LINEAR SEARCH

Searching an element within an array an array of n elements, where each Consider element is a key (e.g., a number). The articular key(number) in the array. task is to

The simplest method is a sequential search or linear search. The idea is to simply search the array, element by element, from the beginning until the key is found or the end of the list is reached. If found, the corresponding position in the array is printed; otherwise, a message will have to be displayed that the key(number) is not found. Now, the implementation of the program will be

Program:

```
#include
<stdio.h>
#include
<stdlib.h>      int
main()
{
int n,i,key, FOUND=0, a[30]; // array declaration printf("\n How many numbers:"); scanf("%d",&n); // array size
printf("\n Enter the array elements:
\n");for(i=0 ; i<n; i++)
{
scanf("%d", &a[i]);
}
printf("\n Enter the key to be searched: ");scanf("%d",&key);

for(i=0 ; i<n; i++) // searching an element in an arrayif(a[i] == key)
{
printf("\n Found at %d",i);FOUND=1;
}
if(FOUND = = 0)
printf("\n NOT FOUND...");return 0;
}
```

Output

```
How many numbers: 6
Enter the array elements:
21
33
46
52
27
Enter the key to be searched: 73
NOT FOUND
```

Output

```
How many numbers: 6
Enter the array elements:
21
33
46
52
27
Enter the key to be searched: 52
Found at 3
```

BINARY SEARCHING

In ***Binary searching*** the drawbacks of sequential search can be

eliminatedThe binary search halves the size of the list to search

in each Iteration

Logic : Binary search can be explained simply by the analogy of searching for a page in a book. Suppose a reader is searching for page 90 in a book of 150 pages. The reader would first open the book at random towards the latter half of the book. If the page number is less than 90, the reader would open at a page to the right; if it is greater than 90, the reader would open at a page to the left, repeating the process till page 90 was found.

Binary search requires sorted data (in ascending order) to operate on.

In binary search, the following procedure is implemented.

- Look at the middle element of the list.
- If it is the value being searched, then the job is done.
- If the value that is being searched is smaller than the middle element, then continuewith the bottom half of the list.
- If the value that is being searched is larger than the middle element, then continuewith the top half of the list.

Eg:-Depiction of binary search algorithm (the number to be searched is greater than midvalue)

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	23	27	29	32	34	41	46	47	49	52	55	68	71	74	77	78
1st iteration	L						M									H
2nd iteration								L				M				H
3rd iteration								L	M	H						

Algorithm*:* The algorithm determines the position of T in the LIST.

1. START
2. PRINT “ENTER THE NO. OF ELEMENTS IN THE ARRAY”
3. INPUT

N4. I=0

5. PRINT “ENTER ARRAY ELEMENT”
6. INPUT

LIST(I) 7.

I=I+1

8. IF I<N THEN GOTO STEP 5
9. PRINT “ENTER THE ELEMENT TO SEARCH”
10. INPUT T
11. HIGH = N - 1
12. LOW = 0

13. FOUND = 0
14. MID = (HIGH + LOW)/ 2
15. IF T = LIST [MID]
        FOUND = 1
    ELSE IF T < LIST[MID]
        HIGH = MID-1
    ELSE
    LOW = MID+1
16. IF (FOUND =0) and (HIGH > = LOW) THEN GOTO STEP 14

17. IF FOUND =0 THEN PRINT "NOT FOUND"
18. ELSE PRINT "FOUND AT", MID.
19. STOP

The C program for this algorithm is as follows:

```
#inc
lude
<std
io.h
>
#inc
lude
<std
lib.h
> int
mai
n()
{
int a[30],n,i,t,low,mid,high,found=0;
printf("\n Enter the number of
elements in the array:");
scanf("%d",&n);
printf("\n Enter the
elements of the array:");
for(i=0 ; i< n; i++)
scanf("%d", &a[i]);
printf("\n Enter the
element to search :");
scanf("%d",&t);
low = 0; high = n - 1;
while(high >= low)
{
mid = (low + high) / 2;if(a[mid] == t)
{
found = 1;break;
}else if (t < a[mid])high = mid - 1;
 else
low = mid + 1;
}
if(found==0)
printf("\n NOT FOUND");else
printf("\n FOUND AT %d",mid);return 0;
}
```

Output

Enter the number of elements in the array: 9
Enter the number of elements in the array 9
Enter the elements of the array:
Enter the elements of the array:

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
Enter the element to search: 7	Enter the element to search: 7
FOUND AT 6	NOT FOUND