IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-12 E-ISSN: 2347-2693

An Adaptive Sorting Algorithm for Almost Sorted List

Rina Damdoo[1]*, Kanak Kalyani[2]

[1*] Department of CSE, RCOEM, Nagpur, India
[2] Department of CSE, RCOEM, Nagpur, India

*Corresponding Author: damdoor@rknec.edu, Tel.: +91-9324425240

Available online at: www.ijcseonline.org

Received: 20/Nov/2017, Revised: 29/Nov/2017, Accepted: 17/Dec/2017, Published: 31/Dec/2017

***Abstract*—**Sorting algorithm has a great impact on computing and also attracts a grand deal of research. Even though many sorting algorithms are evolved, there is always a scope of for a new one. As an example, Bubble sort was first analyzed in 1956, but due to the complexity issues it was not wide spread. Although many consider Bubble sort a solved problem, new sorting algorithms are still being evolved as per the problem scenarios (for example, library sort was first published in 2006 and its detailed experimental analysis was done in 2015) [11]. In this paper new adaptive sorting method Paper sort is introduced. The method is simple for sorting real life objects, where the input list is almost but not completely sorted. But, one may find it complex to implement when time-and-space trade-off is considered.

***Keywords*—**Adaptive sort, Time complexity, Paper sort, Sorted list, Un-sorted list

I. INTRODUCTION

A sorting algorithm is known as adaptive, if it takes benefit of already sorted elements in the list i.e. while sorting if the source list has few already sorted elements, taking this into consideration, adaptive algorithms will not try to re-arrange those. Insertion sort has been time-honoured as a well known and established optimal-most comparison-based sorting algorithm for small arrays [1], [2], [4], [10].

Non-adaptive algorithms do not consider already sorted elements in the list. They compel each element to be re-arranged to confirm their final position in the list. To achieve this position, elements may incur swapping through and fro.

Paper sort combines advantages of Insertion sort and Merge sort. The name Paper sort has been given to this new sorting algorithm because the way of sorting is similar to the way, students' examination Answer papers are arranged roll number wise.

II. PROPOSED APPROACH

Given a list of numbers, in paper sort we split the list into two lists sorted and unsorted. Consider the first number as the key element of the sorted list SL (say K) which is initially empty. Then the second position element (say X) is compared with K, and is added at end to sorted list if it is greater than K. If it is not, it is added to unsorted list USL which is initially empty.

Now the third position element is compared with the last (say L) number in the sorted list and is added at end to sorted list if it is larger than L, otherwise it is added at the end of unsorted list USL. This procedure is continued until all the elements are scanned and we get two lists sorted list SL and unsorted list USL. Recursively above procedure is repeated on USL until we get an empty USL.

Now there are multiple sorted lists which can be merged in the reverse order they are generated i.e. last and second last lists are first merged to get a temporary list. Then temporary and third last list are merged to give a new temporary list and so on finally to get a single SL. Paper sort with some random numbers is explained in Figure 1.

III. IMPLEMENTATION

- **Splitting Procedure**

While implementing a program for this sort in C language, I found that shifting of numbers in an array in order to make space for next number in sorted sequence is a time consuming task. So without changing the original method little changes can be done to it, so that shifting can be avoided. This can be done by swapping the numbers X and the number getting added to sorted list where X is number in array next to position where sorted list ends. Explanation provided below is actual execution of algorithm after every pass.

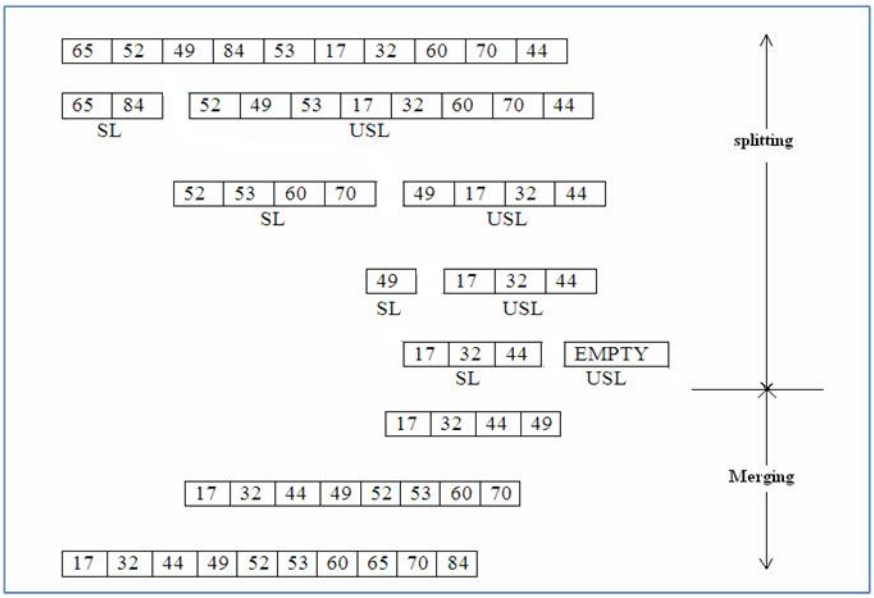

Figure1: Demonstration of Paper sort

65	52	49	84	53	17	32	60	70	44

Pass 1 starts

65									
65	52								
65	52	49							

Now when 84 is scanned it must be added after 65. In order to make space for 84, instead of shifting 52 and 49 in array we swap 52 and 84 to get

65	84	49	52						

Remaining numbers are all added as it is to get

65	84	49	52	53	17	32	60	70	44

PASS 1 ends

PASS 2 starts for numbers 49, 52,53,17,32,60,70,44

65	84	49	52	53	17	32			

Now when 60 is scanned it must be added after 53 so swap 17 and 60 to get

65	84	49	52	53	60	70	17		

Now when 70 is scanned it must be added after 60 so swap 32 and 70 to get

65	84	49	52	53	60	70	17	32	
65	84	49	52	53	60	70	17	32	44

PASS 2 ends

PASS 3 starts for numbers 17, 32, 44

65	84	49	52	53	60	70	17		
65	84	49	52	53	60	70	17	32	
65	84	49	52	53	60	70	17	32	44

PASS 3 ends with empty USL and three sorted lists (65, 84), (49, 52, 53, 60, 70), (17, 32, 44)

- **Merging procedure**

Merging adjacent runs is done with the help of temporary memory. Algorithm uses a temporary memory equal to size of array. Then, it merges elements of last two generated

sorted lists into temporary array (say B). After merging all elements of B are copied to original array. Question here is as the list is not uniformly divided into sub lists, how sub lists bounds must be remembered. Answer is very simple, we have to sacrifice a little on memory and use one more array (say BOUND), which will store the sub lists bounds. A simple merge algorithm runs left to right or right to left depending on which run is smaller on the temporary memory and original memory of the sub list. The final sorted run is stored in the original memory of the initial runs. Paper sort searches for appropriate element in some order and takes advantage of already positioned elements, so it is an adaptive algorithm.

IV. PSEUDO CODE FOR PAPER SORT

Given Figure 2 is Pseudo code for Paper Sort. This algorithm has been executed on 20 integers in C language as shown.

V. TIME COMPLEXITY MEASUREMENT

1. Best case: If the list is already arranged in required sequence then Paper sort has best case time complexity $\Omega(n)$. Insertion sort also makes use of the already sorted elements but it works well for small set of numbers.

2. Worst case: Worst case will occur when the list is exactly inversely arranged. Paper sort has Worst case time complexity $O(n^2)$. This algorithm is not intended to be used by application where data is not almost sorted.

3. Average case: In average case Paper sort has average case time complexity $\Theta(n \log n)$. When space utilization is major concern, space complexity will not be comparable with merge sort.

VI. CONCLUSION AND FUTURE WORK

Every sorting algorithm has some advantages and disadvantages. Space and Time complexity trade-off is always under consideration.

Proposed algorithm, Paper sort, can be applied to data which is almost sorted. Paper sort is an easy to understand method as compared to heap sort, quick sort. If the list is almost sorted, paper sort is a better option than merge sort. But it requires extra memory to save the Bounds of lists, which can be considered as drawback of this method. Even if proposed sorting algorithm is not a divide and conquer algorithm, number of passes required to sort N elements are not necessarily N-1. Comparative analysis of Paper sort and Merge sort is depicted in Table 1.

Three specific types of psychological complexity that affect a programmer's ability to comprehend software have been identified as problem complexity, system design complexity, and procedural complexity. The proposed algorithm can be refined to reduce the space complexity [3], [5], [6].

```
void PaperSort()
{
    int A[20]={65,52,49,84,53,17,32,60,70,44};
    int B[20], BOUND[10];
    int i=0, j=0, k, t, p, q;
    while (i<20)
    {
        for (k=i+1; k<20; k++)
          if (A[k] > A[i])
          {    t= A[k];
               A[k]= A[i+1];
               A[i+1] = t;
               i++;
          }
          BOUND[j++]= i;
           i++;
    }
    j--;
    while (j >= 1)
    {    k=0;
         p= (j-2>=0 ? BOUND[j-2] + 1:0);
         q=BOUND[j-1] + 1;
         while (p <= BOUND[j-1] && q<20)
         {
             if ( A[p] <= A[q])
                  B[k++]= A[p++];
             else
                  B[k++] = A[q++];
         }
         while (p <= BOUND[j-1])
             B[k++] = A[p++];
         while (q<20)
             B[k++] = A[q++];
         for (i=0; i<k; i++)
             A[BOUND[j-2] + 1 + i] = B[i];
         j--;
 }
 for (k=0; k<20; k++)
         printf("%d ", B[k]);
}
```

Figure 2: Pseudo code for Paper Sort

Table 1: Comparative analysis of Paper sort and Merge sort

S. No	Parameter for comparison	Paper sort	Merge Sort
1	Number of Passes	7	12
2	Number of comparisons	21	13

References

[1] E. Horowitz, S. Sahani, "Fundamentals of Computer Algorithms", Computer Science Press, Rockville, Md., 1998.

[2] D. Knuth, "The Art of Computer Programming", volume 3 "Sorting and searching", Second edition, Assison Wesley, 1998.

[3] C.L. Liu, "Analysis of Sorting Algorithms", Proceedings of Switching and Automata Theory, 12th Annual Symposium, East Lansing, MI, USA, pp. 207-215, 1971.

[4] M. Devi, S. Charaya, "Enhancing the Efficiency of Radix sot by using clustering Mechanism: A Review", IJSRD, Volume 4 Issue 5, pp.847- 850, 2016.

[5] J. Darlington, "A synthesis of several sorting algorithms", Acta Informatica II, Springer-Verlag, pp. 1-30, 1978.

[6] John Darlington, Remarks on "A Synthesis of Several Sorting Algorithms", Springer Berlin / Heidelberg", Volume 13, March 1980, pp. 225-227.

[7] A. Andersson, T. Hagerup, S. Nilsson, R. Raman, Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, 1995.

[8] V. Paul, "Entropy, Search, Complexity- Algorithms by Kolmogorov Complexity (A Survey)", Bolyai Society Springer, pp. 209-232, 2007

[9] R. Harter, "A Computer Environment for Beginners' Learning of Sorting Algorithms: Design and Pilot Evaluation", ERIC, Journal Number 795978, Computers & Education, volume 51 No.2, pp. 708-723, 2008

[10] A. Bharadwaj, S. Mishra, "Comparison of Sorting Algorithms based on Input Sequences", International Journal of Computer Applications, Volume 78 No.14, pp.7-10, 2013

[11] N. Faujdar, S. Ghrera, "A Detailed Experimental Analysis of Library sort Algorithm", INDICON, IEEE, pp. 1-6, 2015

Authors Profile

Prof. Rina Damdoo received the Masters in Technology in Computer Science and Engineering from Nagpur University in 2012. Currently she is serving as Assistant Professor at Shri Ramdeobaba College of Engineering and Management, Nagpur. She has a teaching experience of 15 years with expertise in subjects like Data Structures, Operating Systems, System Software, Design Patterns. Her research interest includes N-grams, Statistical Machine Learning and sentiment analysis in the domain of Template messaging.
Email: damdoor@rknec.edu

Prof. Kanak Kalyani received the Masters in Technology in Computer Science from Visvesvaraya National Institute of Technology, Nagpur (VNIT Nagpur) in 2010. Currently she is serving as Assistant Professor at Shri Ramdeobaba college of Engineering and Management, Nagpur and has a teaching experience of 7 years. She has expertise in subjects like Data Structures, Object oriented Programming, Mobile Application Programming and Salesforce Technology. Her research interest includes Business Intelligence and Traffic Mining.
Email: kalyanik@rknec.edu

