Searching

Copyright © 2021, ABES Engineering College
Downloaded from pickywallpapers.com

Searching :

Searching is the process of finding a given value position in a list of values.

It decides whether a search key is present in the data or not.

It is the algorithmic process of finding a particular item in a collection of items.

There are several methods of searching in the list of items.
Some of them are listed below:

- Linear/ Sequential Search (Brute Force approach)
- Binary Search (Decrease and conquer approach)
- Ternary Search
- Jump Search
- Exponential Search
- Indexed Sequential Search (Combination of Sequential and Binary Search)
- Hashing

Copyright © 2021, ABES Engineering College

Real world Problems

There are many other scenarios in which searching is performed as a frequently carried out operation.

Some of them are listed below:

- Dictionary where meaning is required to be searched for a given the word,
- Catalog from which description is required to be searched for a particular item e.g. e-retail stores like Amazon, Flipkart
- Daily attendance through fingerprint scan (biometric search),
- Telephone directory of mobile phone in which a telephone number is required to be searched through name,
- Keyword search in Google, etc.
- Research article search at Research Gate, Sci-hub, etc.

Copyright © 2021, ABES Engineering College

Linear Search

A Linear/Sequential search simply scans each element at a time sequentially; that's why it is also known as sequential search.

Example:

Suppose we have to find a mobile number for some person. The Mobile No is stored in the address book of the phone. If we scan from the first contact and scroll it down one by one until the desired mobile no is found, this process is also a sequential search.

Copyright © 2021, ABES Engineering College

Algorithm

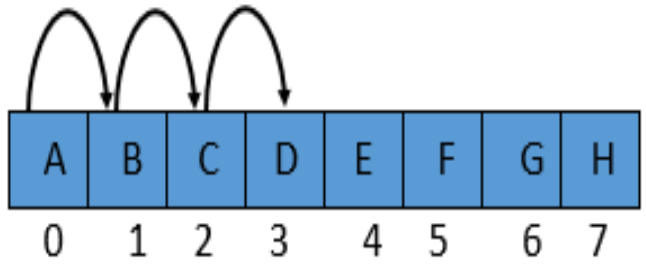
ALGORITHMLinearSearch(A[ ], N, SearchKey)
BEGIN:

```
        FOR i=0 to N-1 DO
                IF A [ i ] = =SearchKey THEN
                RETURN i
        RETURN -1
        //invalid index indicating search element is not found
```

END;

Example

Let us consider the following example. The below-given figure shows an array of character values having 8 data items.



If we want to search 'D', then the searching begins from the 0th element and scans each element is scanned one by one till the search element is not found.

Copyright © 2021, ABES Engineering College

Analysis of algorithm

Time Complexity

Best-case complexity: Ω(1), when the element is found at the first position.

Worst-case complexity: O(n), when the element is found at the last index or element is not present in the array.

Average-case Complexity: For the average case analysis, we need to consider the probability of finding the element at every position. In a set of randomly arranged data elements, finding the search element at any place is equally likely. In an element data set of n size, the probability of finding the element at every position will be 1/n.

Total Effort = ∑ Probability * No of Comparisons

= 1/n * 1 + 1/n *2 + 1/n*3+ … + 1/n*(n-1) + 1/n*(n)

=1/n*(1+2+3+ … + n)

= 1/n * ∑ n

= 1/n * n*(n+1)/2

= (n+1)/2

= θ(n)

Copyright © 2021, ABES Engineering College

Analysis of Algorithm

Space Complexity: In the algorithm written above, we just need a loop counter as additional memory. The space complexity thus is constant i.e., **θ(1).**

Copyright © 2021, ABES Engineering College

Linear Search in 2-D Array(Matrices)

downloaded from pickywallpapers.com

ALGORITHM LinearSearchIn2D(A[m][n], SearchKey)

BEGIN:

```
        FOR i=0 to m-1 DO
                FOR j=0 to n-1 DO
IF A [i][j]  = = SearchKey THEN
                                        WRITE(“Element found at Row i, Column j”)
        WRITE(“Element not present”)
```

END;

Time complexity = $O(n^2)$

Copyright © 2021, ABES Engineering College

Practice Problem

Examples:

Given number is 225: Then both the above algorithms return true because 25 is present in 225 and 225 is divisible by 25.

Given number is 175: 25 is not present so return false. 175 is divisible by 25 so returns true.

Given number is 149: Both the Algorithm returns false as given number does not contain 25 and also not divisible by 25

ALGORITHM Print(A[],N)

BEGIN:

```
        FOR i=0 TO N DO
                IF Contains25(A[i]) || DivisibleBy25(A[i])
                        WRITE(“LIKE”)
                ELSE
                        WRITE(“DISLIKE”)
```

END;

Copyright © 2021, ABES Engineering College

Practice Problem

ALGORITHM Contains25(N)
BEGIN:

```
        WHILE N!=0 DO
                IF N%100==25 THEN
                        RETURN 1
                N=N/10
        RETURN 0
```

END;

ALGORITHM DivisibleBy25(N)
BEGIN:

```
        RETURN N%25==0
```

END;

Copyright © 2021, ABES Engineering College

Disadvantage

- In the linear search the data elements are randomly arranged.
- In case we have the data elements arranged in some order, the search effort can be brought down?
- Linear Search still takes o(n) time
- Can we Reduce ?

Copyright © 2021, ABES Engineering College

Binary Search

Binary search is an approach that can be applied ..
If the Binary search is performed on a sorted array, the procedure goes like:

1. Find the middle element of the array.
2. Compare the mid element with the search element.
3. There are three possible cases.

The mid element is the same as the search element. Search can be declared successful in this case.

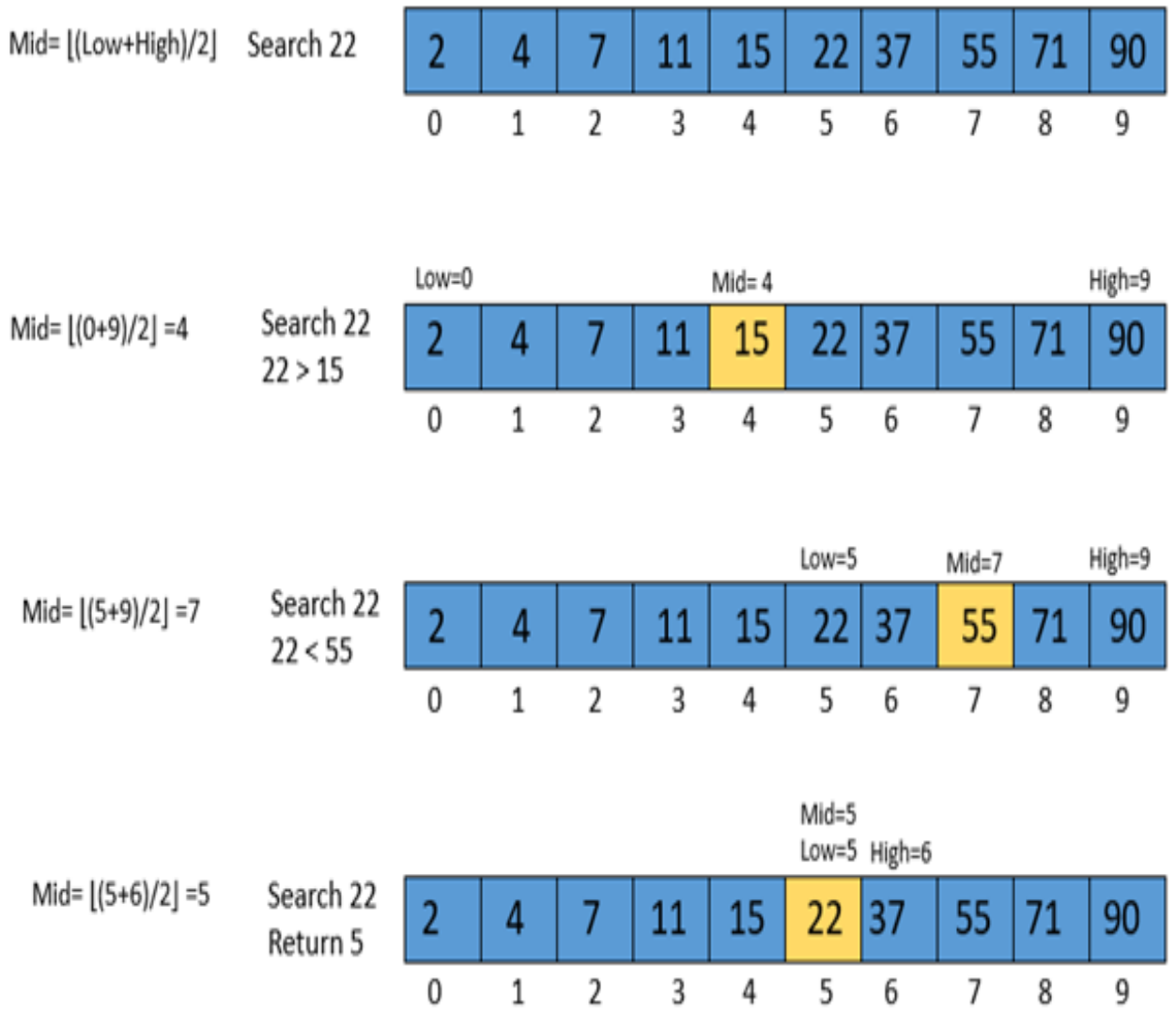
1. If the search element is less than the mid element, the search area should be restricted to the left half of mid only
2. Similarly, if it is greater than the mid element, search area should be the right half of mid.
3. The process is followed until the search element is found.

Copyright © 2021, ABES Engineering College

Solution Approach

Example:

- Let us understand the working of binary search through an example:
- Suppose we have an array of 10 size, which is indexed from 0 to 9 as shown in the below figure and we want to search element 22 (Key) in the given array.



Copyright © 2021, ABES Engineering College

Algorithm

ALGORITHM Binary Search (A[],N, SearchKey)
BEGIN:
Low=0
High = N-1
WHILE Low <=High DO
Mid = [(Low +High)/2]
IFA[Mid] = = Searchkey THEN
        RETURN Mid
ELSE
        If SearchKey<A[Mid] THEN
                High= Mid-1
        ELSE
                Low= Mid+1
RETURN -1   // invalid index indicating that element is not found
END;

Copyright © 2021, ABES Engineering College

Analysis of Algorithm

Time complexity:

In binary search, best-case complexity is Ω(1), where the element is found at the middle index in the first run.

For the worst-case analysis, let us have a second look at the algorithm. Constant number of statements is required to be executed for dividing the search area. The number of elements in either of the selected half will be N/2. A binary search is performed on the selected area recursively. The Recurrence given below justifies this paragraph.

T(N) = T(N/2) +C

T(N) is the time complexity of Binary search in an array of size N. T(N/2) is the Time complexity of Binary Search on an array of size N/2.

T(N) = T(N/2) +C

= [T(N/ 4) + C] +C

= T(N/22) + 2C

= [T(N/8) +C]+ 2C

= T(N/23) + 3C

…

= T(N/2K) + K.C

Copyright © 2021, ABES Engineering College

Analysis of Algorithm

After K divisions, the length of array becomes 1

Length of array = N⁄2K = 1

=>N = 2K

Applying log function on both sides

=>log2 (N) = log2 (2K)

=>log2 (N) = K log2 (2)

=>K = log2 (N)

T(N) =T(N/2) +K.C

=T (1) + log2 (N).C

Searching in a one size array requires constant computations

= C' + log2 (N).C

= O(log2 N)

Space Complexity: The algorithm takes high, low, and mid variables in the logic. Count of 3 is constant; hence the space complexity of Binary Search is θ(1).

Copyright © 2021, ABES Engineering College

Recursive Binary Search

The recursive approach of Binary search is similar to the iterative one. It assumes that every time a part of the array is selected for search, We can perform a Binary search on that array recursively.

ALGORITHM BinarySearch (A[ ], Low, High SearchKey)

- **ALGORITHM BinarySearch (A[ ], Low, High SearchKey)**
- BEGIN:
- IF Low<= High THEN
- Mid = ⌊(Low +High)/2⌋
- IF A[Mid] = = SearchKey THEN
- RETURN Mid
- ELSE
- IF SearchKey<A[Mid] THEN
- BinarySearch(A[ ], low, Mid–1, SearchKey)
- ELSE
- BinarySearch(A[ ], Mid+1, High, SearchKey)
- RETURN –1
- END;

The recursive Binary Search approach is easy to write, but it increases the space complexity because of pending activation records. The space taken by recursive Binary Search is $O(\log_2 N)$

Copyright © 2021, ABES Engineering College

Comparison of Linear Search with Binary Search

	Linear Search	Binary Search
Working	Linear search iterates through all the elements and compares them with the key which has to be searched.	Binary search wisely decreases the size of the array which has to be searched and compares the key with the mid element every time.
Prerequisites	Data can be random or sorted the algorithm remains the same, so there is no need for any pre-work.	It works only on a sorted array, so sorting an array is a prerequisite for this algorithm.
Use Case	We are generally preferred for smaller and randomly ordered datasets.	We are preferred for comparatively larger and sorted datasets.
Effectiveness	Less efficient in the case of larger datasets.	More efficient in the case of larger datasets.
Time Complexity	Best-case complexity - Ω(1) Worst-case complexity - O(n)	Best-case complexity - Ω(1) Worst-case complexity- O(log2n)

Copyright © 2021, ABES Engineering College

Ternary Search

Ternary Search

Ternary Search is nothing, but it is a variation of Binary Search. The Binary Search divides the search half into two, and the Ternary divides into 3. In Ternary Search, if x=3, then divide the search intervals into 3 approximately equal subparts. There will be two middle values: middle1 and middle2.

Let us suppose that the element to be searched is Searchkey,

1- Compare the Searchkey with middle1 and middle2

2- If Searchkey is found, then return else decide in which search interval item belongs in next step.

Part1 | Part2 | Part3

Let us assume that array size is n, beg=0 and end=n-1.

Copyright © 2021, ABES Engineering College

Algorithm

ALGORITHM TernarySearch(A[n], Beg, End, SearchKey)
BEGIN:

```
IF End <Beg THEN
        RETURN -1
Middle1=Beg+(End – Beg)/3
Middle2= Middle1+(End – Beg)/3
IF A[Middle1]== SearchKey THEN
        RETURN Middle1
IF A[Middle2]== SearchKey THEN
        RETURN Middle2
IF A[Middle1]> SearchKey THEN
        RETURN TernarySearch(A, Beg, Middle1 – 1, SearchKey)
IF A[middle2]>SearchKey THEN
        RETURN TernarySearch(A, Middle1+1, Middle2 – 1, SearchKey)
RETURN TernarySearch(A, Middle2+1, End, SearchKey)
```

END;

Copyright © 2021, ABES Engineering College

Analysis of Algorithm

Time Complexity

The time complexity of ternary search = $\log_3 n$ because total function call is $\log_3 n$, but in binary search complexity is $\log_2 n$ i.e. number of function call is more than ternary search. It seems the ternary search is better but ternary search needs more comparison than binary search.

Number of comparisons in binary search = $2\log_2 n+2$

Number of comparisons in binary search = $4\log_3 n+4$

From above it is clear that

$2\log_2 n+2 < 4\log_3 n+4$

It means binary search takes less comparison than ternary search so overall complexity is not reduced significantly.

Copyright © 2021, ABES Engineering College

Index Sequential Search

Index Sequential Search

- It is a searching technique that uses sequential searching and random-access searching methods. In this searching, a given sorted array of n elements is divided into groups based on the group size. Then we create an index array that contains the starting index of each group. This index array also stores the indexes of each group in increasing order.
- If we want to search an element called a key in the given array, we find the index group to present that search element.
- Following are the steps to implement Index Sequential Search

1. Read the search element from the user
2. Divide the array into groups according to group size.
3. Create the index array that contains starting index of groups.
4. If group is present and the first element of the group is less than or equal to key element
   goto next group
   Else
   Apply linear search in previous group
5. Repeat step 4 for all groups.

Copyright © 2021, ABES Engineering College

Application

Application

This index sequential search or access to search or access records in the Database. It accesses database records very quickly if the index table is organized correctly. The main advantage of the indexed sequential is that it reduces the search time for a given item because sequential search is performed on the smaller range compared to the large table.

Copyright © 2021, ABES Engineering College

Algorithm

ALGORITHM: IndexSequentialSearch(A[ ], N, KEY)

BEGIN:

```
index[ ], i, flag = 0,j=0,x=0,start,end,bs
bs=3
FOR i = 0 TO N STEP+3 DO
        index[x] = i
        x=x+1
IF KEY<A[Index[0]] THEN
        WRITE("ITEM NOT FOUND")
        EXIT
ELSE IF KEY>A[Index[x-1]] THEN
        start=index[x-1]
        end=n
ELSE
        FOR i = 1 TO x DO
                IF KEY<A[index[i] THEN
                        start=index[i-1]
                        end=index[i]
                        BREAK
FOR i=start to end
                IF KEY==arr[i]
                flag=1
                break
If flag==1
                WRITE ELEMENT IS FOUND
ElSE
                WRITE ELEMENT NOT FOUND
END;
```

Copyright © 2021, ABES Engineering College

```
void indexse(int arr[],int n,int key){
    int index[20],i,flag=0;
    int j=0,x=0,start,end,bs;
    printf("enter block size");
    scanf("%d",&bs);
    for(i=0;i<n;i=i+bs)
        index[x++]=i;
    if(key<arr[index[0]] )
    {       printf("key is not found in array");
        exit(0);     }
    else if(key>=arr[index[x-1]])    {
        start=index[x-1];
        end=n;     }
    else
    {   for(i=1;i<x;i++)
        if(key<arr[index[i]])       {
            start=index[i-1];
            end=index[i];
                break;  }    }
    for(i=start;i<=end;i++)    {
        if(key==arr[i])       {
            flag=1;             break;       }    }
    if(flag==1)
        printf("key is found at postion %d",i+1);
    else
        printf("key not found"); }
void main() {
    int arr[]={1,3,4,6,9,10,11};
    int n=sizeof(arr)/sizeof(arr[0]);
    int key,i;
    printf("\nenter thekey which you want to
search");
    scanf("%d",&key);
    indexse(arr,n,key);
}
```

downloaded from pickywallpapers.com

Copyright © 2021, ABES Engineering College