CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING

Department of Information Technology

Subject Name: Data Structures & Algorithms
Subject Code: IT259

Semester: III
Academic Year: 2024-25

Practical List
(Jun – Dec 2024)

Sr. No.	Aim of the Practical	DATE	SIGN
1.	1.1 Implement linear search using iterative and recursive using an array. 1.2 Left Rotation on Array https://www.hackerrank.com/challenges/array-left-rotation/problem?isFullScreen=true 1.3 Find All Duplicates in an Array https://leetcode.com/problems/find-all-duplicates-in-an-array/ **Practical Assignment** 1.4 Write a program to find non repeating element in an array (Unique element).		
2.	2.1 Implement binary search iterative and Recursive using an array. 2.2 Implement binary search for a long array of integers to find the required element. However, when the array size is quite large, the equation for finding the mid index may give the value which is out of range of integers. Implement the Binary search with a modified equation for finding mid. "the integer overflow problem" with binary search:		

	With a vast list of elements, “right” would be a very large value. Suppose your ‘left’ and ‘right’ are 16 bit unsigned integers. That means, they can only have a maximum value of 2^16 = 65536. Example: Consider: left = 65530 and right = 65531 left + right = 131061 (beyond the integer range). It will give garbage value It is known as an integer overflow. **Practical Assignment** 2.3 Kth Missing Positive Number https://leetcode.com/problems/kth-missing-positive-number/description/ 2.4 Search a 2D Matrix II https://leetcode.com/problems/search-a-2d-matrix-ii/		
3.	3.1 Implement Sorting Algorithm(s). (a) Bubble Sort (b) Selection Sort (c) Insertion Sort		
	Practical Assignment 3.2 Sort-colors (Leetcode) https://leetcode.com/problems/sort-colors/ 3.3 Sort https://leetcode.com/problems/sort-an-array/		
4.	4.1 Implement following operations of singly linked list. (a) Insert a node at front (b) Insert a node at end (c) Insert a node at specific position (d) Delete a node (e) Print nodes in Reverse (f) Traversal of the Linked List		

	4.2 Remove Nth Node from End of List https://leetcode.com/problems/remove-nth-node-from-end-of-list/ 4.3 Swap Nodes in Pairs https://leetcode.com/problems/swap-nodes-in-pairs/ **Practical Assignment** 4.4 Middle of the Linked List https://leetcode.com/problems/middle-of-the-linked-list/ 4.5 Reverse Linked List https://leetcode.com/problems/reverse-linked-list/		
5.	5.1 Implement following operations of doubly linked list. (a) Insert a node at front (b) Insert a node at end (c) Insert a node after given node information (d) Delete a node at front (e) Count number of nodes (f) Traversal of the Linked List **Practical Assignment** 5.2 Merge Two Sorted Lists https://leetcode.com/problems/merge-two-sorted-lists/ 5.3 Merge k Sorted Lists https://leetcode.com/problems/merge-k-sorted-lists/		

6.	6.1 Implement stack using array (static stack). https://practice.geeksforgeeks.org/problems/implement-stack-using-array/1 6.2 Implement stack using linked list (dynamic stack). https://practice.geeksforgeeks.org/problems/implement-stack-using-linked-list/1		

7.	7.1 Convert Infix to Postfix https://www.hackerrank.com/contests/2022-23-dsa-lab/challenges/convert-infix-expression-to-postfix-expression- 7.2 Evaluate Reverse Polish Notation https://leetcode.com/problems/evaluate-reverse-polish-notation/ 7.3 Valid Parentheses https://leetcode.com/problems/valid-parentheses/		
8.	8.1 Implement queue using array (static queue). https://practice.geeksforgeeks.org/problems/implement-queue-using-array/1 8.2 Implement queue using linked list (dynamic queue). https://practice.geeksforgeeks.org/problems/implement-queue-using-linked-list/1		
	Practical Assignment 8.3 Implement Stack using Queues https://leetcode.com/problems/implement-stack-using-queues/ 8.4 Implement Queue using Stacks https://leetcode.com/problems/implement-queue-using-stacks/		

9.	9.1 Implement Binary Tree with following operations. (a) Binary Tree Inorder Traversal https://leetcode.com/problems/binary-tree-inorder-traversal/ (b) Binary Tree Preorder Traversal https://leetcode.com/problems/binary-tree-preorder-traversal/ (c) Binary Tree Postorder Traversal https://leetcode.com/problems/binary-tree-postorder-traversal/ (d) Binary Tree Levelorder Traversal https://leetcode.com/problems/binary-tree-level-order-traversal/		
10.	10.1 Implement Binary Search Tree (BST) Insertion https://www.hackerrank.com/challenges/binary-search-tree-insertion/problem		
	10.2 Implement searching in Binary Search Tree (BST) https://leetcode.com/problems/search-in-a-binary-search-tree/ **Practical Assignment** 10.3 Kth Smallest Element in a BST https://leetcode.com/problems/kth-smallest-element-in-a-bst/		
11.	11.1 Implement DFS of Graph. https://practice.geeksforgeeks.org/problems/depth-first-traversal-for-a-graph/1		
	11.2 Implement BFS of Graph. https://practice.geeksforgeeks.org/problems/bfs-traversal-of-graph/1 **Practical Assignment** 11.3 Detect cycle in an undirected graph https://www.geeksforgeeks.org/problems/detect-cycle-in-an-undirected-graph/1		

12.	12.1 Implementing a Hash Table for Student Records Management <u>Background:</u> You are tasked with implementing a hybrid hash table to manage student records efficiently. Each student record consists of a unique student ID (key) and the corresponding student score (data). The hash table will support two methods of collision handling: separate chaining and linear probing. This flexibility ensures efficient handling of collisions, enabling you to choose the most suitable method based on different scenarios.		

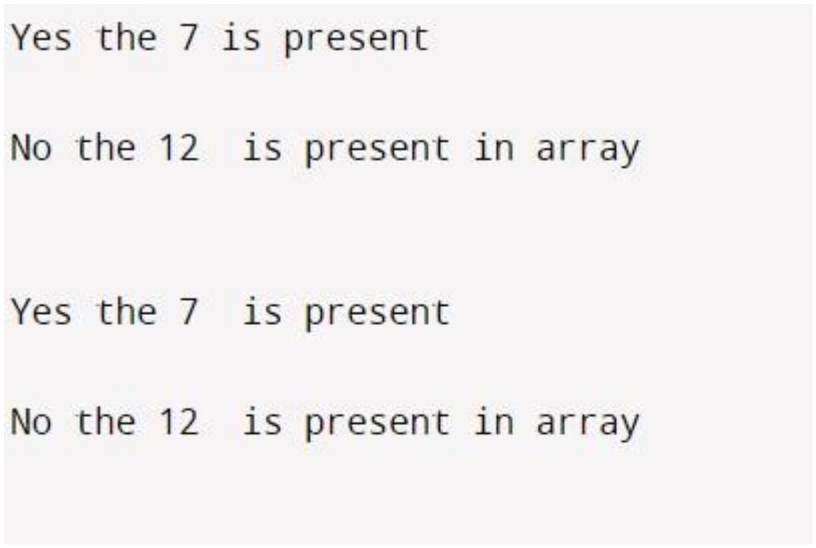
Practical - 1

1.1 Implement linear search using iterative and recursive using an array.

[VS code] CODE:

```
 #include <iostream>  using
namespace std;

 // Iterative  bool isPresent(int arr[],int
s,int data)
 {
 for(int i=0;i<s;i++)
 {
     if(data == arr[i])
     {
         return true;
     }
 }
 return false;
 }
 // Recursive  bool linearSearch(int
arr[],int s,int data)
 {

     if(arr[s-1]==data && s-1>=0)
     {
         return true;
     }
     else if(s-1>=0){  linearSearch(arr,s-1,data);
     }
     return false;
 }


 int main()
 {
 int arr[10]={1,2,3,4,5,6,7,8,9,10};  int
data =7;
 data =7;
 int a = isPresent(arr,9,data);
 if(a==1)
       cout <<"Yes the "<<data<<" is present\n"<<endl;  else
       cout <<"Yes the "<<data<<"  is present\n"<<endl;
 data =12;  a =
isPresent(arr,9,data);
if(a==1)
       cout <<"Yes the "<<data<<" is present in array\n"<<endl;  else
              cout <<"No the "<<data<<"  is present in array\n"<<endl;

 cout<<endl;  data =7;  a =
linearSearch(arr,9,data);
if(a==1)
       cout <<"Yes the "<<data<<" is present\n"<<endl;  else
       cout <<"Yes the "<<data<<"  is present\n"<<endl;
 data =12;  a =
linearSearch(arr,9,data);
if(a==1)
       cout <<"Yes the "<<data<<" is present in array\n"<<endl;  else
              cout <<"No the "<<data<<"  is present in array\n"<<endl;


 return 0;
 }
```

OUTPUT :



```
Yes the 7 is present

No the 12  is present in array


Yes the 7  is present

No the 12  is present in array
```

1.2 Left Rotation on Array CODE:

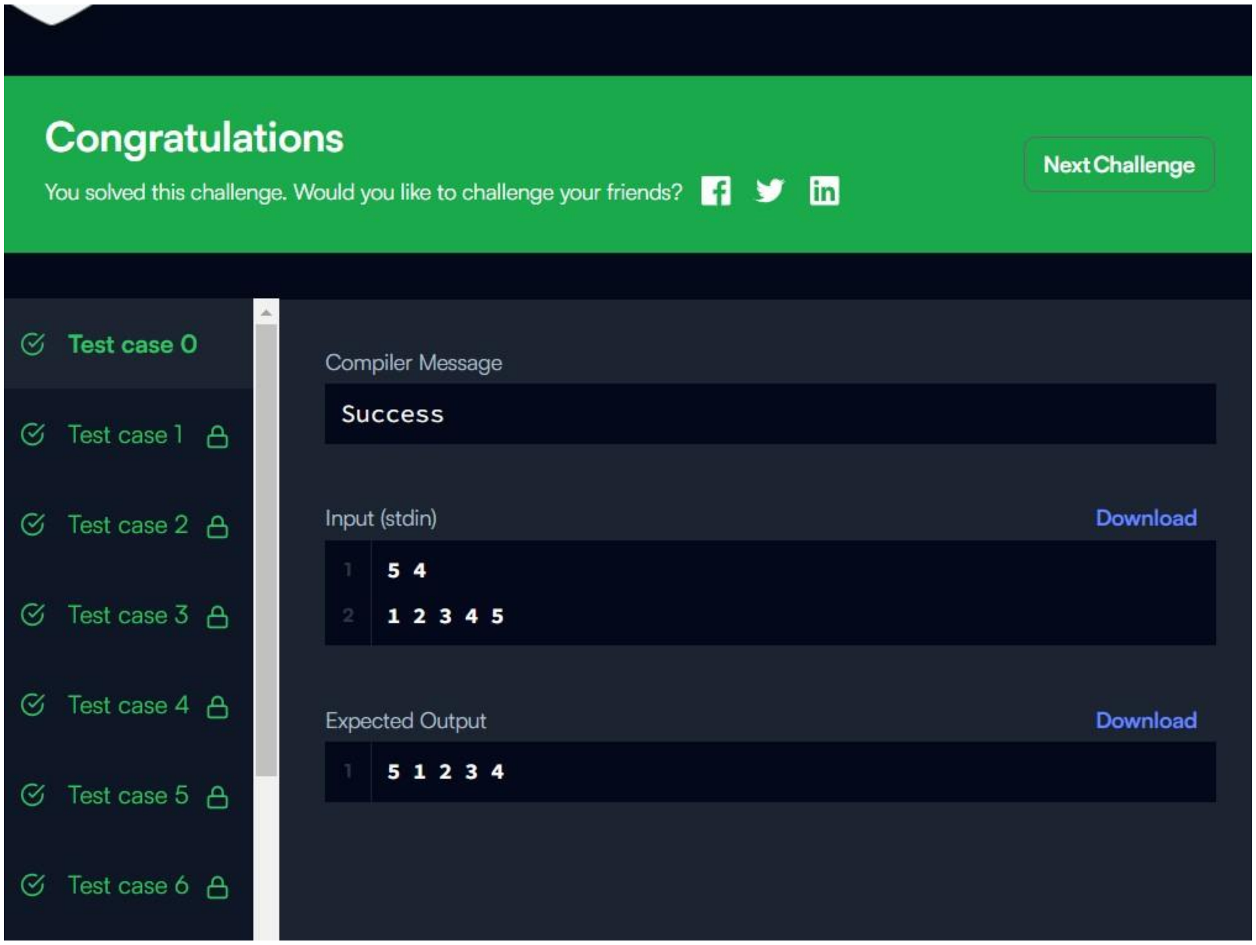
```
#include  < bits/stdc++.h >

using  namespace  std;

string ltrim( const  string &);  string
rtrim( const  string &);  vector<string>
split( const  string &);

/*
 *   Complete the 'rotateLeft' function below.
 *
 *   The function is expected to return an INTEGER_ARRAY.
 *   The function accepts following parameters:
```

```
 *   1. INTEGER d
 *   2. INTEGER_ARRAY arr
  */

 vector< int > rotateLeft( int  d, vector< int > arr) {
 // A[i] = B[(d+i)%s]  vector<
int > v(arr.size());  for ( int  i= 0
;i<arr.size();i++)  {
        v[i] = arr[(d+i)%arr.size()];
 }  return
v;
 }

 int  main()
 {
         ofstream fout(getenv( "OUTPUT_PATH" ));

         string first_multiple_input_temp;  getline(cin,
        first_multiple_input_temp);

         vector<string> first_multiple_input =
 split(rtrim(first_multiple_input_temp));

         int  n = stoi(first_multiple_input[ 0 ]);

         int  d = stoi(first_multiple_input[ 1 ]);

         string arr_temp_temp;  getline(cin,
        arr_temp_temp);

         vector<string> arr_temp = split(rtrim(arr_temp_temp));

         vector< int > arr(n);
```

```
    for  ( int  i =  0 ; i < n; i++) {  int
          arr_item = stoi(arr_temp[i]);

           arr[i] = arr_item;
    }

    vector< int > result = rotateLeft(d, arr);

    for  (size_t i =  0 ; i < result.size(); i++) {  fout <<
          result[i];

           if  (i != result.size() -  1 ) {  fout <<
                 " " ;
           }
    }

   fout <<  "\n" ;

    fout.close();

    return  0 ;
}

string ltrim( const  string &str) {  string
      s(str);

    s.erase(
           s.begin(),  find_if(s.begin(), s.end(), not1(ptr_fun< int ,  int
          >(isspace)))
    );

    return  s;
}
```

```
string rtrim( const  string &str) {  string
     s(str);

      s.erase(  find_if(s.rbegin(), s.rend(), not1(ptr_fun< int ,
int >(isspace))).base(),
            s.end()  );

      return  s;
}

vector<string> split( const  string &str) {
     vector<string> tokens;

      string::size_type start =  0 ;  string::size_type end
     =  0 ;

      while  ((end = str.find( " " , start)) != string::npos)  {
           tokens.push_back(str.substr(start, end - start));

            start = end +  1 ;
      }

     tokens.push_back(str.substr(start));

      return  tokens;
}
```

OUTPUT :



Compiler Message

```
Success
```

Input (stdin)

```
5 4
1 2 3 4 5
```

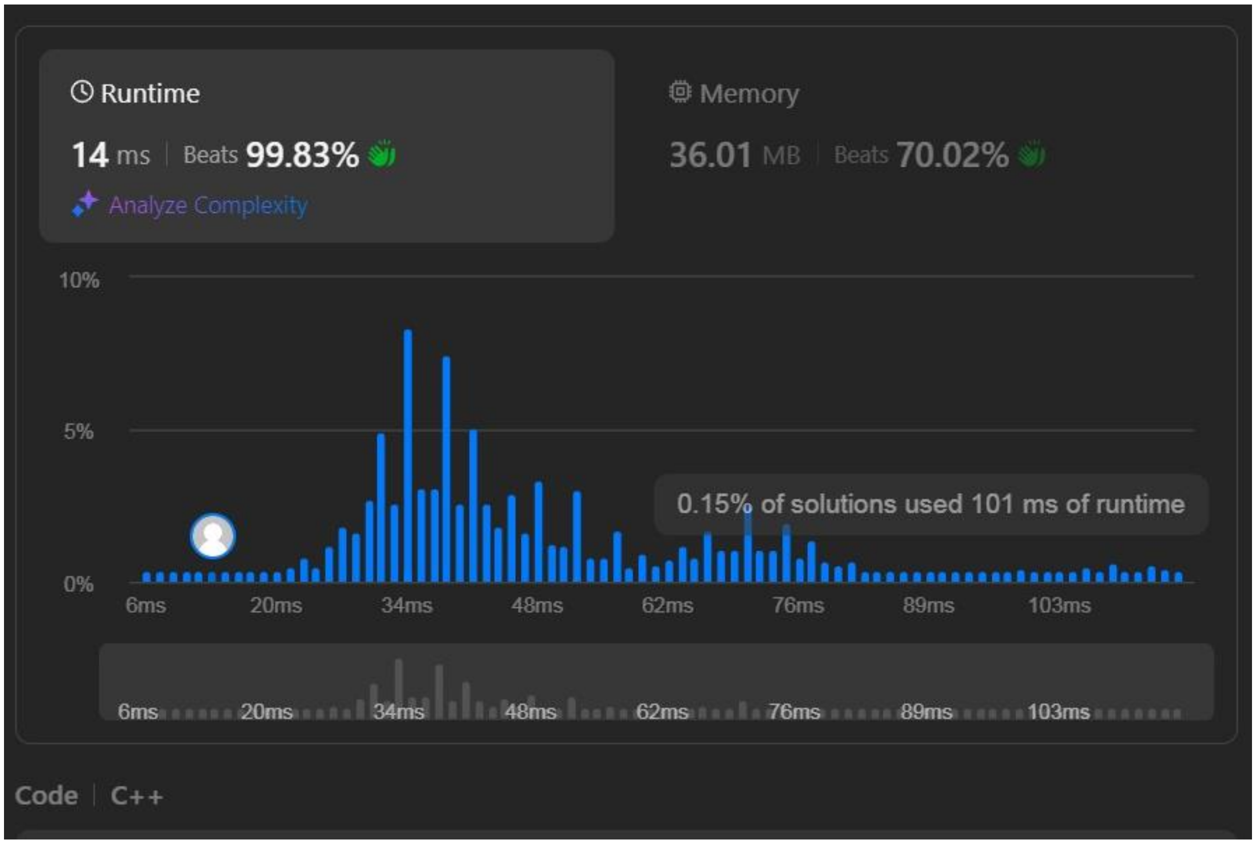
Expected Output

```
5 1 2 3 4
```

1.3 Find All Duplicates in an Array

```
class Solution {  public:

        vector<int> findDuplicates(vector<int>& nums) {
         // check some thing this is really good trick  ios::sync_with_stdio(false);
         cin.tie(nullptr);  cout.tie(nullptr);
          vector<int> v;
         //  if(nums.size()==1)
         //  {
         //     return {};
         //  }
        //  sort(nums.begin(),nums.end());

      // 1 2 2 4
      //



        for(int i=0;i<nums.size();i++)
        {
          int j=abs(nums[i])-1;
           nums[j]*=-1;
          if(nums[j]>0)
           {
                 v.emplace_back(abs(nums[i]));
           }
          // -1 1 2
        }

        // for(int i=1;i<nums.size();i++)
        // {
        //    if(nums[i] <0)
        //    {
        //      v.push_back(abs(nums[i]));
        //    }
        // }
        return v;
    }
};
```

OUTPUT :

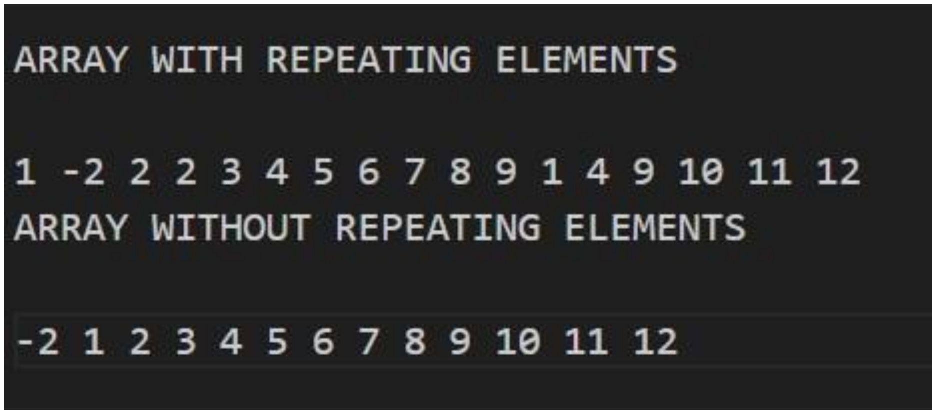


1.4 Write a program to find non repeating element in an array (Unique element). CODE :

```
#include <bits/stdc++.h>  using
namespace std;

void uniqueElements(int arr[],int n)
{
   set<int> st;
   for(int i=0;i<n;i++)
   {
      st.insert(arr[i]);
```

```
    }
 cout <<" \n";
    for(auto i= st.begin();i!=st.end();i++)
    {
     cout << *i <<" ";
    }
 }
  void printArr(int arr[],int n)
  {
     int i=0;
    cout <<"\n";
    while(i <n)
     {
         cout<<arr[i]<<" ";
        i++;
     }
  }
 int main()
 {
 int arr[] = {1,2,2,3,4,5,6,7,8,9,1,4,9};  int arr1[] =
{1,-2,2,2,3,4,5,6,7,8,9,1,4,9,10,11,12};  int n =
sizeof(arr)/sizeof(arr[0]);  int m =
sizeof(arr1)/sizeof(arr[0]);
 // unionOfarray(arr,n,arr1,m);
 cout <<"\nARRAY WITH REPEATING ELEMENTS \n";  printArr(arr1,m);
 cout <<"\nARRAY WITHOUT REPEATING ELEMENTS \n";
uniqueElements(arr1,m);

 return 0;
 }
```

OUTPUT :



```
ARRAY WITH REPEATING ELEMENTS

1 -2 2 2 3 4 5 6 7 8 9 1 4 9 10 11 12
ARRAY WITHOUT REPEATING ELEMENTS

-2 1 2 3 4 5 6 7 8 9 10 11 12
```

Practical-2

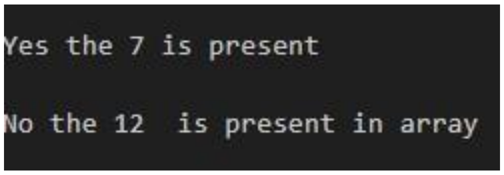
2.1 Implement linear search using iterative and recursive using an array. [VS code]

CODE :

// Using iterative

```
bool isPresent(int arr[],int s,int data)
{
      // binary search  iterative
      // 1) the array must be sorted
    int  lb =0 ,ub =s-1;  int mid;
      while(lb<=ub)
      {
      mid = lb +(ub-lb)/2;  if(arr[mid]==data)
          {
              return true;
          }
          else if(arr[mid]<data)
          {
              lb =mid+1;
          }
          else if(arr[mid]>data)
          {
              ub = mid-1;
          }
      }
      return  false;

}
```

Output:

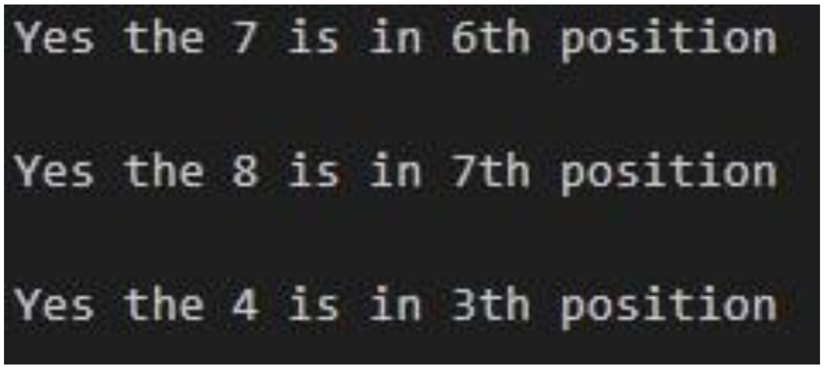


```
Yes the 7 is present

No the 12  is present in array
```

// recursive method

```
int binarySearch(int arr[],int lb,int ub,int data)
{
      int mid = (lb+ub)/2;
       if(lb<=ub)
       {

            if(arr[mid]==data)
            {
                return mid;
            }
            else if( arr[mid]<data)
            {
                    return binarySearch(arr,mid+1,ub,data);
            }
            else if(arr[mid]>data)
            {
              return binarySearch(arr,lb,mid-1,data);
            }


       }
       else{
        return -1;
       }
}
```

OUTPUT :



```
Yes the 7 is in 6th position

Yes the 8 is in 7th position

Yes the 4 is in 3th position
```

2.2 Implement binary search for a long array of integers to find the required element. However, when the array size is quite large, the equation for finding the mid index may give the value which is out of range of integers. Implement the Binary search with a modified equation for finding mid.
“the integer overflow problem” with binary search:

VS code 2

With
a vast list of elements, “right” would be

a very large value.
Suppose your ‘left’ and ‘right’ are 16 bit unsigned integers. That means, they can only have

a maximum value of 2^16 =
65536.

Example:
Consider: left = 65530 and right
= 65531

left
+ right = 131061 (beyond the integer range). It will give garbage value It is known as an integer overflow.

ANS :

MODIFIED FORMULA : mid = lb +(ub-lb)/2;

```
bool isPresent(int arr[],int s,int data)
{
      // binary search  iterative
      // 1) the array must be sorted
    int  lb =0 ,ub =s-1;  int mid;
      while(lb<=ub)
      {
      mid = lb +(ub-lb)/2;  if(arr[mid]==data)
          {
              return true;
          }
          else if(arr[mid]<data)
          {
              lb =mid+1;
          }
          else if(arr[mid]>data)
          {
              ub = mid-1;
          }
      }
      return  false;

}
```

2.3 Kth Missing Positive Number

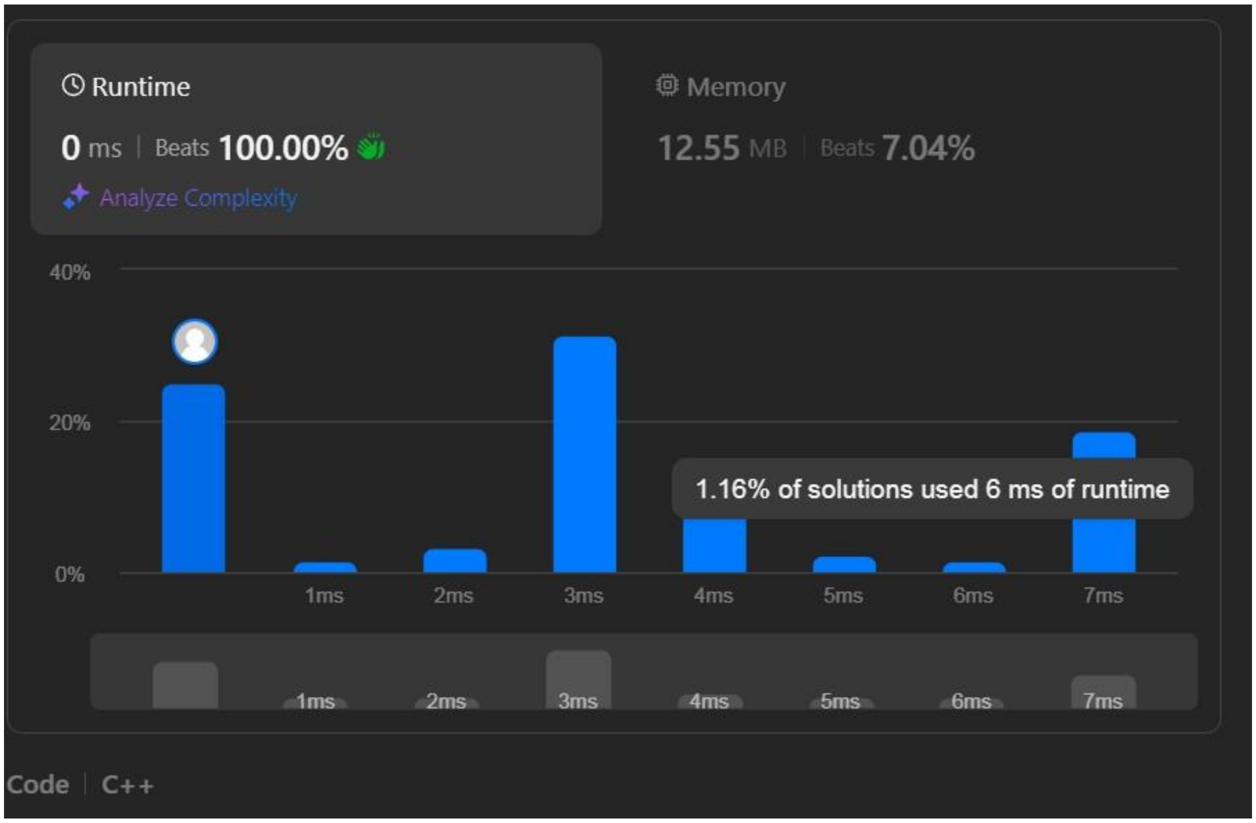
CODE :

```
class Solution {  public:
```

```
int findKthPositive(vector<int>& arr, int k) {
    auto            fastio=[]()                    {
    std::ios::sync_with_stdio(false);
    cin.tie(nullptr);   cout.tie(nullptr);   return
    nullptr;
   }();//
   //brute force
   //   int p=k;
   //   int q=0;
   //   for(int i=1;k>0;i++)
   //   {
   //    q=0;
   //    for(int j=0;j<arr.size();j++)
   //    {
   //        if(i==arr[j])
   //        {
   //            q=1;
   //        }
   //    }
   //    if(q==0)
   //    {
   //       p=i;
   //        k--;
  //    }

   //   }



   //   return p;
     //better  int lb =0 ,ub
  =arr.size()-1;  int mid = lb
  +(ub -lb)/2;  while(lb
  <=ub)
      {
```

```
                mid = lb +(ub -lb)/2;  int a

               = arr[mid] - (mid+1);

                if(a >= k)
                {
                 ub =mid-1;
                }
                else
                {
                lb =mid+1;
                }

            }
            return k+ub+1;
    }
};
```

OUTPUT :

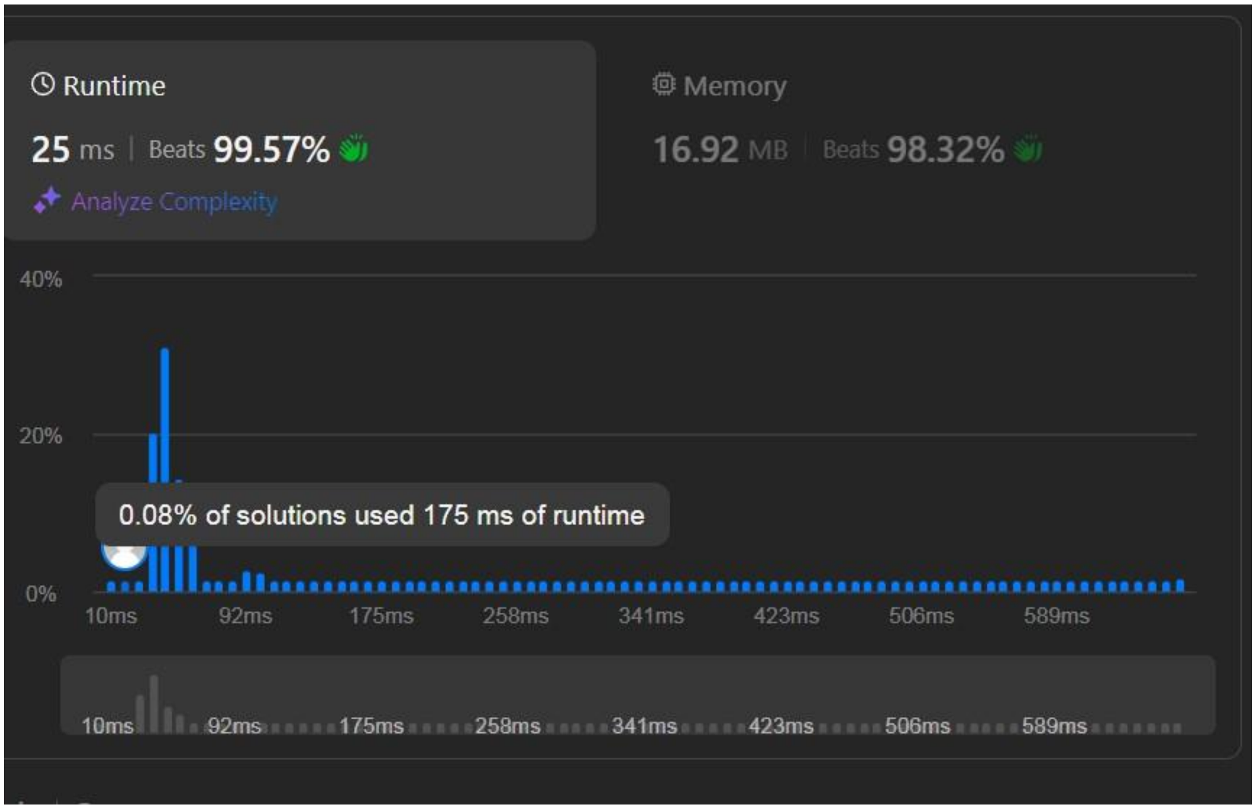

Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
12.55 MB | Beats 7.04%
40%
20%
0%
1ms
2ms
3ms
4ms
5ms
6ms
7ms
1.16% of solutions used 6 ms of runtime
Code | C++

2.4 Search a 2D Matrix II

CODE : class

```
Solution {
public:
        bool searchMatrix(vector<vector<int>>& matrix, int target) {
        ios::sync_with_stdio(false);
       cin.tie(nullptr);
       cout.tie(nullptr);  int row =
       matrix.size();  int col =
       matrix[0].size();  int r = 0;  int
       c = col - 1;  while (r < row
       && c >= 0) {  if (matrix[r][c]
       == target) {  matrix.clear();
       return true;
            } else if (matrix[r][c] < target) {

               r++;
            } else {
              c--;  r
              = 0;
            }
        }
        matrix.clear();  return
       false;
    }
};
```

OUTPUT :


Runtime
25 ms | Beats 99.57%
Analyze Complexity
Memory
16.92 MB | Beats 98.32%
40%
20%
0%
0.08% of solutions used 175 ms of runtime
10ms
92ms
175ms
258ms
341ms
423ms
506ms
589ms

Practical-3

3.1 Implement Sorting Algorithm(s).

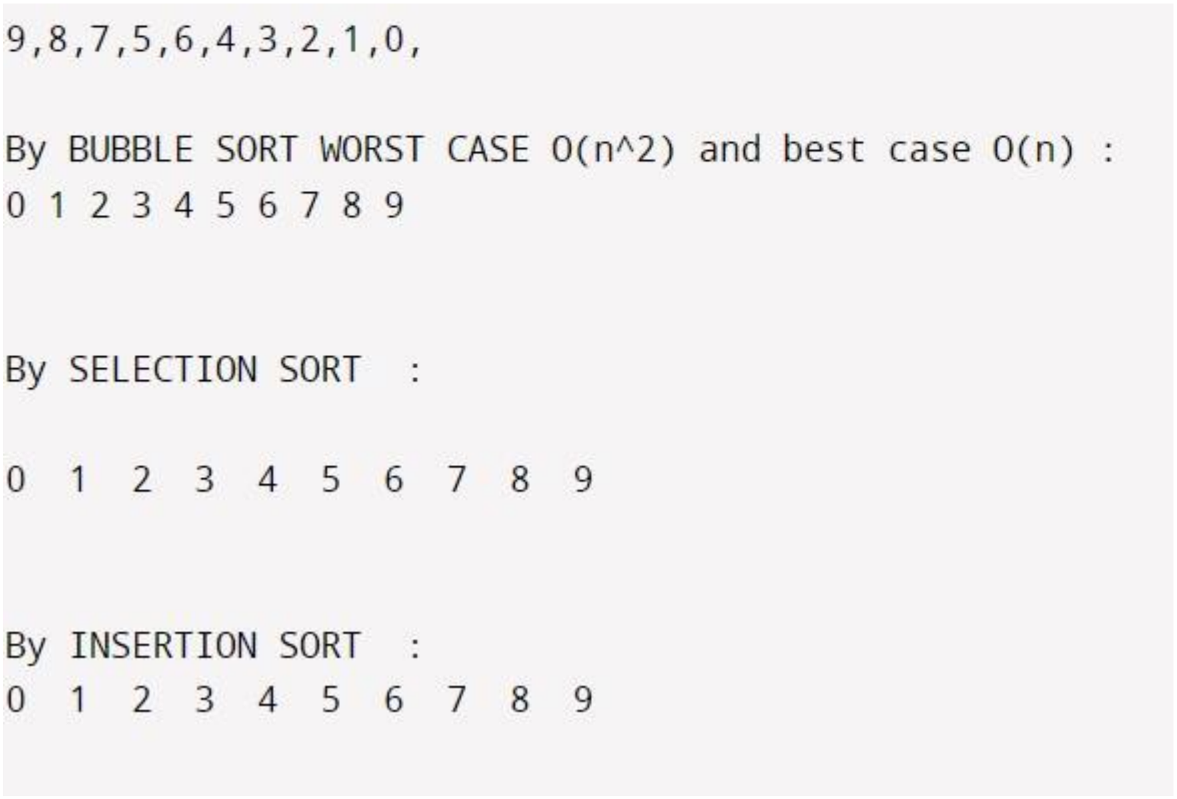
- **a) Bubble Sort**
- **b) Selection sort**
- **c) Insertion sort CODE:**

```
#include<bits/stdc++.h>  using namespace

std;

   void bubble_sort(int arr[],int s)
{
      cout <<endl;
      //
        int flag =0;  for(int i=0;i<s-
       1;i++)
        {
            flag=0;
             for(int j=1;j<s-i;j++)
             {
                   if(arr[j-1]>arr[j])
                   {
                           int temp;  temp = arr[j-1];
                          arr[j-1] = arr[j];  arr[j]
                          =temp;  flag =1;
                   }


             }
              if(flag ==0)
                 {
                               break;
                 }
        }

        for(int i=0;i<s;i++)
        {

            cout << arr[i]<<" ";
        }
        cout <<endl;
}

void selection_sort(int arr[],int s)
{
      cout <<endl;
     int m =arr[0];
        for(int i=0;i<s-1;i++)
        {
                 m = arr[i];
             for(int j=i+1;j<s;j++)
             {
                           if(m>arr[j])
                   {
                          int temp;  temp = m;
                          arr[i] = arr[j];
                                  m =temp;

                   }


             }

        }
        for(int i=0;i<s;i++)
        {

            cout << arr[i]<<"  ";
        }
        cout <<endl;

}

void selection_sort1(int arr[],int s)
{
      cout <<endl;
```

```
    int minIndex =0;
      for(int i=0;i<s-1;i++)
      {
            minIndex = i;
         for(int j=i+1;j<s;j++)
         {
             if(arr[minIndex]>arr[j])
             {
                         minIndex =j;
             }


         }
         if(minIndex != i)
         {
            int temp;  temp = arr[i];  arr[i] =
                 arr[minIndex];
                              arr[minIndex]=temp;


         }

      }
      for(int i=0;i<s;i++)
      {

         cout << arr[i]<<"  ";
      }
      cout <<endl;

}
void minimum(int arr[],int s)
{

    int m = arr[0];

    for(int i=0;i<s;i++)
    {

       if(m>arr[i])
```

```
         {
            int temp;  temp =m;
            m= arr[i];  arr[i] =temp;

         }
     }
     cout <<"MINIMUM : "<<m<<endl;
}

void insertion_sort(int arr[],int s)
{
   cout <<endl;
    for(int i=1;i<s;i++)
    {

       int temp = arr[i],j;  for(j=i-1;j>=0;j--)
       {
 // 9 8 7
 // best case O(n)
 // worst case O(n^2)  if(arr[i]<arr[j])
            {
                        arr[i] = arr[j];
            }
            else{  break;
            }
       }

          arr[i] = temp;
    }

     for(int i=0;i<s;i++)
       {

           cout << arr[i]<<"  ";
       }
       cout <<endl;

}
void insertion_sort_with_while(int arr[],int s)
{
    int i=1,j=i-1;
     while(i<s)
     {
          int temp =arr[i];

          while(j>=0 && arr[i]>arr[j])
          {
              if(arr[i]<arr[j])
              {

                  arr[i] = arr[j];
              }


                     j--;
          }
          temp =arr[i];
             i++;
     }
for(int i=0;i<s;i++)
       {

           cout << arr[i]<<"  ";
       }
       cout <<endl;
}
 int main()
 {
     int arr[10] ={9,8,7,5,6,4,3,2,1,0};  cout <<endl;

 for(int i=0;i<10;i++)
       {

           cout << arr[i]<<",";
       }
       cout <<endl;
         cout <<endl;
        cout<<"By BUBBLE SORT WORST CASE O(n^2) and best case O(n) :";
    bubble_sort(arr,10);
       cout <<endl;  cout <<endl;
    // minimum(arr,10);  cout<<"By SELECTION
      SORT  :";
        cout <<endl;
    // selection_sort(arr,10);  selection_sort1(arr,10);

      cout <<endl;
      cout <<endl;
    // minimum(arr,10);  cout<<"By INSERTION
      SORT  :";
        cout <<endl;
    // selection_sort(arr,10);
  //   insertion_sort(arr,10);  insertion_sort_with_while(arr,10);

 }
```

OUTPUT :

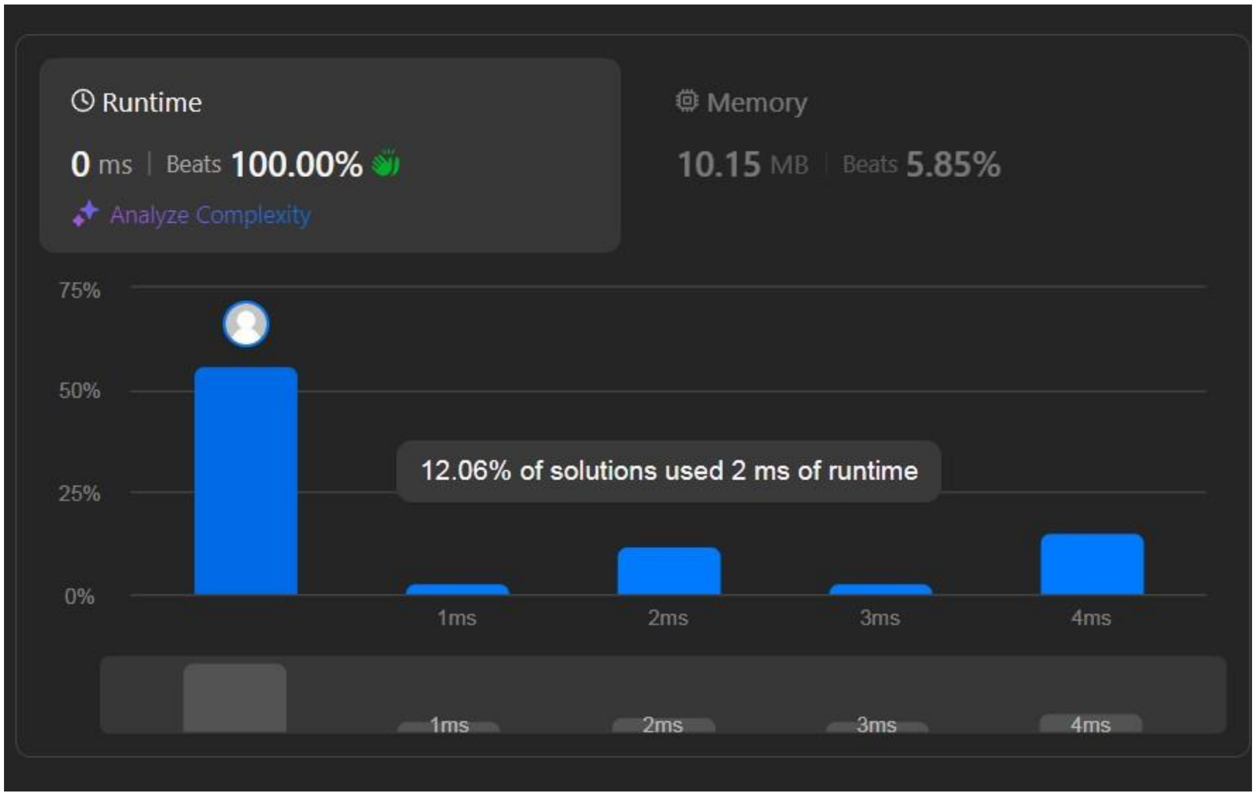


```
9,8,7,5,6,4,3,2,1,0,

By BUBBLE SORT WORST CASE O(n^2) and best case O(n) :
0 1 2 3 4 5 6 7 8 9


By SELECTION SORT   :

0   1   2   3   4   5   6   7   8   9


By INSERTION SORT   :
0   1   2   3   4   5   6   7   8   9
```

3.2 Sort-colors (Leetcode) CODE :

```
 #include <bits/stdc++.h>  using
namespace std;  class Solution {
public:

                      void sortColors(vector<int>& nums) {
  // brute force
        // for(int i=0;i<nums.size();i++)
        // {
        //     for(int j=i+1;j<nums.size();j++)
        //     {
        //         if(nums[i]>nums[j])
        //            {
        //             int temp;
        //               temp = nums[i];
        //               nums[i]=nums[j];
        //               nums[j]=temp;
        //               }
        //     }
        // }
     //    Better  int left=0;  int
    right=nums.size()-1;  int mid=left;

     for(int i=0;i<nums.size();i++)
     {
        if(nums[mid]==0 || nums[mid]==2)
        {
            if(nums[mid]==0)
            {
                                swap(nums[mid],nums[left]);
                left++;  mid++;
            }
            else if(nums[mid]==2)
            {
                                swap(nums[mid],nums[right]);
                right--;
            }

        }
        else
        {
            mid++;
        }
    }
    // return nums;
    }
};
int main() {

    // Write C++ code here
    Solution s;
   vector<int> v ={2,0,2,1,0,1};
  s.sortColors(v);  for(auto it : v)
   {
       cout<<it<<" ";
   }



    return 0;
}
```

OUTPUT :


Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

10.15 MB | Beats 5.85%

12.06% of solutions used 2 ms of runtime

Practical - 4

4.1 Implement following operations of singly linked list.

```
 #include <bits/stdc++.h>
using namespace std;
 //linked list
 // node
 // start --> [1 |address]-->[2 |address]-->[3 |address]-->[4 |nullptr]  struct node
 {
           int data;  node *next; // structure referencing to itself
 }*start=nullptr;


 void inputLL()
 {
           int data;  cout <<"Enter Data :";  cin >> data;

           while(data !=-1)
           {


           node *nextnode =new node();  nextnode->data=data;
          nextnode->next=nullptr;  if(start==nullptr)
                     {
                                           start=nextnode;
                     }
                     else
                     {
                               node *ptr=start;  while(ptr->next!=nullptr)
                               {
                                                     ptr=ptr->next;
                               }
                                          ptr->next=nextnode;
                     }
```

```
                    cin >> data;
        }


}
void removeformFront()
{
        if(start==nullptr)
        {
                cout <<"list is empty";
        }
        else
        {
                node *ptr=start;  start=ptr->next;  delete ptr;

        }
}
void removefromEnd()
{
        if(start==nullptr)
        {
                cout <<"LL is EMPTY!!!\n";
        }
        else
        {
                node * prePtr=nullptr;  node *ptr= start;
                        ptr =start->next;
                while(ptr->next!=nullptr)
                {
                        prePtr =ptr;  ptr=ptr->next;
                }
                prePtr->next=nullptr;
          delete ptr;
        }
}
```

```
void insertformFront(int data)
{
        if(start==nullptr)
        {
                cout <<"list is empty";
        }
        else
        {
                node *ptr=new node();  ptr->next=start;  ptr-
               >data=data;  start=ptr;

        }
}
void Print()
{
        node *ptr =start;  cout<<"Start-->";
       while(ptr!=nullptr)
        {
                cout <<ptr->data<<"-->";  ptr=ptr->next;
        }
                               cout<<"NullPtr";
}

void print(node *nextnode)
{
        if(nextnode ==nullptr)
        {
                                return ;
        }
        print(nextnode->next);  cout<<nextnode->data<<" ";
}
void reverseLL()
{
        node *nextPtr =start;  stack<int> st;
       while(nextPtr !=nullptr)
```

```
        {
                                        st.push(nextPtr->data);
                nextPtr=nextPtr->next;
        }
        nextPtr =start;
        while(!st.empty())
        {
                 nextPtr->data = st.top();  st.pop();
                nextPtr = nextPtr->next;
        }
}
void reversePrint()
{
        print(start);
}

int main()
{
        int p=0,a;  while(p!=9){  cout <<"\n::::::::::LINKED
       LIST::::::::::\n";  cout<<"1) insert in LL (-1 to end)\n";
       cout<<"2) Traverse in LL\n";  cout<<"3) display LL\n";
       cout<<"4) Remove from front\n";  cout<<"5) Insert form
       Front \n";  cout<<"6) Remove from End \n";  cout<<"7)
       Reverse print  \n";  cout<<"7) Reverse Linked_List  \n";
       cout<<"9) Exit\n";
        cout <<"::::::::::::::::::::::::::\n";  cin>>p;  switch(p)
        {
                case 1:

                        inputLL();  break;
                case 2:  Print();  break;
                case 3:
                 Print();  break;

                case 4:
                removeformFront();  break;
                case 5:
                insertformFront(20);  break;
                case 6:
                  removefromEnd();  break;
                case 7:
                  reversePrint();  break;
                case 8:
                  reverseLL();  break;

        }
        }
  return 0;
}
```

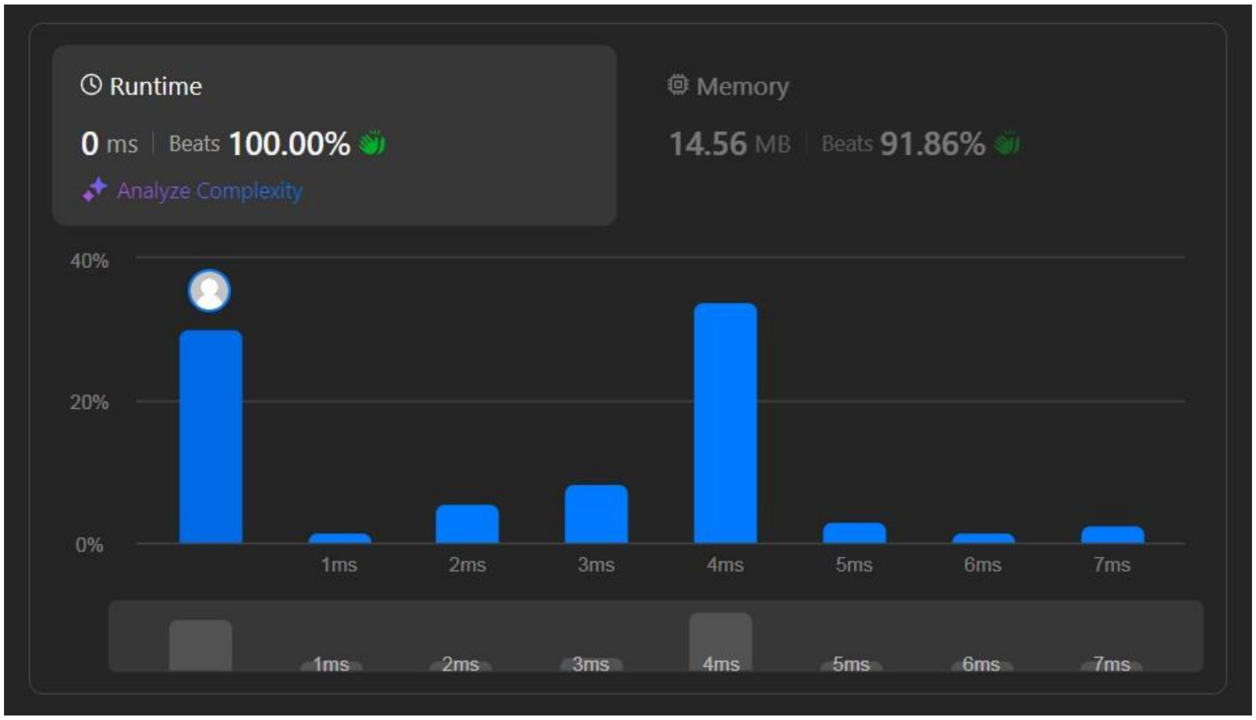
OUTPUT :

4.2 Remove Nth Node from End of List

```
int length(ListNode *head){  int len =0;
      while(head !=nullptr){  len++;  head =head-
      >next;
       }
       return len;
   }
   ListNode* removeNthFromEnd(ListNode* head, int n) {  if(head->next
     ==nullptr)return nullptr;
        int len = length(head);  int k =len-n-1;
       if(len==n)return head->next;

        ListNode * ptr =head;  while(k>0){  ptr
       =ptr->next;
              k--;
        }
```

```
                    if(ptr->next!=nullptr)  ptr->next =ptr->next->next;
        return head;
    }
```

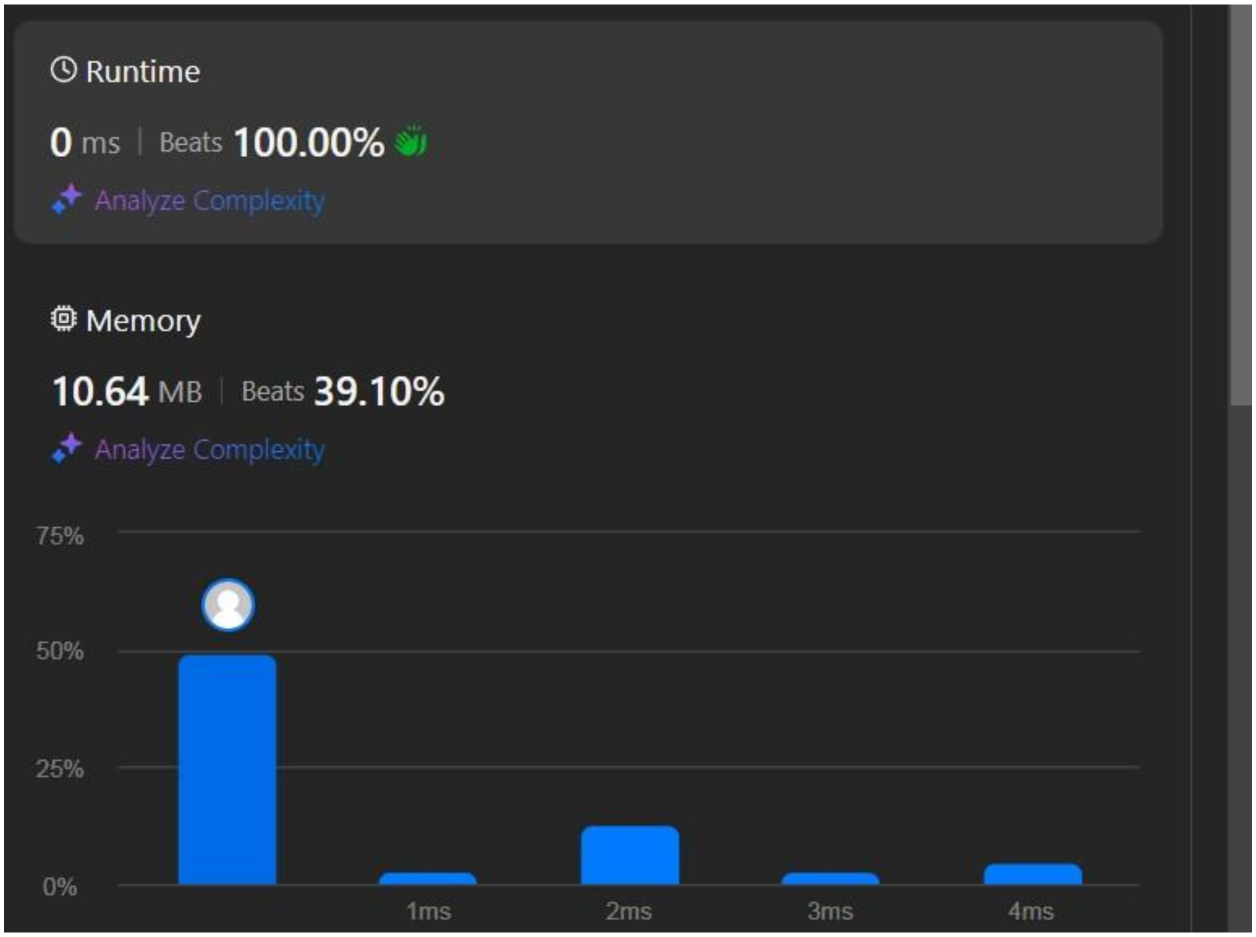
OUTPUT :



4.3 Swap Nodes in Pairs

```
ListNode* swapPairs(ListNode* head) {  if (!head || !head-
      >next) {
          return head;
       }

       ListNode* t = swapPairs(head->next->next);  ListNode* p =
      head->next;  p->next = head;  head->next = t;  return p;
    }
```

OUTPUT :


Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

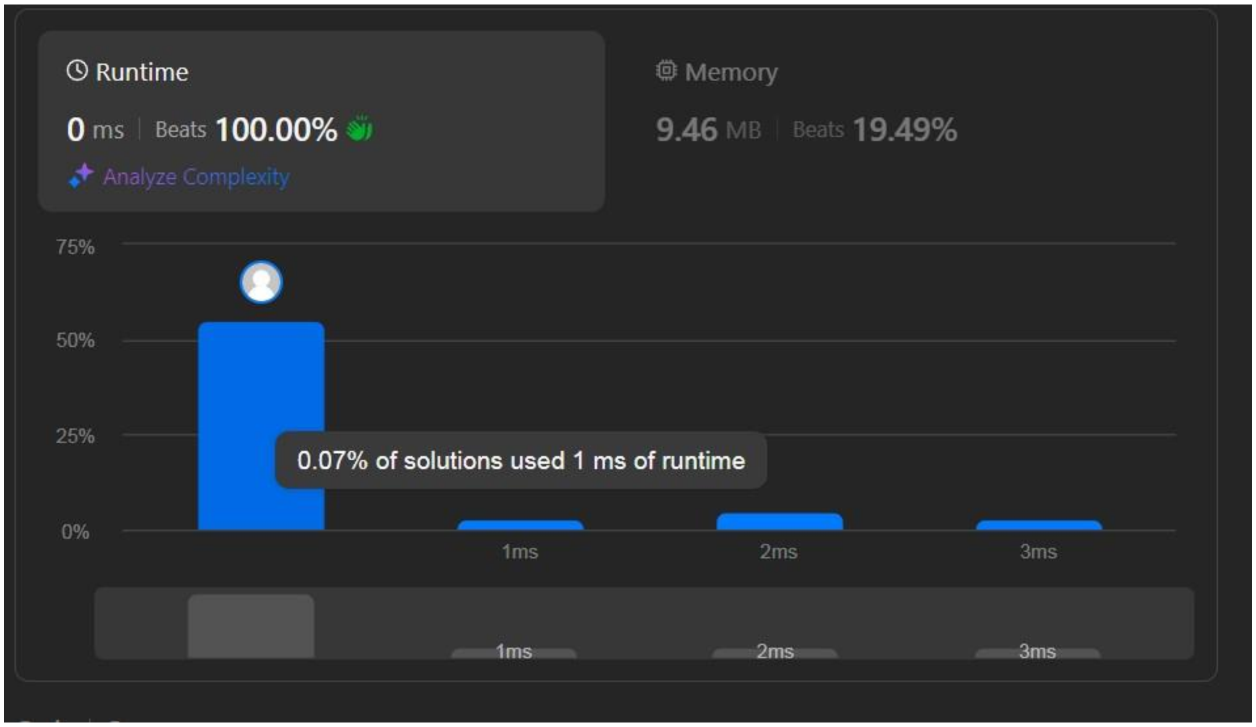
10.64 MB | Beats 39.10%

Analyze Complexity

4.4 Middle of the Linked List

```
ListNode* middleNode(ListNode* head) {

      ios::sync_with_stdio(false);  cin.tie(nullptr);
     cout.tie(nullptr);
         ListNode * slow=head;  ListNode * fast=head;
         while(fast !=nullptr && fast->next != nullptr)
         {
            slow=slow->next;
           fast =fast->next->next;
         }
         return slow;
   }
```

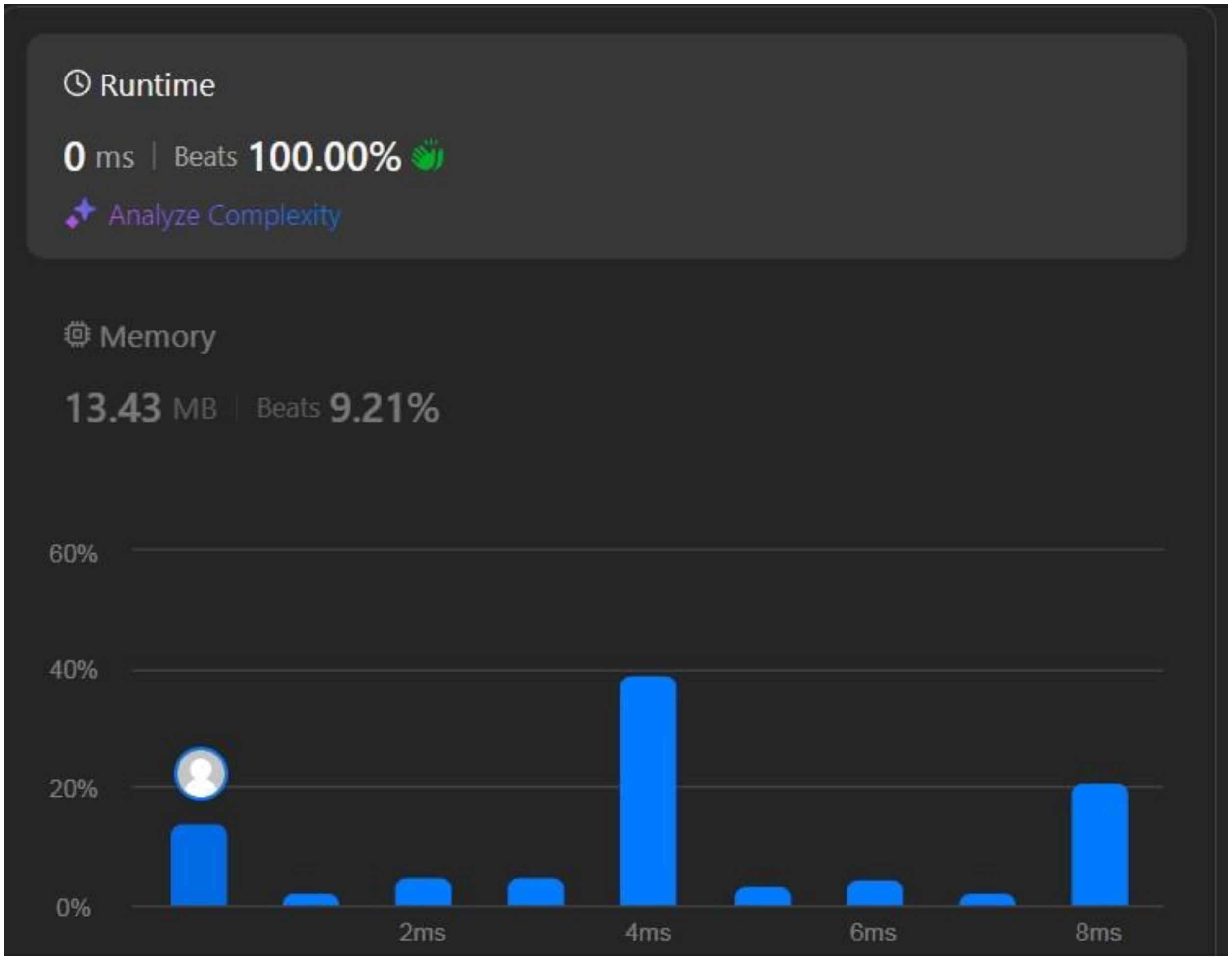
OUTPUT :



4.5 Reverse Linked List

```
void reverse(ListNode *&head){  ListNode *ptr =nullptr;
   while(head !=nullptr){
       ListNode * node =new ListNode(head->val,ptr);  ptr =node;  head
      =head->next;
    }
    head=ptr;
  }
 ListNode* reverseList(ListNode* head) {  reverse(head);
   return head;
 }
```

OUTPUT :



Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

13.43 MB | Beats 9.21%

Practical - 05

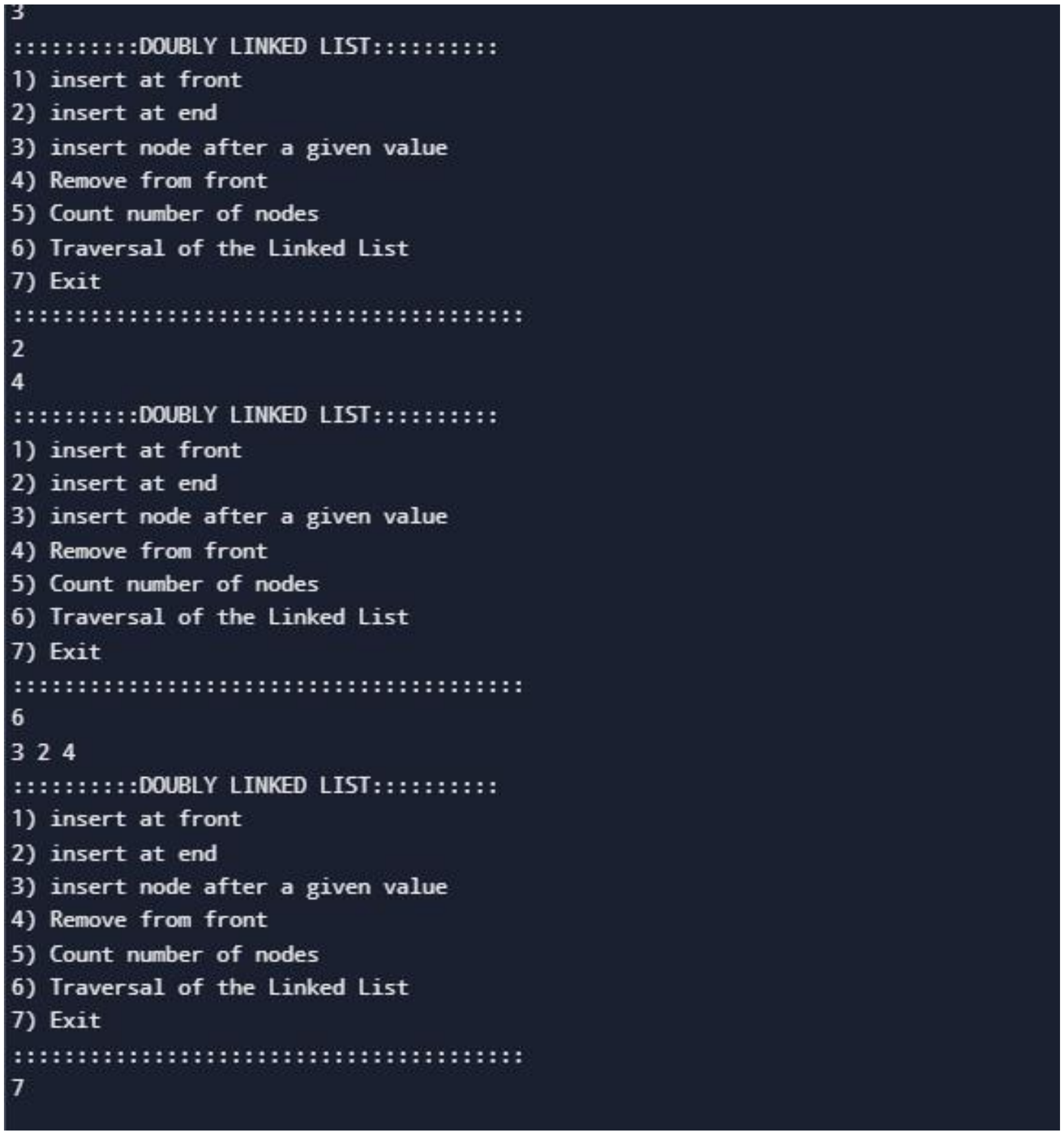
5.1 :

AIM : Implement following operations of doubly linked list.
(a) Insert a node at front
(b) Insert a node at end
(c) Insert a node after given node information
(d) Delete a node at front
(e) Count number of nodes (f) Traversal of the Linked List

CODE :

```
#include <bits/stdc++.h>  using namespace std;

struct Node
{
        int data;  Node *next;
        Node *prev;
        Node(int data){  this->data = data;  next
                =nullptr;  prev = nullptr;
        }
}*head=nullptr;

void insertEnd(int data)
{
        Node *node =new Node(data);  if(head ==nullptr)head
        =node;  else{
                Node *ptr =head;  while(ptr->next!=nullptr)ptr =ptr->next;
                ptr->next =node;  node->prev =ptr;
        }
}
void insertFront(int data)
{
        Node *node =new Node(data);  if(head ==nullptr)head
        =node;  else{
                node->next = head;  head->prev =node;  head
               =node;
        }
}
void insert(int data,int ahead)
{
        Node *node =new Node(data);  if(head ==nullptr)head
       =node;  else{
                Node *ptr =head;  while(ptr->data!=ahead)ptr =ptr->next;
               node->next = ptr->next;  ptr->next->prev =node;  ptr->next
               =node;  node->prev =ptr;
        }
}
void deleteFront(){  if(head ==nullptr)return;  head
       =head->next;  head->prev =nullptr;
}
int countNodes(){  Node *ptr =head;  int len =0;
               while(ptr!=nullptr){  ptr =ptr->next;  len++;
                }
                return len;
}

void display()
{
        if(head==nullptr){
                        cout<<"LL is Empty!!!\n";  return;
                }
        Node *ptr =head;  while(ptr!=nullptr){  cout<<ptr-
       >data<<" ";  ptr=ptr->next;
        }
        cout<<"\n";
}

// swap Node of LL as well as data  int main()
{
 int p=0,a,data,ahead;  while(p!=7){  cout <<"\n::::::::::DOUBLY
LINKED LIST::::::::::\n";  cout<<"1) insert at front\n";  cout<<"2) insert
at end \n";  cout<<"3) insert node after a given value\n";  cout<<"4)
Remove from front\n";  cout<<"5) Count number of nodes\n";  cout<<"6)
Traversal of the Linked List\n";  cout<<"7) Exit\n";
 cout <<":::::::::::::::::::::::::::::::::::::::\n";  cin>>p;  switch(p)
 {
          case 1:

                    cin>>data;
                    insertFront(data);  break;
          case 2:

                    cin>>data;  insertEnd(data);  break;
          case 3:

                                                  cin>>ahead>>data;
           insert(data,ahead);  break;
          case 4:
            deleteFront();  break;
          case 5:
            cout<<countNodes();  break;
          case 6:
            display();  break;
 }
                              cout<<"23dit071 ANSH SINGH\n";
 }

 return 0;
}
```

OUTPUT :



```
3
::::::::::DOUBLY LINKED LIST::::::::::
1) insert at front
2) insert at end
3) insert node after a given value
4) Remove from front
5) Count number of nodes
6) Traversal of the Linked List
7) Exit
::::::::::::::::::::::::::::::::::::::::
2
4
::::::::::DOUBLY LINKED LIST::::::::::
1) insert at front
2) insert at end
3) insert node after a given value
4) Remove from front
5) Count number of nodes
6) Traversal of the Linked List
7) Exit
::::::::::::::::::::::::::::::::::::::::
6
3 2 4
::::::::::DOUBLY LINKED LIST::::::::::
1) insert at front
2) insert at end
3) insert node after a given value
4) Remove from front
5) Count number of nodes
6) Traversal of the Linked List
7) Exit
::::::::::::::::::::::::::::::::::::::::
7
```

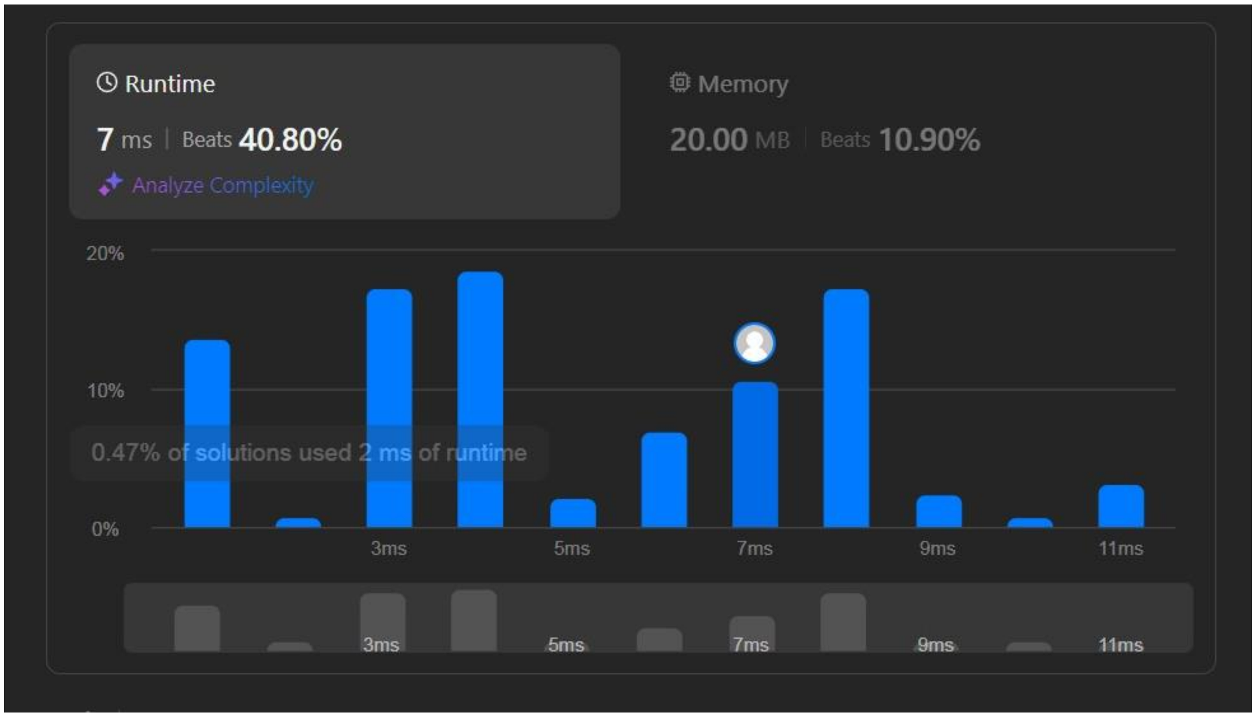
Practical Assignment

5.2 Merge Two Sorted Lists
https://leetcode.com/problems/merge-two-sorted-lists/

CODE ::

```
ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
     ios::sync_with_stdio(false);  cin.tie(nullptr);
    cout.tie(nullptr);  // edge casess  if (list1 == nullptr &&
    list2 == nullptr) {
        return list1;
     } else if (list1 == nullptr) {  return list2;
     } else if (list2 == nullptr) {  return list1;
     }
     // left node right node

   ListNode left;
   ListNode *ptr = &left;

     // left

     while(list2 !=nullptr && list1 != nullptr ){

        if(list1->val <= list2->val){  ptr->next =list1;
            list1 =list1->next;
        }
        else {  ptr->next =list2;  list2 =list2->next;
        }

       ptr =ptr->next;
     }
     if(list1 != nullptr)
     {
 ptr->next = list1;
     }
     else if(list2 != nullptr){

ptr->next = list2;
     }
   return left.next;
  }
```

OUTPUT :



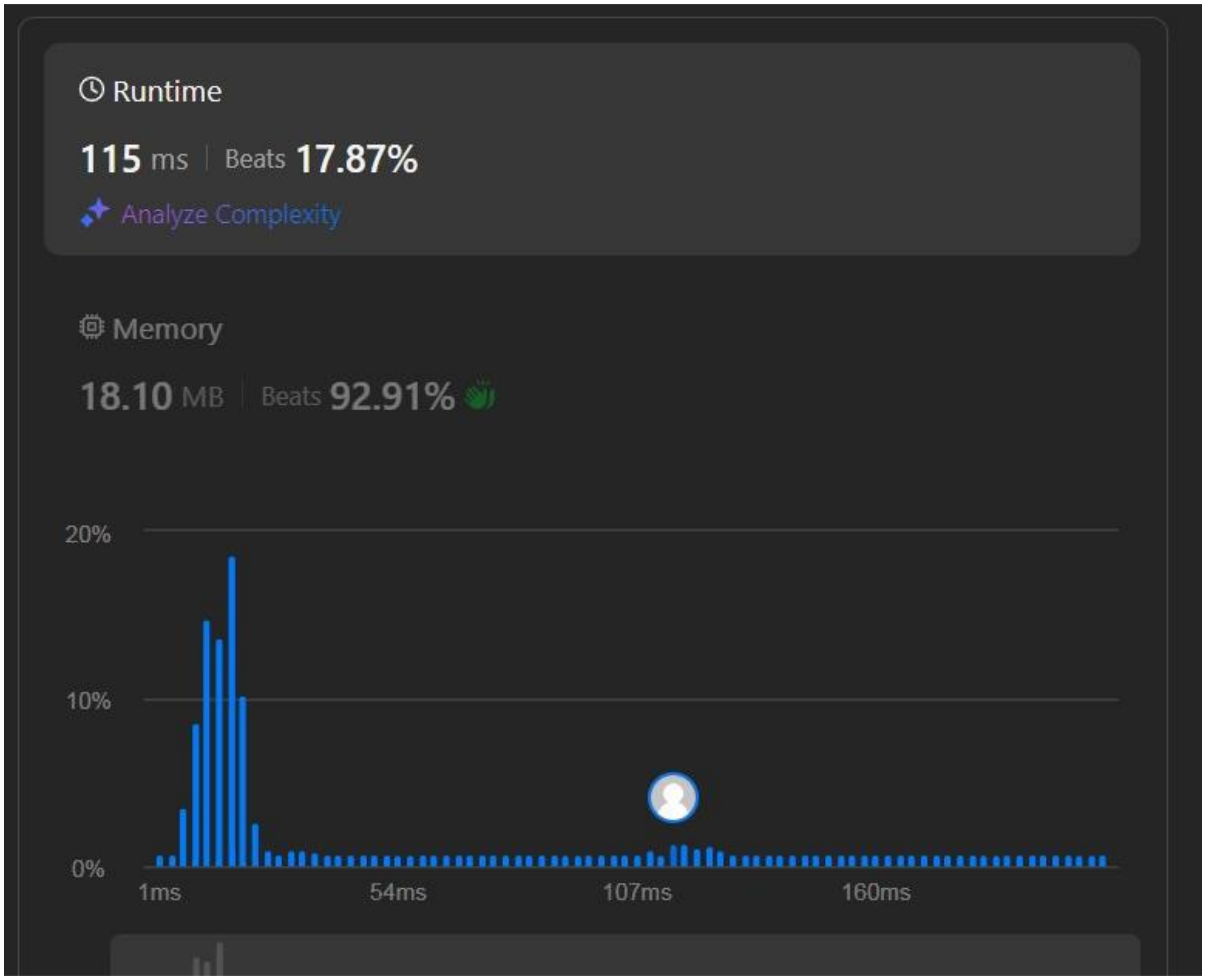
5.3 Merge k Sorted Lists https://leetcode.com/problems/merge-k-sorted-lists/

CODE :

```
 ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
       // edge casess  if (list1 == nullptr && list2 == nullptr) {
           return list1;
       } else if (list1 == nullptr) {  return list2;
       } else if (list2 == nullptr) {  return list1;
       }
       // left node right node

       ListNode left;  ListNode* ptr = &left;  while (list2 != nullptr

      && list1 != nullptr) {

           if (list1->val <= list2->val) {
               ptr->next = list1;  list1 = list1->next;
```

```
            } else {  ptr->next = list2;  list2 = list2-
              >next;
            }

            ptr = ptr->next;
        }
        if (list1 != nullptr) {
            ptr->next = list1;
        } else if (list2 != nullptr) {

            ptr->next = list2;
        }
        return left.next;
    }
    ListNode* mergeKLists(vector<ListNode*>& lists) {  if (lists.size()
       == 0)  return nullptr;
        if (lists.size() == 1)
            return lists[0];
        ListNode* mergelist = nullptr;
        for (auto list : lists) {
                                        mergelist = mergeTwoLists(list, mergelist);
        }
        return mergelist;
    }
```

OUTPUT :


Runtime

115 ms | Beats 17.87%

Analyze Complexity

Memory

18.10 MB | Beats 92.91%

Practical-6

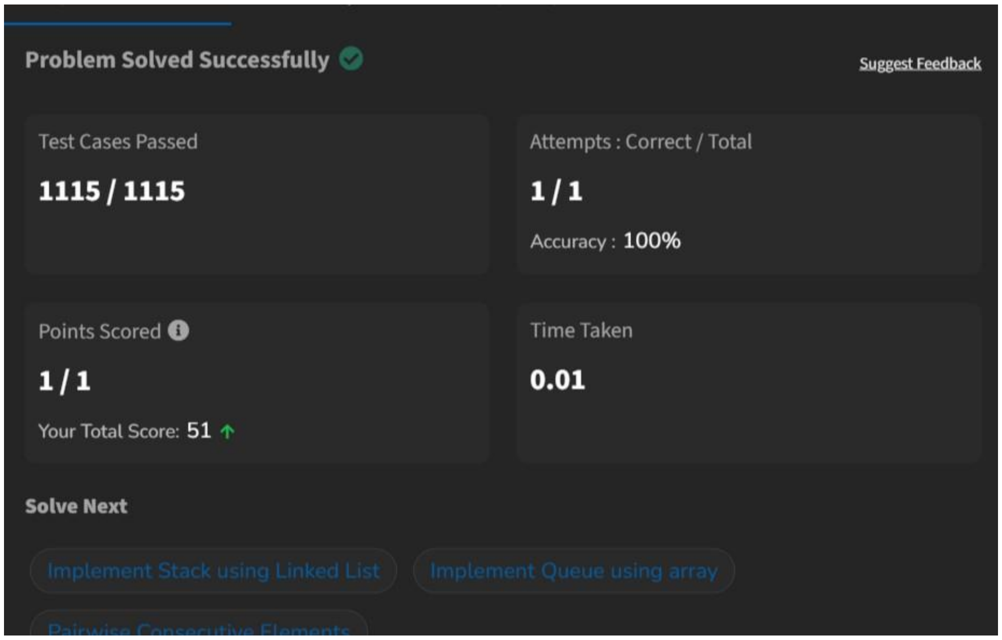
6.1 Implement stack using array (static stack).

https://practice.geeksforgeeks.org/problems/implement-stack-using-array/1

CODE :

```
void MyStack ::push(int x) {
    // Your Code  if(top==-1)top=0;
   arr[++top]=x;



}

// Function to remove an item from top of the stack.
int MyStack ::pop() {  // Your Code  if(top==-1)return
-1;  return arr[top--];
}
```

OUTPUT :



Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
1 / 1
Your Total Score: 51

Time Taken
0.01

Solve Next

Implement Stack using Linked List
Implement Queue using array
Pairwise Consecutive Elements

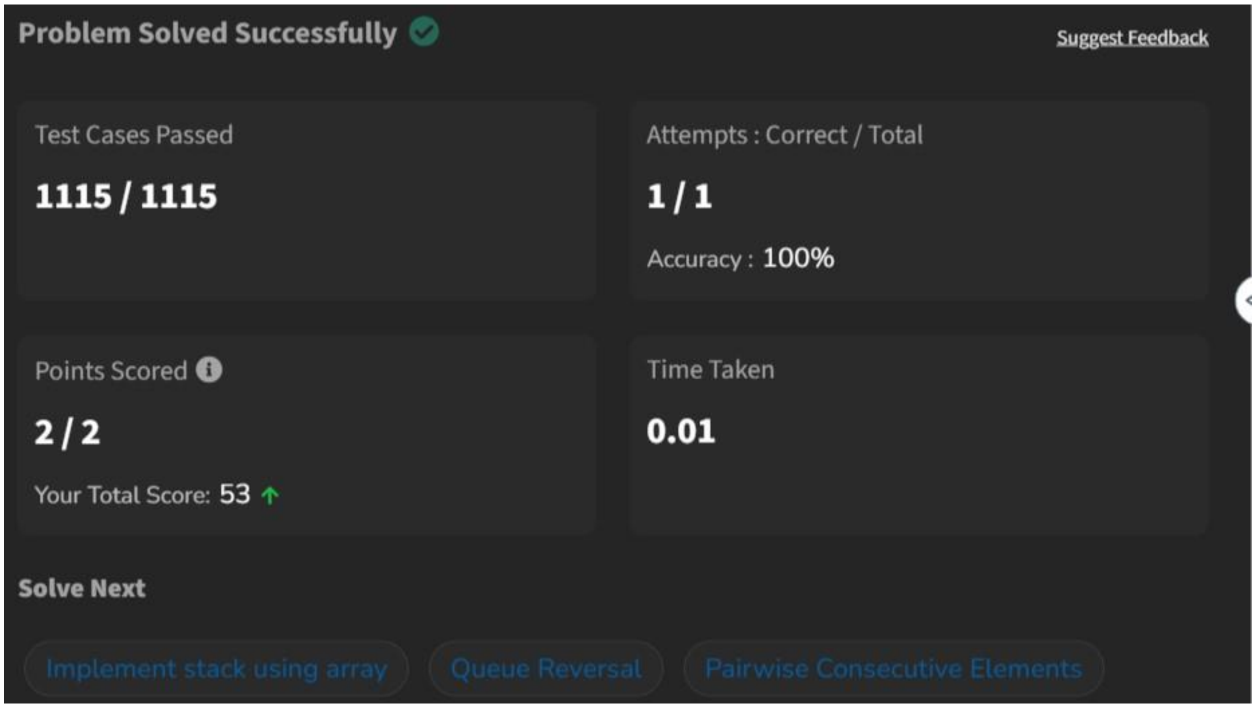
6.2 Implement stack using linked list (dynamic stack).

https://practice.geeksforgeeks.org/problems/implement-stack-using-linked-list/1

CODE :

```
void push(int x) {
        StackNode* node = new StackNode(x);  node->next

       = top;

        top = node;
    }

    int pop() {  if (top ==
       NULL) {

            return -1;
        }


        int poppedData = top->data;


        StackNode* temp = top;  top
       = top->next;


        delete temp;  return
       poppedData;
    }
```

OUTPUT :



Problem Solved Successfully

Suggest Feedback

Test Cases Passed

1115 / 1115

Attempts : Correct / Total

1 / 1

Accuracy : 100%

Points Scored

2 / 2

Your Total Score: 53

Time Taken

0.01

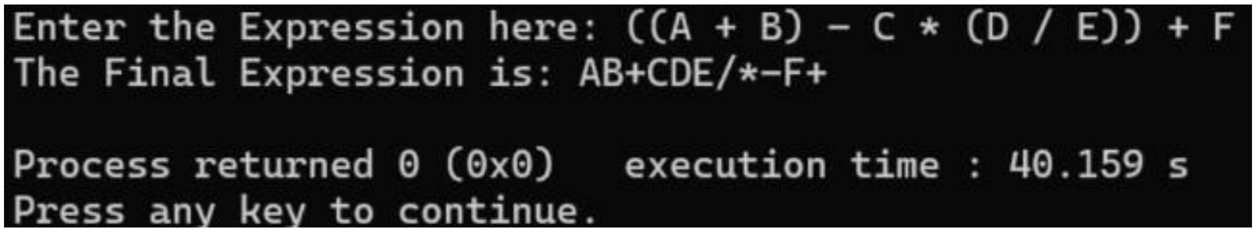
Solve Next

Practical - 7

7.1 Convert Infix to Postfix CODE:

```
 #include<bits/stdc++.h>
 #include<iostream>
#include<stack>  using namespace
std;

 int prec(char c){  if(c == '^'){
return 3;
 } else if(c == '*' || c == '/'){  return 2;
 } else if(c == '+' || c == '-'){  return 1;
} else {  return -1;
 }
 }

 void infixToPostfix(string infix){

stack<char> st;  string res;  for(int i=0;

i<infix.size(); i++){  char c = infix[i];  if(c

== ' ') {

 continue;
 }

 if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9')){  res +=
c;
 }
 else if(c == '('){
st.push(c);
 }
 else if(c == ')'){  while(!st.empty() && st.top()
!= '('){  res += st.top();  st.pop();
 }
 if(!st.empty()) {  st.pop();
 }
 }
 else{  while(!st.empty() && prec(c) <= prec(st.top())){  res
+= st.top();  st.pop();
 }
 st.push(c);
 }
 }

 while(!st.empty()){  res +=
st.top();  st.pop();

 }

 cout << "The Final Expression is: " << res << endl;
 }

 int main(){  string infix;  cout << "Enter the
Expression here: ";  getline(cin, infix);
infixToPostfix(infix);  return 0;
}
```

OUTPUT :



```
Enter the Expression here: ((A + B) - C * (D / E)) + F
The Final Expression is: AB+CDE/*-F+

Process returned 0 (0x0)   execution time : 40.159 s
Press any key to continue.
```

7.2 Evaluate Reverse Polish Notation PROBLEM LOGIC:

```
class Solution {  Public:
 int evalRPN(vector<string>& tokens) {
stack<int> st;  for (int i = 0; i < tokens.size(); i++)
{  if (tokens[i] == "+") {  int val1 = st.top();
st.pop();  int val2 = st.top();  st.pop();  int res = val2
+ val1;  st.push(res);  } else if (tokens[i] == "-") {
int val1 = st.top();  st.pop();  int val2 = st.top();
st.pop();  int res = val2 - val1;  st.push(res);  else if
(tokens[i] == "*") {
 int val1 = st.top();  st.pop();  int
val2 = st.top();  st.pop();  int res
= val2 * val1; st.push(res);
 } else if (tokens[i] == "/") {  int val1 =
st.top();  st.pop();  int val2 = st.top();
st.pop();  int res = val2 / val1;
st.push(res);  } else {
st.push(stoi(tokens[i]));
 }
 }
 return st.top();
 }
 };
```

7.3 Valid Parentheses PROBLEM LOGIC:

```
 class Solution {  public:
 bool isValid(string s) {  stack<char> st;  for (int
i = 0; i < s.size(); i++) {  if (s[i] == '(' || s[i] ==
'{' || s[i] == '[') {  st.push(s[i]);  } else if (s[i] ==
')') {  if (st.empty()) {  return false;
 }
 if (st.top() == '(') {  st.pop();
} else {  return false;
 }
 } else if (s[i] == ']') {  if
(st.empty()) {  return false;
}  if (st.top() == '[') {
st.pop();  } else {  return false;
 }
 } else if (s[i] == '}') {  if
(st.empty()) {  return false;
 }
 if (st.top() == '{') {  st.pop();
} else {  return false;
 }
 }
 }
 if (st.empty()) {  return
true;  } else {  return false;
 }
 }
 };
```

Practical-8

8.1 Implement queue using array (static queue).

CODE:

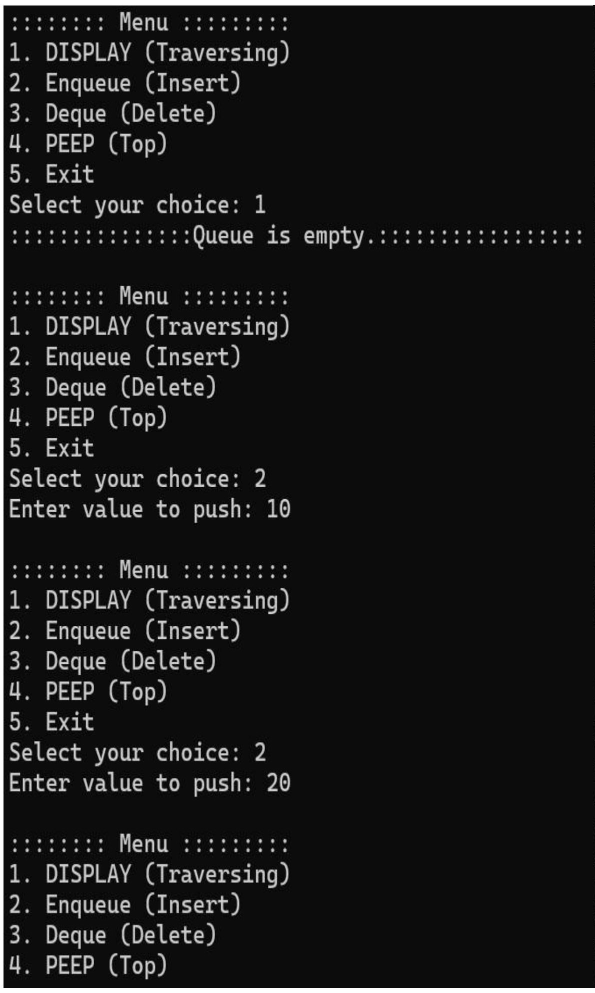
```
#include<iostream> using

namespace std;

#define MAX 5

int

queue1[MAX];

int front1 = -1; int

rear1 = -1;

void display() {

if (front1 == -1)

   {          cout << "::::::::::::::::Queue is

empty.::::::::::::::::" << endl;

   }else{          for (int i = front1;

i <= rear1; i++)

       {

           cout << queue1[i] << endl;

       }

   }

}

void Enqueue(int data)

{     if (rear1 ==

MAX - 1)
```

```
    {          cout << "::::::::::::::Queue is
overflow.::::::::::::::" << endl;     }else{          if(front1
== -1 && rear1 == -1){             front1 = 0;
rear1 = 0;
queue1[rear1] = data;
        }else{             rear1 =
rear1 + 1;
queue1[rear1] = data;
        }
    }
}
void Dequeue()
{     if (front1
== -1)
    {          cout << ":::::::::::::Queue is
underflow.::::::::::::::::" << endl;
    }else{        if(front1 ==
rear1){        front1 = -1;
rear1 = -1;        }else{
front1 = front1 + 1;
        }
    }
}
void peep()
{     if (rear1 == -
```

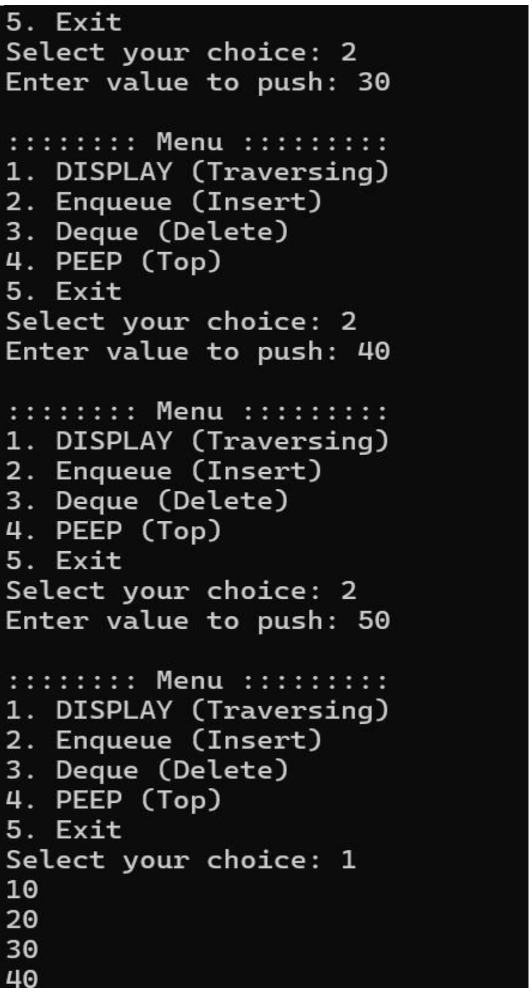
1)

```
{

        cout << "Stack is empty." << endl;
    }

    cout << "\nTop element: " << queue1[rear1] << endl;

}

int main()

{    int opt;    do    {        cout << "\n::::::::

Menu :::::::::" << endl;

cout   <<   "1.   DISPLAY   (Traversing)"

<<endl;

cout  <<  "2.  Enqueue  (Insert)"  <<  endl;

cout  <<  "3.  Deque  (Delete)"  <<  endl;

cout << "4. PEEP (Top)" << endl;

cout << "5. Exit" << endl;

cout << "Select your choice: ";

cin >> opt;

        switch

(opt)

{           case 1:

display();

break;

case 2:

int data;

cout << "Enter value

to push: ";
cin >> data;
Enqueue(data);
break;
case 3:
Dequeue();
break;
case 4:
peep();
break;
case 5:
cout << "Exiting program." << endl;
break;
default:
        cout << "Invalid choice. Try again." << endl;
    }
  } while (opt != 5);
  return 0;
}
```

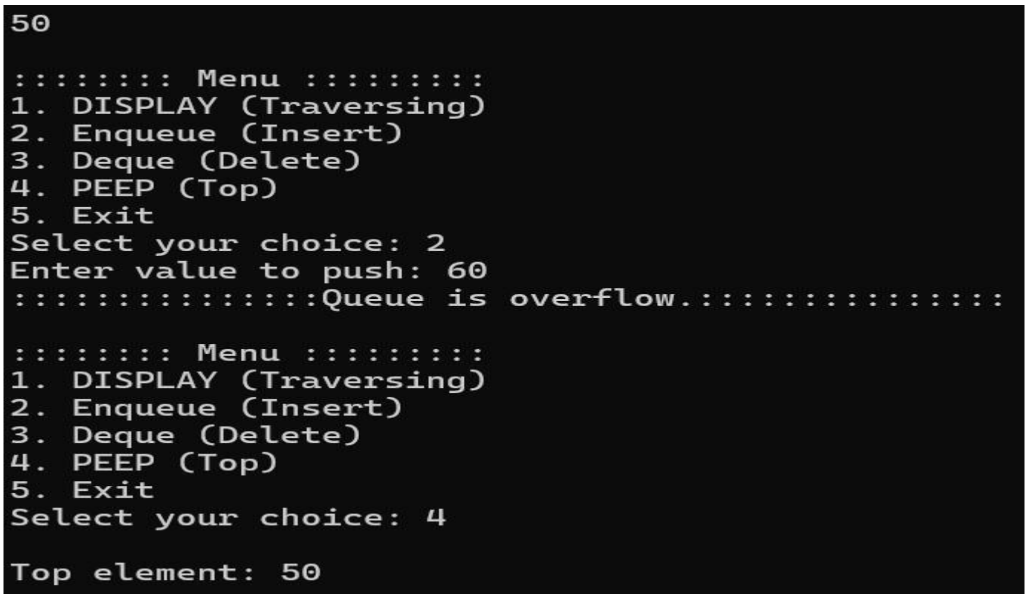
OUTPUT:



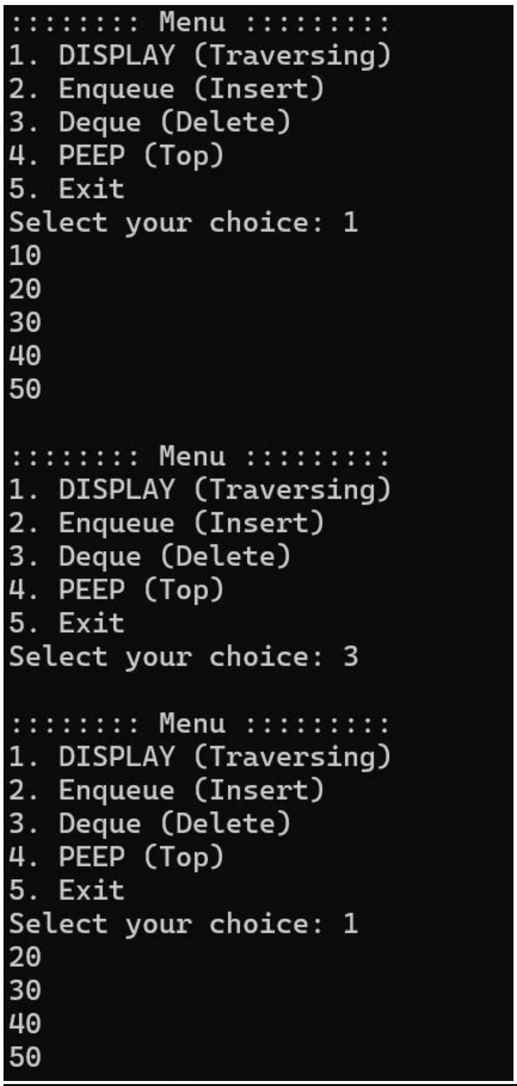
```
:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 1
::::::::::::::Queue is empty.:::::::::::::::

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 2
Enter value to push: 10

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 2
Enter value to push: 20

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
```



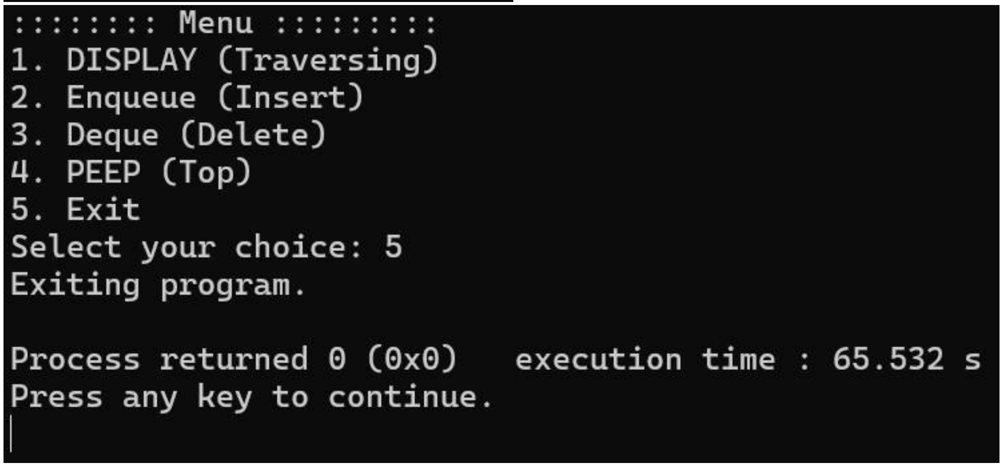
```
5. Exit
Select your choice: 2
Enter value to push: 30

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 2
Enter value to push: 40

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 2
Enter value to push: 50

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 1
10
20
30
40
```



```
50

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 2
Enter value to push: 60
:::::::::::::::Queue is overflow.:::::::::::::::

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 4

Top element: 50
```



```
:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 1
10
20
30
40
50

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 3

:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 1
20
30
40
50
```



```
:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 5
Exiting program.

Process returned 0 (0x0)   execution time : 65.532 s
Press any key to continue.
```

****

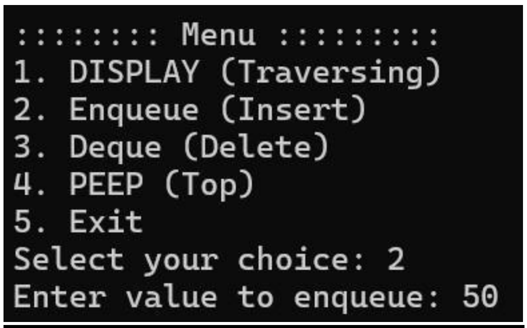
8.2 Implement queue using linked list (dynamic queue).

CODE:

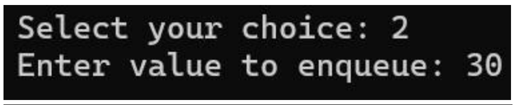
```
#include <iostream>
using namespace std;

struct Node {
int data;
   Node* next;
};

class Queue {
private:
Node* front;
   Node* rear;  /

public:

   Queue() : front(nullptr), rear(nullptr)
{}

   void display() {
if (isEmpty()) {
         cout << "::::::::::::::Queue is empty.::::::::::::::::" << endl;
return;
      }
      Node* temp = front;
while (temp != nullptr) {
cout << temp->data << endl;
         temp = temp->next;
      }
   }

   void Enqueue(int data) {
Node*    newNode    =    new
Node();      newNode->data
= data;         newNode->next
= nullptr;         if
(isEmpty()) {             front = rear =
newNode;
        } else {              rear-
>next = newNode;
rear = newNode;
        }
    }

    void Dequeue() {
if (isEmpty()) {
            cout << "::::::::::::::Queue is underflow.:::::::::::::::::" << endl;
return;
        }
        Node* temp = front;
front = front->next;
delete temp;         if (front
== nullptr) {             rear
= nullptr;
        }
    }

    void peep() {
if (isEmpty()) {
            cout << "Queue is empty." <<
endl;              return;
        }
        cout << "\nFront element: " << front->data << endl;
    }

    bool isEmpty() {
return front == nullptr;
    }
};
```

```
int main() {     Queue queue;     int opt;     do
{         cout << "\n:::::::: Menu :::::::::" <<
endl;         cout << "1. DISPLAY
(Traversing)" << endl;         cout << "2.
Enqueue (Insert)" << endl;         cout << "3.
Deque (Delete)" << endl;         cout << "4.
PEEP (Top)" << endl;         cout << "5. Exit"
<< endl;         cout << "Select your choice: ";
        cin >> opt;
          switch
(opt) {
case 1:
queue.display();
break;             case 2:
int data;
                cout << "Enter value to
enqueue: ";                 cin >> data;
queue.Enqueue(data);
                break;
case 3:
                queue.Dequeue();
break;             case 4:
queue.peep();
break;
            case 5:
                cout << "Exiting program."
<< endl;                 break;
default:
                cout << "Invalid choice. Try again." << endl;
        }
    } while (opt != 5);

    return 0;
}
```

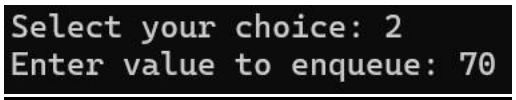
OUTPUT:



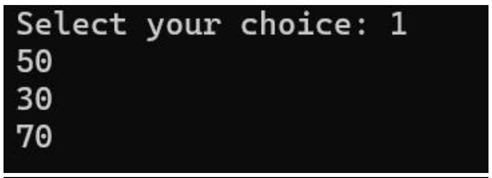
```
:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. Enqueue (Insert)
3. Deque (Delete)
4. PEEP (Top)
5. Exit
Select your choice: 2
Enter value to enqueue: 50
```



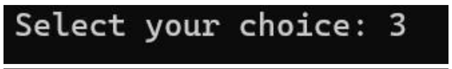
```
Select your choice: 2
Enter value to enqueue: 30
```



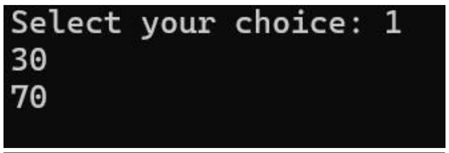
```
Select your choice: 2
Enter value to enqueue: 70
```



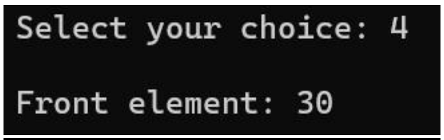
```
Select your choice: 1
50
30
70
```



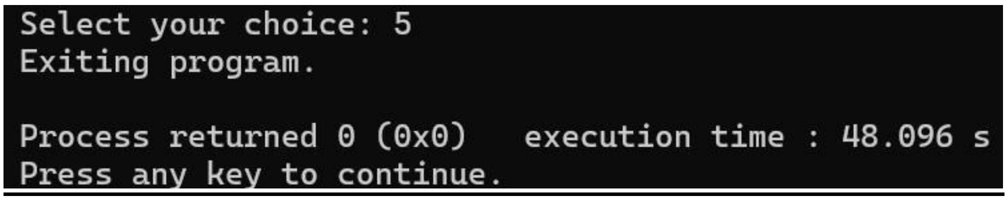
```
Select your choice: 3
```



```
Select your choice: 1
30
70
```



```
Select your choice: 4

Front element: 30
```



```
Select your choice: 5
Exiting program.

Process returned 0 (0x0)   execution time : 48.096 s
Press any key to continue.
```

****

Practical Assignment

8.3 Implement Stack using Queues

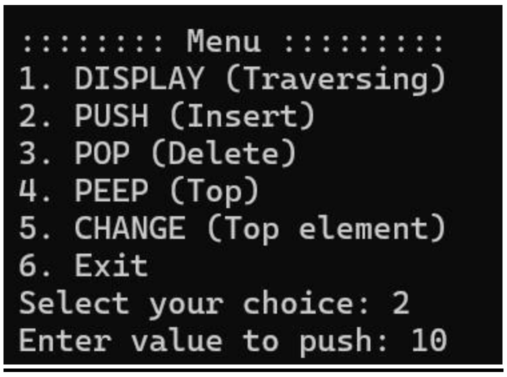
CODE:

```
#include <iostream>

#include <queue> using

namespace std;


class StackUsingQueues { private:

queue<int> q1, q2;


public:


   void push(int data) {

q1.push(data);

   }


   void pop() {

if (q1.empty()) {            cout <<

"Stack is underflow." << endl;

return;
```

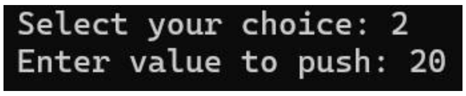
```
        }


        while (q1.size() != 1) {
q2.push(q1.front());            q1.pop();

        }

        q1.pop();


        swap(q1, q2);
    }


    void display() {         if (q1.empty())
{            cout << "Stack is empty." <<
endl;            return;
        }
        queue<int> temp = q1;
while (!temp.empty()) {
cout << temp.front() << endl;
temp.pop();
        }
    }


    void peep() {         if (q1.empty()) {
cout << "Stack is empty." << endl;
return;
        }
        queue<int> temp = q1;
while (temp.size() > 1) {
temp.pop();
        }
        cout << "Top element: " << temp.front() << endl;
    }

    void change(int data) {         if
(q1.empty()) {              cout << "Stack
is empty." << endl;             return;
        }

        while (q1.size() != 1) {
q2.push(q1.front());            q1.pop();
        }

q1.pop();
q1.push(dat
a);

        while (!q2.empty()) {
q1.push(q2.front());            q2.pop();
        }
    }
    bool isEmpty() {
return q1.empty();
    }
};

int main() {
StackUsingQueues stack;
    int opt;

    do {          cout << "\n::::::: Menu
:::::::::" << endl;         cout << "1.
DISPLAY (Traversing)" << endl;
cout << "2. PUSH (Insert)" << endl;
cout <<
"3. POP (Delete)" << endl;         cout << "4. PEEP (Top)" << endl;         cout <<
"5. CHANGE (Top element)" << endl;         cout << "6. Exit" << endl;         cout
<< "Select your choice: ";         cin >> opt;
          switch
(opt) {
case 1:
stack.display();                 break;
case 2: {                 int data;
cout << "Enter value to push: ";
cin >> data;
stack.push(data);                 break;
}             case 3:
```

```
stack.pop();                break;
case
4:                stack.peep();
break;
            case 5: {
int data;                 cout << "Enter new value to change
the top element: ";                cin >> data;
stack.change(data);                break;
}            case 6:
                cout << "Exiting program."
<< endl;                break;
default:
                cout << "Invalid choice. Try again." << endl;
        }
    } while (opt != 6);

    return 0;
}
```

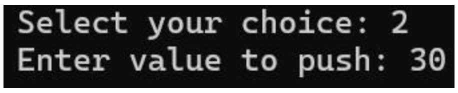
OUTPUT:



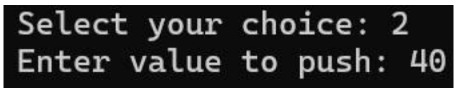
```
:::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. PUSH (Insert)
3. POP (Delete)
4. PEEP (Top)
5. CHANGE (Top element)
6. Exit
Select your choice: 2
Enter value to push: 10
```



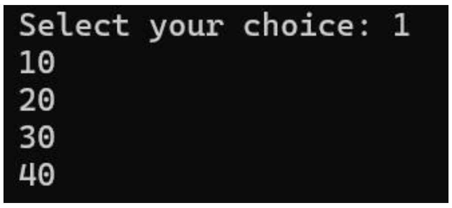
```
Select your choice: 2
Enter value to push: 20
```



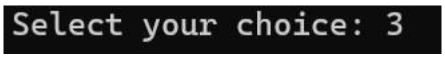
```
Select your choice: 2
Enter value to push: 30
```



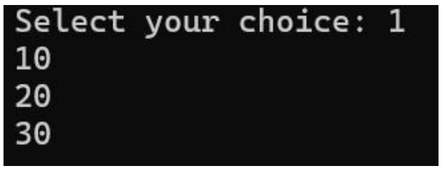
```
Select your choice: 2
Enter value to push: 40
```



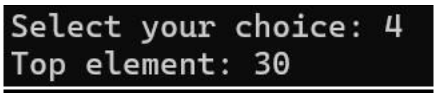
```
Select your choice: 1
10
20
30
40
```



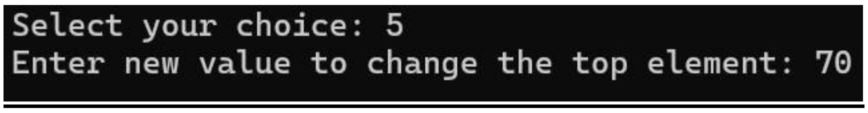
```
Select your choice: 3
```



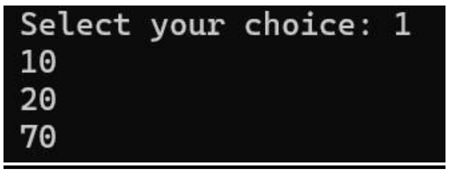
```
Select your choice: 1
10
20
30
```



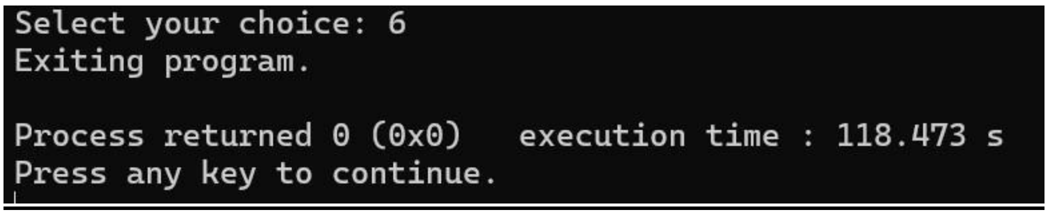
```
Select your choice: 4
Top element: 30
```



```
Select your choice: 5
Enter new value to change the top element: 70
```



```
Select your choice: 1
10
20
70
```



```
Select your choice: 6
Exiting program.

Process returned 0 (0x0)   execution time : 118.473 s
Press any key to continue.
```

****

**************** **8.4 Implement Queue using Stacks**

CODE:

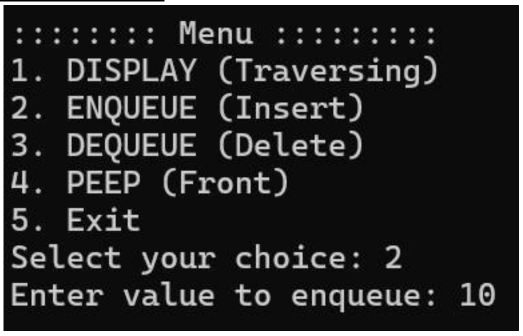
```
#include <iostream>

#include <stack> using

namespace std;


class QueueUsingStacks { private:

stack<int> s1, s2;
```

```
public:

   void enqueue(int data) {
s1.push(data);
   }

   void dequeue() {        if (s1.empty()
&& s2.empty()) {            cout <<
"Queue is underflow." << endl;
            return;
        }
         if
(s2.empty()) {
while
(!s1.empty()) {
s2.push(s1.top());
s1.pop();
            }
        }
         if
(!s2.empty()) {
s2.pop();
        }
   }
   void peep() {        if (s1.empty()
&& s2.empty()) {            cout <<
"Queue is empty." << endl;
           return;
       }
         if
(s2.empty()) {
while
(!s1.empty()) {
s2.push(s1.top());
s1.pop();
           }
       }

       cout << "Front element: " << s2.top() << endl;
   }

   void display() {        if (s1.empty()
&& s2.empty()) {            cout <<
"Queue is empty." << endl;
           return;
       }

       stack<int> temp1 = s1, temp2;
while (!temp1.empty()) {
temp2.push(temp1.top());
temp1.pop();
```

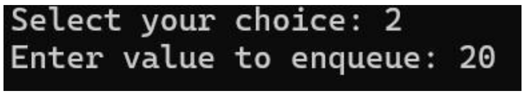
```
        }

        while (!temp2.empty()) {
cout << temp2.top() << endl;
temp2.pop();
        }

        stack<int> temp3 = s2;
while (!temp3.empty()) {
cout << temp3.top() << endl;
temp3.pop();

        }
    }

    bool isEmpty() {         return
s1.empty() && s2.empty();
    }
};

int main() {
QueueUsingStacks queue;
    int opt;

    do {          cout << "\n:::::::: Menu
::::::::" << endl;         cout << "1.
DISPLAY (Traversing)" << endl;
```

```
cout << "2. ENQUEUE (Insert)" << endl;
cout << "3. DEQUEUE (Delete)" <<
endl;        cout << "4. PEEP (Front)" <<
endl;        cout << "5. Exit" << endl;
cout << "Select your choice: ";        cin
>> opt;
          switch
(opt) {
case 1:
queue.display();                break;
case 2: {                int data;
cout << "Enter value to enqueue: ";
               cin >> data;
queue.enqueue(data);
break;            }            case 3:
queue.dequeue();
break;            case 4:
queue.peep();
break;            case 5:
cout << "Exiting program."
<< endl;                break;
default:
               cout << "Invalid choice. Try again." << endl;
        }
    } while (opt != 5);
    return 0;

}
```

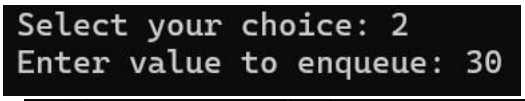
OUTPUT:



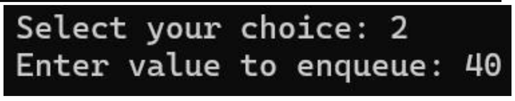
```
::::::: Menu :::::::::
1. DISPLAY (Traversing)
2. ENQUEUE (Insert)
3. DEQUEUE (Delete)
4. PEEP (Front)
5. Exit
Select your choice: 2
Enter value to enqueue: 10
```



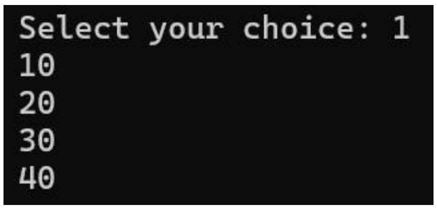
```
Select your choice: 2
Enter value to enqueue: 20
```



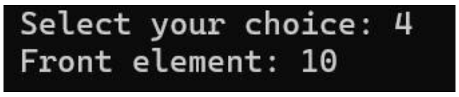
```
Select your choice: 2
Enter value to enqueue: 30
```



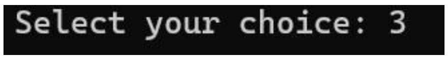
```
Select your choice: 2
Enter value to enqueue: 40
```



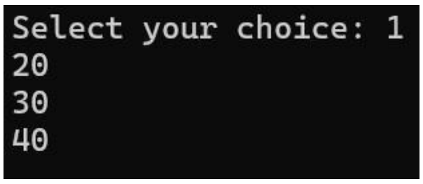
```
Select your choice: 1
10
20
30
40
```



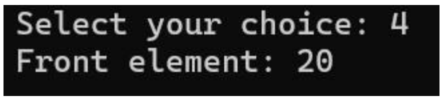
```
Select your choice: 4
Front element: 10
```



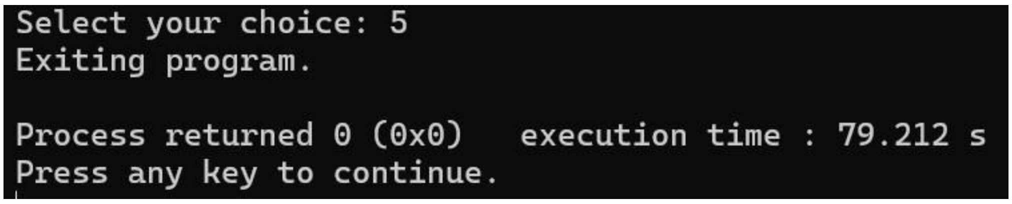
```
Select your choice: 3
```



```
Select your choice: 1
20
30
40
```



```
Select your choice: 4
Front element: 20
```



```
Select your choice: 5
Exiting program.

Process returned 0 (0x0)   execution time : 79.212 s
Press any key to continue.
```

X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—X—
X—X—X—X

Practical-9

9.1 Implement Binary Tree with following operations.

(a) Binary Tree Inorder Traversal

PROBLEM LOGIC:

****

(b) Binary Tree Preorder Traversal

PROBLEM LOGIC:

```
/**
*       Definition for a binary tree node.
*       struct TreeNode {
*       int val;
*       TreeNode *left;
*       TreeNode *right;
*       TreeNode() : val(0), left(nullptr), right(nullptr) {}
*       TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*       TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
*       right(right) {}
*       };  */ class Solution { public:
    vector<int> preorderTraversal(TreeNode* root) {
vector<int> res;          stack<TreeNode*> st;
if (root == nullptr) {           return res;         }
st.push(root);                while  (!st.empty())  {
TreeNode* node = st.top();             st.pop();
res.push_back(node>val);                if (node-
>right != nullptr) {                st.push(node-
>right);
            }            if (node->left
!= nullptr) {                st.push(node->left);
            }        }
return res;
    }
};
```

*** (c) Binary Tree Postorder Traversal**

PROBLEM LOGIC:

```
/**
*       Definition for a binary tree node.
*       struct TreeNode {
*       int val;
*       TreeNode *left;
*       TreeNode *right;
*       TreeNode() : val(0), left(nullptr), right(nullptr) {}
*       TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*       TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
*       right(right) {}
*       };  */ class Solution { public:
    vector<int> postorderTraversal(TreeNode*
root) {         stack<TreeNode*> st1;
stack<TreeNode*> st2;         vector<int> res;
if (root == nullptr) {             return res;
        }         st1.push(root);         while
(!st1.empty()) {             TreeNode* node =
st1.top();             st1.pop();
st2.push(node);             if (node->left !=
nullptr) {                 st1.push(node->left);
            }             if (node->right
!= nullptr) {                 st1.push(node->right);
            }
}         while (!st2.empty()) {
TreeNode* node = st2.top();
st2.pop();
res.push_back(node->val);
}         return res;
    }
};
```

*** (d) Binary Tree Levelorder Traversal**

PROBLEM LOGIC:

```
/**
```

```
*       Definition for a binary tree node.
*       struct TreeNode {
*       int val;
*       TreeNode *left;
*       TreeNode *right;
*       TreeNode() : val(0), left(nullptr), right(nullptr) {}
*       TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*       TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
*       right(right) {}
*       };  */ class Solution { public:
    vector<vector<int>> levelOrder(TreeNode*
root) {         vector<vector<int>> res;
queue<TreeNode*> q;         if (root == nullptr) {
return res;          }
        q.push(root);         while
(!q.empty()) {             int size =
q.size();             vector<int> temp;

            for (int i = 0; i < size; i++) {
TreeNode* temp1 = q.front();
                q.pop();
temp.push_back(temp1->val);                 if
(temp1->left != nullptr) {
                    q.push(temp1->left);
}                   if (temp1->right != nullptr)
{
                    q.push(temp1->right);
}             }             res.push_back(temp);
}         return res;
    }
};
```

Practical-10

10.1 Implement Binary Search Tree (BST)

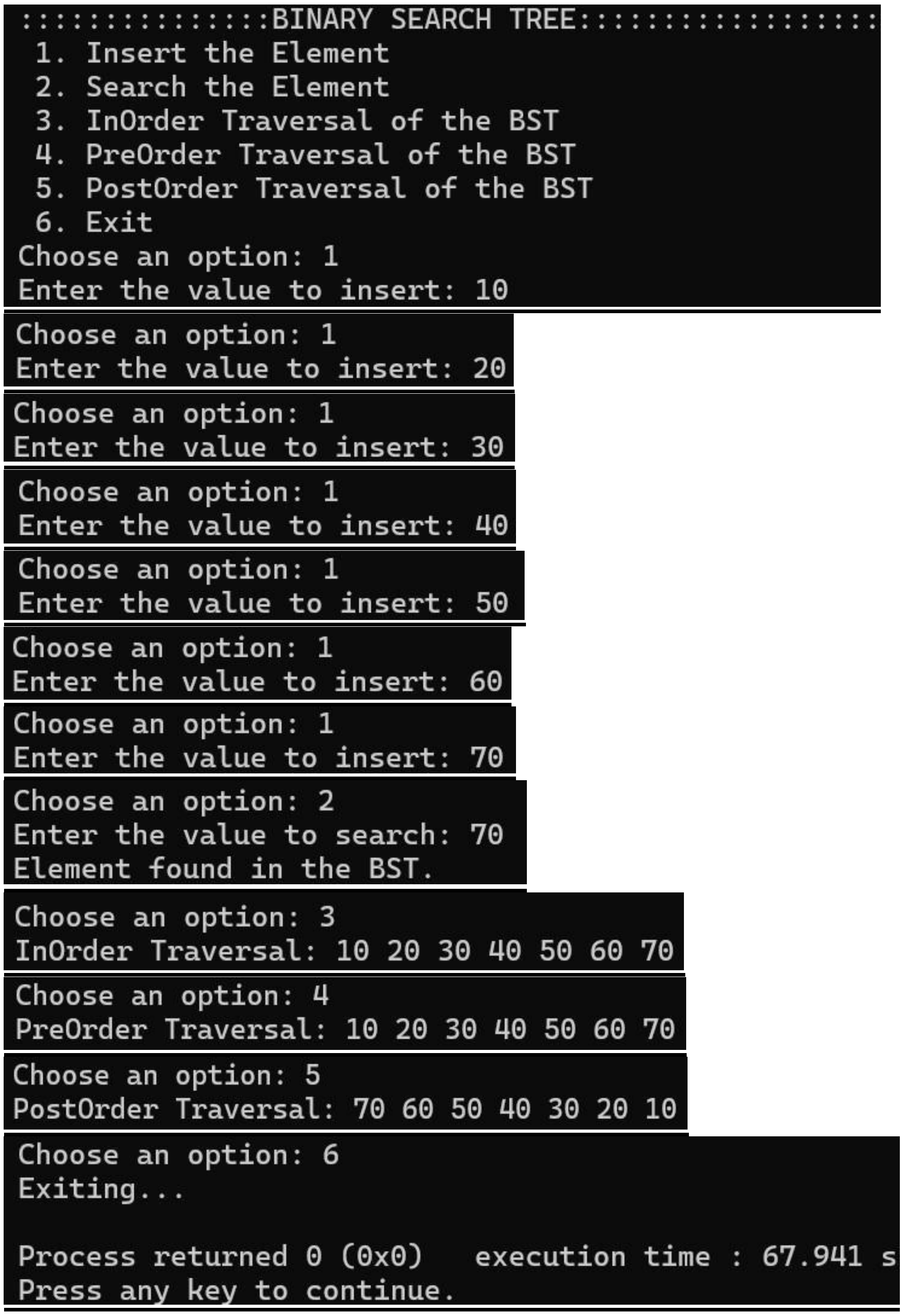
Insertion <u>CODE:</u>

```
#include <bits/stdc++.h> #include
<iostream>
using namespace std;

struct Node {
int data;
Node* left;
    Node* right;

    Node(int value) {
data = value;
left = nullptr;
right = nullptr;
    }
};

Node* insert(Node* root, int
value) {      if (root == nullptr) {
return new Node(value);
    }

    if (value < root->data) {
        root->left = insert(root->left,
value);     } else if (value > root-
>data) {          root->right =
insert(root->right, value);
    }

    return root;
}

bool search(Node* root, int value) {
```

```
if (root == nullptr) {         return
false;
    }

    if (value == root->data) {
return true;
    }

    if (value < root->data) {
return search(root->left, value);
    } else {
        return search(root->right, value);
    }
}

void inOrder(Node* node) {
if (node == nullptr) {
        return;
    }
    inOrder(node->left);
cout << node->data << "
";
inOrder(node->right);
}

void preOrder(Node* node) {
if (node == nullptr) {
        return;
    }
    cout << node->data << " ";
preOrder(node>left);     preOrder(node-
>right);
}

void postOrder(Node* node) {
if (node == nullptr) {
        return;
    }
    postOrder(node->left);     postOrder(node-
>right);     cout
<< node->data << " ";
}

int main() {     Node*
root = nullptr;     int
option;
    int value;

    do {         cout << "::::::::::::::BINARY
SEARCH
TREE:::::::::::::::::";         cout << "\n 1. Insert the
Element";         cout << "\n 2. Search the Element";
cout << "\n 3. InOrder Traversal of the BST";         cout
<< "\n 4. PreOrder Traversal of the BST";         cout <<
"\n 5. PostOrder Traversal of the BST";
        cout << "\n 6. Exit";
cout << "\nChoose an option:
";         cin >> option;

        switch (option) {
case 1:
                cout << "Enter the value to insert:
";                cin >> value;
                root = insert(root, value);
break;              case 2:                cout <<
"Enter the value to search: ";                cin
>> value;                if (search(root, value)) {
cout << "Element found in the BST.\n";
                } else {                    cout <<
"Element not found in the BST.\n";
                }
break;
             case 3:
```

```
cout << "InOrder Traversal: ";
inOrder(root);
cout << "\n";
break;
             case 4:
 cout << "PreOrder Traversal: ";
preOrder(root);
cout << "\n";
 break;
             case
5:
                cout << "PostOrder Traversal: ";
postOrder(root);
cout << "\n";
     break;
             case 6:
cout << "Exiting...\n";
                break;

default:
   cout << "Invalid option! Please try again.\n";
        }

    } while (option != 6);

    return 0;
}
```

Output:



```
 ::::::::::::::BINARY SEARCH TREE:::::::::::::::::
  1. Insert the Element
  2. Search the Element
  3. InOrder Traversal of the BST
  4. PreOrder Traversal of the BST
  5. PostOrder Traversal of the BST
  6. Exit
 Choose an option: 1
 Enter the value to insert: 10
Choose an option: 1
Enter the value to insert: 20
Choose an option: 1
Enter the value to insert: 30
Choose an option: 1
Enter the value to insert: 40
Choose an option: 1
Enter the value to insert: 50
Choose an option: 1
Enter the value to insert: 60
Choose an option: 1
Enter the value to insert: 70
Choose an option: 2
Enter the value to search: 70
Element found in the BST.
Choose an option: 3
InOrder Traversal: 10 20 30 40 50 60 70
Choose an option: 4
PreOrder Traversal: 10 20 30 40 50 60 70
Choose an option: 5
PostOrder Traversal: 70 60 50 40 30 20 10
Choose an option: 6
Exiting...

Process returned 0 (0x0)   execution time : 67.941 s
Press any key to continue.
```

****

10.2 Implement searching in Binary Search Tree (BST)

PROBLEM LOGIC:

```
/**
*       Definition for a binary tree node.
*       struct TreeNode {
*       int val;
*       TreeNode *left;
*       TreeNode *right;
*       TreeNode() : val(0), left(nullptr), right(nullptr) {}
*       TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*       TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
*       right(right) {}
*       };  */ class Solution { public:
   TreeNode* searchBST(TreeNode* root, int
val) {          if (root == nullptr || root->val == val)
{            return root;
       }        if (val < root->val) {
return searchBST(root->left, val);
}        return searchBST(root-
>right, val);
   }
};
```

** Practical Assignment 10.3 Kth Smallest Element in a BST**

PROBLEM LOGIC:

```
/**
*       Definition for a binary tree node.
*       struct TreeNode {
*       int val;
*       TreeNode *left;
*       TreeNode *right;
*       TreeNode() : val(0), left(nullptr), right(nullptr) {}
*       TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
*       TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
*       right(right) {}
*       };  */ class Solution { public:
    vector<int> res;     void
inOrder(TreeNode* ptr) {         if
(ptr == nullptr) {              return;
}         inOrder(ptr->left);
res.push_back(ptr->val);
inOrder(ptr->right);
    }     int kthSmallest(TreeNode* root,
int k) {         inOrder(root);         return
res[k - 1];
    } };
```

Practical-11

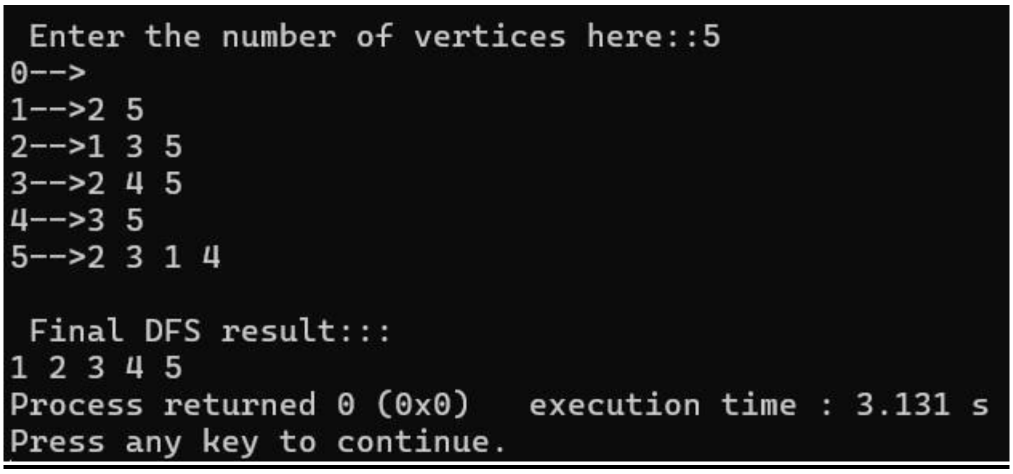
11.1 Implement DFS of Graph.

CODE:

```
//Implement DFS of Graph.
#include<iostream>
#include<vector>
#include<stack>

using namespace std;

void dfsUtil(int node, vector<int> adj_list[], vector<bool> &visited,
vector<int> &dfs_result) {

    visited[node] = true;

    dfs_result.push_back(node);

    for (auto neighbor : adj_list[node]) {

        if (!visited[neighbor]) {

            dfsUtil(neighbor, adj_list, visited, dfs_result);
        }
    }
}

vector<int> dfs(vector<int> adj_list[], int no_of_vertices) {
vector<bool> visited(no_of_vertices + 1, false);     vector<int>
dfs_result;

    dfsUtil(1, adj_list, visited, dfs_result);

    return dfs_result;
}
```

```
void addEdge(vector<int> adj_list[], int u, int v) {
adj_list[u].push_back(v);
adj_list[v].push_back(u);  }

int main() {
   int no_of_vertices;

   cout << "\n Enter the number of vertices here::";
cin >> no_of_vertices;

   vector<int> adj_list[no_of_vertices + 1];

   addEdge(adj_list, 1, 2);
addEdge(adj_list, 2, 3);
addEdge(adj_list, 2, 5);
addEdge(adj_list, 3, 4);
addEdge(adj_list, 3, 5);
addEdge(adj_list,
1, 5);     addEdge(adj_list, 4, 5);

   for (int i = 0; i <= no_of_vertices; i++) {

      cout << i << "-->";

      for (auto j = adj_list[i].begin(); j != adj_list[i].end(); j++) {

         cout << *j << " ";
      }

      cout << endl;
   }

   vector<int> result = dfs(adj_list, no_of_vertices);

   cout << "\n Final DFS result:::\n";

   for (auto i : result) {
        cout << i << " ";
    }

    return 0;
}
```

~~**Output:**~~



```
 Enter the number of vertices here::5
0-->
1-->2 5
2-->1 3 5
3-->2 4 5
4-->3 5
5-->2 3 1 4

 Final DFS result:::
1 2 3 4 5
Process returned 0 (0x0)   execution time : 3.131 s
Press any key to continue.
```

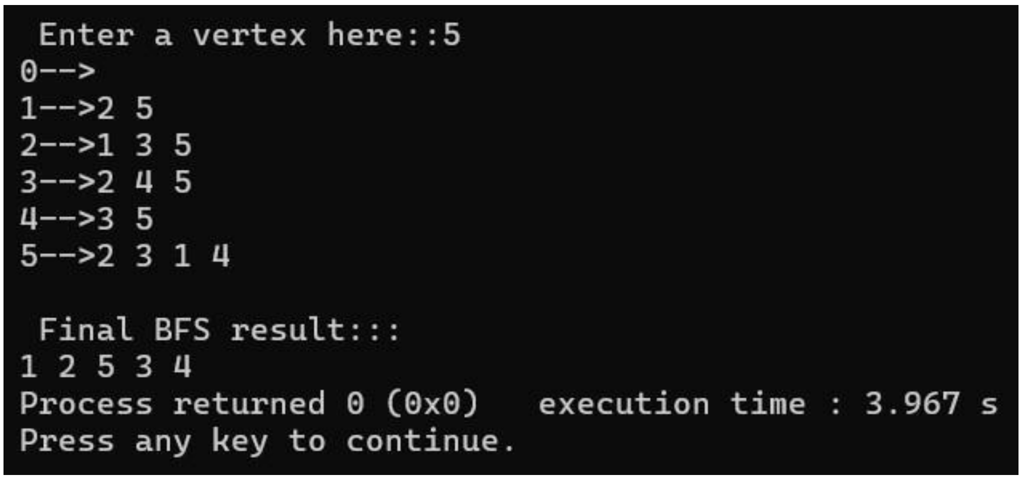
11.2 Implement BFS of Graph.

CODE:

```
//Implement BFS of Graph.
#include<iostream>
#include<vector>
#include<queue>

using namespace std;


vector<int> bfs(vector<int> adj_list[],int no_of_vertices){

        //create a one array

        int visited[no_of_vertices + 1] = {0};
        queue<int> q;

        vector<int> bfs_result;


        visited[1] = 1;

        q.push(1);

        while(!q.empty()){


           int node = q.front();//1 2 5 3

           q.pop();
           bfs_result.push_back(node);//1  2


           for(auto i : adj_list[node]){ //2 5  1 5 3


if(!visited[i]){

visited[i] = 1;

                      q.push(i);
                 }
           }
        }

    return bfs_result;
}

void addEdge( vector<int> adj_list[],int u,int v){
        adj_list[u].push_back(v);

        adj_list[v].push_back(u);


}

int main(){

    int no_of_vertices;

    cout<<"\n Enter a vertex here::";

    cin>>no_of_vertices;

    //array of vector

    vector<int> adj_list[no_of_vertices + 1];


    addEdge(adj_list,1,2);

    addEdge(adj_list,2,3);

    addEdge(adj_list,2,5);

    addEdge(adj_list,3,4);

    addEdge(adj_list,3,5);

    addEdge(adj_list,1,5);

    addEdge(adj_list,4,5);

    for(int i = 0;i<=no_of_vertices;i++){

            cout<<i<<"-->";
```

```
        for(auto j = adj_list[i].begin(); j != adj_list[i].end();j++){

            cout<<*j<<" ";

}

         cout<<endl;
   }

   vector<int> result = bfs(adj_list,no_of_vertices);

   cout<<"\n Final BFS result:::\n";

   for(auto i : result){


      cout<<i<<" ";
   }

   return 0;
}
```

Output:



```
 Enter a vertex here::5
0-->
1-->2 5
2-->1 3 5
3-->2 4 5
4-->3 5
5-->2 3 1 4

 Final BFS result:::
1 2 5 3 4
Process returned 0 (0x0)   execution time : 3.967 s
Press any key to continue.
```

Practical Assignment

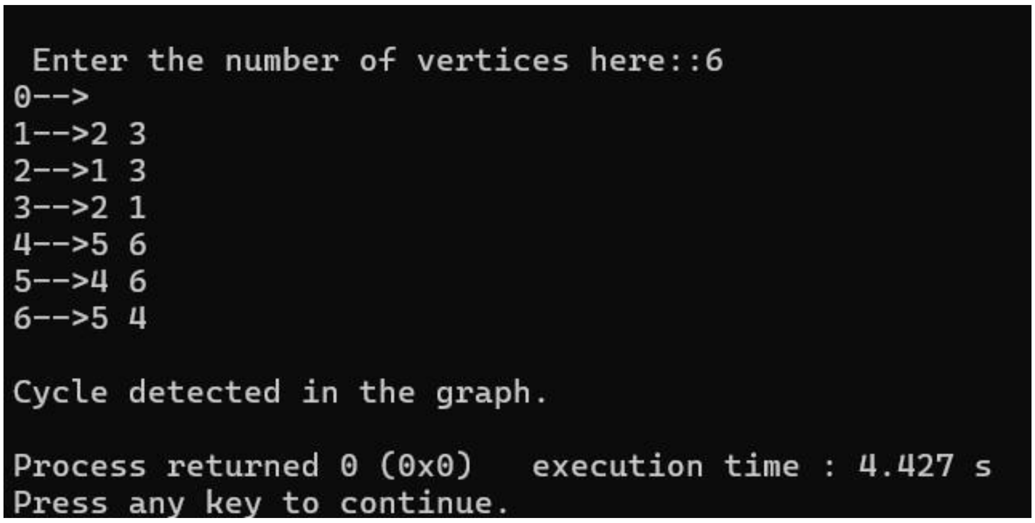
11.3 Detect cycle in an undirected graph

CODE:

```
//Detect cycle in an undirected graph
#include <iostream>
#include <vector>


using namespace std;

bool dfs(int node, int parent, vector<int> adj_list[], vector<bool> &visited) {


    visited[node] = true;


    for (auto neighbor : adj_list[node]) {


        if (!visited[neighbor]) {


            if (dfs(neighbor, node, adj_list, visited)) {


return true;
            }
        }


        else if (neighbor != parent) {
```

```
return true;
        }     }
return false;
}

bool detectCycle(vector<int> adj_list[], int no_of_vertices) {

    vector<bool> visited(no_of_vertices + 1, false);

    for (int i = 1; i <= no_of_vertices; i++) {
                    if
(!visited[i]) {
if (dfs(i, -1,
adj_list, visited)) {


return true;
            }
        }     }
return false;
}

void addEdge(vector<int> adj_list[], int u, int v) {
    adj_list[u].push_back(v);
adj_list[v].push_back(u);
}

int main() {     int
no_of_vertices;

    cout << "\n Enter the number of vertices here::";
cin >> no_of_vertices;



    vector<int> adj_list[no_of_vertices + 1];

    addEdge(adj_list, 1, 2);
addEdge(adj_list, 2, 3);      addEdge(adj_list,
3, 1); // This edge creates a cycle
addEdge(adj_list, 4, 5);     addEdge(adj_list, 5, 6);
    addEdge(adj_list, 6, 4); // This edge creates a cycle in another component
for (int i = 0; i <= no_of_vertices; i++) {

        cout << i << "-->";

        for (auto j = adj_list[i].begin(); j != adj_list[i].end(); j++) {

            cout << *j << " ";
        }
        cout << endl;
    }

    if (detectCycle(adj_list, no_of_vertices)) {


        cout << "\nCycle detected in the graph.\n";
    } else {

        cout << "\nNo cycle detected in the graph.\n";
    }

    return 0;
}
```

Output:



```
 Enter the number of vertices here::6
0-->
1-->2 3
2-->1 3
3-->2 1
4-->5 6
5-->4 6
6-->5 4

Cycle detected in the graph.

Process returned 0 (0x0)   execution time : 4.427 s
Press any key to continue.
```

Practical-12

12.1 Implementing a Hash Table for Student Records Management

Background:

You are tasked with implementing a hybrid hash table to manage student records efficiently. Each student record consists of a unique student ID (key) and the corresponding student score (data). The hash table will support two methods of collision handling: separate chaining and linear probing. This flexibility ensures efficient handling of collisions, enabling you to choose the most suitable method based on different scenarios.

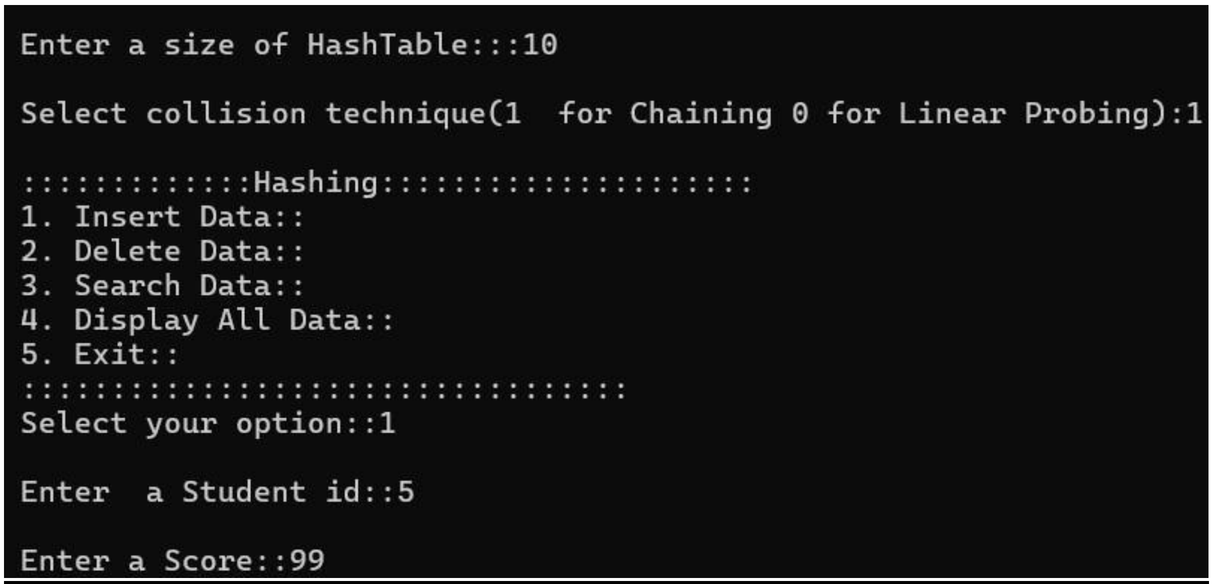
CODE:

```
#include<iostream>
#include<vector>

using namespace std;

struct Student{
         int
s_id;


        float score;

        Student(int st_id,float st_score){
              s_id =
st_id;


            score = st_score;


        }
};

//for Chaining
vector<vector<Student>> Hashtable;
```

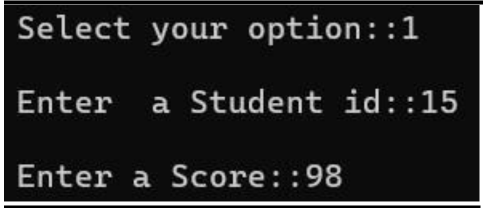
```
//for linear probing
vector<Student> Hashtable1;
vector<bool> isOccupied;


int hashFunction(int key,int hashtablesize){

   return key % hashtablesize;
}

void insertRecord(int s_id,float score,bool isChaining,int hashtablesize){

        int hashIndex = hashFunction(s_id,hashtablesize);

        if(isChaining){

            Hashtable[hashIndex].push_back(Student(s_id,score));

        }else{

            int originalIndex = hashIndex;

            while(isOccupied[hashIndex]){

                hashIndex = (hashIndex +1 ) % hashtablesize;

                if(hashIndex == originalIndex){

                    cout<<"\n HashTable is full::";
return ;
                }
            }

            Hashtable1[hashIndex] = Student(s_id,score);

            isOccupied[hashIndex] = true;
        }
}

void searchRecord(int s_id,bool isChaining,int hashtablesize){

    int hashIndex = hashFunction(s_id,hashtablesize);
bool flag;
    if(isChaining){

        for(auto it: Hashtable[hashIndex]){

            if(it.s_id == s_id){

                cout<<"[ ID: "<<it.s_id<<" Score: "<<it.score<<" ]";
flag = true;                break;
            }

        }
        if(flag == false){

            cout<<"\n Data is not present in this HashTable::";
        }

    }else{

        int originalIndex = hashIndex;

        bool flag = false;

        while(isOccupied[hashIndex]){

            if(Hashtable1[hashIndex].s_id == s_id){
flag = true;                cout<<"[ ID:
"<<Hashtable1[hashIndex].s_id<<" Score:
"<<Hashtable1[hashIndex].score<<" ]";
```

```
return;
            }

            hashIndex = (hashIndex + 1) % hashtablesize;

            if(originalIndex == hashIndex){

                cout<<"\n Record is not present in this hashTable";

break;
            }
        }


        if(flag == false){

             cout<<"\n Record is not present in this hashTable";

        }
    }
}

void removeRecord(int s_id,bool isChaining,int hashtablesize){

    int hashIndex = hashFunction(s_id,hashtablesize);

    if(isChaining){

        for(auto  i= Hashtable[hashIndex].begin(); i !=
Hashtable[hashIndex].end();i++ ){

            if(i->s_id == s_id){

                Hashtable[hashIndex].erase(i);
                cout<<"\n Delete Done";

break;
            }
        }

    }else{


        int temp = 0;

         for(auto  it: Hashtable1 ){

            if(it.s_id == s_id){

                Hashtable1[temp] = Student(-1,-1);

                cout<<"\n Delete Done";

                isOccupied[temp] = false;

break;
            }

            temp++;

        }
    }
}

void displayAllRecords(int hashtablesize,bool isChaining){

        if(isChaining){

        for(int i = 0;i<hashtablesize;i++){
            cout<<i<<"-->";

            for(auto it:Hashtable[i]){

                cout<<"[ ID: "<<it.s_id<<" Score: "<<it.score<<" ] -->";

            }
            cout<<"NULL"<<endl;

        }
        }else{

            cout<<"\n ::: HashTable Using Linear Probing\n";
int i =0;
            for(auto it : Hashtable1){

cout<<endl;
cout<<i<<"-->";
                 cout<<"[ ID: "<<it.s_id<<" Score: "<<it.score<<" ]";
i++;
            }
        }
}

int main(){

    int hashtablesize,choice;

    bool isChaining;

    int s_id;
    float score;

    cout<<"\n Enter a size of HashTable:::";

    cin>>hashtablesize;
    // now no of rows are fixed

    Hashtable.resize(hashtablesize);

    Hashtable1.resize(hashtablesize,{-1,-1});

    isOccupied.resize(hashtablesize,false);

    cout<<"\n Select collision technique(1  for Chaining 0 for Linear
Probing):";

    cin>>choice;

    isChaining = choice;

    int option;

    do{
        cout<<"\n ::::::::::::Hashing:::::::::::::::::::";

        cout<<"\n 1. Insert Data::";

        cout<<"\n 2. Delete Data::";

        cout<<"\n 3. Search Data::";

        cout<<"\n 4. Display All Data::";

        cout<<"\n 5. Exit::";
          cout<<"\n
::::::::::::::::::::::::::::::::";

        cout<<"\n Select your option::";

        cin>>option;
        switch(option){
          case
1:
            cout<<"\n Enter  a Student id::";
cin>>s_id;              cout<<"\n Enter a
Score::";              cin>>score;

            insertRecord(s_id,score,isChaining,hashtablesize);

break;
case 2:
            cout<<"\n Enter  a Student id::";
cin>>s_id;
            removeRecord(s_id,isChaining,hashtablesize);
break;
         case 3:
            cout<<"\n Enter  a Student id::";
cin>>s_id;
            searchRecord(s_id,isChaining,hashtablesize);
break;

case 4:
displayAl
lRecords(
hashtable
size,isCh
aining);
break;
        }

    }while(option != 5);

    return 0;
}
```

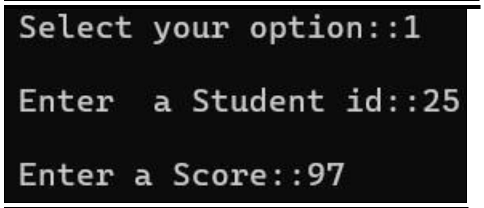
Output: [For Chaining]



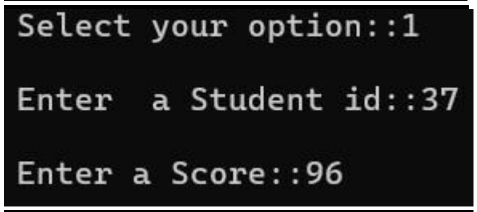
```
Enter a size of HashTable:::10

Select collision technique(1  for Chaining 0 for Linear Probing):1

::::::::::::Hashing::::::::::::::::::::
1. Insert Data::
2. Delete Data::
3. Search Data::
4. Display All Data::
5. Exit::
::::::::::::::::::::::::::::::::::
Select your option::1

Enter  a Student id::5

Enter a Score::99
```



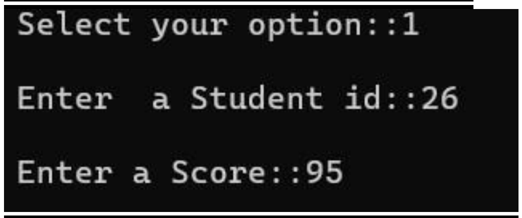
```
Select your option::1

Enter  a Student id::15

Enter a Score::98
```



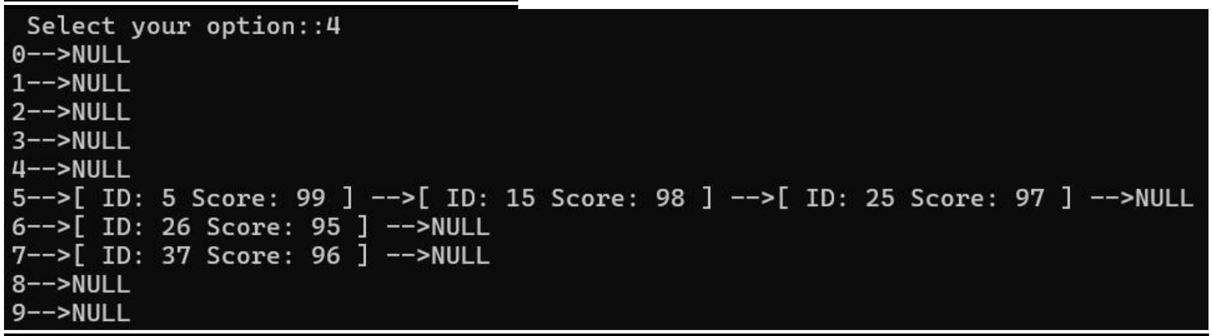
```
Select your option::1

Enter  a Student id::25

Enter a Score::97
```



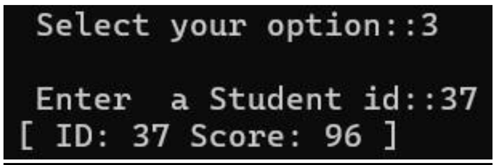
```
Select your option::1

Enter  a Student id::37

Enter a Score::96
```



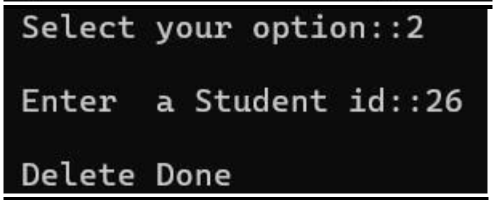
```
Select your option::1

Enter  a Student id::26

Enter a Score::95
```



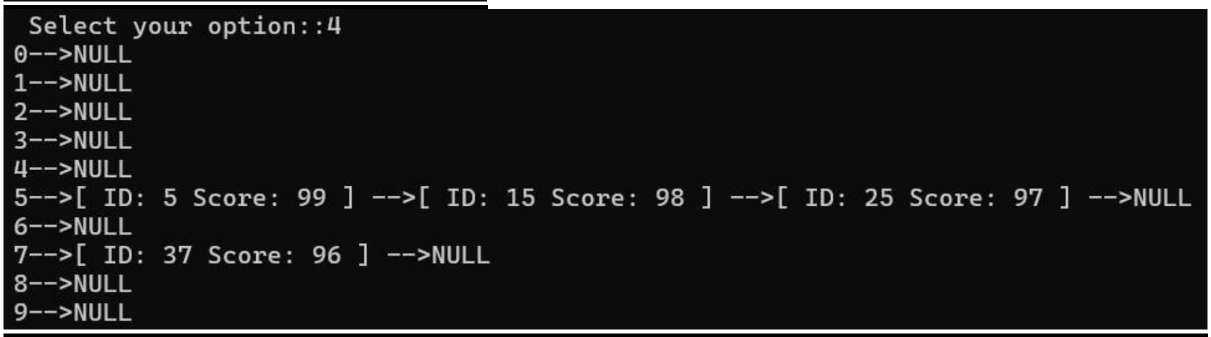
```
 Select your option::4
0-->NULL
1-->NULL
2-->NULL
3-->NULL
4-->NULL
5-->[ ID: 5 Score: 99 ] -->[ ID: 15 Score: 98 ] -->[ ID: 25 Score: 97 ] -->NULL
6-->[ ID: 26 Score: 95 ] -->NULL
7-->[ ID: 37 Score: 96 ] -->NULL
8-->NULL
9-->NULL
```



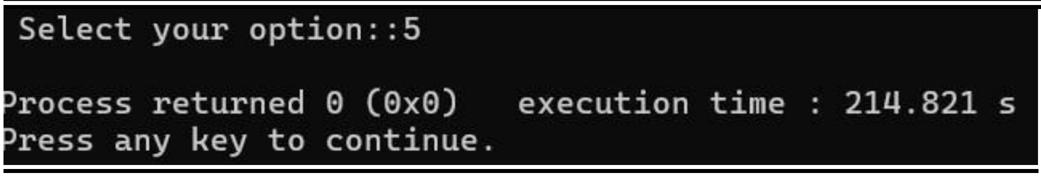
```
 Select your option::3

 Enter  a Student id::37
[ ID: 37 Score: 96 ]
```



```
 Select your option::2

 Enter  a Student id::26

 Delete Done
```

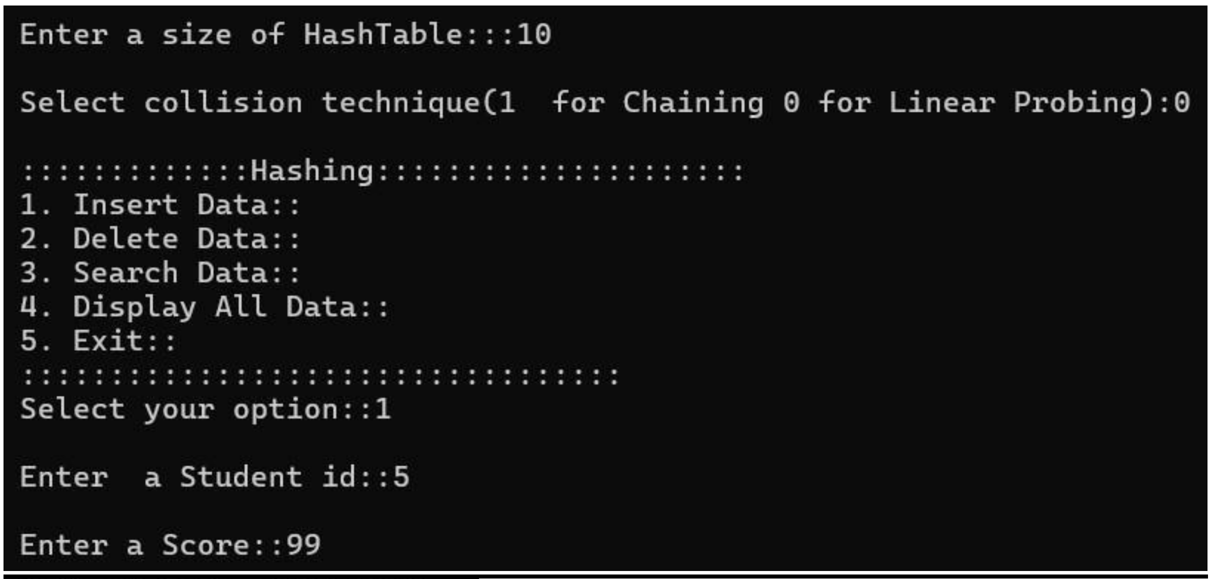


```
 Select your option::4
0-->NULL
1-->NULL
2-->NULL
3-->NULL
4-->NULL
5-->[ ID: 5 Score: 99 ] -->[ ID: 15 Score: 98 ] -->[ ID: 25 Score: 97 ] -->NULL
6-->NULL
7-->[ ID: 37 Score: 96 ] -->NULL
8-->NULL
9-->NULL
```

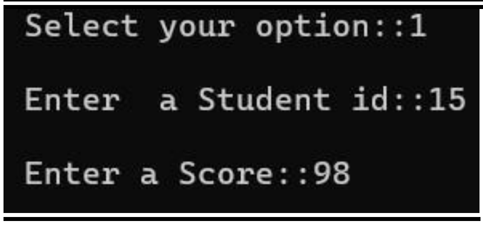


```
 Select your option::5

Process returned 0 (0x0)   execution time : 214.821 s
Press any key to continue.
```

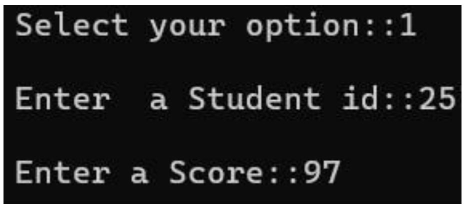
Output: ~~[For Chaining]~~



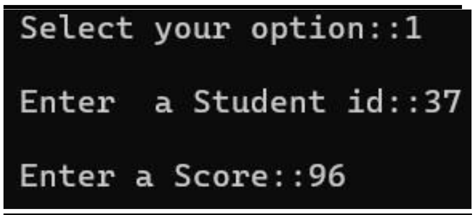
```
 Enter a size of HashTable:::10

 Select collision technique(1  for Chaining 0 for Linear Probing):0

 ::::::::::::Hashing::::::::::::::::::::
 1. Insert Data::
 2. Delete Data::
 3. Search Data::
 4. Display All Data::
 5. Exit::
 ::::::::::::::::::::::::::::::::::
 Select your option::1

 Enter  a Student id::5

 Enter a Score::99
```



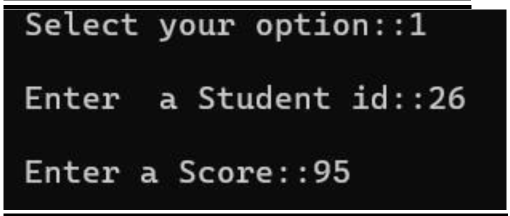
```
 Select your option::1

 Enter  a Student id::15

 Enter a Score::98
```



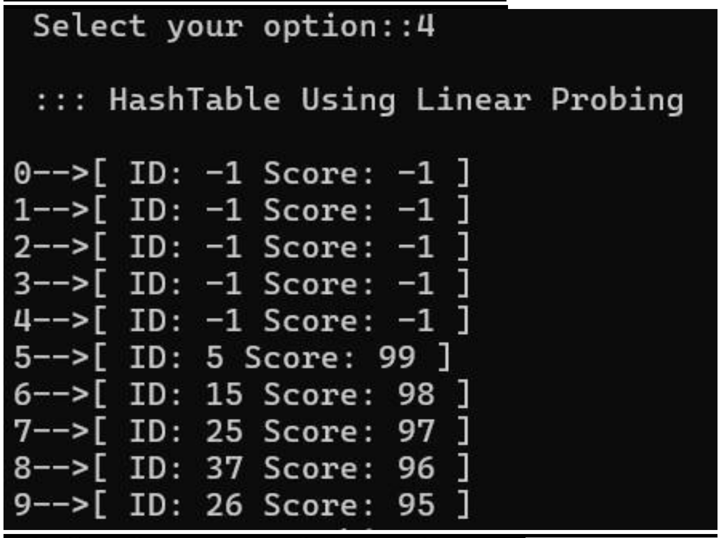
```
Select your option::1

Enter  a Student id::25

Enter a Score::97
```



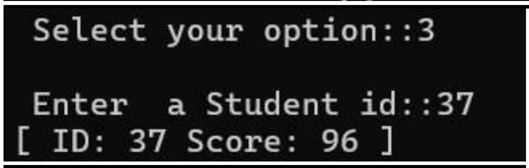
```
Select your option::1

Enter  a Student id::37

Enter a Score::96
```



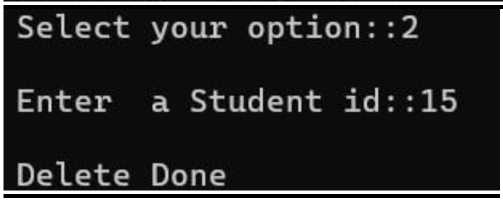
```
Select your option::1

Enter  a Student id::26

Enter a Score::95
```



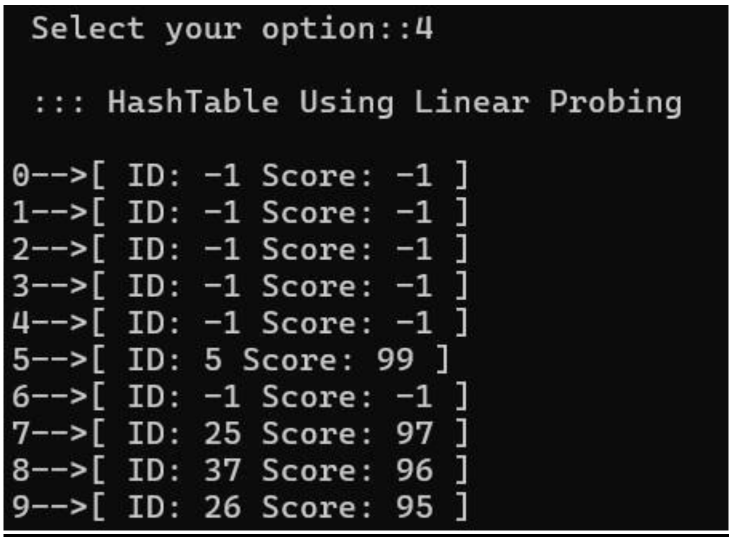
```
Select your option::4

::: HashTable Using Linear Probing

0-->[ ID: -1 Score: -1 ]
1-->[ ID: -1 Score: -1 ]
2-->[ ID: -1 Score: -1 ]
3-->[ ID: -1 Score: -1 ]
4-->[ ID: -1 Score: -1 ]
5-->[ ID: 5 Score: 99 ]
6-->[ ID: 15 Score: 98 ]
7-->[ ID: 25 Score: 97 ]
8-->[ ID: 37 Score: 96 ]
9-->[ ID: 26 Score: 95 ]
```



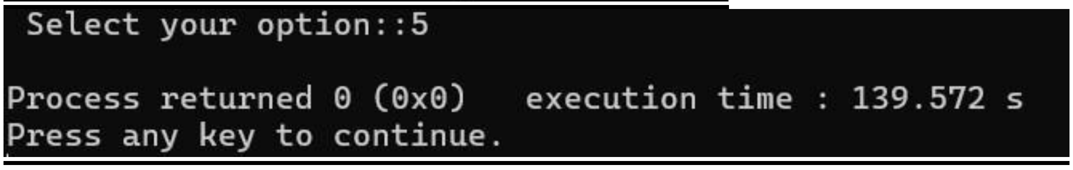
```
Select your option::3

Enter  a Student id::37
[ ID: 37 Score: 96 ]
```



```
Select your option::2

Enter  a Student id::15

Delete Done
```



```
 Select your option::4

 ::: HashTable Using Linear Probing

0-->[ ID: -1 Score: -1 ]
1-->[ ID: -1 Score: -1 ]
2-->[ ID: -1 Score: -1 ]
3-->[ ID: -1 Score: -1 ]
4-->[ ID: -1 Score: -1 ]
5-->[ ID: 5 Score: 99 ]
6-->[ ID: -1 Score: -1 ]
7-->[ ID: 25 Score: 97 ]
8-->[ ID: 37 Score: 96 ]
9-->[ ID: 26 Score: 95 ]
```



```
 Select your option::5

Process returned 0 (0x0)   execution time : 139.572 s
Press any key to continue.
```