

**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** | Janmejay Pandya |
| **PRN No.** | 22070122086 |
| **Batch** | 22-26 |
| **Class** | CS-B |
| **Academic Year & Semester** | 2023 semester 3 |
| **Date of Submission** | 29-8-23 |
|  | |
| **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.   1. Discuss on Best case and Worst case time complexities of   Linear search, binary search, bubble sort, Insertion sort, selection sort, merge sort, quick sort.      2.  **Linear Search:**  Best Case: O(1)  The best case occurs when the element being searched for is found in the very first comparison.  Worst Case: O(n)  The worst case happens when the element being searched for is either at the end of the list, or it's not present at all. In such cases, you need to examine every element.  **Binary Search:**  Best Case: O(1)  The best case occurs when the element being searched for is found at the middle of the sorted array.  Worst Case: O(log n)  The worst case happens when the element is not present, or it's at one of the extreme ends. In each comparison, the search space is reduced by half.  **Bubble Sort:**  Best Case: O(n)  The best case occurs when the input array is already sorted. In this case, the algorithm performs only one pass through the array to confirm that no swaps were needed.  Worst Case: O(n^2)  The worst case happens when the input array is sorted in reverse order. In each pass, it needs to swap elements throughout the entire array.  **Insertion Sort:**  Best Case: O(n)  The best case occurs when the input array is already sorted. In this case, the algorithm can simply iterate through the array without making any swaps.  Worst Case: O(n^2)  The worst case occurs when the input array is sorted in reverse order. In each step, it needs to compare and shift elements, leading to a quadratic time complexity.  **Selection Sort:**  Best Case: O(n^2)  The best case is the same as the worst case for selection sort, as it always needs to search for the minimum element, even if the array is partially sorted.  Worst Case: O(n^2)  The worst case occurs when the input array is sorted in reverse order. In each pass, it needs to search for the minimum element and swap it with the first unsorted element.  **Merge Sort:** Best Case: O(n log n)  The best case is the same as the average and worst cases for merge sort because it always divides the array into halves and then merges them. It consistently maintains a time complexity of O(n log n).  **Quick Sort:**  Best Case: O(n log n)  The best case occurs when the pivot choice consistently divides the list into nearly equal halves.  Worst Case: O(n^2)  The worst case occurs when the pivot choice is poor and consistently divides the list into highly imbalanced sublists. In this case, it degrades to a quadratic time complexity. |
| **Source Code/Algorithm/Flow Chart:** | A.      B. |
| **Output Screenshots (if applicable)** |  |
| **Conclusion** | Thus we have studied different sorting algorithms and their time complexities. |