Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)

Term work

on

Design & Analysis of Algorithms LAB

(PCS 409)

2021-22

Submitted to:

Dilip Kumar Gangwar

Asst. Professor

GEHU, D.Dun

Submitted by:

Vipul Chauhan

University Roll No.:2018855

Class Roll no./Section:62/A

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, DEHRADUN

ACKNOWLEDGMENT

I would like to particularly thank my DAA Lab Faculty **Mr Dilip Kumar Gangwar** for his patience, support and encouragement throughout the completion of this Term work.

At last, but not the least I greatly indebted to all other persons who directly or indirectly helped me during this course.

Vipul Chauhan

Univ. Roll No.- 2018855

B.Tech CSE-A-IV Sem

Session: 2021-2022

GEHU, Dehradun

Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)

Table of Contents

	program should also find number of comparisons and number of swaps required.	
9	Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements in the array or not. (use sorting) (Time Complexity = O(n log n))	31-34
10	Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array.	35-39
11	Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.	40-43
12	Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))	44-47
13	Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and print it. (Time Complexity = O(n)) (Hint: Use counting sort)	48-51
14	Given an unsorted array of integers, design an algorithm and implement it using a program to find whether two elements exist such that their sum is equal to the given key element. (Time Complexity = O(n log n))	52-56
15	You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find list of elements which are common to both. (Time Complexity = O(m+n))	57-60
16	Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)	
17	Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)	
18	Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.	

19	After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay's house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends' location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend's location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra's algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay's results?	
20	Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.	
21	Given a directed graph with two vertices (source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.	
22	Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)	
23	Implement the previous problem using Kruskal's algorithm.	
24	Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.	
25	Given a graph, Design an algorithm and implement it using a program to implement Floyd- Warshall all pair shortest path algorithm.	

26	Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction xi of item i, where 0 ≤xi≤ 1.	
27	Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach)	
28	Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.	
29	Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.	
30	Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.	
31	Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.	
32	Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.	
33	Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.	
34	Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that	

	appears in the same relative order, but not necessarily contiguous.	
35	Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).	
36	Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.	
37	Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = O(n))	
38	Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicate within a small window of size k < n.	
39	Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a*b = c*d, where a, b, c and d are distinct elements of array.	
40	Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,. is sequence of ugly numbers.	
41	Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.	

Graphic Era
HILL UNIVERSITY
Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)

DEPARTMENT OF CSE
STUDENT LAB REPORT SHEET

Photograph
Passport Size

Name of Student..... Mob. No.....

Address Permanent.....

Father's Name..... Occupation .....

Mob. No ..... Mother's Name .....

Occupation..... Mob. No. .....

Section..... Branch..... Semester..... Class Roll No..... Grade A B C

Local Address..... Email ..... Marks 5 3 1

S.No.	Practical	D.O.P.	Date of Submission	Grade (Viva)	Grade (Report File)	Total Marks (out of 10)	Student's Signature	Teacher's Signature
1	Program-01							
2	Program-02							
3	Program-03							
4	Program-04							
5	Program-05							
6	Program-06							
7	Program-07							
8	Program-08							
9	Program-09							

10	**Program-10**							
11	**Program-11**							
12	**Program-12**							
13	**Program-13**							
14	**Program-14**							
15	**Program-15**							
16	**Program-16**							
17	**Program-17**							
18	**Program-18**							
19	**Program-19**							
20	**Program-20**							
21	**Program-21**							
22	**Program-22**							
23	**Program-23**							
24	**Program-24**							
25	**Program-25**							
26	**Program-26**							
27	**Program-27**							
28	**Program-28**							
29	**Program-29**							
30	**Program-30**							
31	**Program-31**							
32	**Program-32**							

33	**Program-33**							
34	**Program-34**							
35	**Program-35**							
36	**Program-36**							
37	**Program-37**							
38	**Program-38**							
39	**Program-39**							
40	**Program-40**							
41	**Program-41**							

Program 1

Problem Statement: Given an array of nonnegative integers, design a linear

algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(n), where n is the size of input)

Algorithm:

```
Algorithm linearSearch(a[0...n-1],n,key){

        no_of_comparisons = 0

        for i=0 to n
        {

                no_of_comparisons++ if(a[i] == key)
                Print " key is found"
                return no_of_comparisons

        }
        if(i==n)
        Print " key is NOT found"

        return no_of_comparisons

}
```

Complexity Analysis:
Time Complexity:
Worst Case: When Element is either Present at last index of array or not present in array
T(n) = (n) [Linear]
Best Case: When Element is Present at first index or nearby index.
T(n)= (1) [Constant]
Average Case: When Element is Present in between first and last index of array.
T(n) = (n) [Linear]

Space Complexity:

Worst Case=Best Case = Average case = (1) [Constant]

Source code:

/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/

```
#include<iostream>
using namespace std;
int main()
{
	int N,T,search,count;
	cin>>T;
	while(T--)
	{
		count=0;
		cin>>N;
		int arr[N];
		for(int i=0;i<N;i++)
		{
			cin>>arr[i];
		}
		cin>>search;
		for(int i=0;i<N;i++)
		{
			if(search==arr[i])
			{
				cout<<"Present "<<count+1<<endl;
				break;
			}
			count++;
		}
		if(count==N)
		{
			cout<<"Not Present "<<N<<endl;
		}

	}
	return 0;
}
```

Output

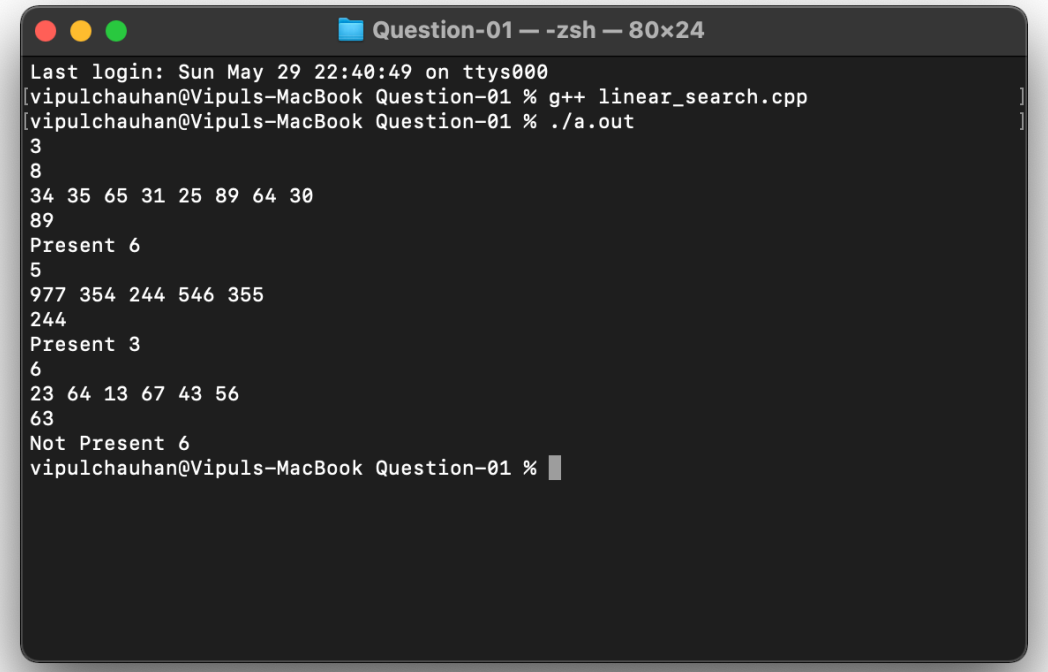
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 2

Problem Statement: Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the array or not. Also, find the total number of comparisons for each input case.

Algorithm:

```
Algorithm find_element(arr[0...n-1], n ) {

        low =0, high = n - 1 , count = 0

        while(low <= high)
        {

                count++
                mid = (low + high) / 2 IF( arr.mid == key)
                {

                        print " Key found at ", mid;
                        print " Total comparisons are", count

                        Terminate program

                }
                ELSE IF( arr.mid < key)

                        low = mid + 1

                ELSE

                        high = mid - 1

        }
        print "Element not found"
        print "Total comparisons", count

}
```

Complexity Analysis:

Time Complexity:

Worst Case:
T(n) = O(log n) [logarithmic]
Best Case: When Element is Present at middle index of array T(n)= (1) [Constant]
Average Case:
T(n) = (log n) [logarithmic]

Space Complexity:

Worst Case= Best Case = Average case = (1) [Constant]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/
#include<iostream>
using namespace std;
int main()
{
	int N,T,search,count,low,high,mid,flag;
	cin>>T;
	while(T--)
	{
		count=0;
		flag=0;
		cin>>N;
		int arr[N];
		for(int i=0;i<N;i++)
		{
			cin>>arr[i];
		}
		cin>>search;
		low=0;
		high=N-1;
		while(low<=high)
		{
			mid=(low+high)/2;
			if(search==arr[mid])
			{
				cout<<"Present "<<count<<endl;
				flag=1;
```

```
                    break;
                }
                else if(search<arr[mid] )
                {
                    high=mid-1;
                    count++;
                }
                else
                {
                    low=mid+1;
                    count++;
                }
            }
            if(flag==0)
            {
                cout<<"Not Present "<<count<<endl;
            }
        }

        return 0;
}
```

Output

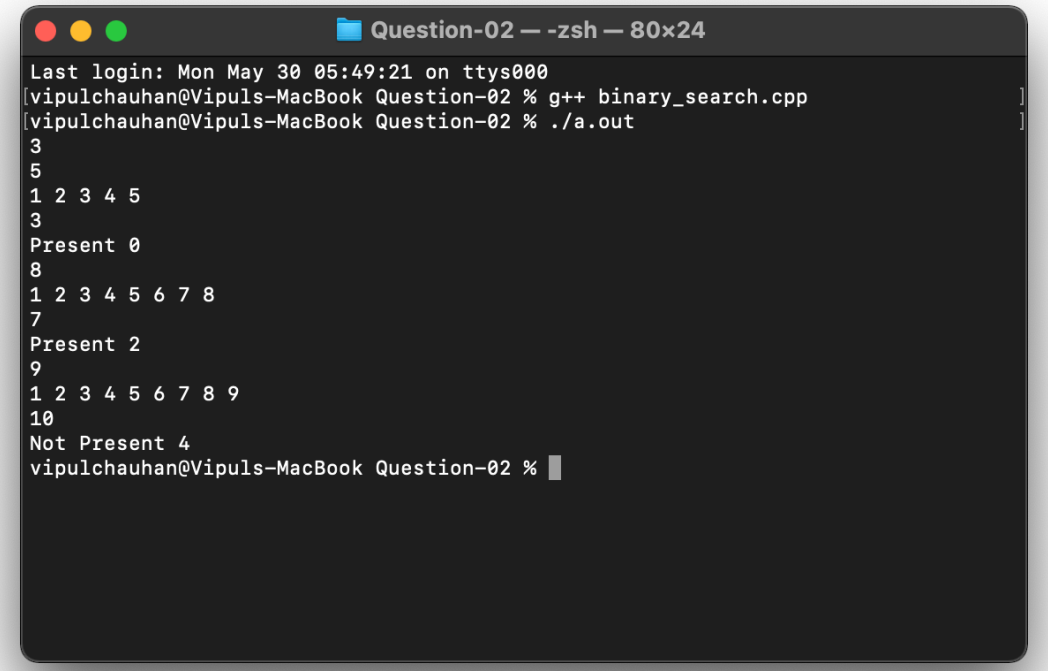
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 3

Problem Statement: Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array arr[n], search at the indexes arr[0], arr[2], arr[4],. ,arr[2^k] and so on. Once the interval (arr[2^k] < key < arr[2^{k+1}]) is found, perform a linear search operation from the index 2^k to find the element key.

Algorithm:

STEP 1: Initialize k = 0
STEP 2: Jump from index 2^k to index 2^(k+1);
STEP 3: Repeat Step 2 till key > arr[n^k] and key < arr[2^(k+1)] else move to next Step
STEP 4: IF(arr[2^k] < key < arr[2^{k+1}]) {

Perform linear search between this and search for element

}

ELSE {

print “Element not found”

}

STEP 5: END

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = O(root n) Best Case: T(n)= (root n) Average Case: T(n) = (root n)

Space Complexity:

Worst Case= Best Case = Average case = (1) [Constant]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
using namespace std;
void linear_search(int *A,int i,int n,int search)
{
   for(;i<n;i++)
   {
      if(A[i]==search)
      {
         cout<<search<<" is found at index "<<i+1<<endl;
         return;
      }
   }
   cout<<search<<" is not found "<<endl;
}
void jump(int *A,int n,int search)
{
   int i;
   for(i=1;i<n;i=i*2)
   {
      if(A[i]>search)
      {
         break;
      }
   }
   linear_search(A,i/2,n,search);
```

```
}

int main()
{
    int N,T,search;
            cin>>T;
            while(T--)
            {
                cin>>N;
                int arr[N];
                for(int i=0;i<N;i++)
                {
                            cin>>arr[i];
                }
                cin>>search;
        jump(arr,N,search);
            }
    return 0;
}
```

Output

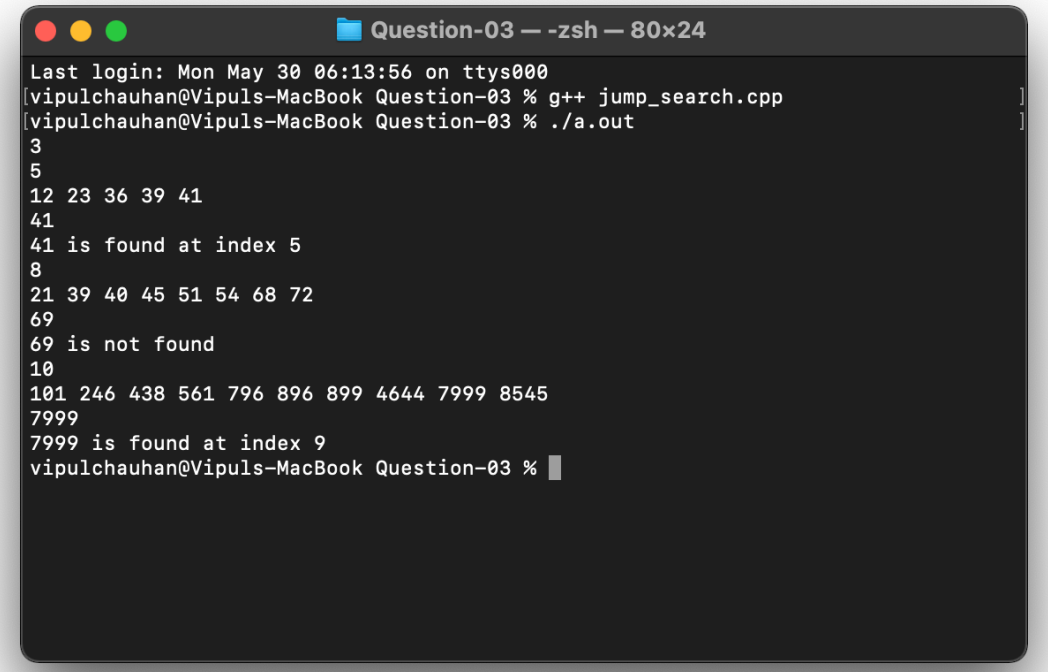
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 4

Problem Statement: Given a sorted array of positive integers containing few duplicate elements, design an algorithm and implement it using a program to find whether the given key element is present in the array or not. If present, then also find the number of copies of the given key.

Algorithm:

```
Algorithm count_duplicate(arr[0...n], n) {

        count = 0
        low = 0
        high = n - 1
        while (low <= high) {

                mid = (low+high) / 2

                if(arr.mid == key){

                        count ++

                        low = mid +1

                }

                ELSE IF ( arr.mid < key)

                        low = mid + 1

                ELSE
                        high = mid - 1

        }
        IF count == 0

                print “Element not found”

        ELSE

                print “ELement found {count} times”

}
```

Complexity Analysis:

Time Complexity:
Worst Case:T(n) = (log n) [logarithmic]

Best Case: T(n)= (1) [Constant]

Average Case:T(n) = (log n) [logarithmic]

Space Complexity:

Worst Case= Best Case = Average case = (1) [Constant]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/


#include<iostream>
using namespace std;
int occurrence(int *arr,int n,int x)
{
   int *low=lower_bound(arr,arr+n,x);
   if(low==(arr+n)||*low!=x)
   {
      return 0;
   }
   int *high=upper_bound(arr,arr+n,x);
   return high-low;
}
int main()
{
   int T,N,search;
   cin>>T;
   while(T--)
   {
      cin>>N;
      int arr[N];
      for(int i=0;i<N;i++)
      {
         cin>>arr[i];
      }
      cin>>search;
      cout<<search<<"-"<<occurrence(arr,N,search)<<endl;
   }
}
```

Output

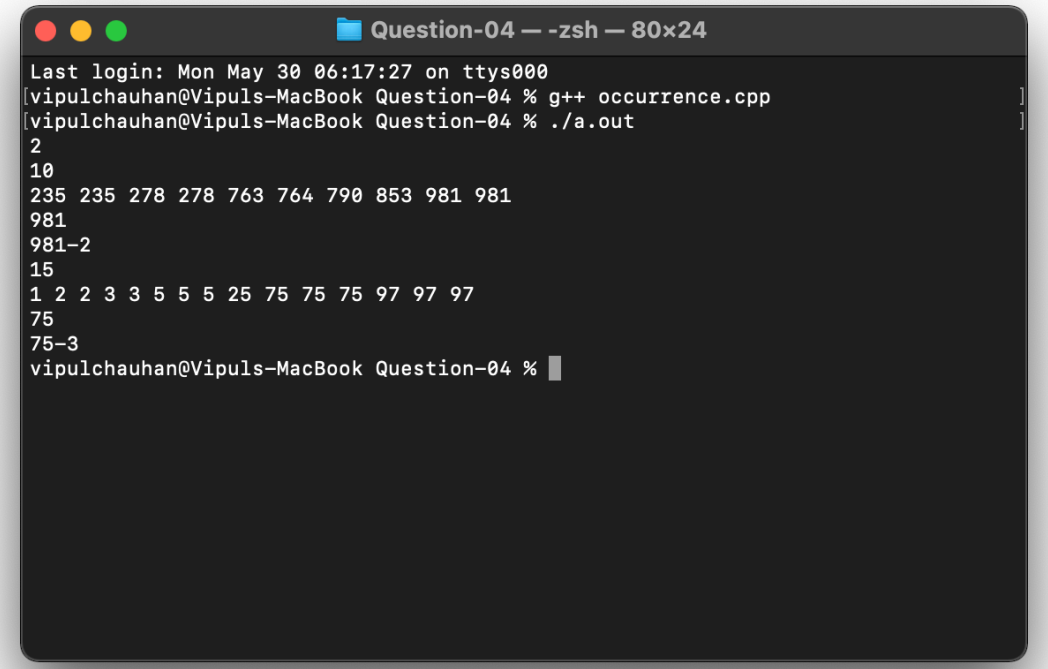
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 5

Problem Statement: Given a sorted array of positive integers, design an algorithm and implement it using a program to find three indices i, j, k such that arr[i] + arr[j] = arr[k]. Your Program Should run for T no of Test cases imputed by user.

Algorithm:

```
Algorthm sum(a[0 to n-1],n) {

        Sort the Array

        Flag=true
        for (k=0 to n-1) {

        i=0

        j=n-1

        while(i<j){

                if(a[i]+a[j] = a[k]){

                        Flag=false

                        Print i,j,k

                        i++
                        j++

                }

                If(a[i]+a[j]<a[k])

                        i++

                ELSE
                        j- -

        }

        If(Flag = true)
                Print “No Sequence Found”

}
```

Complexity Analysis:

Time Complexity:

Worst Case: Two nested loops traversing till n T(n) = $O(O^2)$

Best Case: Two nested loops traversing till n T(n)= $O(O^2)$

Average Case: Two nested loops traversing till n T(n) = $O(O^2)$ [Linear]

Space Complexity:

Worst Case=Best Case = Average case = $O(1)$ [Constant]

Source code:

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
#include<vector>
using namespace std;
vector<int> pairs(int *a,int n)
{
   vector<int> A;
   for(int i=0;i<n;i++)
   {
      for(int j=i+1;j<n;j++)
      {
         for(int k=j+1;k<n;k++)
         {
            if(a[i]+a[j]==a[k])
            {
               A.push_back(i);
               A.push_back(j);
               A.push_back(k);
            }
         }
      }
   }
   return A;
}
int main()
{
   int T,N;
```

```
    cin>>T;
    while(T--)
    {
        cin>>N;
        int A[N];
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        vector<int> B=pairs(A,N);
        if(B.size()==0)
        {
            cout<<"No Sequence found"<<endl;
        }
        else
        {
            for(int i=0;i<B.size();i++)
            {
                if(i%3!=0 ||i==0)
                {
                    cout<<B[i]<<" ";
                }
                else
                {
                    cout<<" and ";
                }
            }
            cout<<endl;
        }
    }
    return 0;
}
```

Output

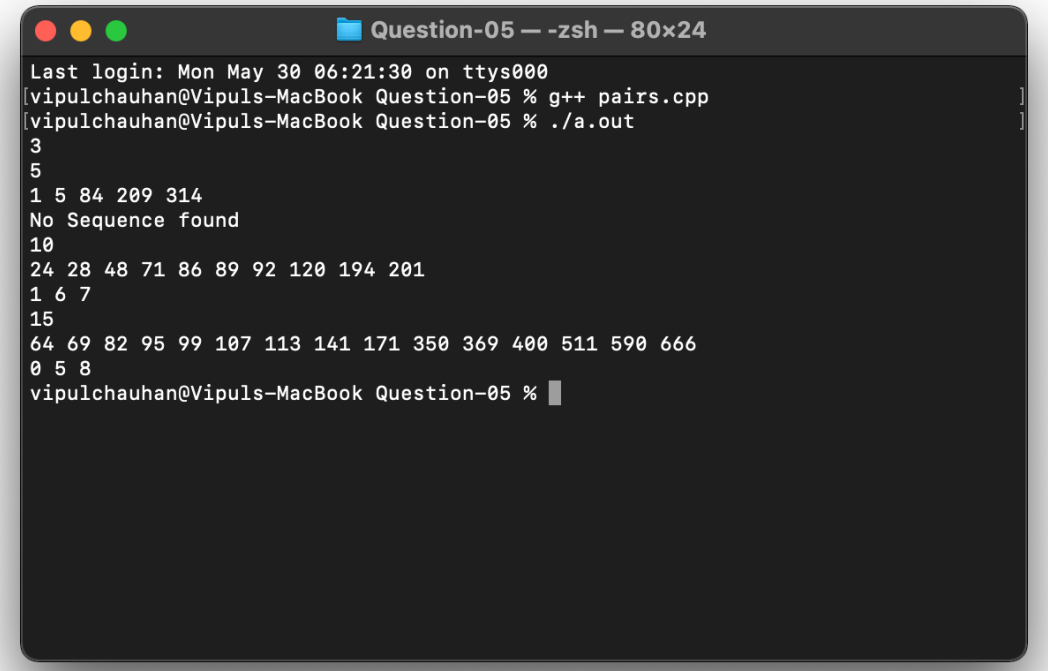
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 6

Problem Statement: Given an array of nonnegative integers, design an algorithm and a program to count the number of pairs of integers such that their difference is equal to a given key, K.

Algorithm

```
difference (a[0 to n-1] , n, k){

        Sort the array

        count,i, r = 0;

        while(r < n){

                if(arr[r] - arr[l] = k){

                        count++;

                l++; r++;

                }

                else if(arr[r] - arr[l] > k)

                        i++;

                else

                        r++;

        }
        Print Count

}
```

Complexity Analysis:

Time Complexity:
Worst Case: Sorting with complexity nlogn
T(n) = $O(n \log n)$
Best Case: Sorting with complexity nlogn
T(n) = $O(n \log n)$
Average Case: Sorting with complexity nlogn
T(n) = $O(n \log n)$

Space Complexity:
Worst Case=Best Case = Average case = $O(1)$ [Constant]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
using namespace std;
int pairs(int *a,int n,int key)
{
   int count=0;
   for(int i=0;i<n;i++)
   {
      for(int j=0;j<n && j!=i;j++)
      {
         if(abs(a[i]-a[j])==key)
         {
            count++;
         }
      }
   }
   return count;
}
int main()
{
   int T,N,key;
   cin>>T;
   while(T--)
   {
      cin>>N;
      int A[N];
```

```
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        cin>>key;
        int B=pairs(A,N,key);
        if(B==0)
        {
            cout<<"No Sequence found"<<endl;
        }
        else
        {
            cout<<B<<endl;
        }
    }
    return 0;
}
```

Output

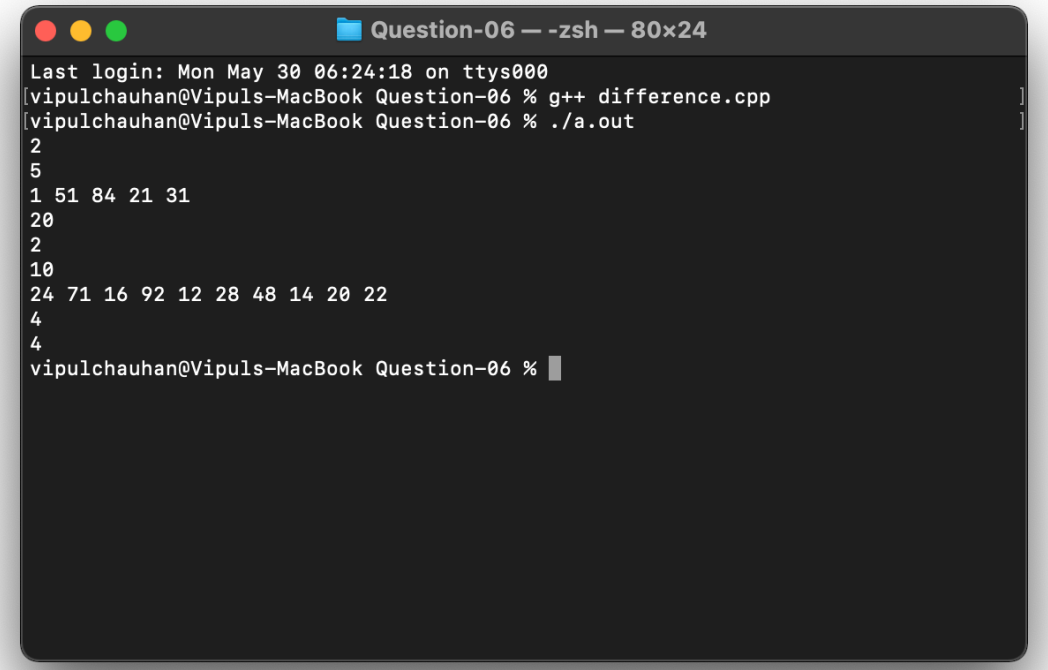
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 7

Problem Statement: Given an unsorted array of integers, design an algorithm and a program to sort the array using insertion sort. Your program should be able to find number of comparisons and shifts (shifts - total number of times the array elements are shifted from their place) required for sorting the array.

Algorithm:

Step 1- If the element is the first element, assume that it is already sorted. Return 1.

Step2 -Pick the next element, and store it separately in a key. Step3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element.

Step 5- Else, shift greater elements in the array towards the right. Step 5 - Insert the value. Step 6 - Repeat until the array is sorted.

Complexity Analysis:

Time Complexity:

Worst Case:T(n) = $O(n^2)$

Best Case:T(n)= $O(n)$ [Linear]

Average Case: T(n) = $O(n^2)$

Space Complexity:

Worst Case=Best Case = Average case = $O(1)$ [Constant]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
using namespace std;
void insertionSort(int *A,int N)
{
   int comp=0,shift=0;
   for(int i=0;i<N;i++)
   {
      int key=A[i];
      int j=i-1;
      while(j>=0 && A[j]>key)
      {
         shift++;
         comp++;
         A[j+1]=A[j];
         j--;
      }
      shift++;
      A[j+1]=key;
   }
   shift--;
   for(int i=0;i<N;i++)
   {
      cout<<A[i]<<" ";
   }
   cout<<endl<<"comparision = "<<comp<<endl<<"shifts = "<<shift<<endl;
```

```
}
int main()
{
    int T,N;
    cin>>T;
    while(T--)
    {
        cin>>N;
        int A[N];
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        insertionSort(A,N);
    }
    return 0;
}
```

Output

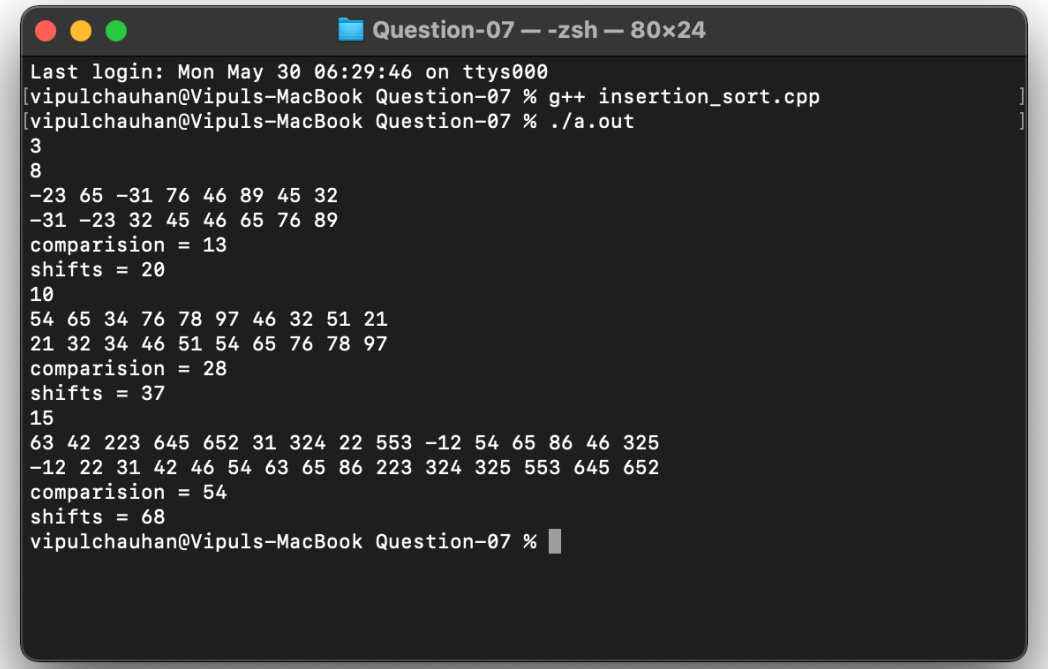
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 8

Problem Statement: Given an unsorted array of integers, design an algorithm

and implement a program to sort this array using selection sort. Your program should also find number of comparisons and number of swaps required. Agorithm:

Algorithm

```
SelectionSort(arr[] , n){
        S = 0; Comp = 0;
        for(I = 0 to n-1){
                min = I;
                for(j = i+1 to n-1) {
                        comp++
                        if( arr[j] < arr[min] ){
                                min = j;
                        }
                }
        }
        S++;
        Swap(arr[min],arr[i]);
}
Print Comp; Print S;
```

Complexity Analysis:

Time Complexity:

Worst Case:T(n) = $O(n^2)$

Best Case:T(n)= $O(n^2)$

Average Case:T(n) = $O(n^2)$

Space Complexity:

Worst Case=Best Case = Average case = $O(1)$ [Constant]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
using namespace std;
void selectionSort(int *A,int N)
{
   int comp=0,swaps=0;
   for(int i=0;i<N-1;i++)
   {
      int min=i;
      for(int j=i+1;j<N;j++)
      {
         comp++;
         if(A[j]<A[min])
         {
            min=j;
         }
      }
      swaps++;
      swap(A[min],A[i]);
   }
   for(int i=0;i<N;i++)
   {
      cout<<A[i]<<" ";
   }
   cout<<endl<<"comparision = "<<comp<<endl<<"swaps = "<<swaps<<endl;
}
```

```
int main()
{
    int T,N;
    cin>>T;
    while(T--)
    {
        cin>>N;
        int A[N];
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        selectionSort(A,N);
    }
    return 0;
}
```

Output

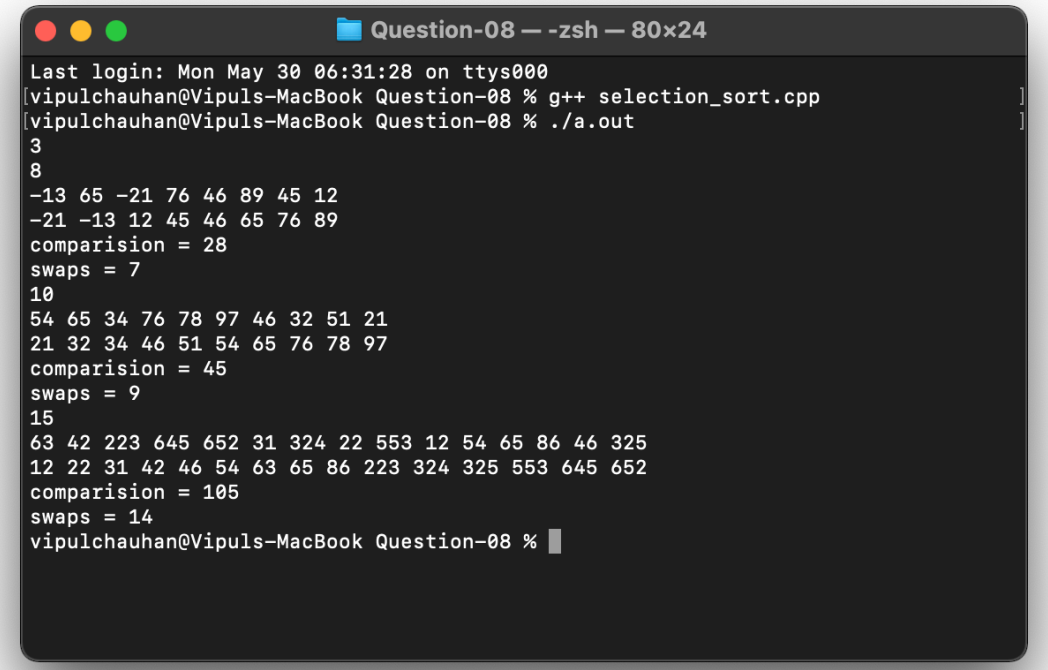
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 9

Problem Statement: Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements in the array or not.

Algorithm:

```
Function MERGE_SORT(arr, low, high)

Step - 1- IF( low < high)
Step - 2- set mid = (low + high)/2
Step - 3- MERGE_SORT(arr, low, mid)

Step - 4- MERGE_SORT(arr, mid + 1, high)

Step - 5- MERGE (arr, low, mid, high)

        end of IF

END MERGE_SORT

Function find_duplicate(arr, n)

{

        MERGE_SORT(arr, 0 , n-1)

        for i = 1 to n-1, Step 1

                IF(arr[i] == arr[i - 1])

                        return arr[i]

                end IF

        END for

}
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = O(n log n)

Best Case: T(n)= O(n log n)

Average Case: T(n) = O(n log n)

Space Complexity:

Worst Case= Best Case = Average case =O (n)

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/
#include<iostream>
using namespace std;

void merge(int a[],int l,int mid,int h)
{
   int i=l,j=mid+1,k=0;
   int temp[h-l+1];
   while(i<=mid && j<=h)
   {
      if(a[i]<a[j])
         temp[k++]=a[i++];
      else
         temp[k++]=a[j++];
   }
   while(i<=mid)
      temp[k++]=a[i++];
   while(j<=h)
      temp[k++]=a[j++];
   int x,y=0;
   for(x=l;x<=h;x++)
   {
      a[x]=temp[y];
      y++;
   }
}
void mergesort(int a[],int l,int h)
{
```

```
    if(l<h)
    {
        int mid= (l+h)/2;
        mergesort(a,l,mid);
        mergesort(a,mid+1,h);
        merge(a,l,mid,h);
    }
}
int main()
{
    int T,N;
    string ans;
    cin>>T;
    while(T--)
    {
        ans="NO";
        cin>>N;
        int A[N];
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        mergesort(A,0,N-1);
        for(int i=0;i<N-1;i++)
        {
            if(A[i]==A[i+1])
            {
                ans="YES";
            }
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

Output

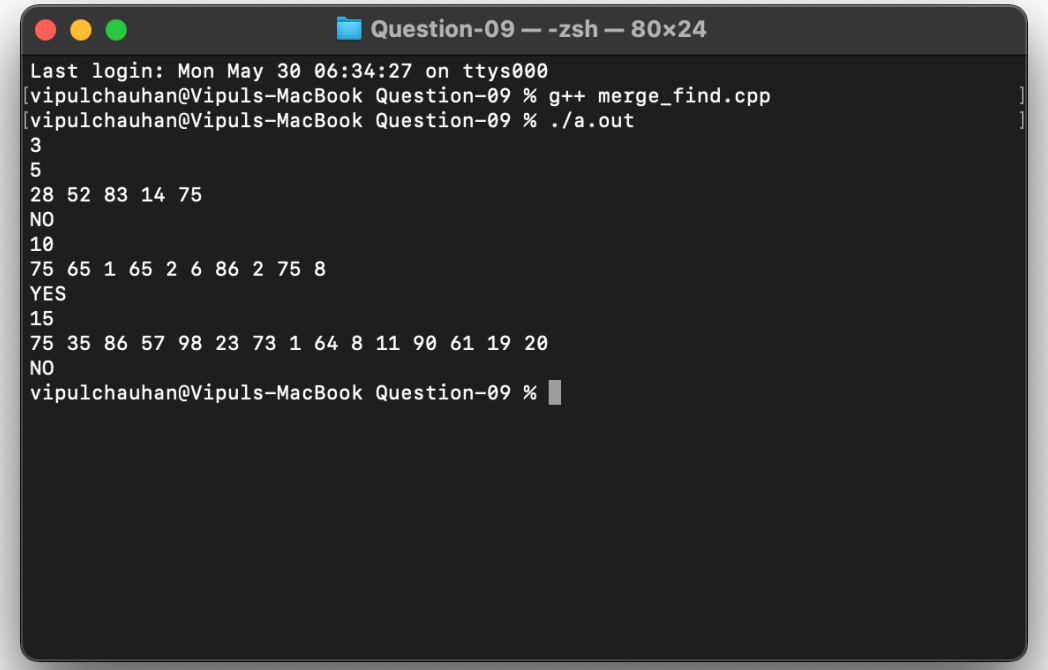
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 10

Problem Statement: Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array.

Algorithm:

MERGE_SORT(arr, beg, end)

1.if beg<end
2.set mid =(beg + end)/2

3.MERGE_SORT(arr,beg,mid)

4.MERGE_SORT(arr,mid+1,end)

5.MERGE (arr,beg,mid,end)

6.endof if
7.END MERGE_SORT

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = O(nlogn)

Best Case: T(n)= O(nlogn)

Average Case: T(n) = O(nlogn)

Space Complexity:

Worst Case=Best Case = Average case = O(logn) (Stack space)

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include <iostream>
using namespace std;
int merge(int *arr,int l,int mid,int h,int *comp)
{
   int inver=0;
   int i=l,j=mid+1,k=l;
   int temp[h+1];
   while(i<=mid && j<=h)
   {
      (*comp)++;
      if(arr[i]<=arr[j])
      {
         temp[k++]=arr[i++];
      }
      else
      {
         inver+=(mid+1-i);
         temp[k++]=arr[j++];
      }
   }
   while(i<=mid)
   {
      temp[k++]=arr[i++];
   }
   while(j<=h)
```

```
    {
        temp[k++]=arr[j++];
    }
    for(int p=l;p<=h;p++)
    {
        arr[p]=temp[p];
    }
    return inver;
}
int mergesort(int *arr,int l,int h,int *comp)
{
    int mid;
    int inver=0;
    if(l<h)
    {
        mid=(l+h)/2;
        inver+=mergesort(arr,l,mid,comp);
        inver+=mergesort(arr,mid+1,h,comp);
        inver+=merge(arr,l,mid,h,comp);
    }
    return inver;
}
int main()
{
    int N,T;
    cin>>T;
    while(T--)
    {
        int comp=0,inver=0;
        cin>>N;
        int arr[N];
        for(int i=0;i<N;i++)
```

```
        {
            cin>>arr[i];
        }
        inver=mergesort(arr,0,N-1,&comp);
        for(int i=0;i<N;i++)
        {
            cout<<arr[i]<<" ";
        }
        cout<<endl;
        cout<<"Comparison :- "<<comp<<endl<<"Inversion :- "<<inver<<endl;
    }
    return 0;
}
```

Output

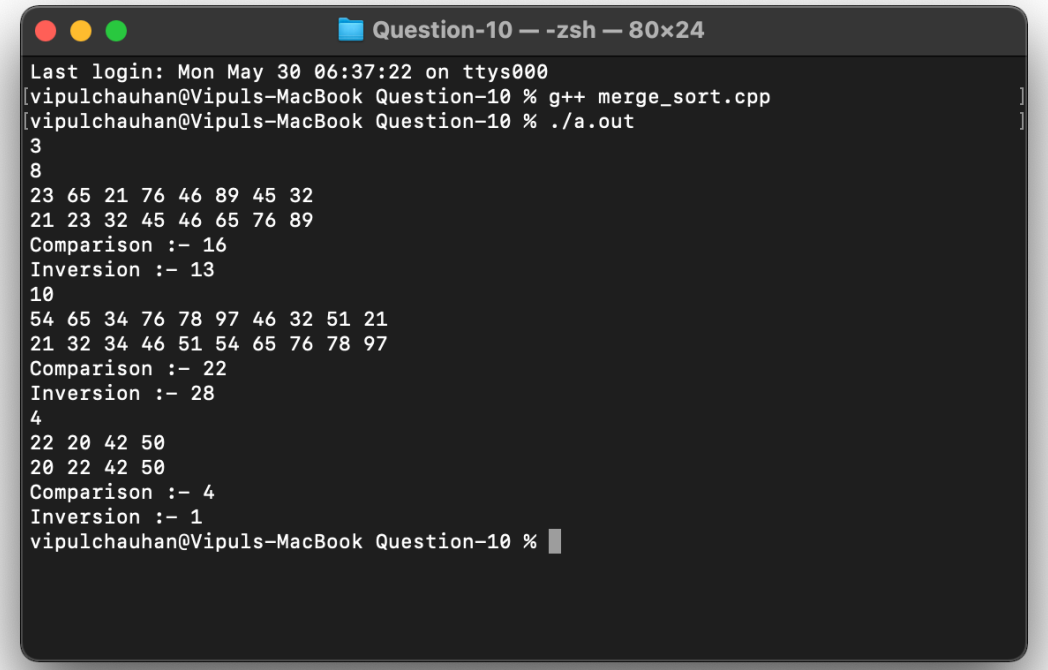
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 11

Problem Statement: Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.

Algorithm:

randQuickSort(arr[], low, high)
1. If low >= high, then EXIT.
2. While pivot 'x' is not a Central Pivot.

(i) Choose uniformly at random a number from [low..high]. Let the randomly picked number number be x.
(ii) Countelementsinarr[low..high]thataresmaller than arr[x]. Let this count be sc.
(iii) Count elements in arr[low..high] that are greater than arr[x]. Let this count be gc.
(iv) Let n = (high-low+1). If sc >= n/4 and gc >= n/4, then x is a central pivot.

3. Partition arr[low..high] around the pivot x.

4.// Recur for smaller elements

randQuickSort(arr, low, sc-1)

5. // Recur for greater elements

randQuickSort(arr, high-gc+1, high)

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = O(nlogn)

Best Case: T(n)= O(nlogn)

Average Case: T(n) = O(nlogn)

Space Complexity:

Worst Case=Best Case = Average case = O(logn) (Stack space)

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int comp,swp;
int partion(int *arr,int l,int r)
{
   srand(time(NULL));
   int random=(l+rand())%(r+l);
   swap(arr[random],arr[r]);
   int i,j,pivot;
   pivot=arr[r];
   i=l-1;
   for(j=l;j<r;j++)
   {
      comp++;
      if(arr[j]<pivot)
      {
         i++;
         swp++;
         swap(arr[i],arr[j]);
      }
   }
   swp++;
   swap(arr[i+1],arr[r]);
   return i+1;
}
```

```
void randomize_quick_sort(int arr[],int l,int r)
{
   if(l<r)
   {
      int pivot=partion(arr,l,r);
      randomize_quick_sort(arr,l,pivot-1);
      randomize_quick_sort(arr,pivot+1,r);
   }
   return;
}
int main()
{
   int T,N;
   cin>>T;
   while(T--)
   {
      comp=0;
      swp=0;
      cin>>N;
      int arr[N];
      for(int i=0;i<N;i++)
      {
         cin>>arr[i];
      }
      randomize_quick_sort(arr,0,N-1);
      for(int i=0;i<N;i++)
      {
         cout<<arr[i]<<" ";
      }
      cout<<endl<<"comparision = "<<comp<<endl<<"swaps = "<<swp<<endl;
   }
   return 0;
}
```

Output

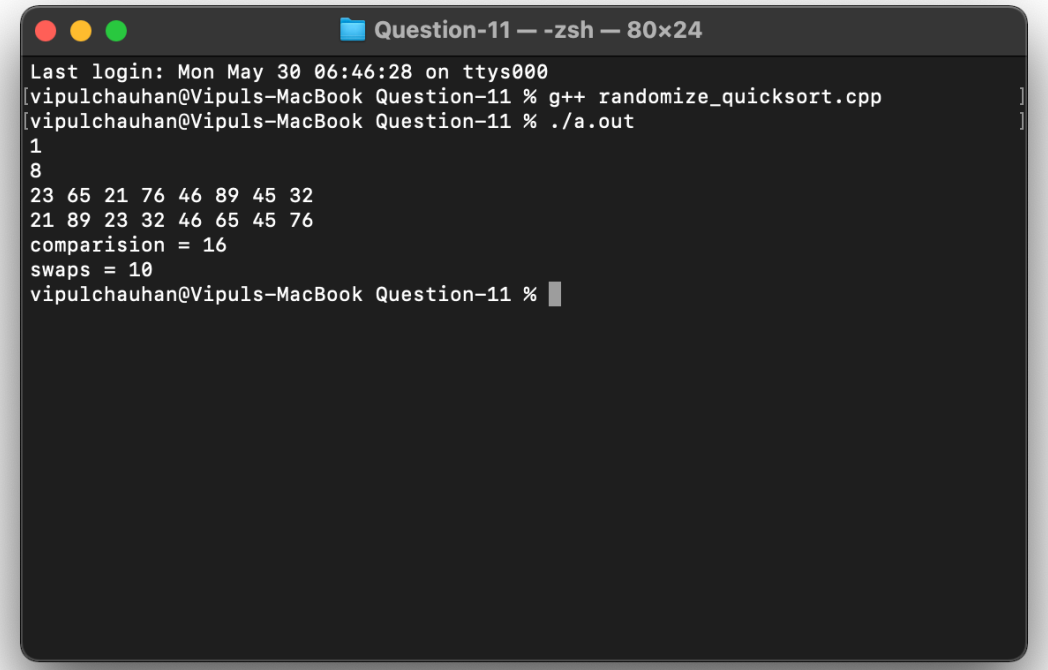
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 12

Problem Statement: Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

Algorithm (QuickSelect)

Step 1:Pick a pivot element and partition the array accordingly Step 2:Pick the rightmost element as pivot

Step 3:Reshuffle the array such that pivot element is placed at its rightful place — all elements less than the pivot would be at lower indexes, and elements greater than the pivot would be placed at higher indexes than the pivot

Step 4: If pivot is placed at the kth element in the array, exit the process, as pivot is the kth largest element

Step 5:If pivot position is greater than k, then continue the process with the left subarray, otherwise, recur the process with right subarray

Complexity Analysis:

Time Complexity:

Worst Case:T(n) = O(n)

Best Case: T(n)= (n)

Average Case: T(n) = (n)

Space Complexity:

Worst Case= Best Case = Average case = (log n) [Constant]

Source code:

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include <iostream>
using namespace std;
int paritition(int arr[],int l,int r)
{
   int i,j,pivot;
   pivot=arr[r];
   i=l-1;
   for(j=l;j<r;j++)
   {
      if(arr[j]<pivot)
      {
         i++;
         swap(arr[i],arr[j]);
      }
   }
   swap(arr[i+1],arr[r]);
   return i+1;
}
int kthSmallest(int arr[],int l,int r,int k)
{
   int parIndex=paritition(arr,l,r);
   if(parIndex==k-1)
      return arr[parIndex];
   if(parIndex>k-1)
      return kthSmallest(arr,l,parIndex-1,k);
```

```
    else
        return kthSmallest(arr,parIndex+1,r,k);
}
int main()
{
    int T,N,K;
    cin>>T;
    while(T--)
    {
        cin>>N;
        int A[N];
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        cin>>K;
        if(K>N)
            cout<<"not present";
        else
        {
            cout<<kthSmallest(A,0,N-1,K)<<endl;
        }
    }
    return 0;
}
```

Output

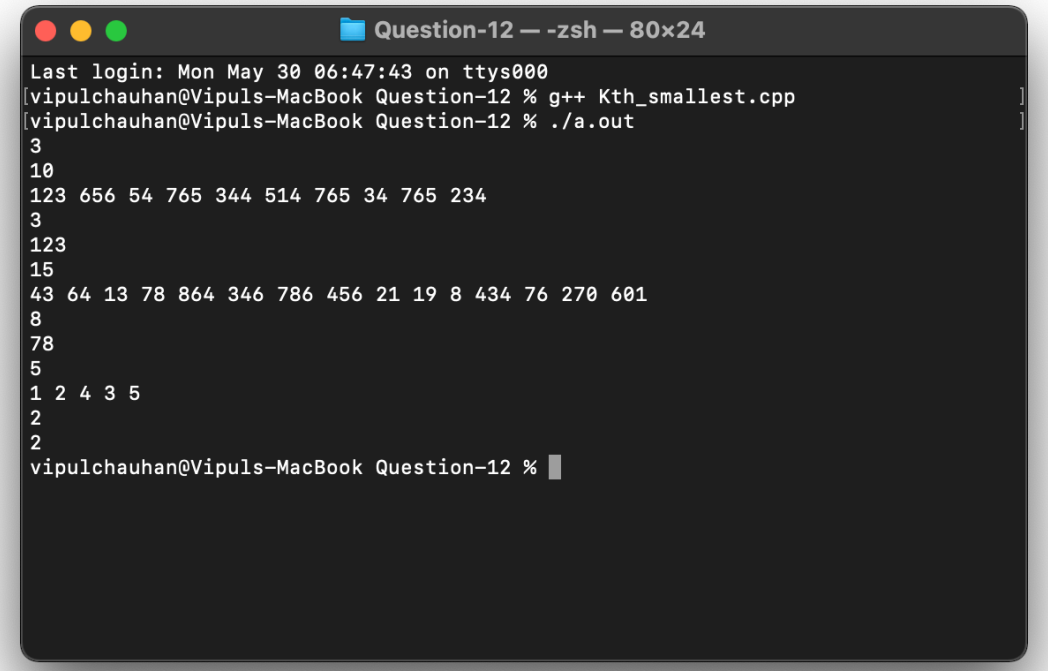
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 13

Problem Statement: Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and print it. (Time Complexity = O(n)) (Hint: Use counting sort)

Algorithm:

```
getMaxOccurringChar(str){
        Create a array count to keep the count of individual characters

        count[ASCII_SIZE] = {0}

        len = length_of_string(str)

        max = 0
        result = ‘’
        for i = 0 to len - 1{

                        count[ str.char_at(i) ]++
                        if( max < count[ str.char_at(i) ]{

                                max = count[ str.char_at(i) ]

                                result = str.char_at(i)

                        }

        }

        return result

}
```

Complexity Analysis:
Time Complexity:
// n is the number of elements in the array and k is the range of the elements (ASCII code of characters in string)

Worst Case: When data is skewed and range is large T(n) = O(n+k) [Linear]
Best Case: When all elements(characters) are same T(n)= (n) [Linear]

Average Case: N & K equally dominant T(n) = (n+k) [Linear]

Space Complexity:

Worst Case = Best Case = Average case = (k) [linear]

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
#include<climits>
#include<vector>
using namespace std;
int main()
{
   int T,N;
   cin>>T;
   while(T--)
   {
      vector<char> v;
      char temp;
      cin>>N;
      for(int i=0;i<N;i++)
      {
         cin>>temp;
         v.push_back(temp);
      }
      vector<int> freq(26,0);
      for(auto it:v)
         freq[it-'a']++;
      int mx=INT_MIN,ind;
      for(int i=0;i<26;i++)
      {
         if(mx<freq[i])
```

```
            {
                ind=i;
                mx=freq[i];
            }
        }
        if(mx!=1)
            cout<<char(ind+'a')<<"-"<<mx<<endl;
        else
            cout<<"No Duplicate Present"<<endl;
    }
    return 0;
}
```

Output

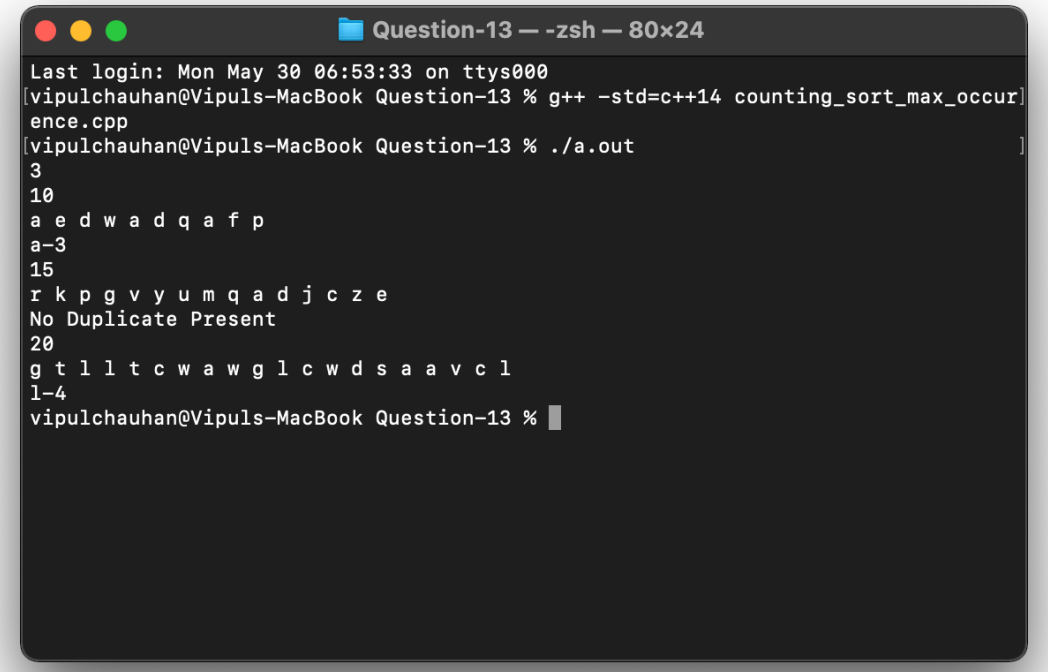
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 14

Problem Statement: Given an unsorted array of integers, design an algorithm and implement it using a program to find whether two elements exist such that their sum is equal to the given key element. (Time Complexity = O(n log n))

Algorithm:

```
findPair(num[0...n-1], n , target) {

        sort(nums)
        low = 0
        high = n -1
        while( low < high) {

                IF(nums[low] + nums[high] is equals to target) {

                        print “Pair found”, nums[low], nums[high]

                        terminate program

                }

                IF( nums[low] + nums[high] < target)

                        low++

                ELSE

                        high--

        }

        If no pair found — print “No pair found”

}
```

Complexity Analysis:

Time Complexity:

Worst Case: When data is skewed and range is large T(n) = O(n+k) [Linear]
Best Case: When all elements(characters) are same T(n)= (n) [Linear]
Average Case: N & K equally dominant T(n) = (n+k) [Linear]

Space Complexity:

Worst Case = Best Case = Average case = (k) [linear]

Source Code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include <iostream>
#include <vector>
using namespace std;
vector<int> twoSum(int arr[],int n,int k)
{
   int st=0,end=n-1;
   vector<int> ans;
   while(st<end)
   {
      if(arr[st]+arr[end]==k)
      {
         ans.push_back(arr[st]);
         ans.push_back(arr[end]);
         return ans;
      }
      else if(arr[st]+arr[end]>k)
         end--;
      else
         st++;
   }
   return ans;
}
void merge(int *arr,int l,int mid,int h)
{
   int i=l,j=mid+1,k=l;
```

```
    int temp[h+1];
    while(i<=mid && j<=h)
    {
        if(arr[i]<arr[j])
        {
            temp[k++] = arr[i++];
        }
        else
        {
            temp[k++] = arr[j++];
        }
    }
    while(i<=mid)
    {
        temp[k++] = arr[i++];
    }
    while(j<=h)
    {
        temp[k++] = arr[j++];
    }
    for(int p=l;p<=h;p++)
    {
        arr[p]=temp[p];
    }
}
void merge_sort(int *arr,int l,int h)
{
    if(l<h)
    {
        int mid=(l+h)/2;
        merge_sort(arr,l,mid);
        merge_sort(arr,mid+1,h);
```

```
        merge(arr,l,mid,h);
    }
}
int main()
{
    int T,N,key;
    cin>>T;
    while(T--)
    {
        cin>>N;
        int A[N];
        for(int i=0;i<N;i++)
        {
            cin>>A[i];
        }
        merge_sort(A,0,N-1);
        cin>>key;
        vector<int> ans=twoSum(A,N,key);
        if(ans.empty()==1)
        {
            cout<<"No Such Element Exist"<<endl;
        }
        else
        {
            for(auto it:ans)
            cout<<it<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

Output

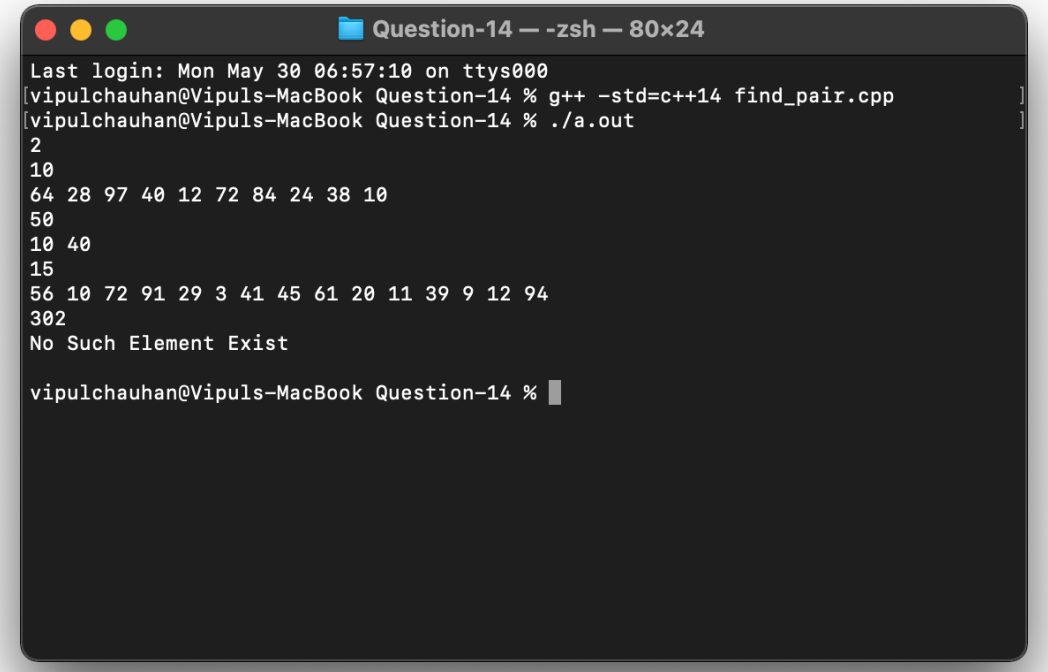
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 15

Problem Statement: You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find list of elements which are common to both. (Time Complexity = O(m+n))

Algorithm:

```
printIntersection(arr1[] , arr2[], m(size of arr1) , n(size of arr2) ) {

        i=0
        j=0
        WHILE ( i < m and j < n) {

                IF ( arr1[i] < arr2[j] )

                        i++

                ELSE IF( arr2[j] < arr1[i] )

                        j++

                ELSE {

                        print arr2[j]

                        i++
                        j++

                }

        }

}
```

Complexity Analysis:

Time Complexity:

Worst Case: When data is skewed and range is large T(n) = O(m+n) [Linear]
Best Case: When all elements(characters) are same T(n)= (m+n) [Linear]

Average Case: N & K equally dominant T(n) = (m+n) [Linear]

Space Complexity:
Worst Case = Best Case = Average case = (1)

Source code :

```
/*
Name: Vipul Chauhan
University Roll No: 2018855
Section: A
*/

#include<iostream>
using namespace std;
int main()
{
   int m,n;
   cin>>m;
   int arr1[m];
   for(int i=0;i<m;i++)
   {
      cin>>arr1[i];
   }
   cin>>n;
   int arr2[n];
   for(int i=0;i<n;i++)
   {
      cin>>arr2[i];
   }
   int i=0,j=0;
   while(i<m && j<n)
   {
      if(arr1[i]<arr2[j])
         i++;
      else if(arr1[i]>arr2[j])
         j++;
      else
```

```
        {
            cout<<arr2[j]<<" ";
            i++;
            j++;
        }
    }
    return 0;
}
```

Output

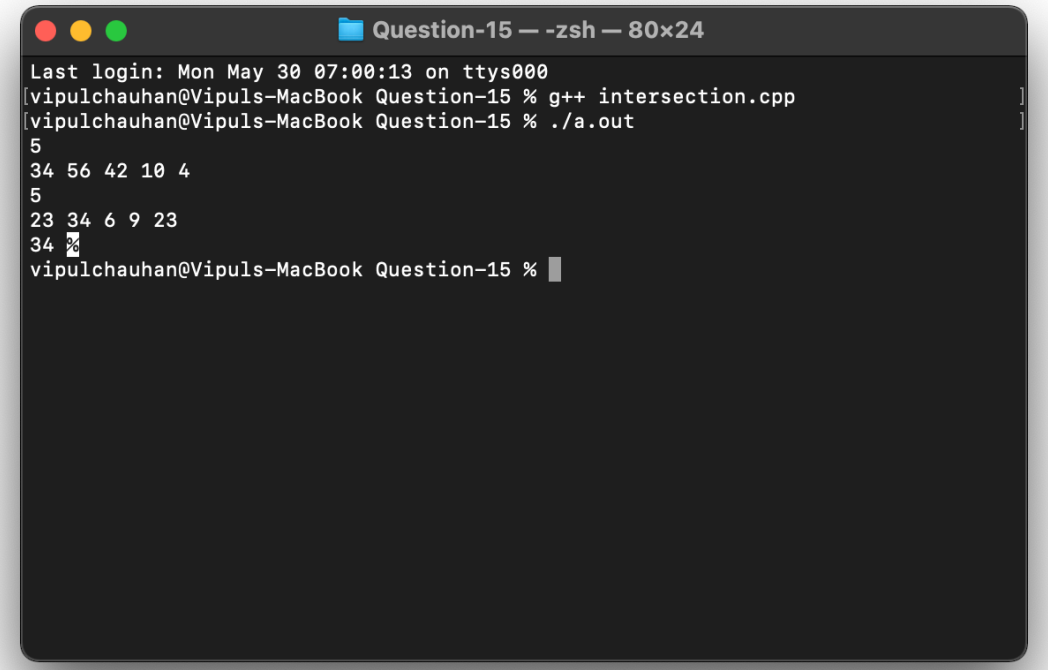
/*

Name: Vipul Chauhan

University Roll No: 2018855

Section: A

*/



Program 16

Problem Statement: Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)

Algorithm:

DFS Algorithm:

1. if **start==end** return **1** since we have to reached our destination.
2. Mark **start** as **visited**.
3. Traverse directly connected vertices of **start** and recur the function **dfs** for every such unexplored vertex.
4. return **0** if we do not reach our destination

Complexity Analysis:

Time complexity: O(V+E), V= vertices ,E = edges
Space complexity: O(V)

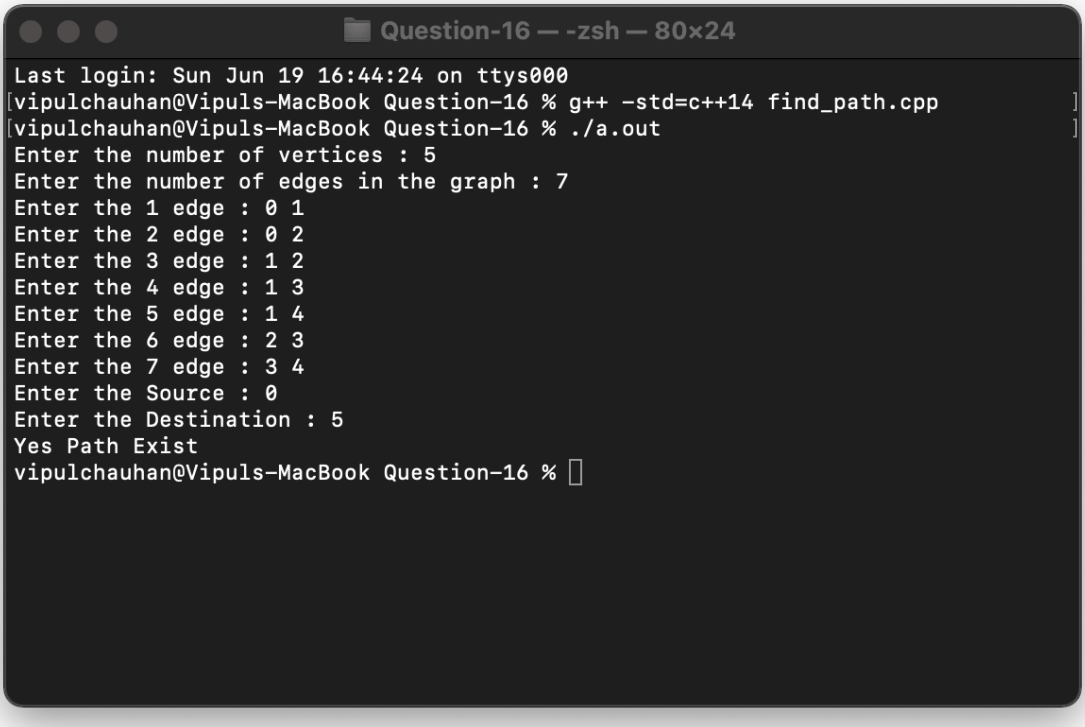
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include<iostream>
#include<vector>
using namespace std;
void addEdge(vector<int> A[],int u,int v)
{
   A[u].push_back(v);
   A[v].push_back(u);
}
void dfs(vector<int> A[],int u,bool visited[])
{
   visited[u]=true;
   for(auto it=A[u].begin();it!=A[u].end();it++)
   {
      if(!visited[*it])
      {
         dfs(A,*it,visited);
      }
   }
}
int main()
{
   int V,E;
   cout<<"Enter the number of vertices : ";
   cin>>V;
   vector<int> A[V];
   bool visited[V];
   cout<<"Enter the number of edges in the graph : ";
   cin>>E;
   for(int i=0;i<E;i++)
   {
      int u,v;
      cout<<"Enter the "<<i+1<<" edge : ";
      cin>>u>>v;
      addEdge(A,u,v);
   }
   cout<<"Enter the Source : ";
   int source,destination;
```

```
    cin>>source;
    cout<<"Enter the Destination : ";
    cin>>destination;
    dfs(A,source,visited);
    if(visited[destination]==true)
    {
       cout<<"Yes Path Exist"<<endl;
    }
    else
    {
       cout<<"No Such Path Exist"<<endl;
    }
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
***/**



Program 17

Problem Statement: Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

Algorithm:

1. Assign RED color to the source vertex (putting into set U).
2. Color all the neighbors with BLUE color (putting into set V).
3. Color all neighbor's neighbor with RED color (putting into set U).
4. This way, assign color to all vertices such that it satisfies all the constraints of m way coloring problem where m = 2.
5. While assigning colors, if we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices (or graph is not Bipartite)

Complexity Analysis:

Time complexity: O(V+E).
Space complexity: O(V+E).

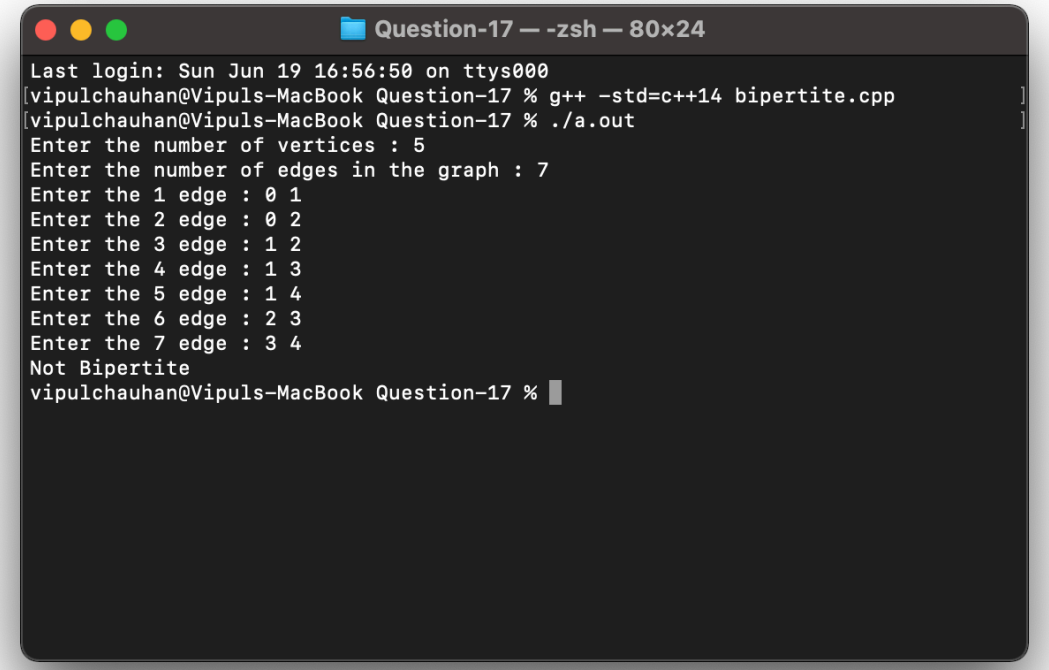
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include<iostream>
#include<queue>
#include<vector>
using namespace std;
void addEdge(vector<int> A[],int u,int v)
{
   A[u].push_back(v);
   A[v].push_back(u);
}
bool isBipartite(int V,vector<int>A[])
{
   vector<int> color(V,-1);
   queue<pair<int,int> > q;
   for(int i=0;i<V;i++)
   {
      if(color[i]==-1)
      {
         q.push({i,0});
         color[i]=0;
         while(!q.empty())
         {
            pair<int,int> p=q.front();
            q.pop();
            int a=p.first;
            int b=p.second;
            for(int j:A[a])
            {
               if(color[j]==b)
               {
                  return 0;
               }
               if(color[j]==-1)
               {
                  color[j]=(b)?0:1;
                  q.push({j,color[j]});
               }
            }
```

```
            }
        }
    }
    return 1;
}
int main()
{
    int V,E;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    vector<int> A[V];
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        int u,v;
        cout<<"Enter the "<<i+1<<" edge : ";
        cin>>u>>v;
        addEdge(A,u,v);
    }
    if(isBipartite(V,A))
    {
        cout<<"Yes Bipertite"<<endl;
    }
    else
    {
        cout<<"Not Bipertite"<<endl;
    }
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 18

Problem Statement: Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

Algorithm:

1. Create the graph using the given number of edges and vertices.
2. Create a recursive function that initializes the current index or vertex, visited, and recursion stack.
3. Mark the current node as visited and also mark the index in recursion stack.
4. Find all the vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices, If the recursive function returns true, return true.
5. If the adjacent vertices are already marked in the recursion stack then return true.
6. Create a wrapper class, that calls the recursive function for all the vertices and if any function returns true return true. Else if for all vertices the function returns false return false.

Complexity Analysis:

Time complexity: O(V+E).
Space complexity: O(V).

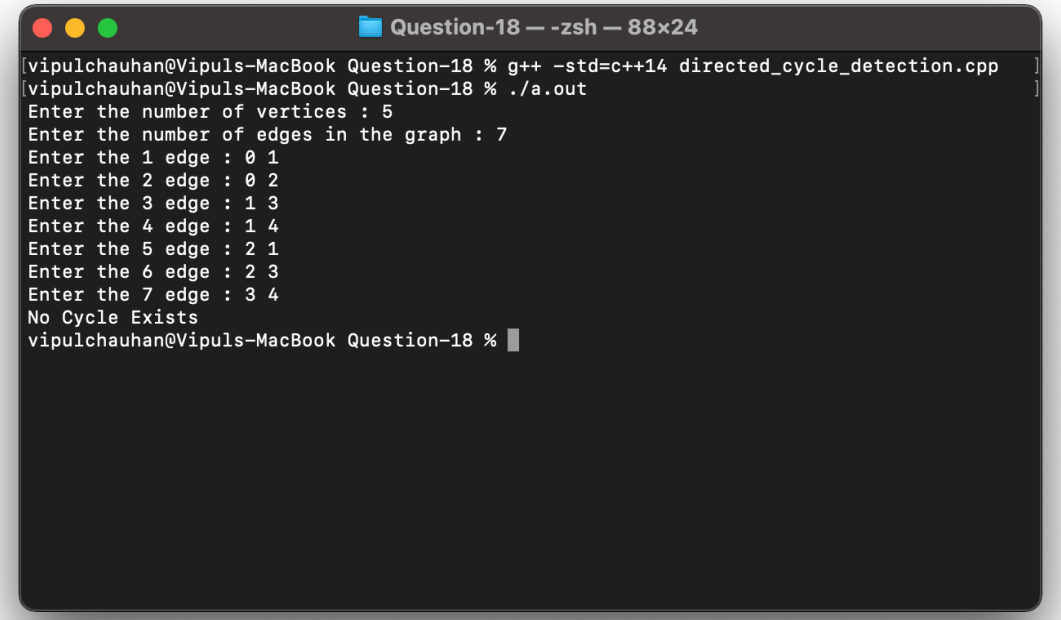
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include<iostream>
#include<vector>
using namespace std;
void addEdge(vector<int>A[],int u,int v)
{
   A[u].push_back(v);
}
bool isCycleUtil(vector<int> A[],int v,bool visited[],bool recStack[])
{
   if(visited[v]==false)
   {
      visited[v]=true;
      recStack[v]=true;
      for(auto i=A[v].begin();i!=A[v].end();i++)
      {
         if(!visited[*i] && isCycleUtil(A,*i,visited,recStack))
         {
            return true;
         }
         else if(recStack[*i])
         {
            return true;
         }
      }

   }
   recStack[v]=false;
   return false;
}
bool cycle(vector<int>A[],int V)
{
   bool visited[V];
   bool recStack[V];
   for(int i=0;i<V;i++)
   {
      visited[i]=false;
      recStack[i]=false;
```

```
    }
    for(int i=0;i<V;i++)
    {
        if(!visited[i] && isCycleUtil(A,i,visited,recStack))
        {
            return true;
        }
    }
    return false;
}
int main()
{
    int V,E;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    vector<int> A[V];
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        int u,v;
        cout<<"Enter the "<<i+1<<" edge : ";
        cin>>u>>v;
        addEdge(A,u,v);
    }
    if(cycle(A,V))
    {
        cout<<"Yes Cycle Exist"<<endl;
    }
    else
    {
        cout<<"No Cycle Exists"<<endl;
    }
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 19

Problem Statement: After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay's house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends' location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend's location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra's algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay's results? (Hint: Print shortest path and distance from friends' location to Akshay's house).

Algorithm:

1) Initialize distances of all vertices as infinite.

2) Create an empty **p**riority_**q**ueue **pq**. Every item of pq is a pair (weight, vertex). Weight (or distance) is used as first item of pair as first item is by default used to compare two pairs.
3) Insert source vertex into pq and make its distance as 0.
4) While either pq doesn't become empty
   a) Extract minimum distance vertex from pq. Let the extracted vertex be u.
   b) Loop through all adjacent of u and do following for every vertex v.

```
// If there is a shorter path to v
// through u.
If dist[v] > dist[u] + weight(u, v)

   (i) Update distance of v, i.e., do
        dist[v] = dist[u] + weight(u, v)
   (ii) Insert v into the pq (Even if v is
        already there)
```

5) Print distance array dist[] to print all shortest paths.

Complexity Analysis:

Time complexity: O(ELogV)
Space complexity: O(V+E)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <list>
#include <queue>
#include <climits>
#define INF INT_MAX
using namespace std;

typedef pair<int,int> myPair;
class Graph
{
   public:
      int V;
      list<myPair> *adj;
      vector<int> parent;
      Graph(int V)
      {
         this->V=V;
         adj=new list<myPair>[this->V];
         parent.assign(V,-1);
      }
      void addEdge(int u,int v,int w)
      {
         adj[u].push_back(make_pair(v,w));
         adj[v].push_back(make_pair(u,w));
      }
      void print_path(int j)
      {
         if(parent[j]==-1)
         {
            return;
         }
         print_path(parent[j]);
         cout<<j<<" ";
      }
      void dijkstra_shortest_path(int src)
      {
         priority_queue<myPair,vector<myPair>,greater<myPair> > pq;
         vector<int> dist(this->V,INF);
         dist[src]=0;
```

```
        list<myPair>::iterator it;

        pq.push(make_pair(0,src));
        while(!pq.empty())
        {
            int u=pq.top().second;
            pq.pop();

            for(it=adj[u].begin();it!=adj[u].end();++it)
            {
                int v=it->first;
                int w=it->second;
                if(dist[v]>dist[u] + w)
                {
                    dist[v]=dist[u] + w;
                    pq.push(make_pair(dist[v],v));
                    parent[v]=u;
                }
            }
        }
        for(int i=0;i<this->V;i++)
        {
            cout<<src<<" ";
            print_path(i);
            cout<<" : "<<dist[i]<<endl;
        }
    }
};
int main()
{
    int V,E,s;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    Graph g(V);
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)    {
        int u,v,w;
        cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
        cin>>u>>v>>w;
        g.addEdge(u,v,w);
    }
    cout<<"Enter the source vertex : ";
    cin>>s;
    g.dijkstra_shortest_path(s);
    return 0;
}
```

OUTPUT

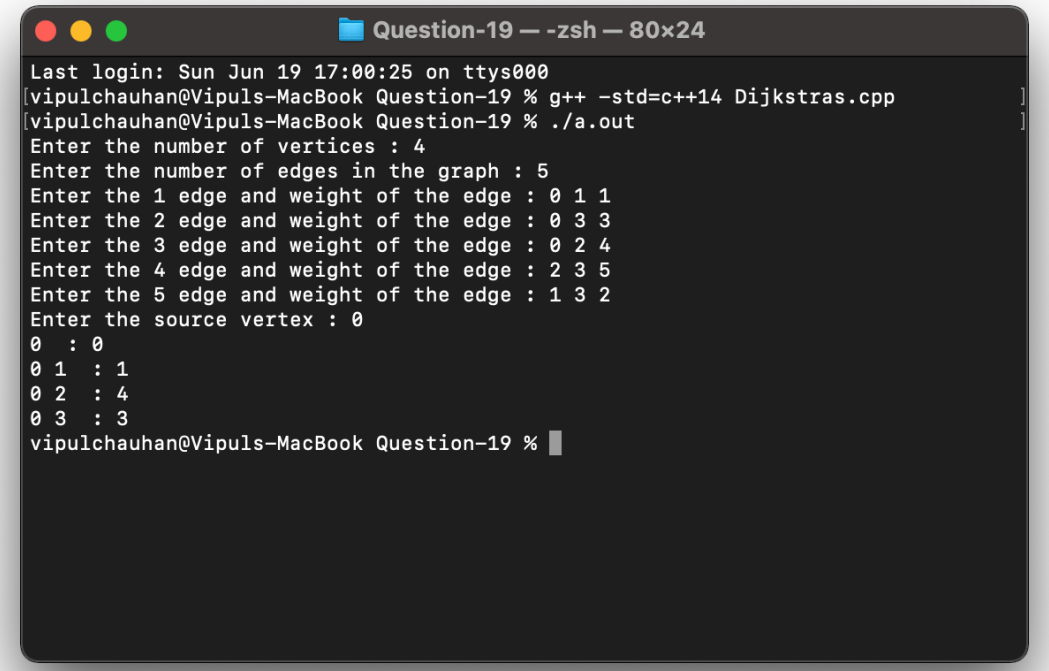
/*

Name : Vipul Chauhan

Univ. Roll No. : 2018855

Section : A

*/



Program 20

Problem Statement: Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

Algorithm:

1) This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
…..**a)** Do following for each edge u-v
If dist[v] > dist[u] + weight of edge uv, then update dist[v]
dist[v] = dist[u] + weight of edge uv

3) This step reports if there is a negative weight cycle in graph. Do following for each edge u-v . If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”

Complexity Analysis:

Time complexity:O(V+E)
Space complexity:O(N)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
#define INF INT_MAX
using namespace std;

class Graph
{
    public:
        int V;
        vector<vector<int> > edges;
        vector<int> parent;
        Graph(int V)
        {
            this->V=V;
            parent.assign(V,-1);
        }
        void addEdge(int u,int v,int w)
        {
            edges.push_back({u,v,w});
        }
        void print_path(int j)
        {
            if(parent[j]==-1)
            {
                return;
            }
            print_path(parent[j]);
            cout<<j<<" ";
        }
        void bellmanFord(int src)
        {
            vector<int> dist(this->V,INF);
            dist[src]=0;
            for(int i=0;i<this->V-1;i++)
            {
                for(auto e:edges)
```

```
            {
                int u=e[0];
                int v=e[1];
                int w=e[2];
                int c=dist[v];
                dist[v]=min(dist[v],dist[u]+w);
                parent[v]=(c==dist[v])?parent[v]:u;
            }
        }
        for(int i=0;i<this->V;i++)
        {
            cout<<src<<" ";
            print_path(i);
            cout<<" : "<<dist[i]<<endl;
        }
    }
};
int main()
{
    int V,E,s;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    Graph g(V);
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        int u,v,w;
        cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
        cin>>u>>v>>w;
        g.addEdge(u,v,w);
    }
    cout<<"Enter the source vertex : ";
    cin>>s;
    g.bellmanFord(s);
    return 0;
}
```

OUTPUT

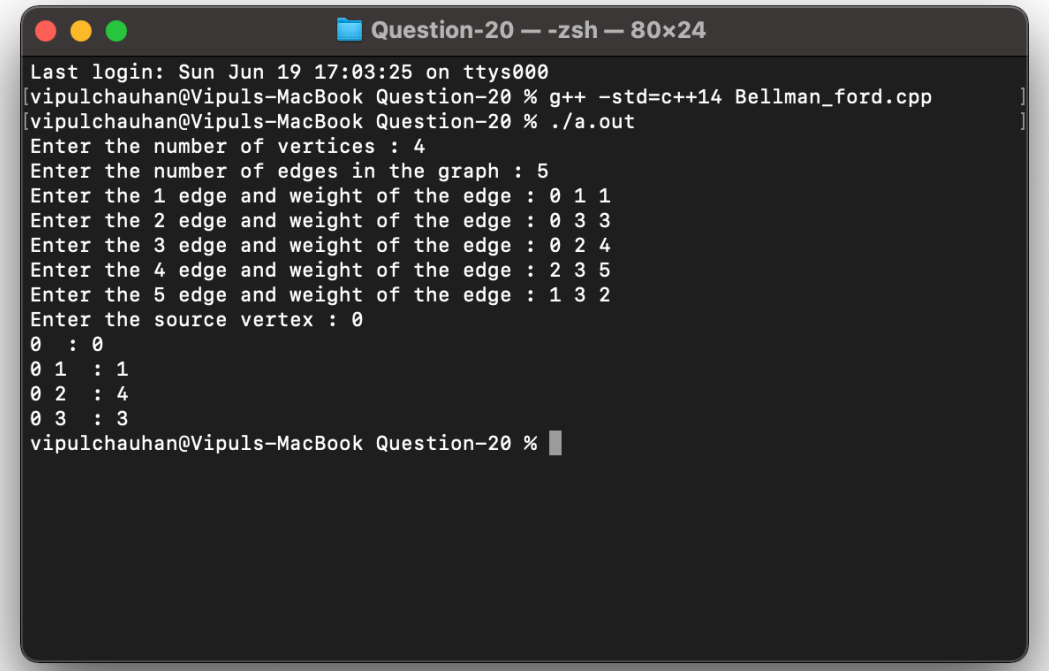
/*

Name : Vipul Chauhan

Univ. Roll No. : 2018855

Section : A

*/



Program 21

Problem Statement: Given a directed graph with two vertices (source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

Algorithm:

- graph[i][j] indicates the weight of an edge from vertex i to vertex j and a value INF(infinite) indicates no edge from i to j.
- start from u, go to all adjacent vertices, and recur for adjacent vertices with k as k-1, source as adjacent vertex and destination as v.

Complexity Analysis:

Time complexity: $O(V^K)$
Space complexity :O(VE)

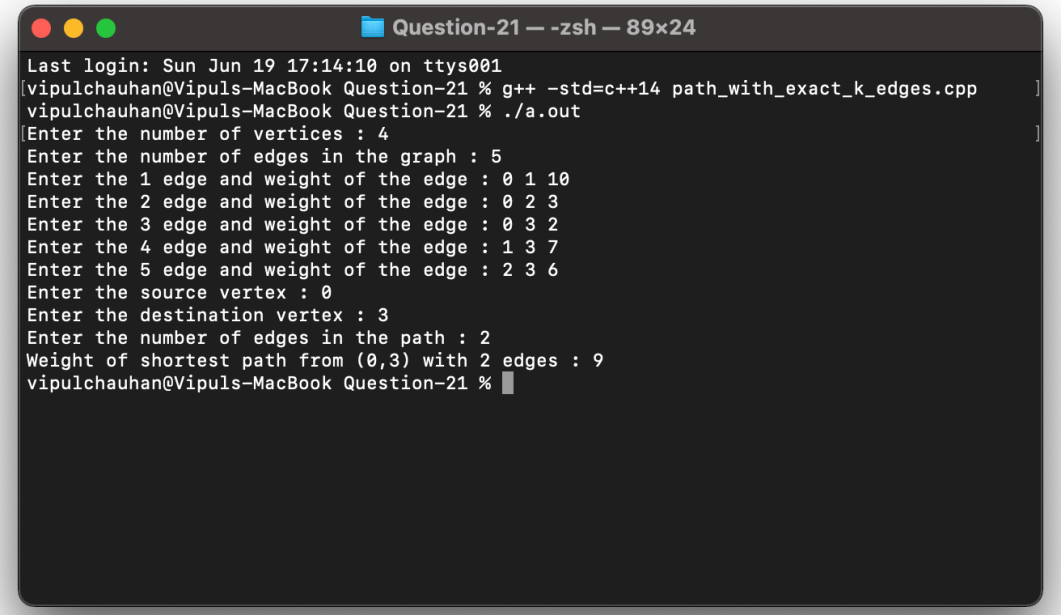
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <iostream>
#include <climits>
#include <vector>
using namespace std;
#define INF INT_MAX

int shortestPath(vector<vector<int> > graph, int s, int d, int k,int V)
{
    if(k==0 && s==d)
        return 0;
    if(k==1 && graph[s][d]!=INF)
        return graph[s][d];
    if(k<=0)
        return INF;

    int res=INF;
    for(int i=0;i<V;i++)
    {
         if (graph[s][i]!=INF && s!=i && d!=i)     {
                int rec_res=shortestPath(graph,i,d,k-1,V);
                if(rec_res!=INF)
                        res=min(res,graph[s][i]+rec_res);
         }
    }
    return res;
}
int main()  {
    int V,E,s,d,k;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    vector<vector<int> > graph(V,vector<int>(V,0));
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)    {
        int u,v,w;
        cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
        cin>>u>>v>>w;
```

```
        graph[u][v]=w;
    }
      cout<<"Enter the source vertex : ";
    cin>>s;
    cout<<"Enter the destination vertex : ";
    cin>>d;
    cout<<"Enter the number of edges in the path : ";
    cin>>k;
      int ans=shortestPath(graph,s,d,k,V);
    if(ans==0 || ans==INF)
    {
        cout<<"no path of length k is available"<<endl;
    }
    else
    {
        cout<<"Weight of shortest path from ("<<s<<","<<d<<") with "<<k<<"
edges : "<<ans<<endl;
    }
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 22

Problem Statement: Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing analgorithm which will find minimum cost required to connect these cities. (use Prim's algorithm).

Algorithm:

1) Create a set *mstSet* that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign the key value as 0 for the first vertex so that it is picked first.
3) While mstSet doesn't include all vertices
a) Pick a vertex *u* which is not there in *mstSet* and has a minimum key value.
b) Include *u* to mstSet.
c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if the weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as the weight of *u*-*v*

Complexity Analysis:

Time complexity:$O(V^2)$
Space complexity:O(V+E)

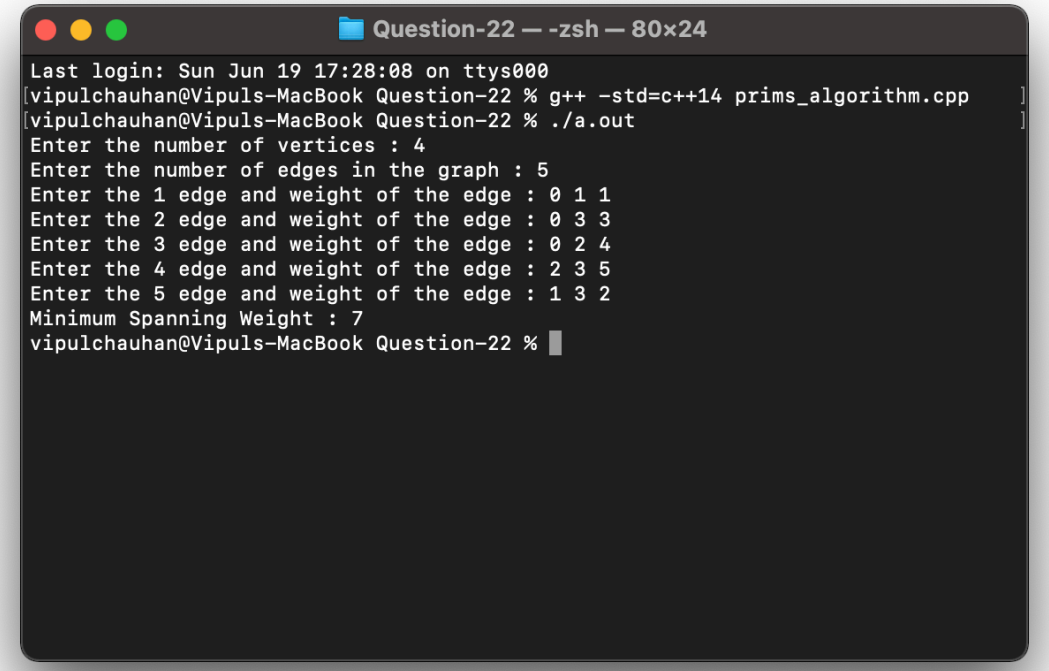
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/
#include <iostream>
#include <vector>
using namespace std;

int minnode(int n,int keyval[],bool mstset[])
{
    int mini=numeric_limits<int>::max();
    int mini_index;
    for (int i=0;i<n;i++)
    {
        if(mstset[i]==false && keyval[i]<mini)
       {
            mini=keyval[i],mini_index=i;
        }
    }
    return mini_index;
}
void findcost(int n,vector<vector<int> > city)
{
    int parent[n];
    int keyval[n];
    bool mstset[n];
    for (int i=0;i<n;i++)
    {
        keyval[i]=numeric_limits<int>::max();
        mstset[i]=false;
    }
    parent[0]=-1;
    keyval[0]=0;
    for(int i=0;i<n-1;i++)
    {
        int u=minnode(n,keyval,mstset);
        mstset[u]=true;
        for(int v=0;v<n;v++)
       {
            if(city[u][v] && mstset[v]==false && city[u][v]<keyval[v])
          {
                    keyval[v]=city[u][v];
                    parent[v]=u;
```

```
            }
        }
    }
    int cost=0;
    for(int i=1;i<n;i++)
        cost+=city[parent[i]][i];
    cout<<"Minimum Spanning Weight : "<<cost<<endl;
}
int main()
{
    int V,E;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    vector<vector<int> > g(V,vector<int> (V,0));
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        int u,v,w;
        cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
        cin>>u>>v>>w;
        g[u][v]=w;
        g[v][u]=w;
    }
    findcost(V,g);
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 23

Problem Statement: Implement the previous problem using Kruskal's algorithm.

Algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Complexity Analysis:

Time complexity:O(ElogV) or O(ElogE)
Space complexity:O(logE)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

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

class DSU
{
public:
    int* parent;
    int* rank;
    DSU(int n)
    {
        parent=new int[n];
        rank=new int[n];

        for(int i=0;i<n;i++)
      {
            parent[i]=-1;
            rank[i]=1;
        }
    }
    int find(int i)
    {
        if(parent[i]==-1)
            return i;
        return parent[i]=find(parent[i]);
    }
    void unite(int x,int y)
    {
        int s1=find(x);
        int s2=find(y);

        if(s1!=s2)
      {
            if(rank[s1]<rank[s2])
        {
                parent[s1]=s2;
                rank[s2]+=rank[s1];
            }
```

```
                else
        {
                        parent[s2]=s1;
                        rank[s1]+=rank[s2];
                }
            }
    }
};

class Graph
{
public:
    vector<vector<int> > edgelist;
    int V;
    Graph(int V)
   {
      this->V=V;
   }
    void addEdge(int x,int y,int w)
    {
            edgelist.push_back({w,x,y});
    }
    void kruskals_mst()
    {
            sort(edgelist.begin(),edgelist.end());
            DSU s(V);
            int ans = 0;
            cout<<"Following are the edges in the constructed MST"<< endl;
            for(auto edge : edgelist)
     {
                int w=edge[0];
                int x=edge[1];
                int y=edge[2];
                if(s.find(x)!= s.find(y))
        {
                        s.unite(x,y);
                        ans+=w;
                        cout<<x<<" -- "<<y<<" == "<<w<< endl;
                }
            }
            cout<<"Minimum Cost Spanning Tree: "<<ans<<endl;
    }
};
int main()
{
    int V,E,s;
```

```
    cout<<"Enter the number of vertices : ";
    cin>>V;
    Graph g(V);
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        int u,v,w;
        cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
        cin>>u>>v>>w;
        g.addEdge(u,v,w);
    }
    g.kruskals_mst();
     return 0;
}
```

OUTPUT

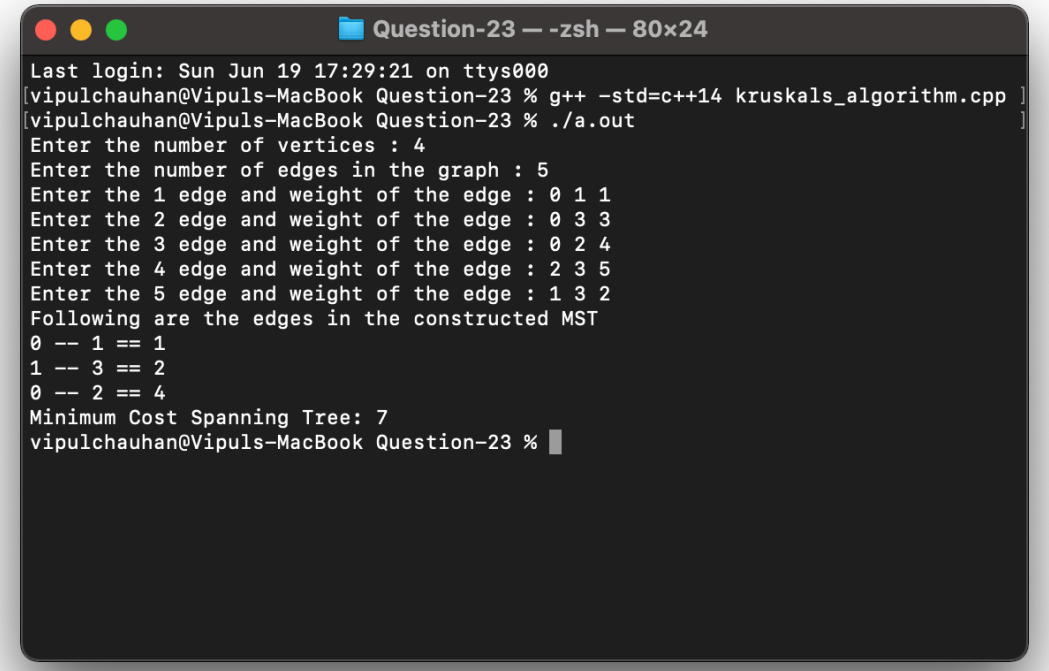
/*

Name : Vipul Chauhan

Univ. Roll No. : 2018855

Section : A

*/



Program 24

Problem Statement: Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

Algorithm:

- Initialize a visited array of Boolean datatype,to keep track of vertices visited so far. Initialize all the values with **false**.
- Initialize an array **weights[]**, representing the maximum weight to connect that vertex. Initialize all the values with some minimum value.
- Initialize an array **parent[]**, to keep track of the **maximum spanning tree**.
- Assign some large value, as the weight of the first vertex and parent as **-1**, so that it is picked first and has no parent.
- From all the unvisited vertices, pick a vertex **v** having a maximum weight and mark it as visited.
- Update the weights of all the unvisited adjacent vertices of **v**. To update the weights, iterate through all the unvisited neighbors of **v**. For every adjacent vertex **x**, if the weight of the edge between **v** and **x** is greater than the previous value of **v**, update the value of **v** with that weight.

Complexity Analysis:

Time complexity: $O(V^2)$
Space complexity: $O(V^2)$

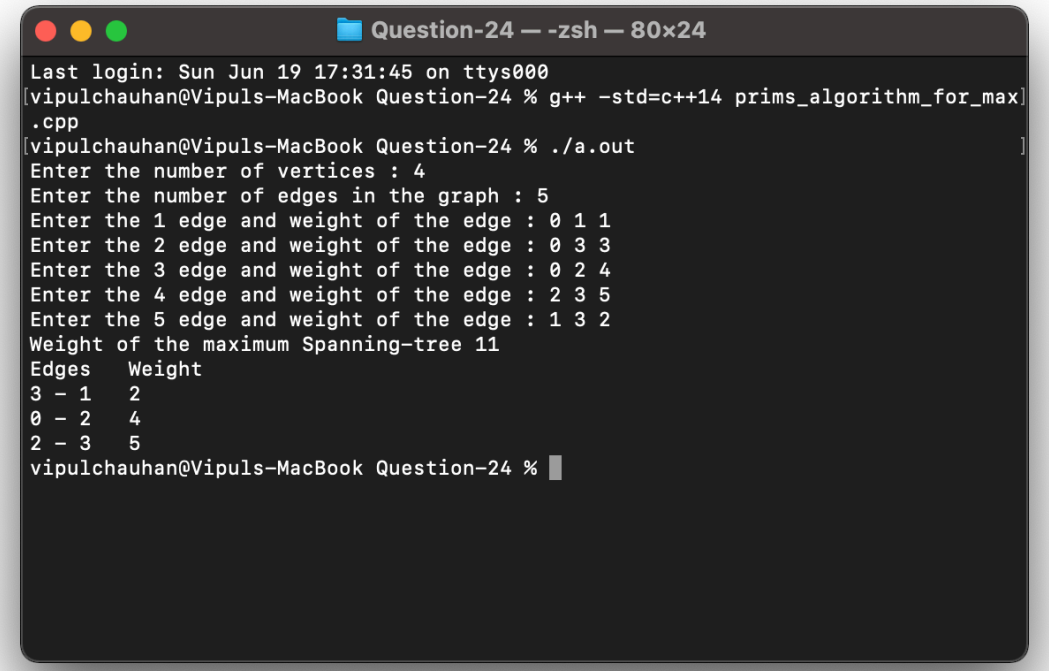
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <climits>
#include <vector>
using namespace std;
int V;
int findMaxVertex(bool visited[], int weights[])
{
    int index=-1;
    int maxW=INT_MIN;
    for (int i=0;i<V;i++)
   {
            if(visited[i]==false && weights[i]>maxW)
      {
                maxW=weights[i];
                index = i;
            }
    }
    return index;
}
void printMaximumSpanningTree(vector<vector<int> > graph,int parent[])
{
    int MST = 0;
    for(int i=1;i<V;i++)
   {
            MST+=graph[i][parent[i]];
    }
    cout<<"Weight of the maximum Spanning-tree "<<MST<<endl;
    cout << "Edges \tWeight\n";
    for(int i=1;i<V;i++)
   {
            cout<<parent[i]<<" - "<<i<<" \t"<< graph[i][parent[i]]<<endl;
    }
}
void maximumSpanningTree(vector<vector<int> > graph)
{
    bool visited[V];
    int weights[V];
    int parent[V];
```

```
    for (int i=0;i<V;i++)
   {
            visited[i]=false;
            weights[i]=INT_MIN;
    }
    weights[0]=INT_MAX;
    parent[0]=-1;
    for (int i=0;i<V-1;i++)
   {
            int maxVertexIndex=findMaxVertex(visited,weights);
            visited[maxVertexIndex]=true;
            for (int j=0;j<V;j++)
      {
                    if (graph[j][maxVertexIndex]!=0 && visited[j]==false)
        {
                            if (graph[j][maxVertexIndex]>weights[j])
            {
                                    weights[j]=graph[j][maxVertexIndex];
                                    parent[j]=maxVertexIndex;
                            }
                    }
            }
    }
    printMaximumSpanningTree(graph,parent);
}
int main()
{
     int E;
   cout<<"Enter the number of vertices : ";
   cin>>V;
   vector<vector<int> > g(V,vector<int> (V,0));
   cout<<"Enter the number of edges in the graph : ";
   cin>>E;
   for(int i=0;i<E;i++)
   {
      int u,v,w;
      cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
      cin>>u>>v>>w;
      g[u][v]=w;
      g[v][u]=w;
   }
   maximumSpanningTree(g);
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 25

Problem Statement: Given a graph, Design an algorithm and implement it using a program to implement FloydWarshall all pair shortest path algorithm.

Algorithm:

For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.
1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j].

Complexity Analysis:

Time complexity:$O(V^3)$
Space complexity:$O(V^2)$

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <iostream>
#include <climits>
#include <vector>
using namespace std;
#define INF INT_MAX

int V;
void printSolution(vector<vector<int> > dist)
{
     cout<<"The following matrix shows the shortest distances between
every pair of vertices"<<endl;
     for(int i=0;i<V;i++)
   {
            for(int j=0;j<V;j++)
      {
                  if(dist[i][j]==INF)
                        cout<<"INF"<<" ";
                  else if(i!=j)
                        cout<<dist[i][j]<<" ";
          else
             cout<<0<<" ";
             }
             cout<<endl;
     }
}
void floydWarshall(vector<vector<int> > graph)
{
     vector<vector<int> > dist(V,vector<int>(V,INF));
   int i,j,k;
     for (i = 0;i<V;i++)
            for (j=0;j<V;j++)
                  dist[i][j]=graph[i][j];
     for (k=0;k<V;k++)
   {
            for(i=0;i<V;i++)
      {
                  for(j=0;j<V;j++)
```

```
            {
                        if(dist[i][j]>(dist[i][k]+dist[k][j]) && (dist[k][j]!=INF &&
dist[i][k] != INF))
                                dist[i][j]=dist[i][k]+dist[k][j];
                    }
                }
        }
        printSolution(dist);
}
int main()
{
     int E;
    cout<<"Enter the number of vertices : ";
    cin>>V;
    vector<vector<int> > g(V,vector<int> (V,INF));
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        int u,v,w;
        cout<<"Enter the "<<i+1<<" edge and weight of the edge : ";
        cin>>u>>v>>w;
        g[u][v]=w;
        g[v][u]=w;
    }
      floydWarshall(g);
      return 0;
}
```

OUTPUT

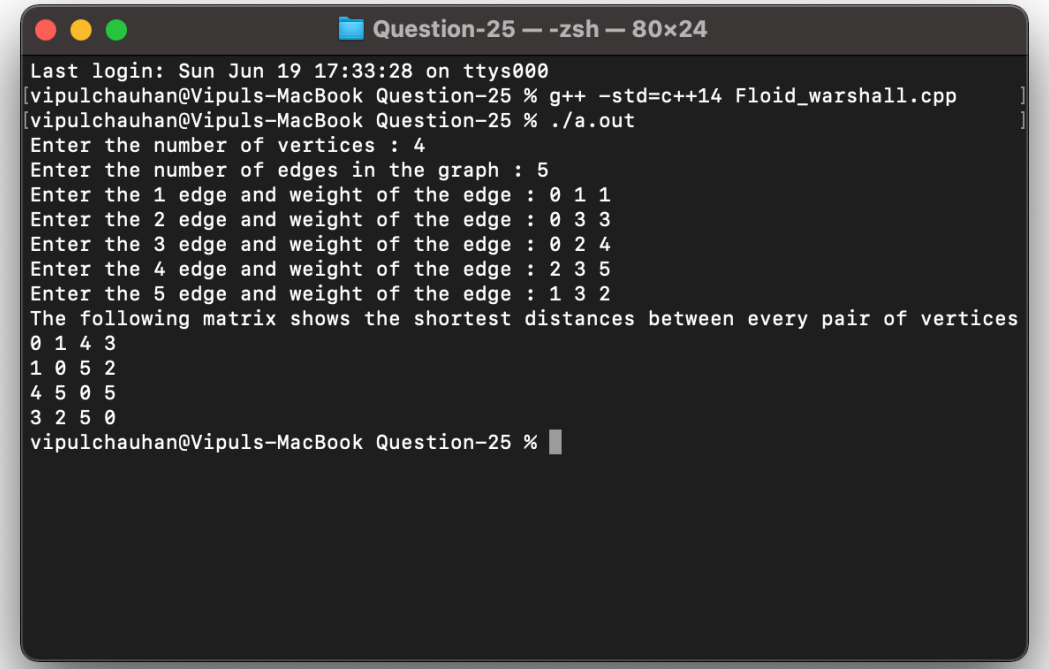
/*

Name : Vipul Chauhan

Univ. Roll No. : 2018855

Section : A

*/



Program 26

Problem Statement: Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction xi of item i, where $0 \leq xi \leq 1$.

Algorithm:

1. Calculate thr ratio(value/weight) for each item.
2. Sort all the items in decreasing order of the ratio.
3. Initialize res =0, curr_cap = given_cap.
4. Do the following for every item "i" in the sorted order:

```
while(i.weight){
    if(i.weight<=curr_cap)
    {
        curr_cap -= i.weight;
        res += i.value;
    }
    else
    {
        res += (i.value * (curr_cap/i.weight));
        return res;
    }
}
```

5. Return res.

Complexity Analysis:

Time complexity: O(nlogn)
Space complexity: O(1)

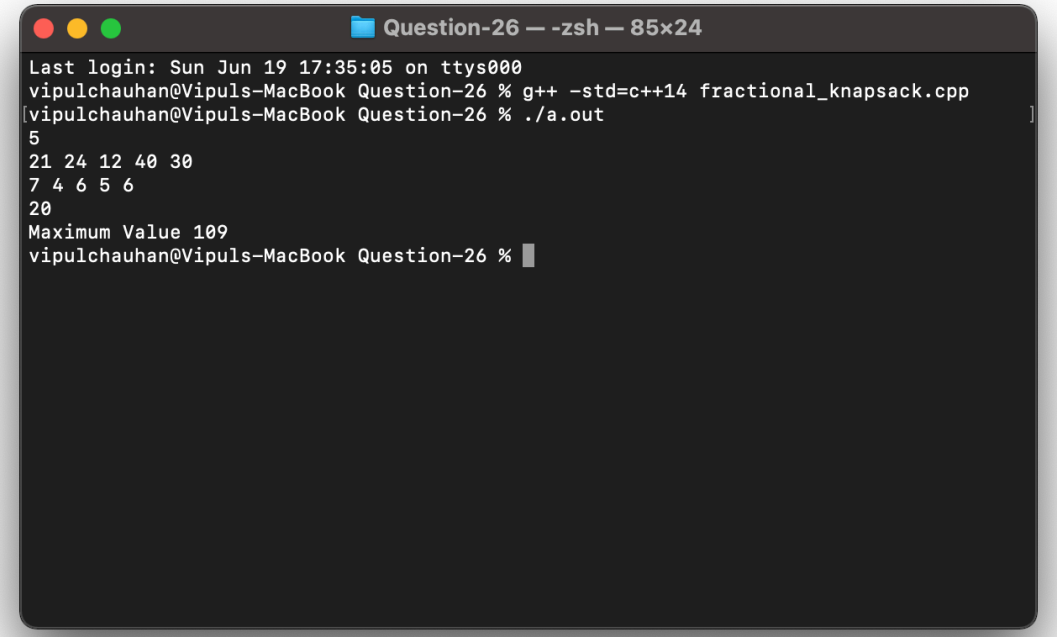
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool compare(const pair<int,double> a,const pair<int,double> b)
{
   return a.second>b.second;
}
int main()
{
   int N,W;
   cin>>N;
   vector<int> val(N);
   vector<int> wt(N);
   for(int i=0;i<N;i++)
   {
      cin>>val[i];
   }
   for(int i=0;i<N;i++)
   {
      cin>>wt[i];
   }
   cin>>W;
   double ans=0;
   vector<pair<int,double> > vp;
   for (int i=0;i<N;i++)
   {
      vp.push_back(make_pair(i,(double)val[i]/wt[i]));
   }
   sort(vp.begin(), vp.end(), compare);

   for (int i=0;i<N;i++)
   {
      if(W>=wt[vp[i].first])
      {
         ans+=val[vp[i].first];
         W-=wt[vp[i].first];
```

```
            continue;
        }
        ans+=W*vp[i].second;
        break;
    }
    cout<<"Maximum Value "<<ans << endl;
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 27

Problem Statement: Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach).

Algorithm:

Node represents a file with a given size also given nodes are greater than 2

1. Add all the nodes in a priority queue (Min Heap).{pq.poll = file size}
2. Initialize count = 0 // variable to store file computations.
3. Repeat while (size of priority Queue is greater than 1)
    0. int weight = pq.poll(); pq.pop;//pq denotes priority queue, remove 1st smallest and pop(remove) it out
    1. weight+=pq.poll() && pq.pop(); // add the second element and then pop(remove) it out
    2. count +=weight;
    3. pq.add(weight) // add this combined cost to priority queue;
4. count is the final answer

Complexity Analysis:

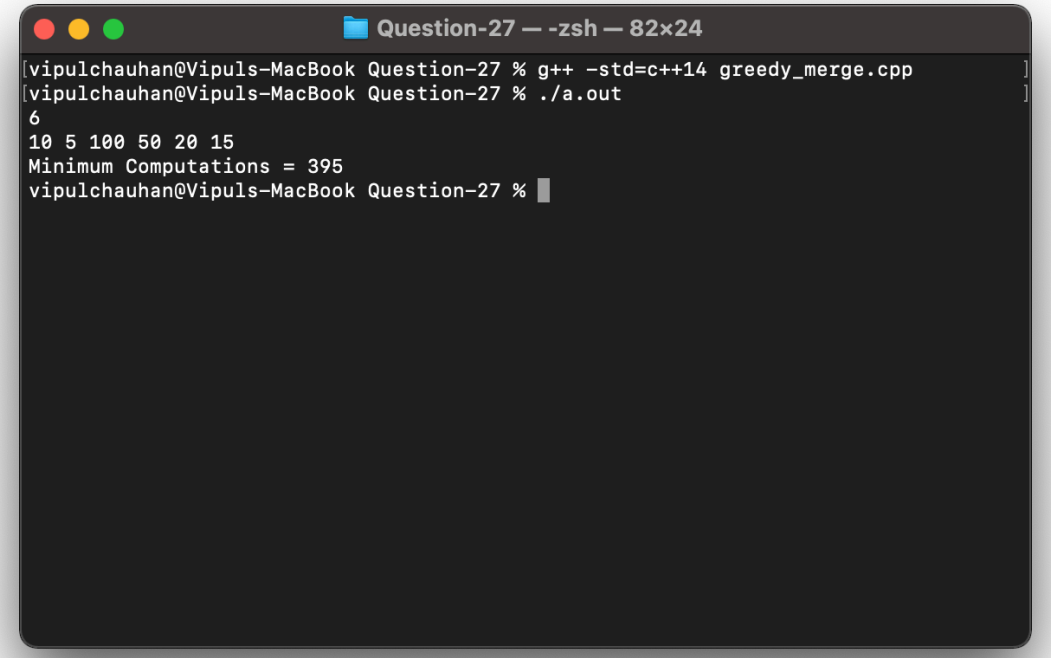
Time complexity: O(nlogn)
Space complexity: O(n)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <queue>
using namespace std;
int minComputation(int size, int files[])
{
     priority_queue<int,vector<int>,greater<int> > pq;
     for(int i=0;i<size;i++)
    {
              pq.push(files[i]);
    }
      int count = 0;
      while (pq.size()>1)
    {
              int first_smallest=pq.top();
              pq.pop();
              int second_smallest=pq.top();
              pq.pop();
              int temp=first_smallest+second_smallest;
              count+=temp;
              pq.push(temp);
     }
     return count;
}
int main()
{
     int N;
    cin>>N;
    int files[N];
    for(int i=0;i<N;i++)
    {
       cin>>files[i];
    }
     cout<<"Minimum Computations = "<<minComputation(N,files)<<endl;
     return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 28

Problem Statement: Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

Algorithm:

We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.
1) Sort the activities according to their finishing time
2) Select the first activity from the sorted array and print it.
3) Do the following for the remaining activities in the sorted array.

    a) If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it.

Complexity Analysis:

Time complexity: O(nlogn)
Space complexity: O(n)

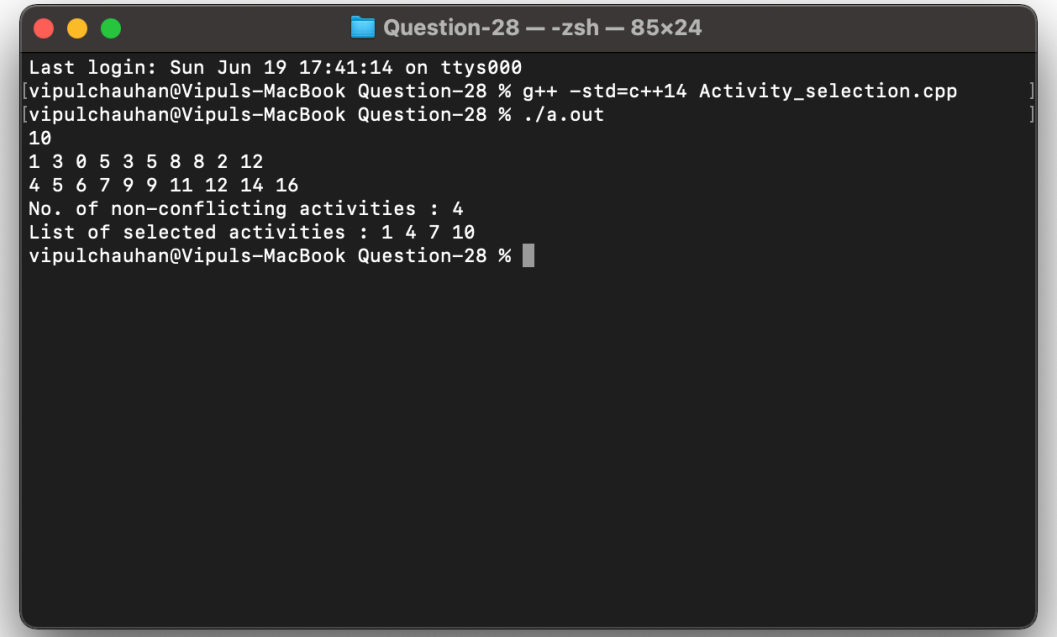
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

bool compare(const vector<int> &a, const vector<int> &b)
{
    return a[1]<b[1];
}

int main()
{
    int N;
    cin>>N;
    vector<int> start(N);
    vector<int> finish(N);
    vector<int> selected;
    for(int i=0;i<N;i++)
    {
        cin>>start[i];
    }
    for(int i=0;i<N;i++)
    {
        cin>>finish[i];
    }
    vector<vector<int> > vp;
    for (int i=0;i<N;i++)
        vp.push_back({start[i],finish[i]});

    sort(vp.begin(),vp.end(),compare);
    int take=1;
    int end=vp[0][1];
    selected.push_back(1);
    for(int i=1;i<N;i++)
    {
        if(vp[i][0]>=end)
        {
            take++;
```

```
            end=vp[i][1];
            selected.push_back(i+1);
        }
    }
    cout<<"No. of non-conflicting activities : "<<take<<endl;
    cout<<"List of selected activities : ";
    for(auto i:selected)
    {
        cout<<i<<" ";
    }
    cout<<endl;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 29

Problem Statement: Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

Algorithm:

1) Sort all jobs in decreasing order of profit.
2) Iterate on jobs in decreasing order of profit.For each job, do the following :

   a)Find a time slot i, such that slot is empty and i < deadline and i is greatest.Put the job in this slot and mark this slot filled.
   b)If no such i exists, then ignore the job.

Complexity Analysis:

Time complexity: O(nlogn)
Space complexity: O(n)

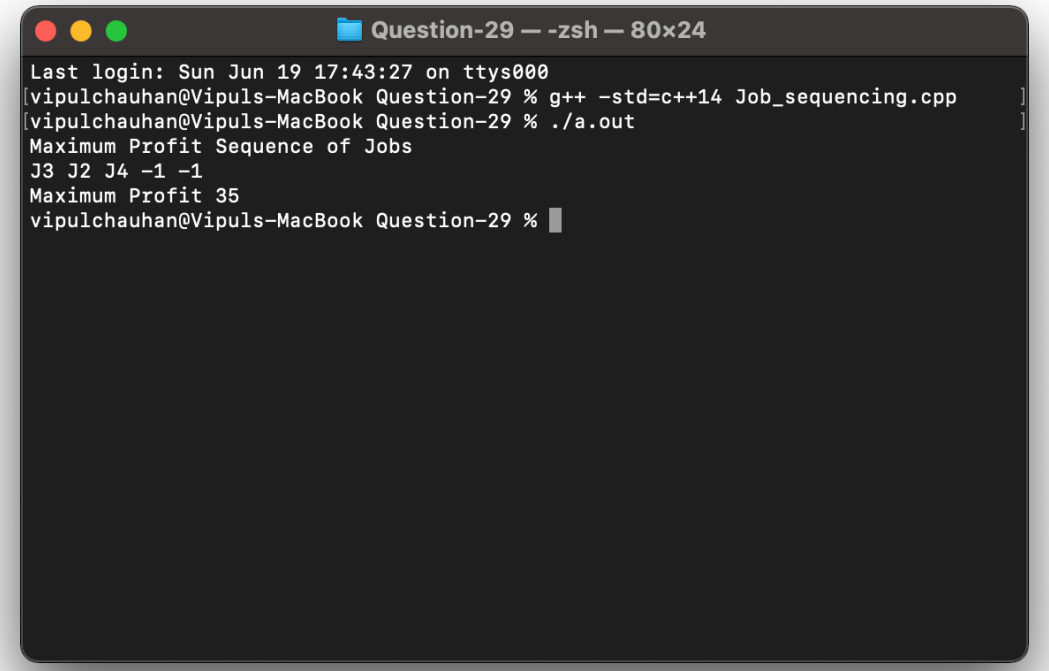
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool compare(const pair<string,pair<int, int> > a, const pair<string,pair<int,
int> > b)
{
    return a.second.second>b.second.second;
}

int main()
{
    vector<string> Jobs={"J1", "J2", "J3", "J4", "J5"};
    vector<int> Profit={20, 15, 10, 5, 1};
    vector<int> Deadline={2, 2, 1, 3, 5};

    int n=Jobs.size();
    int ind=0;

    vector<string> slot(*max_element(Deadline.begin(), Deadline.end()) + 1,
"-1");
    slot[0]="occupied";

    vector<pair<string,pair<int, int> > > vp;

    for (int i = 0; i < n; ++i)
    {
        vp.push_back({Jobs[i],{Deadline[i],Profit[i]}});
    }
    sort(vp.begin(),vp.end(),compare);
    for (int i=0;i<n;i++)
    {
        if (slot[Deadline[vp[i].second.first]] == "-1")
        {
            slot[Deadline[vp[i].second.first]] = Jobs[vp[i].second.first];
            ind += vp[i].second.second;
            continue;
```

```
        }
        int index = Deadline[vp[i].second.first];
        for (int j = index - 1; j > 0; --j)
        {
            if (slot[j]=="-1")
            {
                slot[j] = Jobs[vp[i].second.first];
                ind += vp[i].second.second;
                break;
            }
        }
    }

    cout << "Maximum Profit Sequence of Jobs \n";
    for (int i = 1; i < slot.size(); ++i)
        cout<<slot[i]<<" ";
    cout << endl;

    cout << "Maximum Profit " << ind << endl;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 30

Problem Statement: Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array.

Algorithm:

1. Create a hashmap to store a key-value pair, i.e. element-frequency pair.
2. Traverse the array from start to end.
3. For every element in the array, insert the element in the hashmap if the element does not exist as key, else fetch the value of the key (array[i]), and increase the value by 1
4. If the count is greater than half then print the majority element and break.
5. If no majority element is found print “No Majority element”

Complexity Analysis:

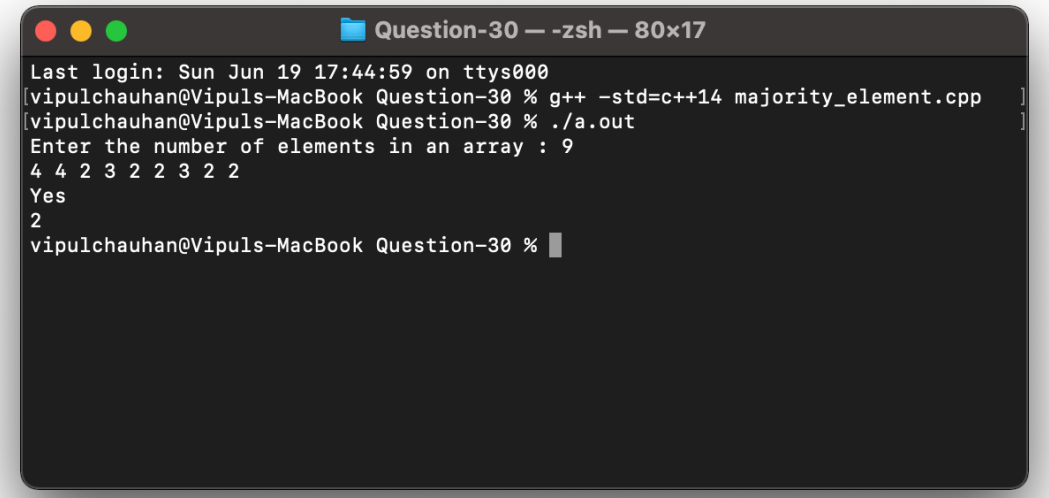
Time complexity: O(n).
Space complexity: O(n).

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <bits/stdc++.h>
using namespace std;
void findMajority(vector<int> arr, int size)  {
    unordered_map<int, int> m;
    for(int i = 0; i < size; i++)
        m[arr[i]]++;
    int count = 0;
    for(auto i : m)
    {
        if(i.second > size / 2)

        {
            count =1;
            cout << "Majority found :- " << i.first<<endl;
            break;
        }
    }
    if(count == 0)
        cout << "No Majority element" << endl;
}



int main()
{
    vector<int> arr;
    int n ;
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>arr[i];
      }
    findMajority(arr, n);
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 31

Problem Statement: Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

Algorithm:

- Calculate the cost for each placement and return the minimum value. In a chain of matrices of size n, we can place the first set of parenthesis in n-1 ways.
- When we place a set of parenthesis, we divide the problem into subproblems of smaller size. Therefore, the problem has optimal substructure property and can be easily solved using recursion.
- Minimum number of multiplication needed to multiply a chain of size n = Minimum of all n-1 placements.

Complexity Analysis:

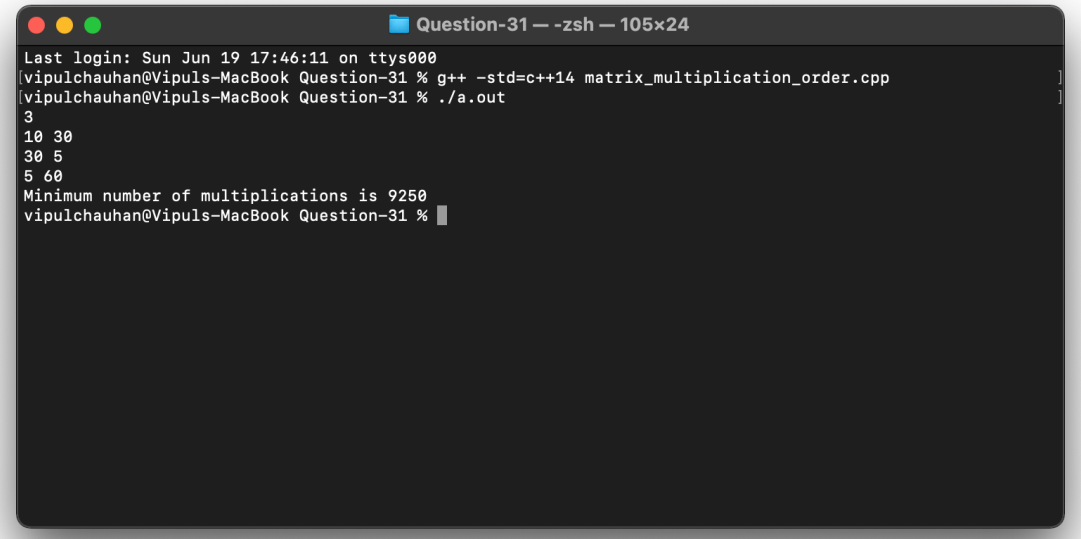
Time complexity: $O(n^3)$
Space complexity: $O(n^2)$

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <bits/stdc++.h>
using namespace std;
int dp[100][100];
int matrixChainMemoised(int* p, int i, int j)
{
    if (i == j)
        return 0;
    if (dp[i][j] != -1)
        return dp[i][j];

    dp[i][j] = INT_MAX;

   for (int k = i; k < j; k++)
    {
        dp[i][j] = min(dp[i][j], matrixChainMemoised(p, i, k)
                               + matrixChainMemoised(p, k + 1, j)
                               + p[i - 1] * p[k] * p[j]);
    }
    return dp[i][j];
}
int MatrixChainOrder(int* p, int n)
{
    int i = 1, j = n - 1;
    return matrixChainMemoised(p, i, j);
}
int main()
{
   int N;
   cin>>N;
   int arr[N*2];
   for(int i=0;i<N*2;i++)
   {
      cin>>arr[i];
   }
    cout<<"Minimum number of multiplications is
"<<MatrixChainOrder(arr,1,2*N-1)<<endl;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 32

Problem Statement: Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.

Algorithm:

- If we are at s[m-1], we can take as many instances of that coin (unbounded inclusion) i.e **count(S, m, n – S[m-1])** ; then we move to s[m-2]. After moving to s[m-2], we can't move back and can't make choices for s[m-1] i.e **count(S, m-1, n)**.
- Finally, as we have to find the total number of ways, so we will add these 2 possible choices, i.e **count(S, m, n – S[m-1]) + count(S, m-1, n) ;**

Complexity Analysis:

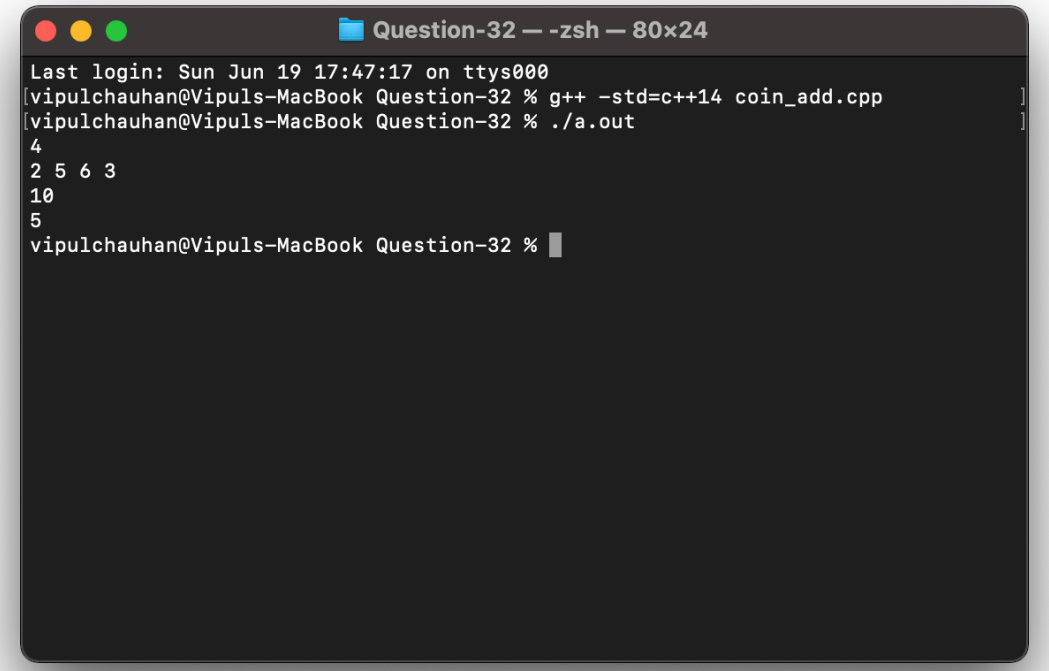
Time complexity: O(mn)
Space complexity: O(n)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/
#include<bits/stdc++.h>
using namespace std;

int count( int S[], int m, int n )
{
    int i, j, x, y;
    int table[n + 1][m];

    for (i = 0; i < m; i++)
        table[0][i] = 1;
    for (i = 1; i < n + 1; i++)
    {
        for (j = 0; j < m; j++)
        {
            x = (i-S[j] >= 0) ? table[i - S[j]][j] : 0;     // for including s[j]
            y = (j >= 1) ? table[i][j - 1] : 0;             // for excluding s[j]
            table[i][j] = x + y;                            // for total count
        }
    }
    return table[n][m - 1];
}


int main()
{
    int N,num;
   cin>>num;
   int arr[num];
   for(int i=0;i<num;i++)
   {
      cin>>arr[i];
   }
   cin>>N;
    cout<<count(arr,num,N)<<endl;
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 33

Problem Statement: Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.

Algorithm:

Let isSubsetSum(arr, n, sum/2) be the function that returns true if there is a subset of arr[0..n-1] with sum equal to sum/2

The isSubsetSum problem can be divided into two subproblems :

a) isSubsetSum() without considering last element
(reducing n to n-1)

b) isSubsetSum considering the last element
(reducing sum/2 by arr[n-1] and n to n-1)

If any of the above subproblems return true, then return true.

isSubsetSum (arr, n, sum/2) = isSubsetSum (arr, n-1, sum/2) ||
isSubsetSum (arr, n-1, sum/2 - arr[n-1])

Complexity Analysis:

Time complexity: O(sum*n)
Space complexity: O(sum*n)

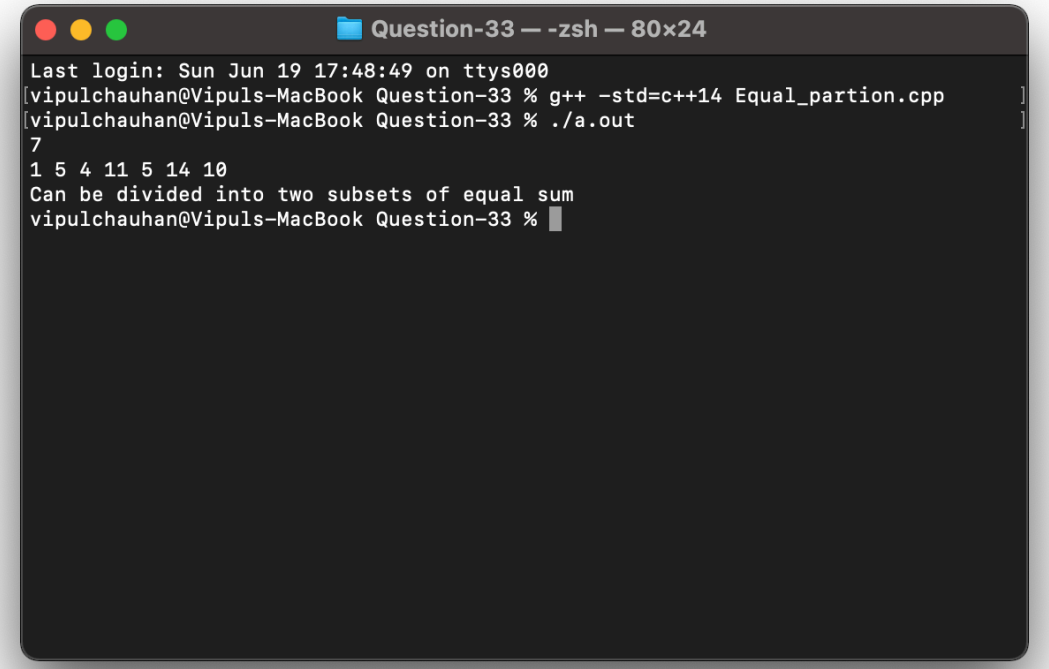
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <bits/stdc++.h>
using namespace std;
bool isSubsetSum(int arr[], int n, int sum,vector<vector<int> >& dp)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (dp[n][sum] != -1) {
        return dp[n][sum];
    }
    if (arr[n - 1] > sum)
        return isSubsetSum(arr, n - 1, sum, dp);


    return dp[n][sum]SubsetSum(arr, n - 1, sum, dp)   ||  isSubsetSum(arr, n
- 1,              sum - arr[n - 1],dp);
}

bool findPartiion(int arr[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += arr[i];
    if (sum % 2 != 0)
        return false;
    vector<vector<int> > dp(n + 1,vector<int>(sum + 1, -1));
    return isSubsetSum(arr, n, sum / 2, dp);
}
int main()
{
   int n;
   cin>>n;
   int arr[n];
   for(int i=0;i<n;i++)
   {
      cin>>arr[i];
   }
```

```
    if(Equal_Sum_Partition(arr,n)==true)
        cout<<"Can be divided into two subsets of equal sum"<<endl;
    else
        cout<<"Can not be divided into two subsets of equal sum"<<endl;
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 34

Problem Statement: Given two sequences, Design an algorithm and implement it using a program to find the length of the longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

Algorithm:

Let the input sequences be X[0..m-1] and Y[0..n-1] of lengths m and n respectively.

And let L(X[0..m-1], Y[0..n-1]) be the length of LCS of the two sequences X and Y.

Following is the recursive definition of L(X[0..m-1], Y[0..n-1]).

- If last characters of both sequences match (or X[m-1] == Y[n-1]) then
  L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])
- If last characters of both sequences do not match (or X[m-1] != Y[n-1]) then
  L(X[0..m-1], Y[0..n-1]) = MAX (L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2]))

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(m*n)

Space complexity :

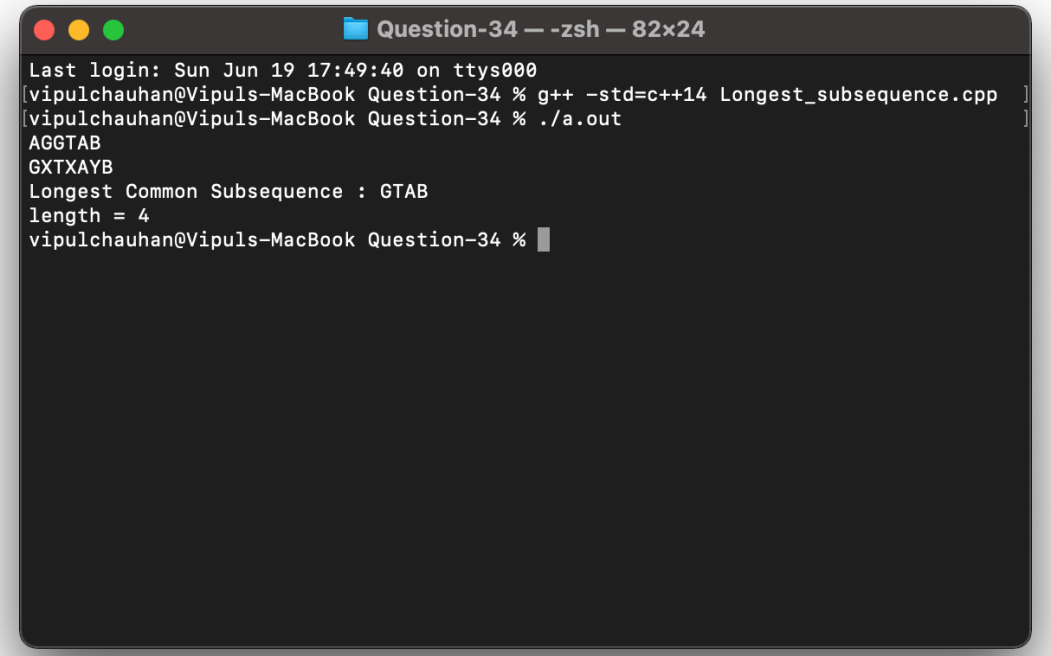
Worst Case = Best Case = Average case = O(2n)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

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

void solve(string text1,string text2){
   int n = text1.size();
       int m = text2.size();

       int dp[n+1][m+1];

       for(int i=0;i<=n;i++){
          for(int j = 0;j<=m;j++){
             if(i == 0 || j == 0) dp[i][j] = 0;
          }
       }

       for(int i =1;i<=n;i++){
          for(int j =1; j<=m; j++){
             if(text1[i-1] == text2[j-1]) dp[i][j] = 1+dp[i-1][j-1];
             else {
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
             }
          }
       }
       cout<<"Longest common subsequence = "<<dp[n][m];
}
int main()
{
 string s1,s2;
 cout<<"Subsequence 1 : ";    cin>>s1;
 cout<<"Subsequence 2 : ";    cin>>s2;

 solve(s1,s2);
 return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 35

Problem Statement: Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

Algorithm:

1. In a DP[][] table let's consider all the possible weights from '1' to 'W' as the columns and weights that can be kept as the rows.
2. The state DP[i][j] will denote maximum value of 'j-weight' considering all values from '1 to i^{th}'. So if we consider 'wi' (weight in 'i^{th}' row) we can fill it in all columns which have 'weight values > wi'. Now two possibilities can take place:
   - Fill 'wi' in the given column.
   - Do not fill 'wi' in the given column.
3. Now we have to take a maximum of these two possibilities, formally if we do not fill 'ith' weight in 'jth' column then DP[i][j] state will be same as DP[i-1][j] but if we fill the weight, DP[i][j] will be equal to the value of 'wi'+ value of the column weighing 'j-wi' in the previous row. So we take the maximum of these two possibilities to fill the current state.

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(N*W)

Space complexity :

Worst Case = Best Case = Average case = O(N*W)

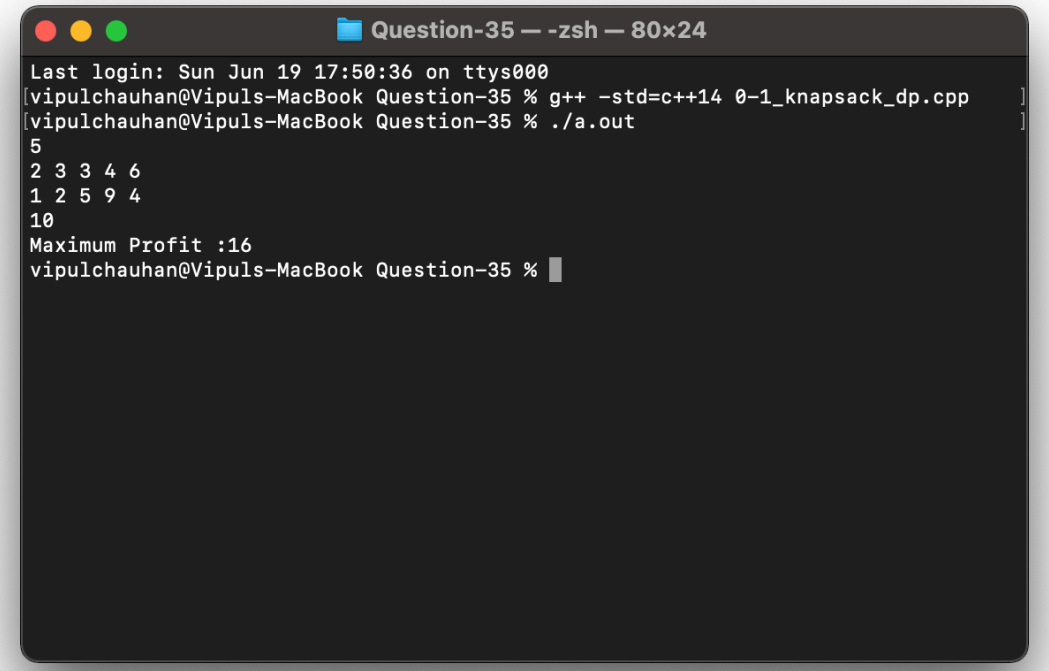
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <bits/stdc++.h>

using namespace std;

void solve(int W, vector<int>  wt, vector<int>  val, int n){
  int i, w;
      vector<vector<int>> K(n + 1, vector<int>(W + 1));
    for(i = 0; i <= n; i++)
    {
        for(w = 0; w <= W; w++)
        {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] +
                              K[i - 1][w - wt[i - 1]],
                              K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }
    cout<<"Maximum Profit :"<< K[n][W];
}


int main()
{
 int n;
 cin>>n;
 vector<int> wt;
 vector<int>  val;
 int W;
 for(int i=0;i<n;i++){
    int x;
    cin>>x;
    wt.push_back(x);
 }
```

```
 for(int i=0;i<n;i++){
    int x;
    cin>>x;
    val. push_back (x);
 }
 cin>>W;
 solve(W,wt,val,n);
 return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 36

Problem Statement: Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.

Algorithm:

1. Sort the given string in non-decreasing order and print it.
   The first permutation is always the string sorted in non-decreasing order.
2. Start generating next higher permutation.
   Do it until next higher permutation is not possible.
   If we reach a permutation where all characters are sorted in non-increasing order, then that permutation is the last permutation

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(n*n!)

Space complexity :

Worst Case = Best Case = Average case = O(n)

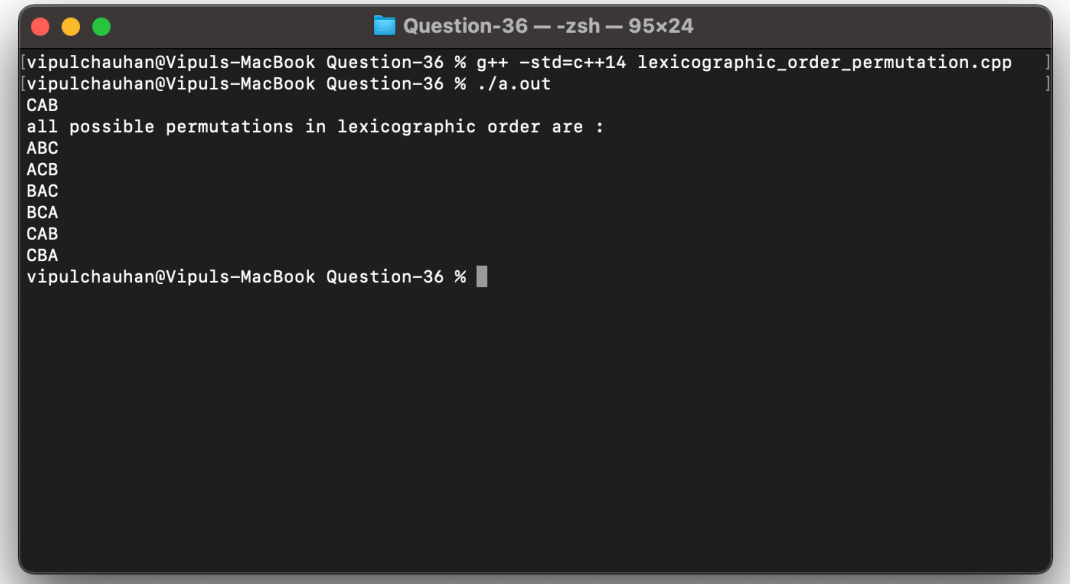
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <iostream>
#include <cstdlib>
#include <algorithm>
using namespace std;
#define MAX 20;
int compare(const void *a,const void *b)
{
   return (*(char *)a-*(char *)b);
}
void swap(char* a,char* b)
{
   char t = *a;
   *a = *b;
   *b = t;
}
int findCeil(char str[],char first,int l,int h)
{
   int ceilIndex=l;
   for(int i=l+1;i<=h;i++)
      if(str[i]>first && str[i]<str[ceilIndex])
         ceilIndex=i;

   return ceilIndex;
}
void reverse(char str[],int l,int h)
{
     while(l<h)
     {
            swap(&str[l],&str[h]);
            l++;
            h--;
     }
}
```

```
void sortedPermutations(char str[])
{
     int size=strlen(str);
     qsort(str,size,sizeof(str[0]),compare);
     bool isFinished=false;
     while(!isFinished)
     {
              cout<<str<<endl;
              int i;
              for(i=size-2;i>=0;i--)
                 if(str[i]<str[i+1])
                          break;

              if(i==-1)
                       isFinished=true;
              else
              {
                       int ceilIndex=findCeil(str,str[i],i+1,size-1);
                       swap(&str[i],&str[ceilIndex]);
                       reverse(str,i+1,size-1);
              }
     }
}
int main()
{
   char str[20];
   cin>>str;
   cout<<"all possible permutations in lexicographic order are :"<<endl;
   sortedPermutations(str);
   return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 37

Problem Statement: Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = O(n))

Algorithm:

1. Covert the input string into the same case.
2. Sort the string.
3. Create a loop and in it
   For (char i : s)
   {
   ans[i]++;
   }
   To count the frequencies
4. Print all the counted frequencies.

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(n)

Space complexity :

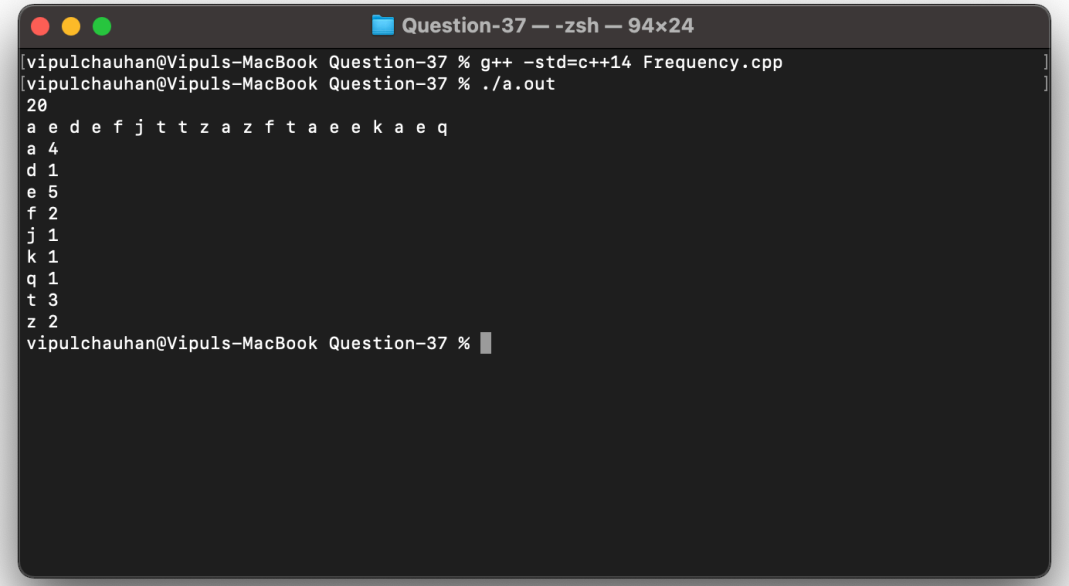
Worst Case = Best Case = Average case = O(n)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <vector>
#include <map>
using namespace std;

int main()
{
    int n;
    cin>>n;
    map<char,int> m;
    vector<char> s(n);
    for(int i=0;i<n;i++)
    {
        cin>>s[i];
        m[s[i]]++;
    }
    for(auto i:m)
    {
        cout<<i.first<<" "<<i.second<<endl;
    }
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 38

Problem Statement: Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicate within a small window of size k < n.

Algorithm:

1) Create an empty hashtable.
2) Traverse all elements from left from right. Let the current element be 'arr[i]'

a) If current element 'arr[i]' is present in hashtable, then return true.
b) Else add arr[i] to hash and remove arr[i-k] from hash if i is greater than or equal to k.

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(n)

Space complexity :

Worst Case = Best Case = Average case = O(n)

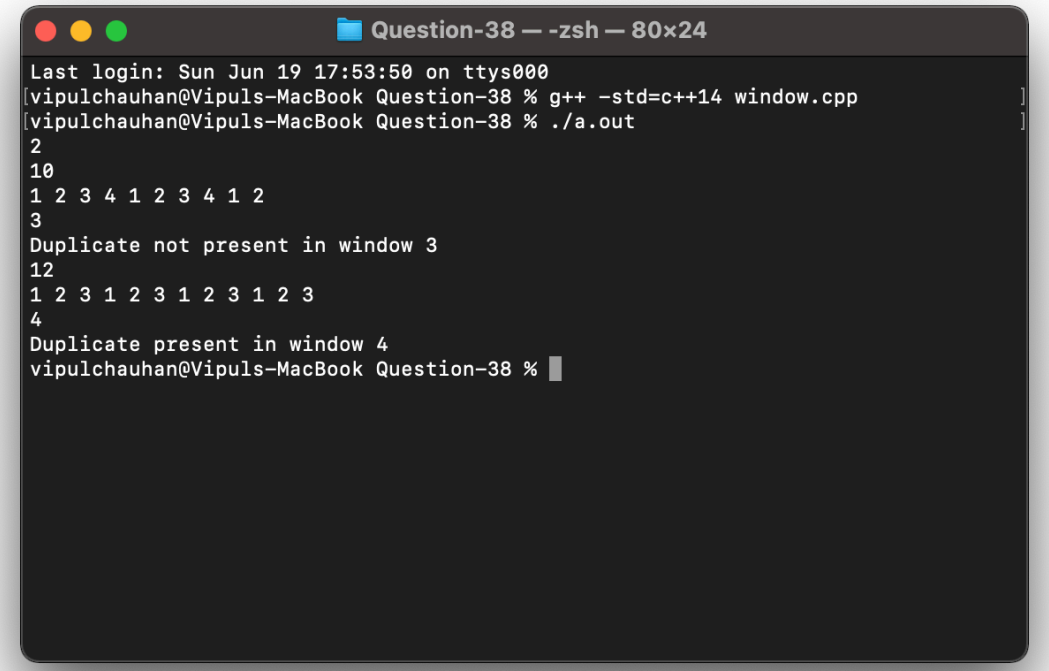
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <vector>
#include <set>
using namespace std;

bool check(vector<int> a,int n,int k)
{
    set<int> ans;
    for(int i=0;i<n;i++)
    {
        if(ans.find(a[i])!=ans.end())
        {
            return true;
        }
        ans.insert(a[i]);
        if(i>=k)
            ans.erase(a[i-k]);
    }
    return false;
}



int main()
{
    int T;
    cin>>T;
    while(T--)
    {
        int n,k;
        cin>>n;
```

```
        vector<int> arr;
        for(int i=0;i<n;i++)
        {
            int x;
            cin>>x;
            arr.push_back(x);
        }
        cin>>k;
        if(check(arr,n,k))
            cout<<"Duplicate present in window "<<k<<endl;
        else
            cout<<"Duplicate not present in window "<<k<<endl;
    }
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 39

Problem Statement: Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a*b = c*d, where a, b, c and d are distinct elements of array.

Algorithm:

```
Loop i = 0 to n-1 :
   Loop j = i + 1 to n-1 :
      calculate sum
      If in hash table any index already exist
         Then print  (i, j) and previous pair
         from hash table
      Else update hash table
   EndLoop;
EndLoop;
```

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = $O(n^2 \text{ Log } n)$

Space complexity :

Worst Case = Best Case = Average case = $O(n)$

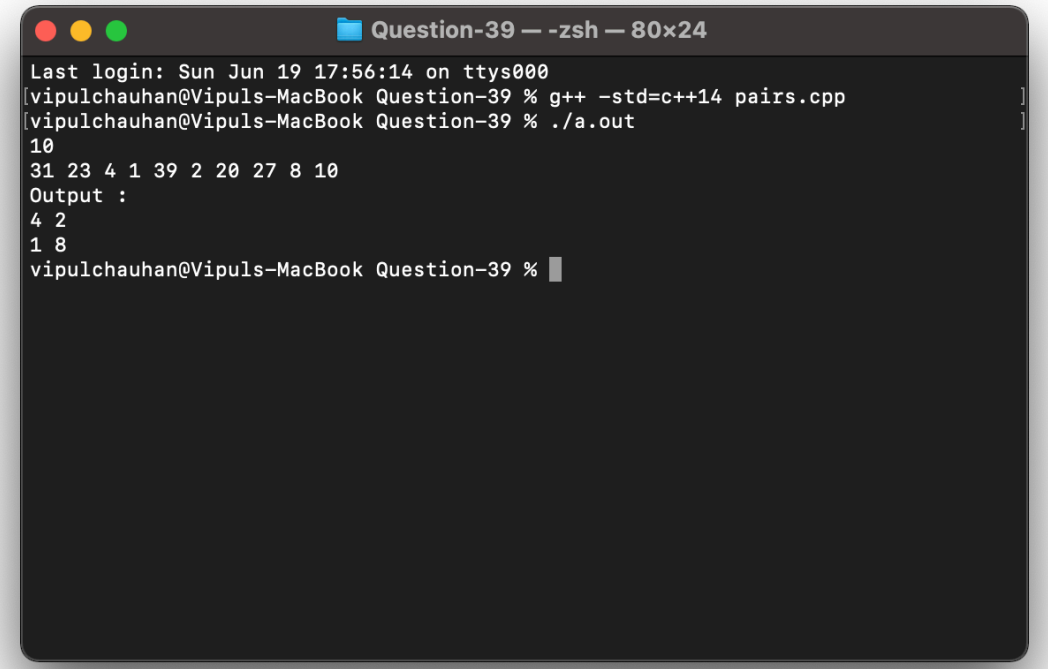
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <vector>
#include <map>

using namespace std;

bool algo(int n,vector<int> arr)
{
   map<int,pair<int,int> > Hash;
   for(int i=0;i<n;i++)
   {
      for(int j=i+1;j<n;j++)
      {
         int mul=arr[i]*arr[j];
         if(Hash.find(mul)==Hash.end())
            Hash[mul]=make_pair(i,j);
         else
         {
            pair<int,int> pp=Hash[mul];
            cout<<arr[pp.first]<<" "<<arr[pp.second]<<endl<<arr[i]<<"
"<<arr[j]<<endl;
            return true;
         }
      }
   }
   cout<<"No pairs found";
   return false;

}

int main()
{
   int n;
   cin>>n;
```

```
    vector<int> arr;
    for(int i=0;i<n;i++)
    {
        int x;
        cin>>x;
        arr.push_back(x);
    }
    cout<<"Output : "<<endl;
    algo(n,arr);
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 40

Problem Statement: Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,. is sequence of ugly numbers.

Algorithm:

1.Loop for all positive integers till the ugly number count is smaller than n.

2.If an integer is ugly than increment ugly number count.

3.To check if a number is ugly,divide the number by gretest divisible power of 2,3 and 5,if th number becomes 1 than it is an ugly number otherwise not.

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(n)

Space complexity :

Worst Case = Best Case = Average case = O(n)

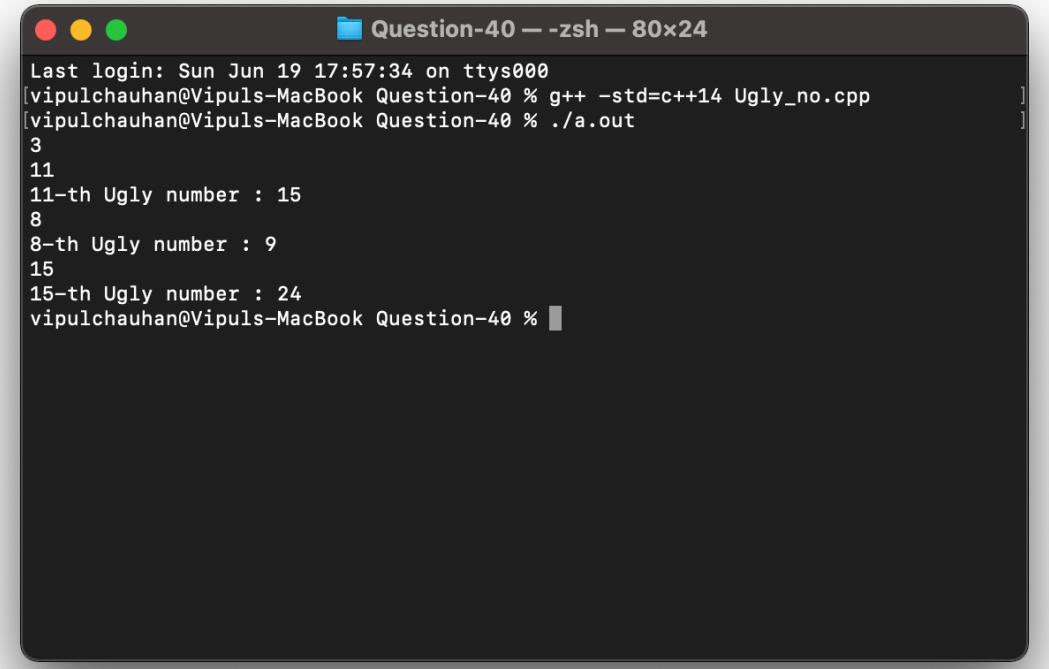
Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/


#include <iostream>
using namespace std;

int maxDivide(int a,int b)
{
        while(a%b==0)
                a=a/b;
        return a;
}
int isUgly(int no)
{
        no=maxDivide(no,2);
        no=maxDivide(no,3);
        no=maxDivide(no,5);

        return (no==1)?1:0;
}
int getNthUglyNo(int n)
{
        int i=1;
        int count=1;
        while (n > count)
        {
                i++;
                if(isUgly(i))
                        count++;
        }
        return i;
}
int main()
{
   int T,N;
   cin>>T;
```

```
    while(T--)
    {
        cin>>N;
        unsigned long long int ans=getNthUglyNo(N);
             cout<<N<<"-th Ugly number : "<<ans<<endl;
    }
         return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/



Program 41

Problem Statement: Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.

Algorithm:

1. Do DFS traversal of the given graph. While doing traversal keep track of last finished vertex 'v'. This step takes O(V+E) time.
2. If there exist mother vertex (or vertices), then v must be one (or one of them). Check if v is a mother vertex by doing DFS/BFS from v. This step also takes O(V+E) time.

Complexity Analysis:

Time complexity :

Worst Case = Best Case = Average case = O(V+E)

Space complexity :

Worst Case = Best Case = Average case = O(V)

Source Code:

```
/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/

#include <iostream>
#include <vector>
#include <list>
using namespace std;

class Graph
{
    public:
        int V;
        list<int> *adj;
        Graph(int V)
        {
            this->V=V;
            adj=new list<int>[V];
        }
        void DFSUtil(int v, vector<bool> &visited)
        {
            visited[v]=true;
            list<int>::iterator i;
            for(i=adj[v].begin();i!=adj[v].end();i++)
                if(!visited[*i])
                    DFSUtil(*i,visited);
        }
        void addEdge(int v, int w)
        {
            adj[v].push_back(w);
        }
        int findMother()
        {
            vector <bool> visited(V,false);
            int v=0;
            for(int i=0;i<V;i++)
            {
                if(visited[i]==false)
```

```
            {
                DFSUtil(i,visited);
                v=i;
            }
        }
        fill(visited.begin(),visited.end(),false);
        DFSUtil(v,visited);
        for(int i=0;i<V;i++)
        if(visited[i]==false)
            return -1;

        return v;
    }
};
int main()
{
    int V,E,u,v;
    cout<<"Enter the number of vertices in the graph : ";
    cin>>V;
    Graph g(V);
    cout<<"Enter the number of edges in the graph : ";
    cin>>E;
    for(int i=0;i<E;i++)
    {
        cout<<"Enter the edge "<<i+1<<" : ";
        cin>>u>>v;
        g.addEdge(u,v);
    }
    cout<<"A mother vertex is "<<g.findMother()<<endl;
    return 0;
}
```

OUTPUT

/*
Name : Vipul Chauhan
Univ. Roll No. : 2018855
Section : A
*/