<u>**TITLE: SEARCHING**</u>

Finding the essential information from a group of items stored as elements in the computer memory is referred to as searching in data structures. Among them are:

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.

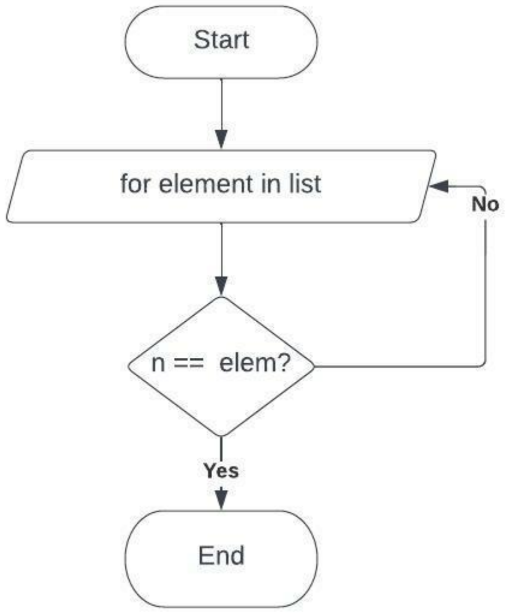
Algorithm for Linear search:

```
Input: n, list[n]

Func bubbleSort(n,list)

    for element in list:

        if n == element:
            return True
```



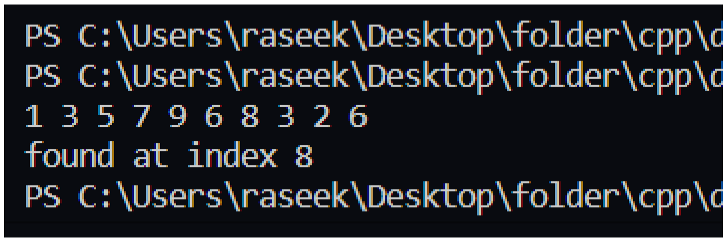
flowchart for linear search

WAP to implement Linear Search.

```
#include <stdio.h>

int linearSearch(int arr[],int size,int searchFor){
    for (int i=0;i<size;i++){
        if (arr[i] == searchFor){
            return i;
        }
    }
    return -1;
}

int main(){
    int arr[] = {1,3,5,7,9,6,8,3,2,6};
    int size = sizeof(arr) / sizeof(int);
    int searchFor = 2;
    int indexof = linearSearch(arr,size,searchFor);
    for (int i=0;i<size;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    if (indexof >=0){
        printf("found at index %d",indexof);
    }else{
        printf("not found");
    }
}
```



```
PS C:\Users\raseek\Desktop\folder\cpp\d
PS C:\Users\raseek\Desktop\folder\cpp\d
1 3 5 7 9 6 8 3 2 6
found at index 8
PS C:\Users\raseek\Desktop\folder\cpp\d
```

Discussion:

In sequential searching, we start by looking for the needed item in the first element of the list. If the item or key matches the first element, we print the element as found; if not, we continue the search process by comparing the key with each element of the list until the searched item and element present in the list are matched. The element is not found is printed if the entire list has been searched and the key cannot be located

Binary Search: Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n)

Algorithm for selection sort:

1) Low = 0, high= len(list) - 1
2) Set mid = (low + high) / 2
3) If n equals list[mid] then, return index of n
4) If list[mid] > n, high = mid – 1, else low = mid + 1
5) Repeat step 2 to 4 until (low <= high)

Pseudocode:

```
Input: n,list[n]

Func BinarySearch(n,list[n])

    Low = 0 , high = sizeoflist – 1

    While (low <= high):

        mid = (low+high)/2

        If (lis[mid] == n), return True

        If ( list[mid] > n ) then high = mid - 1

        else low = mid + 1

    end while
```

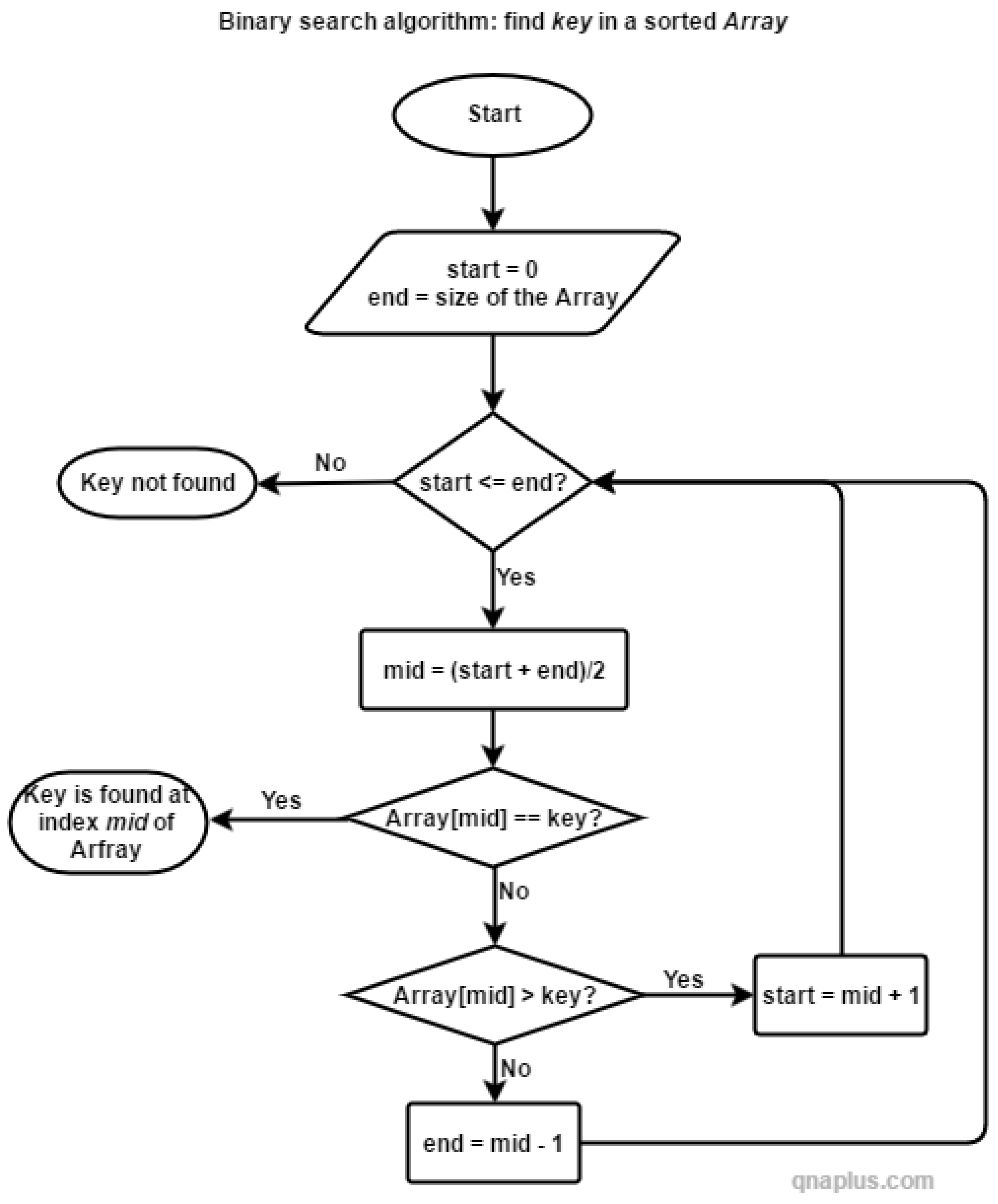
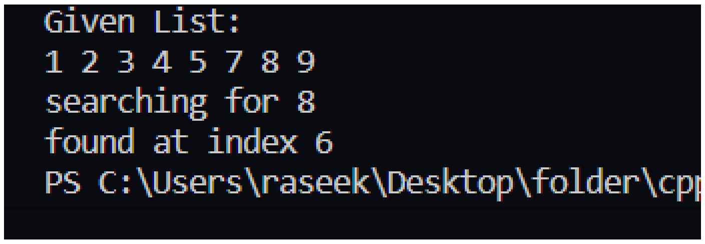


Figure: Binary Search

WAP to implement Binary Search.

```
#include <stdio.h>
int binarySearch(int arr[],int size,int searchFor){
    int low = 0;
    int high = size - 1;
    int mid;
    while (low <= high){
        mid = (low + high) / 2;
        if (arr[mid] == searchFor){
            return mid;
        }
        if (arr[mid] > searchFor){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }
}
int main(){
    int arr[] = {1,2,3,4,5,7,8,9};
    int size = sizeof(arr) / sizeof(int);
    int searchFor = 8;
    int indexof = binarySearch(arr,size,searchFor);
    printf("Given List:\n" );
    for (int i=0;i<size;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    printf("searching for %d\n",searchFor);
    if (indexof >=0){
        printf("found at index %d",indexof);
    }else{
        printf("not found");
    }
}
```



```
Given List:
1 2 3 4 5 7 8 9
searching for 8
found at index 6
PS C:\Users\raseek\Desktop\folder\cp
```

Discussion:

In binary searching, the low pointer is first placed at position 0 in the array, and the high pointer is initially set at the last member of the list. The middle value is then determined by using (low+high)/2, and if the key value and the value present in the middle position match, we print the element's location. We search the item in the subarray to the left of the middle item if the searched item cannot be found and the middle term is greater than the searched item. In the absence of that, we begin looking in the subarray to the right of the main item. Up till the subarray's size is reduced to zero, we repeat this operation on the subarray as well.

Hashing: Hashing is designed to solve the problem of needing to efficiently find or store an item in a collection. Hashing is a technique to make things more efficient by effectively narrowing down the search at the outset.

Algorithm for insertion sort:

1) Get hash using n % sizeoflist
2) If hash[key] == n, return key
3) Else calculate hash for n by add 1 until n is discovered
4) If n is not discovered then n is not present in the list.

Pseudocode:

```
func searchFromHashTable(int hash[],int searchFor,int size)
   int key = getKey(searchFor,size)
   if (hash[key] == searchFor)
      return key
   for (int i=0;i<size;i++)
      key = getKey(key+1,size)
      if (hash[key] == searchFor)
         return key
   return -1
```

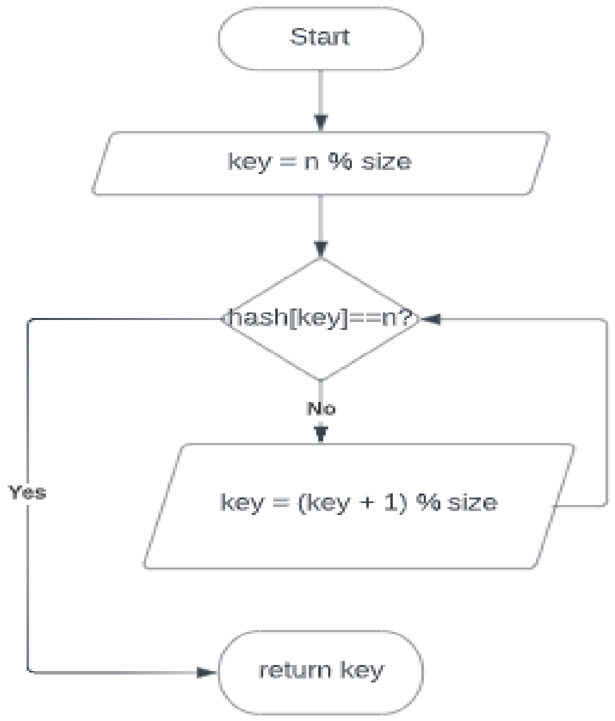


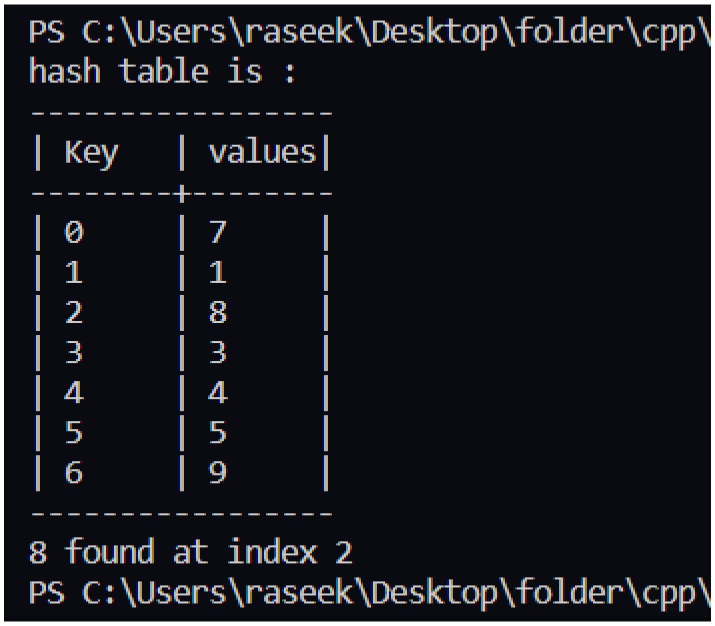
Figure: Hashing search

WAP to implement insertion sort.

```
#include <stdio.h>
#include <stdlib.h>

int getKey(int x,int size){
   return x % size;
}
void generateHashTable(int *hash,int *arr,int size){
   int key;
   for (int i=0;i<size;i++){
      hash[i] = -1;
   }
   for (int i=0;i<size;i++){
      key = getKey(arr[i],size);
      if (hash[key] == -1){
         hash[key] = arr[i];
      }
      else{ // collision occured
         //solving collision through linear probing
         while (hash[key] != -1){
                key = getKey(key+1,size);
            }
            hash[key] = arr[i];
        }
    }
}
int searchFromHashTable(int hash[],int searchFor,int size){
    int key = getKey(searchFor,size);
    if (hash[key] == searchFor){
        return key;
    }
    for (int i=0;i<size;i++){
        key = getKey(key+1,size);
        if (hash[key] == searchFor){
            return key;
        }
    }
    return -1;
}
int main(){
    int arr[] = {1,4,5,7,3,8,9};
    int size = sizeof(arr) / sizeof(int);
    int hash[size]; //key will be the index
    generateHashTable(&hash[0],&arr[0],size);
    printf("hash table is : \n");
    printf("-----------------\n");
    printf("| Key \t| values|\n");
    printf("--------+--------\n");
    for (int i=0;i<size;i++){
        printf("| %d\t| %d\t|\n",i,hash[i]);
    }
    printf("-----------------\n");

    int searchFor = 8;
    int result = searchFromHashTable(hash,searchFor,size);
    printf("%d found at index %d",searchFor,result);
}
```



```
PS C:\Users\raseek\Desktop\folder\cpp\
hash table is :
-----------------
| Key   | values|
--------+--------
| 0     | 7     |
| 1     | 1     |
| 2     | 8     |
| 3     | 3     |
| 4     | 4     |
| 5     | 5     |
| 6     | 9     |
-----------------
8 found at index 2
PS C:\Users\raseek\Desktop\folder\cpp\
```

Conclusion:

Several algorithm, including searching and hashing, were carried out at this lab. Binary searching, sequential searching, and hash table respectively, were used.