Searching is a technique of finding the location of a given item in the linear array. The search is successful if the given item is found in the array otherwise it is unsuccessful.
Two types of searching technique are –

1. Linear Search (Sequential search)
2. Binary Search

1. Linear search:

In this technique, the array is usually unordered, i.e. not sorted. The given item is searched sequentially in the array, that is, the given item is compared with each element of the array one by one to find its presence.

```
#include <stdio.h>
int main()
{
     int a[50],n,i,key,loc;
     int  linear_search(int *,int,int);

     printf("\n Enter no. of elements : ");
     scanf("%d",&n);

     printf("\nEnter %d elements of array\n",n);

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

     printf("\nEnter item to search : ");
     scanf("%d",&key);

     loc=linear_search(a,n,key);

     if(loc==-1)
          printf("\nKey does not exist in the array.....\n");
     else
          printf("\nKey is found at array index %d\n",loc);

     return(0);
}

int linear_search(int a[],int n, int key)
{
     int i;
     for(i=0;i<n;i++)
     {
          if(a[i] == key)
               return(i);
     }
     return(-1);
}
```

Time Complexity of Linear Search Method:

- Best Case Time Complexity: It occurs when the first comparison between **key and a[i]** returns a match. So, best case time complexity, $T_b(n) = O(1)$.
- Worst Case Time Complexity: It occurs when search for the key is unsuccessful.
  Total number of comparisons required = n+1.
  Hence, worst case time complexity, $T_w(n) = O(n)$.

[Note :

- **Time complexity** of an algorithm is the **amount of computer time** that it needs to run to completion.
  - Best case Time Complexity:
    It is a measure of minimum time that an algorithm will require for an input size "n".
  - Worst case Time Complexity:
    It is a measure of maximum time that an algorithm will require for an input size "n".

]

2. Binary search:

In this technique, the array must be sorted. It works as follows –

1. It first compares key to the middle element of the array.
   - ✓ if key is less than the middle value, searching focuses on left half otherwise on right half.
   - ✓ In either case, next step is to compare key to the middle value of the selected half.
2. This process continues until the key is found or the range is empty.

The working of this technique is illustrated through the following example.
Let us consider sorted array A with 7 elements as follows –

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
3	11	15	29	35	46	66

Let item to be searched be 15, i.e. KEY = 15.

Iteration 1:

We have LB = 0 and UB = 6. So MID = $\lfloor (LB + UB) / 2 \rfloor = \lfloor (0 + 6) / 2 \rfloor = 3$
Compare KEY with A[MID] and as 15 < A[3] , hence, UB = MID – 1 = 3 – 1 = 2 and LB remains unchanged.
As LB < UB, we go for next iteration.

Iteration 2:
Now, LB = 0 and UB = 2. So MID = ⌊ (0 + 2) / 2 ⌋ = 1
As 15 > A[1], LB = MID + 1 = 1 + 1 = 2 and UB remains unchanged.
As LB = UB, we go for next iteration

Iteration 3:
Now, LB = 2 and UB = 2. So MID = ⌊ (2 + 2) / 2 ⌋ = 2
As 15 = A[2], the search terminates on success.
Hence, item **15 is found at index 2**.

```
#include <stdio.h>

int main()
{
     int a[50],n,i,key,loc;
     int  binary_search(int *,int,int,int);

     printf("\n Enter no. of elements : ");
     scanf("%d",&n);

     printf("\nEnter %d elements in ascending order\n",n);

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

     printf("\nEnter item to search : ");
     scanf("%d",&key);

     loc=binary_search(a,0,n-1,key);

     if(loc==-1)
          printf("\nKey does not exist in the array.....\n");
     else
          printf("\nKey is found at array index %d\n",loc);

     return(0);
}

int binary_search(int a[],int lb, int ub, int key)
{
     int mid;

     while(lb<=ub)
     {
          mid=(lb+ub)/2;
```

```
            if(key == a[mid])
                  return(mid);
            else if(key < a[mid])
                  ub=mid-1;
            else
                  lb=mid+1;
      }
      return(-1);
}
```

==========

Merit: It is very efficient searching technique for large input size n.

Let us suppose, there are 10,00,000 elements in an array A.

Now, $2^{10} = 1024 > 1000$.

Therefore, $2^{20} > (1000)^2 = 10,00,000$.

Hence total number of comparisons required is **at most 20** to find the location of any item in the list.

Demerits:

The list must be sorted.

Time Complexity of Binary Search:

1. Best Case: It occurs if the **first comparison** between **A[mid] and key** returns a match. So $T_b(n) = O(1)$.

2. Worst Case: It occurs when key is greater than maximum element in the array. Hence, worst case time complexity, $T_w(n) = O(\log n)$

Differentiate between Linear Search and Binary search:

Linear Search	Binary Search
Input data need not be sorted	Input data must be sorted
Worst Case time complexity is O(n)	Worst case time complexity is O(log n)
It accesses data sequentially	It accesses data randomly.
It is preferable for small list	It is efficient for large list.

Problem:

Write a **recursive** C- function for Binary Search. Also call it from main().