



**Batch:** A-4 **Roll No.:** 16010422211 **Experiment: 08**

**Aim: Demonstrate the use of Sorting in binary search algorithm**



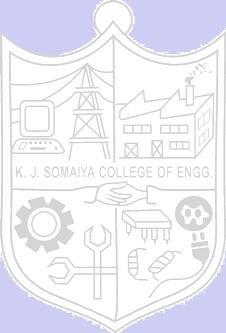
**Resources Used:** Turbo C/ C++ editor and C compiler.



**Theory:**

# Searching –

Search is a process of finding a value in a list of values. In other words, searching is the process of locating given value position in a list of values.

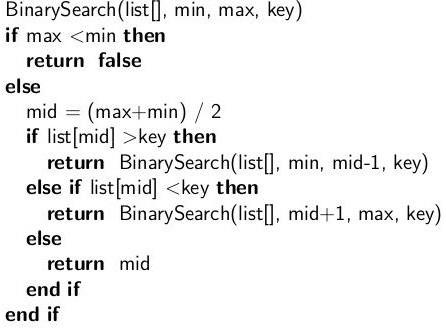
The **binary search algorithm** can be used with only sorted list of element. That means, binary search can be used only with list of element which are already arranged in a order. The binary search cannot be used for list of element which are in random order. This search process starts comparing of the search element with the middle element in the list. If both are matched, then the result is "element found". Otherwise, we check whether the search element is smaller or larger than the middle element in the list. If the search element is smaller, then we repeat the same process for left sub-list of the middle element. If the search element is larger, then we repeat the same process for right sub-list of the middle element. We repeat this process until we find the search element in the list or until we left with a sub- list of only one element. And if that element also doesn't match with the search element, then the result is "Element not found in the list".

By implementing and observing the interplay between sorting and binary search, this lab aims to highlight the importance of a sorted dataset for efficient binary search operations.

# Algorithm :

1. **PreCondition -** Sort the given input array using any sorting algorithm as counting sort**,** quick sort or merge sort

# Binary Search algorithm -



**Result**:

#include <stdio.h>  
#include <stdbool.h>  
  
void swap(int\* a, int\* b) {  
    int temp = \*a;  
    \*a = \*b;  
    \*b = temp;  
}  
  
void bubbleSort(int arr[], int n) {  
    int i, j;  
    bool swapped;  
  
    for (i = 0; i < n - 1; i++) {  
        swapped = false;  
        for (j = 0; j < n - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                swap(&arr[j], &arr[j + 1]);  
                swapped = true;  
            }  
        }  
        if (!swapped)  
            break;  
    }  
}  
  
void printArray(int arr[], int size) {  
    int i;  
    for (i = 0; i < size; i++)  
        printf("%d ", arr[i]);  
}  
  
int binarySearch(int arr[], int left, int right, int target) {  
    while (left <= right) {  
        int mid = left + (right - left) / 2;  
  
        if (arr[mid] == target) {  
            return mid;  
        }  
  
        if (arr[mid] < target) {  
            left = mid + 1;  
        } else {  
            right = mid - 1;  
        }  
    }  
  
    return -1; // Element not found  
}

int main() {

int arr[] = {77, 45, 32, 16, 8, 4, 99};

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

printArray(arr, n);

int target = 32;

int result = binarySearch(arr, 0, n - 1, target);

if (result != -1) {

printf("\nElement %d found at index %d\n", target, result);

} else {

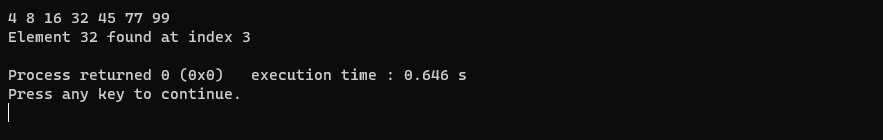
printf("\nElement %d not found in the array\n", target);

}

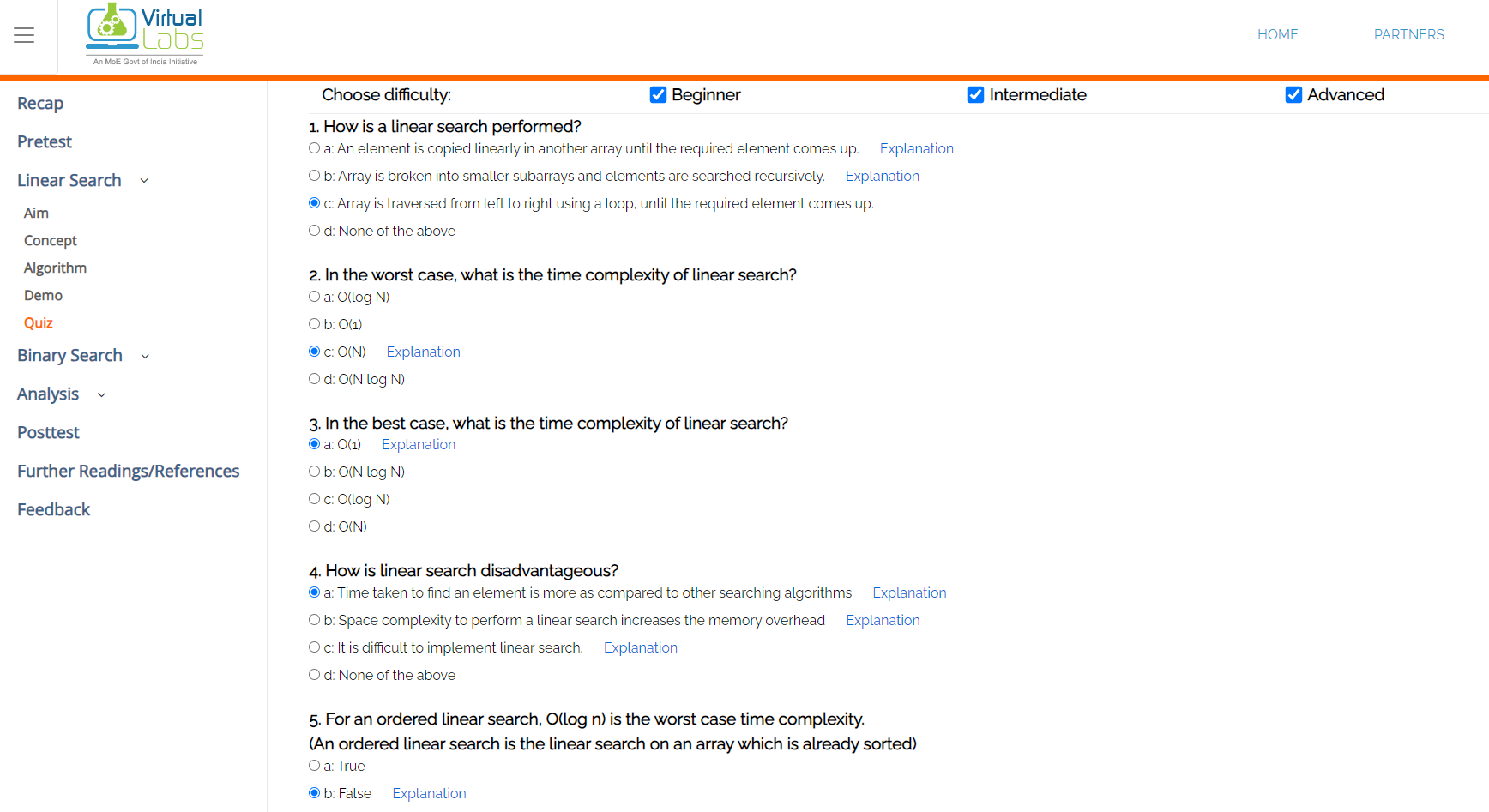
return 0;

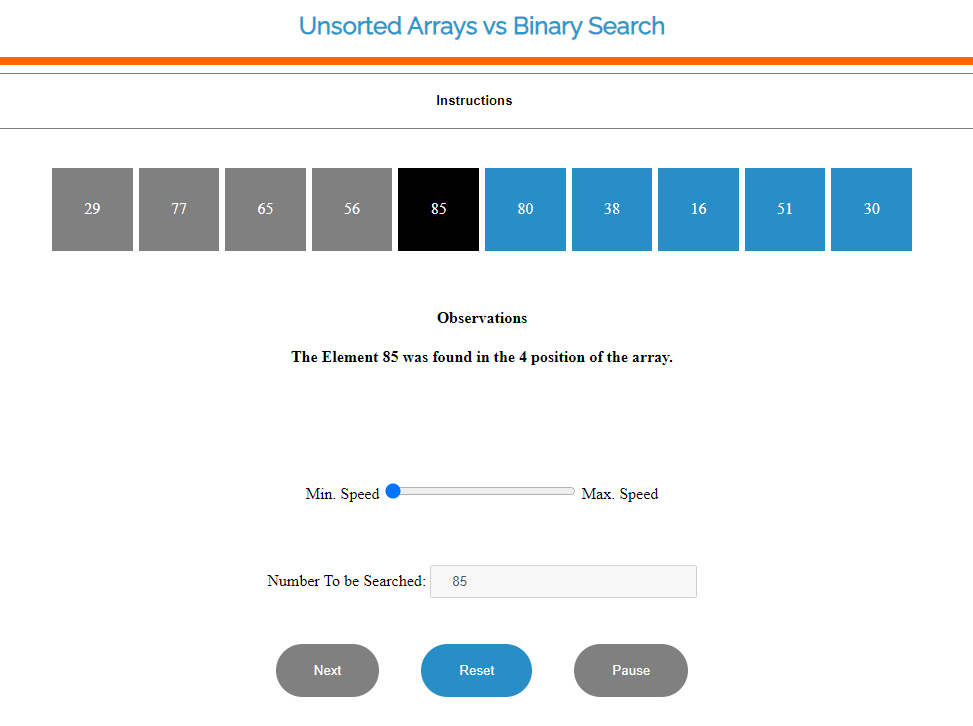
}

**Output:**

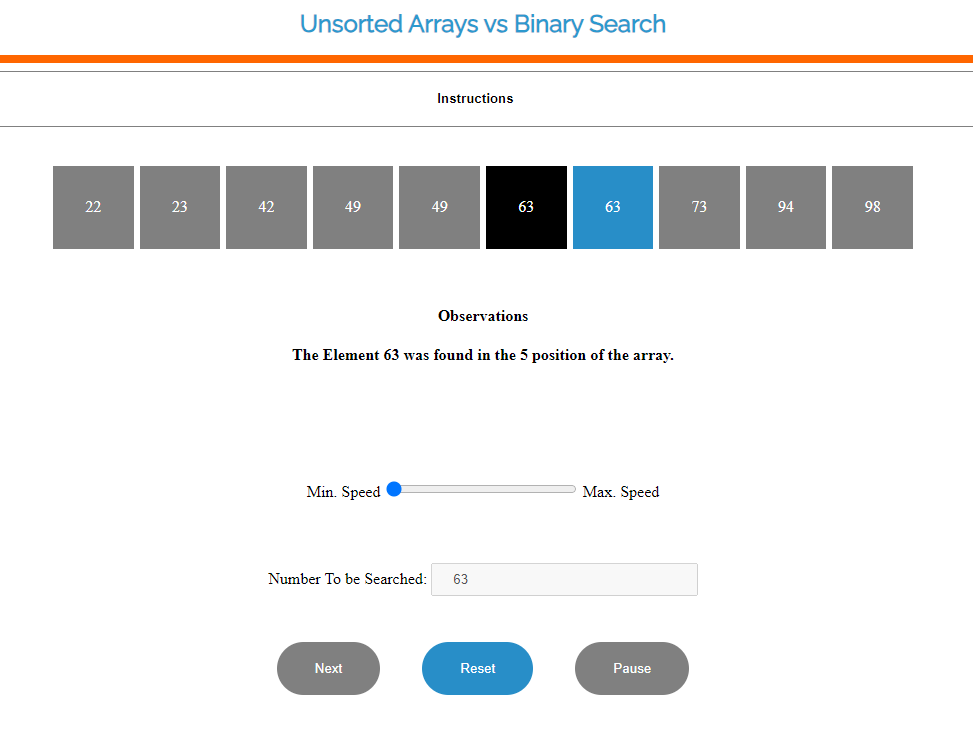


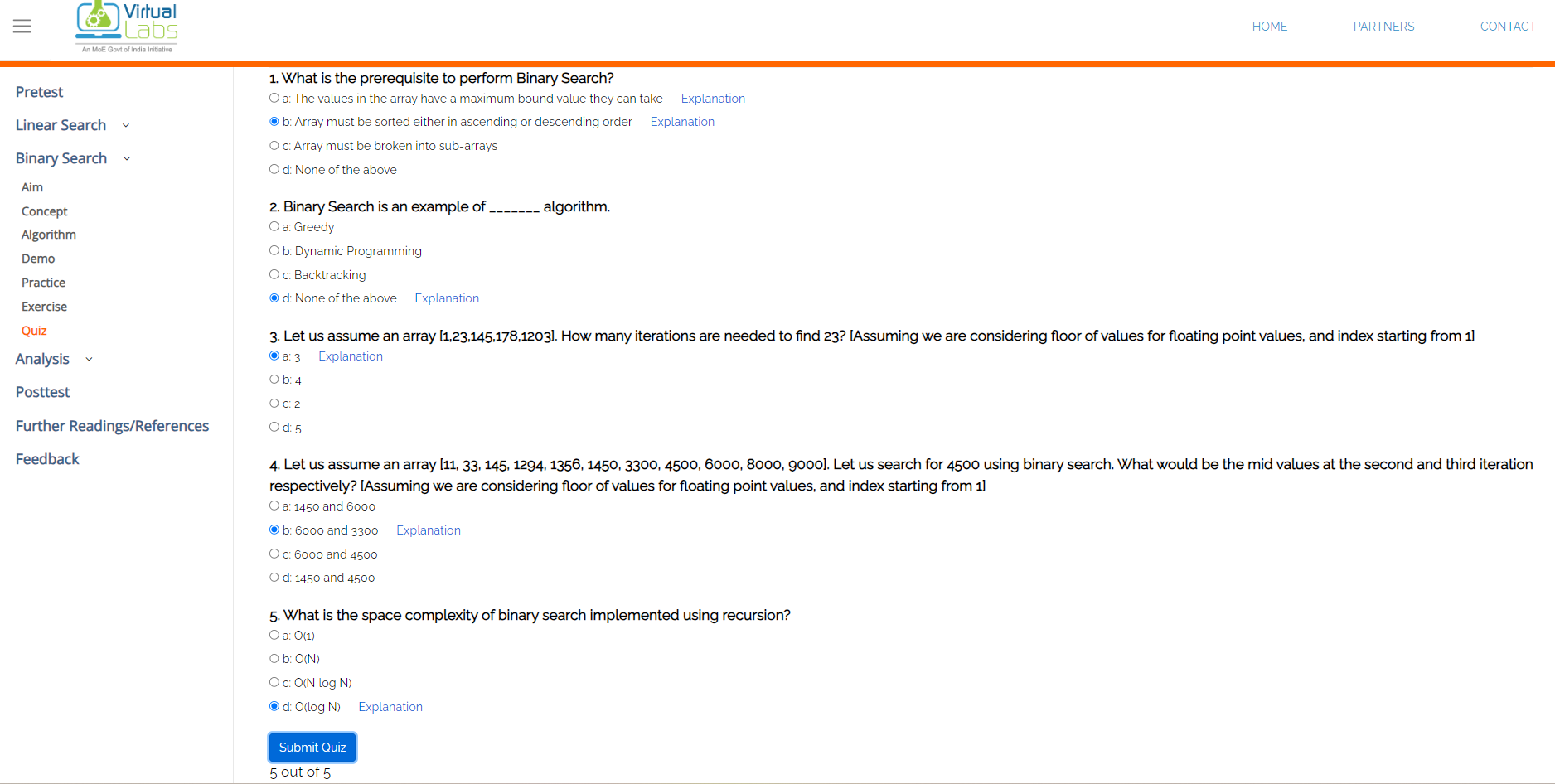
**Linear Search:**



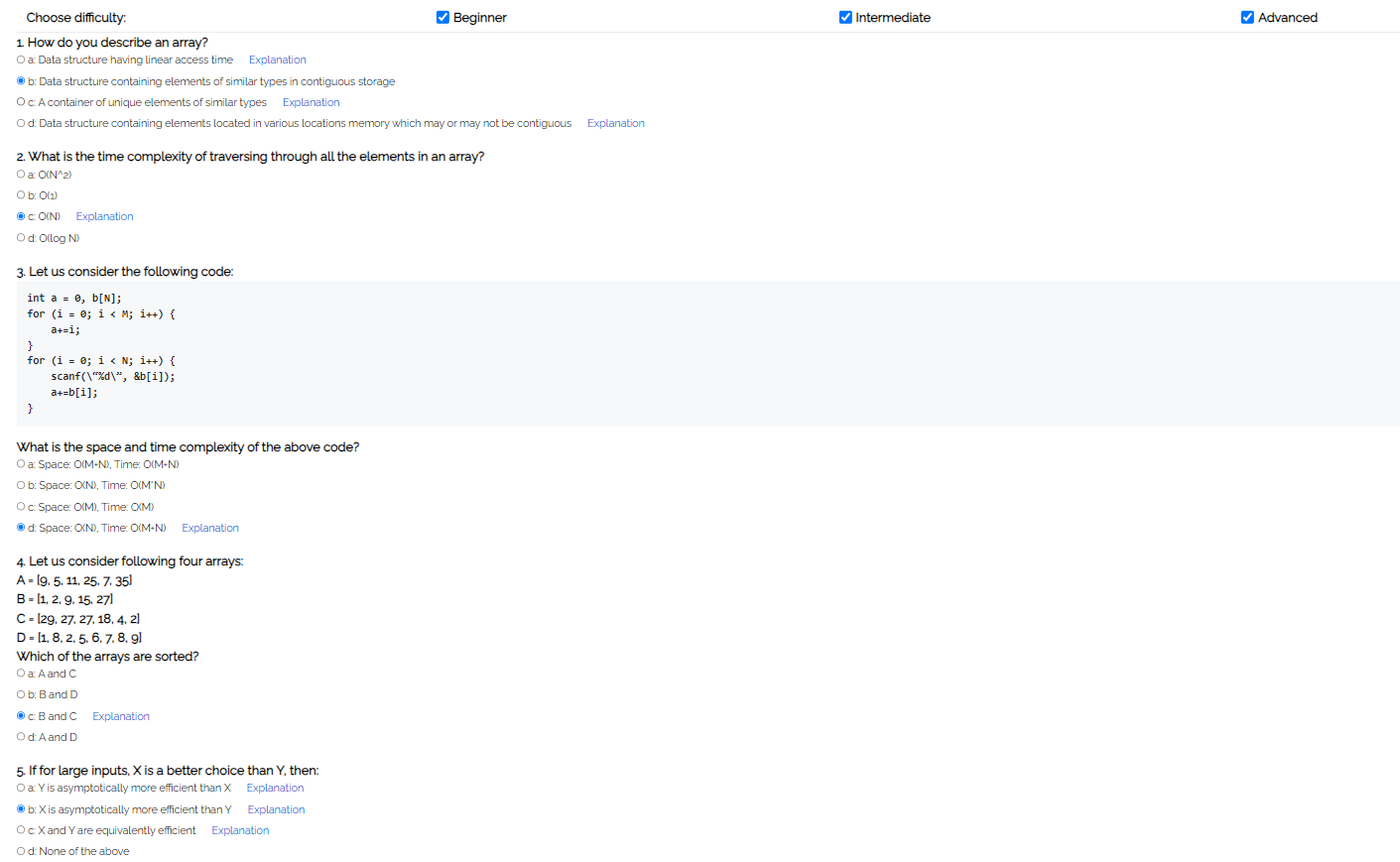


**Binary Search:**

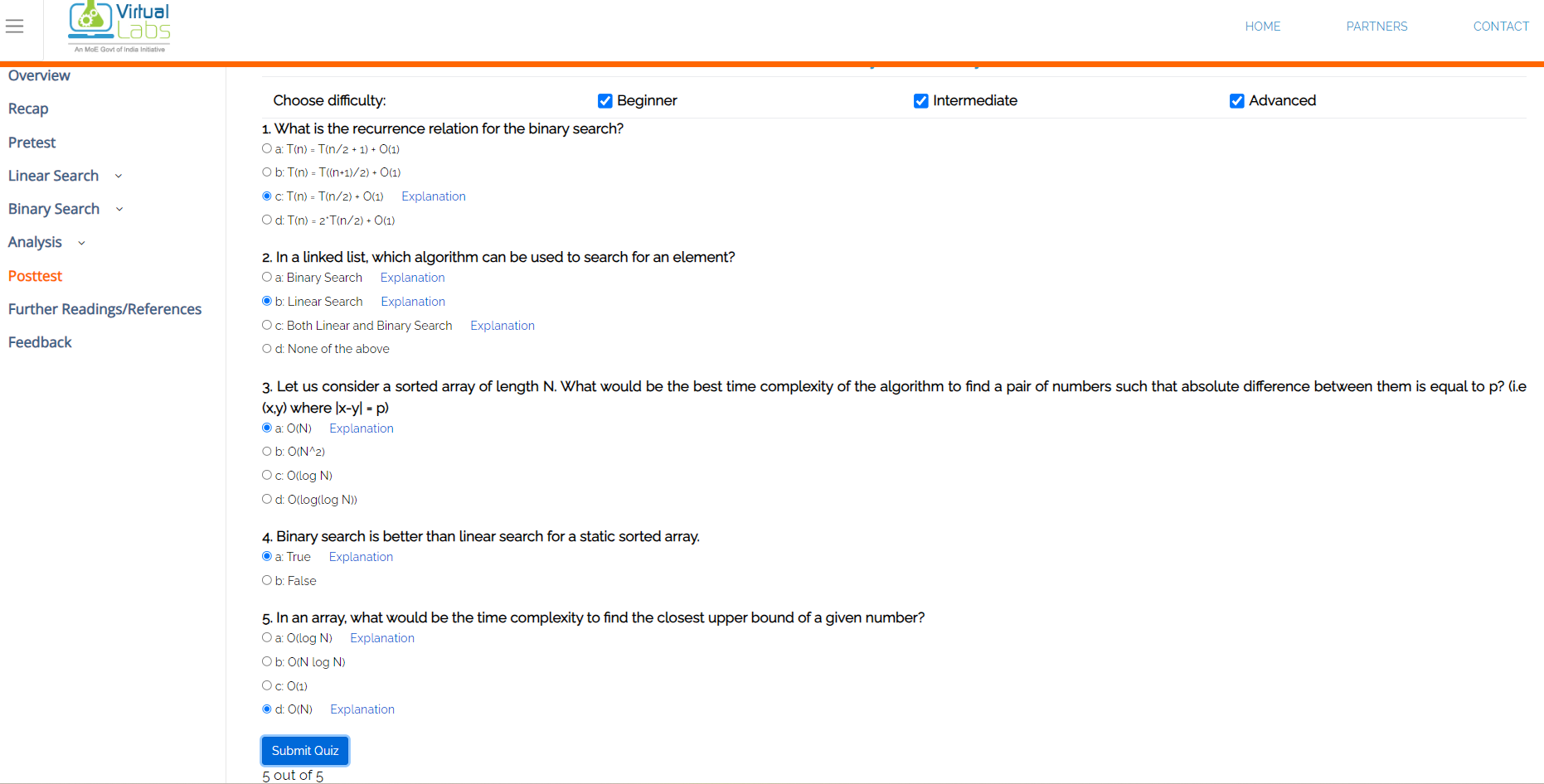




**Pretest:**



**Posttest:**

****

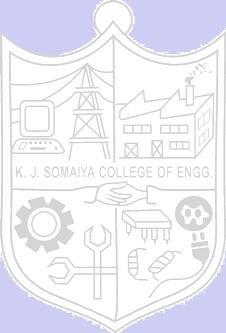
Activity:

1. Write C program to demonstrate the use of Sorting in binary search algorithm.

2. Demonstrate sorting and searching algorithms in Virtual Lab (screen shots of simulation result and quiz)

**Link: https://ds1-iiith.vlabs.ac.in/exp/unsorted-arrays/linear-search/linear-search-algorithm.html**



**References:**

**Books/ Journals/ Websites:**

* Y. Langsam, M. Augenstin and A. Tenenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.
* Vlabs on binary search and counting sort.