International Journal of Research In Science & Engineering
Volume: 2 Issue: 6 December 2016
e-ISSN: 2394-8299
p-ISSN: 2394-8280

Run-Time Analysis of Searching and Hashing Algorithms with C#

Sourabh Shastri[1] Prof. Vibhakar Mansotra[2], Arjun Singh[3], Brij Mohan[4]

[1,3,4]*Department of Computer Science & IT, Bhaderwah Campus, University of Jammu, J&K, India*
[2]*Department of Computer Science & IT, University of Jammu, J&K, India*

ABSTRACT

The analysis of algorithms is a subject that has always arouses enormous inquisitiveness. It helps us to determine the efficient algorithm in terms of time and space consumed. There are valid methods of calculating the complexity of an algorithm. In general, a suitable solution is to calculate the run time analysis of the algorithm. The present study documents the comparative analysis of five different searching and hashing algorithms of data structures viz. Linear search, Binary search, Interpolation search, Division method hashing and Mid square method hashing. The implementation is carried out in Visual Studio C# by creating a graphical user interface to calculate the running time of these algorithms for the searching of an element in a randomly generated array.

Keywords: ***Algorithm, Complexity, Searching, Hashing, Running Time, Data Structures***

1. INTRODUCTION

The word algorithm comes from the name of a Persian mathematician, A. J. M ibn Musa al Khwarizmi [1]. The development of an algorithm is elementary to computer programming and is an essential module of computer science studies that can be used for the solution of a problem. After the preparation of an algorithm, the next step is to program the algorithm through a programming language by using mathematical and data processing methods. An algorithm is a well-organized order of instructions for solving a given problem. Developing of the algorithm is the foremost step to solve any problem. Input, Output, Definiteness, Finiteness and Effectiveness are the characteristics of an algorithm that means the algorithm accepts some input, the algorithm generates some output, that each action should be clear and unambiguous, that the algorithm terminates after a finite number of instructions and that each instruction must be sufficiently fundamental respectively [2].

The term analysis of algorithms or complexity is used to find out which of them is efficient in terms of space and time. To analyze an algorithm, it is necessary to know on what inputs the algorithm is taking less time and on what inputs the algorithm is taking more time. The three analysis of an algorithm are worst case, best case and average case. The worst case runtime complexity of the algorithm describes the maximum number of steps taken on any instance of size n. The best case runtime complexity of the algorithm describes the minimum number of steps taken on any instance of size n. The average case runtime complexity of the algorithm describes an average number of steps taken on any instance of size n [3].

Searching is one of the most fundamental operations which are mostly used in computer studies to find out the particular location of an element in a collection of huge volume of data. Efficient searching is significant to get the maximum throughput of the system and finally the system throughput provides utmost efficiency to the users [4]. The complexity of searching algorithms measures the running time of a function in which n number of items are to be stored from which an element or its location is to be searched. Different concepts such as searching sequentially, partitioning the list into two halves etc. are used in different algorithms of searching. Many searching algorithms cannot be made better than O(log n) time complexity even on the average. Hashing is a kind of searching method to achieve better performance by computing the address by applying a hash function to a given key value [5]. A hash function H: K→L is a hash function where K is a set of keys and L is the set of memory addresses [6]. A good hash function must be easy to compute and most importantly minimize collisions.

In this paper, we have created a graphical user interface for the implementation of various searching and hashing algorithms which include linear search, binary search, interpolation search, division method

hashing and mid square method hashing. For calculating the running time of these searching and hashing algorithms, Visual Studio C# language has been selected. We have run each algorithm for searching the same element for ten different runs having different values of input size i.e. N = 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000 and 100000 and finally calculated the average running time for each algorithm separately. User has to input the element to search and the value of N i.e. how many elements are required from which the element is to be searched and a random number generator generates N number of elements randomly.

2. WORKING PROCEDURE OF ALGORITHMS

Searching is one of the most vital algorithm in computer science. Now a day's data increases exponentially hence a very efficient searching algorithm is required to retrieve the data from huge volume. In this section, we discuss the working procedure of all the five algorithms used for the run time analysis in the study viz. linear search, binary search, interpolation search, division method hashing and mid square method hashing.

2.1 Linear Search

Linear search is also known as sequential search in which an element can be searched in a sequential manner among a collection of elements. In this type of searching, searching process starts from one end scans the whole list upto the other end of the list to find the key element. Linear search supports both unordered list as well as ordered list [7]. In an unordered list, we have to scan the whole list to find whether the element is present in the list or not but in an ordered list if any array element exceeds the key element while scanning, it means the list does not contain the key element.

2.2 Binary Search

The binary search algorithm can only be applied on sorted lists, then the main list is divided into two lists and afterwards the key element is searched. It is based on the divide and conquer approach [8]. The sorted list is firstly divided into two parts. Next the key value is compared with the mid element of the main list. If it is not the key element then the element is either less than or greater than the mid value. Accordingly the search will continue to the half of the list where it suits. If the key value is less that the mid element then the search is limited to the left sub list otherwise to the right sub list in case of key value is greater than mid value. At the end, when the key value is found, the list is automatically combined.

2.3 Interpolation Search

Interpolation search is a modified search algorithm of binary search algorithm that also works on ordered array and reduces complexity. If the elements of an array are uniformly distributed then in that case the complexity is O(log log n) but if the elements are not fairly uniformly distributed then complexity comes O(n) [9].

2.4 Division Method

In division method of hashing, the integer key is divided by some number say m which is equal to table size and k is the key [8]. The table size should not be a power of two, otherwise it will give more collision [10]. The remainder value is taken as the hashing value. The method can be expressed as follows: $H(k) = k(\text{mod } m)$ or $H(k) = k(\text{mod } m) + 1$. Second function is used when the hash table is in range from 1 to m rather than 0 to m-1. The number m i.e. memory locations should be greater than the number of records of keys in k i.e keys which uniquely determine records [5].

2.5 Mid-Square method

In mid-square method the key is multiplied with itself and middle digit is chosen as the address of that element in the hashing table. It is defined as follows H(k): k * k [11].

e-ISSN: 2394-8299
p-ISSN: 2394-8280

3. GRAPHICAL IMPLEMENTATION TO MEASURE THE PERFORMANCE OF ALGORITHMS USING C#

We have implemented all the five algorithms to measure the performance using C# language and calculated the running time in milliseconds by using System.Diagnostics.Stopwatch class. The data set for the analysis contains random numbers. We ran all algorithms ten times for each value of input length N (i.e. 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000 and 100000) and tried to find running time of searching the same element. For taking the length of the array, element to search and selection of the algorithm to search an element in a random array, a graphical user interface is created as shown in the figure 1.

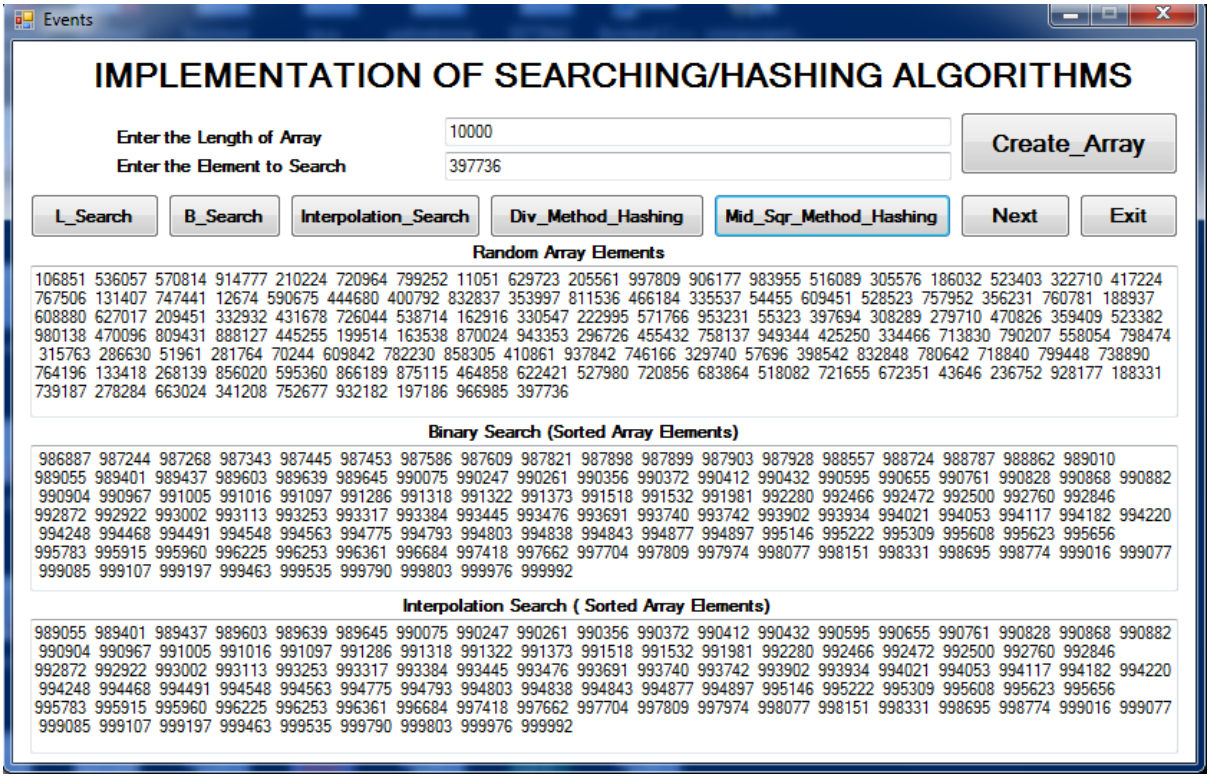

Figure 1: Graphical user interface for choosing length, element and algorithm

Figure 2 displays the graphical view of generating the running time in milliseconds for all the algorithms mentioned above with ten runs. We have also calculated the average running time in milliseconds based upon the running times of ten runs.

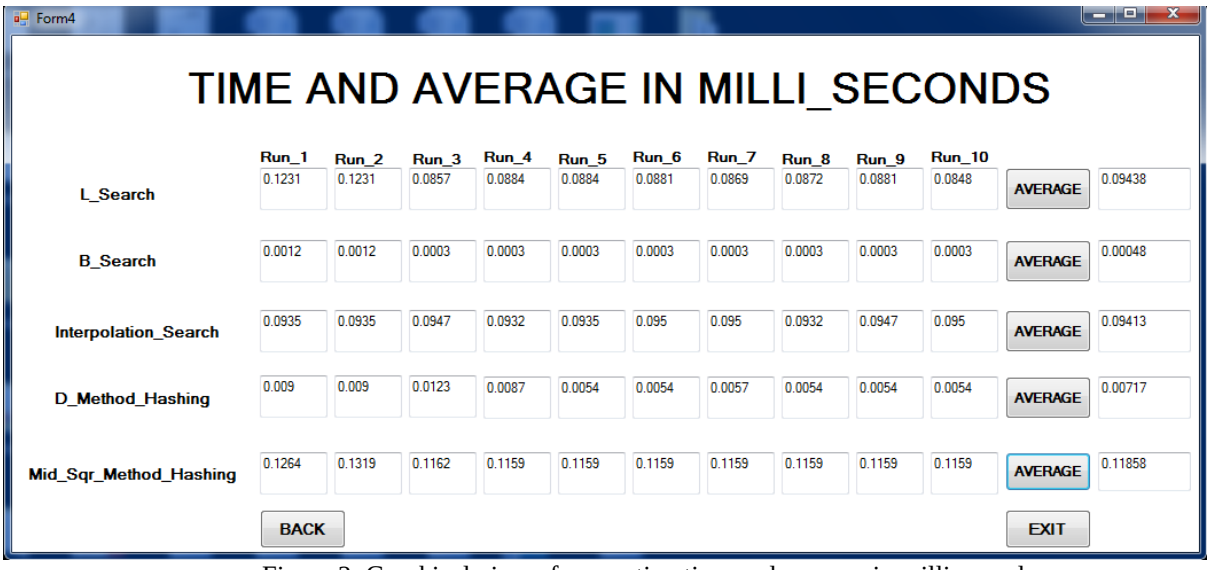

Figure 2: Graphical view of generating time and average in milliseconds

e-ISSN: 2394-8299
p-ISSN: 2394-8280

Table 1-10 displays the running time and average of all the algorithms for ten runs. The running time is calculated for the input length of N i.e. 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000 and 100000 elements.

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.1231	0.1231	0.0857	0.0884	0.0884	0.0881	0.0869	0.0872	0.0881	0.0848	0.09438
B_Search	0.0012	0.0012	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.00048
I_Search	0.0935	0.0935	0.0947	0.0932	0.0935	0.095	0.095	0.0932	0.0947	0.095	0.09413
D_M_Hash	0.009	0.009	0.0123	0.0087	0.0054	0.0054	0.0057	0.0054	0.0054	0.0054	0.00717
M_S_Hash	0.1264	0.1319	0.1162	0.1159	0.1159	0.1159	0.1159	0.1159	0.1159	0.1159	0.11858

Table 1: Running time and average for 10,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.1975	0.1975	0.2482	0.1802	0.1911	0.1808	0.1802	0.1789	0.1802	0.1789	0.19135
B_Search	0.0044	0.0044	0.0012	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0.00142
I_Search	0.8422	0.8422	1.2245	0.8274	0.8133	0.8255	0.8338	0.8249	0.8261	0.805	0.86649
D_M_Hash	0.0699	0.0699	0.0455	0.0499	0.0499	0.0499	0.0499	0.0499	0.0442	0.0449	0.04989
M_S_Hash	0.1488	0.1545	0.1398	0.211	0.2315	0.2289	0.1404	0.211	0.1391	0.211	0.1816

Table 2: Running time and average for 20,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.3912	0.3912	0.2892	0.2682	0.2674	0.2662	0.2739	0.2706	0.2681	0.2681	0.29546
B_Search	0.0032	0.0032	0.0012	0.0006	0.0006	0.0006	0.0006	0.0012	0.0006	0.0006	0.00124
I_Search	0.1693	0.1693	0.1706	0.1706	0.1699	0.1706	0.1699	0.1674	0.1706	0.1706	0.16988
D_M_Hash	0.0102	0.0102	0.0096	0.0096	0.0089	0.0089	0.0089	0.0089	0.0089	0.0089	0.0093
M_S_Hash	0.3521	0.3566	0.3438	0.3431	0.3438	0.3431	0.3771	0.3438	0.3431	0.3438	0.34903

Table 3: Running time and average for 30,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.5163	0.5163	0.481	0.3585	0.3553	0.3566	0.3553	0.3553	0.3566	0.3553	0.40065
B_Search	0.0053	0.0057	0.0012	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0.0006	0.00168
I_Search	0.8441	1.8441	1.6549	1.6459	1.6466	1.6543	2.1161	1.6818	1.6927	1.6543	1.74348
D_M_Hash	0.0865	0.0865	0.0846	0.0846	0.0846	0.0846	0.0846	0.0853	0.084	0.0846	0.08499
M_S_Hash	0.4676	0.4733	0.4592	0.4939	0.4939	0.508	0.6247	0.4634	0.4631	0.4939	0.49047

Table 4: Running time and average for 40,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.483	0.483	0.4483	0.4464	0.6266	0.6427	0.447	0.4445	0.4445	0.477	0.4913
B_Search	0.0064	0.0064	0.0012	0.0006	0.0006	0.0006	0.0012	0.0006	0.0006	0.0006	0.00188
I_Search	0.6395	0.6395	0.6337	0.63317	0.6177	0.6177	0.6716	0.6286	0.617	0.6639	0.636222
D_M_Hash	0.0538	0.0538	0.0532	0.0532	0.0532	0.0532	0.0532	0.0532	0.0532	0.0525	0.05325
M_S_Hash	0.3842	0.3893	0.3457	0.3444	0.3803	0.3771	0.3444	0.3701	0.381	0.3784	0.36949

Table 5: Running time and average for 50,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.5708	0.5708	0.5375	0.5311	0.5311	0.5304	0.5304	0.5304	0.5304	0.5272	0.53901
B_Search	0.0064	0.0064	0.0012	0.0006	0.0012	0.0006	0.0006	0.0006	0.0006	0.0012	0.00194
I_Search	2.0609	2.0609	1.9429	1.975	1.9673	1.966	1.9532	2.3457	1.9949	1.9718	2.02386
D_M_Hash	0.1751	0.1751	0.1738	0.1738	0.1725	0.1744	0.1744	0.1738	0.1808	0.1738	0.17475
M_S_Hash	0.6959	0.7023	0.6863	0.6857	0.685	0.7075	0.6869	0.6863	0.6857	0.6857	0.69073

Table 6: Running time and average for 60,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.6741	0.6741	0.6266	0.6446	0.7011	0.6254	0.6273	0.626	0.6273	0.6266	0.64531
B_Search	0.0038	0.0038	0.0012	0.0025	0.006	0.0006	0.0012	0.0006	0.0012	0.0006	0.00161
I_Search	4.5023	4.5023	3.6158	3.5562	0.5844	3.9231	3.6165	3.5542	3.5574	3.5818	3.7994
D_M_Hash	0.3232	0.3232	0.3207	0.3136	0.2995	0.3162	0.3162	0.2989	0.3162	0.3566	0.31843
M_S_Hash	0.8133	0.821	0.8024	0.8018	0.8018	0.8018	0.8024	0.8364	0.8011	0.88	0.8162

Table 7: Running time and average for 70,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.7588	0.7588	0.9788	0.7068	1.0282	0.8165	0.7075	0.7075	1.0025	0.9268	0.83922
B_Search	0.0064	0.0064	0.0012	0.0012	0.0006	0.0006	0.0012	0.0012	0.0012	0.0006	0.00206
I_Search	2.0744	2.0744	1.9756	2.0077	2.0077	1.9885	2.2874	2.0494	2.0237	2.0077	2.04965
D_M_Hash	0.1064	0.1064	0.1058	0.1058	0.1058	0.1045	0.1051	0.1045	0.1058	0.1045	0.10546
M_S_Hash	0.9262	0.9345	0.9159	0.9147	0.9608	0.9519	0.9159	0.9147	0.923	0.9153	0.92729

Table 8: Running time and average for 80,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	0.8556	0.8556	0.7896	0.796	0.796	0.7966	0.796	0.873	0.8358	0.796	0.81902

e-ISSN: 2394-8299
p-ISSN: 2394-8280

B_Search	0.0051	0.0051	0.0012	0.0012	0.0012	0.0006	0.0006	0.0012	0.0012	0.0012	0.00186
I_Search	4.3201	4.3201	4.1534	4.1809	4.4131	4.2072	4.206	4.206	4.2464	4.154	4.24072
D_M_Hash	0.2238	0.2238	0.2206	0.22	0.2213	0.2206	0.2213	0.2206	0.2206	0.2206	0.22132
M_S_Hash	1.0628	1.0699	1.032	1.0314	1.0372	0.0314	1.1212	1.1212	1.3808	1.0308	1.05687

Table 9: Running time and average for 90,000 elements

Algorithm	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10	Average
L_Search	1.3528	1.3528	0.9435	0.8973	0.8954	0.8954	0.9647	2.3406	1.2072	0.9692	1.18189
B_Search	0.0044	0.0044	0.0012	0.0006	0.0012	0.0006	0.0006	0.0006	0.0012	0.0006	0.00154
I_Search	3.7762	3.7762	3.7922	3.7358	4.0507	3.7544	3.7274	3.7531	3.8307	3.7557	3.79524
D_M_Hash	0.338	0.338	0.3316	0.329	0.3746	0.3143	0.3438	0.4053	0.3848	0.3322	0.34916
M_S_Hash	1.1751	1.1809	1.1437	1.1424	1.1417	1.2912	1.1988	1.1424	1.2206	1.2931	1.19299

Table 10: Running time and average for 100,000 elements

4. RESULTS AND DISCUSSIONS

From the tables 1-10, it is very much clear that in binary search and division method hashing, it takes less time to search an element from a random array as compared to linear search, interpolation search and mid square method hashing. The five algorithms used for the investigation were run ten times for the same length of random array and their average was also calculated. We have taken different lengths of random array i.e. 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000 and 100000. Figure 3 displays the average time taken in milliseconds by all the five algorithms taken into consideration during the present study. Out of all results, the binary search is taking the least time in all the cases and hence is the most efficient among other algorithms discussed in the study.

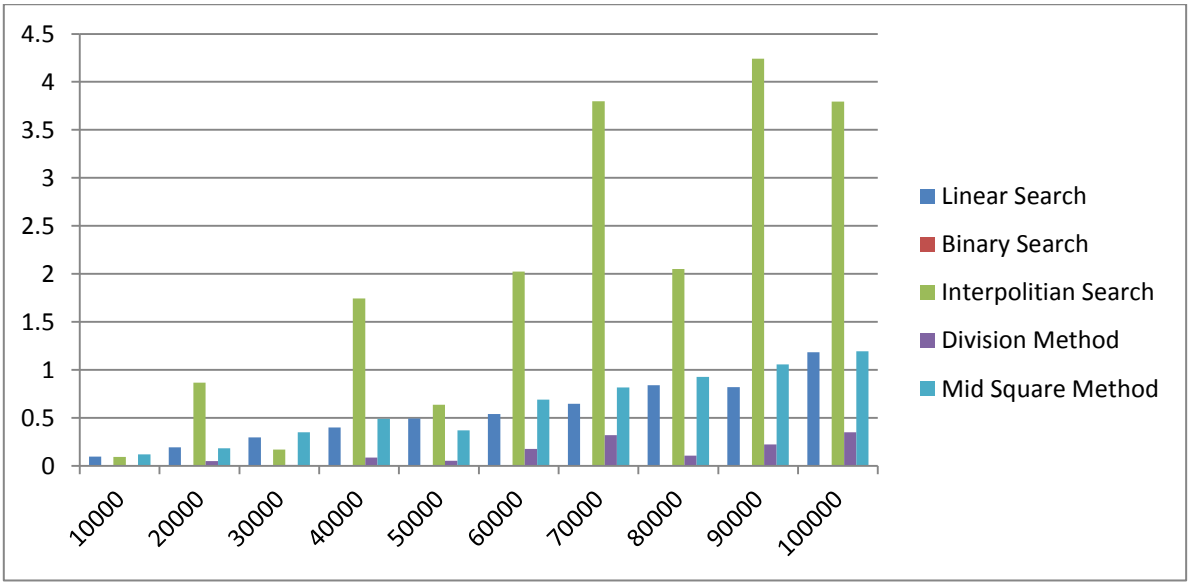

Figure 3: Average running time of algorithms in milliseconds having N different random elements

5. CONCLUSION

In this study, we have studied about various searching and hashing algorithms and their comparison. There are advantages and disadvantages in all algorithms. Binary search has benefit of time complexity over linear search. Binary search has worst-case complexity of O(log n) whereas linear search has O(n). Interpolation search is an improved variant of binary search but the data collection should be equally distributed in addition to sorted form. Interpolation search works better than binary search for uniformly distributed array. To find the running time of all algorithms, we have created a graphical user interface using C# language in which data elements are randomly chosen for different data sets of N = 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000 and 100000. The running time for each algorithm and their average is calculated and displayed graphically. The average running time of all algorithms is shown with the help of a graph in figure 3. From the tables 1-10 and figure 3 comparison, it is clear that binary search is the best and efficient for large data sets among all the algorithms discussed in the study.

REFERENCES

[1] S. K. Basu, *Design Methods and Analysis of Algorithms*, First ed., India: PHI Learning Private Limited, 2008.
[2] Yashavant Kanetkar, Data *Structures through C*, 2nd ed., India: BPB Publications, 2010.
[3] Sourabh Shastri, Vibhkar Mansotra and Anand Sharma, "Sorting Algorithms and their Run-Time Analysis with C#", *International Journal of Computer Science and Information Technology & Security*, vol. 5, issue. 5, pp. 388-395, Oct. 2015.
[4] Muhammad Usman, Zaman Bajwa and Mudassar Afzal, "Performance Analysis of Searching Algorithms in C#", *International Journal of Research in Applied Science & Engineering Technology*, vol. 2, issue 12, pp. 511-513, Dec. 2014.
[5] ISRD Group, *Data Structures using C*, 2nd ed., India: The McGraw-Hill Companies, 2006.
[6] Seymour Lipschutz, G A Vijayalakshmi Pai, Data *Structures*, Special Indian ed., India: Tata MacGraw-Hill, 2006.
[7] Narasimha Karumanchi, *Data Structures and Algorithms Made Easy*, 2nd ed., India: CareerMonk Publications.
[8] Kamlesh Kumar Pandey and Narendra Pradhan, "A Comparison and Selection on Basic Type of Searching Algorithm in Data Structure", *International Journal of Computer Science and Mobile Computing*, vol. 3, issue 7, pp.751-758, July 2014.
[9] Debadrita Roy and Arnab Kundu, "A Comparative Analysis of Three Different Types of Searching Algorithms in Data Structure", *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 3, issue 5, pp.6626-6630, May 2014.
[10] S. K. Srivastava and Deepali Srivastava, *Data Structures through C in depth*, First ed., India: BPB Publications, 2010.
[11] Ashok N. Kamthane, *Introduction to Data Structures in C*, First ed., India: Pearson Education, 2009.