Volume 4, Issue 12, December 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

List Search Algorithm using Queue

Shaik. Kareem Basha, L. Harika, M. Ravi Kanth
CSE Dept,. HITAM
Hyderabad, India

Abstract*— Searching is a process of finding whether the given key element belongs to list or not. In the discipline of Computer Science, the problem of search has assumed enormous significance. It spans a variety of applications beginning from searching for a key in a list of data elements to searching for a solution to a problem in its search space. A variety of search algorithms and procedures appropriate to the problems are proposed. We enumerate search algorithms related to the problem of looking for a key K in a list of data elements. When the list of data elements is represented as a linear list, various existing search algorithms such as Linear search or Sequential search, Transpose Sequential search, Interpolation search, binary search and Fibonacci search are applicable. The proposed List Search Algorithm is based upon Divide and Conquer problem solving strategy. In this algorithm, a given input list is divided into left sub list and right sub list based upon the mid element, key element is compared with the mid element and we are using Queue data structure to store the indices of left sub list and right sub list. Two variations of the proposed algorithm are designed by considering Unordered Input list and Ordered Input list. We followed the various stages of Software Development Life Cycle to demonstrate the proposed search algorithm. Section I will give introduction about proposed search algorithm. In section II, proposed Algorithm is Analysed and Designed. In section III, proposed algorithm is Implemented using C programming Language. In section IV, we will test the implementation of proposed algorithm using different test cases. In section V, we conclude the proposed Algorithm.

***Keywords*— queue, left sub list, right sub list, Un Ordered list search, Ordered list search, key element**

I. INTRODUCTION

Searching is a process of finding whether a given key element belongs to input list or not. In the discipline of Computer Science, the problem of search has assumed enormous significance. It spans a variety of applications beginning from searching for a key in a list of data elements to searching for a solution to a problem in its search space. Innumerable problems exist where one searches for patterns-images, voice, text, hyper text, photographs etc in a set of data or patterns, for the solution of the problem concerned. We enumerate search algorithms related to the problem of looking for a key K in a list of data elements. When the list of data elements is represented as a linear list, the search procedures such as Linear search or Sequential search, Transpose Sequential search, Interpolation search, Binary search and Fibonacci search are applicable. When the list of data elements is represented using non linear data structures such as Binary Search Trees or AVL Trees or BTrees etc, the appropriate tree search techniques unique to the data structure representation may be applied. Hash tables also promote efficient searching. Search techniques such as Breadth First Search and Depth First Search are applicable on Graph Data Structures. The Proposed Search Algorithm is based upon Divide and Conquer problem solving strategy, This algorithm takes starting index, ending index of input list, divides the list into two sub lists based on mid element, which is compared with the key element. If both are equal, search is successful otherwise the indices of left sub list and right sub list are placed into Queue Data Structure. The indices of a sub list are removed from queue, mid element of sub list is found, compared it with key element, this process continues until the queue becomes empty. Two variations of List Search Algorithm are proposed based upon the order of Input List. They are UnOrderedListSearch Algorithm, which performs search operation on UnOrdered Input list of elements and OrderedListSearch Algorithm, which perform search operation on Ordered Input list of elements.

II. ANALYSIS AND DESIGN OF PROPOSED ALGORITHM

In Fig 2.1 the given input list X1 X2 X3 ----------Xn is divided into left sub list and right sub list based upon the middle element Xa. Given key element E is compared with Xa, if both are equal then true is returned by the proposed algorithm otherwise, the indices of left sub list (1,a-1) and indices of right sub list (a+1,n) are placed into queue. If queue is not empty then indices of list are considered from queue and the above steps are repeated. If the queue becomes empty then the process is stopped, and the proposed algorithm declares key element E does not belongs to list.

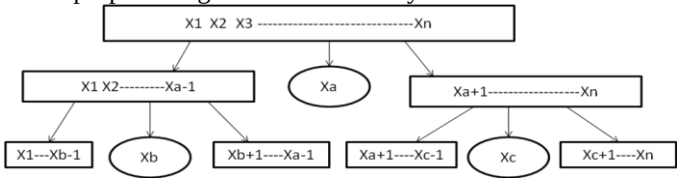

Fig 2.1 Dividing Input list into sub lists

Two variations of proposed algorithm are designed based upon the Ordered or UnOrdered list.
The proposed UnOrderedListSearch Algorithm is as follows:

```
Algorithm UnOrderedListSearch(low,high,key)
//low is the starting index of UnOrdered List
//high is the ending index of UnOrdered List
//key is the element to be search with in the List
{
while(low<=high) // if list is breakable
{
 mid=(low+high)/2; // finding mid element of the list
if(input[mid]==key) // if mid element and key are equal
return true; // element found
rear=rear+1;
queue[rear].low=low; // store starting index of left sub list into queue
queue[rear].high=mid-1; // store ending index of left sub list into queue
rear=rear+1;
queue[rear].low=mid+1; // store starting index of right sub list into queue
queue[rear].high=high; // store ending index of right sub list into queue
if(front<rear) // if queue is not empty
{
 front=front+1;
low=queue[front].low; // Assign starting index of list to low
high=queue[front].high; // Assign ending index of list to high
}
else  break; // if queue becomes empty
}
return false; // element not found
}
```

Algorithm 2.1 UnOrdered List Search using Queue

In Algorithm 2.1, An unsorted input array list is taken as input, it is divided into left sub list and right sub list based upon the mid element. Mid element is compared with key element, if both are equal then true is returned by the proposed algorithm otherwise the indices of left sub list and right sub list are placed into queue. If queue is not empty then taking indices of list and repeating the above steps until the queue becomes empty.

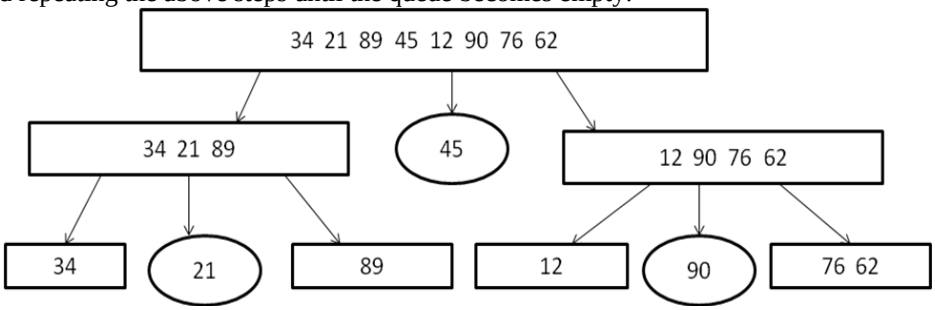

Fig 2.2 UnOrdered Input list is divided into sub lists to search 90

In Fig 2.2 UnOrdered input list consists of elements {34 21 89 45 12 90 76 62} and key element is 90 which is to be search within the list. The given list is divided into 2 sub lists {34 21 89} and {12 90 76 62} based on the mid element 45. Mid element 45 is compared with key element 90, both are not equal so the indices of left sub list {1,3} , indices of right sub list {5,8} are placed into queue. If queue is not empty then indices of list {1,3} are considered and the above process is repeated until 90 is found with in the list.
The time taken by UnOrderedListSearch Algorithm is as follows:

$T(n)=1+T(n/2)+T(n/2)$
$T(n)=1+2T(n/2)$
$T(n)=1+2(1+2T(n/4))$
$T(n)=1+2+2^2T(n/4)$
$T(n)=1+2+2^2(1+2T(n/8))$
$T(n)=1+2+2^2+2^3T(n/8)$
$T(n)=(1+2+2^2)+2^3T(n/2^3)$
$T(n)=(1+2+2^2+\text{-------}+2^{k-1})+2^kT(n/2^k)$
$T(n)=(2^k-1)+2^kT(n/2^k)$
Let $n=2^k$ $k=\log n$
$T(n)=n-1+n.T(1)$
$T(n)=n-1+n=2n-1$

T(n)=O(n)

The proposed OrderedListSearch algorithm is as follows:

```
Algorithm OrderedListSearch(low,high,key)
//low is the starting index of Ordered List
//high is the ending index of Ordered List
//key is the element to be searched with in the list
{
while(1)
{
if(low<=high) // if list is breakable
{
mid=(low+high)/2; // find mid element of the list
if(input[mid]==key) // if key and mid element of list are equal
return true; // element found
else if(key<input[mid]) // key is smaller than mid element{
rear=rear+1;
queue[rear].low=low; // store starting index of left sub list into queue
queue[rear].high=mid-1; // store ending index of left sub list into queue
} else if(key>input[mid]) // key is greater than mid element{
rear=rear+1;
queue[rear].low=mid+1; // store starting index of right sub list into queue
queue[rear].high=high; // store ending index of right sub list into queue
}  }
if(front<rear)  // if queue is not empty{
front=front+1;
low=queue[front].low; //assign starting index of list to low
high=queue[front].high; // assign ending index of list to high
}
else  break; // if queue become empty
}
return false; // element not found
}
```

Algorithm 2.2 Ordered List Search Algorithm

In algorithm 2.2 the list is divided into two sub lists based on the mid element m. Given key element k is compared with mid element m, if both are equal then the proposed algorithm returns true otherwise if the key element k is smaller than mid element k, then the indices of left sub list are placed into queue, otherwise the indices of right sub list are placed into queue. If the queue is not empty then indices of list are taken from queue and the above process is repeated until element found within the list or queue becomes empty.

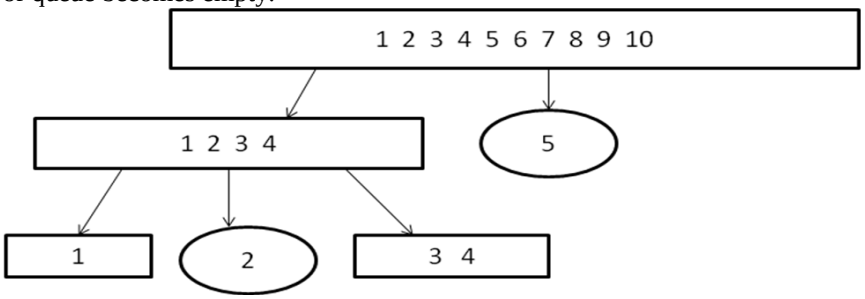

Fig 2.3 Division of Input list into sub lists to search 2

In Fig 2.3 the given input list is an ordered list, which consists of elements {1 2 3 4 5 6 7 8 9 10} and key element is 2, mid element 5 is compared with key element 2, both are not equal but 2 is smaller than 5, so indices of left sub list (1,4) are placed into queue. If queue is not empty, then indices of list are taken from queue and the process is repeated until key element 2 is found within the list.

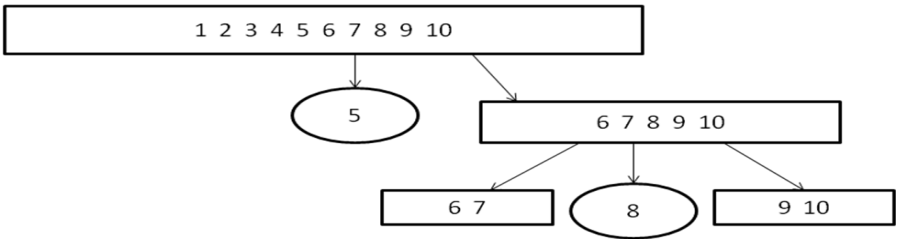

Fig 2.4 Division of Input list into sub lists to search 8

In Fig 2.4 the given input list is an ordered list, which consists of elements {1 2 3 4 5 6 7 8 9 10} and key element is 8, mid element 5 is compared with key element 8, both are not equal but 8 is greater than 5, so indices of right sub list (6,10) are placed into queue. If queue is not empty, then indices of list are taken from queue and the process is repeated until key element 8 is found within the list.

The time taken by OrderedListSearch Algorithm is as follows:

$T(n)=1+T(n/2)$
$T(n)=1+1+T(n/4)$
$T(n)=1+1+1+T(n/8)$
$T(n)=3+T(n/8)$
$T(n)=k+T(n/2^k)$
Let $n=2^k$ $k=\log n$
$T(n)=\log n+T(1)$
$T(n)=O(\log n)$

III. IMPLEMENTATION OF PROPOSED ALGORITHM USING C

The Proposed Algorithm is implemented by using C Programming Language, which is a robust, structure oriented programming language, which provides different concepts like functions, pointers, structures, arrays and so on to solve complex problems.

```
/* Implementation of UnOrdered List Search Algorithm using queue*/
# include <stdio.h>
# include <conio.h>
struct queue { //queue structure to store indices of left sub list and right sub list
  int low;
  int high;
 };
 struct queue q[40]; // instance of queue structure
int front=0,rear=0; // indices to operate queue
int list[100],key;
void UnOrderedListSearch(int low,int high)  {
 int mid;
 while(low<=high)  // if list is breakable{
    mid=(low+high)/2; // find mid element of the list
    if(list[mid]==key) // if key and mid element are equal {
    printf("\nElement %d found",key);
    return;   }
    rear++;
    q[rear].low=low; //store starting index of left sub list into queue
    q[rear].high=mid-1; //store ending index of left sub list into queue
    rear++;
    q[rear].low=mid+1; // store starting index of right sub list into queue
    q[rear].high=high; // store ending index of right sub list into queue
    if(front<rear)  // if queue is not empty {
     front++;
     low=q[front].low; //assign starting index of list to low
     high=q[front].high;  //assign ending index of list to high
    }
    else   break; // if queue becomes empty
 }
 printf("\nElement %d not found",key);
}
 void main()  {
 int i,n;
 clrscr();
 printf("\nImplementation of Searching Technique");
 printf("\nList contains how many elements:\n");
 scanf("%d",&n);
 printf("\nEnter %d elements:\n",n);
 for(i=1;i<=n;i++)
 scanf("%d",&list[i]);
 printf("\nEnter search element:\n");
 scanf("%d",&key);
```



```
 UnOrderedListSearch(1,n);
 getch();
 }
```

Program 3.1 Implementation of UnOrdered List Search Algorithm

```
/* Implementation of Ordered List Search Algorithm using queue*/
# include <stdio.h>
# include <conio.h>
struct queue  { // queue structure to store indices of left sub list and right sub list
  int low;
  int high;
 };
 struct queue q[20]; // instance of queue structure
 int rear=0,front=0;
 int input[50];
 void OrderedListSearch(int low,int high,int key) {
   int mid;
   while(1) {
    if(low<=high)  // if list is breakable{
    mid=(low+high)/2; // find mid element of list
    if(input[mid]==key) // if key and mid element are equal {
    printf("\nElement found");
    return;
    } else if(key<input[mid]) // if key is smaller than mid element consider left sub list {
    rear++;
    q[rear].low=low; // place starting index of left sub list into queue
    q[rear].high=mid-1; //place ending index of left sub list into queue
    } else   // if key is greater than mid element consider right sub list{
    rear++;
    q[rear].low=mid+1; // place starting index of right sub list into queue
    q[rear].high=high; // place ending index of right sub list into queue
    }
    }
    if(front<rear)  // if queue is not empty  {
    front++;
    low=q[front].low; // assign starting index of list to low
    high=q[front].high; // assign ending index of list to high
    }
    else break // if queue becomes empty;
   }
   printf("\nElement not found");
   }
    void main() {
    int i,n,key;
    clrscr();
    printf("\nOrdered List contains how many elements:\n");
    scanf("%d",&n);
    printf("\nEnter %d elements of ordered list:\n",n);
    for(i=1;i<=n;i++)
    scanf("%d",&input[i]);
    printf("\nEnter search key element:\n");
    scanf("%d",&key);
    OrderedListSearch(1,n,key);
    getch();
    }
```

Program 3.2 Ordered List Search Algorithm

IV. TESTING OF PROPOSED ALGORITHM

Different Test cases are considered to test the result of Program 3.1 and Program 3.2. The following are the screen shot, which shows the result of variations of proposed algorithm based upon the ordered or unordered input list. Consider the following test case of unordered list which consists of 10 elements {12 1 34 2 3 78 4 100 87 25}, and key element 27 is to be searched with in the list is taken as input to test the working of program 3.1 and the result is as shown in Fig 4.1.

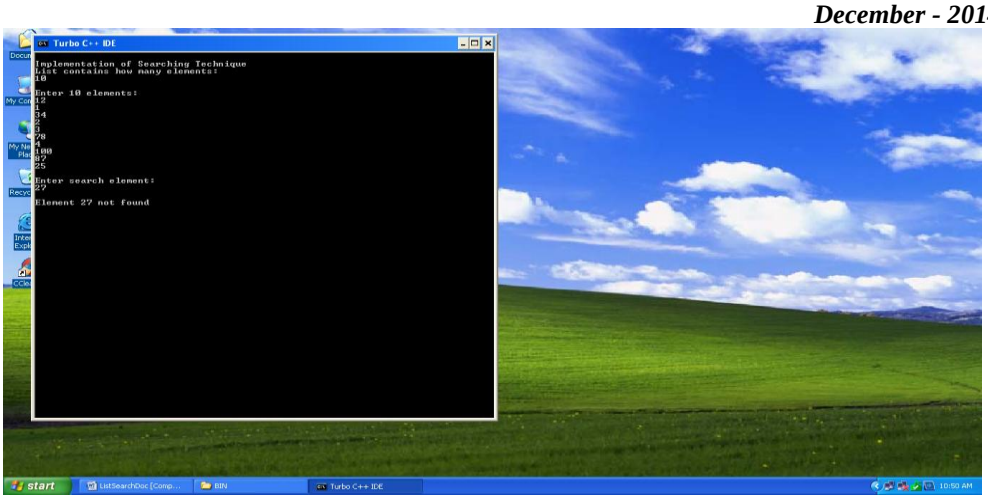

Fig 4.1 screen shot of UnOrdered List Search

Consider the following test case of unordered list which consists of 10 elements {1 5 6 7 10 12 16 18 22 25}, and key element 27 is to be searched with in the list is taken as input to test the working of program 3.1 and the result is as shown in Fig 4.2.

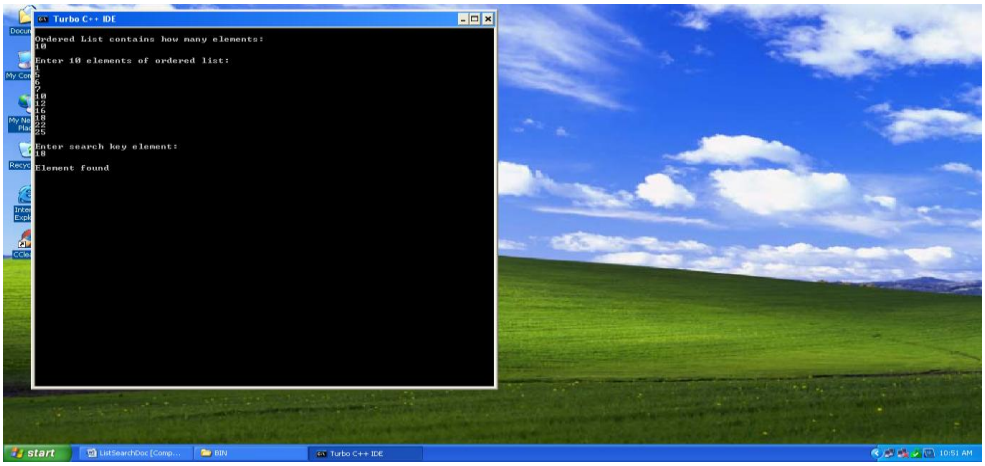

Fig 4.2 screen shot of Ordered List Search

V. CONCLUSION

We conclude that in the proposed Algorithm, if the list is UnOrdered list, then we are dividing it into left sub list and right sub list based upon mid element. Mid element is compared with key element if both are equal then it returns true otherwise the indices of left sub list and right sub list are placed into queue. If queue is not empty, then indices of a list are taken from queue and the above process is repeated until queue becomes empty or key element found. If the list is Ordered list, then mid element is compared with key element. If both are equal then proposed algorithm returns true, otherwise if key element is smaller than mid element then indices of left sub list are placed into queue otherwise indices of right sub list are placed into queue. If queue is not empty, then indices of list are taken from queue and the above process is repeated until queue becomes empty or key element found.

REFERENCES

[1] Aho, Alfred V. and Jeffrey D. Ullman [1983]. Data Structures and Algorithms. AddisonWesley, Reading, Massachusetts.

[2] Cormen, Thomas H., Charles E. Leiserson and Ronald L. Rivest [1990]. Introduction toAlgorithms. McGraw-Hill, New York.

[3] Knuth, Donald. E. [1998]. The Art of Computer Programming, Volume 3, Sorting and Searching. Addison-Wesley, Reading, Massachusetts.

