|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory (using C/C++)** | **Semester:** | **III** |
| **Date of Performance:** | **17 / 10 / 2003** | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Prof. Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 7**

**Title: Searching Techniques**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Write a C/C++ program for linear search and binary search. |

|  |
| --- |
| **COs to be achieved:** |
| **CO4:** Demonstrate sorting and searching methods |

|  |
| --- |
| **Books/Journals/Websites referred:** |
|  |

|  |
| --- |
| **Tools required:** |
| **DEV C/C++ compiler/ Code blocks C compiler** |

|  |
| --- |
| **Theory:** |
| To search an element in a given array, there are two popular algorithms available:   1. **Linear Search:** Linear search is a very basic and simple search algorithm. In Linear search, we search an element or value in a given array by traversing the array from the starting, till the desired element or value is found. 2. **Binary Search:** Binary Search is used with sorted array or list. In binary search, we follow the following steps: 3. We start by comparing the element to be searched with the element in the middle of the list/array. 4. If we get a match, we return the index of the middle element. 5. If we do not get a match, we check whether the element to be searched is less or greater than in value than the middle element. 6. If the element/number to be searched is greater in value than the middle number, then we pick the elements on the right side of the middle element(as the list/array is sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the step 1. 7. If the element/number to be searched is lesser in value than the middle number, then we pick the elements on the left side of the middle element, and start again from the step 1.   Binary Search is useful when there are large numbers of elements in an array and they are sorted.  So a necessary condition for Binary search to work is that the list/array should be sorted. |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.** 2. For the linear search program:    1. We initialized an array and defined its size.    2. We implemented the linearSearch function that takes the array, its size, and the element to be searched as parameters.    3. Inside the linearSearch function, we used a simple loop to iterate through each element of the array and check if it matches the element we are looking for.    4. If the element is found, we return its index; otherwise, we return -1 to indicate that the element is not present in the array.    5. In the main function, we called the linearSearch function and printed the result. 3. For the binary search program: 4. We initialized an array and defined its size. 5. We implemented the binarySearch function that takes the array, left and right indices, and the element to be searched as parameters. 6. Inside the binarySearch function, we used a while loop to divide the search space in half until the element is found or the search space is exhausted. 7. If the element is found, we return its index; otherwise, we return -1 to indicate that the element is not present in the array. 8. In the main function, we called the binarySearch function and printed the result. 9. **Explain your program logic and methods used.**   For the linear search program, we used a simple iterative approach to check each element in the array until we find the desired element. This approach has a time complexity of O(n) since in the worst case we might need to iterate through all the elements.  For the binary search program, we used a divide and conquer strategy where we repeatedly divide the search space in half. This approach assumes that the array is sorted. It has a time complexity of O(log n) since at each step we reduce the search space by half.   1. **Explain the Importance of the approach followed by you.**   The linear search is a simple and straightforward approach that works for any type of array, regardless of whether it is sorted or not. However, it may not be efficient for large arrays as its time complexity is O(n).  The binary search, on the other hand, is efficient for sorted arrays. It reduces the search space by half at each step, making it much faster than linear search for large arrays. However, it requires the array to be sorted initially. Its time complexity is O(log n). |

|  |
| --- |
| **C/C++ Code implemented:** |
| 1. **Linear Search:**   #include <stdio.h>  int linearSearch(int arr[], int n, int x) {      for (int i = 0; i < n; i++) {          if (arr[i] == x) {              return i;          }      }      return -1; // Element not found  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      int arr[] = {2, 3, 4, 10, 40};      int n = sizeof(arr) / sizeof(arr[0]);      int x = 10;      int result = linearSearch(arr, n, x);      if (result == -1) {          printf("\nElement is not present in the array.");      } else {          printf("\n\nElement is present at index %d", result);      }      return 0;  }   1. **Binary Search:**   #include <stdio.h>  int binarySearch(int arr[], int l, int r, int x) {      while (l <= r) {          int mid = l + (r - l) / 2;          if (arr[mid] == x) {              return mid;          }          if (arr[mid] < x) {              l = mid + 1;          } else {              r = mid - 1;          }      }      return -1; // Element not found  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      int arr[] = {2, 3, 4, 10, 40};      int n = sizeof(arr) / sizeof(arr[0]);      int x = 10;      int result = binarySearch(arr, 0, n - 1, x);      if (result == -1) {          printf("\n\nElement is not present in the array.");      } else {          printf("\n\nElement is present at index %d", result);      }      return 0;  } |

|  |
| --- |
| **Output:** |
| 1. **Linear Search:**      1. **Binary Search:** |

|  |
| --- |
| **Conclusion:** |
| In this experiment, the comparison of various searching techniques highlights the trade-offs between time complexity, data distribution, and sorting requirements. Understanding the nuances of each method is essential for selecting the most efficient searching algorithm based on the characteristics of the dataset. |

|  |
| --- |
| **Post Lab:** |
| 1. **Compare various searching techniques.**   There are several searching techniques, each with its advantages and disadvantages. Let's compare some of the commonly used searching techniques:   1. Linear Search:    1. Works well for small data sets or unsorted arrays.    2. Requires a full scan of the array, resulting in a time complexity of O(n) in the worst case.    3. Simple to implement and understand. 2. Binary Search:    1. Suitable for sorted arrays, as it operates by repeatedly dividing the search interval in half.    2. Has a time complexity of O(log n) in the worst case, making it efficient for large datasets.    3. Requires the array to be sorted initially.    4. Not suitable for linked lists, as random access is not allowed in linked lists. 3. Interpolation Search:    1. More efficient than binary search for uniformly distributed arrays.    2. Uses the position formula to find the probable position of the element.    3. Works best for uniformly distributed sorted arrays, where elements are uniformly distributed throughout the range. 4. Exponential Search:    1. Useful for unbounded/infinite arrays.    2. First, it finds the range where the element may be present and then performs a binary search in that range.    3. Works well when the element is closer to the first element in the array. 5. Hashing:    1. Provides constant-time average case complexity for searching.    2. Requires additional space for hash table creation.    3. Well-suited for situations where the data can be easily hashed. 6. Ternary Search:    1. An advanced searching algorithm that divides the array into three parts.    2. Requires the array to be sorted.    3. Can be more efficient than binary search in certain cases. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |