Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

IS BINARY SEARCH TECHNIQUE FASTER THAN LINEAR SEARCH TECHNIQUE?

Asagba P. O, Osaghae E. O. and Ogheneovo, E. E.
Department of Computer Science,
University of Port Harcourt,
Choba, Port Harcourt, Rivers State.

Received:03.02.10
Accepted: 2.6.10

ABSTRACT
Over the years, computer scientists have conducted researches on linear and binary search algorithms, and concluded that binary search algorithm is faster than linear search algorithm. Although it was a good observation yet, the sorting time before the application of binary search appeared not to have been considered. This paper critically tries to conduct experiments on the execution time of linear search, binary search without inclusion of sorting time, and binary search with the inclusion the sorting time.

Keywords: Search algorithms, Linear search, Binary search, Bubble sort

INTRODUCTION

Searching a list for a particular item is a common task in computer algorithm. However, in real life applications, the number of elements in the list is implemented as an array of objects. The goal is to find a particular item in a list of items, by searching the entire list or indicate that the item being searched for is not in the list. In information processing, searching has two distinct tasks that can be recognized: the process of storing a key; where a key is entered into the table and the process of retrieval of a key, where a key is accessed in the table (Collins, 1992). Searching is one of the most time-consuming processes of many processing systems H,owever, there is a great need for efficient search algorithms (Horowitz and Sahni, 1978).

The sequential search algorithm, is the simplest method for searching when the keys are stored as an array and then looked up through the array sequentially, for every search key K. If K(k) is found such that, K=K(k), then the search is successful and k is returned and, if no key in the array matches the key k, the search is unsuccessful. There are two basic types of search algorithms, which are linear search and binary search algorithms. The linear search algorithm searches for a particular item on a list of items and the average case of the search is O(n) and it is very efficient for searching a key in an array of less than 200 elements. The binary search algorithm cannot search for an item in a list except the list has to be sorted either in ascending or descending order. Binary search algorithm is efficient because it is based on divide-and-conquer strategy; which divides the list into two parts and searches one part of the list thereby reducing the search time. The computing time of binary search is O(log n) (Hubbard, 2000; Horowitz and Sahni, 1978).

In real applications, the list of items are often records (e.g., student records), and the list is implemented as an array of objects. Given a collection of objects, the goal of searching, is to find a particular object in the collection or to recognize that the object does not exist in the collection. Often, the objects have key values on which one searches the data values which correspond to the information one wishes to

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

retrieve, once an object is found (Horowitz and Sahni, 1978).

LINEAR SEARCH ALGORITHM

The simplest method for searching a list of elements is the linear (or sequential) search, which simply checks the first item, then the second item, and so on until it finds the target item or reach the end of the list. The algorithm of linear search is as follow:

```
Location = -1
i = 0
while (list[i] != target) and (i < list_length) i = i + 1
if (list[i] == target)
location = i
```

A while loop is appropriate since the target could be anywhere in the array or perhaps not in the list at all. The program must be able to exit the search loop as soon as the target is located and to iterate over the entire item in the array, if necessary. The initial value of the control variable i is 0 and increments by 1 each time through the loop, since valid index value ranges from 0 to the number of items (list length) minus one. The loop repeats as long as the target is not found and i is less than the number of items. The expression list [i] != target tests whether the value of the target variable is the same as that of the current list of elements. The condition that allows the loop to continue is the fact that the current element of the list is not the same as the value of target (Rodger, 2003).

For a small list, linear search may be faster because of the speed of the simple increment compared with the division needed in binary search. There are two types of linear search which are unordered linear search, and ordered linear search. The advantages of linear search are: Linear search examines each list item in turn until the target is located or the end of the list is reached. Linear search is the simplest search algorithm, which simply examines each element of the list in order. Linear search can also be used directly on any unprocessed list and the list does not have to be sorted.

BINARY SEARCH ALGORITHM

The strategy of binary search is to check the middle (approximately) item in the list. If it is not the target and the target is smaller than the middle item, the target must be in the first half of the list. If the target is larger than the middle item, the target must be in the last half of the list. Thus, one successful comparison reduces the number of items left to check by half. The search continues by checking the middle item in the remaining half of the list. If it is not the target, the search narrows to the half of the remaining part of the list that the target could be in. The splitting process continues until the target is located or the remaining lists consist of only one item. If that item is not the target, then it is not in the list.

Here are the main steps of the binary search algorithm, expressed in pseudo code:

```
Procedure BINSRCH
// given an array A (1:n) of elements in nondecreasing order.//
// n ≥ 0, determine if x is present, and if so, set j such that x = A (j) //
// else j = 0. //
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
```

End BINSRCH (Horowitz and Sahni, 1978).

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

The advantage of binary search are:
(i) Binary search uses the result of each comparison to eliminate half of the list from further searching.
(ii) Binary search reveals whether the target is before or after the current position in the list, and that information can be used to narrow the search.
(iii) Binary search is significantly better than linear search for large lists of data.
(iv) Binary search maintains a contiguous subsequence of the starting sequence where the target value is surely located.

The advantages of binary search are:
(i) Binary search can interact poorly with the memory hierarchy (i.e. caching), because of its random-access nature. For in-memory searching, if the interval to be searching is small, a linear search may have superior performance simply because it exhibits better locality of reference.
(ii) Binary search algorithm employs recursive approach and this approach requires more stack space.
(iii) Programming binary search algorithm is very difficult and error prone (Kruse, 1999).

BUBBLE SORT ALGORITHM
Bubble sort algorithm is based on the idea that each comparison of two keys is allowed systematically by interchanging the pairs of keys that are out of order, until no more such pairs are left. The algorithm works as follows. Given a sequence of unordered keys, K(1), K(2), . . ., K(n), generally not all distinct, the bubble sort keeps scanning through the sequence, exchanging adjacent keys, if necessary when no exchanges are required on some scan, the sequence is sorted. The following is a pseudo-code representation of Bubble sort:

```
Repeat
 Pivot = A[1]
 For k := 2 to n do
85          ،[k – 1] > A[k – 1] then
            'ivot := A[ k – 1]
           A[k – 1] := A[k]
           A[k] := pivot
       Endif
  endfor
endrepeat
```

Sorting algorithm puts elements of a list in a certain order. Efficient sorting is important to optimizing the use of other algorithms (such as binary search) that require sorted lists to work correctly (Donald, 1997).

ANALYSIS OF AN ALGORITHM
It is important to know how much of a particular resource (such as time or storage) is required for a given algorithm. Some analysis of algorithm areas is as follows:

- An algorithm analysis technique allows us to contrast the efficiency of different solution for a problem.
- Algorithms must efficiently use time and memory.
- Algorithm analysis techniques focus on gross differences in algorithm efficiency.

In analyzing algorithms, there is a need to develop a fundamentals law that are independent of such consideration as programming techniques, compiler, operating system, computer architecture and input data. An algorithm time requirement can be expressed as a function of the problem size. This is called the algorithm's growth rate; the growth-rate of an algorithm tells us how quickly the algorithm grows as a function of the "size of the problems". Algorithms can be classified by the amount of time they need to complete compared to their input size. Some algorithms complete in linear time relative to input size, some do so in an exponential amount of time or even worse, and some even half. Additionally, some problems may have multiple algorithms of differing complexity,

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

while other problem might have no algorithms or no known efficient algorithms. There is also mapping from some problem to other problems. Owing to this, it was found to be more suitable to classify the problems themselves instead of the algorithms into equivalence classes based on the complexity of the best possible algorithm for them (Horowitz and Sahni, 1978).

COMPUTING THE EXECUTION TIME OF LINEAR AND BINARY SEARCH ALGORITHMS

This paper discusses the conduction of experiment on comparative study based on evaluating the execution time of linear search, binary search after bubble sort and binary search with bubble sort. The experiment was started by implementing the algorithm which computes the execution time of the linear search, binary search after bubble sort and binary search with bubble sort. We used the worst case scenario whereby, the element we are searching for is not in the list. The C++ programming language was used to implement the algorithm which is depicted in appendix A (Stroustrup, 1994), and the output results of the execution time is displayed in table 1.

Table1: Execution time of linear search, binary search after bubble sort and binary search with bubble sort.

n	**Linear Search**	**Binary Search With bubble sort**	**Binary Search After bubble sort**
500	0.58 seconds	0.55 seconds	0.72 seconds
600	0.77 seconds	0.91 seconds	0.89 seconds
700	0.92 seconds	0.92 seconds	0.86 seconds
800	1.09 seconds	1.09 seconds	1.09 seconds
900	1.06 seconds	1.04 seconds	1.03 seconds
1000	1.27 seconds	1.40 seconds	1.41 seconds
2000	2.59 seconds	2.61 seconds	2.75 seconds
3000	3.93 seconds	3.92 seconds	3.84 seconds
4000	5.10 seconds	5.27 seconds	5.16 seconds
5000	6.45 seconds	6.69 seconds	6.67 seconds
6000	7..61 seconds	7.99 seconds	7.73 seconds
7000	11.48 seconds	14.41 seconds	10.13 seconds
8000	10.28 seconds	10.61 seconds	10.23 seconds
9000	11.56 seconds	12.09 seconds	11.63 seconds
10000	12.77 seconds	13.34 seconds	12.78 seconds
20000	25.49 seconds	27.81 seconds	25.36 seconds
30000	38.14 seconds	43.22 seconds	37.83 seconds
40000	51.14 seconds	60.11 seconds	50.70 seconds
50000	63.63 seconds	78.47 seconds	63.61 seconds
100000	127.06 seconds	185.38 seconds	125.72 seconds

This search progresses when the execution time values of linear search are used to plot a graph. The graph is plotted by using range of integer numbers against execution time (in seconds). The graph of linear search execution time is displayed in figure . Information in the graph shows that searching for a number in a range of integer numbers, constantly increases with time.

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

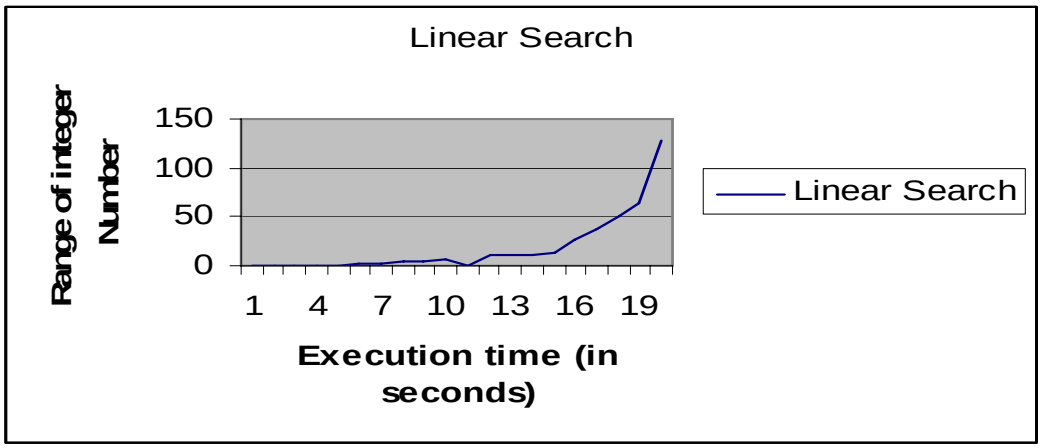

Figure 1: The execution time of linear search

Again another experiment was conducted, this time, the execution time of binary search with bubble sort have been conducted was extracted from table 1. The word "with", means that the execution time of binary search was included with that of the bubble sort. Then, execution time of binary search after bubble sort was also extracted from table 1. The word "after", means that the execution time of binary search was computed excluding the bubble sort time. The execution time values are used to plot two graphs and then displayed in figure 2. We observe that binary search after bubble sort is faster than binary search with bubble sort.

Figure 2: The execution time of Binary search after bubble sort and Binary search with bubble sort.

The research went further to extract the execution time of linear search, binary search with bubble sort and binary search after bubble sort from table 1. Using the same set

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

of integer values, the execution time of these three instances are used to plot three graphs. Some observation are noted. The observations are that for range of integer values between 500 to 40000, it is noticed that only binary search with bubble sort experiences higher execution time but linear search and bubble search after bubble sort have very similar execution time. Further more, a small difference in execution time between linear search and binary search after bubble sort is noticeable in table 1. It is clear that binary search after bubble sort is therefore faster than linear search when the range of integer values passes 1000. The graph of these three instances is displayed in figure 3.

In this paper, it is clear that binary search algorithm is truly not faster than linear search in all instances as claimed by existing literatures (Horowitz and Sahni, 1978; Rodger, 2007). Algorithm authors should have considered the execution time of sorting algorithm when included in binary search execution time.

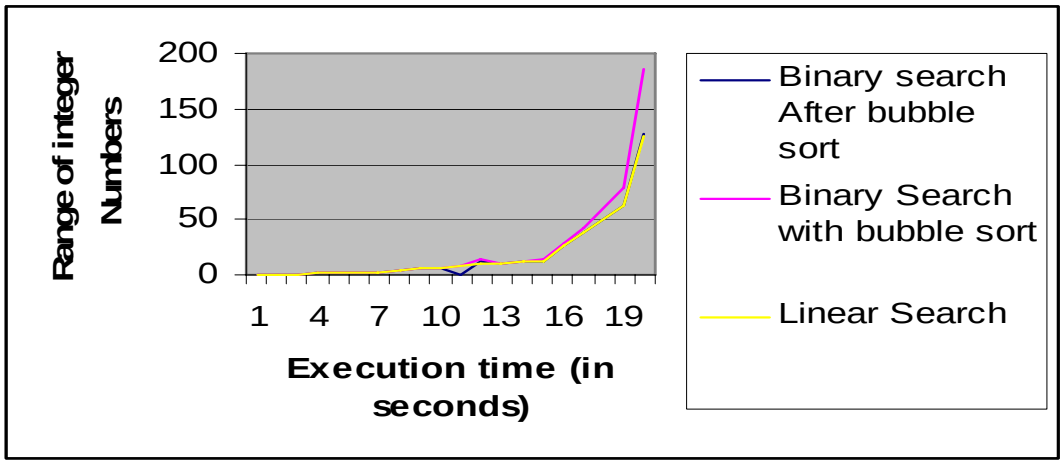

Figure 3: The execution time of linear search, Binary search after bubble sort, and Binary search with bubble sort.

This paper establishes the fact that execution time of binary search algorithm is not faster than execution time of linear search algorithm, when the execution time of the sorting algorithm is included to the binary search algorithm. It is also established in this paper that binary search is faster than linear search after a sorting technique has been performed on it.

IMPLEMENTATION

In this paper, C++ programming language is used to implement the linear and binary search algorithms (Josuttis, 1999). The source code is depicted in appendix A, and the output results generated by running the source codes are displayed in table 1. The specification of the computer system used for the implementation is: Intel Pentium M processor, 1.70 GHZ (speed of processor) and 504 MB of RAM space.

CONCLUSION

This paper compares the execution time of searching for an integer number among a list of integer numbers, using linear and binary search algorithms. Existing literatures have it that binary searc, without considering the

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

sorting time incurred in the binary searching. The emergence of this paper has establishes the fact that sorting time of the numbers before a binary search is conducted, is never considered. If sorting time is added to the binary search time then, execution time of binary search is not faster than linear search technique.

Although, it is very difficult to experiment on algorithms in different order of complexities however, this project work has introduced a standard of how to compare algorithms of different order of complexities. Hidden and unwritten facts about some algorithm can be exposed if researchers carefully examine the components of the algorithm.

REFERENCES

Booch G. (1995), *Object Oriented Analysis and Design,* second Edition, Addison-Wesley.

Collins W. J. (1992), *Data Structures,* first Edition Addison-Wesley publishing company, page 397, U.S.A.

Donald K. (1997), *The Act Computer Programming,* Volume 3, Sorting and Searching, Third Edition, Addison-Wesley.

Horowitz E. and Sahni (1978), *Fundamental of Computer Algorithms,* Computer science press, Inc, Maryland, U.S.A.

Josuttis N. M. (1999), *The C++ Standard Library; A Tutorial and Reference,* Addison-Wesley, Reading, M.A.

Kruse R. L. (1999), *Data Structures and Program design in C++,* Prentice Hall, page 280, USA.

Rodger J. (2007), *Searching Arrays: Algorithms and Efficiency linear versus Binary Search,* Rai Unversity (http:en.wikipedia.org/wiki/binary-searchalgorithm), last accessed on September 20, 2008.

Stroustrup B. (1994), *The Design and Evolution of C++,* Addison-wesley Reading M. A.

APPENDIX A

```
#include <iostream.h>
#include <cstdlib>
#include <time.h>
#define limit 1000000

void outputlist ( int );
int lsearch ( int, int );
void bsort ( int, int );
void searching ( int, int, int );
int binsearch ( int, int, int );
int pivot, list [ limit ];
int number;
clock_t begin, end;

int main ( )
{
 int key;
```

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

```
 int random_integer;
 cout<< "\n\n Program for linear and Binary Search Method \n";
 cout<< "============================\n\n";
 cout<< " How many Elements to be searched: ";
 cin >>number;

//Random numbers from 0 to 99
for (int i = 1; i <= number; i++)
 {
   random_integer = (rand() % 100);
   list[i]=random_integer;
 }

cout<<"\n\n The Elements entered are  \n\n";
outputlist ( number );
cout<<"\n\n Enter the key: ";
cin>> key;
int first = 1;
searching (first, number, key);

int v; cin>>v;
return 0;

}

void searching (int f, int n1, int k1 )
  {
    int v1, v2;
   clock_t begin1, end1, begin2, end2, begin3, end3;

   cout<<"\n\n Elements After linear search are \n\n";
   begin1 = clock ( );
   outputlist (n1);
   v1 = lsearch (n1, k1 );
   end1 = clock ( );

cout <<"\n\n Elements After binary search with bubble sort are \n\n";
begin2 = clock ( );
bsort ( f, n1 );
v2 = binsearch (f, n1, k1 );
outputlist ( number );
end2 = clock ( );

cout <<"\n\n Elements After binary search without bubble sort are \n\n";
```

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

```
begin3 = clock ( );
v2 = binsearch (f, n1, k1 );                    91
outputlist ( number );
end3 = clock ( );

cout<<"\n\n\n\n\n\n\n\n The number of Elements in the array is "<<n1;

cout<<"\n\n\n After linear search \n";
cout<<"================== \n";
if (v1 > n1 )cout<<"\n The element "
  <<k1<< " is not in the list \n";
  if (v1 <= n1 )cout<<"\n The element number  "<<k1
  <<" was found at index "<<v1<<" of the list\n";
cout<<"\n The execution time was: "
      <<(end1 - begin1 ) / ( float ) CLOCKS_PER_SEC
     <<" seconds "<<"\n";

cout<<"\n\n\n After binary search with bubble sort \n";
cout<<"================== \n";
if (v2 > n1 )cout<<"\n The element "
  <<k1<< " is not in the list \n";
  if (v2 <= n1 )cout<<"\n The element number "<<k1
  <<" was found at index "<<v2<<" of the list\n";
cout<<"\n The execution time was: "
      <<(end2 - begin2 ) / ( float )CLOCKS_PER_SEC
     <<" seconds "<<"\n\n";

cout<<"\n\n\n After binary search After bubble sort \n";
cout<<"================== \n";
if (v2 > n1 )cout<<"\n The element "
  <<k1<< " is not in the list \n";
  if (v2 <= n1 )cout<<"\n The element number "<<k1
  <<" was found at index "<<v2<<" of the list\n";
cout<<"\n The execution time was: "
      <<(end3 - begin3 ) / ( float )CLOCKS_PER_SEC
     <<" seconds "<<"\n\n";
}

int lsearch (int right, int k )
{
 int low, mid, found, first, high;
  high = right;
  low = 1;
  while (( low <= high)&&(k != list[low] ))
   {
```

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

```
      low++;
    }
    if (k==list[low] ) return low;
    else return high + 1;
    }
   void outputlist (int n3)
    {
      cout<<" ";
      for ( int i = 1; i <= n3; i++)
      cout<<list [i] << ", ";
     }

 void bsort ( int first, int last )
  {
     int j, pass, temp;
     for ( int i = 1; i < last; i++)
      {
        for (j = i + 1; j <= last; j++)
         {
          if (list [i]  >= list [j] )
             {
              temp = list [i];
               list [i] = list [j];
                list [j] = temp;
                 }
         }
}
}

int binsearch (int left, int right, int k)
 {
   int low, high, mid, num = number;
   high = right;
   low = left;
   bool found = true;
   if (low < high )
    {
      mid = ( low + high ) /2;
      if ( k < list [mid] )
       {
         mid = binsearch (left, mid - 1, k );
       }
      if ( k > list [mid] )
       {
         mid = binsearch ( mid + 1, right, k );
```

Scientia Africana, Vol. 9 (No. 2) December 2010. Pp 83-92
Faculty of Science, University of Port Harcourt, Printed in Nigeria *ISSN 1118-1931*

```
}
if (k != list [mid] ) return num + 1;
}
}
```