Question 32:

Given 3 strings of all having length < 100, write a program to find the longest common subsequence in all three given sequences.

Time Complexity:

$T(n) = O(n^3)$

Code:

```
/* 3 strings Longest Common Sequence Dynamic Programming Approach */
#include <bits/stdc++.h>
using namespace std;

int max3(int a, int b, int c)
{
    if(a > b)
    {
        if(a > c)
            return a;
        return c;
    }
    else
    {
        if(b > c)
            return b;
        return c;
    }
}
void lcs(char *A, char *B, char* C, int m, int n, int p)
{
    int len[m+1][n+1][p+1];
    for(int i = 0;i<m+1;i++)
        for(int j = 0;j<n+1;j++)
            for(int k = 0;k<p+1;k++)
                len[i][j][k]=0;

    for(int i = 0;i<= m;i++)
    {
        for(int j = 0;j<=n; j++)
        {
            for(int k = 0;k<=p;k++)
            {
                //if both are 0
                if(i == 0 || j == 0 || p == 0)
                    len[i][j][k] = 0;
                //if last characters match
                else if(A[i-1] == B[j-1] && B[j-1] == C[k-1])
                    len[i][j][k] = 1 + len[i-1][j-1][k-1];
                //if last characters do not match
                else
                    len[i][j][k] = max3(len[i-1][j][k], len[i][j-1][k], len[i][j][k-
1]);
            }

        }
    }
    cout<<"Length is : "<<len[m][n][p]<<endl;
    int index = len[m][n][p];
    char s[index+1];
    s[index] = '\0';
    int i = m, j = n, k = p;
    while(i > 0 && j > 0 && k > 0)
    {
        //end char match
```

```
        if(A[i-1] == B[j-1] && B[j-1] == C[k-1])
        {
            s[index-1] = A[i-1];
            i--;
            j--;
            k--;
            index--;
        }
        //end char do not match
        else if(A[i-1] > B[j-1])
        {
            if(A[i-1] > C[k-1])
                i--;
            else
                k--;
        }
        else
        {
            if(B[j-1] > C[k-1])
                j--;
            else
                k--;
        }

    }
    cout<<s<<endl;;
}

int main()
{
    char A[] = "ABCDEFG";
    char B[] = "AAADEF";
    char C[] = "KKKADEF";

    int m = strlen(A);
    int n = strlen(B);
    int p = strlen(C);
    lcs( A, B, C, m, n, p );
    return 0;
}
```

Question 33:

Write a program to find the contiguous subsequence of maximum sum (a subsequence of length zero has sum zero). A contiguous subsequence of a list S is a subsequence made up of consecutive elements of S. For instance, if S is 5; 15;-30; 10;-5; 40; 10; then 15;-30; 10 is a contiguous subsequence but 5; 15; 40 is not. For the preceding example, the answer would be 10;-5; 40; 10, with a sum of 55. Give a linear-time algorithm for the following task:

Time Complexity:

T(n) = O(n)

CODE:

```
//Larest Contiguous Subarray Sum in Linear Time COmplexity.
#include <bits/stdc++.h>
using namespace std;

int kadane(int a[], int n)
{
    int max_ending_here = 0;
    int max_so_far = 0;

    for(int i = 0;i<n;i++)
    {
        max_ending_here = max_ending_here + a[i];
        if(max_ending_here < 0)
            max_ending_here = 0;
        if(max_so_far < max_ending_here)
            max_so_far = max_ending_here;
    }
    return max_so_far;
}
int main()
{
    int n;
    cin>>n;
    int a[n];
    for(int i = 0;i<n;i++)
        cin>>a[i];

    cout<<kadane(a,n)<<endl;
    return 0;
}
```

Question 34:

A subsequence is palindromic if it is the same whether read left to right or right to left. For instance, the sequence A;C; G; T; G; T;C; A; A; A; A; T;C;G has many palindromic subsequences, including A;C; G;C;A and A; A; A;A (on the other hand, the subsequence A;C; T is not palindromic). Devise an algorithm that takes a sequence x[1 : : : n] and returns the (length of the) longest palindromic subsequence. Its running time should be O(n2).

Time Complexity:

$T(n) = O(2^n)$

CODE:

```
/* Longest Palindromic Subsequence Recursive Solution */
#include <bits/stdc++.h>
using namespace std;

int lps(char *s, int i, int j)
{
    //Case 1
    if(i == j)
        return 1;
    //Case 2
    if(s[i] == s[j] && j == i+1)
        return 2;
    //Case 3
    if(s[i] == s[j])
        return lps(s, i+1,j-1) + 2;
    //Case 4
    return max(lps(s, i+1,j), lps(s, i,j-1));
}
int main()
{
    char s[]="abcdgcdeeab";
    int n = strlen(s);
    cout<<lps(s,0,n-1)<<endl;
    return 0;
}
```

Question 35:

A list of n positive integers a1; a2; : : : ; an; and a positive integer t is given. Write a program to find subset of the ai's add up to t? (You can use each ai at most once.).

Time Complexity:

This is an NP Complete problem. The complexity of above problem is exponential.

CODE:

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

bool subsetsum(int set[], int n, int sum)
{
    if(sum == 0)
        return true;
    //n equal 0 and sum > 0
    if(n == 0 && sum > 0)
        return false;
    //last element greater than sum
    if(set[n-1] > sum)
        return subsetsum(set, n-1, sum);

    /*  check for two conditions
        includeing the last eleent
        excluding tyhe least element
    */
    return (subsetsum(set, n-1, sum) || subsetsum(set, n-1, sum - set[n-1]));
}

int main()
{
    int n,s;
    cin>>n;
    int a[n];
    for(int i = 0;i<n;i++)
        cin>>a[i];
    cout<<"Enter a sum to find\n";
    cin>>s;
    cout<<subsetsum(a,n,s)<<endl;
    return 0;

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

bool subsetsum(int set[], int n, int sum)
{
    if(sum == 0)
        return true;
    //n equal 0 and sum > 0
    if(n == 0 && sum > 0)
        return false;
    //last element greater than sum
    if(set[n-1] > sum)
        return subsetsum(set, n-1, sum);

    /*check for two conditions
        includeing the last eleent
        excluding tyhe least element
    */
```

```
    return (subsetsum(set, n-1, sum) || subsetsum(set, n-1, sum - set[n-1]));
}

int main()
{
    int n,s;
    cin>>n;
    int a[n];
    for(int i = 0;i<n;i++)
        cin>>a[i];
    cout<<"Enter a sum to find\n";
    cin>>s;
    cout<<subsetsum(a,n,s)<<endl;
    return 0;

}
```

Question 36:

A vertex cover of a graph G = (V;E) is a subset of vertices S ⊆ V that includes at least one endpoint of every edge in E. Give & write a linear-time algorithm for finding the size of the smallest vertex cover of T. For instance, in the following tree, possible vertex covers include {A;B;C;D;E; F;G} and {A;C;D; F} but not {C;E;F}. The smallest vertex cover has size 3: {B;E;G}.

Time Complexity:

The problem of finding a vertex cover for a graph is NP Complete. But it can be solved in polynomial time for trees. Thus, the below solution runs in polynomial time.

CODE:

```
/* Minimum Vertex Cover Dunamic Programming Implementation */
#include <bits/stdc++.h>
using namespace std;

struct node
{
    int data;
    int vertex_cover;
    struct node* left;
    struct node* right;
};
typedef struct node Node;

Node* getNode(int info)
{
    Node* create = (Node* )malloc(sizeof(Node));
    create->left = NULL;
    create->right = NULL;
    create->data = info;
    create->vertex_cover = 0;
    return create;
}

int vcover(Node* root)
{
    if(root == NULL)
        return 0;
    if(root->left == NULL && root->right == NULL)
        return 0;

    if(root->vertex_cover != 0)
    {
        return root->vertex_cover;
    }

    //size including root element
    int r_in = 1 + vcover(root->left) + vcover(root->right);

    //size excluding root eleemnet
    int r_out = 0;
    if(root->left)
        r_out += 1 + vcover(root->left->left) + vcover(root->left->right);
    if(root->right)
        r_out += 1 + vcover(root->right->right) + vcover(root->right->left);

    root->vertex_cover = min(r_out, r_in);
    return min(r_out, r_in);
```

```
}
int main()
{
    Node* root                = getNode(20);
    root->left                = getNode(8);
    root->left->left          = getNode(4);
    root->left->right         = getNode(12);
    root->left->right->left   = getNode(10);
    root->left->right->right  = getNode(14);
    root->right               = getNode(22);
    root->right->right        = getNode(25);
    cout<<"Minimum Vertex Cover : "<<vcover(root)<<endl;
    return 0;
}
```

Question 37:

Write a program for the following 3-PARTITION problem. Given integers a1; : : : ; an, we want to determine whether it is possible to partition of {1; : : : ; n} into three disjoint subsets I;J;K which are equal.

For example, for input (1; 2; 3; 4; 4; 5; 8) the answer is yes, because there is the partition (1; 8), (4; 5), (2; 3; 4). On the other hand, for input (2; 2; 3; 5) the answer is no. Write a program and analyze a dynamic programming algorithm for 3-PARTITION that runs in time polynomial in n.

Time Complexity:

Polynomial Time Complexity.

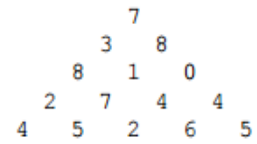
CODE:

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

//3 Partition Problem

int main()
{

    int n,a[100000],dp[1000][1000],sum,ans=0,i,j,b[100000],sum1=0;
    cin>>n;
    for(i=0;i<n;i++)
    {
    cin>>a[i];
    sum=sum+a[i];
    }

    if(sum%3!=0)
    ans=0;

    else
    {


    for(i=0;i<=n;i++)
    dp[0][i]=1;

     for(i=1;i<=sum;i++)
     dp[i][0]=0;

         for(i=1;i<=sum;i++)
         {
             for(j=1;j<=n;j++)
             {
                  dp[i][j]=dp[i][j-1];

                if(a[j]<=i)
                dp[i][j]=(dp[i][j] || dp[i-a[j]][j-1]);

            }
         }

   int x,s[100000];
   memset(s,0,sizeof(s));
    int y=sum/3;
    while(y>0)
    {
```

```
        for(j=1;j<=n;j++)
        {
            if(dp[y][j]==1)
            {
                x=j;
                break;
            }
        }
        s[x]=1;
        y=y-a[x];
    }



int k=0;
for(i=0;i<n;i++)
{
    if(s[i]==0)
    {
    b[k]=a[i];
    k++;
    sum1=sum1+b[k];
    }
}

int dp1[1000][1000];

 for(i=0;i<=k;i++)
    dp1[0][i]=1;

     for(i=1;i<=sum1;i++)
     dp1[i][0]=0;

         for(i=1;i<=sum1;i++)
         {
             for(j=1;j<=k;j++)
             {
                  dp1[i][j]=dp1[i][j-1];

                if(b[j]<=i)
                dp1[i][j]=(dp1[i][j] || dp1[i-b[j]][j-1]);

             }
         }

         if(dp1[sum1/2][k])
         ans=1;
}
if(ans==1)
cout<<"Yes";
else
cout<<"No";

return 0;
}
```

Question 38:

Write a program that calculates the highest sum of numbers passed on a route that starts at the top and ends somewhere on the base.



For the above figure shows a number triangle and its output is 30(7,3,8,7,5). Each step can go either diagonally down to the left or diagonally down to the right.

Time Complexity:

$T(n) = O(2^n)$

CODE:

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

int maxcost(int a[][100], int m, int n)
{
    if(n < 0 || m < 0)
        return 0;
    if(n == 0 && m == 0)
        return a[m][n];
    else
    {
        return a[m][n] + max(maxcost(a, m-1, n-1), maxcost(a, m-1, n));
    }
}
void init(int a[][100])
{
    for(int i = 0;i<100;i++)
    {
        for(int j = 0;j < 100;j++)
        {
            a[i][j] = 0;
        }
    }
}
int main()
{
    int n;
    cin>>n;
    int a[100][100];
    for(int i = 0;i<n;i++)
    {
        for(int j = 0;j <= i;j++)
        {
            cin>>a[i][j];
        }
    }
    int cost[n];

    for(int i = 0;i<n;i++)
    {
        cost[i] = maxcost(a,3,i);
    }
    int max = cost[0];
```

```
    for(int i = 1;i<n;i++)
    {
        if(cost[i] > max)
            max = cost[i];
    }
    cout<<"Max Cost is : "<<max<<endl;
    return 0;
}
```

Question 40.1:

Write a program using dynamic programming for yours own two problems and prove its complexity in polynomial.

Time Complexity:

T(n) = O(n)

CODE:

```
/* nth Fibonacci Number using Bottom Up Dynamic Programming Approach
    Fibonacci Series used is : 0 1 1 2 3 5 8 13 ---- */

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

int fib(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    int a = 0, b=1, c;
    for(int i = 1; i <= n-2;i++)
    {
        c = a + b;
        a = b;
        b = c;
    }
    return c;
}
int main()
{
    int n;
    cin>>n;
    cout<<n<<" th fibonacci sequence is : "<<fib(n)<<endl;
    return 0;
}
```

Question 40.2:

Write a program using dynamic programming for yours own two problems and prove its complexity in polynomial.

Time Complexity:

T(n) = O(sum*n)

CODE:

```
/* 2 partition problem */
#include <bits/stdcpp.h>
using namespace std;

bool findPartiion (int arr[], int n)
{
    int sum = 0;
    int i, j;

    for (i = 0; i < n; i++)
      sum += arr[i];

    if (sum%2 != 0)
       return false;

    bool part[sum/2+1][n+1];

    for (i = 0; i <= n; i++)
      part[0][i] = true;

    for (i = 1; i <= sum/2; i++)
      part[i][0] = false;

     for (i = 1; i <= sum/2; i++)
     {
       for (j = 1; j <= n; j++)
       {
         part[i][j] = part[i][j-1];
         if (i >= arr[j-1])
           part[i][j] = part[i][j] || part[i - arr[j-1]][j-1];
       }
     }
     return part[sum/2][n];
}

int main()
{
  int arr[] = {3, 1, 1, 2, 2, 1};
  int n = sizeof(arr)/sizeof(arr[0]);
  if (findPartiion(arr, n) == true)
     printf("Can be divided into two subsets of equal sum");
  else
     printf("Can not be divided into two subsets of equal sum");
  getchar();
  return 0;
}
```