INDEX

Ex. No.: 12

Date :

Implementation of Searching Techniques

Aim: To write a C-program to implement Linear Search and Binary Search.

1. Linear search:

Linear search or **sequential search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

A linear search runs in at worst linear time and makes at most n comparisons, where n is the length of the list. If each element is equally likely to be searched, then linear search has an average case of n/2 comparisons, but the average case can be affected if the search probabilities for each element vary. Linear search is rarely practical because other search algorithms and schemes, such as the binary search algorithm and hash tables, allow significantly faster searching.

1.1 Algorithm

Step 1 - Start from the leftmost element of array and one by one compare x with each element of array.
Step 2 - If x matches with an element, return the index.
Step 3 - If x doesn't match with any of elements, return -1.

1.2 Program

```
//Program to Demonstrate linear search algorithm
#include<stdio.h>
#include<conio.h>

void main()
{
  int list[20],size,i, key;

  printf("Enter size of the list: ");
  scanf("%d",&size);

  printf("Enter any %d integer values: ",size);
  for(i = 0; i < size; i++)
  scanf("%d",&list[i]);
  while(1)
    {
        printf("\nEnter the element to be Search: ");
        scanf("%d",&key);

        // Linear Search Logic
        for(i = 0; i < size; i++)
        {
            if(key == list[i])
            {
                printf("Element is found at %d index", i);
                break;
            }
        }
        if(i == size)
            printf("Given element is not found in the list!!!");
    }
  getch();
}
```

Worst Case Performance: O(n)
Average Case Performance: O(n)
Best Case Performance: O(1)
Space Complexity: O(*n*) total, O(*1*) auxiliary

2. Binary search:

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. 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. If the search ends with the remaining half being empty, the target is not in the array.

2.1 Algorithm

i. Compare x with the middle element.
ii. If x matches with middle element, we return the mid index.
iii. Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
iv. Else (x is smaller) recur for the left half.

2.2 Program

```
//C program to Demonstrate binary search
#include<stdio.h>
#include<conio.h>

void main()
{
   int first, last, middle, size, i, key, list[100];
   //clrscr();

   printf("Enter the size of the list: ");
   scanf("%d",&size);

   printf("Enter %d integer values in ascending order\n", size);

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

   while(1)
   {
        for (i = 0; i < size; i++)
            printf("  %d",list[i]);

        // Binary search logic

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

       first = 0;
       last = size - 1;
       middle = (first+last)/2;

       while (first <= last) {
          if (list[middle] < key)
             first = middle + 1;
          else if (list[middle] == key) {
             printf("Element found at index %d.\n",middle);
             break;
          }
          else
             last = middle - 1;

          middle = (first + last)/2;
       }
       if (first > last)
          printf("Element Not found in the list.");

   }
   getch();
}
```

Worst Case Performance: O(log n)
Average Case Performance: O(log n)
Best Case Performance: O(1)
Space Complexity: O(*n*) total, O(*1*) auxiliary