UNIT – 4

Searching Techniques

Poonam Dhand(Assistant Prof., Computer Dept., GCG Ludhiana)

Paramjit Kaur(Assistant Prof., Computer Dept., GCG Ludhiana)

4.1 SEARCHING – AN INTRODUCTION

Searching is the process of locating given value position in a list of values, i.e. search is a process of finding a value in a list of values. Search is said to be successful if the element being searched is found or unsuccessful when the element being searched is not found. A search typically answers in either **True** or **False** as to whether the item is present. On occasion it may be modified to return where the item is found. Two techniques are used for searching:

1) **Linear/ Sequential Search.**
2) **Binary Search.**

4.2 LINEAR SEARCH

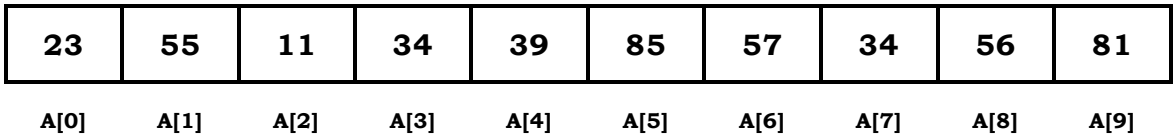
A linear search is the basic and simple search algorithm. In Linear search, a sequential search is made over all elements one by one. In other words linear search searches an element or value from an array till the desired element is not found in sequential order. Every element is checked and if it is found then that particular item is returned, otherwise the search continues till the end of the data collection. Searching begins from first element and continues until the desired element is found or end of the file is reached. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.

Basic Idea : working of Linear Search

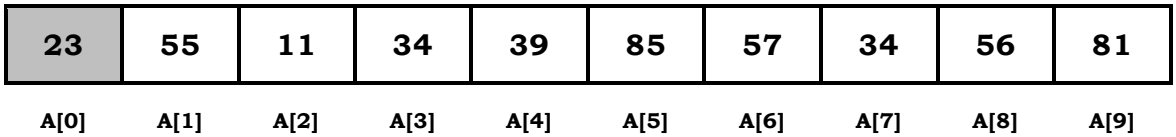
- Read the search element which is to be searched
- Initially Compare, the search element with the first element in the list. If both are matching, then display "element found" and terminate the function
- If both are not matching, then compare search element with the next element in the list.
- Repeat steps until the search element is compared with the last element in the list.
- If the last element in the list is also doesn't match, then display "Element not found" and terminate the function.

Example

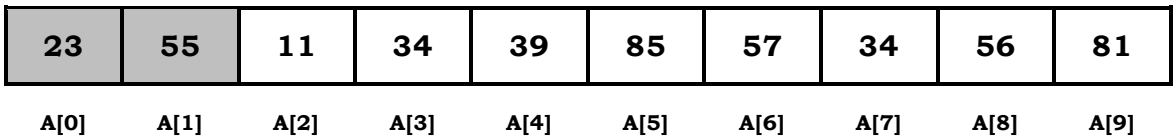
Suppose we are given a array of 10 elements. We are to search element 72. i.e. **X = 85.**



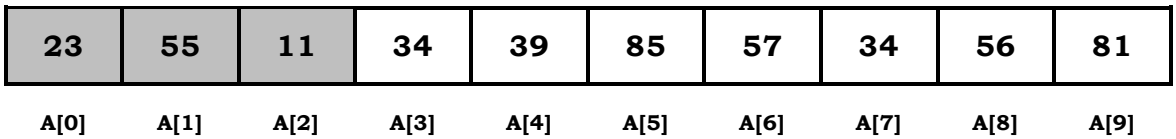
Step 1: It compares the x with first element located at index 0.Element is not matched then it moves to the next element.



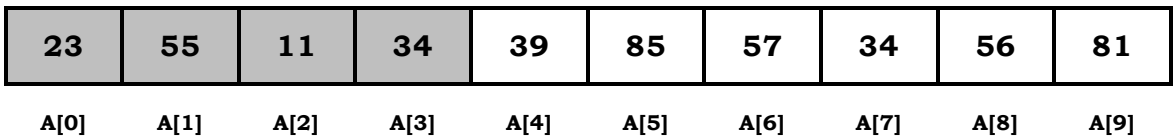
Step 2: It compares the x with next element located at index 1. Element is not matched then it moves to the next element.



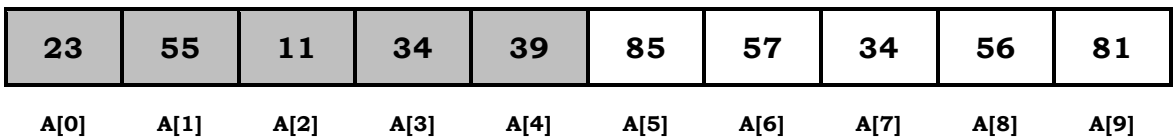
Step 3: It compares the x with next element located at index 2. Element is not matched then it moves to the next element.



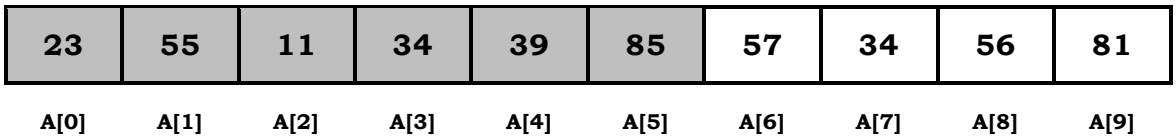
Step 4: It compares the x with next element located at index 3. Element is not matched then it moves to the next element.



Step 5: It compares the x with next element located at index 4. Element is not matched then it moves to the next element.



Step 6: It compares the x with next element located at index 5. The desired element 85 is found at 5 location. Here the process is stopped and terminate the function.



Note: If the last element in the list is also doesn't match, then display "Element not found" and terminate the function.

Algorithm

```
Linear Search ( A, x)
// Given an array A[i :x] and x represent number of elements
Set found to false
Set position to -1
Set index to 0
While found is false and index < x
  If A[index] is equal to search value
    found = true
          position = index
  End if
  Add 1 to index
End while
Return position
```

Pseudocode

```
procedure linear_search (list, value,x)

  int index = 0;
        //Position of search value
        int position = -1;
        //Indicate whether the value is found
        bool found = false;

        while (index < x && !found)
        {
                if (list[index] == value)
                {
                        //The value is found

                        found = true;
                        position = index;
                }

                index++;
        }
```

```
            return position;
}


end procedure
```

Program:

```
/*searching an element using linear search technique*/
#include<stdio.h>
#include<conio.h>
void main(){
  int list[20],size,i,sElement;

  printf("Enter size of the list: ");
  scanf("%d",&size);

  printf("Enter any %d integer values: ",size);
  for(i = 0; i < size; i++)
    scanf("%d",&list[i]);

  printf("Enter the element to be Search: ");
  scanf("%d",&sElement);

  // Linear Search Logic
  for(i = 0; i < size; i++)
  {
     if(sElement == list[i])
     {
        printf("Element is found at %d index", i);
        break;
     }
  }
  if(i == size)
     printf("Given element is not found in the list!!!");
  getch();
}
```

Output:

```
Enter the size of the list: 6
16    2   37   40   55   29
Enter value to be search:40
Element found at index 4
```

Complexity analysis:
The time complexity of linear search algorithm in best case is **O(1)** and in average and worst case is **O(n) .**

Advantages:

- The linear search is simple - It is very easy to understand and implement.
- From an operational standpoint, linear search also is very resource efficient - it does not require copying/partitioning of the array being search, and thus is memory-efficient. It also operates equally well on both unsorted and sorted data.
- It does not require the data in the array to be stored in any particular order.

Disadvantages:

- The primary disadvantage of linear search is that it has a very poor O(n) general efficiency. That is, the performance of the algorithm scales linearly with the size of the input.
- It is slow and time consuming

4.3 BINARY SEARCH

Binary search is the most popular Search algorithm based on Divide and conquer technique. It works on a sorted array list either in ascending or descending order. In this method firstly find the middle element and to search an element compare the value with the elements in the middle position of the array. If the value is matched, then search is successful otherwise list is divided into two halves(one from the first element to the middle element i.e. first half or lower half and second half or upper half from middle to last element . If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array.

Basic idea: Working of Binary search

Let **X** be the element to be searched in an array **A[0 – (n-1)] .**

1. Select the element which have to be searched
2. Find the middle value of an array using formula:
   mid= (low + high)/2.
   Here low refers to lower bound of an array i.e. **0** and high refers to upper bound of an array i.e. **n-1**.
3. If the element to be searched is the middle element then the search terminates otherwise.
4. If the element to be searched is less than the middle element then the search is performed in left sub array and **high** is set to **mid-1**.

5. Otherwise, if the element to be searched is greater than the middle element then the search is performed in right sub array and **low** is set to **mid+1**. In this way the problem size is reduced to half after every comparison.
6. Repeat the same process until we find the search element in the list or until sublist contains only one element.
7. If that element also doesn't match with the search element, then display "Element not found in the list!" and terminate the function.

Example:
Suppose we are given a sorted array of 10 elements. We are to search element 72. i.e. **X = 72.**

11	18	23	34	39	46	57	69	72	81
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Low=0, high=9 mid=0 + 9/2 =4

11	18	23	34	39	46	57	69	72	81
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

As X >Mid. So low becomes mid+1 and high remain same.
Low=5, high=9 mid=(5 + 9)/2 i.e. 7.

11	18	23	34	39	46	57	69	72	81
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

As X >Mid. So low becomes mid+1 and high remain same.
Low=8, high=9 mid=(8 + 9)/2 i.e. 8.

11	18	23	34	39	46	57	69	72	81
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Now X=Mid. The element is found and the search terminated in 3 steps.

Algorithm

BinSearch(a, i, n, x)
// Given an array a[i :n] of elements in nondecreasing order, 1 ≤ i ≤ J, determine
// whether x is present, and if so, return j such that x — a[j]; else return 0.

```
        {
                if (n = i) then // If Small(P)
                {
                        if (x = a[i]) then return n;
                        else return 0;
                }
        else
        { // Reduce P into a smaller subproblem.
                mid := [(i + n/2)];
                if (x = a[mid]) then return mid;
                else if (x < a[mid]) then
                        return BinSearch(a, i, mid - 1, x );
                else return BinSearch(a, mid +1, n, x);
        }
}
```

Explanation: Above algorithm has four inputs an array **a[ ],** first element **i**, last element **n** and the element to be searched **x**. It recursively calculate the middle element and check the presence of **x**.

Pseudocode

```
Procedure binary_search
  A ← sorted array
  n ← size of array
  x ← value to be searched

  Set lowerBound = 1
  Set upperBound = n

  while x not found
    if upperBound < lowerBound
      EXIT: x does not exists.

    set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

    if A[midPoint] < x
      set lowerBound = midPoint + 1

    if A[midPoint] > x


     set upperBound = midPoint - 1

    if A[midPoint] = x
      EXIT: x found at location midPoint

  end while
```

```
end procedure
```

Program:

```
/* implementation of the binary search*/
#include<stdio.h>
#include<conio.h>
void main()
{
   int first, last, middle, size, i, sElement, list[100];
   clrscr();

   printf("Enter the size of the list: ");
   scanf("%d",&size);

   printf("Enter %d integer values in Assending order\n", size);

   for (i = 0; i < size; i++)
      scanf("%d",&list[i]);

   printf("Enter value to be search: ");
   scanf("%d", &sElement);

   first = 0;
   last = size - 1;
   middle = (first+last)/2;

   while (first <= last) {
      if (list[middle] < sElement)
         first = middle + 1;
      else if (list[middle] == sElement) {
         printf("Element found at index %d.\n",middle);
         break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
   }
   if (first > last)
      printf("Element Not found in the list.");
   getch();
}
```

Output:

```
Enter the size of the list: 6
Enter 6 integer values in Assending order:
10    20   30   40   50   60
Enter value to be search:40
Element found at index 4
```

Complexity analysis:
The time complexity of binary search algorithm in best case is **O(1)** and in average and worst case is **O(log(n)).**

Advantages

- **The binary search** is much more efficient than **the linear search**. Every time it makes a comparison and fails to find the desired item, it eliminates half of the remaining portion of the array that must be searched.
- Binary search can have random access to the data but linear search only requires sequential access.

Disadvantage

- Binary search process only on sorted list.

QUESTIONS

➢ MULTIPLE CHOICE QUESTIONS

1. **Which of the following is not the required condition for binary search algorithm?**
   a) The list must be sorted
   b) There should be the direct access to the middle element in any sub list
   c) There must be mechanism to delete and/or insert elements in list.
   d) Number values should only be present
2. **The Average case occurs in linear search algorithm.**
   a) when item is somewhere in the middle of the array
   b) when item is not the array at all
   c) when item is the last element in the array
   d) Item is the last element in the array or item is not there at all
3. **Binary search algorithm cannot be applied to.**
   a) Sorted linked list
   b) Sorted Binary tree
   c) Sorted Linear Array
   d) Pointer Array
4. **Complexity of linear search algorithm is:**
   a) O(n)
   b) O(log n)
   c) O(n2)
   d) O(n log n)

5. Finding the location of a given item in a collection of items is called.

a) Discovering
b) Finding
c) Searching
d) Mining

6. What is the worst case time complexity of linear search algorithm?

a) O(1) b) O(n)
c) O(log n) d) $O(n^2)$

Answers (MCQs)

1. (c) 2. (a) 3. (a) 4. (a) 5. (c) 6. (c)

➢ FILL IN THE BLANKS

1. To compute the maximum _____ comparisons are necessary and sufficient.
2. In Linear search, a _____is made over all elements one by one.
3. The worst case time complexity of linear search algorithm is_____.
4. Binary search process only on _____.
5. The time complexity of binary search algorithm in best case is _____.

Answers (Fill in the blanks)

1. **n – 1.**
2. **Sequential search**
3. **O(log n)**
4. **Sorted list**
5. **O(n)**

➢ TRUE/FALSE

1. Finding the location of a given item in a collection of items is called searching.
2. The binary search is much more efficient than the linear search.
3. Linear search algorithm is used for large array list elements .
4. Complexity of binary search algorithm in worst and average case is O(n).
5. Comparisons are necessary and sufficient for computing both the minimum and the maximum is 3n-3/2.

Answers (True/False)

1. (T) 2. (T) 3. (F) 4. (F) 5. (T)

➢ SHORT ANSWER QUESTIONS

Q1. Define Searching?

Ans: Searching is the process of locating given value position in a list of values, i.e. search is a process of finding a value in a list of values. Search is said to be successful

if the element being searched is found or unsuccessful when the element being searched is not found. A search typically answers in either **True** or **False** as to whether the item is present. On occasion it may be modified to return where the item is found. Two techniques are used for searching:

1) **Linear/ Sequential Search.**
2) **Binary Search.**

Q2. Explain advantages and disadvantages of binary search.

Ans: Advantages

- **The binary search** is much more efficient than **the linear search**. Every time it makes a comparison and fails to find the desired item, it eliminates half of the remaining portion of the array that must be searched.
- Binary search can have random access to the data but linear search only requires sequential access.

Disadvantage

- Binary search process only on sorted list.

Q3. Write the complexity of Linear search and Binary search.

Ans: **Complexity analysis:**

The time complexity of linear search algorithm in best case is **O(1)** and in average and worst case is **O(n) .**

The time complexity of binary search algorithm in best case is **O(1)** and in average and worst case is **O(log(n)) .**

➢ LONG ANSWER QUESTIONS

Q1. Explain Linear search with example.

Ans: Refer Section **4.2.**

Q3. Explain Binary search with example.

Ans: Refer Section **4.3.**

EXERCISE

Q1. Search the element given array(24,13,56,35,46,27,78,48) using binary search.

Q2. Define searching. Explain different types of searching.

□□□□□