Q1. DP1

predict the complexity of the code:

```
vector<vector<int > > v(100,vector <int> (100,-1) );
int func(int n,int m)
{
    if((n==0)||(m==0))
    {
        return 0;
    }
    else
    {
        if(dp[m][n]!=-1)
        {
            return dp[m][n];
        }
        else
        {
            return func(n-1,m-2)+func(n-2,m-1);
        }

    }


}
```

O(n^2)

O(n logn)

O(2^n)

O(n^3)

Q2. DP2

predict the complexity of the code:

```
vector<vector<int > > v(100,vector <int> (100,-1) );
int func(int n,int m)
{
    if((n==0)||(m==0))
    {
        return 0;
    }
    else
    {
        if(dp[m][n]!=-1)
        {
            return dp[m][n];
        }
        else
```

```
            {    lli x;
                 int i;
                 for(i1;i<n;i++)
                 {
                     for(j=1;j<m;j++)
                     {
                         x=x+ func(n-i,m-j);
                     }
                 }
                 dp[m][n]=x;
                 return x;
            }

        }


    }
```

O(n^3)

O(n^4)

O(2^n)

O(n^2)

Q3. DP3

Predict output of the following code: Input(7)

```
 vector<int> dp(11,-1);

int func(int x)

{
    if((x==0)||(x==1))
        return 2;
    else
    {
        if(dp[x]!=-1)
            return dp[x];
        else
        {

            dp[x]=func(x-1)+func(x-2);
            return dp[x];
        }

    }
}

int main() { for(i=0;i<5;i++) { cout<<func(i)<<" "; } }
```

2 2 4 8 12

2 2 4 6 10

2 4 6 10 16

Q4. DP4

predict the complexity of the code:

```
vector<vector<int > > dp(100,vector <int> (100,-1) );
int func(int n,int m)
{   lli x;
    if((n==0)||(m==0))
    {
        return 0;
    }
    else
    {
        if(dp[m][n]!=-1)
        {
            return dp[m][n];
        }
        else
        {
            dp[n][m]= func(n-1,m-2)+func(n-2,m-1);
            return dp[n][m];

        }

    }


}
```

O(n^2)

O(n logn)

O(2^n)

O(n^3)

Q5. DP5

Complexity of matrix chain multiplication using Dynmaic Programming:

O(n^2)

O(n^3)

O(n^2 logn)

O(n^4)

Q6. DP6

Efficient (Memory) Table size of Matrix Chain Multipilcation of Matrix A1 to AN:

O(n^2)

B. O(n^2/2)

C. O(n)

None of these

Predict True and False: every Greedy Solution Can Be solved using dynamic Programming

True

False

Q8. DP8

Given two sequences A and B, what will be the length of the longest subsequence present in both of them if we define L[m][n] , LCS of strings A[1….m] and B [1…..n] ?

L[i][j]=L[i-1][j-1]+1 , when A[i]==B[j]

L[i][j]=(abs(L[i-1][j]-L[i][j-1])+L[i][j-1]+L[i-1][j]) /2 , when A[i]!=B[j]

L[i][j]=max(L[i-1][j],L[i]][j-1]) , when A[i]!=B[j]

All of these

Q9. DP9

Given an array A consisting of n integers , we define beauty of the array as maximum sum of some contiguous elements with performing atmost one operation which is , you can multiply any subarray with ‘x’ but only once.

What will be the maximum subarray sum for this input

N=5,x=-2

A=[-3,8,-2,1,-6]

(Bonus :: Can you write the effiicient code for it ?)

0

22

24

12

Q10. DP10

Given two strings , what could be the best complexity that you can solve in , the K-ordered LCS of the given strings ?

A k-ordered LCS is defined to be the LCS of two sequences if you are allowed to change at most k elements in the first sequence to any value you wish to.

Defining N and M as the length of the given strings respectively.

O(N*M*logk)

O(*N*M*k)

O(N*M)

O(N*k*logM)