Topic SEARCHING & SORTING. Unit No. 02

* SEARCHING :- Searching is a process of finding some element in the list. If the element is present in the list, then the process is called successful, and the process returns the location of that element ; otherwise the search is called unsuccessful.

Two popular methods are Linear search and Binary search.

1] LINEAR SEARCH :-

1) It is also called as Sequential Search algorithm.
2) It is the simplest searching algorithm.
3) Here, we simply traverse the array / list completely and match each element of the array with the element that we want to search.
This process is continued till we find the element (succesful search) or till we reach the end of the array (unsuccessful search).

Topic Unit No.

4) It is widely used to search an element from the unordered list/array.

5) The worst-case time complexity of linear search is O(n).

6) Steps :-

i) First, we have to traverse the array elements using a for loop.

ii) In each iteration of for loop, compare the search element with the current array element and -

- If the element is found, then return its corresponding index
- If the element is not found then move on to the next element.

iii) If there is no match found or the search element is not present in the given array, the simply return -1 or print the statement as "Invalid Number".

Topic Unit No.

7) Algorithm :-

i) Start

ii) Declare the required variables. like a[10], n, key, i, etc. int flag = 0;

iii) Read the array size in 'n'.

iv) Read the 'n' array elements.

v) Read the element to be searched in variable 'key'.

vi) Now start traversing the array from 0th location using for loop...

```
for (i = 0; i < n; i++)
```

vii) Check if the key matches with the array elements.

```
if (a[i] == key) // successful search
```

— Set flag to 1.

```
flag = 1 and
break; // if element found
```

// no need to traverse the further array.

— Close the for loop.

viii) If match not found then print "Invalid value".

Topic Unit No.

— for this use flag variable.

// Initially, flag is set to 0. So if
// match is not found its value will
// remain 0 only.

∴ if(flag==0)
printf("Invalid Number");

ix) Stop.

8) Example :-

Consider the following array :

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

key = 41.

↑ key ≠ 70

70	40	30	11	57	41	25	14	52

↑ key ≠ 40

70	40	30	11	57	41	25	14	52

↑ key ≠ 30

70	40	30	11	57	41	25	14	52

↑ key ≠ 11

Topic

Unit No.

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

key = 41

↑ key ≠ 52

70	40	30	11	57	41	25	14	52

↑ key = 41

Now, the element to be searched is found. So algorithm will return its index ⇒ [5].

Key element 41 is at location [5].

9) Pseudo Code :—

```
# include <stdio.h>
void main()
{
  int a[10], n, i, key, flag = 0
 printf("Enter array size");
  scanf("%d", &n);
 printf("Enter %d array elements", n);
 for (i=0; i<n; i++)
    scanf("%d", &a[i]);
 printf("\n Enter the key element");
    scanf("%d", &key);
```

Topic Unit No.

```
for (i=0; i<n; i++)
{
    if (key == a[i])
    { flag = 1;
      break;
    }
}
if (flag == 1)
printf ("\n Element %d present at location
        %d", key, i);
else
printf ("\n Invalid Element");
}
```

10) Time Complexity :-

i) Best Case = O(1)

Occurs when the element to be searched is available at the very first position of the array.

ii) Average Case = O(n)

Occurs when the key element is at the end of the array or somewhere at middle.

Topic Unit No.

iii) Worst Case :- O(n)
Occurs when the key element is at end of the array or not available in the array, but we have to traverse the whole array.

11) Advantages of Linear Search :-

i) Simple to implement and easy to understand.

ii) Can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.

iii) Does not require any additional memory.

iv) Well suited for small datasets.

12) Disadvantages :-

i) Slow for large data sets ⇒ time comp. O(n).

ii) Not suitable for large data sets.

iii) Less efficient.

Topic Unit No.

13) Space Complexity :- O(1).

14) Application :-
For searching operations in array of size < 100.

2] BINARY SEARCH :-

1) It is the search technique that works efficiently on sorted lists / arrays. Hence, to search an element into some array using binary search technique, we must ensure that the list is sorted.

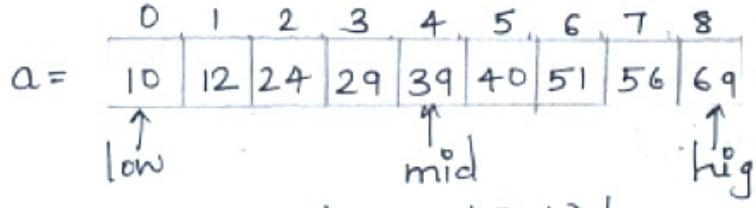
2) It follows divide and conquer approach in which the array is divided into 2 halves.
Then the key element is compared with the middle element of the array.
If match found, then location of middle element is returned.
Otherwise, we search the element into either of the halves depending upon the result produced through the match.

Topic Unit No.

Note :-
Binary search is implemented on sorted array elements. If the array is not sorted, we have to first sort it.

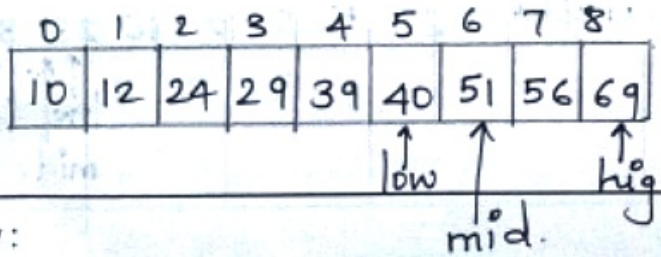
3) Example :-

i) Consider the following array.
Let the key be 56



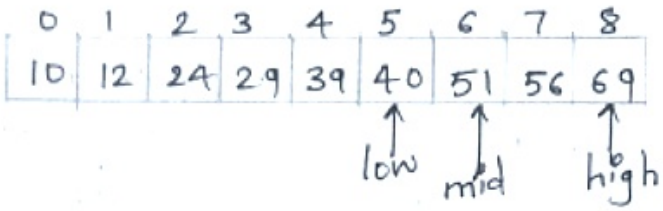
mid = (low + high)/2
mid = (0+8)/2
mid = 4

a[mid] = 39 ; key = 56
Check, if (key == a[mid]) ⇒ No.
Check, if (key < a[mid]) ⇒ No.
Check, if (key > a[mid]) ⇒ Yes.
Set low = mid+1.

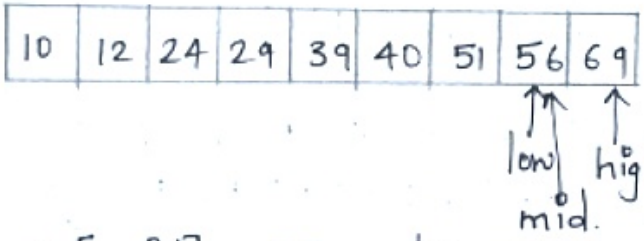


⇒ mid = (5+8)/2 = 6

Topic Discard the 1st half. Unit No.



a[mid] = 51 ; key = 56.
Here, key > a[mid]
so, set low = mid + 1.

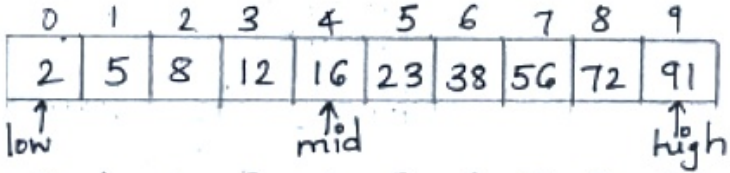


⇒ mid = (7+8)/2
= 7.

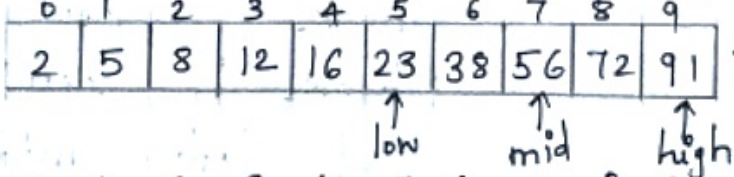
a[mid] = 56 ; key = 56
Here, a[mid] = key.
So return the location of mid where 56 is present.

ii)

Search 23.

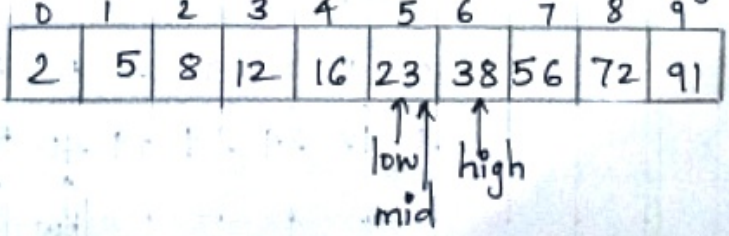


23 > 16, Consider 2nd half.



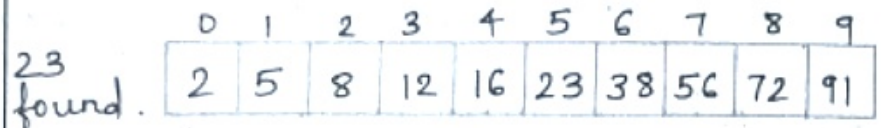
→ low = mid + 1

23 < 56, consider 1st half.



→ high = mid - 1

Topic Unit No.



Return mid position = 5

4) Algorithm :-

i) Start

ii) Declare the required variables like : a[10], key, i, low, mid, high, n

iii) Read the array size in 'n'.

iv) Read the 'n' array elements

v) Read the key element to be searched.

vi) Set,
low = 0
high = n-1
mid = (low + high)/2

vii) while (low <= high)
{
- Check if (key == a[mid])
  If yes, return the mid value.
- Else check if (key < a[mid])
  set high = mid - 1

Topic Unit No.

- Else set low = mid + 1 // if (key > a[mid]).
- Update mid = (low + high)/2

Close while loop.

viii) If the key element not available then,
if (low > high)
⇒ print "Element not found"

ix) Stop.

5> Pseudo Code :-

```
#include <stdio.h>
#include <conio.h>
int main()
{
 int a[10], key, n, i, low, high, mid;
 printf("Enter array size");
  scanf("%od", &n);
 printf("Enter %od array elements", n);
 for(i=0; i<n; i++)
    scanf("%d", &a[i]);
```

Topic Unit No.

```
printf(" Enter the key element to be deleted");
scanf("%d", &key);

low = 0;
high = n-1;
mid = (low + high)/2;

while (low <= high)
{
   if (a[mid] < key)
         low = mid+1;
   else if (a[mid] == key)
   {
      printf(" Key %d found at %d location",
                  key, mid);
      break;
   }
   else
         high = mid-1;

   mid = (low+high)/2;
}
if(low > high)
   printf("Invalid element");
getch();
return(0);
}
```

Topic | Unit No.

6) Time Complexity :-

i) Best Case = O(1).
Occurs when the element to be searched is found in 1st comparison
i.e. when the 1st middle element itself is the key element.

ii) Average Case = O(log n)
Occurs when the element is somewhere in between and we have to keep on reducing the array.

iii) Worst Case = O(log n)
Occurs when we have to keep reducing the search space till it has only one element or the key element is not atall available in the array.

7) Space Complexity :- O(1). as no extra space is used.

8) Applications :-
i) Used to search element in large data sets.
ii) Dictionary.
iii) Computer graphics.

Topic Unit No.

9) Advantages :-

i) Faster than linear search.

ii) Used for larger data sets.

iii) Simple to implement and easy to understand.

iv) As the array size increases, the time complexity increases logarithmically.

10) Disadvantages :-

i) Array should always be sorted. If array is not sorted, we first have to sort it. This adds an additional O(N * log N) time complexity for sorting step.