Simulating Binary Search Technique Using Different Sorting Algorithms

B. G. Balogun
Department of Computer Science
University of Ilorin
Ilorin, Nigeria

J. S. Sadiku
Department of Computer Science
University of Ilorin
Ilorin, Nigeria

Abstract

In this study, the binary search method is simulated using twenty different data sets. The data are all numeral and generated randomly. The objective is to find out which sorting method should be used if there is need to apply binary search method on unsorted data. Another objective is to establish the extent to which binary search is efficient. To achieve these objectives a C# code is developed. The code generates the required random numbers, it also sorts them using bubble sort, insertion sort, and quick sort techniques before applying the binary search algorithm. The internal clock of the computer is set to monitor the time durations of the computations. The results show that except for small data sets both the bubble and insertion sort methods should not be employed. Rather the quick sort method should be used to sort large data sets. This agrees with the existing asymptotic analysis of the complexities of these sorting methods. However, the insertion sort performs better than does bubble sort. It is also observed that linear search may be preferred to binary search if searching is to be carried out on unsorted data.

Keywords*:* Binary search, simulation, sorting, algorithms

1. Introduction

Many of the tasks of computer science in general, and artificial intelligence in particular can be phrased in terms of a search for the solution to the problem at hand. Indeed, basic search techniques provide the key to many historically important accomplishments in the area of artificial intelligence.

These include applications in such areas as (Firebaugh 1987)

- Board games and puzzles (Tic-tac-toe, chess, Towers of Hanoi)
- Scheduling and routing problems (Traveling salesman Problem)
- Language parsing interpretation (search for structure and meaning)
- Logic programming (search for facts and implications)
- Computer vision and pattern recognition
- Rule-based expert systems

An important property of most of the significant search problems studied in artificial intelligence is that they suffer from combinatorial explosion. That is, the number of states which must be searched generally grows very rapidly with the size and complexity of the system studied. Various strategies for effective search have emerged; some of them are;

- Binary search
- Breadth – First Search
- Depth – First Search
- Hill – Climbing heuristic
- Best – First heuristic

In this study we consider binary search technique. This technique has been found to be a very efficient method of searching. It is a divide-and-conquer method; its disadvantage is that it cannot be applied on unsorted data.

A binary search problem can be stated as follows:

Let a_i, $1 \leq i \leq n$ be a list of elements which are sorted in non-decreasing order. Consider the problem of determining whether a given element x is present in the list.
In case x is present, we are to determine a value j such that $a_j = x$. if x is not in the list then j is to be set to zero.
Divide-and-conquer suggests breaking up any instance $I_1 = (k\text{-}l, a_1, \ldots a_{k-1}, x)$, $I_2 = (1, a_k, x)$ and $I_3 = (n\text{-}k, a_{k+1}, \ldots a_n, x)$.

The search problem for two of these three instances is easily solved by comparing x with a_k. If $x = a_k$ then $j = k$ and I_1 and I_3 need not be solved. If $x<a_k$ then for I_2 and I_3, j = o and only I_1 remains to be solved. If $x>a_k$, then for I_1 and I_2, j= 0 and only I_3 remains to be solved. After a comparison with a_k, the instance remaining to be solved (if any) can be solved by using this divide-and-conquer scheme again. If k is always chosen such that a_k is the middle element then the resulting search algorithm is known as binary search (Ellis Horowitz, 1978; Mark Allen, 2007)

```
Procedure BINSRCH (A, n, x, j)
// given an array A(1:n) of elements in nondecreasing order,//
// n>=o, determine if x is present, and if so, set j such that x =A(j)//
// else j = o.//
Integer low, high, mid, j, n;
low <—— 1;     high <—— n
while low <= high do
mid <— [(low + high)/2]
Case
                  : x<A (mid): high<  ——mid – 1
                  : x<A (mid): low  <  ——mid + 1
                  : else: j<——mid; return
          endcase
      repeat
      j <——0
  endBINSRCH
```

Algorithm 1: Binary Search

2. *Research Motivation and Methodology*

Binary search has been found to be very efficient searching technique for sorted data (Francis Shield; 1983). This assertion however does not state categorically the method used in accomplishing the sorting in the first instance, meaning that all has not been said about the efficiency of the binary search. The efficiency, given that the data items have been sorted is O(logN) while the efficiency of the linear search is O(N).

It is necessary therefore to take into consideration the efficiency of the sorting technique employed in sorting the data before applying binary search. This is the focus of this study. For this purpose three sorting techniques are considered namely: bubble sort, insertion sort and quick sort. They are applied one at a time on the sets of data items used on the study before the application of binary search technique. The objective is to know at what point should the sorting methods be used in order to get optimal efficiency. In other words, should we always use quick sort, bubble sort or insertion sort (among others)?

To achieve the objective 20 different data sets are generated randomly. The sizes of the sets (all of which are numeral) are 50, 100, 200, 500, 1000, 2000, 5000, 10,000, 20,000, 30,000, 40,000, 50,000, 60,000, 70,000, 80,000, 90,000, 100,000, 200,000, 300,000 and 400,000.

A C# code to accomplish the following is then written.

- Generate random numbers for the 20 sets listed above
- Implement bubble sort
- Implement insertion sort
- Implement quick sort
- Implement linear search
- Implement binary search

The code also sets the system clock to know the duration of time taken by different computations.
The following computations are done and the time used on them by the computer noted.

BinBS	=	Bubble sort based binary search
BinQS	=	Quick sort based binary search
BinIS	=	Insertion sort based binary search
BS	=	Binary search only
Lin	=	Linear search only

3. Results and Analysis

The C# code and the outputs are presented below. The outputs are grouped into two. The first group consists of data sets 50 to 100000; while the second group consists of data sets 100000 to 400000. The second group is considered as containing fairly large sets and so only the quick sort results in addition to the BS and Lin results are displayed.

3.1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace TestCompareNew
{class Program
   { static void Main(string[] args)
      {Stopwatch s = new Stopwatch();
         int MaxData = 100000;
Repeat: Console.Clear();
int[] DataSet = { 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000,
90000,MaxData };
Random random = new Random();
Console.WriteLine("|-------|--------------|----|--------------|--------------|");
Console.WriteLine("|-------|--------------|--This will return only time spent in millisecond--|--------------|-----------
---|");
Console.WriteLine("|-------|----------------------|----------------------|----------------------|----------------------|");
Console.WriteLine("|Set\t|\tBinBS\t\t|\tBinQS\t\t|\tBinIS\t\t|\tLin\t\t    |");
Console.Write("|-------|----------------------|----------------------|----------------------|----------------------|");
int[] NumArray = new int[MaxData];
NumArray = RandomNumber(MaxData);
for (int i = 0; i < DataSet.Length; i++)
{
int ind = DataSet[i] - 1;
///////////////////////////////////////////////////////
///////////////////////////////////////////////////////

/////////////Bubble sort and Binary Search starts/////////
  int[] NumArray1 = PickRandomNumber(NumArray, 0, DataSet[i]);
                s.Start();
                int[] ArrayNum = BubbleSort(NumArray1);
                int SearchedVal = BinarySearch(ArrayNum, NumArray1[ind]);
s.Stop();
```

```
Console.Write("\n|" + DataSet[i]);
Console.Write("\t|\t" + s.Elapsed.TotalMilliseconds+"\t");

/////////////Bubble sort and Binary Search ends/////////

/////////////Quick sort and Binary Search starts/////////
int[] NumArray2 = PickRandomNumber(NumArray, 0, DataSet[i]);
s.Restart();
ArrayNum = quickSort(NumArray2, 0, NumArray2.Length - 1);

SearchedVal = BinarySearch(ArrayNum, NumArray2[ind]);
                    s.Stop();

Console.Write("\t|\t" + s.Elapsed.TotalMilliseconds + "\t");

/////////////Quick sort and Binary Search ends/////////

/////////////Insertion sort and Binary Search starts/////////
int[] NumArray3 = PickRandomNumber(NumArray, 0, DataSet[i]);
s.Restart();
ArrayNum = InsertSort(NumArray3);

SearchedVal = BinarySearch(ArrayNum, NumArray3[ind]);
s.Stop();

/////////////Insertion sort and Binary Search ends/////////
Console.Write("\t|\t" + s.Elapsed.TotalMilliseconds + "\t");

/////////////Linear Search starts/////////
int[] NumArray4 = PickRandomNumber(NumArray, 0, DataSet[i]);
s.Restart();

SearchedVal = LinearSearch(NumArray4, NumArray4[ind]);
s.Stop();

Console.WriteLine("\t|\t" + s.Elapsed.TotalMilliseconds + "\t\t|");
Console.Write("|-------|----------------------|----------------------|----------------------|----------------------|");

/////////////Linear Search ends/////////
                }
            Console.WriteLine("\nDo you want to repeat the simulation? 1 -- Yes; 0 -- No ");
            string resp = Console.ReadLine();
            if (resp == "1")
            {
                goto Repeat;
            }
            else if (resp == "0")
            {
                Console.WriteLine("Good Bye");
                Console.ReadLine();
            }
        }
        public static int[] RandomNumber(int ListNum)
        {
            Random randomNew = new Random();
            int[] Number = new int[ListNum];
 for (int i = 0; i < ListNum; i++)
            {
```

```
            int num = randomNew.Next();
            Number[i] = num;
        }
        return Number;
    }

    public static int[] PickRandomNumber(int[] FromArray, int init, int ListNum)
    {
        Random randomNew = new Random();
        int[] Number = new int[ListNum];
        int icount = 0;
        for (int i = init; i < ListNum; i++)
        {
            Number[icount] = FromArray[i];
            icount++;
        }
        return Number;
    }
    //////// BubbleSort
    public static int[] BubbleSort(int[] array)
    {
        int temp;
        int j;
        int i = array.Length - 1;
        while (i > 0)
        {
            int swap = 0;
            for (j = 0; j < i; j++)
            {
                if (array[j].CompareTo(array[j + 1]) > 0)
                {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swap = j;
                }
            }
            i = swap;
        }
        return array;
    }

    //??/////Insertion Sort
    static int[] InsertSort(int[] array)
    {
        int i, j;
        for (i = 1; i < array.Length; i++)
        {
            int value = array[i];
            j = i - 1;
while ((j >= 0) && (array[j].CompareTo(value) > 0))
            {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = value;
        }
        return array;
    }
    //////// QuickSort
    private static void quickSwap(int[] Array, int Left, int Right)
    {
        int Temp = Array[Right];
        Array[Right] = Array[Left];
        Array[Left] = Temp;
    }
    public static int[] quickSort(int[] Array, int Left, int Right)
    {
        int LHold = Left;
        int RHold = Right;
        Random ObjRan = new Random();
        int Pivot = ObjRan.Next(Left, Right);
        quickSwap(Array, Pivot, Left);
        Pivot = Left;
        Left++;
        while (Right >= Left)
        {
            int cmpLeftVal = Array[Left].CompareTo(Array[Pivot]);
            int cmpRightVal = Array[Right].CompareTo(Array[Pivot]);

            if ((cmpLeftVal >= 0) && (cmpRightVal < 0))
            {
                quickSwap(Array, Left, Right);
            }
            else
            {
                if (cmpLeftVal >= 0)
                {
                    Right--;
                }
                else
                {
                    if (cmpRightVal < 0)
                    {
                        Left++;
                    }
                    else
                    {
                        Right--;
                        Left++;
                    }
                }
            }

        }
        quickSwap(Array, Pivot, Right);
```

```
            Pivot = Right;
            if (Pivot > LHold)
            {
                quickSort(Array, LHold, Pivot);
            }
            if (RHold > Pivot + 1)
            {
                quickSort(Array, Pivot + 1, RHold);
            }

            return Array;
        }
        public static int BinarySearch(int[] array, int value)
        {
            int low = 0, high = array.Length - 1, midpoint = 0;

            while (low <= high)
            {
                midpoint = low + (high - low) / 2;

                // check to see if value is equal to item in array
                if (value == array[midpoint])
                {
                    return midpoint;
                }
                else if (value < array[midpoint])
                    high = midpoint - 1;
                else
                    low = midpoint + 1;
            }

            // item was not found
            return -1;
        }
        public static int LinearSearch(int[] array, int item)
        {
            int searchItem = item;

            int len = array.Length;
            for (int j = 0; j < len; j++)
            {
                if (array[j] == searchItem)
                {
                    return j;

                }
                if (j == len - 1)
                {
                    return -1;
                }
            }
            return -1;

        }
    }
}
```

```
|-------|---------------|----|---------------|---------------|
|-------|--------------|--This will return only time spent in millisecond--|--------------|--------------|
|-------|----------------------|----------------------|----------------------|----------------------|
|Set  |      BinBS          |      BinQS         |      BinIS          |      Lin            |
|-------|----------------------|----------------------|----------------------|----------------------|
|50    |       0.9674         |      1.6193         |      0.5427         |      0.3727         |
|-------|----------------------|----------------------|----------------------|----------------------|
|100   |       0.4997         |      0.6948         |      0.0538         |      0.0032         |
|-------|----------------------|----------------------|----------------------|----------------------|
|200   |       0.5023         |      1.5506         |      0.1751         |      0.0051         |
|-------|----------------------|----------------------|----------------------|----------------------|
|500   |       4.0547         |      3.8436         |      1.072          |      0.0083         |
|-------|----------------------|----------------------|----------------------|----------------------|
|1000 |       12.6108        |      8.0684         |      4.4294         |      0.0147         |
|-------|----------------------|----------------------|----------------------|----------------------|
|2000 |       50.6395        |      16.099         |      17.1159        |      0.0256         |
|-------|----------------------|----------------------|----------------------|----------------------|
|5000   |   321.5405        |      37.1292        |      131.5038       |      0.0603         |
|-------|----------------------|----------------------|----------------------|----------------------|
|10000  | 1285.5006       |      71.8507        |      436.5314      |      0.1174         |
|----------|--------------------|----------------------|----------------------|----------------------|
|20000  |      5150.9551  |      168.4701       |      1741.1228      |      0.2328         |
|----------|--------------------|----------------------|----------------------|----------------------|
|30000  |    11638.3872  |      226.806        |      4145.6249      |      0.6781         |
|----------|--------------------|----------------------|----------------------|----------------------|
|40000  |  20414.1276     |      292.9878       |      6933.4839      |      0.4709         |
|----------|--------------------|----------------------|----------------------|----------------------|
|50000  |    31203.4899   |      372.8973       |    10917.2642      |      0.5787         |
|----------|--------------------|----------------------|----------------------|----------------------|
|60000  |  45536.2483     |      468.4171       |      16241.34       |      0.6961         |
|----------|--------------------|----------------------|----------------------|----------------------|
|70000  |  63380.0997     |      505.5233      |    21143.4313      |      0.818          |
|----------|--------------------|----------------------|----------------------|----------------------|
|80000  |  82410.0914     |      622.7206       |      29061.6939   |      0.9232         |
|----------|-------------------- |----------------------|----------------------|----------------------|
|90000  | 112526.9173    |      700.1806      |  36341.1778       |      1.0643         |
|---------|---------------------|----------------------|----------------------|----------------------|
|100000 | 126516.8047   |      774.3558      |  45896.8454       |      1.158          |
|-------|-----------------------|----------------------|----------------------|----------------------|
Do you want to repeat the simulation? 1 -- Yes; 0 – No
|--Return only Qick with Binary Search, Binary Search Only and Linear Search--|
|-----------|------------------------------|------------------------------|------------------------------|
|Set        |      BinQS                   |      BS                      |      Lin                     |
|-----------|------------------------------|------------------------------|------------------------------|
|100000  |       819.4682              |      0.0032                  |      1.67                   |
|-----------|------------------------------|------------------------------|------------------------------|
|200000  |       1637.5814             |      0.0032                  |      2.3353                 |
|-----------|------------------------------|------------------------------|------------------------------|
|300000  |       2477.2522             |      0.0057                  |      3.5299                 |
|-----------|------------------------------|------------------------------|------------------------------|
|400000  |       3267.3536             |      0.0032                  |      4.6533                 |
|-----------|------------------------------|------------------------------|------------------------------|

Do you want to repeat the simulation? 1 -- Yes; 0 – No
```

Fig. 2 Output of the C# code

3.2 Analysis of Results

For data set 50 both the bubble sort based binary search and insertion sort based binary search take less time than does the quick sort based binary search. The same observation is true for data sets 100 and 200. And except for the data set 100 insertion sort performs better than bubble sort! This implies that except for small data sets 200 and below, bubble sort is not efficient. Quick sort on the other hand is not impressive when used on small data sets (500 and below), it is very efficient for large data sets.

These results tally with the asymptotic results where bubble sort has order N complexity, the insertion sort also has order N complexity and quick sort has order NlogN complexity. The noticed superiority of insertion sort noted here is not visible in the asymptotic results. This should not be surprising since asymptotic approach is analytical. Furthermore the definition of big Oh disregards the effect of constants. Hence O(N/2) and O(N/4) are both regarded as O(N) (Richard and Marcus, 2004).

For data sets 100,000, 200,000, 300,000 and 400,000, the quick sort method out performs both the insertion sort and bubble sort. We can deduce that for small data sets either bubble sort or insertion sort can be used as a basis for sorting before applying binary search technique. However for large data sets quick sort should be used as the basis.

We also observe from the output that binary search method on its own takes very little time to search once the items are sorted. However, linear search outperforms the binary search if sorting time required in binary search is taken into consideration (Glenn J., 2007).

4. Conclusion

In this study, the subsisting assertion that quick sort is superior to both the bubble sort and insertion sort in general, has been corroborated. We also note that insertion sort is more efficient than bubble sort. It is also observed that linear search may be preferred if the data items to be searched have not been sorted. The efficiency implicit in the binary search assumes that the items to be searched have been sorted but this may not be true in all cases. Finally, if there is need to sort data before being searched, then it is recommended that both bubble sort and insertion sort methods can be used for small data sets while quick sort should be used otherwise.

C# is used for coding in this study, it is therefore recommended that other programming languages should also be used in order to carry out a comparative analysis of the study.

References

Ellis Horowitz and Sartaj Sahni (1978) Fundamentals of Computer Algorithms; Computer Science Press, Inc. U.S.A.
Firebaugh (1987): Artificial Intelligence; A knowledge – Based Approach.
Francis Scheid (1983): Theory and Problems of Computers and Programming, Schaum's Outline Series
Glenn J. (2007): Computer Science; an Overview Tenth Edition, Pears on Education, New Jersey, U.S.A.
Mark Allen Weiss (2007): Data Structures and Algorithm Analysis in Java; Pearson, Addison Wesley.
Richard J. and Marcus S. (2004): Algorithms; Pearson, Practice Hall.