Linear Search

Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.

It compares the element to be searched with all the elements present in the array and when the element is **matched** successfully, it returns the index of the element in the array, else it return `-1`.
Linear Search is applied on unsorted or unordered lists, when there are fewer elements in a list.

Features of Linear Search Algorithm

1. It is used for unsorted and unordered small list of elements.
2. It has a time complexity of **O(n)**, which means the time is linearly dependent on the number of elements, which is not bad, but not that good too.
3. It has a very simple implementation.

Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found.
The time complexity of Linear search algorithm is **O(n)**,

Implementing Linear Search

Following are the steps of implementation that we will be following:

1. Traverse the array using a `for` loop.
2. In every iteration, compare the `target` value with the current value of the array.
    - If the values match, return the current index of the array.
    - If the values do not match, move on to the next array element.
3. If no match is found, return `-1`.

```
/* linear search*/
main()
{ intarr[10],count,num,c=0;
  /* enter elements*/
for(count=0;count<10;count++)
  {
printf("Enter a no");
scanf("%d",&arr[count]);
  }

printf("enter the element to search");
scanf("%d",&num);
  /* search */
for(count=0;count<10;count++)
  {
if (arr[count]==num)
   {     printf("%d  is present in  the position %d”,num,count+1);
c++;
   }
  }
if (c==0)
printf("element not present");
getch();
}
```

Binary Search

Binary Search is used with sorted array or list. In binary search, we follow the following steps:

1. We start by comparing the element to be searched with the element in the middle of the list/array.
2. If we get a match, we return the index of the middle element.
3. If we do not get a match, we check whether the element to be searched is less or greater than in value than the middle element.
4. If the element/number to be searched is greater in value than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the step 1.
5. If the element/number to be searched is lesser in value than the middle number, then we pick the elements on the left side of the middle element, and start again from the step 1.

Binary Search is useful when there are large number of elements in an array and they are sorted.

So a necessary condition for Binary search to work is that the list/array should be sorted.

Features of Binary Search

1. It is great to search through large sorted arrays.
2. It has a time complexity of **O(log n)** which is a very good time complexity.
3. It has a simple implementation.

Binary Search is applied on the sorted array or list of large size. It's time complexity of **O(log n)** makes it very fast as compared to other sorting algorithms. The only limitation is that the array or list of elements must be sorted for the binary search algorithm to work on it.

Implementing Binary Search Algorithm

Following are the steps of implementation that we will be following:

1. Start with the middle element:
    - If the **target** value is equal to the middle element of the array, then return the index of the middle element.
    - If not, then compare the middle element with the target value,
        - If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
        - If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.
2. When a match is found, return the index of the element matched.
3. If no match is found, then return `-1`

```
/* Binary search*/

main()
{ intarr[10],count,num,flag=0,low=0,mid,high=10,rcount,temp;
  /* enter elements*/
for(count=0;count<10;count++)
  {
printf("Enter a no");
scanf("%d",&arr[count]);
  }
  /* linear sorting */
  for(rcount=0;rcount<10;rcount++)
  {
    for(count=rcount+1;count<10;count++)
    {
if (arr[rcount]>arr[count])
      { temp=arr[rcount];
          arr[rcount]=arr[count];
          arr[count]=temp;
      }
   }
  }
  /* sorted element*/
  for(count=0;count<10;count++)
printf("\n%d",arr[count]);

printf("enter the element to search");
scanf("%d",&num);

  /* search */
while(low<high)
  {
   mid=(low+high)/2;
   if (arr[mid]==num)
   {  flag=1;
      break;
   }
   else
   {
    if (arr[mid]<num)
     low=mid+1;
    else
     high=mid-1;
    }
  }
  if (flag==1)
printf("yes");
  else
printf("no");
getch();
}
```