

**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** | Pranay Vasoya |
| **PRN No.** | 22070122154 |
| **Batch** | 2022-26 |
| **Class** | CS-B3 |
| **Academic Year & Semester** | A.Y 2023-24  3rd SEM |
| **Date of Submission** | 3/09/2023 |
|  | |
| **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) | | Selection Sort | O(N^2) | O(N^2) | O(N^2) | | Insertion Sort | O(N) | 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) |  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.  Time Complexity analysis:   * **Linear Search**  1. Best Case O(1) : The element being searched for is found at the very beginning of the list, resulting in a constant time complexity. 2. Worst Case O(N): The element might be at the end of the list, requiring linear search through the entire list.  * **Binary Search**  1. Best Case O(1): It occurs when the target element is found in the middle of the array on the first comparison. 2. Worst Case O(log(N)): It occurs when the target element is not present in the array, and the algorithm must continue dividing the array until it concludes that the element is not there.  * **Bubble Sort**  1. Best Case O(N): When the array is already sorted. 2. Worst Case O(N^2):It occurs when given array is in reverse order  * **Selection Sort**  1. Best Case O(N^2):It occurs when the input list is already sorted in ascending order 2. Worst Case O(N^2):It occurs when the input list is sorted in descending order or in any other order that requires maximum work to sort  * **Insertion Sort**   1.Best Case O(N):It occurs when array is already sorted and no swaps are needed.  2. Worst Case O(N^2):When array is sorted in reverse order and needs n swaps.   * **Merge Sort**  1. Best- and worst-case O(N\*log(N)): This method divides the array into half until it cannot be divided. Then merges and sorts them.  * **Quick Sort**  1. Best Case O(N\*log(N)):If the pivot is the middle element of the array. 2. Worst Case O(N^2):If the pivot is already sorted |
| **Source Code/Algorithm/Flow Chart:** | Linear Search with multiple occurrences #include<stdio.h>  int main()  {  int a[10],n,ele,i,count=0;  printf("Enter the no of elements in array: ");  scanf("%d",&n);    for(i=0;i<n;i++)  {  printf("\nEnter value of element: ");  scanf("%d",&a[i]);  }  printf("Enter the element to be searched: ");  scanf("%d",&ele);  for(i=0;i<n;i++)  {  if(a[i]==ele)  {  printf("\nFound at index %d",i);  count++;  }  }  printf("\n");  if(count==0)  {  printf("Element %d not found",ele);  }  else  {  printf("Element %d found %d times ",ele,count);  }  return 0;  }   1. Binary Search with menu   #include<stdio.h>  #include<stdlib.h>  int binary\_recursive(int a[], int low, int high, int ele) {  if (low > high)  {  return -1;  }  int mid = (low + high) / 2;  if (a[mid] == ele) {  return mid;  } else if (a[mid] > ele) {  return binary\_recursive(a, low, mid - 1, ele);  } else {  return binary\_recursive(a, mid + 1, high, ele);  }  }  int binary\_iterative(int a[], int n, int ele) {  int low = 0;  int high = n - 1;  while (low <= high) {  int mid = (low + high) / 2;  if (a[mid] == ele) {  return mid;  } else if (a[mid] > ele) {  high = mid - 1;  } else {  low = mid + 1;  }  }  return -1;  }  int main()  {  int a[20],n,ele,i,count=0,choice;    printf("Enter the no of elements in array: ");  scanf("%d",&n);    for(i=0;i<n;i++)  {  printf("\nEnter value of element: ");  scanf("%d",&a[i]);  }    printf("\nEnter the element to be searched: ");  scanf("%d",&ele);    printf("\nEnter 1 - Iterative binary search \n Enter 2 - Recursive binary search");  printf("\nEnter your choice : ");  scanf("%d",&choice);  switch(choice)  {  case 1:  {  printf("Iterative\n");  int result=binary\_iterative(a,n-1,ele);  if(result==-1)  {  printf("Element not found ");    }  else  {  printf("Element %d found at %d index",ele,result);    }  break;    }  case 2:  {  printf("Recursive\n");  int result=binary\_recursive(a,0,n-1,ele);  if(result==-1)  {  printf("Element not found ");    }  else  {  printf("Element %d found at %d index",ele,result);    }  break;  }  default:  {  printf("Invalid choice");  }  }  return 0;  } |
| **Output Screenshots (if applicable)** | 1.      1. Binary Search Menu |
| **Conclusion** | Thus we have studied different sorting algorithms and their time complexities. |