```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HW3
{

    public struct PriceStruct
    {
        public int price;
        public int day;
    }

    public struct Q7_Thief_Items
    {
        public int value;
        public int weight;
    }

    //----- Variables created for LCS problem
    public enum LCS_Dir {Up, Left, Diagonal, End };
    public struct LCS_Cell{
        public LCS_Dir dir;
        public int lcs_len;
    }

    class Program
    {
        static void Main(string[] args)
        {

            //--------------------------------------------------------------------------------
                ---//
            //                   Question #3 - Cut Rod Problem with Cut
                Cost                       //
            //--------------------------------------------------------------------------------
                ---//

            int Q3_n = 5;
            int c = 2;
            int[] s; //---- Need to dummy initialize to prevent compile error
            int[] prices = Create_Cut_Rod_Prices(Q3_n, 10, 20, 30);
            //int[] prices = { -1, 10, 20, 30, 999, 10 };
            Console.WriteLine("Question #3 - Create an algorithm for the cut rod problem where
                 cutting the rod has a cost c = {0}"  ,c);
            PrintArray(prices);
            int max_revenue = Q3_Cut_Rod_With_Cost(Q3_n, prices, c,  out s);

            //---- Print the prices
            Console.WriteLine("Maximum possible revenue is: {0}" , max_revenue);
            Q3_Print_Cut_Rod_Pieces(s);
            Console.Write("\n\n\n");

            //--------------------------------------------------------------------------------
                ---//
            //                   Question #4 - Prove Longest Common                        ↩
                Subsequence                  //
            //---------------------------------------------------------------------------- ↩
                ---//

            int[] X = { -1, 1, 0, 0, 1, 0, 1, 0, 1 };
            int[] Y = { -1, 0, 1, 0, 1, 1, 0, 1, 1, 0 };
            LCS_Cell[,] LCS_Matrix;

            Console.Write("Question #4 - Find the longest subsequnce for sequences: \n\nX =    ↩
                ");
            PrintArray(X);
            Console.Write("\nY = ");
            PrintArray(Y);

            //---- Parse the sequences and generate the best subsequence
            Q4_Longest_Common_Subsequence(X, Y, out LCS_Matrix);
            Q4_Print_Subsequence(X, LCS_Matrix);
            Console.Write("\n\n\n");

            //---------------------------------------------------------------------------- ↩
                ---//
            //     Question #5 - Reconstruct LCS Array without Using Array B (i.e.         ↩
                Arrows)       //
            //---------------------------------------------------------------------------- ↩
                ---//

            Console.WriteLine("Question #5 - Reconstruct the LCS Without Using Array \"b      ↩
                \" (i.e. arrow array).");
            Q5_Reconstruct_LCS(X, Y, LCS_Matrix);
            Console.Write("\n\n\n");

            //---------------------------------------------------------------------------- ↩
                ---//
            //                       Question #6 - Find Maximum                            ↩
                Subarray                        //
            //---------------------------------------------------------------------------- ↩
                ---//
            int Q6_n = 10;
            int[] list_of_numbers_Q6 = Create_Array_for_Max_SubArray(Q6_n, 50, 5);
            //int[] list_of_numbers_Q6 = { -1, 21, 19, 17, 15, 13, 11, 9, 5, 5, 4 };
            Console.WriteLine("Question #6 - Find the maximum subarray in O(n) time.  Input    ↩
                array:");
            PrintArray(list_of_numbers_Q6);

            //---- Extract Results and print the results
            Tuple<int, int, int> Q6_out = Q6_Dynamic_Max_Subarray(list_of_numbers_Q6, Q6_n);
            Console.WriteLine("Maximum Subarray is {0} and started on day {1} and ended on day ↩
                 {2}.\n\n\n", Q6_out.Item1, Q6_out.Item2, Q6_out.Item3);

            //---------------------------------------------------------------------------- ↩
                ---//
            //                           Question #7 - Thief                               ↩
                Problem                             //
```

```
            //---------------------------------------------------------------------------- ↩
                ---//

            int Q7_n = 5;
            int max_weight = 10;
            Console.WriteLine("Question #7 - The thief has an option to steal {0} objects.     ↩
                \nThe maximum weight he can carry is {1}." +
                              "\nDetermine the best objects for him to steal.  The object      ↩
                       properties are below.\n", Q7_n, max_weight);


            Q7_Thief_Items[] objects = Q7_Generate_Thief_Objects(Q7_n, 3, 7, 10, 20);
            int[,] stolen_items;
            Print_Thief_Objects(objects);

            //----- Solve the thief problem
            Q7_Thief_Max_Value(objects, max_weight,  out stolen_items);
            Q7_Reconstruct_Stolen_Items_List(objects, stolen_items);

            int x = 0;
        }

        //----------------------------------------------------------------------------- /↩
            /
        //                   Question #3  - Cut Rod Problem with Cut                       ↩
            Cost                        //
        //----------------------------------------------------------------------------- /↩
            /


        static int Q3_Cut_Rod_With_Cost(int n, int[] prices, int cost_per_cut, out int[] s)
        {
            int[] r = new int[n+1];
            s = new int[n+1]; //----- S is a two dimensional array containing the cuts for     ↩
                indexes 1 to n
            int i, j;
            int r_temp;

            //---- Iterate through the possibilities
            for (i = 1; i < n+1; i++)
            {
                r[i] = prices[i];
                s[i] = i;
                for (j = i - 1; j >= i / 2; j--) //---- Use of i/2 is explained in note below
                {
                    r_temp = r[j] + r[i - j] - cost_per_cut;
                    if (r_temp > r[i])
                    {
                        r[i] = r_temp;
                        s[i] = i-j;
                    }
                }
            }

            return r[n];
        }

        static void Q3_Print_Cut_Rod_Pieces(int[] s)
        {
            int n = s.Length-1;
            string str = s[n].ToString() ;
            n -= s[n];
            while (n > 0)
            {
                str += ", " + s[n].ToString();
                n -= s[n];
            }
            Console.WriteLine("The rod pieces are length: " + str);
        }

        static int[] Create_Cut_Rod_Prices(int n, int min_starting_price, int 
            max_starting_price, int max_cost_change)
        {
            int cnt;
            int[] list_of_numbers;
            Random rand = new Random();

            //------ Populate Memory
            list_of_numbers = new int[n + 1];

            //---- Initialize starting price
            list_of_numbers[1] = rand.Next(min_starting_price, max_starting_price + 1);

            for (cnt = 2; cnt <= n; cnt++)
            {
                list_of_numbers[cnt] = list_of_numbers[cnt - 1] + rand.Next(1, max_cost_change 
                     + 1);
            }

            return list_of_numbers;

        }


        //------------------------------------------------------------------------------- /
            /
        //                 Question #4  - Prove Longest Common 
            Subsequence                  //
        //------------------------------------------------------------------------------- /
            /

        static void Q4_Longest_Common_Subsequence(int[] X, int[] Y, out LCS_Cell[,] 
            LCS_Matrix){
            int m = X.Length-1;
            int n = Y.Length-1;
            int i, j;
            LCS_Cell temp_LCS_Cell;

            //---- Initialize array referred to as b and c
            LCS_Matrix = new LCS_Cell[m + 1, n + 1];
            //----- Initialize the arrays
            temp_LCS_Cell.dir = LCS_Dir.End;
            temp_LCS_Cell.lcs_len = 0;
            for (i = 0; i <= m; i++) LCS_Matrix[i, 0] = temp_LCS_Cell;
            for (j = 0; j <= n; j++) LCS_Matrix[0, j] = temp_LCS_Cell;

            //----- Iterate through all cells in the array and generate the matrix values
            for (i = 1; i <= m; i++)
            {
                for (j = 1; j <= n; j++)
                {
                    //---- X[i] and Y[i] are the same so mark as part of a sequence
                    if (X[i] == Y[j])
                    {
                        temp_LCS_Cell.dir = LCS_Dir.Diagonal;
                        temp_LCS_Cell.lcs_len = LCS_Matrix[i-1,j-1].lcs_len + 1;
                    }

                    //---- X[i] and Y[j] not in the same sequence so point to longest
                        subsequence
                    else if(LCS_Matrix[i-1,j].lcs_len >= LCS_Matrix[i,j-1].lcs_len){
                        temp_LCS_Cell.dir = LCS_Dir.Up;
                        temp_LCS_Cell.lcs_len = LCS_Matrix[i-1,j].lcs_len;
                    }
                    else{
                        temp_LCS_Cell.dir = LCS_Dir.Left;
                        temp_LCS_Cell.lcs_len = LCS_Matrix[i, j-1].lcs_len;
                    }
                    LCS_Matrix[i, j] = temp_LCS_Cell; //--- Store data structure into matrix
                }
            }
        }

        static void Q4_Print_Subsequence(int[] X, LCS_Cell[,] LCS_Matrix)
        {
            int i = LCS_Matrix.GetLength(0)-1;
            int j = LCS_Matrix.GetLength(1)-1;
            string print_str = "";

            while (LCS_Matrix[i, j].dir != LCS_Dir.End)
            {
                if (LCS_Matrix[i, j].dir == LCS_Dir.Diagonal)
                {
                    //---- Generate sequence
                    if (print_str == "") print_str = X[i].ToString();
                    else print_str = X[i].ToString() + ", " + print_str;
                    i--;
                    j--;
                }
                else if (LCS_Matrix[i, j].dir == LCS_Dir.Left) j--;

                else                                        i--;
            }
            Console.WriteLine("A longest common subsequence is: {0}.", print_str);
        }

        //------------------------------------------------------------------------------- /↩
            /
        //     Question #5 - Reconstruct LCS Array without Using Array B (i.e.               ↩
            Arrows)       //
        //------------------------------------------------------------------------------- /↩
            /

        static void Q5_Reconstruct_LCS(int[] X, int[] Y, LCS_Cell[,] LCS_Matrix)
        {
            //---- Subtracting since C/C++/C# start at index 0 so have dummy index 0           ↩
                increasing length by 1
            int i = X.Length - 1;
            int j = Y.Length - 1;
            string print_str = "";

            while (i > 0 && j > 0)
            {
                //----- In this case, the two values are equal, move diagonally
                if (X[i] == Y[j])
                {
                    if (print_str == "") print_str = X[i].ToString();
                    else print_str = X[i].ToString() + ", " + print_str;
                    i--;
                    j--;
                }
                //----- Elements do not match so take the path (up or left) with the longest   ↩
                    common subsequence
                else if (LCS_Matrix[i - 1, j].lcs_len >= LCS_Matrix[i, j - 1].lcs_len) i--;
                else j--;
            }

            Console.WriteLine("A longest common subsequence is: {0}.", print_str);
        }


        //------------------------------------------------------------------------------- /↩
            /
        //                       Question #6 - Find Maximum                                    ↩
            Subarray                         //
        //------------------------------------------------------------------------------- /↩
            /

        static Tuple<int, int, int> Q6_Dynamic_Max_Subarray(int[] list_of_prices, int n)
        {
            Tuple<int, int, int> Output_Results;
            int max_subarray = -1, max_subarray_start=-1, max_subarray_end =-1;
            PriceStruct[] MaxPrice, MinPrice;
            PriceStruct temp_price;
            int i;

            //----- Initialize arrays containing the maximum and minimum prices for the        ↩
                previous segments
            MaxPrice = new PriceStruct[n+1]; //--- MaxPrice is the maximum price from index i 
                to index n
            MinPrice = new PriceStruct[n+1]; //--- MinPrice is the minimum price from index 1 
                to index i

            //----- For each day i = 1 to n, find the max price between that day and all later 
                 days
            temp_price.price = list_of_prices[n];
            temp_price.day = n;
            MaxPrice[n] = temp_price;
            for (i = n - 1; i >= 1; i--)
            {
                //----- Check if current price is higher than all previous prices
                if (list_of_prices[i+1] > MaxPrice[i + 1].price)
                {
                    temp_price.price = list_of_prices[i+1];
                    temp_price.day = i+1;
                    MaxPrice[i] = temp_price;
                }
                else MaxPrice[i] = MaxPrice[i+1];
            }

            //----- For each day i = 1 to n-1, find the min price for all days before that day
            temp_price.price = list_of_prices[1];
            temp_price.day = 1;
            MinPrice[1] = temp_price;
            for (i = 2; i < n; i++)
            {
                //----- Check if current price is higher than all previous prices
                if (list_of_prices[i] < MinPrice[i-1].price)
                {
                    temp_price.price = list_of_prices[i];
                    temp_price.day = i;
                    MinPrice[i] = temp_price;
                }
                else MinPrice[i] = MinPrice[i - 1];
            }


            //---- Iterate through the days to find the maximum profit made by selling on each 
                 day.
            max_subarray = int.MinValue;//---- Set to minimum value so always overwritten
            for (i = 1; i < n; i++)
            {
                if (MaxPrice[i].price - MinPrice[i].price > max_subarray)
                {
                    max_subarray = MaxPrice[i].price - MinPrice[i].price;
                    max_subarray_start = MinPrice[i].day;
                    max_subarray_end = MaxPrice[i].day;
                }
            }


            //----- Return the results
            Output_Results = Tuple.Create<int, int, int>(max_subarray, max_subarray_start, 
                max_subarray_end);
            return Output_Results;
        }


        static int[] Create_Array_for_Max_SubArray(int n, int max_starting_value, int 
            max_daily_change)
        {
            int cnt;
            int[] list_of_numbers;
            int daily_change;
            int new_value;
            Random rand = new Random();

            //------ Populate Memory
            list_of_numbers = new int[n+1];

            //---- Initialize starting price
            list_of_numbers[0] = -1;
            list_of_numbers[1] = rand.Next(1, max_starting_value+1);

            for (cnt = 2; cnt <= n; cnt++)
            {
                daily_change = rand.Next( -1*(max_daily_change+1), (max_daily_change + 
                    1) ); //---- Normalize max daily change between -max_daily_change to 
                    max_daily_change
                new_value = list_of_numbers[cnt - 1] + daily_change;

                if (new_value > 0)
                    list_of_numbers[cnt] = new_value;
                else
                    list_of_numbers[cnt] = 0;
            }

            return list_of_numbers;

        }

        //--------------------------------------------------------------------------------- /
            /
        //                          Question #7  - Thief 
            Problem                           //
        //--------------------------------------------------------------------------------- /
            /

        static void Q7_Thief_Max_Value(Q7_Thief_Items[] objects, int max_weight, out int[,] 
            stolen_items)
        {
            int n = objects.Length - 1; //------Subtracting 1 from length due to making array 
                from 1 to n
            int i, j;
            int v, w;
            stolen_items = new int[n + 1, max_weight + 1];

            //------iterate through the weights and set them to zero
            for (j = 0; j <= max_weight; j++)
                stolen_items[0, j] = 0;

            //----- Initialize the items to zero value and zero weight
            for (i = 0; i <= n; i++)
                stolen_items[i, 0] = 0;

            //------ Build a two dimensional array similar to LCS problem
            for (i = 1; i <= n; i++)
            {
                w = objects[i].weight;
                v = objects[i].value;

                //----- j is the available weight
                for (j = 1; j <= max_weight; j++)
                    //---- Check if this item added to an earlier weight is bigger than 
                       current value
                    if (j - w >=0 && stolen_items[i - 1, j] < stolen_items[i-1, j - w] + v)
                        stolen_items[i, j] = stolen_items[i-1, j - w] + v;
                    else
                        stolen_items[i, j] = stolen_items[i - 1, j];
            }

            Console.WriteLine("The thief can steal a maximum of ${0}.\n" , stolen_items[n, 
                max_weight]);

        }

        static void Q7_Reconstruct_Stolen_Items_List(Q7_Thief_Items[] objects,  int[,] 
            stolen_items)
        {

            int i = stolen_items.GetLength(0) - 1;
            int j = stolen_items.GetLength(1) - 1;
            int numb_stolen_items =0;
            int[] list_stolen_items = new int[i];

            while (i > 0 && j > 0)
            {
                //------check if item i is part of the optimal solution
                if (stolen_items[i, j] == stolen_items[i - 1, j])
                    i--;
                //---- Check if this is the minimum weight for this optimal solution
                else if (stolen_items[i, j] == stolen_items[i, j - 1])
                    j--;
                else
                {
                    //---- Item i was stolen so decrement weight by weight of object i
                    list_stolen_items[numb_stolen_items] = i;
                    numb_stolen_items++;
                    j -= objects[i].weight;
                    i--;
                }
            }
            //---- Print the stolen items.  Print them backwards so they are in ascending
                order
            Console.Write("The objects that were stolen were: " );
            for (i = numb_stolen_items; i > 0; i--)
            {
                if (i != numb_stolen_items)
                    Console.Write(",\t");

                Console.Write(list_stolen_items[i-1].ToString());
            }
            Console.Write(".\n\n");
        }


        static Q7_Thief_Items[] Q7_Generate_Thief_Objects( int n, int min_weight, int
            max_weight, int min_value, int max_value)
        {
            Q7_Thief_Items[] objects = new Q7_Thief_Items[n + 1];//----- Add one so the code
                can treat the array as 1 to n.
            int i;
            Q7_Thief_Items temp_thief_object;
            Random rand = new Random();

            for (i = 1; i <= n; i++)
            {
                temp_thief_object.value = rand.Next(min_value, max_value + 1);
                temp_thief_object.weight  = rand.Next(min_weight, max_weight + 1);
                objects[i] = temp_thief_object;
            }

            return objects;
        }

        static void Print_Thief_Objects(Q7_Thief_Items[] objects)
        {
            int cnt;
            int n = objects.Length;
            string comma_str;
            string[] print_str= { "", "", "" };


            print_str[0] = "Item# = [\t";
            print_str[1] = "Value = [\t";
            print_str[2] = "Weight = [\t";

            for (cnt = 1; cnt < n; cnt++)
            {
                if (cnt > 1) comma_str = ",\t";
                else comma_str = "";

                print_str[0] += comma_str + cnt.ToString();
                print_str[1] += comma_str + objects[cnt].value.ToString();
                print_str[2] += comma_str + objects[cnt].weight.ToString();

            }
            for (cnt = 0; cnt < print_str.Length; cnt++ )
            {
                print_str[cnt] += "\t]";
                Console.WriteLine(print_str[cnt]);
            }
            Console.Write("\n");
        }

        //---------------------------------------------------------------------------------- /
            /
        //                                Helper
            Functions                                   //
        //---------------------------------------------------------------------------------- /
            /

        static void PrintArray(int[] print_array)
        {
            int cnt;
            int n = print_array.Length;

            Console.Write("[ ");
            for (cnt = 1; cnt < n; cnt++)
            {
                Console.Write(Convert.ToString(print_array[cnt]));
                if (cnt + 1 != n) Console.Write(", ");
                else Console.WriteLine(" ]");
            }
        }
    }
}
```