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	Onkar Mendhapurkar
PRN No.	22070122135
Batch	2022-2026
Class	CSE-B2
Academic Year & Semester	2023-2024 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	Best Case
Linear Search	O(1)	O(N)	O(N)
Binary Search	O(1)	O(logN)	O(logN)
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)
Merge Sort	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))
Quick Sort	O(Nlog(N))	O(Nlog(N))	O(N^2)

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.

   a. Linear Search
      - Best Case Time Complexity: Best Case scenario appears when the element being searched for is found in the first position of the list, eg: {5,1,3,2,8} if the target element is 5, Then the time complexity will be O(1), Algorithm will terminate in first comparison.
      - Worst Case Time Complexity: Worst Case Scenario Appears when the element to be found is present at last position of the list or is not present in the list. Eg: {5,1,3,2,8} if the search element is 8 or it is 9, Then this time complexity will be O(N), Algorithm will terminate in N comparisons.

   b. Binary Search
      - Best Case Time Complexity: The Target element is found at the middle of the array in the first comparison. Time complexity will be O(1).
      - Worst Case Time Complexity: The Target element is not present in the array then the algorithm needs to keep dividing the search algo needs to keep dividing the search in half until the search has a empty element in middle. The time complexity will be O(logN).

   c. Bubble Sort
      - Best Case Time Complexity: Best case is when the array is already sorted, the algo only needs to make a single pass to determine that no swaps are necessary. The time complexity is O(N).

- Worst Case Time Complexity: Worst Case is when the array is sorted in reverse order, hence it requires the maximum number of swaps. The time complexity then changes to O(logN).

d. Insertion Sort:
   - Best Case Time Complexity: Best case is when the array is when the input array is already sorted, Hence making minimal shifts during procedure, then the time complexity becomes O(N).
   - Worst Case Time Complexity: It occurs when the input array is sorted in reverse order, Hence the position of the element has to be shifted n^2 times, The time complexity becomes O(N^2).

e. Selection Sort:
   - Best Case Time Complexity: In the best-case scenario, the input array is already sorted. However, regardless of the input order, Selection Sort still needs to perform the same number of comparisons and swaps for each element in the array. Time complexity then becomes O(N^2).
   - Worst Case Time Complexity: In the worst-case scenario, the input array is in reverse order. In each iteration, the algorithm needs to find the minimum element in the remaining unsorted portion of the array, which requires (n – i) comparisons, The time complexity becomes O(N^2).

f. Merge Sort:
   - Best Case Time Complexity: In the best case scenario, Merge Sort still needs to divide the array into sub-arrays and merge them, just like in the worst-case scenario. As a result, the best-case time complexity of Merge Sort remains O(Nlog(N)).
   - Worst Case Time Complexity: The worst-case time complexity of Merge Sort is O(Nlog(N)). This worst-case scenario occurs when the input array needs to be fully divided and merged at each level of recursion.

g. Quick Sort:
   - Best Case Time Complexity: the pivot chosen happens to be the median element of the array. This means that during each partitioning step, the array gets evenly divided into two parts. Hence Time complexity becomes O(Nlog(N)).
   - Worst Case Time Complexity: The worst case scenario occurs when the pivot chosen is consistently the smallest or largest element in the array, causing an imbalanced partition. Time Complexity becomes O(N^2).

Source Code/Algorithm/Flow Chart:

A. Linear Search With Multiple Occurences

- Source Code

```c
#include <stdio.h>

void LS(int arr[],int size,int target){
    int i,occurrences=0;
    for (i=0;i<size;i++){
        if (arr[i] == target) {
            printf("Found at index %d\n",i);
            occurrences++;
        }
    }
    if (occurrences==0){
        printf("Element not found in the array.\n");
    }
    else{
        printf("Total occurrences: %d\n",occurrences);
    }
}

int main(){
    int arr[]={5,4,2,3,4,6,3,78,11,3};
    int size=sizeof(arr)/sizeof(arr[0]);
    int target=3;
    LS(arr,size,target);
    return 0;
}
```

B. Binary Search

1. Binary Search Using Iteration: Source Code:

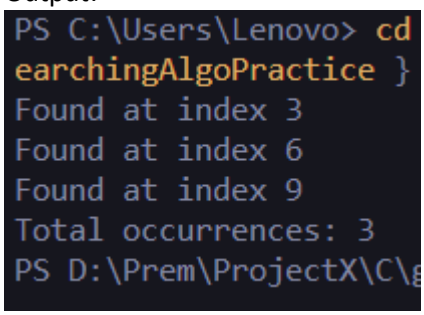
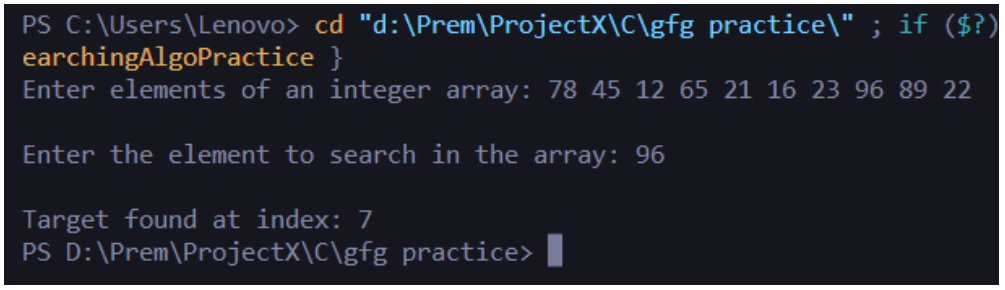
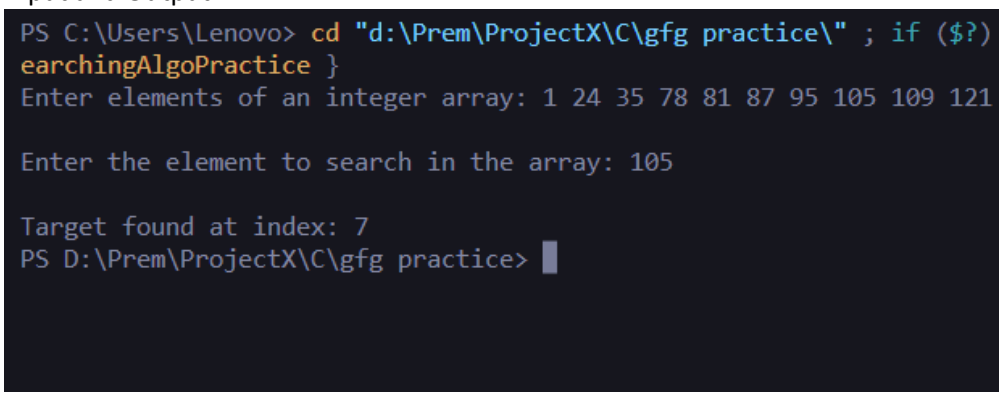
```c
#include <stdio.h>

int binarySearch(int arr[],int target){
    int left = 0, right = 9,mid;
    while(left<=right){
        mid = (left+right)/2;
        if(arr[mid]==target){
            return mid;
        }
        else if(arr[mid]>target){
            right = mid-1;
        }
        else{
```

```
            left = mid+1;
        }
    }
    return -1;
}

int main() {
    int arr[10],i,target,num;
    printf("Enter elements of an integer array: ");
    for(i=0;i<10;i++){
        scanf("%d",&arr[i]);
    }
    printf("\nEnter the element to search in the array: ");
    scanf("%d",&target);

    if(binarySearch(arr,target)==-1){
        printf("\nTarget not found!");
    }
    else{
        printf("\nTarget found at index:
%d",binarySearch(arr,target));
    }
    return 0;
}
```

2. Binary Search Using Recursion: Source Code:

```
#include <stdio.h>

int recursiveBinSearch(int arr[],int low,int high,int target){
    int mid;
    while(low<=high){
        mid=(low+high)/2;
        if(arr[mid]==target){
            return mid;
        }
        else if(target>arr[mid]){
            return recursiveBinSearch(arr,mid+1,high,target);
        }
        else{
            return recursiveBinSearch(arr,low,mid-1,target);
        }
    }
    return -1;
}
```

```c
int main() {
    int arr[10],i,target,num;
    printf("Enter elements of an integer array: ");
    for(i=0;i<10;i++){
        scanf("%d",&arr[i]);
    }
    printf("\nEnter the element to search in the array: ");
    scanf("%d",&target);
    if(recursiveBinSearch(arr,0,9,target)==-1){
        printf("\nTarget not found!");
    }
    else{
        printf("\nTarget found at index:
%d",recursiveBinSearch(arr,0,9,target));
    }
    return 0;
}
```

Output Screenshots (if applicable)	A. Linear Search With Multiple Occurences. • Output:  B. Binary Search 1. Binary Search Using Iteration. • Input and Output:  2. Binary Search Using Recursion. • Input and Output: 
Conclusion	Thus we have studied different sorting algorithms and their time complexities.