Assignment Number

Problem Statement

Program in C to implement linear and binary search.

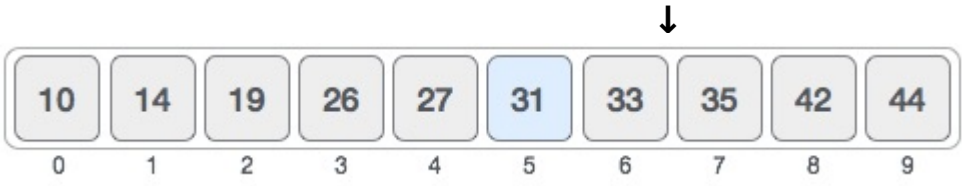
Theory

To search an element in a given array, it can be done in two ways:
Linear search & Binary search.

Linear Search

A linear search is the basic and simple search algorithm. A linear search searches an element or value from an array till the desired element or value is not found and it searches in a sequence order. It compares the element with all the other elements given in the list and if the element is matched it returns the value index else it return -1. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.

In the following example, to find the element 35 by linear searching the searching have to go through all the other elements before 35.

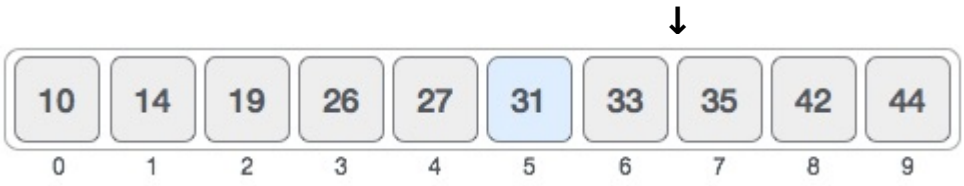


Binary Search

Binary Search is applied on a sorted array or list. In binary search, we first compare the value with the elements in the middle position of the array. If the value is matched, then we return the value. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We

repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array.

In the following example, to find 35 by binary searching we will first start from 27 and see whether 35 is greater or lesser than 27 and after that we will resize the array to the half of the previous array by the right or left elements of the array respectively. And we will continue this process until the element 35 will be found or the resized array's first index extends the last index.



Algorithm

Input specification: An array (Sorted for linear/Binary search or Unsorted for linear search) say **a[]**, where the search will be done, The element which need to be searched, say **find** and the number of elements in the array a[], say **n**.

Output specification: Success message of the search with the position of the element or appropriate failure message.

Steps:

Algorithm for method linear_search(a[], n, find):

Step 1 : For (c = 1 to n)

a) If (a[c] == find)
   Then
   i. Print "Element " search " is present at location " c
   ii. Exit
   EndIf

b) Set c=c+1

Step 2 : End For

Step 3 : Print "Element " search " is not present in array."

Algorithm for method binary_search(a[], n, find):

Step 1 : Set first = 1
Step 2 : Set last = n
Step 3 : Set middle = (first+last)/2
Step 4 : While (first <= last)
  a. If (a[middle] < find) Then
    i. Set first = middle + 1
  b. Else If (a[middle] == find) Then
    i. Print “Element ” search" is present at location " middle
    ii. Exit
  c. Else
    i. Set last = middle - 1
  d. Set middle = (first + last)/2
  EndWhile
Step 5 : If (first > last) Then
Step 6 : Print search" is not present in array."

Source Code

```
#include <stdio.h>

long linear_search(long a[], long n, long find);
long binary_search(long a[], long n, long find);

int main()
{
  long array[100], search, i, n, position;
  int ch;

  printf("Input number of elements in array: ");
  scanf("%ld", &n);

   for (i = 0; i < n; i++){
            printf("Enter the element no %d: ",i+1);
```

```
            scanf("%ld", &array[i]);
        }
        printf("Input number to search: ");
        scanf("%ld",&search);
        printf("1.Linear search\n2.Binary search\nEnter your choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1: position = linear_search(array, n, search);
                    break;
            case 2: position = binary_search(array, n, search);
                    break;
        }
    if (position == -1)
        printf("%d is not present in array.\n", search);
    else
        printf("%d is present at location %d.\n", search, position);

        return 0;
}
long linear_search(long a[], long n, long find)
{
  long c;
  for (c = 0 ;c < n ; c++ )
  {
    if (a[c] == find)
      return c+1;
  }
  return -1;
}
long binary_search(long a[], long n, long find)
{
        int first = 0; long last = n-1, middle;
        middle = (first+last)/2;
        while (first <= last) {
            if (a[middle] < find)
```

```
                first = middle + 1;
            else if (a[middle] == find) {
                return middle+1;
            }
        else
            last = middle - 1;
        middle = (first + last)/2;
        }
        if (first > last)
            return -1;
}
```

Input and Output

Set 1 :

Input number of elements in array: 5
Enter the element no 1: 38
Enter the element no 2: 291
Enter the element no 3: 382
Enter the element no 4: 410
Enter the element no 5: 491
Input number to search: 67
1.Linear search
2.Binary search
Enter your choice: 1
67 is not present in array.

Set 2 :

Input number of elements in array: 5
Enter the element no 1: 12
Enter the element no 2: 38
Enter the element no 3: 59
Enter the element no 4: 70
Enter the element no 5: 89
Input number to search: 59

1.Linear search
2.Binary search
Enter your choice: 2
59 is present at location 3.

Discussion

1. Linear search is rarely used practically because other search algorithms such as the binary search algorithm and hash tables allow significantly faster searching comparison to linear search.
2. A linear search scans one item at a time, without jumping to any item .
    a. The worst case complexity is O(n), sometimes known an O(n) search.
    b. Time taken to search elements keep increasing as the number of elements are increased.
3. A binary search however, cut down your search to half as soon as you find middle of a sorted list.
    a. The middle element is looked to check if it is greater than or less than the value to be searched.
    b. Accordingly, search is done to either half of the given list.
4. Input data needs to be sorted in Binary Search and not in Linear Search
5. Linear search does the sequential access whereas Binary search access data randomly.
6. Time complexity of linear search O(n) , Binary search has time complexity O(log n).
7. Linear search performs equality comparisons and Binary search performs ordering comparisons.