|  |  |
| --- | --- |
| **Assignment No:1** | |
| **Aim:** | Develop a C++ program to implement both Linear and Binary search algorithms on an array of numbers. |
| **Objective:** | Gain a thorough understanding of searching algorithms, focusing on their implementation, analysis of time complexity, and applicability in various scenarios. This will enhance problem-solving skills and prepare students to select appropriate searching techniques based on the characteristics of the data and the requirements of the application. |
| **Theory:** | Searching is a fundamental operation in computer science, used to retrieve specific elements from a data structure. Linear and Binary searches are two basic techniques for finding an element in an array:   * **Linear Search** involves sequentially checking each element of the array until the desired element is found or the array ends. * **Binary Search** is a more efficient algorithm that works on sorted arrays by repeatedly dividing the search interval in half.   **Advantages of Searching:**   1. **Data Retrieval Efficiency:**    1. Searching algorithms are crucial for quick data retrieval, especially in large datasets. 2. **Binary Search Efficiency:**    1. Binary search is significantly faster than linear search for large datasets, with a time complexity of O(log n) compared to O(n) for linear search. 3. **Optimized Data Structures:**    1. Effective searching algorithms are essential for optimized operations in data structures like trees, graphs, and databases. 4. **Foundational for Advanced Algorithms:**    1. Searching is a building block for more complex algorithms, such as sorting, pattern matching, and data mining.   **Applications:**   * Searching for a specific record in a database * Finding a file in a computer system * Implementing lookup tables and dictionaries * Real-time data retrieval in applications like GPS systems and search engines * Task scheduling and prioritization in operating systems |
| **Algorithm:** | **Linear Search:**  1. Start  2. Initialize the array and the size of the array.  3. Input the element to search (key).  4. Initialize a flag to indicate whether the key is found or not.  5. Iterate through each element of the array from the first element to the last:  - Compare the current element with the key.  - If the current element matches the key:  - Output the position of the key.  - Set the flag to indicate the key is found.  - Exit the loop.  6. If the key is not found (flag remains unchanged):  - Output "key not found".  7. End  **Binary Search:**  1. Start  2. Initialize the array, the size of the array, and the search range (first and last indices).  3. Input the element to search (key).  4. Initialize a flag to indicate whether the key is found or not.  5. While the search range is valid (first index is less than or equal to the last index):  - Calculate the middle index of the current search range.  - Compare the middle element with the key:  - If the middle element matches the key:  - Output the position of the key.  - Set the flag to indicate the key is found.  - Exit the loop.  - If the key is greater than the middle element:  - Update the search range to the upper half (set first index to mid + 1).  - If the key is less than the middle element:  - Update the search range to the lower half (set last index to mid - 1).  6. If the key is not found (flag remains unchanged):  - Output "key not found".  7. End |
| **Program:** | **Linear Search:**  #include <iostream>  using namespace std;  void linearSearch(int arr[], int size, int key) {  bool found = false;  for (int i = 0; i < size; i++) {  if (arr[i] == key) {  cout << "Key found at position: " << i + 1 << endl;  found = true;  break;  }  }  if (!found) {  cout << "Key not found" << endl;  }  }  int main() {  int n, key;  cout << "Enter the number of elements in the array: ";  cin >> n;  int arr[n];  cout << "Enter the elements of the array: ";  for (int i = 0; i < n; i++) {  cin >> arr[i];  }  cout << "Enter the key to search: ";  cin >> key;  linearSearch(arr, n, key);  return 0;  }  **Binary Search:**  #include <iostream>  using namespace std;  void binarySearch(int arr[], int size, int key) {  int first = 0; int last = size - 1;  bool found = false;  while (first <= last) {  int mid = (first + last) / 2;  if (arr[mid] == key) {  cout << "Key found at position: " << mid + 1 << endl;  found = true;  break;  else if (arr[mid] < key) {  first = mid + 1;  }  else {  last = mid - 1;  }  }  if (!found) { cout << "Key not found" << endl;  }  }  int main() {  int n, key;  cout << "Enter the number of elements in the array: ";  cin >> n;  int arr[n];  cout << "Enter the sorted elements of the array: ";  for (int i = 0; i < n; i++) {  cin >> arr[i];  }  cout << "Enter the key to search: ";  cin >> key;  binarySearch(arr, n, key);  return 0;  } |
| **Output:** | **Linear Search Output:**  Enter the number of elements in the array: 5  Enter the elements of the array: 10 25 30 45 60  Enter the key to search: 30  Key found at position: 3  **Binary Search Output:**  Enter the number of elements in the array: 5  Enter the sorted elements of the array: 10 20 30 40 50  Enter the key to search: 35  Key not found |
| Conclusion: Computing and comparing both the Linear and the Binary search algorithms bring home the point that effective retrieval of data is a necessary aspect. Knowledge of the both algorithms and their time complexities enables the students to differentiate between which technique is apt for which kind of dataset and which one is required as per the requirements from the application end.  1. Linear Search is very simple and most appropriate for datasets which are unsorted or perhaps too small to require checking them in a sequence. The time complexity of O(n) is too inefficient for large databases as the computer goes into an iteration of one by one checking of elements.  2. Binary Search is, therefore, an optimization algorithm for sorted arrays. It runs very fast with a very large size of n datasets, as it halves the search interval in every iteration. It requires a sorted dataset, which usually adds overhead in unsorted data scenarios.  This two basic search algorithms will be understood in an all-inclusive manner by the students through this project as they improve their knowledge of when to apply which method in the enhancement of the problem-solving skill. When it deals with a small or unsorted array, linear search is preferably used, but for large sorted datasets, the most suitable approach is binary search as it highly efficient. Indeed, the very basis for the construction of further sophisticated algorithms applied in various domains such as data mining, artificial intelligence, and database indexing is knowledge of search algorithms. Therefore, understanding searching techniques is essential for optimal design of appropriate data structures, improvement in application performance, and solution of complex computational problems. | |
| **Date:** | |
| **Staff Sign:** | |
| \***END**\* | |