Symbiosis Institute of Technology

Faculty of Engineering

CSE- Academic Year 2023-24

Data Structures – Lab   Batch 2022-26

Lab Assignment No:- 1,2,3	
Name of Student	Mihir Yogesh Hebalkar
PRN No.	22070122120
Batch	2022-2026
Class	CS B-2
Academic Year & Semester	A.Y 2023-24 3rd Sem
Date of Submission	28/08/2023
Title of Assignment:	A. Implement following searching algorithm: Linear search with multiple occurrences B. Implement following searching algorithms in menu: 1. Binary search with iteration 2. Binary search with recursion
Theory:	1. Prepare table for following searching and sorting algorithms for their best case, average case and worst case time complexities. Linear search, binary search, bubble sort, Insertion sort, selection sort, merge sort, quick sort.

Algorithm	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(N)	O(N)
Binary Search	O(1)	O(log(N))	O(log(N))
Bubble Sort	O(N)	O(N^2)	O(N^2)

Insertion Sort	O(N)	O(N^2)	O(N^2)
Selection Sort	O(N^2)	O(N^2)	O(N^2)
Quick Sort	O(Nlog(N))	O(Nlog(N))	O(N^2)
Merge Sort	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))

2. Discuss on Best case and Worst case time complexities of
   Linear search, binary search, bubble sort, Insertion sort, selection sort, merge sort, quick sort.

Time Complexity analysis:

A. Linear Search

1. Best Case(O(1)) : When the element to be found is the first element of the array. Then the program is executed only once.

2. Worst Case(O(N)) : If the element to be found is the last element of the array or not present in the array at all.

B. Binary Search

1. Best Case(O(1)) : The element to be found is the middle element of the array. Then the program is executed only once.

2. Worst Case(O(log(N))) : If the target element is not found, the program keeps executing i.e dividing the array into half until low>high.

C. Bubble Sort

1. Best Case(O(N)) : When the array is sorted, only one pass is needed and no swapping is required.

2. Worst Case(O(N^2)) : When the array is sorted in reverse order from what is required and requires n passes(maximum).

D. Insertion Sort

1. Best Case(O(N)) : When the array is already sorted, requiring no swaps.

2. Worst Case(O(N^2)) : When the array is sorted in reverse order. Then n passes are executed with n swaps.

E. Selection Sort

1. Best Case and Worst Case (O(N^2)) : Irrespective of the input array's order, selection sort still makes the same number of swaps and passes for each element in the array.

	F. Quick Sort 1. Best Case(O(Nlog(N))) : If the pivot element is the middle element of the array. 2. Worst Case(O(N^2)) : If the pivot element is already sorted i.e if the pivot is the first element is already the smallest or largest element. **G. Merge Sort** 1. Best Case and Worst Case (O(N^2)) : This algorithm splits the array into half until it cannot be divided, then merges and sorts them in given order.

Source Code/Algorithm/Flow Chart:

A. Multiple Occurrence Linear Search

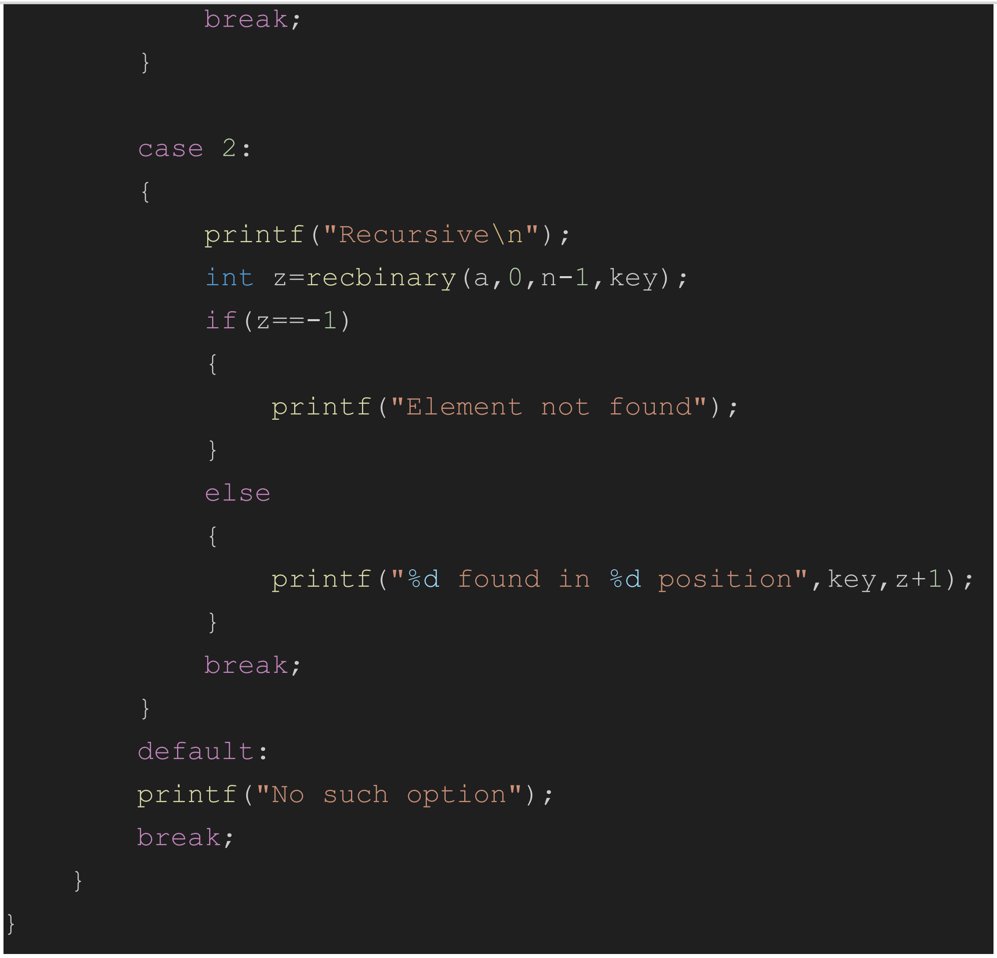
Code:

```
//Multiple occurrences
#include <stdio.h>

void lsmul(int a[],int n, int key)
{
    int i,c=0;
    printf("\nFound at \n");
    for(i=0;i<n;i++)
    {
        if(a[i]==key)
        {
            printf("Position : %d\n",i+1);
            c++;
        }
    }
    if(c==0)
    {
        printf("\nElement not present in the array");
    }
    else
    {
        printf("\n%d has occurred %d times.",key,c);
    }


}
int main()
{
    int a[100],n,key,i;
    printf("Enter number of elements : ");
    scanf("%d",&n);
    printf("Enter elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Enter element to find : ");
    scanf("%d",&key);
    lsmul(a,n,key);

    return 0;
}
```

B. Binary Search with Menu

Code:

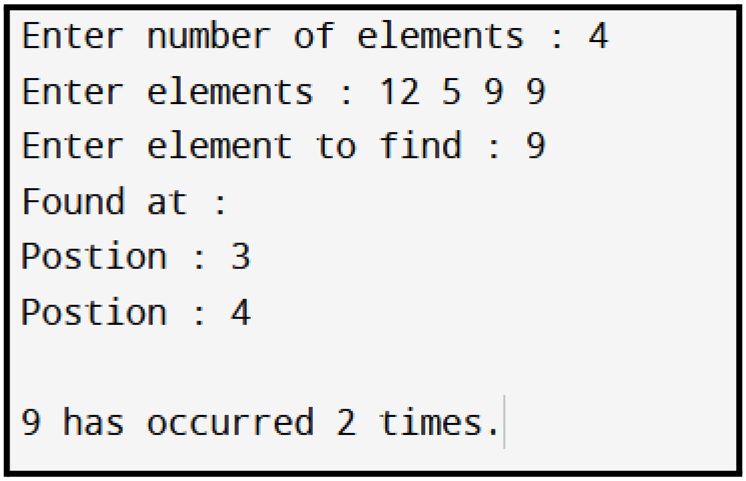
```
#include<stdio.h>
#include<stdlib.h>

//Bubble Sort function
void sort(int l,int a[])
{
    int i,j;
    for(i=0;i<l-1;i++)
    {
        for(j=0;j<l;j++)
        {
            if(a[j]>a[j+1])
            {
                int temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
}

//Iterative Binary Sort
int binsearch(int l,int a[],int num)
{
    int s=0,d=0;
    int mid=(s+l)/2;
    while(s<l)
    {
        if(num>a[mid])
        {
            s=mid+1;
            mid=(s+l)/2;
        }
        else if(num<a[mid])
        {
            l=mid;
            mid=(s+l)/2;
        }
        else if(num==a[mid])
        {
            d++;
            return mid;
            break;
        }
    }
    if(d==0)
    {
        return 0;
    }
}


//Recursive Binary Sort
int recbinary(int arr[],int l,int h,int key)
{
    if(l<h)
    {
        int mid;
        mid=(l+h)/2;
        if(arr[mid]<key)
        {
            return recbinary(arr,l,mid-1,key);
        }
        if(arr[mid]>key)
        {
            return recbinary(arr,mid+1,h,key);
        }
        if(arr[mid]==key)
        {
            return mid;
        }
    }
    else{
        return -1;
    }


}

//Driver Code
void main()
{
    int i,n,a[50],key,option;
    printf("Enter number of elements : ");
    scanf("%d",&n);
    printf("Enter elements of array: ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Enter element to be found : ");
    scanf("%d",&key);
    sort(n,a);
    printf("\nBinary Search\nEnter 1: Iterative\nEnter 2: Recursive\n");
    printf("Enter your option : ");
    scanf("%d",&option);
    switch(option)
    {
        case 1:
        {
            printf("Iterative\n");
            int z=binsearch(n-1,a,key);
            if(z==0)
            {
                printf("Element not found ");
            }
            else
            {
                printf("%d found in %d position",key,z+1);
            }
```



```
            break;
        }

        case 2:
        {
            printf("Recursive\n");
            int z=recbinary(a,0,n-1,key);
            if(z==-1)
            {
                printf("Element not found");
            }
            else
            {
                printf("%d found in %d position",key,z+1);
            }
            break;
        }
        default:
        printf("No such option");
        break;
    }
}
```

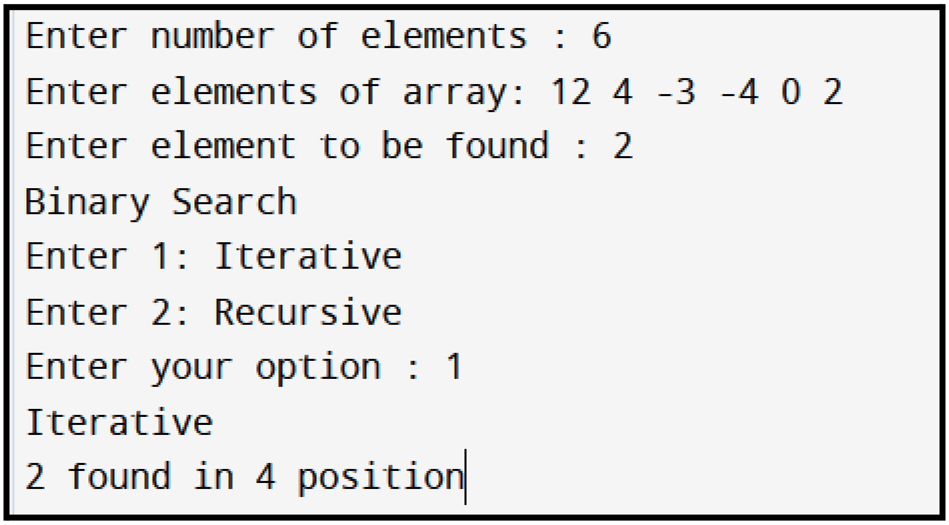
Output Screenshots (if applicable)

A. Linear Search



```
Enter number of elements : 4
Enter elements : 12 5 9 9
Enter element to find : 9
Found at :
Postion : 3
Postion : 4

9 has occurred 2 times.
```

B. Binary Search



```
Enter number of elements : 6
Enter elements of array: 12 4 -3 -4 0 2
Enter element to be found : 2
Binary Search
Enter 1: Iterative
Enter 2: Recursive
Enter your option : 1
Iterative
2 found in 4 position
```

Conclusion	Thus we have studied different sorting algorithms and their time complexities.