**Batch: A3 Roll. No.: 16010121051**

**Experiment:**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:**  Implementation of hashing concept |

**Objective:** To understand various hashing methods

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO4** | Demonstrate sorting and searching methods. |

**Websites/books referred:**

**1.** **stackoverflow**

**2. geeksforgeeks**

**3. Rema Thareja**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Abstract**: -

*(Define Hashing ,hash function, list collision handling methods)*

*Searching is the process of finding a given value position in a list of values. It decides*

*whether a search key is present in the data or not. It is the algorithmic process of finding*

*a particular item in a collection of items. It can be done on internal data structure or on*

*external data structure.*

*Searching methods( Linear search, binary search, hash search):*

*Linear Search: A linear search or sequential search is a method for finding an element*

*within a list. It sequentially checks each element of the list until a match is found or the*

*whole list has been searched.*

*Binary Search: Binary search, also known as half-interval search, logarithmic search,*

*or binary chop, is a search algorithm that finds the position of a target value within a*

*sorted array. Binary search compares the target value to the middle element of the*

*array.*

*Hash Search: One of the most common approaches is to use a hash function to*

*transform one or more characteristics of the searched-for item into a value that is used*

*to index into an indexed hash table. Hash-based searching has better average-case*

*performance than the other search algorithms described.*

*Algorithm Binary Search:*

*Condition : Only applicable to sorted arrays.*

* Compare x with the middle element.*

* If x matches with the middle element, return the mid index.*

* Else If x > mid element, search in the right half.*

* Else search in the left half.*

*Hashing: (Define hashing, collision and list collision handling methods)*

*Hashing: Hashing is the process of converting a given key into another value. A hash*

*function is used to generate the new value according to a mathematical algorithm.*

*Collision: A collision occurs when more than one value to be hashed by a particular*

*hash function hash to the same slot in the table or data structure (hash table) being*

*generated by the hash function.*

*Collision handling methods:*

* Open Hashing (Separate chaining)*

* Closed Hashing (Open Addressing)*

* Liner Probing*

* Quadratic probing*

* Double hashing*

**Code and output screenshots:**

**#include <bits/stdc++.h>**

**using namespace std;**

**int binarySearch(int arr[], int l, int r, int x){**

**while (l <= r) {**

**int m = l + (r - l) / 2;**

**if (arr[m] == x)**

**return m; //middle element**

**if (arr[m] < x)**

**l = m + 1; //left half**

**else**

**r = m - 1; //right half**

**}**

**return -1; //element not prsent**

**}**

**int main(){**

**int n;**

**cout<<"Enter number of elements: ";**

**cin>>n;**

**int arr[n];**

**cout<<"Enter array elements in sorted order: ";**

**for(int i=0;i<n;i++){**

**cin>>arr[i];**

**}**

**int x;**

**cout<<"Enter the number to be searched: ";**

**cin>>x;**

**int result = binarySearch(arr, 0, n - 1, x);**

**if(result == -1)**

**cout << "Element is not present in array";**

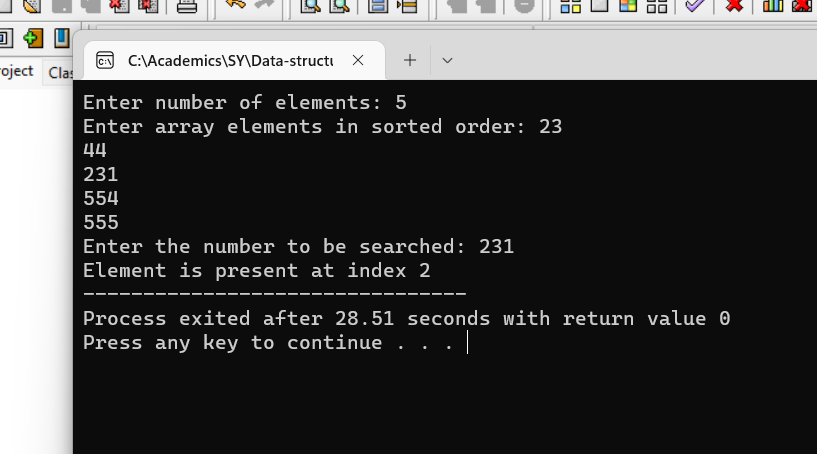
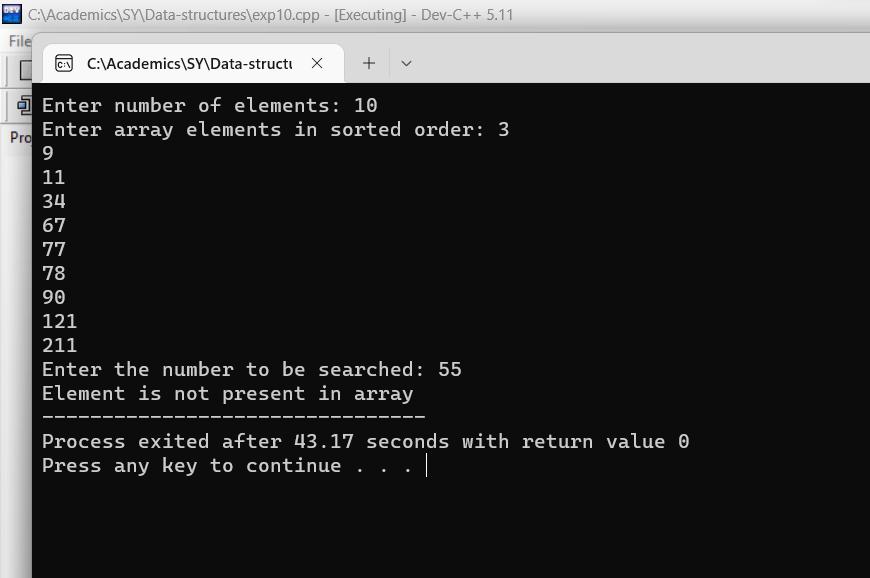
**else**

**cout << "Element is present at index " << result;**

**return 0;**

**}**

**Outputs:**

****

**Conclusion: -Searching algorithms were implemented successfully.**

**Post lab questions-**

1. **Compare and contrast various collision handling methods.**

- In Open Hashing each cell in the array points to a list containg the collisions.

The hashing has produced the same index for all items in the linked list.

- In Closed Hashing you use only one array for everything. You store the

collisions in the same array. The trick is to use some smart way to jump

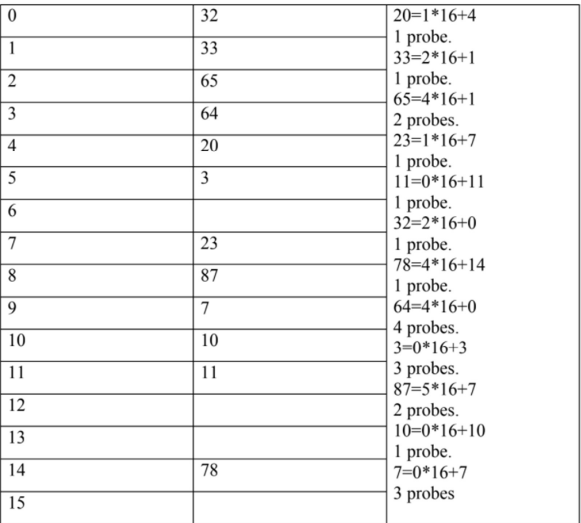
from collision to collision until you find what you want. And do this in a

reproducible / deterministic way.

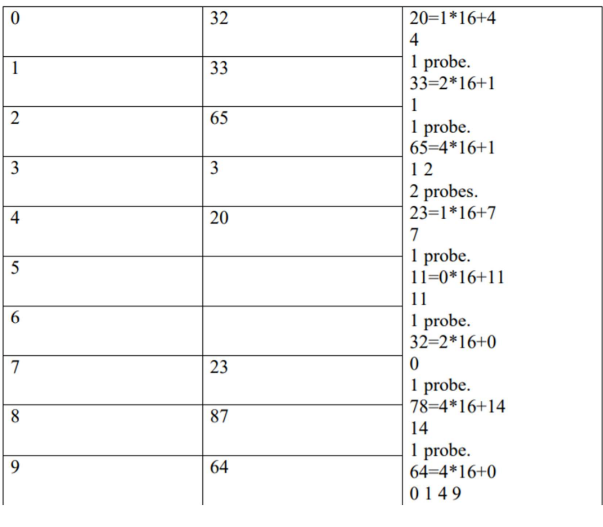
1. **Store the given numbers in bucket of size 16, resolve the collisions if any with**
   1. **Linear probing**
   2. **Quadratic hashing**
   3. **Chaining**

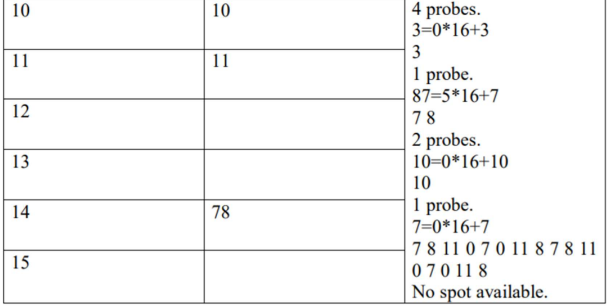
**20, 33, 65, 23, 11, 32, 78, 64, 3, 87, 10, 7**

**A]Linear probong**

****

**B]Quardic hashing**

****

****

C]chaining

