



# Symbiosis Institute of Technology

## Faculty of Engineering

### CSE- Academic Year 2023-24

### Data Structures – Lab Batch 2022-26

#### Lab Assignment No:- 1,2,3

Lab Assignment No:- 1,2,3					
Name of Student		Ayushi Kapgate			
PRN No.		22070122093			
Batch		2022-2026			
Class		CS-B1			
Academic Year & Semester		2023, SEM-III			
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	Avg 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)
		Merge Sort	O(n log n)	O(n log n)	O(n log n)
		Quick Sort	O(n log n)	O(n log n)	O(n^2)

	<p>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.</p> <p><b><u>Linear Search:</u></b>  <b>Best Case:</b> <math>O(1)</math> - The element being searched for is found at the beginning of the array.  <b>Worst Case:</b> <math>O(n)</math> - The element being searched for is found at the end of the array or is not present at all. In the worst case, every element in the array needs to be checked.</p> <p><b><u>Binary Search:</u></b>  <b>Best Case:</b> <math>O(1)</math> - The element being searched for is found at the middle of the sorted array.  <b>Worst Case:</b> <math>O(\log n)</math> - The element being searched for is not present, and the search reduces the search range by half in each step.</p> <p><b><u>Bubble Sort:</u></b>  <b>Best Case:</b> <math>O(n)</math> - The array is already sorted, and no swaps are needed.  <b>Worst Case:</b> <math>O(n^2)</math> - The array is sorted in reverse order, and each element needs to be swapped with every other element.</p> <p><b><u>Insertion Sort:</u></b>  <b>Best Case:</b> <math>O(n)</math> - The array is already sorted, and each new element can be inserted in constant time.  <b>Worst Case:</b> <math>O(n^2)</math> - The array is sorted in reverse order, and each new element requires shifting all previous elements.</p> <p><b><u>Selection Sort:</u></b>  <b>Best Case:</b> <math>O(n^2)</math> - The algorithm doesn't have any optimization for the initial ordering of the array.  <b>Worst Case:</b> <math>O(n^2)</math> - Regardless of the input ordering, the algorithm requires <math>n</math> iterations for finding the minimum element in each pass.</p> <p><b><u>Merge Sort:</u></b>  <b>Best Case:</b> <math>O(n \log n)</math> - The array is divided evenly at each step, leading to balanced merging.  <b>Worst Case:</b> <math>O(n \log n)</math> - The array is divided unevenly at each step, still leading to efficient merging due to divide-and-conquer.</p> <p><b><u>Quick Sort:</u></b>  <b>Best Case:</b> <math>O(n \log n)</math> - The pivot chosen at each step divides the array evenly.  <b>Worst Case:</b> <math>O(n^2)</math> - The pivot chosen is the smallest or largest element in each step, leading to uneven partitioning.</p>
<p><b>Source Code/Algorithm/Flow Chart:</b></p>	<p>1. Binary search with iteration:</p>

```

main.c
1  #include<stdio.h>
2  int bsi(int arr[],int n,int key){
3      int l=0;
4      int r=n-1;
5      while(l<=r){
6          int mid=l+(r-l)/2;
7          if(arr[mid]==key){
8              return mid;
9          }
10         else if(arr[mid]<key){
11             l=mid+1;
12         }
13         else{
14             r=mid-1;
15         }
16     }
17     return -1;
18 }
19 int main(){
20     int arr[]={1,2,3,4,5,6,7,8,9};
21     int n=sizeof(arr)/sizeof(arr[0]);
22     int key=6;
23     int f=bsi(arr,n,key);
24     if(f!=-1){
25         printf("element found at %d",f);
26     }
27     else{
28         printf("element not found.");
29     }
30 }
31

```

for [www.onlinegdb.com...](http://www.onlinegdb.com...)

## 2. Binary search with recursion:

```

1  #include<stdio.h>
2  int bsr(int arr[],int l,int r,int key){
3      if(l<=r){
4          int mid=l+(r-l)/2;
5          if(arr[mid]==key){
6              return mid;
7          }
8          else if(arr[mid]<key){
9              return bsr(arr,mid+1,r,key);
10         }
11         else{
12             return bsr(arr,l,mid-1,key);
13         }
14     }
15     return -1;
16 }
17 int main(){
18     int arr[]={1,2,3,4,5,6,7,8,9};
19     int n=sizeof(arr)/sizeof(arr[0]);
20     int key=3;
21     int f=bsr(arr,0,n-1,key);
22     if(f!=-1){
23         printf("element found at %d",f);
24     }
25     else{
26         printf("element not found.");
27     }
28 }
29

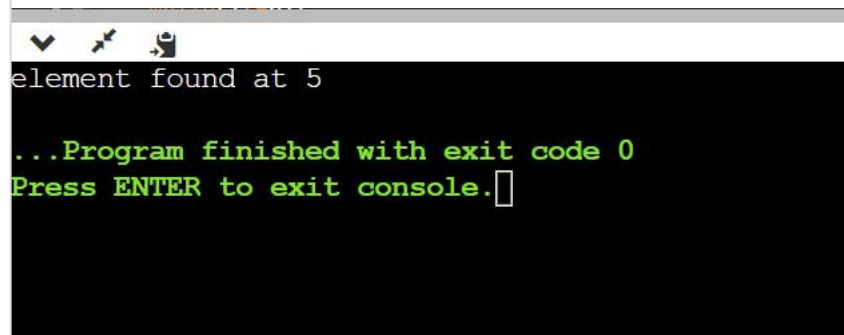
```

### 3. Linear search with multiple occurrences:

```
1  #include <stdio.h>
2  void linearSearchMultiple(int arr[], int size, int target) {
3      int occurrences = 0;
4
5      for (int i = 0; i < size; i++) {
6          if (arr[i] == target) {
7              printf("Found %d at index %d\n", target, i);
8              occurrences++;
9          }
10     }
11
12     if (occurrences == 0) {
13         printf("%d not found in the array.\n", target);
14     } else {
15         printf("%d appeared %d times in the array.\n", target, occurrences);
16     }
17 }
18
19 int main() {
20     int arr[] = {4, 2, 7, 2, 8, 2, 1};
21     int size = sizeof(arr) / sizeof(arr[0]);
22     int target = 2;
23
24     linearSearchMultiple(arr, size, target);
25
26     return 0;
27 }
28
```

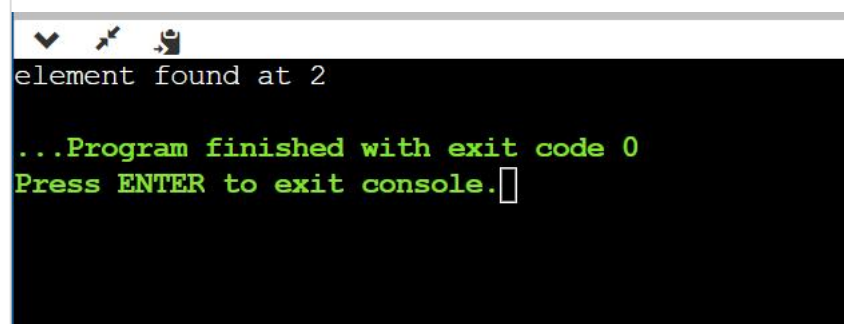
#### Output Screenshots (if applicable)

##### 1. Binary search with iteration:



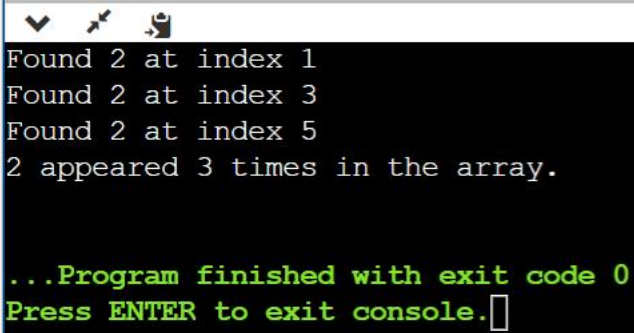
```
element found at 5
...Program finished with exit code 0
Press ENTER to exit console.
```

##### 2. Binary search with recursion:



```
element found at 2
...Program finished with exit code 0
Press ENTER to exit console.
```

3. Linear search with multiple occurrences:



```
Found 2 at index 1
Found 2 at index 3
Found 2 at index 5
2 appeared 3 times in the array.

...Program finished with exit code 0
Press ENTER to exit console.
```

**Conclusion**

Thus we have studied different sorting algorithms and their time complexities