SUNWAY

INT'L BUSINESS SCHOOL

Programme Name: ________**BCS HONS**____

Course Code: ___**CSC 2516**________

Course Name: _______**Data Structure and Algorithm**______________________________

Assignment / Lab Sheet / Project / Case Study No. _**1**___

Date of Submission: _______**7/12/2021**_____________

Submitted By:

Student Name**: Dipesh Tha Shrestha**

IUKL ID: **041902900028**

Semester**: Fourth Semester**

Intake**: September 2019**

Submitted To:

Faculty Name**: Prakash Chandra Sir**

Department**: LMS**

1. **Briefly explain the following terms:-**
    a. **Abstract Data Type (ADT)**

An ADT is a mathematical model of a data structure that specifies the type of data stored, the operations supported on them, and the types of parameters of the operations. An ADT specifies what each operation does, but not how it does it. Typically, an ADT can be implemented using one of many different data structures. A useful first step in deciding what data structure to use in a program is to specify an ADT for the program. In simple, Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of value and a set of operations.

b. **Data Structure**

A data structure is a particular way of organizing data in a computer so that it can be used effectively. A data structure is a specialized format for organizing, processing, retrieving and storing data. Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc.

c. **Array**

Arrays are defined as the collection of similar type of data items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

d. **Stack**

Stack is a linear data structure which follows a particular order in which the operations are performed. It is a simple data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack and the only element that can be removed is the element that is at the top of the stack, just like a pile of objects.

e. **Queue**

A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT. Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.

2.

a. **Write an algorithm for Binary search.**

Answer: An algorithm for Binary search is given below:

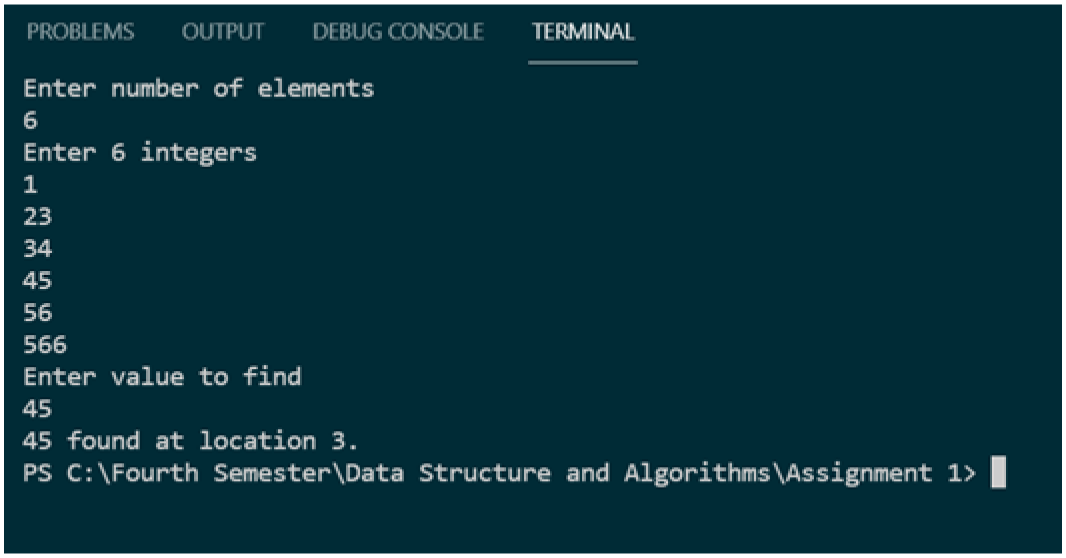
Step 1 - Read the search element from the user.
Step 2 - Find the middle element in the sorted list.
Step 3 - Compare the search element with the middle element in the sorted list.
Step 4 - If both are matched, then display "Given element is found!!!" and terminate the function.
Step 5 - If both are not matched, then check whether the search element is smaller or larger than the middle element.
Step 6 - If the search element is smaller than middle element, repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.
Step 7 - If the search element is larger than middle element, repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.
Step 8 - Repeat the same process until we find the search element in the list or until sublist contains only one element.
Step 9 - If that element also doesn't match with the search element, then display "Element is not found in the list!!!" and terminate the function.

b. Write a C program to implement Binary search.

Code:

```
#include <stdio.h>
int main()
{
 int c, first, last, middle, n, search, array[100];

 printf("Enter number of elements\n");
 scanf("%d", &n);

 printf("Enter %d integers\n", n);

 for (c = 0; c < n; c++)
  scanf("%d", &array[c]);

 printf("Enter value to find\n");
 scanf("%d", &search);

 first = 0;
 last = n - 1;
 middle = (first+last)/2;

 while (first <= last) {
  if (array[middle] < search)
   first = middle + 1;
  else if (array[middle] == search) {
   printf("%d found at location %d.\n", search, middle);
   break;
  }
  else
   last = middle - 1;

  middle = (first + last)/2;
 }
 if (first > last)
  printf("Not found! %d isn't present in the list.\n", search);
  return 0;
}
```

Output:



```
Enter number of elements
6
Enter 6 integers
1
23
34
45
56
566
Enter value to find
45
45 found at location 3.
PS C:\Fourth Semester\Data Structure and Algorithms\Assignment 1>
```

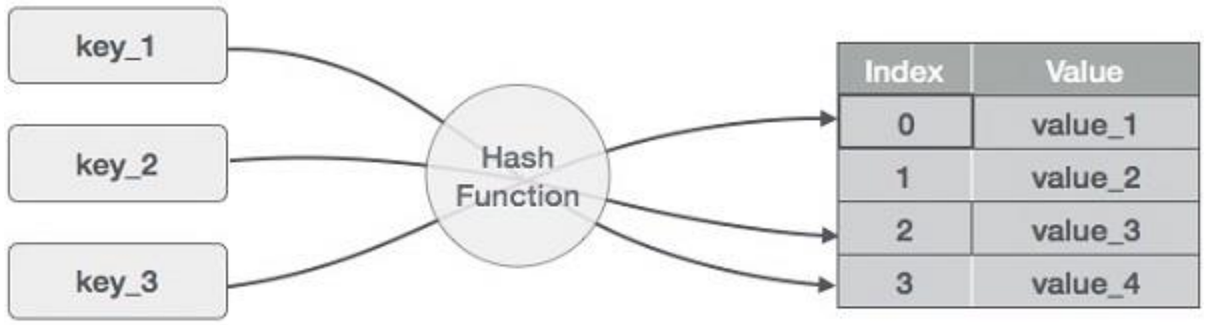
c. Explain the best case and worst case efficiency of binary search algorithms

Answer: The best case for a binary search is finding the target item on the first look into the data structure, so O (1). The worst case for a binary search is searching for an item which is not in the data. In this case, each time the algorithm did not find the target, it would eliminate half the list to search through, so O (log n).

3. With proper diagram, explain the following:

a) Concept of hashed list search.

A Hash table is a data structure that stores some information, and the information has basically two main components, i.e., key and value. The hash table can be implemented with the help of an associative array. The efficiency of mapping depends upon the efficiency of the hash function used for mapping. Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.



(1,20)
(2,70)
(42,80)
(4,25)
(12,44)
(14,32)
(17,11)
(13,78)
(37,98)

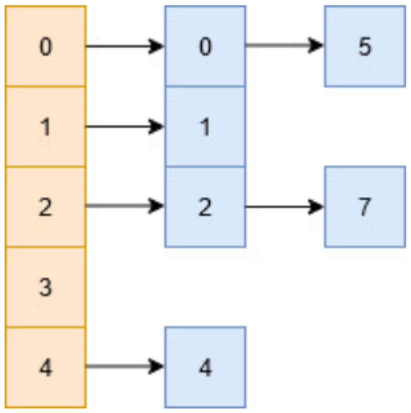
Sr.No.	**Key**	**Hash**	**Array Index**
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	42	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	14	14 % 20 = 14	14
7	17	17 % 20 = 17	17

8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

b) Technique to resolve collisions in hashed list search.

Answer:
There are two types of collision resolution techniques.
Separate chaining (open hashing)
Open addressing (closed hashing)

Separate chaining
In this technique, a linked list is created from the slot in which collision has occurred, after which the new key is inserted into the linked list. This linked list of slots looks like a chain, so it is called **separate chaining**. It is used more when we do not know how many keys to insert or delete.



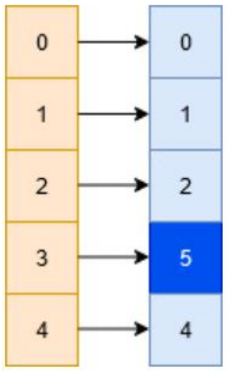
Time complexity
Its worst-case complexity for searching is o(n).
Its worst-case complexity for deletion is o(n).

Open addressing

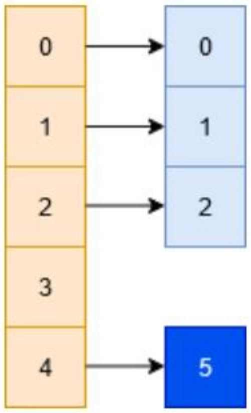
Open addressing is collision-resolution method that is used to control the collision in the hashing table. There is no key stored outside of the hash table. Therefore, the size of the hash table is always greater than or equal to the number of keys. It is also called **closed hashing**.

The following techniques are used in open addressing:

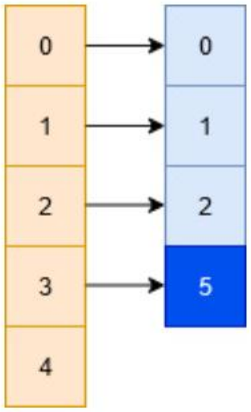
Linear probing: In this, when the collision occurs, we perform a linear probe for the next slot, and this probing is performed until an empty slot is found. In linear probing, the worst time to search for an element is O(table size). The linear probing gives the best performance of the cache but its problem is clustering. The main advantage of this technique is that it can be easily calculated.



Quadratic probing: This method lies in the middle of great cache performance and the problem of clustering. The general idea remains the same, the only difference is that we look at the Q(i) increment at each iteration when looking for an empty bucket, where Q(i) is some quadratic expression of i. A simple expression of Q would be Q(i) = i^2, in which case the hash function looks something like this:H(x,i)=(H(x)+i^2)%N



Double hashing; This method is based upon the idea that in the event of a collision we use an another hashing function with the key value as an input to find where in the open addressing scheme the data should actually be placed at.



<u>Thank you</u>