IBBUL
Lapai Journal of Applied and Natural Sciences
LAJANS Vol 4(1): 90-95

RESEARCH PAPER

Comparison of The Performance of 2d Binary Search Algorithm With Row Wise And Column Wise Search Algorithm On Sorted Matrix

Oladipupo E.T., Adeagbo C.O., Alao K.A. and Siyaka O.H.

Department of Computer Science, Federal Polytechnic, Bida, Nigeria

ABSTRACT

Binary Search Algorithm is an efficient algorithm when it comes to searching items within a sorted linear array. The technique, with little modification, can be applied to a sorted matrix and also maintain its efficiency. Row-Wise and Column-Wise Search Algorithm is also a very efficient algorithm that performs better than other algorithms when it comes to searching within a sorted matrix. This paper reports the findings when the performance analysis of the two algorithms are carried out using the following parameters: analysis of the operations performed by the functions used in the implementation, worst case iteration complexity, number of conditions check during the execution of each function that implements the algorithms and worst case time complexity of the algorithms. 2D Binary search algorithm was found to be more user friendly, simpler and time effective in that it requires lesser number of operations to be performed by the function that implements it, lesser number of iterations, lesser number of condition checks and smaller runtime complexity.

Keywords: Runtime, complexity, algorithm, worst case

INTRODUCTION

Computers are used to process huge amount of data and so searching information from these large database is a crucial operation. A search strategy is a procedure that decides the next query to be posed based on the outcome of previous queries (Parveen, 2013). A search problem is defined by a search space which is the set of objects among which a search for the solution is to be made and a goal condition which depicts the characteristics of the objects that is to be found in the search space (Milos, 2018). All Information retrieval system requires search operation to be performed from massive databases implemented by various data structures. The Searching problem deals with finding a given value called a search key, in a given set (or a multiset, which permit several elements to have the same value) (Anany, 2012). Searching algorithm ranges from straight forward sequential search algorithm - a searching techniques which does not expect the data to be arranged in specific order (Chitra, 2013) - to very efficient but limited binary search and algorithms based on representing the underlying set in a different for more productive to searching. Some searching algorithms work faster but require more memory. Some are very fast but are applicable to sorted arrays. A matrix is said to be sorted if all elements in each row and column are sorted in non-decreasing (lexicographical) order, respectively. searching a sorted matrix for the occurrence of a given element (key), which we call the matrix search problem is a common problem that normally arise in image processing and computational biology (Baase, 1988; Cosnar, Duprat, & Ferreira, 1990). In this paper a report is made regarding the performance of 2-D binary search and 'row wise and column wise search' algorithms.

There are two ways to search for data in a list: sequential search and binary search. A sequential search is used when the items in a list are in random order; a binary search is used when the item is used in the item in a list are in sorted order. Binary search is a more efficient algorithm (Francis, 1983), however, the fact that the data has to be sorted before it can be applied was considered as disadvantage by Ancy & Chanchal(2014). Binary search also refers to as half interval searches,

Received 18 June, 2019
Accepted 23 September, 2019
Address Correspondence to:
taiwotheophilus@gmail.com

repeatedly target the center of the search structure and divide the search space in half.

Rehan (2016) developed Grid searching algorithm as an improvement over the existing limitations of the 2D array sequential search. The performance of the developed algorithm was compared with the sequential search algorithm. It was found out that Grid Searching technique is working well for all input values and outperforms Sequential Searching in such aspect as analysis of functions, worst case iterations, worst case Time complexity of both algorithms and total no of conditional checking. Karthick (2016) proposed Odd Even Based Binary Search algorithm whose worst case time complexity is O(log2(N-M) where N is the total number of elements in the array and M is total number of even number if the Key (item being searched for) is odd or total number of odd if the Key is even. The results obtained from the performance graphs, test results and function calls show that Odd Even Based Binary search algorithm is better than classical Binary Search Algorithm. Quadratic Search Algorithm, a variant of Binary Search algorithm was found to be 2 times faster than classical Binary Search (Parveen, 2013). Vimal & Kumbharana (2015) developed a solution in which binary search is applied to linked list. Binary Search is applied to two dimensional sorted array in Ashima, Meenu and Swati (2016).

METHODOLOGY

Two searching algorithms Binary Search and Row wise and Column Wise search Algorithms are applied to sorted matrix of various dimensions. The number of iterations performed by each of the algorithms to find a specified item on the matrix are measured and recorded. The implementation of the algorithms is done using Matrix Laboratory (MatLab) Language). The insertion sort algorithm was applied for sorting of the matrices before Matrix Binary Search (2DBS) and Row-wise-column wise search (RCS) algorithms were applied. The code for the implementation two algorithms is given below.

```
function sortedarray = insertionsort(inputarr,row,col)
 i = 1;
for x = 1:row
   for y = 1:col
      temparr(i) = inputarr(x,y);
      i = i+1;
   end
end
for j = 2:length(temparr)
   value = temparr(j);
   k= j;
   while ((k > 1) && temparr(k-1)>value)
      temparr(k)= temparr(k-1);
      k= k-1;
   end
   temparr(k) = value;
end
i = 1;
for x = 1:row
   for y = 1:col
     inputarr(x,y)= temparr(i);
      i = i+1;
   end
end
sortedarray = inputarr;

function [f,row,col,it,cmp] = Matrix_Binary_Search(arr, cols,last, key)
%[rows,cols] = size(arr);
f = 0;
row = 0;
col = 0;
first = 0;
%last = rows * cols - 1;
it = 0;
cmp = 0;
while((first <= last)&& (f == 0))
   middle = floor((first + last)/2);
   row = floor(middle/cols)+ 1;
   col = mod(middle,cols) + 1;
   it = it + 1;
   cmp = cmp + 1;
   if(arr(row,col)== key)
      f = 1;
      return
   elseif (arr(row,col)< key)
     first =  middle + 1;
   else
      last = middle - 1;
   end
end
return
```

```
function [f, r,c,iterations, comparison] = Row_wise_Column_wise_Search(mat,row,col, key)
    r = 1;
    c = col;
    f = 0;
    iterations = 0;
    comparison = 0;
    it = 0;
    cmp = 0;
    while(r <= row && r>=1 && c>1 && c <= col)
        it = it + 1;
        cmp = cmp + 1;
        if mat(r,c) < key
          r = r+1;
        else
          if mat(r,c) > key
            c = c - 1;
          else
            if mat(r,c) == key
              f = 1;
              iterations = it;
              comparison = cmp;
              return;
            end
          end
        end
    end
    iterations = it;
    comparison = cmp;
    return
```

RESULTS

Performance Analysis Of 2d Binary Search And Row Wise And Column Wise Algorithms

Three metrics are used in analyzing the performance of the two algorithms. The metrics are asymptotic analysis of the function used in the implementation of the algorithm, the worst case iteration complexity and the number of time conditions are checked during when each function is executed.

The Analysis of the Number of Operations Performed by the two Functions.

The analysis are carried out as follows:

1. Analysis of function of binary search:

For a matrix of dimension N x M, the number of elements in such a matrix is N*M.

When N*M = 32 BinarySearch is called to reduce size to N*M = 16

When N *M = 16 BinarySearch is called to reduce size to N*M = 8
When N * M = 8 BinarySearch is called to reduce size to N*M = 4
When N*M = 4 BinarySearch is called to reduce size to N*M = 2
When N*M = 2 BinarySearch is called to reduce size to N*M =1
In general, assuming that the product N*M is a power of 2, then we have

$$2^k = N * M$$

Taking the logarithm of both sides we have

$$\mathrm{K}\log_2 2 = \log_2 N * M$$
$$\therefore K = \log_2 N * M$$

A close look at the code of binary search function shows that for each run of WHILE loop, there are 4 assignment operations and 1 comparison operation. Therefore, the total number of operations is 5. The total number of operations for a matrix of dimension N x M is therefore 5K.

Since $K = \log_2 N * M$ then the total number of operation is equal to $5\log_2(N * M)$
The function f(N,M) for 2D binary search can be written as

$$\mathrm{f(N, M)} = 5\log_2(N \times M)$$

Assuming that N = M then

$$\mathrm{f(N)} = 5\log_2 N^2$$
$$\mathrm{f(N)} = 10\log_2 N$$

2. Analysis of Row wise and column wise search:

For a matrix with dimension N X M, there are (N + M-1) comparison operations and (N + M-1) assignment operations. Therefore, the total number of operations for the Row wise and Column wise function is 2(N + M-1). The function g (N, M) for row wise and column wise search can be written as: $g(N, M) = 2(N + M - 1)$
For N = M,

$$g(N) = 2(2N - 1)$$

Table 1: Analysis of Functions

N	*M*	*Number of elements*	*2D Binary Search*	*Row wise and Column wise Search*
1	1	1	0	2
4	4	16	20	14
5	5	25	20	18
10	10	100	30	38
15	15	225	35	58
20	20	400	40	78
30	30	900	45	118
40	40	1600	50	158
50	50	2500	55	198
60	60	3600	55	238
70	70	4900	60	278
80	80	6400	60	318
90	90	8100	60	358
100	100	10000	65	398

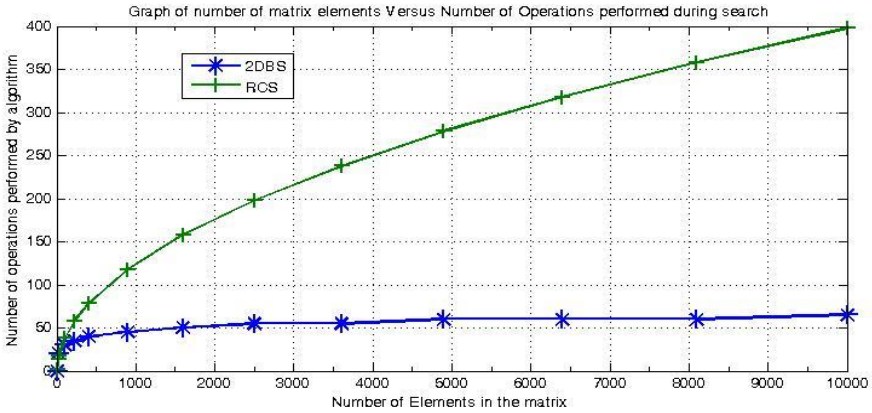

Figure 1: Analysis of Number of operation performed searching functions

Analysis of Worst Case Iteration Complexity and Condition Checking

The Worst-case iteration complexity measures the resources (e.g. running time, memory) that an algorithm requires in the worst-case. It gives an upper bound on the resources required by the algorithm. Thus the worst-case of an algorithm is the total number of iteration(s) required for searching an element which is not present in the 2D array/matrix, Worst Case iteration complexity shows the total no. of iteration required for fulfilling a NOT FOUND condition. Condition checking is the number of conditions required to find a particular element in the matrix irrespective of the presence of the element in the matrix. The program was executed by testing with various array of different dimensions. During the testing it was found out that the number of iterations and the total number of conditions that are checked for all the worst cases have equal values in all cases. Table2 shows the recorded number of iterations /condition checking when a particular element could not be found in the array (worst case).

Table2: Worst Case iteration/Condition Checking

Dimension	Worst Case iterations/ condition checking	
	2DBS	RCS
5 X 5	5	5
10 X10	7	10
20 X 20	8	38
30 X 30	10	58
40 X 40	10	98
50 X 50	11	98
60 X 60	12	116
70 X 70	13	138
80 X 80	13	157
90 X 90	13	177
100 X 100	13	196

Table 3: Worst Case Time Complexity

Dimension	Worst Case time complexity (s)	
	2DBS	RCS
5 X 5	0.00011	0.00013
10 X10	0.00013	0.00016
20 X 20	0.000149	0.000152
30 X 30	0.000146	0.000168
40 X 40	0.000160	0.000175
50 X 50	0.000173	0.000174
60 X 60	0.000160	0.000163
70 X 70	0.000161	0.000161
80 X 80	0.000180	0.000195
90 X 90	0.000181	0.000192
100 X 100	0.000255	0.000257

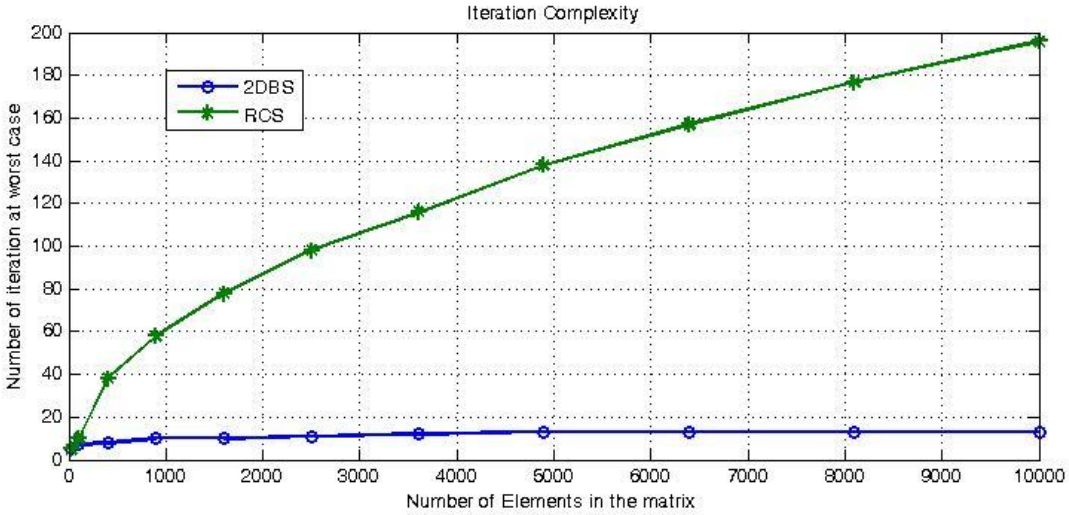

Figure 2: Graph of Worst Cast Iteration sand condition checking of BS2D and RCS

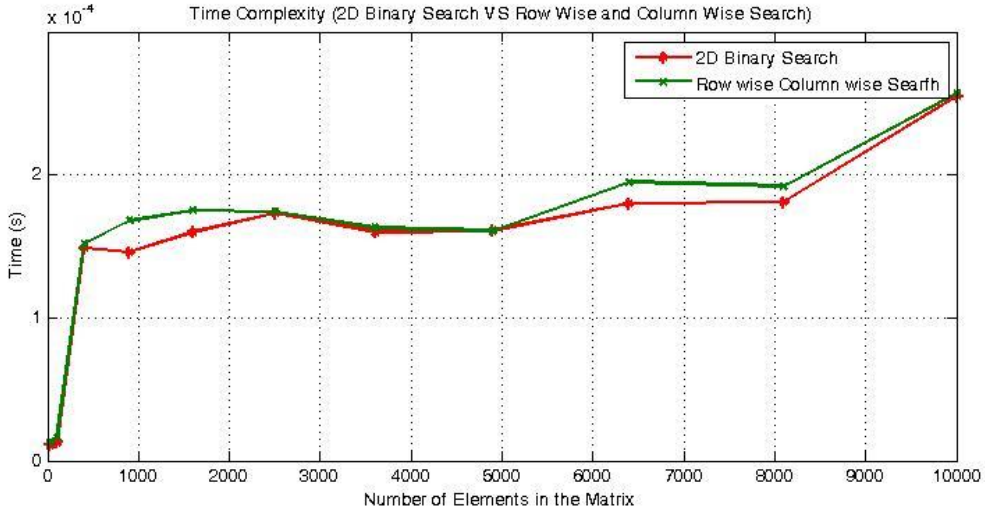

Figure 3: Graph of Worst Case Time Complexity of 2DBS and RCS

Analysis of Worst Case Time Complexity

The worst-case time complexity indicates the longest running time performed by an algorithm given any input of size n (if the algorithm satisfies NOT FOUND condition then that is the longest running time of the algorithm), and thus this guarantees that the algorithm finishes on time and gives us the total worst-case time complexity.

Table 3 shows the recorded time for each algorithm when the program was tested to search for different items using matrix of different dimensions. The time taken to complete the search when 'NOT FOUND' report is made is taken in each case.

DISCUSSION

From the above graphs (Figure 1 and Figure 2) in all cases, 2D Binary Search requires fewer iterations, fewer condition checking and fewer number of operations performed by the function than row wise and column wise search algorithm. Figure 3 shows that the worst case time complexity of 2D Binary Search algorithm is also smaller than that of Row wise and Column wise search Algorithm. It may therefore be said that that 2D Binary Search Algorithm is highly efficient with an increase in the order of matrix with respect to Row Wise Column Wise searching.

Conclusion

Performance analysis of 2D Binary Search algorithm and that of Row wise and Column Wise Search has been performed using analysis of the operations performed by the function that implement each algorithm, the number of iterations of worst cases, the number of condition checking and the worst case time complexity for both algorithms when they are applied to sorted array. It is discovered that 2D Binary search performs better than row wise and column wised search in terms of number of operations performed by the function, worst case time complexity, worst case iterations complexity and number of condition checking.

Recommendation

2D Binary Search algorithm is recommended for application that require searching from sorted matrix..

REFERENCES

Anany, L. (2012). *Introduction to the Design and Analysis of Algorithms* (3rd ed.). New Jersey, Upper Saddle River, USA: Pearson Education Inc.

Ancy, O., & Chanchal, P. (2014). Binary Search Algorithm. *International Journal of Innovative Research in Technology, 1*(5), 800-803.

Ashima, G., Meenu, V., & Swati, G. (2016). Implementation and Application of Binary Search in 2-D Array. *International Journal of Institutional & Industrial Research, 1*(1), 30-31.

Baase, S. (1988). *Computer Algorithms.* Assison-Wesley: Reading MA.

Chitra, R. (2013). *Data Structures Using C.* Bangalore: Subhas Publishers.

Cosnar, M., Duprat, J., & Ferreira, A. (1990). The Complexity of Searching in X+Y and other Multiset. *Information Process*, 103-109.

Francis, S. (1983). *Theory and Problems of Computers and Programming.* Schaum's Outline Series.

Karthick, S. (2016). Odd Even Based Binary Search. *International Journal of Computer Engineering and Technology, 7*(5), 40-55. Retrieved October 2, 2018, from http://www.iaeme.com/ijcet/issues.asp?JType=IJCET &VType=5

Milos, H. (2018). Problem Solving by Searching. *CS 1571 Introduction to AI Lecture 3*. Unpublished.

Parveen, K. (2013). Quadratic Search: A New and Fast Searching Algorithm (An Extension of classical Binary Search Strategy). *International Journal of Computer Applications, 65*(14), 43-46. doi:0975 – 8887

Rehan, G. (2016). Grid Searching : Novel Way of Searching 2D Array. *International Journal of Computer Applications Technology and Research, 5*(1), 26-33.

Vimal, P. P., & Kumbharana, C. K. (2015, July). Comparing Linear Search and Binary Search Algorithms to Search and Element from a LInear List Implemented through Static Array, Dynamic Array and Liniked List. *International Journal of Computer Applications, 121*(3), 13-17.