RESEARCH ARTICLE OPEN ACCESS

Behavior Analysis of Memorized Sort on Randomly Colonized Data Sets

Toqeer Ehsan*, M. Usman Ali**, Meer Qaisar Javed**
*(Department of Computer Science, University of Gujrat, Pakistan)
**(Department of Information Technology, University of Gujrat, Pakistan)

ABSTRACT
Memorized sort performs sorting by computing sorted sub-sequences of different sizes in a random data set. Each sub-sequence can be considered as a data colony populated with random numbers. These colonies have different population of elements due to their random nature. Increased population of colonies can give better performance of memorized sort algorithm. The running time of memorized sort may vary as the entropy of the colonized list is changed. This paper shows the behavior of memorized sort on colonized random data sets with increased population.

Keywords – Colonization, Complexity, Hybrid Search, Memsort, Sub-sequence

I. INTRODUCTION

Divide and conquer is an efficient technique to sort the given set of elements because it divides the problems into smaller sub-problems [2][10]. These algorithms can be designed in top-down as well as bottom-up fashion [5]. Memorized sort is a sorting algorithm that solves the sorting problem in a bottom-up mechanism [5]. It is a divide and conquer algorithm based on dynamic programming [5]. Dynamic programming is used to solve optimization problems by memorizing the solutions of overlapping sub-problems [2][10]. Memorized sort computes the pre-sorted sub-sequences and combines them by using the memorizations [5]. Memorized sort performs well on randomized data sets [5]. Random data set can be further divided in the form of independent data sets on the basis of their random nature. We call these divisions as data colonies [4].

Data colonies are autonomous data sets that could be treated independently [4]. We can apply any sorting or searching technique to any data colony [4]. Memorized search is a searching technique that finds elements from a list when data is colonized randomly [4]. Colonies are computed only once and a method of memorizations is introduced which is based on dynamic programming [4]. It performs efficient searching on random data. Memorized search can be implemented in different fashions like sequential memorized search, binary memorized search and hybrid memorized search [4]. Same colonization technique is used in memorized sort and it performs better then merge sort [5]. Memorized sort is an efficient sorting algorithm which performs sorting efficiently on random data just like quick sort [5]. Unlike quick sort, worst case of memorized sort is similar to merge sort algorithm [5]. The algorithm may change its behavior when there is random data with diverse density of colonies. This paper focuses on the behavior analysis of memorized sort when data is colonized randomly. Next section describes the process of memorized sorting and memorized searching in little more detail.

II. PREVIOUS WORK

Before implementation of memorized sort on colonized data, let's have a look on the process of memorized sort [5], colonization of data [4] and memorized search [4].

2.1 MEMORIZED SORT

Memorized sort also known as Memsort, is an algorithm which is based on dynamic programming and works in divide and conquer nature [5]. The implementation of the algorithm is of bottom-up nature [5]. Steps of the algorithms are:
Step1: Compute the sorted sub-sequences and memorize them.
Step2: Combine the sub-sequences with bottom-up approach.
Step3: Repeat step2 until there is only one sequence.

First step computes the sorted sub-sequences be checking every element with next element in the array. If next element is greater than previous then both elements belong to the same sub-sequence and vice versa. All the computed sub-sequences are memorized in a table, typically an array. Sub-sequences are not saved as a whole but the starting and ending of the sequence. For example let's consider the follow array 'A' of elements.

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	6	8	13	2	6	1	7	9	5	8

After computing sorted sub-sequences we have to memorize the indexes, so memorized sort creates a new array 'R' for memorizations. After first step, array 'R' would look like.

Index	0	1	2	3	4	5	6
Value	0	2	4	7	9	-1	

Starting value is '0' which is obvious that the first sequence starts with '0' index. Ending of the sequence is the next value of the array which is '2' at R[1]. It means there are three elements in first sub-sequence. The last element of the array is -1 which shows the end of array. The length of the array 'R' depends on the size of the input array. If size of input array is 'n' then the length of 'R' must be at least n/2+2. Once all the sub-arrays are memorized, algorithm starts combining them. In our example there are four sub-sequences, [6,8,13], [2,6], [1,7,9] and [5,8]. Memorized sort combines first two sequences then last two sequences. After first iteration, we would have half number of sequences. Algorithm repeats this step until there is only one sub-sequence left which is actual array but sorted. Table.1 shows the values of 'R' after multiple iterations.

Table.1: Values of array 'R'

Index	0	1	2	3	4	5	6
Value	0	2	4	7	9	-1	
I-1	0	4	9	-1			
I-2	0	9	-1				

Memorized sort is an efficient technique which gives better performance on random data but the complexity class remains similar as merge sort. Upper bound to memorized sort is O(nlgn) but it performs much less number of comparisons in average case.

2.2 COLONIZATION OF DATA

The concept of colonization comes from the sorted sub-sequences of a random array [4]. A sub-sequence can be considered as an independent data colony whose address is saved in array 'R'. In the above example there are four colonies (6,8,13), (2,6), (1,7,9) and (5,8). If there are more data colonies then memorized sort takes more number of comparisons to sort the array. After performing some pre-processing on the data to have less number of colonies, the performance of the memorized sort can be increased.

2.3 MEMORIZED SEARCH

Memorized search is a fast searching approach and could be implemented on colonized random data [4]. As each colony is an autonomous unit, different searching techniques can be applied on different colonies on the basis of the population. If some colony has more population then binary search gives optimal results. This kind of search is called binary memorized search. To search a large number of elements, we can perform multiple searching methods on different colonies which is called hybrid memorized search. Hybrid memorized search performs binary search on large colonies and sequential search on smaller colonies. Upper bound to memorized search is still O(n) for random data set but it saves lot of comparisons due to the logarithmic nature of binary search.

III. PERFORMANCE OF MEMORIZED SORT ON COLONIZED DATA

This section shows the performance of memorized search when running on random data sets with various numbers of colonies. The behavior of random data set is totally unpredictable. Now we study the performance behavior of memorized sort on colonized data be decreasing the colonization density. We start with the maximum number of colonies. Let's say there are 40% colonies in an array, it means most of the elements are not sorted and are of minimum size which is 2. When all the elements of the array are sorted in reverse order, there are 50% colonies each of size 2. But we are focused on randomized data we will start the performance analysis with 40% colonies in an array. Fig.1 shows the running time of Memsort, Merge Sort and Quick Sort after running on random data with different number of colonies.

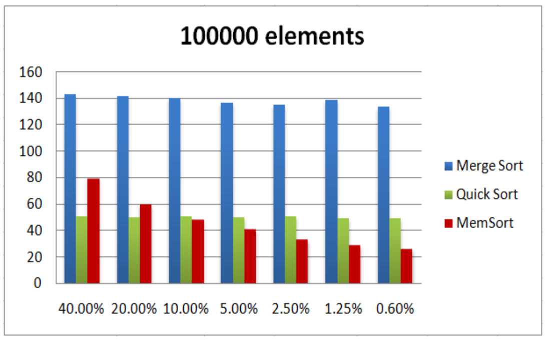
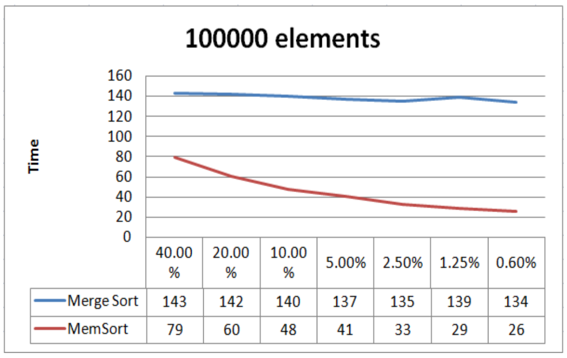


Figure.1: Comparison of running times

Merge sort and quick sort are also efficient sorting algorithms but they do not identify the sorted sub-sequences of already sorted element but Memsort does. Fig.1 is very self explanatory. As we decrease the density of colonies, Memsort starts to take less time to sort same random data set.

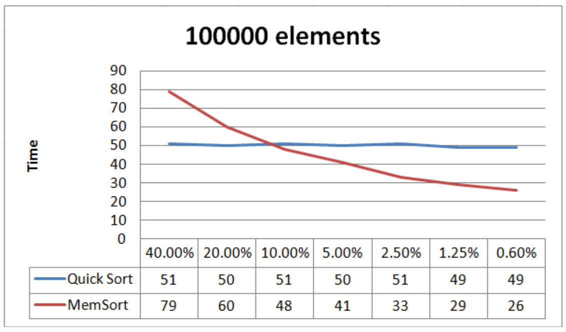
IV. RESULTS

Results are computed after running three sorting algorithms on an array of one hundred thousands of elements. In first run, there are 40% colonies of all number of elements. In next run we run an iteration of colonization procedure and the number of colonies decreased to be half. Fig.2 and fig.3 show the behavior of Memsort, Merge sort and Quick sort based on running times. It is very clear that Merge sort and Quick sort algorithms show similar and consistent behavior in running times even the numbers of colonies are decreasing gradually. But Memsort starts sorting the same random array in less time.



	40.00 %	20.00 %	10.00 %	5.00%	2.50%	1.25%	0.60%
Merge Sort	143	142	140	137	135	139	134
MemSort	79	60	48	41	33	29	26

Figure.2: Behavior comparison of Memsort with Merge sort.



	40.00%	20.00%	10.00%	5.00%	2.50%	1.25%	0.60%
Quick Sort	51	50	51	50	51	49	49
MemSort	79	60	48	41	33	29	26

Figure.3: Behavior comparison of Memsort with Quick sort.

V. CONCLUSION

As the degree of colonization is decreased, Memsort starts showing more linear like behavior which makes it a very efficient sorting algorithm for random data. Worst case of Memsort is similar as the worst case of Merge sort algorithm. Memsort does not change the time complexity class from logarithmic to linear. So the time complexity of the algorithm is still O(nlgn). Running time of Merge sort and Quick sort is directly proportional to the number of elements in an array whereas the running time of Memsort is directly proportional to the entropy of data.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Intoduction to Algorithms*, 3rd ed, MIT Press, 2011.

[3] Frederic H. Murphy, Edward A. Stohr, A Dynamic Programming Algorithm for Check Sorting, *Management Science, 24(1)*, September 1977, 59-70.

[4] T. Ehsan, M. Usman, Q. Javed, An Efficient Searching Technique by Colonization of Random Data Set based on Dynamic Programming, *International Journal of Engineering Research and Applications (IJERA), 3(5)*, 2015-2020.

[5] T. Ehsan, M. Usman, Q. Javed, An Efficient Sorting Algorithm by Computing Randomized Sorted Sub-sequences Based on Dynamic Programming, *International Journal of Computer Science and Network Security (IJCSNS), 13(9)*, September 2013, 51-57.

[6] C. A. R. Hoare, Quicksort, *Computer Journal 5(1)*, 1962, 10-15.

[7] Deepak Abhyankar, Maya Ingle, Elements of Dynamic Programming in Sorting, *International Journal of Engineering Research and Applications (IJERA), 1(3)*, 446-448.

[8] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Pearson Education, 1998.

[9] S. Baase and A. Gelder, *Computer Algorithms:Introduction to Design and Analysis*, Addison-Wesley, 2000.

[10] Anany Levitin, *Introduction to the Design and Analysis of Algorithms*, 2nd ed, Pearson Education, 2007.

[11] D. Abhyankar, M. Ingle, A Performance Study of Some Sophisticated Partitioning Algorithms, *International Journal of Advanced Computer Science and Applications (IJACSA), 2(5)*, 2011, 135-137.