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


namespace CS255_HW4
{
    class Program
    {
        struct Activity
        {
            public int start;   //---- Start time of Activity
            public int end;     //---- End time of Activity
            public int value;   //---- Value of Activity
        }

        static void Main(string[] args)
        {

            int Q3_n = 100;
            int max_end_time = 200;
            int max_activity_value = 5;
            long start_time;
            Stopwatch stop_watch = new Stopwatch();


            //---------------------------------------------------------------//
            //                         Question #3                           //
            //---------------------------------------------------------------//

            //----- Create the list of activities
            Activity[] list_of_activities = Create_Activities_List(Q3_n, max_end_time,
                max_activity_value);

            Q3_Print_Activities(list_of_activities);

            //---- Calculate Brute Force Results Recursively
            stop_watch.Start();
            int activity_results_BF = Q3_Activity_Selection_With_Value_Brute_Force
                (list_of_activities, Q3_n, 1, max_end_time);
            Console.WriteLine("BRUTE FORCE: A maximum value for the set of activities using
                brute force is {0}.", activity_results_BF);
            Console.Write("BRUTE FORCE: Elapsed Time is: {0}.\n\n\n\n\n" ,
                stop_watch.ElapsedMilliseconds);


            //---- Calculate Dynamic Programming Result Bottom Up
            start_time = stop_watch.ElapsedMilliseconds;
            int[, ,] activity_results_DP = Q3_Activity_Selection_with_Value_DP
                (list_of_activities, max_end_time);
            Console.WriteLine("DP: A maximum value for this set of activities using DP is
                {0}.", activity_results_DP[Q3_n, 1, max_end_time]);
            Console.WriteLine("DP: Elapsed Time is: {0}.\n\n\n" ,
                stop_watch.ElapsedMilliseconds - start_time);
            Q3_Print_Activity_Selection_DP(list_of_activities, activity_results_DP,
                list_of_activities.Length - 1, 1, max_end_time);


            //---- Ensure the results are the same
            if (activity_results_BF == activity_results_DP[Q3_n, 1, max_end_time])
                Console.Write("Success: Brute Force and DP yielded the same maximum value." );
            else{
                Console.Write("ERROR, ERROR ERROR: Brute Force and DP yielded the same maximum
                     value.");
                Debug.Assert(false);
            }

            //-----------------------------------------------------------------//
            //                         Question #4                             //
            //-----------------------------------------------------------------//

            int m = 50;
            int total_distance = 500;

            Console.WriteLine("Problem #4: Professor Gecko skating greedy algorithm problem.
                \n");

            //---- This function stores number of stops in index 0 of the returned array
            int[] list_of_possible_stops = Q4_Generate_List_Of_Stops(total_distance, m);

            Q4_MinStops(list_of_possible_stops, list_of_possible_stops[0], m);


        }

        static int[, ,] Q3_Activity_Selection_with_Value_DP(Activity[] list_of_activities,  int
             max_time)
        {
            int n = list_of_activities.Length - 1;
            int i;
            int start_time, end_time;
            bool activity_compatible;
            Activity cur_activity;
            int val_with_activity, val_no_activity;
            int[, ,] activity_selection_results = new int[n + 1, max_time + 1, max_time + 1];

            //---- Iterate through all the activities from 1 to n;
            //---- First Outer Loop
            for (i = 1; i <= n; i++)
            {
                cur_activity = list_of_activities[i];  //---- Extract i-th activity

                //----- Iterate through all combinations of start and end times
                //----- Middle Loop
                for (start_time = 1; start_time < max_time; start_time++)
                {
                    //---- Inner Loop
                    for (end_time = start_time + 1; end_time <= max_time; end_time++)
                    {
                        //---- Verify current activity is compatible with previous set
                        if (cur_activity.start >= start_time && cur_activity.end <= end_time)
                            activity_compatible =  true;
                        else
                            activity_compatible =  false;

                        //----- If current activity is not compatible with current start and stop time, take previous value
                        if (activity_compatible == false)
                        {
                            activity_selection_results[i, start_time, end_time] = activity_selection_results[i - 1, start_time, end_time];
                        }
                        else
                        {
                            //---- Determine the value with and without the task
                            val_with_activity = activity_selection_results[i  - 1, start_time, cur_activity.start];
                            val_with_activity += activity_selection_results[i  - 1, cur_activity.end, end_time];
                            val_with_activity += cur_activity.value;

                            val_no_activity = activity_selection_results[i  - 1, start_time, end_time];

                            activity_selection_results[i, start_time, end_time] = Math.Max(val_with_activity, val_no_activity);
                        }
                    }
                }
            }

            //----- Return Activity Results Matrix
            return activity_selection_results;
        }


        static void Q3_Print_Activity_Selection_DP(Activity[] list_of_activities,  int[, ,] activity_selection_results, int n, int start_time, int end_time)
        {
            Console.Write("An optimal list of activities is = [   " );
            Q3_Print_Activity_Selection_DP_Recursive(list_of_activities, activity_selection_results, n, start_time, end_time);
            Console.WriteLine("]");
        }


        static void Q3_Print_Activity_Selection_DP_Recursive(Activity[] list_of_activities, int[, ,] activity_selection_results, int n, int start_time, int end_time)
        {
            //---- Recursion termination condition
            if (n == 0 || end_time == start_time)  return;

            //----- Check if activity i is part of the optimal solution
            if (activity_selection_results[n, start_time, end_time] ==
                activity_selection_results[n - 1, start_time, end_time])
            {
                Q3_Print_Activity_Selection_DP_Recursive(list_of_activities,
                    activity_selection_results, n - 1, start_time, end_time);
                return;
            }

            //----- Check if activity i is part of the optimal solution
            if (activity_selection_results[n, start_time, end_time] ==
                activity_selection_results[n, start_time, end_time - 1])
            {
                Q3_Print_Activity_Selection_DP_Recursive(list_of_activities,
                    activity_selection_results, n, start_time, end_time - 1);
                return;
            }

            //----- item is part of the sequence so print it
            Q3_Print_Activity_Selection_DP_Recursive(list_of_activities,
                activity_selection_results, n - 1, start_time, list_of_activities[n].start);
            Console.Write("A{0}  ", n);
            Q3_Print_Activity_Selection_DP_Recursive(list_of_activities,
                activity_selection_results, n - 1, list_of_activities[n].end, end_time);

        }



        static int Q3_Activity_Selection_With_Value_Brute_Force(Activity[] list_of_activities,
             int n, int seq_start, int seq_end)
        {

            int activity_not_part_of_sol_val;
            int activity_part_of_sol_val;
            bool n_valid = list_of_activities[n].start >= seq_start && list_of_activities
                [n].end <= seq_end;

            if (1 == n)
                if (n_valid)
                    return list_of_activities[n].value;
                else return 0;

            if (!n_valid) return Q3_Activity_Selection_With_Value_Brute_Force
                (list_of_activities, n - 1, seq_start, seq_end);

            if (2 == n)
            {
                bool first_valid = list_of_activities[1].start >= seq_start &&
                    list_of_activities[1].end <= seq_end;

                //----- Return max of 1 and n if both are valid
                if (first_valid && n_valid)
                {
                    //----- Not overlapping case
                    if (list_of_activities[1].end <= list_of_activities[n].start ||
                        list_of_activities[1].start >= list_of_activities[n].end)
                        return list_of_activities[1].value + list_of_activities[n].value;
                    //---- Overlapping case
                    else return Math.Max(list_of_activities[1].value, list_of_activities
                        [n].value);
                }
                else if (n_valid) return list_of_activities[n].value;
                else if (first_valid) return list_of_activities[1].value;

            }

            //---- Determine the maximum value if item n is NOT part of optimal solution
            activity_not_part_of_sol_val = 0;
            activity_not_part_of_sol_val = Q3_Activity_Selection_With_Value_Brute_Force
                (list_of_activities, n - 1, seq_start, list_of_activities[n].start);
            activity_not_part_of_sol_val += Q3_Activity_Selection_With_Value_Brute_Force
                (list_of_activities, n - 1, list_of_activities[n].end, seq_end);
            activity_not_part_of_sol_val += list_of_activities[n].value;

            //---- Determine the maximum value if item n IS part of optimal solution
            activity_part_of_sol_val = Q3_Activity_Selection_With_Value_Brute_Force
                (list_of_activities, n - 1, seq_start, seq_end);

            return Math.Max(activity_not_part_of_sol_val, activity_part_of_sol_val);

        }



        static Activity[] Create_Activities_List(int n, int end_time, int max_value)
        {
            //---- Create an array of activities
            Activity[] list_of_activities = new Activity[n + 1];
            Activity temp_activity;
            int i;
            Random rand = new Random();


            if (n == 11 && end_time == 25)
            {
                list_of_activities[1].start = 1;
                list_of_activities[1].end = 5;
                list_of_activities[1].value = 1;

                list_of_activities[2].start = 7;
                list_of_activities[2].end = 11;
                list_of_activities[2].value = 1;

                list_of_activities[3].start = 10;
                list_of_activities[3].end = 14;
                list_of_activities[3].value = 1;

                list_of_activities[4].start = 16;
                list_of_activities[4].end = 20;
                list_of_activities[4].value = 1;

                list_of_activities[5].start = 19;
                list_of_activities[5].end = 23;
                list_of_activities[5].value = 1;

                list_of_activities[6].start = 4;
                list_of_activities[6].end = 8;
                list_of_activities[6].value = 1;

                list_of_activities[7].start = 4;
                list_of_activities[7].end = 8;
                list_of_activities[7].value = 1;

                list_of_activities[8].start = 4;
                list_of_activities[8].end = 8;
                list_of_activities[8].value = 1;

                list_of_activities[9].start = 16;
                list_of_activities[9].end = 20;
                list_of_activities[9].value = 50;

                list_of_activities[10].start = 16;
                list_of_activities[10].end = 20;
                list_of_activities[10].value = 1;

                list_of_activities[11].start = 13;
                list_of_activities[11].end = 17;
                list_of_activities[11].value = 1;
            }
            else
            {
                for (i = 1; i <= n; i++)
                {
                    temp_activity.start = rand.Next(1, end_time);
                    temp_activity.end = rand.Next(temp_activity.start + 1, end_time + 1);
                    temp_activity.value = rand.Next(1, max_value + 1);

                    list_of_activities[i] = temp_activity;
                }
            }


            return list_of_activities;
        }



        static void Q3_Print_Activities(Activity[] list_of_activities)
        {

            int i;
            int n = list_of_activities.Length -1;
            string id_str, start_time_str, end_time_str, value_str;
            Activity cur_activity;
            id_str = "ID=\t[  ";
            start_time_str = "Start=\t[  ";
            end_time_str = "End=\t[  ";
            value_str = "Value=\t[  ";

            for (i = 1; i <= n; i++)
            {
                cur_activity = list_of_activities[i];

                id_str += i.ToString() + "\t";
                start_time_str += cur_activity.start.ToString() +  "\t";
                end_time_str += cur_activity.end.ToString() +  "\t";
                value_str += cur_activity.value.ToString() +  "\t";
            }

            //---- Close the string off
            id_str += "]";
            start_time_str += "]";
            end_time_str += "]";
            value_str += "]";

            Console.Write("The activity information is:\n");
            Console.WriteLine(id_str);
            Console.WriteLine(start_time_str);
            Console.WriteLine(end_time_str);
            Console.WriteLine(value_str);
            Console.WriteLine("\n\n\n");
        }

        //---------------------------------------------------------------//
        //                         Question #4                           //
        //---------------------------------------------------------------//


        static void Q4_MinStops(int[] list_of_possible_stops, int n, int m){

            int previous_stop = 0;
            int i;
            int numb_stops = 0;

            Console.Write("Professor Gekko needs to stop at stops: ");

            //---- Iterate through all the stops
            for(i=0; i<n; i++){
                //-----
                if (list_of_possible_stops[i + 1] - previous_stop > m)
                {
                    numb_stops++;
                    previous_stop = list_of_possible_stops[i];
                    Console.Write("#{0},  ", i);
                }
            }

            Console.Write("\n");
            Console.Write("Professor Gekko's journey required {0} stops.\n\n\n" , numb_stops);

        }

        static int[] Q4_Generate_List_Of_Stops(int total_distance, int m)
        {
            Random rand = new Random();
            int[] list_of_possible_stops = new int[total_distance];
            int numb_possible_stops = 0;
            int cur_loc = 0;

            Console.Write("The List of possible stops is printed below.  The ordered pair has ↩
                the form (Stop#, DistanceFromStart).\n" );

            //---- Continue building the list of possible stops
            while (cur_loc < total_distance)
            {
                cur_loc += rand.Next(1, m / 2);
                if (cur_loc <= total_distance)
                {
                    numb_possible_stops++;
                    list_of_possible_stops[numb_possible_stops] = cur_loc;
                    Console.Write("( {0}, {1} )  ", numb_possible_stops, cur_loc);
                }
            }
            list_of_possible_stops[0] = numb_possible_stops;
            Console.WriteLine("\n");

            return list_of_possible_stops;

        }

    }
}
```