International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-3 Issue-1, October 2013

A Novel Sorting Technique to Sort Elements in Ascending Order

Debabrata Swain, G.Ramkrishna, Hitesh Mahapatra, Pramoda Patro, Pravin M.Dhanrao

Abstract- Sorting is an operation to arrange the elements of a data structure in some logical order. In our daily lifes, without knowing about sorting we are doing works in sorted order. So that's why everybody must need an efficient sorting technique which will solve sorting problem with in limited time. So We have discussed about various existing sorting algorithms with their advantage and disadvantage. In this paper, we have proposed a new sorting algorithm which overcomes some common disadvantage of some traditional existing algorithms by properly utilizing the memory. Here, we have compared our algorithm with traditional existing algorithms by using some factors.

Keywords- Various sorting algorithms. Bubble sort, Selection sort, Insertion sort and Quick sort

I. INTRODUCTION

Sorting is the most fundamental algorithmic problem in computer science and a rich source of programming problems for two distinct reasons. First, sorting is a useful operation which efficiently solves many tasks that every programmer encounters. As soon as you recognize your job is a special case of sorting, proper use of library routines make short work of the problem. Second, literally dozens of different sorting algorithms have been developed, each of which rests on a particular clever idea or observation. Most algorithm design paradigms lead to interesting sorting algorithms, including divide-and-conquer, randomization, incremental insertion, and advanced data structures. Many interesting programming/mathematical problems follow from properties of these algorithms. Sorting is an efficient technique which performs the task to arrange the elements in ascending or descending order. Sorting technique is generally used in our day to day life as well as in many computer applications. E.g.- A teacher keeps the answer book of the students in sorted order by applying sorting technique on roll no. Suppose, we may regard a telephone directory as a list, each record having three fields: name, address, and phone number. We may wish to locate the record corresponding to a given number, in which case the phone number field would be key. One of the sorting technique is helpful in order to locate the record by using key. In a database, sorting technique sometimes used to arrange the records by considering one field of the record. Typically, sorting technique arranges the element from highest to lowest or lowest to highest.

Manuscript published on 30 October 2013.
***** Correspondence Author (s)
Prof. Debabrata Swain, received his B.Tech in Computer Science and Engineering from RIT, Berhampur, India
Prof G.Ramakrishna, received his B.Tech in Computer Science and Engineering from RNEC Ongole, Affliated to JNTU, Hyderabad,India,
Prof Hitesh Mohapatra, received his B.Tech in Information Technology from GIET, Gunupur, BPUT,India
Prof Pramoda Patro, received his M.Sc in Mathematics from Khallikote Autonomous College, Berhampur, India
Prof. P. M. Dhanrao has completed B.E. (Computer) From University of Pune, India.

Most sorting algorithms uses a common two steps i.e., compare two elements and exchange them in sorted order. Sorting algorithm are mainly used to manage the data. In some cases, a large number of records can be sorted by applying sorting on a particular common field of each record. That common field is called key. While searching any element by using binary search, we must sort the elements in ascending order before. We characterize sorting methods into two broad categories: i) internal methods(i.e., methods to be used when the list to be sorted is small enough so that the entire sort can be carried out in main memory).ii) external methods (i.e., methods to be used on larger lists).Our proposed algorithm comes under the internal method.

1.1 USES AND APPLICATIONS OF SORTING

We have two important uses of sorting:
1) as an aid in searching
2) as a means for matching entries in lists.

1.2 APPLICATIONS

Sorting also finds application in the solution of many other more complex problems from areas such as optimization, graph theory and job scheduling.Consequently,the problem of sorting has great relevance in the study of computing.
The key to understanding sorting is seeing how it can be used to solve many important programming tasks:

1.2.1 Uniqueness Testing

How can we test if the elements of a given collection of items S are all distinct? Sort them into either increasing or decreasing order so that any repeated items will fall next to each other. One pass through the elements testing if $S[i] = S[i + 1]$ for any $1 \leq i < n$ then finishes the job.

1.2.2 Deleting Duplicates

How can we remove all but one copy of any repeated elements in S? Sort and sweep again does the job. Note that the sweeping is best done by maintaining two indices — *back*, pointing to the last element in the cleaned-out prefix array, and *i*, pointing to the next element to be considered. If $S[back] <> S[i]$, increment *back* and copy $S[i]$ to $S[back]$.

1.2.3 Prioritizing Events

Suppose we are given a set of jobs to do, each with its own deadline. Sorting the items according to the deadline date (or some related criteria) puts the jobs in the right order to process them. Priority queue data structures are useful for maintaining calendars or schedules when there are insertions and deletions, but sorting does the job if the set of events does not change during execution.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

1.2.4 Median/Selection

Suppose we want to find the *k*th largest item in set *S*. After sorting the items in increasing order, this fellow sits in location *S*[*k*]. This approach can be used to find (in a slightly inefficient manner) the smallest, largest, and median elements as special cases.

1.2.5 Frequency Counting

Which is the most frequently occurring element in *S*, i.e., the mode? After sorting, a linear sweep lets us count the number of times each element occurs.

1.2.6 Reconstructing the Original Order

How can we restore the original arrangement of a set of items after we permute them for some application? Add an extra field to the data record for the item, such that the *i*th record sets this field to *i*. Carry this field along whenever you move the record, and later sort on it when you want the initial order back.

1.2.7 Set Intersection/Union

How can we intersect or union the elements of two containers? If both of them have been sorted, we can merge them by repeatedly taking the smaller of the two head elements, placing them into the new set if desired, and then deleting the head from the appropriate list.

1.2.8 Finding a Target Pair

How can we test whether there are two integers *x*, *y*,*z* such that $x + y = z$ for some target *z*? Instead of testing all possible pairs, sort the numbers in increasing order and sweep. As *S*[*i*] increases with *i*, its possible partner *j* such that $S[j] = z - S[i]$ decreases. Thus decreasing *j* appropriately as *i* increases gives a nice solution.

1.2.9 Efficient Searching

How can we efficiently test whether element *s* is in set *S*? Sure, ordering a set so as to permit efficient binary search queries is perhaps the most common application of sorting. Just don't forget all the others!

Simplest sorting algorithm

Idea:

1. Set flag = false
2. Traverse the array and compare pairs of two elements
   - 1.1 If E1 ≤ E2 - OK
   - 1.2 If E1 > E2 then Switch(E1, E2) and set flag = true
3. If flag = true goto 1.

Here, we have discussed some existing algorithms a like Bubble sort, insertion sort, selection sort and Quick sort used for sorting the elements of an array. Each sorting algorithms having some advantages as well as disadvantages that we will discuss in next section. In this paper, we have considered proper utilization of memory and also the simplicity of the algorithm. By taking these two factors, we compared existing algorithms with our proposed algorithm.

II. LITERATURE SURVEY

2.1 Bubble sort:

Simplest sorting algorithm

Idea:

1. Set flag = false
2. Traverse the array and compare pairs of two elements
   - 1.1 If E1 ≤ E2 - OK
   - 1.2 If E1 > E2 then Switch(E1, E2) and set flag = true
3. If flag = true goto 1.

Example:

```
    1   1  23  2  56  9   8   10   100
    2   1  2   23 56  9   8   10   100
    3   1  2   23  9  56  8   10   100
    4   1  2   23  9  8   56  10   100
    5   1  2   23  9  8   10  56   100
---- finish the first traversal ----
----        start again          ----

    1  2   23   9   8   10  56   100
    1   1  2   9   23  8   10  56   100
    2   1  2   9    8  23  10  56   100
    3   1  2   9    8  10  23  56   100
---- finish the second traversal ----
----        start again          ----
```

2.2 Insertion-Sort

One of the simplest sorting algorithms is the insertion sort. Insertion sort consists of *n* - 1 *passes.* For pass *p* = 2 through *n,* insertion sort ensures that the elements in positions 1 through *p* are in sorted order. Insertion sort makes use of the fact that element in positions 1 through *p* - 1 are already known to be in sorted order. Figure 7.1 shows a sample file after each pass of insertion sort.

Figure 2.1 shows the general strategy. In pass *p,* we move the pth element left until its correct place is found among the first *p* elements. The code in Figure 7.2 implements this strategy. The sentinel in *a*[0] terminates the while loop in the event that in some pass an element is moved all the way to the front. Lines 3 through 6 implement that data movement without the explicit use of swaps. The element in position *p* is saved in tmp, and all larger elements (prior to position *p*) are moved one spot to the right. Then tmp is placed in the correct spot. This is the same technique that was used in the implementation of binary heaps.

Original	34	8	64	51	32	21	Positions Moved
After *p* = 2	8	34	64	51	32	21	1
After *p* = 3	8	34	64	51	32	21	0
After *p* = 4	8	34	51	64	32	21	1
After *p* = 5	8	32	34	51	64	21	3
After *p* = 6	8	21	32	34	51	64	4

Figure 2.1 Insertion sort after each pass

2.3 Selection Sort

Insert elements in a priority queue implemented with an unsorted sequence remove them one by one to create the sorted sequence

For example, to sort a list of integers into ascending order, we do the following:

a) Scan the entire list to find the smallest value (selecting the smallest value).

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

b) Exchange that value with the value in the first position of the list.
c) Treat the rest of the values (from second position till the last) as a new list and repeat the first two steps until the list is exhausted.

- Example

	Original	34	8	64	51	32	21
Moved							
	After p=1	*8*	*34*	64	51	32	21
1							
	After p=2	*8*	*34*	*64*	51	32	21
0							
	After p=3	*8*	*34*	*51*	*64*	32	21
1							
	After p=4	*8*	*32*	*34*	*51*	*64*	21
3							
	After p=5	*8*	*21*	*32*	*34*	*51*	*64*
4							

2.4 Quick Sort

Another divide-and-conquer sorting algorithm. To understand quick-sort, let's look at a high-level description of the algorithm

1) *Divide*

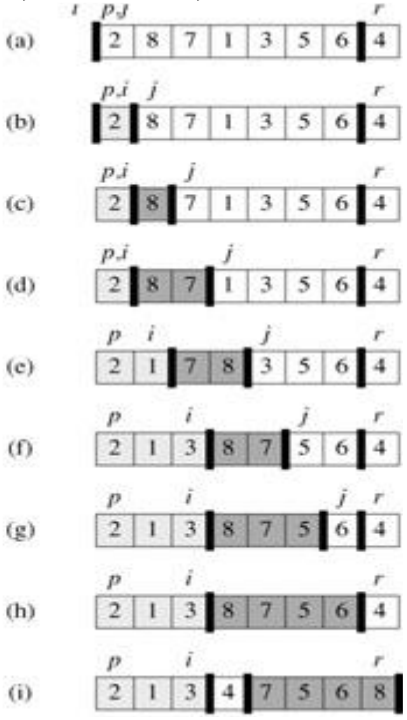
If the sequence S has 2 or more elements, select an element x from S to be your **pivot**. Any arbitrary element, like the last, will do. Remove all the elements of S and divide them into 3 sequences:
L, holds S's elements less than x
E, holds S's elements equal to x
G, holds S's elements greater than x

2)Recurs

Recursively sort L and G

3)Conquer

Finally, to put elements back into S in order, first inserts the elements of L, then those of E, and those of G.



III. PROPOSED ALGORITHM

Here, First, before implementing the algorithm, we asking the user to enter the maximum value for an element to prompt by user. Dynamically create the array, whose size is equal to maximum value entered by user. Then, we assign 0 in every position of the array. Then user is asked to enter the number of elements, which is less than or equal to size of array. While reading each element, algorithm need to check the value entered by the user less than or equal to maximum value of an array or not. If the entered value is less than or equal to maximum value, then only it will stored in array. Otherwise, it will shown an error message "invalid value entered", then user is asked to enter valid value again. The value is assigned must be same as the respective position of the array. Here our proposed algorithm only considered for unique values. After all values entered by user. Our algorithm will place all non-zero values in sequence order starting from 0^{th} index to $n\text{-}1^{th}$ index. After that we find all the elements are arranged in sorted order(Ascending order). Then we will keep the memory i.e., required to hold the non-zero values present in the array and then release the extra memory present in the array using dynamic memory allocation.

Steps for Proposed Algorithm

Sort(ptr,m,n)

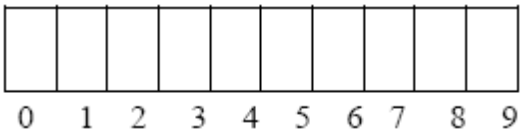
ptr—Integer type pointer used to point to the starting location of the dynamically Allocated memory.
m—Maximum value among all array elements
n—Number of nonzero values entered by user (n<=m)

1) Read maximum value among all array elements i.e., m
2) Create an array dynamically for m-elements using ptr.
3) Assign 0 to all m-elements of Array.
4) Read value of n.
5) for i←0 to n-1
   do
6) Read non-value 'v'
7) if v<=m then
8) *(ptr+(v-1))=v
9) else
10) Display "invalid value entered ,enter again valid value"
11) goto step 7
   [end-loop]
12) Assign variable c to 0
13) for i←0 to m-1
   do
14) if(*(ptr+i)!=0)
   then
   a) *(ptr+c)=*(ptr+i)
   b) c=c+1
   [end-if]
   [end-loop]
15) Now release the m-n memory locations

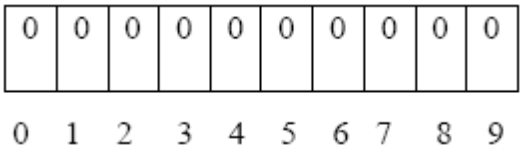
Here we have shown example to Sort values 10, 7,5,3,9 in ascending order using our proposed algorithm Sort values 10, 7,5,3,9 in ascending order

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

1)Here M=10 (Max value among all array elements = Size of Array)



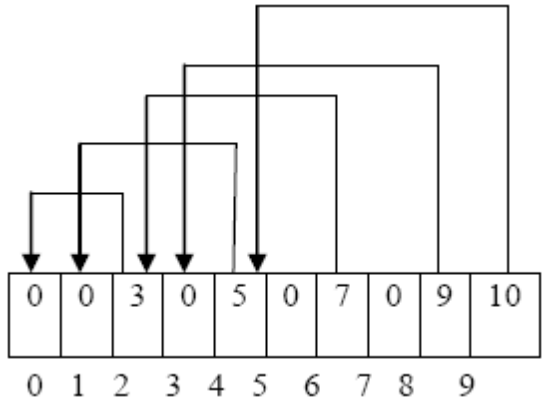
2) Assigning "0" to each array location



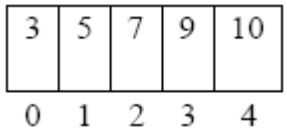
3) Put elements in to the array using the rule arr[v-1]=v

arr[10-1]=10
arr[9]=10

4) Moving all Non-Zero value to sequential indices



5) After releasing the space of (m-n) elements



IV. PERFORMANCE ANALYSIS

Here after seeing the performance of different sorting algorithms we come to a conclusion that our algorithm is better than all traditional algorithms in terms of memory utilization. As we have used the dynamic memory in our algorithm. No traditional algorithm talks about proper memory utilization. Here we have used Random access technique while putting the elements at appropriate positions in the array which faster than sequential access that is used by other traditional algorithms. Also we found that our algorithm is easy to understand and also user friendly as compared to all traditional algorithms. When we are providing limited no of inputs we found that our algorithm is working efficiently as compared to other algorithms. But when no. of inputs are large at that time quick sort is showing good performance. Our algorithm is considering the unique inputs for the array to be sorted. If we can use this algorithm for different applications where no. Of inputs are less, and then we can get a better performance in that particular application.

V. CONCLUSION

Here we have shown our algorithm by using which we can sort the different array elements in Ascending order. Our algorithm is only working for unique elements. Here we have also shown how memory is properly utilized in our policy using dynamic memory allocation. By using this we have released the unused memory after arranging the elements in sorted order.

VI. FUTURE WORK

Here in this algorithm we have only considered unique inputs for the algorithm. So in recent future we are trying to develop the algorithm so that it will work for duplicate inputs. Also in recent future we want to introduce some new concepts in our algorithm so it will so better performance when no. Of inputs are more. Here we have discussed less about the performance of the algorithms in terms of time complexity that we want to do in the future work.

VII. ACKNOWLEDGEMENT

This work is only complete due to the continuous encouragement and guidance of our esteemed Principal Dr.D.N.Kyatanvar(SRES COE,KOPARGAON)and Mr. A. A. Barbind Head of Department, Information Technology Department. We are also thankful to all our colleagues and specially to SRES College Of Engineering ,Kopargaon for providing us with support and technical guidance.

REFERENCES

[1] http://en.wikipedia.org/wiki/Sorting_algorithm
[2] http://en.wikipedia.org/wiki/Selection_sort
[3] http://en.wikipedia.org/wiki/Bubble_sort
[4] http://en.wikipedia.org/wiki/Insertion_sort
[5] http://en.wikipedia.org/wiki/Selection_sort
[6] http://en.wikipedia.org/wiki/Quicksort
[7] http://en.wikipedia.org/wiki/Merge_sort
[8] http://www.cs.manchester.ac.uk/ugt/COMP26912/lecture/lecture-sorting.pdf
[9] http://www.cs.ucf.edu/courses/cop3502/nihan/spr03/sort.pdf
[10] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, fifth Indian printing (Prentice Hall of India private limited), New Delhi-110001
[11] Computer Algorithms by Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, Galgotia publications,5 Ansari road, Daryaganj, New Delhi-110002
[12] C.A.R. Hoare, Quick sort, Computer Journal, Vol. 5, 1, 10-15 (1962)
[13] P. Hennequin, Combinatorial analysis of Quick-sort algorithm, RAIRO: Theoretical Informatics and Applications, 23 (1988), pp. 317–333
[14] Lecture Notes on Design & Analysis of Algorithms G P Raja Sekhar Department of Mathematics I I T Kharagpur

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

[15] Knuth D. (1997) "The Art of Computer Programming, Volume 3: Sorting and Searching'', Third Edition. AddisonWesley, 1997. ISBN 0-201-89685-0. pp. 138–141, of Section 5.2.3: Sorting by Selection
[16] Let Us C by Yashvant Kanethkar, 8th edition (BPB publications).b-14 Connaught place, New Delhi-110001
[17] MERRITT S. M. (1985), "An inverted taxonomy of Sorting Algorithms. Programming Techniques and Data Structures", Communications of ACM, Vol. 28, Number 1, ACM

Prof Debabrata Swain, received her B.Tech in Computer Science and Engineering from RIT, Berhampur, India, in June 2008, and M.Tech in Computer Science from Berhampur University, India in June 2011.He has 4.5 years of experience. Currently he is working as an Asst. Professor in Department Information Technology at SRES college of Engineering, Kopargaon, Maharashtra India .His prime research interest includes System architecture and performance evaluation, Image processing, Microprocessors,Data structure. He has 5 International Publications excluding this work.

Prof G.Ramakrishna, received his B.Tech in Computer Science and Engineering from RNEC Ongole, Affliated to JNTU, Hyderabad,India, in June 2006, and M.Tech in Computer Science and Engineering from Nagarjuna University, India in June 2010.He has 4.10 years of experience. Currently he is working as an Asst. Professor in Department Information Technology at SRES college of Engineering, Kopargaon, Maharashtra India .His prime research interest includes Data Ware Housing & Data Mining, Image processing, Data structure. He has 2 Professional Certifications excluding this work.

Prof Hitesh Mohapatra, received his B.Tech in Information Technology from GIET , Gunupur, BPUT,India, in June 2006, and M.Tech in Computer Science Engineering from CET,Bhubaneswar, BPUT, India in June 2009.He has 6 years of experience. Currently he is working as an Asst. Professor in Department Computer Engineering at SRES college of Engineering, Kopargaon,Maharashtra ,India .His prime research interest includes Software Engineering and Computer Security, Bioinformatic, Data structure. He has 1 International Publications excluding this work.

Prof Pramoda Patro, received her M.sc in Mathematics from Khallikote Autonomous College, Berhampur, India, in June 2007, and M.Tech in Computer Science from Berhampur University, India in June 2009.He has 5.3 years of experience. Currently he is working as an Asst. Professor in Department of Computer Engineering at Amrutvahini college of Engineering, Sangamner, Maharashtra India .His prime research interest includes Fuzzy logic, neural network and Artificial Intelligence. He has 1 International Publications excluding this work.

Prof. P. M. Dhanrao has completed B.E. (Computer) From University of Pune in 2008. He is having total 5.2 years of experience (Industrial + Educational). He is Pursuing M. Tech (CSE) From Rajiv Gandhi University (RGPV). He has presented various papers at National Conference and at National Level Paper presentation. He has attended various workshops Of IIT Bombay, NITTR MHRD GOI, MSBTE Mumbai and cyber Forensics of total 4 Weeks and 4 days. He has attended various seminars organized by University Of Pune. He has worked as Controller Of Examinations. He has judged Various Competitions.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)