Linear Search Algorithm

Searching is the process of finding some particular 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 search methods are Linear Search and Binary Search

What is linear search?

Linear search is also called as **sequential search algorithm.** It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted. The worst-case time complexity of linear search is **O(n).**

The steps used in the implementation of Linear Search are listed as follows -

- First, we have to traverse the array elements using a **for** loop.
- In each iteration of **for loop,** compare the search element with the current array element, and -
  - If the element matches, then return the index of the corresponding array element.
  - If the element does not match, then move to the next element.
- If there is no match or the search element is not present in the given array, return **-1.**

Algorithm

1. Linear_Search(a, n, val) // 'a' is the given array, 'n' is the size of given array, 'val' is the value to search
2. Step 1: set pos = -1
3. Step 2: set i = 1
4. Step 3: repeat step 4 while i <= n
5. Step 4: if a[i] == val
6. set pos = i
7. print pos
8. go to step 6
9. [end of if]
10. set ii = i + 1
11. [end of loop]
12. Step 5: if pos = -1
13. print "value is not present in the array "
14. [end of if]
15. Step 6: exit

Example:

Let the elements are: 10,6,3,8,9,12,14

The search element is : 12

Now it compare 12 with each and every element.

The 12 is available in 6th place.

So the searching process is success and element is found

Linear Search complexity

Now, let's see the time complexity of linear search in the best case, average case, and worst case. We will also see the space complexity of linear search.

1. Time Complexity

Case	Time Complexity
Best Case	O(1)
Average Case	O(n)
Worst Case	O(n)

- **Best Case Complexity -** In Linear search, best case occurs when the element we are finding is at the first position of the array. The best-case time complexity of linear search is **O(1).**
- **Average Case Complexity -** The average case time complexity of linear search is **O(n).**
- **Worst Case Complexity -** In Linear search, the worst case occurs when the element we are looking is present at the end of the array. The worst-case in linear search could be when the target element is not present in the given array, and we have to traverse the entire array. The worst-case time complexity of linear search is **O(n).**

2. Space Complexity

Space Complexity	O(1)

The space complexity of linear search is O(1).

```c
#include <stdio.h>
int linearSearch(int a[], int n, int val) {
  // Going through array sequencially
  for (int i = 0; i < n; i++)
    {
       if (a[i] == val)
       return i+1;
    }
  return -1;
}

 int main() {
int a[] = {70, 40, 30, 11, 57, 41, 25, 14, 52}; // given array
   int val = 41; // value to be searched
   int n = sizeof(a) / sizeof(a[0]); // size of array
   int res = linearSearch(a, n, val); // Store result
   printf("The elements of the array are - ");
   for (int i = 0; i < n; i++)
   printf("%d ", a[i]);
   printf("\nElement to be searched is - %d", val);
   if (res == -1)
   printf("\nElement is not present in the array");
   else
   printf("\nElement is present at %d position of array", res);
   return 0;
}
```

Output

```
The elements of the array are - 70 40 30 11 57 41 25 14 52
Element to be searched is - 41
Element is present at 6 position of array
```

Binary Search Algorithm

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.

Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

> ***NOTE: Binary search can be implemented on sorted array elements. If the list elements are not arranged in a sorted manner, we have first to sort them.***

Algorithm

1. Binary_Search(a, lower_bound, upper_bound, val) // 'a' is the given array, 'lower_bound' is the in dex of the first array element, 'upper_bound' is the index of the last array element, 'val' is the val ue to search
2. Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
3. Step 2: repeat steps 3 and 4 while beg <=end
4. Step 3: set mid = (beg + end)/2
5. Step 4: if a[mid] = val
6. set pos = mid
7. print pos
8. go to step 6
9. else if a[mid] > val
10. set end = mid - 1
11. else
12. set beg = mid + 1
13. [end of if]
14. [end of loop]
15. Step 5: if pos = -1
16. print "value is not present in the array"

17. [end of if]
18. Step 6: exit

Binary Search complexity

Now, let's see the time complexity of Binary search in the best case, average case, and worst case. We will also see the space complexity of Binary search.

1. Time Complexity

Case	Time Complexity
Best Case	O(1)
Average Case	O(logn)
Worst Case	O(logn)

- **Best Case Complexity -** In Binary search, best case occurs when the element to search is found in first comparison, i.e., when the first middle element itself is the element to be searched. The best-case time complexity of Binary search is **O(1).**
- **Average Case Complexity -** The average case time complexity of Binary search is **O(logn).**
- **Worst Case Complexity -** In Binary search, the worst case occurs, when we have to keep reducing the search space till it has only one element. The worst-case time complexity of Binary search is **O(logn).**

2. Space Complexity

Space Complexity	O(1)

- The space complexity of binary search is O(1).

Program: Write a program to implement Binary search in C language.

```c
#include <stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
   int mid;
   if(end >= beg)
   {      mid = (beg + end)/2;
/* if the item to be searched is present at middle */
      if(a[mid] == val)
      {
         return mid+1;
      }
         /* if the item to be searched is smaller than middle, then it can only be in left sub array */
      else if(a[mid] < val)
      {
         return binarySearch(a, mid+1, end, val);
      }
       /* if the item to be searched is greater than middle, then it can only be in right subarray */
      else
      {
         return binarySearch(a, beg, mid-1, val);
      }
   }
   return -1;
}
int main() {
  int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array
  int val = 40; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
```

```
29.  int res = binarySearch(a, 0, n-1, val); // Store result
30.  printf("The elements of the array are - ");
31.  for (int i = 0; i < n; i++)
32.  printf("%d ", a[i]);
33.  printf("\nElement to be searched is - %d", val);
34.  if (res == -1)
35.  printf("\nElement is not present in the array");
36.  else
37.  printf("\nElement is present at %d position of array", res);
38.  return 0;
39. }
```

Output

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array
```