**Batch: B3 Roll. No.: 121**

**Experiment: 10**

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

|  |
| --- |
| **Title:**  Implementation of Searching algorithms |

**Objective:** To understand various searching methods

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO3** | Demonstrate sorting and searching methods. |

**Websites/books referred:**

**1.** Michael T. Goodrich, Roberto Tamassia, and David M. Mount. 2009. Data Structures and Algorithms in C++ (2nd. ed.). Wiley Publishing

**2.** Aaron M. Tenenbaum, Yedidyah Langsam, and Moshe J. Augenstein. Data structures using C.

**3.** [**https://www.geeksforgeeks.org/what-is-hashing/**](https://www.geeksforgeeks.org/what-is-hashing/)

**4.** [**https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/**](https://www.geeksforgeeks.org/open-addressing-collision-handling-technique-in-hashing/)

**5.** [**https://classroom.google.com/u/0/c/NTM5NDM0NzUxMTQ2**](https://classroom.google.com/u/0/c/NTM5NDM0NzUxMTQ2)

**6.<https://www.javatpoint.com/ds-linear-search-vs-binary-search#:~:text=Linear%20search%20is%20a%20search,matched%20with%20a%20searched%20element>.**

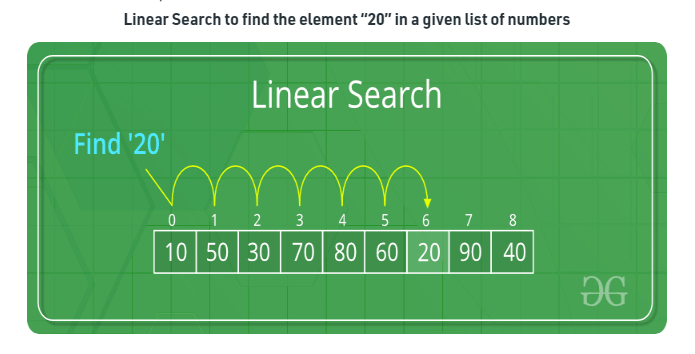
**Abstract**: -

Search is a process to retrieve information stored within some data structure, or calculated in the search space of a problem domain, either with discrete or continuous values.

**Searching methods:**

1. Linear Search

The list or data structure is traversed sequentially and every element is checked for the presence of some information.

****

The procedure is as follows:

1. Step through array of records, one at a time.
2. Look for the record with the matching key.
3. The search stops when the key is found or when the search has examined all elements in the given array of records.

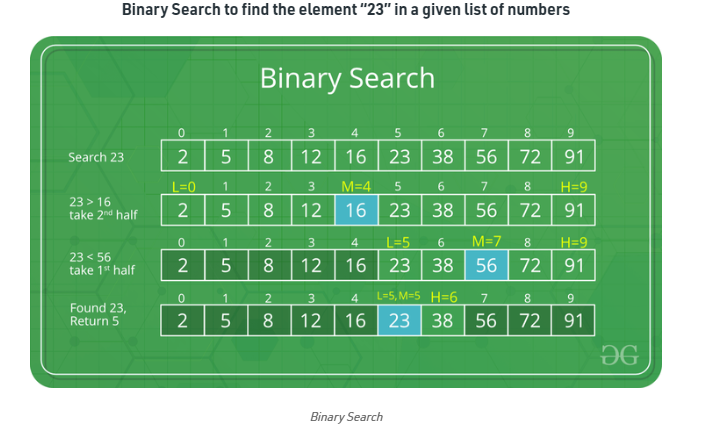
Linear search is useful for the searching of small to medium sized lists. Further, the list does not need to be sorted and the searching is not affected by insertions and deletions. However, it is not the best choice for searching of large lists, as the time taken to search a list is proportional to the number of elements in the list.

1. Binary Search

Binary Search works on the principle of “Divide and Conquer”. It, however, needs the input to be sorted. But on the positive side, it is one of the fastest searching algorithms.

Binary search is an efficient method of searching. It indicates whether the element being searched is present before or after the current position in the list. This information is used to narrow the search. It is especially suited for large data sets.

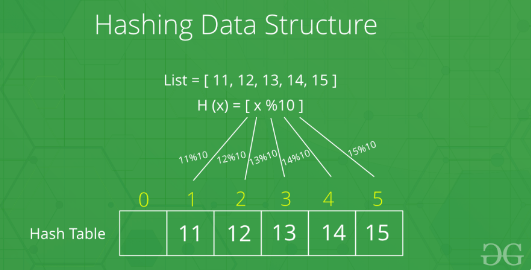
However, Binary Search implemented using recursion takes more space. Also, it gets affected by insertions and deletions in the input list.

****

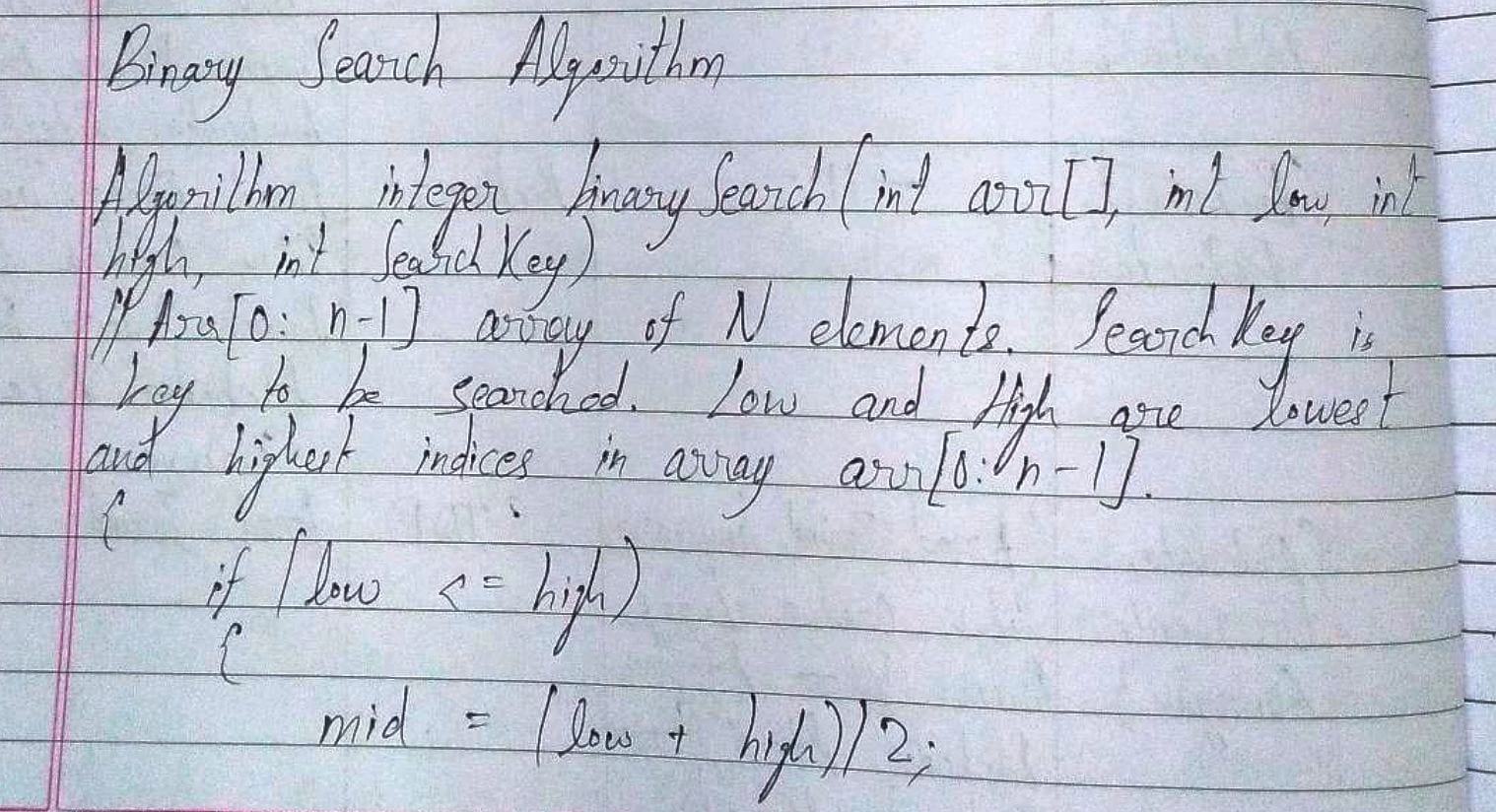
1. Hash Search

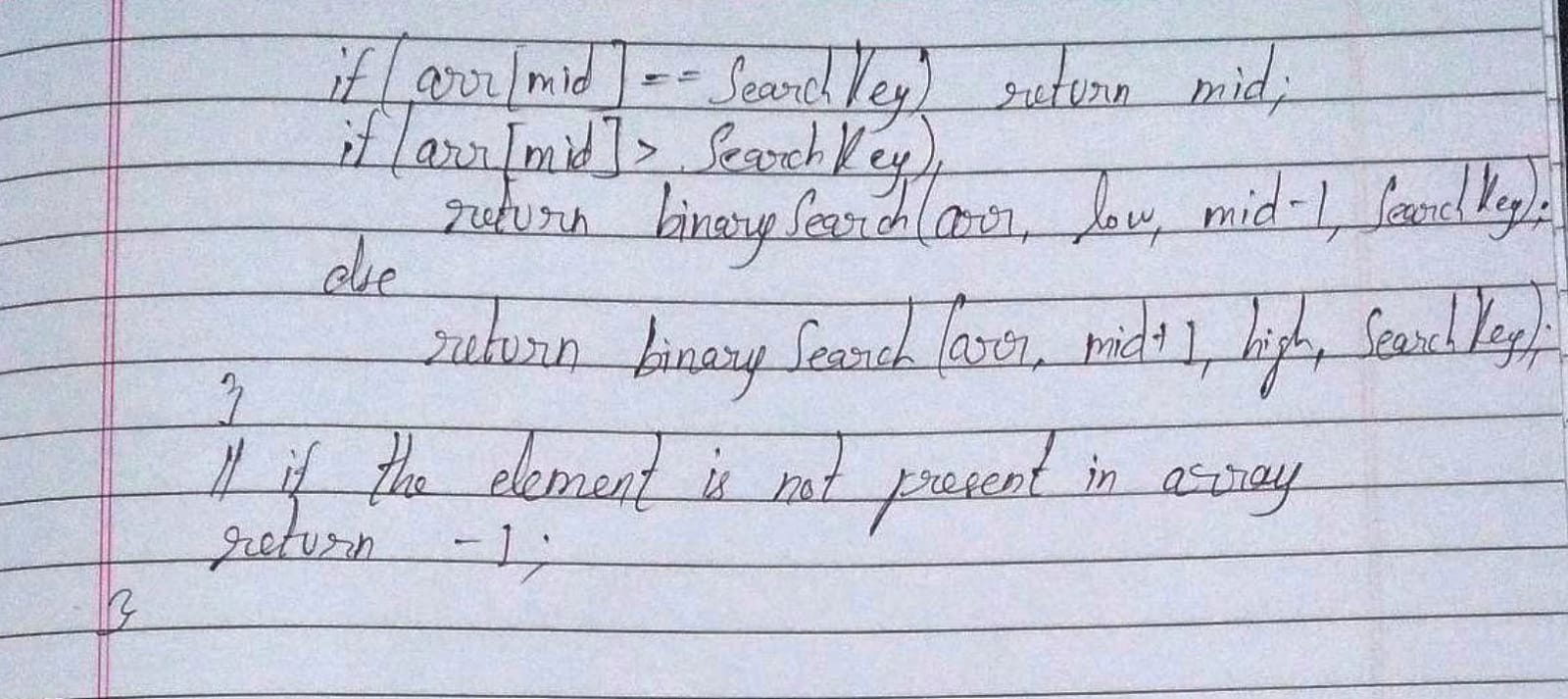
A Hash Table (also known as Hash Map) is a data structure that associates keys (names) with values (attributes). A Hash Table is a collection of pairs. Hash Tables are often used to implement associative arrays. The Worst-case searching, inserting and deleting time for Hash Tables is O(size). Thus, it has a constant time for searching, inserting or deleting irrespective of the size of the collection of pairs, giving it a huge advantage over other searching methods. This is because a Hash Table is an array of some fixed size, usually a prime number.

Hash Tables use an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

****

**Algorithm Binary Search:**





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

Hashing is a concept in Data Structures which consists of the following three parts:

1. Hash Table: An array that stores pointers to records corresponding to a given data. An entry in hash table is NIL if no existing data has hash function value equal to the index for entry. In simple terms, we can say that hash table is a generalization of array. Hash table gives the functionality in which a collection of data is stored in such a way that it is easy to find those items later if required. This makes searching of an element very efficient.
2. Hash Function: A function that converts a given big data value to a small practical integer value. The mapped integer value is used as an index in hash table. So, in simple terms it can be said that a hash function is used to change a given key into a unique slot index. If every key is mapped into a unique slot index, then the hash function is known as a perfect hash function. It is very difficult to create a perfect hash table but the point is to create such a hash function with the help of which the number of collisions are as few as possible. A good hash function has the following properties:
3. Efficiently computable
4. Should uniformly distribute the keys (each table position equally likely for each).
5. Should minimize collisions.
6. Should have a low load factor (number of items in table divided by size of the table).
7. Collision Handling: Since a hash function computes a small number for a big key, there is a possibility that two keys will result in the same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and mush be handled using some collision handling technique.
8. Chaining: The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.
9. Open Addressing: All elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, the table slots are examined one by one until the desired element is found or it is clear that the element is not in the table. There are two types – Linear Probing and Quadratic Hashing.

**Code and output screenshots for Binary search:**

#include<stdio.h>

int binary\_search(int a[], int, int);

void display(int a[], int);

void main()

{

int n, arr[100], i, search, pos;

printf("Enter the size of array, which must be less than or equal to 100: ");

do

{

scanf("%d", &n);

if(n>100)

printf("\nPlease try again! The array size must be less than or equal to 100.\nEnter again: ");

}while(n>100);

printf("\nEnter the array elements in ASCENDING order.");

for(i = 0; i < n; i++)

{

printf("\nFor element %d, enter value: ", (i+1));

scanf("%d", &arr[i]);

}

printf("\nThe array entered is displayed.\n");

display(arr, n);

printf("\nEnter the element to search: ");

scanf("%d", &search);

pos = binary\_search(arr, n, search);

if(pos != -1)

printf("\nThe value is found at position %d.", pos);

else

printf("\nThe value is not found.");

}

int binary\_search(int a[], int n, int search)

{

int first = 0, last, middle;

last = n;

middle = (first + last)/2;

while(first <= last)

{

if(a[middle] < search)

first = middle+1;

else if(a[middle] > search)

last = middle-1;

else

return (middle+1);

middle = (first+last)/2;

}

return -1;

}

void display(int a[], int n)

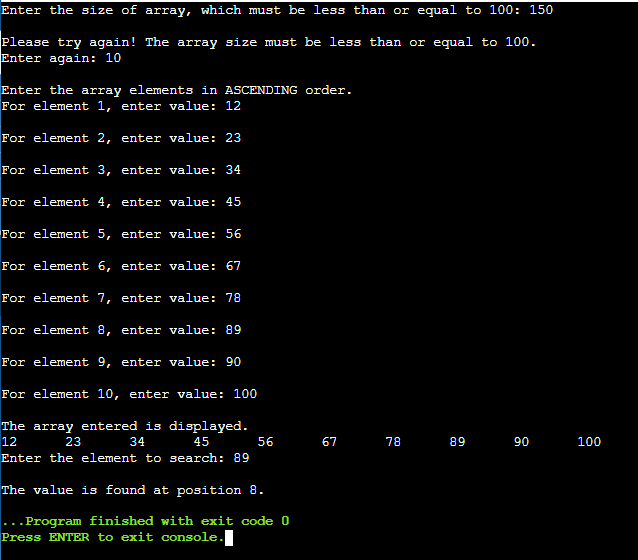
{

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

printf("%d\t", a[i]);

}

**Output:**

****

**Conclusion: -**

Thus, in this experiment, the concept of different types of searching methods has been learnt. Each searching method has its own advantages and disadvantages. Therefore, the selection of a particular searching method is heavily dependent on the situation. Sometimes the method would be too slow (for large lists) or would be too complicated (for small lists), or sometimes there could be other issues. So such things are resolved by understanding the requirement and then applying the most suitable searching method.

**Post lab questions-**

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

**Ans.**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Chaining** | **Open Addressing** |
| **1.** | Chaining is simpler to implement. | Open Addressing requires more computation. |
| **2.** | Hash table never fills up. Thus, more elements can always be added. | The table may become full. |
| **3.** | It is less sensitive to hash function or load factor. | It requires extra care to avoid clustering and load factor. |
| **4.** | Cache performance is not good as keys are stored using linked list. | It provides better cache performance as everything is stored in the same table. |

1. **Compare linear search and binary search**

**Ans.**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Linear Search** | **Binary Search** |
| **1.** | The elements of array don’t need to be sorted. | The elements of array needs to be sorted. |
| **2.** | It is based on Sequential Approach. | It is based on Divide and Conquer Approach. |
| **3.** | It is preferable for small-sized data set. | It is preferable for large-sized data set. |
| **4.** | The worst case scenario for finding an element is O(n). | The worst case scenario for finding an element is O(). |

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**