyrki/Course/Performance-Engineering-1998/TCS(9.11.1998).ps
11. Wanga, X.D. and Y.J. Wu, 2007. An improved heap sort algorithm with nlogn-0.788928n comparisons in worst case. J. Comput. Sci. Technol., 22: 898-903. http://d.wanfangdata.com.cn/Periodical_jsjkxjsxb-e200706012.aspx
12. Gonnet, G.H. and J.I. Munro, 1986. Heaps on heaps. SIAM J. Comput., 15: 964-971.