ISSN No. 0976-5697

Volume 8, No. 3, March – April 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Optimizing the Performance of Quadratic Search using Memorization

Deepti Verma and Dr. Kshama Paithankar
Department of Computer Science
Shri Vaishnav^{SM} Institute of Management,
Indore, India

Abstract: Time is the most important feature that counts for an executing process. Nowadays, applications are more focused to reduce the computational time and thus to increase the speed. Therefore, there is a need to redesign the fundamental algorithms especially for searching methods to achieve this objective. Targeting this, approach of memorization is proposed in this paper to optimize the time of searching during execution. It is an optimization technique used to eliminate re-execution time for pure function. A novel approach of memoized Quadratic Search Algorithm is presented in this paper. Results show remarkable improvement in the performance in terms of time leading to achieve the said objective. This will help to optimize the time in other respective areas of computation processes.

Keywords: Binary Search, Quadratic Search, Optimization, Memoization, Execution Time.

1. INTRODUCTION

Searching is a method which is used in every area especially in the applications of computer science where huge data is stored. As the size of the data increases, time for the searching a particular record also increases. There exist many searching techniques which allow faster retrieval of data such as Linear Search, Binary Search, Interpolation Search etc. and having pros and cons of individual method with respect to different situations [1]. A modification in the classical Binary Search is proposed in the form of Quadratic Search that reduces the complexity and is an improved searching resulting in faster search [2].

Optimization is a technique which is used in every area either to save time or to maximize the profit or output. In Computer Science also, this is used in different way to get the required favorable output. Particularly for the programming it is used by technique called memoization.

Memoization is used for optimization particularly to speed up the computer programs [3]. It is a special case of optimization which basically saves the time by storing the result of the expensive function call's result, which can be further used without recompilation, wherever and whenever required. Memoization is also known as function caching. It was first introduced in 1968 in the context of Artificial Intelligence [4]. It is a way for machines to learn from the past experiences. In other words, programs could "recall" previous compu tations and thus avoid repeated work [5, 6]. The key idea behind this is to speed up the execution of a function by maintaining the cache of its previous computations and look-up into the cache instead of computing data again and again.

Normally, if a function is being called more than once, it will execute its computational code and store result of each iteration. It is time consuming and also directly affects memory utilization. By using memorization, a process verifies whether the calling function is called for the first time. If it is so, it will be executed normally. From the next time, the results will be looked-up in the cache. As a result, Memoized function may reduce the redundancy, computational time and thus the execution time [7, 8]. Here, an algorithm of Quadratic Search with the memoized approach using algorithm *MemQS()* is proposed to optimize the searching time.

In Section 2, background and related work is discussed briefly. Non-memoized and proposed memoized algorithms are presented formally in Section 3 and Section 4 respectively. Section 5 includes the output considering the different cases. Comparison of performance of memoized and non-memoized quadratic search algorithm is discussed in Section 6. Finally, Section 7 covers conclusion with limitations and future scope.

2. BACK GROUND AND RELATED WORK

Different types of searching techniques are available which are applicable for different kinds of applications, and some are highly specialized to specific tasks. For instance, Linear Search is used for searching within unsorted data whereas Binary and Interpolation Search are used for the already sorted list [2, 9]. Modification in the traditional searching methods is required for increasing the efficiency and effectiveness of the algorithm. Hence, a new approach of search with a modification in the classical Binary Search is proposed termed as Quadratic Search method [2]. Though it performs better than Binary Search in terms of time, possibility of further optimization has been observed clearly.

3. QUADRATIC SEARCH

Non-memoized Quadratic Search

Quadratic search is a modified form of Binary Search that works on Divide and Conquer method. Here, Searching is done by comparison with the centre position value as well as with its quarter's position value. Its centre position is calculated by FIRST+LAST/2.
Its quarter's positions are calculated as

P_1 = FIRST+(LAST-FIRST)/4;
P_2 = FIRST+(LAST-FIRST)3/4;

The Algorithm has four conditions after calculating MID, P_1, P_2, when the key is not found at the middle, P_1 and P_2 :
Condition 1 : IF KEY < Middle and also < P_1
Means Key is found in first quarter

then (P_1-1) will become the LAST and the procedure will be repeated.

Condition 2 : IF KEY < Middle and > P_1
Means Key is found in second quarter
Then (P_1+1) will become the FIRST and (MID-1) will become the LAST and the procedure will repeated.

Condition 3 : IF KEY is > Middle and < P_2
Means Key is found in third quarter
Then (MID+1) will become the FIRST and (P_2-1) will become the LAST and the procedure will repeated.

Condition 4 : IF KEY > Middle and also > P_2.
Means Key is found in second quarter
Then (P_2+1) will become the FIRST and the procedure will repeated.

Pictorial representation of the Quadratic Search Method is shown in Figure-1.

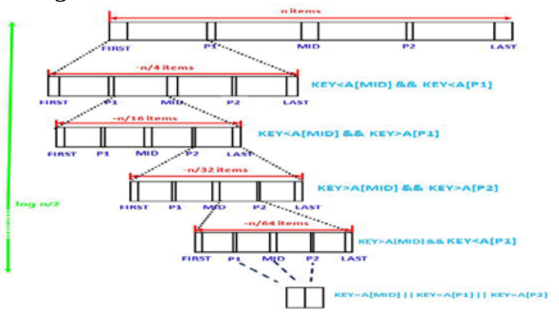

Figure-1: Diagrammatic representation of the Quadratic Search Process

Formal Description of non-memoized Quadratic Search

Quadratic Search is a method for searching through ordered random data[1]. It is retrieving a desired record by key in an ordered file by using the value of the key and the statistical distribution of the keys. It works on Divide and Conquer method [9]. Here, the algorithm *NMQS()* is presented.

Algorithm *NMQS()*

```
/*arr[] represent the data set*/ /*n1
contains the size of data set*/
/*key is  the key value to be searched */
/* first and last holds the first and last index position value of
array*/
/*mid is a class partition value*/
/*$P_1$ and $P_2$ are hold the first quarter and the second quarter
partition position values*/
Start
Step 1: int Quadratic_Search (int arr[], int key, int FIRST ,
        int LAST)
/*Function declaration with the parameters of data set and key
value and the limits of data set.*/

Step 2:   first = 1; last = n;
 /* initializes first with 1 and size of data set is stored in last.*/

Step 3: While( FIRST<=LAST )

Step 4: MID = (FIRST+LAST)/2;
/* find middle position*/
        $P_1$ = FIRST+(LAST-FIRST)/4;
/* find first quarter position*/
        $P_2$ = FIRST+(LAST-FIRST)*3/4;
/* find second quarter position*/

Step 5:  If(KEY==A[MID] || KEY==A[$P_1$] ||
        KEY==A[$P_2$]) return element found; /*
    Search Successful and key found at middle
             position*/
Step 6: Else if(KEY<A[MID] && KEY<A[$P_1$])
           LAST=$P_1$-1;
/* key lies in the first quarter*/

Step 7: Else if(KEY<A[MID] &&
           KEY>A[$P_1$]) FIRST=$P_1$+1;
           LAST=MID-1;
/* key lies in the second quarter */

Step 8 Else if (KEY>A[MID] &&
        KEY>A[$P_2$]) FIRST=$P_2$+1;
 /* key lies in the third quarter */

Step 9: Else if(KEY>A[MID] && KEY<A[$P_2$])
          FIRST=MID+1; LAST=$P_2$-1;
           /* key lies in the fourth quarter */

Step 10 Return element not found

End
```

4. MEMOIZED QUADRATIC SEARCH

Formal Description of memoized Quadratic Search

This memoized algorithm accepts the value entered by the user and verifies the output of non-memoized quadratic search function ***NMQS()*** [10]. If the value is found, it will store in an integer variable, and whenever needed extracts the output stored in a variable namely val with the help of memoized function. The output of memoized and non-memoized algorithm may not always same. It varies on the execution process of the system that how much time it will take to fetch the value from the cache.

Instead of variable, if hash-table or vector is used to store data, it may take large amount of time to process because hash-table and vector are inbuilt classes of *util* package and use their own functions to store and fetch the value.

Algorithm *MemQS()*

```
    /* v is an integer type variable for determining
                              successful search*/
/* st1 and ed1 are  integer type local variables*/
  /* b is an integer type local variable used to retrieve output*/
Start
Step 1: st1=Systemtime    /* to stores starting time */

Step 2: b=d1.memoIps();  /* Retrieving output */
          ed1=Systemtime;  /*getting ending time */

Step 3: tot1=(ed1-st1);
           Message ("Memo Time "+tot1);
 /*Showing total time with
 memoization*/

Step 4: if (v==1)
          Message ("Value Found= "+b);
/*When the value is found then show the output with time*/
```

Step 5: int memoSearch()

Step 6: return val;

End

5. CASE STUDY

The performance of proposed algorithm *MemQS()* and that of *NMQS()* has been evaluated using three cases. Case 1 includes the study for data size 10 whereas Case 2 deals with data size as 25. Data size 50 was considered for study in Case 3. These different cases have been studied for the *NMQS()* as well.

Case 1: Data size 10

Here, the performance of *NMQS()* and *MemQS()* is discussed with the list containing 10 values. Table-1 illustrates the performance in terms of time whereas Figure-2 represents the trend of time including that memoized function improve the performance consistently.

Table-1: Dataset of 10 values (Values from 10 to 100)

SearchValue	Non-memoized Time	Memoized Time	Difference
10	1370	1359	11
20	3623	1359	2264
30	2265	1358	907
40	3171	1359	1812
50	3170	1359	1811
60	3170	1359	1811
70	2717	1359	1358
80	3171	1358	1813
90	3170	1359	1811
100	3170	1358	1812
NotFound	3623	1359	2264

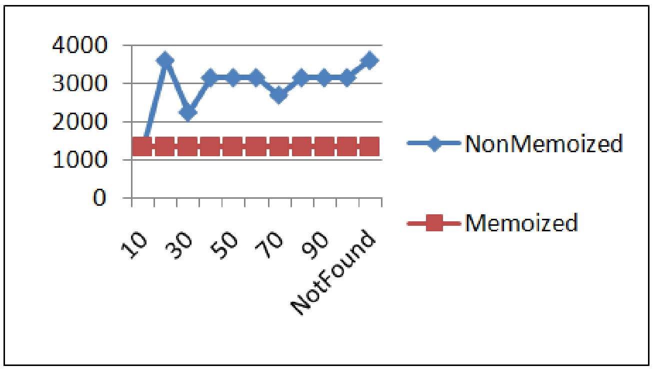

Figure-2: Performance of *MemQS()* vs *NMQS()* in Case 1

Case 2: Data size 25

In this Case, the list of dataset containing 25 values is experimented with proposed *MemQS()* and *NMQS()* as well as shown in Table-2. Similarly, Figure-3 represents the trend of time. In this Case increasing the size of the dataset also results in improving time and thus the performance of algorithm using memoization.

Table-2: Dataset of 25 values (Values from 10 to 250)

Search Value	Non-memoized Time	Memoized Time	Difference	Range
50	3170	1358	1812	10-100
60	3623	1359	2264	10-100
120	3623	1359	2264	100-150
140	4076	1359	2717	100-150
160	3623	1359	2264	150-200
190	2717	1358	1359	150-200
210	3170	1812	1358	200-250
240	3623	1359	2264	200-250
220	3170	1359	1811	random
80	3623	1359	2264	random
Not Found	4076	1812	2264	

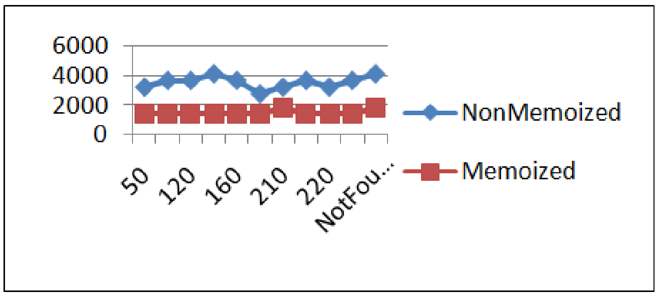

Figure-3: Performance of *MemQS()* vs *NMQS()* in Case 2

Case 3: Data size 50

A list containing 50 values for implementing *NMQS()* and *MemQS()* is used in this Case. Table-3 highlights the outcome in terms of time and Figure-4 represents the trend of this time. Here, it is observed that the performance of *MemQS()* still follows the trend of optimization of time to improve performance of searching.

Table-3: Dataset of 50 values (Values 10 to 500)

Search Value	Non-memoized Time	Memoized Time	Difference	range
50	5435	1359	4076	10-100
80	4076	1812	2264	10-100
110	3623	3170	453	100-200
200	4076	1359	2717	100-200
250	2717	1359	1358	200-300
290	4076	1358	2718	200-300
320	2623	1359	1264	300-400
380	4076	1811	2265	300-400
430	4076	1359	2717	400-500
470	3623	1359	2264	400-500
450	3623	1358	2265	random
480	3623	1359	2264	random
not found	4529	1358	3171	Not Found

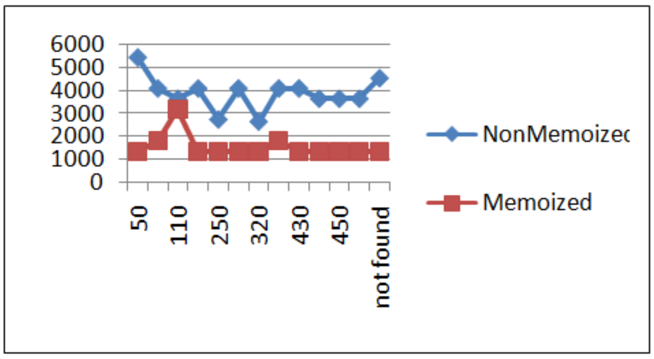

Figure-4: Performance of *MemQS()* vs *NMQS()* in Case 3

6. DISCUSSION

With the help of three cases, it has been noticed that *MemQS()* really improves the performance of searching as compare to regular *NMQS()*. For instance, in Case 1 the time recorded to search the first element is 1370 ms using *NMQS()* whereas using *MemQS()* it is noted 1359 ms thereby reducing the time by 11 ms clearly as shown in Table-4.

Table-4: Comparative Performance in different Cases

case	Index	Search Value	Non Memoized Time	Memoized Time	Difference
1	First	10	1370	1359	11
	Mid	50	3170	1359	1811
	Last	100	3170	1358	1812
2	First	50	5435	1359	4076
	Mid	160	3623	1359	2264
	Last	240	3623	1359	2264
3	First	50	5435	1359	4076
	Mid	290	4076	1358	2718
	Last	470	3623	1359	2264

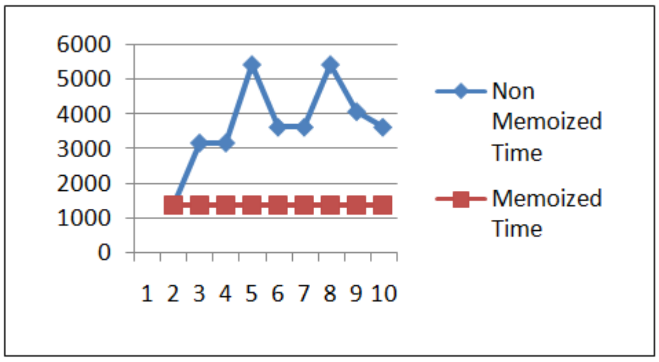

Figure-5: Cumulative bracket of *MemQS()* and *NMQS()*

Similarly, for mid element, time recorded is 3170 ms using *NMQS()* and in *MemQS()* it is 1359 ms. Here, time is reducing by 1811 ms and in the case of last element the searching time is recorded using *NMQS()* is 3170 ms where as using *MemQS()* it is computed as1358 ms thereby reducing by 1812 ms. This trend is observed in Case 2 and Case 3 also as represented by Figure-5. Presently, the performance of *MemQS()* is evaluated for first, middle and last elements of the lists. However, the trends of performance indicate that it may surely be applicable for the element at any position in the list.

7. CONCLUSION

A proposed memoized Quadratic Search is faster than known Quadratic Search method. Since memoization itself is an optimization technique, here also it proves to be effective Technique of optimizing time. Irrespective of the index, memoized Quadratic Search takes consistent time. The proposed algorithm may be verified for the large databases. Recursive implementation may be applied for further optimization.

8. REFERENCES

[1] Kumar, P. “Quadratic Search: A New and Fast Searching Algorithm (An extension of classical Binary search strategy), International Journal of Computer Applications (0975 – 8887) Volume 65, Issue14, March 2013
[2] Horowitz,E. and Sahni,S.: Fundamental of Data Structure Rockville, MD: Computer Science Press, 1982.
[3] Purey J., Paithankar K. Memoization: a Technique to optimize Performance of Searching, National Conference on “Challenges of Globalization and Strategies for Competitiveness” Shri aishnav Institute of Management, Indore, January, 2015, pp 486-491.
[4] Norvig, P. “Techniques for Automatic Memoization with Applications to Context-Free Parsing”, University of California,Volume 17, Issue1, March 1991, pp 91-98 .
[5] Pfeffer, A. “Sampling with Memoization”, School of Engineering and Applied Sciences, Harvard University, 2007.
[6] Ziarek, L; Sivaramakrishnan,K.C.and Jagannathan, S. “Partial Memoization of Concurrency and Communication”, Department of Computer Science Purdue University, Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming, Edinburgh, Scotland, ACM, pp 161–172.
[7] Brown,.D. and Cook, W. sR. “Monadic Memoization Mixins”, Department of Computer Sciences, University of Texas at Austin, 2006.
[8] Purey, J. and Muley, K. “Auto Response Memoization using JAVA”, National Conference on Emerging Technologies in Electronics, Mechanical and Computer Engineering (ETEMC) April 2010.
[9] Demaine, E.D.; Jones, T. and Patrascu,M. “Interpolation Search for Non- Independent Data”, Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) , 2004, pp 529–530.
[10] Verma, D. and Paithankar, K. Interpolation Search: A Memoized Aproach International Journal of Latest Trends in Engineering and Technology, Nov 2016, pp 218-224