IDEAL TREE SORT

Hari Krishna Gurram[1], Dharmaiah Gurram[2]

Abstract: One of the fundamental issues in computer science is ordering a list of items. Although there is a number of sorting algorithms, sorting problem has attracted a great deal of research, because efficient sorting is important to optimize the use of other algorithms. This paper presents Ideal Tree Sort(IT Sort) algorithm which runs in two stages. In first stage preprocessing of the elements is done to overcome the worst case problem of Tree sort. In second stage normal Tree sort is called to sort the elements. This algorithm was analyzed, implemented and tested and the results are promising for a random data, sorted data.

Keywords: Ideal Tree (IT) Sort, Middle heap

1. INTRODUCTION

Today real world getting tremendous amounts of data from various sources like data warehouse, data marts etc. To search for particular information we need to arrange this data in a sensible order. Many years ago, it was estimated that more than half the time on commercial computers was spent in sorting. Fortunately variety of sorting algorithms came into existence with different techniques Many algorithms are well known for sorting the unordered lists. Most important of them are merge sort, heap sort, shell sort and quick sort etc., As stated in [1], sorting has been considered as a fundamental problem in the study of algorithms, that due to many reasons:

- The need to sort the information is inherent in many applications.
- Algorithms often use sorting as a key subroutine.
- Many engineering issues come to the fore when implementing sorting algorithms.
- In algorithm design, there are many essential techniques represented in the body of sorting algorithms.
- Sorting algorithms plays a vital role in various indexing techniques used in data warehousing, and daily transactions in online Transactional processing (OLTP). Efficient sorting is important to optimize the use of other sorting algorithms that require sorted lists correctly.

Sorting algorithms can be classified by:

- Computational complexity: (best, average and worst behavior) of element comparisons in terms list size n. For a typical sorting algorithm best case, average case and worst case is O(n log n), example merge sort.

ISBN - 978-93-81583-55-5

- Number of swaps
- Stability : A sorting algorithm is stable if whenever there are two records X and Y,

with the same key and X appearing before Y in original list, X will be appear before Y in the sorted list.

In this paper, a new sorting algorithm (Ideal Tree sort) is proposed; here the basic idea is first we preprocess the elements to overcome the worst case problem of qTree sort, next this preprocessed data is given as an input to Tree sort.

Section 2 presents the concept of Tree sort algorithm and its merits and demerits, Section 3 discusses IT sorting algorithm. Section 4 shows the implementation results for various sizes of random input and sorted input. Finally, the conclusion was presented in section 5.

2. TREE SORT

A [1] tree sort is a sort algorithm that builds a binary search tree from the keys to be sorted, and then traverses the tree (in-order) so that the keys come out in sorted order. Its typical use is when sorting the elements of a stream from a file. Adding one item to a binary search tree is on average an O(log(n)) process, so adding n items is an O(n log(n)) process, making Tree Sort a so-called, 'fast sort'. But adding an item to an unbalanced binary tree needs O(n) time in the worst-case, when the tree resembles a linked list (degenerate tree), causing a worst case of O(n2) for this sorting algorithm.

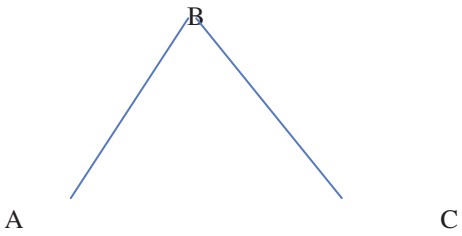
The worst case scenario then is triggered by handing a Tree Sort algorithm an already sorted set. This would make the time needed to insert all elements into the binary tree O(n2). The dominant process in the Tree Sort algorithm is the "insertion" into the binary tree, assuming that the time needed for retrieval is O(n). The worst-case behavior can be improved upon by using a self-balancing binary search tree. Using such a tree, the algorithm has an O(n log(n)) worst-case performance, thus being degree-optimal. But maintaining balanced binary search trees is a complex task, so this paper proposes a new way of balancing binary search tree that removes the worst case behavior of tree sort.

3. IDEAL TREE SORT ALGORITHM

IT algorithm is a two-step process:
STEP1: Construct Middle heap for the given input data.
STEP2: Sort the preprocessed data by using Tree sort.

3.1 Middle Heap Construction:

Middle Heap:
If A, B and C are three elements then middle Heap like below.



A , B and C satisfy the one of below property:

Property 1: B > A AND B < C
Property 2: B < A AND B > C
Example: int input[] = { 0, 1, 2, 3, 4, 5, 6}

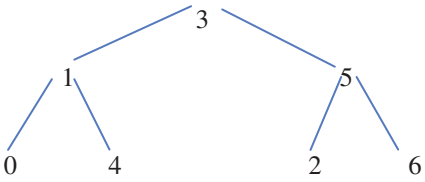


Fig 1: Middle heap for the elements in array

input[].
Algorithm:

```
BuildMiddleHeap(int input[])
{
      n = input.length
      for (int v=n/2-1; v>=0; v--)

                {
                                                        downheap (v)
}
}

downheap(v)
{
       x = v
While( x < n)

{
find left and right elements of input[x]
left = 2*v+1
right = 2*v +2
Find the middle element position in input[] at x, left and right positons.
Swap the elements at positions x and middle.
if( x = middle)
```

```
break;
x = middle
}
}
```

3.2 Call the Tree sort on preprocessed Data:

Once the data is preprocessed give this preprocessed data as an input to tree sort. This preprocessing removes the worst case behavior of normal tree sort algorithm.

4. IMPLEMENTATION RESULTS:

We compare the IT sort with tree sort in all possible scenarios like random data, sorted data in ascending order and sorted data in descending order. Implementation results shows that IT sort shows equal performance as tree sort when random data is given as input and shows better performance when sorted data is given as an input.. The total time taken by IT sort is equal to the Middle heap time plus tree sort time. By constructing middle heap the worst case scenario of Tree sort removed

Fig 2: Comparison of Ideal Tree sort and Tree Sort on random input data

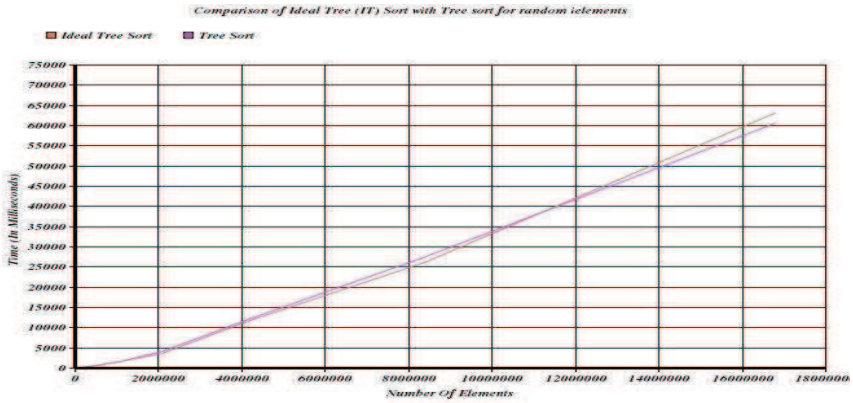


Fig 3: Ideal tree sort on random Data Vs sorted data.

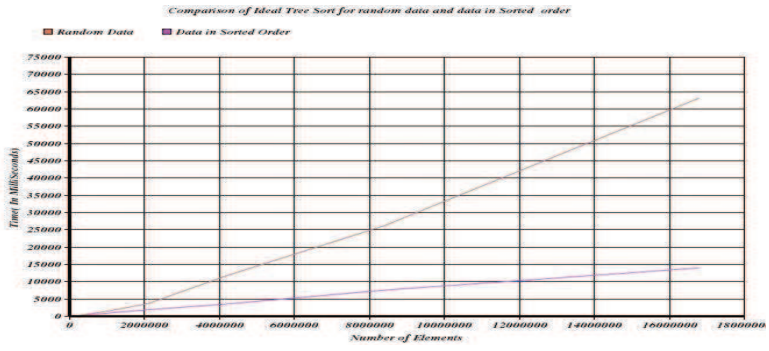


Table 1: Time (MilliSeconds) taken by IT sort and Tree sort to sort random data.

NumberOf Elements	IT Sort	Tree sort
2047	0	0
4095	1	1
8191	3	3
16383	7	10
32767	25	27
65535	65	45
131071	113	154
262143	290	366
524287	701	619
1048575	1559	1533
2097151	3670	4156
4194303	11767	12221
8388607	26192	27512
16777215	63806	60554

Number Of Elements	IT Sort	Tree Sort
2047	5	26
4095	6	97
8191	10	381
16383	11	1233
32767	16	5833
65535	38	20533
131071	60	80908
262143	244	351177

Table 2: Time (Milliseconds) taken by IT Sort and Tree Sort to sort ascending ordered data.

Number of Elements	IT Sort	Tree Sort
2047	6	24
4095	7	99
8191	11	254
16383	13	2505
32767	25	5110
65535	43	20912
131071	81	82034
262143	247	367554

ISBN - 978-93-81583-55-5

Table 3: Time (Milliseconds) taken by IT Sort and Tree Sort to sort Descending ordered data.

NumberOfElements	Random Data	Data inAscending Order	Data in Descending Order
2047	0	5	6
4095	1	6	7
8191	3	10	11
16383	7	11	13
32767	25	16	25
65535	65	38	43
131071	113	60	81
262143	290	244	247
524287	701	400	412
1048575	1559	843	946
2097151	3670	1860	1901
4194303	11767	3229	3554
8388607	26192	7218	7557
16777215	63086	13832	13985

Table 4: Comparison of Time (Milliseconds) taken by IT Sort to sort data which is in random order, sorted In ascending and sorted in descending order.

5. CONCLUSION

IT Sorting algorithm sorts the elements in two steps. In first step all the elements are preprocessed by constructing middle heap. In second step normal tree sort is applied on the preprocessed data. This IT sort algorithm improves the worst case behavior of tree sort and it reduces worst case behavior O(n*n) to O(n logn). Implementation results show that IT Sort algorithm works better than Tree sort in all aspects.

6. REFERENCES

1. Cormen T., Leiserson C., Rivest R., and Stein C., Introduction to Algorithms, McGraw Hill, 2001.
2 .Aho A., Hopcroft J., and Ullman J., The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.
3. Astrachanm O., Bubble Sort: An Archaeological Algorithmic Analysis, Duk University, 2003.
4 Bell D., "The Principles of Sorting," Computer Journal of the Association for Computing Machinery, vol. 1, no. 2,pp. 71-77, 1958.
5. Box R. and Lacey S., "A Fast Easy Sort," Computer Journal of Byte Magazine, vol. 16, no. 4, pp. 315-315,1991.
6. Deitel H. and Deitel P., C++ How to Program, PrenticeHall, 2001.
[7] Friend E., "Sorting on Electronic Computer Systems," Computer Journal of ACM, vol. 3, no. 2, pp. 134-168, 1956.
[8] Knuth D., The Art of Computer Programming, AddisonWesley, 1998
[9] Ledley R., Programming and Utilizing DigitalComputers, McGraw Hill, 1962.

[10] Levitin A., Introduction to the Design and Analysis of Algorithms, Addison Wesley, 2007.

[11] Nyhoff L., An Introduction to Data Structures, Nyhoff Publishers, Amsterdam, 2005.

[12] Organick E., A FORTRAN Primer, Addison Wesley, 1963.

[13] Pratt V., Shellsort and Sorting Networks, GarlandPublishers, 1979.

[14] Sedgewick R., "Analysis of Shellsort and RelatedAlgorithms," in Proceedings of the 4th Annual European Symposium on Algorithms, pp. 1-11, 1996.

[15] Seward H., "Information Sorting in the Application of Electronic Digital Computers to usiness Operations," Masters Thesis, 1954.

[16] Shell D., "A High Speed Sorting Procedure,"Computer Journal of Communications of the ACM, vol. 2, no. 7, pp. 30-32, 1959.

* * * *

---

[1]*Author 1:Hari Krishna Gurram,MTech (CS), UCEK, JNTUK.*
harikrishna553@gmail.com

[2]*Author 2:Dharmaiah Gurram,*
Asst Professor., K L Univversity,Vaddeswaram
dharma.g2012@kluniversity.in