

**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** | KRISH PANCHAL |
| **PRN No.** | 22070122102 |
| **Batch** | B1 (2022-26) |
| **Class** | CS-B1 |
| **Academic Year & Semester** | 2nd Year | 3rd Semester |
| **Date of Submission** | 27/08/2023 |
|  | |
| **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.     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.   1. **Linear Search**:    * Best Case: O(1) - This occurs when the element you are searching for is found at the very beginning of the list.    * Worst Case: O(n) - This occurs when the element is either at the end of the list or not in the list at all, and you have to traverse the entire list. 2. **Binary Search**:    * Best Case: O(1) - This occurs when the element you are searching for is in the middle of the sorted array.    * Worst Case: O(log n) - This occurs when you have to keep dividing the array in half until you find the element or determine it's not in the array. 3. **Bubble Sort**:    * Best Case: O(n) - This occurs when the input array is already sorted, and no swaps are needed.    * Worst Case: O(n^2) - This occurs when the input array is sorted in reverse order, and you need to perform the maximum number of swaps. 4. **Insertion Sort**:    * Best Case: O(n) - This occurs when the input array is already nearly sorted.    * Worst Case: O(n^2) - This occurs when the input array is sorted in reverse order, and you need to perform the maximum number of comparisons and shifts. 5. **Selection Sort**:    * Best Case: O(n^2) - The best case and worst case are the same for selection sort because it always has to find the minimum element.    * Worst Case: O(n^2) - This occurs when the input array is sorted in any order, as you still need to make the same number of comparisons. 6. **Merge Sort**:    * Best Case: O(n log n) - Merge sort has a consistent time complexity. It divides the input into halves and then merges them, leading to a time complexity of O(n log n) in all cases.    * Worst Case: O(n log n) - Same as the best case, merge sort's divide and conquer strategy ensures a consistent time complexity. 7. **Quick Sort**:    * Best Case: O(n log n) - Quick sort performs well when the pivot choice consistently splits the array into roughly equal halves.    * Worst Case: O(n^2) - This can happen when the pivot is consistently chosen as the smallest or largest element, leading to unbalanced partitions. However, good pivot selection techniques (like median of three) usually avoid this. |
| **Source Code/Algorithm/Flow Chart:** | Linear Search with multiple occurrences:    1. Binary Search with iteration and recursion in menu driven code: |
| **Output Screenshots (if applicable)** | Linear Search with multiple occurrences: 1.Element not found:    2.Element found 1 time:    3.Element found 2 times:       1. Binary Search with iteration and recursion in menu driven code:   1. Using Iteration:    2. Using Recursion: |
| **Conclusion** | Thus we have studied different sorting algorithms and their time complexities. |