14) Write a recursive function to find the length of the longest common subsequence.

between two strings.

Sample I/O:

Input: ACDBE", Y = "ABCDE"

Output: 4

//(Here LCS: "ACDE")

Ans:

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

int lcs(string s1,string s2,int n1,int n2)
{
   if(n1==0||n2==0) return 0;
   if(s1[n1-1]==s2[n2-1])
      return 1+lcs(s1,s2,n1-1,n2-1);
   else
   {
      return max(lcs(s1,s2,n1-1,n2),(lcs(s1,s2,n1,n2-1)));
   }
}
int main()
{
   string x,y;
   cin >> x >> y;
   int n1= x.size();
   int n2= y.size();
   int lcs_len = lcs(x,y,n1,n2);
   cout << lcs_len << endl;
   return 0;
}
```

Output:

15) Write a program to count the number of ways to make change for a value N using given.

coin denominations.

Input: coins = {1, 2, 3}, N = 4

Output: 4

//(Ways: {1,1,1,1}, {1,1,2}, {2,2}, {1,3})

Ans:

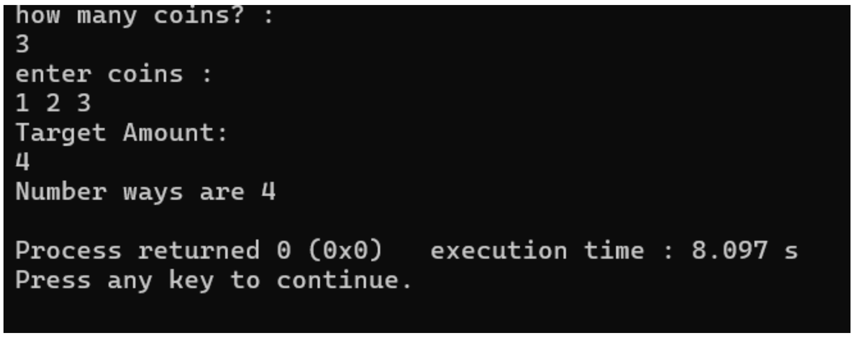
```
#include<bits/stdc++.h>
using namespace std;

int ways(int c[],int nC, int n)
{
   int dp[n+1] = {0};
   dp[0]=1;
   for(int i=0;i<=nC;++i)
   {
      for(int amn= c[i];amn<=n;amn++)
      {
         dp[amn]+=dp[amn-c[i]];
      }
   }
   return dp[n];
}

int main()
{
   int x,y;
   cout << "how many coins? :" << endl;
   cin >> x;
   int z[x];
   cout << "enter coins :" << endl;
   for(int i=0;i<x;++i)
   {
      cin >> z[i];
```

```
    }
    cout << "Target Amount:" << endl;
    cin >> y;
    cout << "Number ways are " << ways(z,x,y) << endl;
    return 0;
}
```

Output:



```
how many coins? :
3
enter coins :
1 2 3
Target Amount:
4
Number ways are 4

Process returned 0 (0x0)   execution time : 8.097 s
Press any key to continue.
```

16) Write a program to find the minimum number of coins needed to make a given value N.

Input: coins = {1, 3, 4}, N = 6

Output: 2

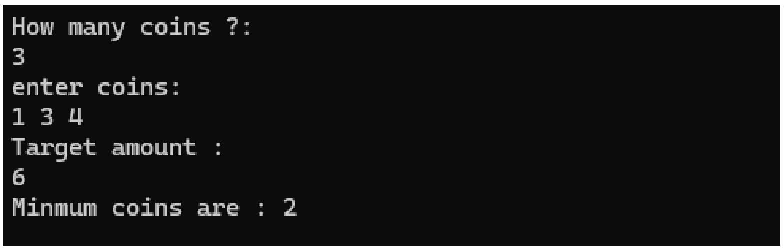
//(Using coins: {3, 3} or {4, 1, 1})

Ans:

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


#define mx 1000
int dp[mx][mx];
int minC(int c[],int n,int t_amnt)
{
    if(t_amnt==0) return 0;
    if(t_amnt<0) return 1e9;
    if(n==0) return 1e9;
    if(dp[n][t_amnt]==-1)
        dp[n][t_amnt]=min(
         minC(c,n,t_amnt-c[n-1])+1,
         minC(c,n-1,t_amnt));
    return dp[n][t_amnt];
}
int main()
{
    int x,y;
    cout << "How many coins ?:" << endl;
    cin >> x;
    int z[x];
    cout << "enter coins:" << endl;
    for(int i=0;i<x;++i)
    {
        cin >> z[i];
    }
    cout << "Target amount :" << endl;
    cin >> y;
    memset(dp,-1,sizeof(dp));
    int r= minC(z,x,y);
    if(r>=1e9) cout << "Not possible" << endl;
    else cout << "Minmum coins are : " << r << endl;
    return 0;
}
```

Output:



```
How many coins ?:
3
enter coins:
1 3 4
Target amount :
6
Minmum coins are : 2
```

17) Write a program to solve the 0/1 knapsack problem. Given weights and values of n.

items, and a knapsack capacity W, determine the maximum total value that can be

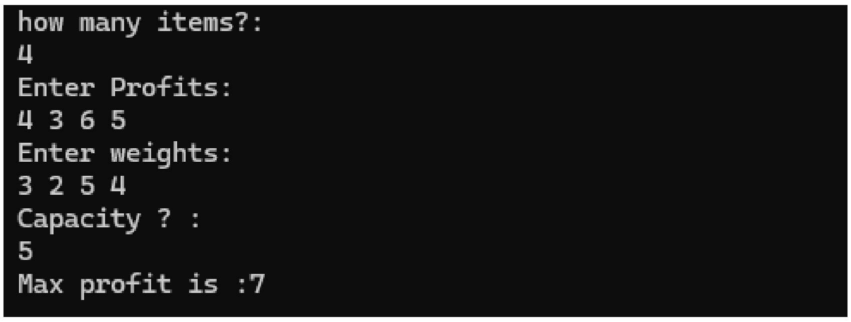
carried.

Ans:

```
#include<bits/stdc++.h>
using namespace std;
int dp[100][100];
int k(int n, int w, int p[],int wt[])
```

```
{
   if(n==0||w==0) return 0;
   if(dp[n][w]!=-1) return dp[n][w];
   if(wt[n-1]>w) return dp[n][w]=k(n-1,w,p,wt);
   else
      return dp[n][w]= max(k(n-1,w,p,wt),
                     p[n-1]+k(n-1,w-wt[n-1],p,wt));
}
int main()
{
   int n;
   cout << "how many items?:" << endl;
   cin >> n;
   int p[n],wt[n];
   cout << "Enter Profits:" << endl;
   for(int i=0;i<n;++i) cin >> p[i];
   cout << "Enter weights:" << endl;
   for(int i=0;i<n;++i) cin >> wt[i];
   int w;
   cout << "Capacity ? :" << endl;
   cin >> w;
   memset(dp,-1,sizeof(dp));
   cout << "Max profit is :" << k(n,w,p,wt) << endl;
   return 0;
}
```

Output:



```
how many items?:
4
Enter Profits:
4 3 6 5
Enter weights:
3 2 5 4
Capacity ? :
5
Max profit is :7
```

Fractional Knapsack:

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

void fractionalknapsack(int n, float value[], float weight[], float capacity) {
    int index[n];
    for (int i = 0; i < n; i++) {
        index[i] = i;
    }

    // Sorting items by value-to-weight ratio
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            float r1 = value[index[i]] / weight[index[i]];
            float r2 = value[index[j]] / weight[index[j]];
            if (r1 < r2) {
                swap(index[i], index[j]);
            }
        }
    }

    float totalvalue = 0.0;
    // cout << "\nItems taken (value, weight, fraction taken):\n";

    for (int i = 0; i < n && capacity > 0; i++) {
        int item = index[i];
        if (weight[item] <= capacity) {
            capacity -= weight[item];
            totalvalue += value[item];
            // cout << value[item] << " " << weight[item] << " (1.0)\n";
        } else {
            float fraction = capacity / weight[item];
            totalvalue += value[item] * fraction;
            // cout << value[item] << " " << weight[item] << " (" << fraction << ")\n";
            capacity = 0;
        }
    }

    cout << "\nMaximum total value: " << totalvalue << endl;
}

int main() {
    int n;
    float capacity;

    cout << "Enter number of items: ";
    cin >> n;

    float value[n], weight[n];

    cout << "Enter values of items:\n";
    for (int i = 0; i < n; i++) {
        cin >> value[i];
    }

    cout << "Enter weights of items:\n";
    for (int i = 0; i < n; i++) {
        cin >> weight[i];
    }

    cout << "Enter capacity of knapsack: ";
    cin >> capacity;

    fractionalknapsack(n, value, weight, capacity);
//
```

```
int main() {
    int n = 3;
    float value[] = {60, 100, 120};
    float weight[] = {10, 20, 30};
    float capacity = 50;

    fractionalknapsack(n, value, weight, capacity);

    return 0;
}
```

```
 return 0;

}
```

Fibonacci DP: n fib printing

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

using namespace std;


int dp[100];

int fibo (int n)

{

   //memset(dp,-1,sizeof(dp));

   if(n<=1) return n;

   if(dp[n]!=-1) return dp[n];

   dp[n]=fibo(n-1)+fibo(n-2);

   return dp[n];

}


int main()

{

   int n;

   cin >> n;

   memset(dp, -1, sizeof(dp));

   for(int i=0;i<n;++i)

   {

      cout << fibo(i) << " ";

   }

  // cout << fibo(n);
```

```
   return 0;
}
```

Fibonacci Recursion: nth fib

```
#include<bits/stdc++.h>
using namespace std;
int nfib(int n)
{
   if(n<=1) return n;
   else return nfib(n-1)+nfib(n-2);
}
int main()
{
   int x;
   cin >> x;
   cout << nfib(x) << endl;
   return 0;
}
```

AD list / Matrix:

```
#include<bits/stdc++.h>
using namespace std;
int graph[10][10];
int main()
{
   int vertex,edges;
   cin >> vertex >> edges;
   int begin,end;
  //  int graph[vertex][vertex];
   for(int i=0;i<edges;i++)
   {
      cin >> begin >> end;
      graph[begin][end]=1;
      graph[end][begin]=1;
   }
//LIST
   for(int i=0;i<vertex;++i)
```

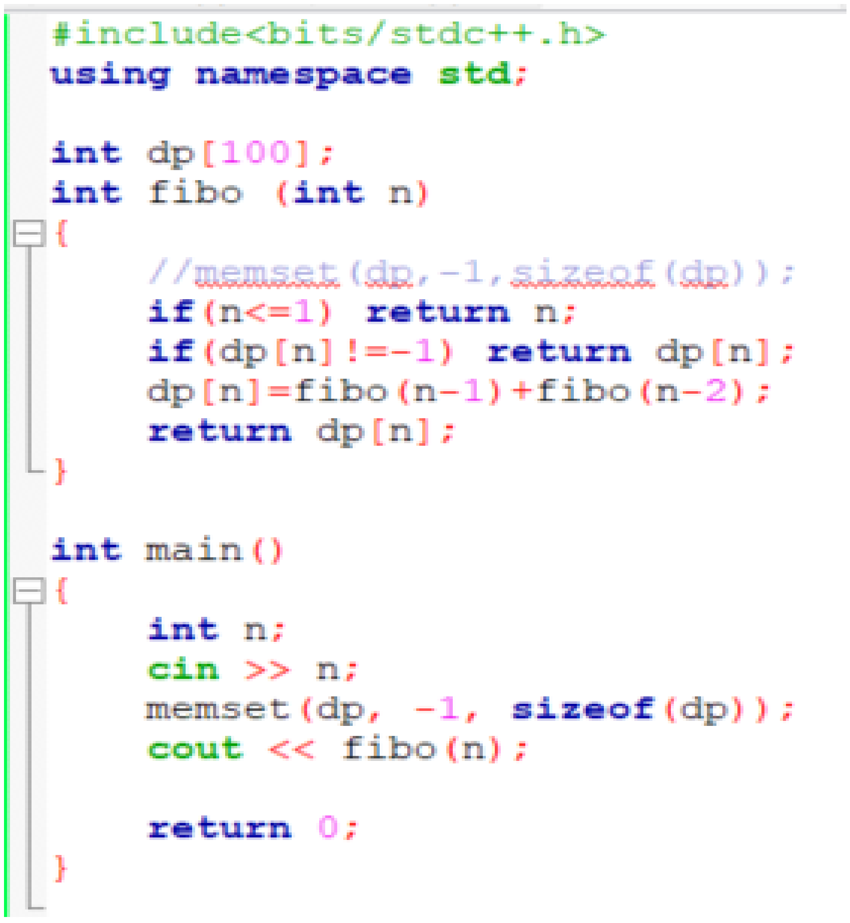
```
    {
        printf("%d->", i);
        for(int j=0;j<vertex;j++)
        {
            if(graph[i][j]==1)
                cout << j << " ";
        }
        cout << "\n";
    }
//MATRIX
    for(int i=0;i<vertex;++i)
    //{
      //  for(int j=0;j<vertex;j++)
        //{
        //    printf("%d " , graph[i][j]);
        //}
        //cout << "\n";
    //} return 0;
}
```

Printing n fibo numbers using recursion :

```
#include <bits/stdc++.h>
using namespace std;
int nfib(int n) {
    if (n <= 1) return n;
    else return nfib(n - 1) + nfib(n - 2);
}
int main() {
    int x;
    cin >> x;

    for (int i = 0; i < x; i++) {
        cout << nfib(i) << " ";
    }
    cout << endl;
    return 0;
}
```

Printing nth fib using dp :



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

int dp[100];
int fibo (int n)
{
    //memset(dp,-1,sizeof(dp));
    if(n<=1) return n;
    if(dp[n]!=-1) return dp[n];
    dp[n]=fibo(n-1)+fibo(n-2);
    return dp[n];
}

int main()
{
    int n;
    cin >> n;
    memset(dp, -1, sizeof(dp));
    cout << fibo(n);

    return 0;
}
```