Objective:

To check/**search** for the element, just compare with the number to each element present in the array if any element equal to the entered number then print the exact position of the number in the array as shown here in the following program.

Description:

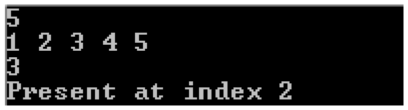
Linear Search. **Linear search**, also known as **sequential search**, is a process that checks every element in the list sequentially until the desired element is found. The computational complexity for **linear search** is O(n)

Code:

```
#include<bits/stdc++.h>

using namespace std;

int searchr(int arr[],int n,int x)

{

   int i;

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

      if(arr[i]==x)

      return i;

   return -1;

}

int main()

{

   int n;

   int x;

   cin>>n;

   int ar[n];

   for(int i=0;i<n;i++) cin>>ar[i];

   cin>>x;

   int result=searchr(ar,n,x);

   (result==-1)? cout<<"Not present in the array"

              : cout<<"Present at index " <<result;
```

```
    return 0;

}
```

Code output:



```
5
1 2 3 4 5
3
Present at index 2
```

Discussion:

A **Linear Search** is the most basic type of **searching** algorithm. A **Linear Search** sequentially moves through your collection (or data structure) looking for a matching value

Objective:

To find a random word you know to be in the dictionary, **binary search** will find your target definition/word after you've checked log2(1000) ≅ 10 words. **Searching** iteratively one-after-the-next would require checking an average (1001/2) ≅ 500 words. **Binary search** is much faster than iterative **search**.

Description:

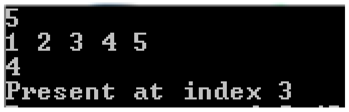
Binary search, also known as half-interval **search**, logarithmic **search**, or **binary** chop, is a **search** algorithm that finds the position of a target value within a sorted array. ... If the **search** ends with the remaining half being empty, the target is not in the array.

Code:

```
#include"bits/stdc++.h"

using namespace std;

int binary(int arr[],int l,int r,int x)

{

    if(r>=l){

        int mid=l+(r-l)/2;

    if(arr[mid]==x)

        return mid;

    if(arr[mid]>x)

        return binary(arr,l,mid-1,x);

    return binary(arr,mid+1,r,x);

    }
```

```
   return -1;
}
int main()
{
   int n,x;
   cin>>n;
   int ar[n];
   for(int i=0;i<n;i++) cin>>ar[i];
   cin>>x;
   int result=binary(ar,0,n-1,x);
   (result==-1)?cout<<"Element is not present in the array"
               :cout<<"Present at index "<<result;
   return 0;
}
```

Code output:



```
5
1 2 3 4 5
4
Present at index 3
```

Discussion:

Binary search compares the target value to the middle element of the array. If they are not equal, the half in which the target cannot lie is eliminated and the **search** continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found.