Government College of Engineering, Jalgaon		
(An Autonomous Institute of Government of Maharashtra)		
Name :	**Semester :** VI	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25	**Subject :** CO356U DAA Lab
Course Teacher : Mr. Vinit Kakde		
Date of Performance :	**Date of Completion :**	

Practical no. 1

Aim : Recursive and non -recursive algorithms for a specific problem and their complexity measures.

Linear Search (non- recursive) :-

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.

Time and Space Complexity of Linear Search Algorithm:

Time Complexity:

- **Best Case:** In the best case, the key might be present at the first index. So the best case complexity is O(1)
- **Worst Case:** In the worst case, the key might be present at the last index i.e., opposite to the end from which the search has started in the list. So the worst-case complexity is O(N) where N is the size of the list.
- **Average Case:** O(N)

Auxiliary Space: O(1) as except for the variable to iterate through the list, no other variable is used.

Applications of Linear Search Algorithm:

- **Unsorted Lists:** When we have an unsorted array or list, linear search is most commonly used to find any element in the collection.
- **Small Data Sets:** Linear Search is preferred over binary search when we have small data sets with
- **Searching Linked Lists:** In linked list implementations, linear search is commonly used to find elements within the list. Each node is checked sequentially until the desired element is found.
- **Simple Implementation:** Linear Search is much easier to understand and implement as compared to Binary Search or Ternary Search.

Advantages of Linear Search Algorithm:

- Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.
- Does not require any additional memory.
- It is a well-suited algorithm for small datasets.

Disadvantages of Linear Search Algorithm:

- Linear search has a time complexity of O(N), which in turn makes it slow for large datasets.
- Not suitable for large arrays.

When to use Linear Search Algorithm?

- When we are dealing with a small dataset.
- When you are searching for a dataset stored in contiguous memory.

Linear Search Algorithm

The algorithm for linear search is relatively simple. The procedure starts at the very first index of the input array to be searched.

Step 1 − Start from the 0th index of the input array, compare the key value with the value present in the 0th index.

Step 2 − If the value matches with the key, return the position at which the value was found.

Step 3 − If the value does not match with the key, compare the next element in the array.

Step 4 − Repeat Step 3 until there is a match found. Return the position at which the match was found.

Step 5 − If it is an unsuccessful search, print that the element is not present in the array and exit the program.

Pseudocode :-

```
procedure linear_search (list, value)

  for each item in the list

    if match item == value

      return the item's location

    end if

  end for

end procedure
```

Binary Search (Recursive) :-

Binary Search is a search algorithm that is used to find the position of an element (target value) in a sorted array. The array should be sorted prior to applying a binary search.

Binary search is also known by these names, logarithmic search, binary chop, half interval search.

Time Complexities

- **Best case complexity: O(1)**
- **Average case complexity: O(log n)**
- **Worst case complexity: O(log n)**

Space Complexity

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

Algorithm :-

Step 1 : Find the middle element of array. using , middle = initial_value + end_value / 2 ;

Step 2 : If middle = element, return 'element found' and index.

Step 3 : if middle > element, call the function with end_value = middle - 1 .

Step 4 : if middle < element, call the function with start_value = middle + 1 .

Step 5 : exit.

Linear Search Program:-

```
#include <stdio.h>
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Return the index of the found element
        }
    }
    return -1; // Return -1 if the element is not found
}
int main() {
    int size, target, index;
    printf("Enter the number of elements: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter %d elements:\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &target);
```

```
index = linearSearch(arr, size, target);
if (index != -1) {
       printf("Element found at index %d\n", index);
  } else {
       printf("Element not found in the array\n");
  }

    return 0;
}
```

Output:-

```
Enter the number of elements: 5
Enter 5 elements:
34
56
78
09
43
Enter the element to search: 78
Element found at index 2

--------------------------------
Process exited after 15.34 seconds with return value 0
Press any key to continue . . .
```

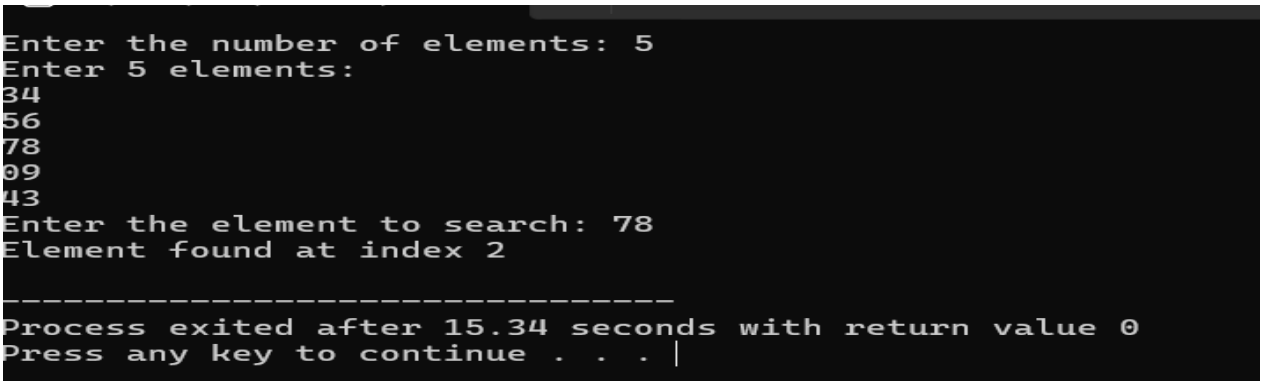
Binary Search Program:-

```
#include <stdio.h>
int binarySearch(int arr[], int left, int right, int target) {
    if (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid;
        }
       if (arr[mid] > target) {
            return binarySearch(arr, left, mid - 1, target);
        }
       return binarySearch(arr, mid + 1, right, target);
    }
  return -1;
}

int main() {
    int size, target, index;
    printf("Enter the number of elements: ");
    scanf("%d", &size);
```

```
    int arr[size];
    printf("Enter %d sorted elements:\n", size);
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &target)
    index = binarySearch(arr, 0, size - 1, target);
    if (index != -1) {
        printf("Element found at index %d\n", index);
    } else {
        printf("Element not found in the array\n");
    }
    return 0;
}
```

Output:-



Question:-

1) **How to perform a recursive binary search?**

- Algorithm for Recursive Binary Search: Make a recursive function, and compare the key's midpoint with the search space's midpoint. And depending on the outcome, either invoke the recursive procedure for the following search space or return the index where the key was discovered.

2)How to perform binary search?

- Below are the basic steps to performing Binary Search. Find the mid element of the whole array, as it would be the search key. Look at whether or not the search key is equivalent to the item in the middle of the interval and return an index of that search key.

3) **Which search algorithm is faster?**
Binary Search is faster with **O(log n)** complexity, while **Linear Search** takes **O(n)** time.

4) **Does Binary Search work on unsorted arrays?**
No, the array must be **sorted** for Binary Search to work.

5) **What is the worst-case time complexity of Linear Search?**
O(n) (when the element is at the end or not found).

Sign of Teacher