RESEARCH ARTICLE OPEN ACCESS

An Efficient Searching Technique by Colonization of Random Data Set Based On Dynamic Programming

Toqeer Ehsan[1,] M. Usman Ali[2,] Meer Qaisar Javed[3]
Faculty of Computing and Information Technology, University of Gujrat, Pakistan

ABSTRACT
The logarithmic behavior of Binary Search to find elements requires data set to be arranged in ascending or descending order. This paper introduces the concept of data colonization which means naturally ordered sub-sequences are grouped together to become a data colony so that any type of searching technique could be applied to any ordered sub-sequence independently. Addresses of the colonies i.e. starting and ending of colonies are remembered by using the concept of memorizations from dynamic programming. We have run different searching techniques in different colonies on the basis of the size of the colony and results are satisfactory after comparing with sequential search. As we keep on increasing the size of the colonies by decreasing the colonizational density of data set, the hybrid memorized technique starts showing more logarithmic behavior rather linear even the list is not arranged.
Keywords **-** Colonization, Complexity, Dynamic programming, Memorization, Hybrid approach

I. INTRODUCTION

Under the study of computational methods, the problem of searching a specific data object from a large number of elements is always has an intellectual significance. Searching plays important role in lot of computational areas like database management systems, spreadsheets, text editors, natural language processors, lists and arrays, internet searching etc. A lot of techniques have been proposed to solve afore mentioned problem of searching with some limitations. These techniques belong to different problem solving techniques, some belong to divide and conquer family like binary search. Binary search exists in a lot of variations and is very efficient technique that could find an element in logarithmic ($\Theta(\lg n)$) time complexity. On the other hand, very famous algorithm called sequential search belongs to brute force methods. Sequential search also called linear search performs the searching in linear time complexity($O(n)$). So the running time of linear search is higher as compared to binary search apparently. Linear search algorithm is equally applicable on ordered and random data but binary search only runs on already ordered data. Both techniques work on different data sets, so if we have the random data set then we are bound to use sequential search to search the specific data objects.

By comparing the logarithmic time complexity with linear we find that the binary search is much more efficient as compared to linear search. But before using the binary search algorithm we first need to sort the list. Sorting of elements which would again cost time because the lower bound to a comparison based sorting algorithm is $\Omega(n \lg n)$. In the applications in which we are dealing with the real time random data, we need an efficient searching technique that could search the desired data set without sorting. In this paper an efficient search algorithm is introduced that performs searching on a pre-processed trained but real time random data set. The results are encouraging when compared with the brute force sequential search. A colonization technique is being used to group the random data in a list which memorizes the addresses of the colonies and algorithm is designed on the bases of dynamic programming. Dynamic programming is a problem solving technique which is used to solve the optimization problems. In our case the optimal solution is the optimal number of comparisons. Memorization technique helps to save some number of comparisons due to the colonization process. The process of colonization is completed before application of the algorithm. Once the data is trained we can apply any searching algorithm including our own algorithm to compute the results, these multiple runs would not affect the original trained data. Our main focus is to improve the average case running time of the algorithm on any type of random inputs.

II. DATA PRE-PROCESSING

As we discussed in the section 1, we are going to perform some operations on the data set to train the data so that our algorithm can perform an efficient searching by saving the number of comparisons. One method is to simply sort the list and then perform searching that would be efficient but the cost of sorting is even higher than brute-force searching. So we are not going to sort the array, all we need is to move the elements into colonies and note the addresses of these groupings.

2.1 COLONIZATION OF DATA

In real world it is very hard to keep the track of house addresses if there are only numbers in big

cities and towns so the towns are divided in the form of blocks and sectors to make it easier to locate any house. After doing that we just limit the scope of the data that would take less time to find any location. Similar technique is being used in this paper which we are calling the colonization of data. We divide the huge data into small portions let us call each portion as a colony; colonies are identified by their registered names and boundaries. Set of colonies could further be divided into sectors or blocks but in this paper we are just using colonies for searching by computing the optimal number of comparisons. Example of colonization can be elaborated by the Fig.1 given below:

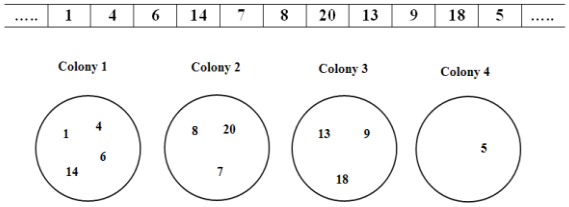

Figure.1: Colonization of random data.

Fig.1 shows the process of colonization by grouping that integer data from a segment of an array into four different colonies. To group the data into portions different approaches could be used, for example to set a threshold on data on the bases of average or minimum number to some maximum number. But here we have exploited the fact that in an array of random numbers, some sequences are already sorted so we don't need that elements to put in order or sort them. Our pre-processing technique computes the pre-sorted sub-sequences and puts each sub-sequence in one colony so that we could perform efficient search on data. After colonization of data or sometimes we call it training of data, some changes could occur in our original data set. If we examine the list first sorted sub-sequence starts from 1 and ends at 14, second sequence starts from 7 and end at 20, now both sub-sequences moved to colony1 and colony2 respectively. Colony 3 in the Fig.1 shows that there are three elements in it which are 9, 13 and 18. But the sequence in the list is 13, 9 and 18 which is not sorted. In the colonization process we are also taking care of unsorted pairs. If two numbers are not sorted, it means they are sorted in reverse order so in the pre-processing algorithm swaps two elements if next element is smaller than previous if it is first comparison for next colony. Computing the element for colony 3, algorithms swaps the elements 13 and 9 before moving them to colony. While pre-processing the data, we don't need to sort the array, so the process of colonization must be completed in linear time complexity.

After colonization process the list, in Fig.1 would be changed a little as given below:

.....	1	4	6	14	7	8	20	9	13	18	5	.....

Figure.2: Data segment after colonization process.

2.2 MEMORIZATIONS

After the colonization process, we need to keep track of the boundary of each colony. Dynamic programming uses a method of memorization to save the start and end of a colony so that algorithm can search within a colony efficiently; otherwise we have to compute the addresses of colonies every time we run a searching algorithm. A new memory or list is created to keep the addresses of the sub-sequences. The length of new list would be almost half of the original list that caters even worst input so the length of the memorized list would be n/2+2 if the length of input list is 'n'. Let we create new list R with length 6+2 as the length of list in above example is 11. Fig.3 shows the value of list R after colonization process. Let us consider the list just as standard array of elements. First element contains starting element of first colony and second element contains end of first colony. Third element contains the end of second colony as the starting of second colony is the end of first colony plus one. After colonization process the array R would look like:

indexes	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
values	0	3	6	9	10	-1		

Figure.3: Values of 'R' after colonization process.

As the size of array 'R' is eight so the indexes of the array start from '0' and end at '7'. Values of 'R' are the index values of input array in the Fig.2. -1 is placed at the end to ensure the end of the array values. Index integer very before to -1 indicates the total number of colonies; here it is four so there are four data colonies in our example.

III. COLONIZATION ALGORITHM

Pre-processing algorithm is an iterative and very intuitive in nature. The steps of the algorithm are as follows:

Step1. Compute the next sorted sub-sequence from input list.

Step2. Populate list 'R' with the indexes of computed sequence.

Step3. To Step1 up to the last sequence.

3.1 PSEUDO CODE

```
Colonization (array, start, end)
Rsize = (end + 1)/2  + 2
Create a new array R of size Rsize
i=start, R[0]=start, flag = 0, j = 1
Col_count=0
while(i < end)
        while( array[i]<=array[i+1] )
                flag = 1
                i=i+1
        if(!flag)
                swap array[i] and array[i+1]
                i=i+1
```

```
                flag = 1
        else
                R[j] = i
                i=i+1, j=j+1
                Col_count=Col_count+1
                flag = 0
endwhile
if (i =end)
        R[j]=i
        j=j+1
        Col_count=Col_count+1
R[j]=-1
```

3.2 ANALYSIS

The algorithm given in the previous section creates a new array 'R' which is used for memorizations of the addresses of the colonies. The term address is referring to the start and end of a colony. In the algorithm outer while loop executes 'n' times if the length of the array is 'n' and indexes are '0' to 'n'. Inner while loop decides the data members that are going to belong to that colony. In the algorithm no element would belong to more than one colony at the same time unless there is some duplication. So the total number of comparisons of the elements to compute the sorted sub-sequences would be equal to 'n' for a list of 'n' numbers. Time complexity of the algorithm could be defined as $\Theta(n)$ asymptotically.

IV. SEARCHING MECHANISM

After the colonization process each colony can be treated as an independent data set so different searching techniques could also be applied to different colonies to achieve more efficiency. First of all we have to check the start and end index of each sub-sequence which would decide whether to enter some specific colony or not. For example if our key is 23, in the colony one the largest element is less than 23, it means key would never be found in colony 1. If in some cases the key element is smaller than last element then we also have to check the lower element as well. Before applying any searching technique to any colony, we ensure whether that colony has the potential to have the key, then we proceed further. In this section we are going to discuss different scenarios to search when we have data in the form of colonies already.

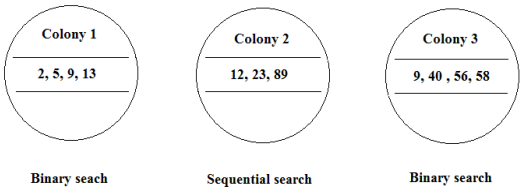

Figure.4: Different searching techniques on different colonies.

4.1 BINARY SEARCH

As the data members in each colony are already sorts so we must use binary search for each colony to search the key element. Doing this we could save some number of comparisons in each colony and when there is a huge data these saved comparisons will result in a better running time. There are some limitations in this approach, if there are more colonies for example 40-50 percent number of colonies then there must be very small grouping of data then binary search will work just like sequential search. It means binary search could be used for searching if there is less number of colonies with greater length. Let's call this technique as Binary memorized search.

4.2 SEQUENTIAL SEARCH

Sequential search could also be used for searching within any colony. Sequential search would work fine as compared to binary search if there are more colonies with less number of elements. In this case our technique would be almost equal to straight forward brute force sequential search with $\Theta(n)$ time complexity. We are calling that technique as Sequential memorized search.

4.3 HYBRID SEARCH

The performance of the searching techniques is dependent on the size and number of colonies. If we use a hybrid approach to search the independent colonies, we could achieve even more efficiency. If some colony has less number of elements let say less than 5 then apply sequential search otherwise binary search. Doing this we could save a lot of comparisons as compared to simple sequential search on random data. To implement this hybrid technique, we need to design such an algorithm that keeps the track of number of colonies. If there are more colonies for example 20-50 percent then use sequential search and if there are less than 20 percent colonies then implement binary search for data searching within a colony. We are calling this technique as Hybrid memorized search.

V. ALGORITHM

Step.0 Colonize the data by running colonization procedure.
Step.1 Compute the total number of colonies.
Step.2 If the size of the colony is greater than five.
Apply Binary search on sub-sequence
Else
Apply Sequential search on sub-sequence

5.1 PSEUDO CODE

```
Hybrid-Mem(array[], R[] , Count, Key)
int k=0,left=0,pos=0,right=0
right=R[1];
while(Count>1)
    if(array[left]<=key)
        if(array[right]>=key)
            if((right - left) == 1)
```

```
                    if (array[left]==key)
                        return left;
                    if(array[right]==key)
                        return right;
                else if((right - left) <= 5)
                    Sequential_Search(array,
Key,left,right)
                else
                    Binary_search(array,Key,left,right)
   Count=Count-1
   K=k+1
   left=R[k]+1
   right=R[k+1]
endwhile
```

5.2 ANALYSIS

Algorithm presented in the section 5.1 takes five parameters as input. First parameter 'array' is our data set after pre-processing, 'R' is the array having information of sub-sequences starting and ending indexes, count is the total count of the sub-sequences and Key is the element to find. Only hybrid memorized search pseudo code is being discusses as it contains both sequential and binary search for the colonies. Our algorithm decides the appropriate inner searching technique on the basis of the size of the colony. We have set the lower size limit on five. If there are more colonies with size greater than five then hybrid memorized search saves some number of comparisons in each colony which will result the better performance. Whether we implement hybrid or binary search it may not change the efficiency class of the algorithm although average case analysis on random inputs gives better results. So we still are bound to claim that time complexity of the algorithm belongs to O(n). But as we keep on decreasing the count of colonies our algorithm start showing logarithmic behavior rather linear because of divide and conquer nature of binary search.

VI. RESULTS

We are going to compare four different algorithms, one is standard sequential search and others are sequential, binary and hybrid on the bases of memorizations. We have run all the algorithms on same random data with same key for searching but to make sure of the comparison of average case, we have counted the number of comparisons of one hundred runs for each algorithm. We are going to analyze the results on the bases of the number of comparisons that each algorithm takes to find some specific element in the array. We have run the algorithms on the random data with different number of colonies and they are showing different behaviors. First we have taken the array having 40% sorted sub-arrays or colonies.

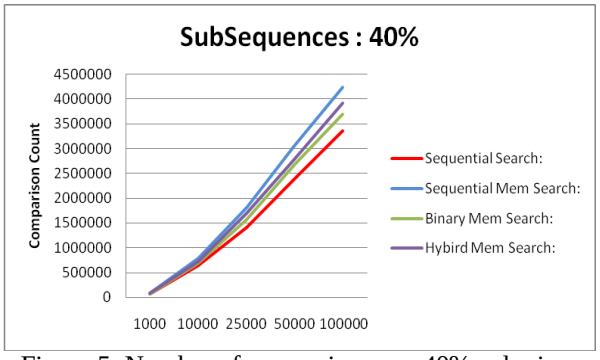

Figure.5: Number of comparisons on 40% colonies

All the algorithms including sequential search show the similar behavior when there are approximately 40% number of sub-sequences. It means most of the sub-sequences are of size two and some may be of size greater than two. So in each case hybrid memorized search and binary memorized search take number of comparisons equal to sequential search. Now we change the input array to have less number of colonies. Let after merging the pair of sub-sequences we will get 20% number of colonies.

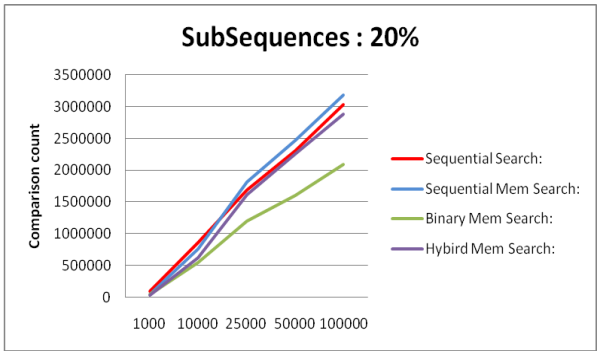

Figure.6: Number of comparisons on 20% colonies

After decreasing the number of colonies binary memorized search computes less number of comparisons as compared to other techniques. Let's further decrease the number of colonies by merging the pairs of the sorted sub-sequences.

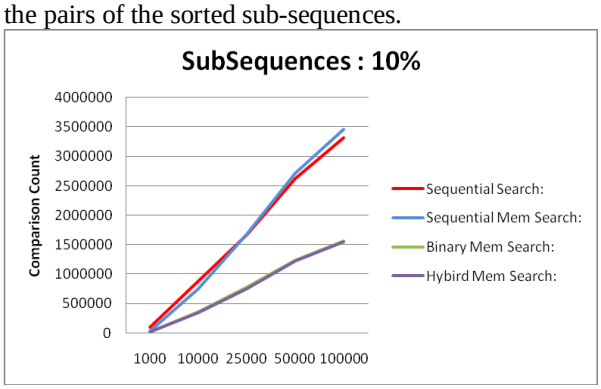

Figure.7: Number of comparisons on 10% colonies

After decreasing the number of colonies to 10% binary memorized search and hybrid memorized search perform well with less number of comparisons as compared to sequential search and sequential

memorized search beacause at 10% colonies, most of the sub-sequences have size greather than five. In hybrid technique we have set the threshod on the size of the sub-array which is five. To use binary search on the sub-array the minimun size of the sub-array must be five so that at least one comparison could be saved. Sequential memorized search acts just like sequential search as there are very less number of comparisons which are saved when running on random data. Now we further decrease the number of colonies by increasing the size of sorted sub-sequences to get the clear picture of the performance behaviors.

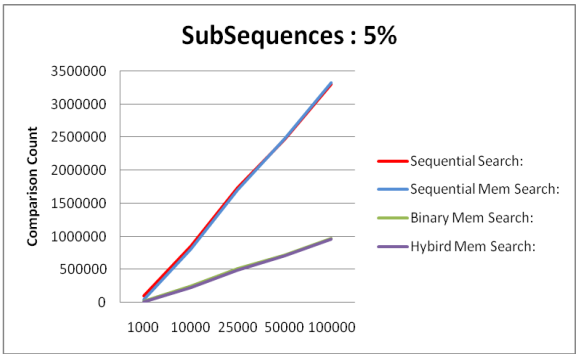

Figure.8: Number of comparisons on 5% colonies

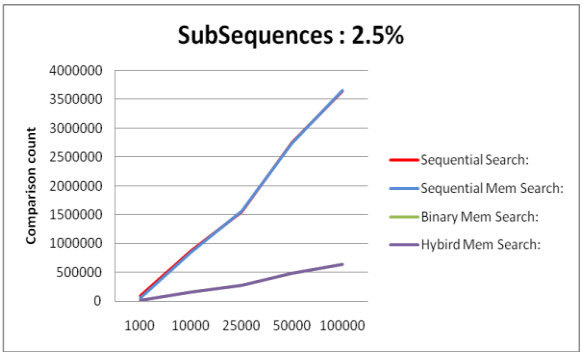

Figure.9: Number of comparisons on 2.5% colonies

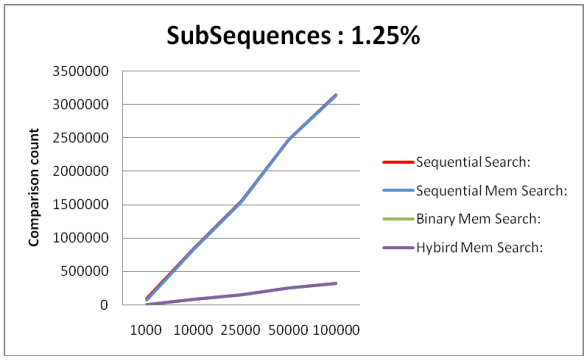

Figure.10: Number of comparisons on 1.25% colonies

VII. CONCLUSION

Binary search based on the divide and conquer technique is an efficient search algorithm and searches the element from an array in logarithmic (O(lgn)) time but it only works on sorted array. We have to use sequential search when searching from a random unsorted array. Colonization technique is introduced to group the already sorted sub-sequences so that we can perform different search algorithms on different colonies on the basis of the size of the colony to save the comparisons. We have calculated the results by using sequential search on each colony, binary search for each colonies and a hybrid technique which selects the appropriate searching algorithm according to the size of the colony. All the algorithms give the same performance when there are almost 40% colonies. But when we gradually decreased the number of colonies by increasing the size of the sorted sub-sequences after merging the pairs of the colonies, hybrid memorized search and binary memorized search start showing similar and efficient behavior. Fig.11 shows the behaviors of the algorithms with different number of sub-sequences. It is concluded that sequential memorized search shows linear behavior no matter what percentage of the data is colonized whereas binary and hybrid memorized search start showing logarithmic behaviors when there are less number of colonies even the whole array is still unsorted.

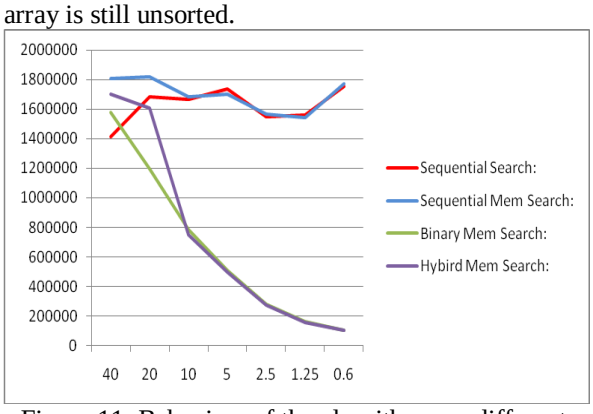

Figure.11: Behaviors of the algorithms on different type of inputs

REFERENCES

[1] T. Ehsan, M. Usman, M. Qaisar, 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.

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

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

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

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

[6] D. E. Knuth, *The Art of Computer Programming*, Vol. 3, Pearson Education, 1998.
[7] Frederic H. Murphy, Edward A. Stohr, A Dynamic Programming Algorithm for Check Sorting, *Management Science, 24(1),* September 1977, 59-70.
[8] 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.
[9] Dr. Anupam Shukla and Rahul Kala, Predictive Sort, *International Journal of Computer Science and Network Security (IJCSNS), 8(6),* June 2008, 314-320.