A sequence has the restricted growth property if it is a sequence of positive integers \( a_1, a_2, \ldots, a_n \) so that

1. \( a_1 = 1 \).
2. \( a_{n+1} \leq \max_{1 \leq i \leq n} (a_i + 1) \).

Devise a polynomial time algorithm to compute the number of such sequences of length \( n \).

**Solution**

This can be solved by dynamic programming. Let \( CRGS(l, m) \) be the number of possible sequences with length \( l \) and maximum number in the sequence \( m \). The total number of such sequences for a length can be calculated by summing over all the sequences with maximum value ranging from 1 to \( n \) for length \( l \). The recurrence can be defined as

\[
CRGS(l, m) = CRGS(l-1, m-1) + m \ast CRGS(l-1, m); (CRGS(l, 1) = 1, CRGS(l, m) = 0 \text{ if } l < m) \tag{1}
\]

We can create a table of size \( l \times m \), where we fill the matrix column by column. The base case i.e. the first column of the table \( CRGS(l, 1) = 1 \) and for the sub-problems where \( l < m \) the number of sequences will be 0 because the maximum element in sequence can’t be bigger than the length of sequence. We can now fill the table starting from second column and apply the recurrence relation for every cell and obtain the value for all combinations of \( l \) and \( m \). After filling the entire table we can calculate the number of sequences by summing the last row, i.e. \( \sum_{i=0}^{n} CRGS(n, i) \).

**Proof of Correctness**

**Base Case** : If \( m = 1 \), then there can be only one possible sequence for every length as the sequence contains only positive integers. If \( l < m \), then there won’t be any sequence which follows the growth pattern, as the maximum element in a sequence of length \( l \) can be \( l \).

**Inductive Hypothesis** : We assume \( CRGS(l-1, m) \) is computed correctly.

**Inductive steps** : We can see that for a length \( l \), we add \( CRGS(l-1, m-1) \) which is the case if the maximum element (\( m \)) is the new element and it's strictly larger than the previous elements. Now, we also add \( m \ast CRGS(l-1, m) \) which covers the cases when the new element being added is less than or equal to the max of its previous elements. There can be any number till the value \( m \) possible at index \( l \), and the multiplication with \( m \) takes care of it. We have added all the possible sequences using these two parts and since we already had them computed, our solution will give us number of possible sequences.

**Runtime Complexity** : We are solving \( m \ast l \) subproblems where the maximum value of \( m \) is \( n \) in our case, so the we have a total \( n^2 \) sub problems. It takes us constant time to solve each sub problem, and our final step where we add all the possible maximum value for length \( n \) is a linear step. Therefore, total runtime is \( O(n^2) \).
**Question 2** (Investment Planning, 40 points). Susan owns a new startup company. They are making profits, but could make more if they upgraded their facilities. She has $n$ plans to upgrade her company’s facilities. Each plan $i$ has an associated cost $c_i$ and an amount of yearly profit $p_i$ that her company would earn after upgrading to this plan (currently the office uses plan 1). Susan is trying to develop a plan for her company’s growth over the next $m$ years. Initially, the company starts with $\$0$ in savings. Each year, Susan’s company can do one of two things. Either it can earn money, adding $p_i$ to its savings if the current building plan $i$ is being used. Alternatively, she can pick some other plan $j$ and pay $c_j$ from her company’s savings (this is only possible of course if she currently has at least that much in savings) in order to change to facility plan $j$ (her company will not be able to make money that year as the upgrade will be disruptive).

Design an algorithm to devise a plan for how Susan can run her company over the course of the next $m$ years in such a way as to maximize the total amount of savings that her company has at the end of that period. For full credit, your algorithm should run in time $O(nm)$ or better.

**Solution** This can be solved by dynamic programming. Let $S(p, y)$ be the maximum possible saving after year $y$ with plan $p$. We assume that $c_i$ and $p_i$ are positive. The final answer would be the $\max_{1\leq i\leq n} S(i, m)$.

We have to compute the maximal saving for a year given any plan. The recurrence can be defined as

$$S(p, y) = \begin{cases} p_1, & \text{if } y = 1 \text{ and } p = 1 \ , \\ -\infty, & \text{if } y = 1 \text{ and } p \geq 2 \ , \\ \max(\max_{1\leq i\leq n} S(i, y-1) - c_p, S(p, y-1) + p_i), & \text{otherwise}. \end{cases}$$

(2)

We create a table of $m \times n$, and since for year 1, company has no savings, Susan can’t switch to any other plan. The company can have only $p_1$ for first year, so we take this as base case and make $S(1, 1) = p_1$ and other columns for year 1 as $-\infty$. Now, we initialize another array to store maximum savings of a year, for base case it would be $p_1$. Now, we use the recurrence formula to compute other entries in the table year wise. We are using the extra space to store the maximum of every year. When we compute for next year we store the maximum saving of that year which will be used later on, and will make every sub-problem solvable in constant time. Now, we can return the $\max_{1\leq i\leq n} S(i, m)$ which will yield the maximum profit.

**Proof of Correctness**

**Base Case** : Susan has plan 1 for the first year and 0 savings. Since, she has zero savings she can’t switch to any plan for the first year hence everything else will be $-\infty$

**Inductive Hypothesis** : Let us assume we have correctly calculated the sub problems till $S(p, y-1)$

**Inductive steps** : From the recurrence relation we can see that we are taking the max of $\max_{1\leq i\leq n} S(i, y-1) - c_p$ and $S(p, y-1) + p_i$. The first part will cover the cases when Susan changes to plan $p$ in the year $y$ in which case we take maximum savings of last year and pick plan $p$ for the current year by deducting the cost of plan $p$. Now, the second part is where she already had plan $p$ in last year and she just accumulated the profit of this year which is $p_i$. Both these cases are possible, but we are trying to maximize the savings so we take the maximum of these two parts.

**Runtime Complexity** There are $m \times n$ subproblems and it will take $O(n)$ for each sub-problem if we don’t store the maximum saving for each year, but since we are saving it, each sub-problem can be solved in constant time which gives us total complexity of $O(mn)$
Question 3 (Leave Planning, 30 points). Michael has some friends who organize day long hiking trips once a month. These friends are poor at planning though and will organize the hike on a random day of the month providing no advance warning. Unfortunately, in order to get time off of work, Michael needs to inform his boss of vacation days at least a day in advance (and in particular before he knows whether or not the hike will take place). This means that Michael is forced to schedule vacation days and hope that the hike will take place on them, however he can be strategic about it, for example, by waiting until the end of the month by which point the hike will have already happened or by guaranteed to occur the next day.

Michael can use a total of $k$ vacation days over the course of the next $m$ months. What is the greatest expected number of hikes that he can arrange to attend? For the purposes of this problem, you may assume that a month is exactly 30 days long.

Solution:

Subproblem: We define the subproblem as $E(i, j, l)$, the maximum expected hiking days for every situation $(i, j, l)$, where $i$ is the number of months left (including the current one), $j$ is the number of days left in the current month, and $l$ is the number of vacation days Michael left, assuming the hiking has not happened in the current month.

Base Case: For any $(i, j, l)$ such that $i$, $j$, or $l$ equals 0, $E(i, j, l) = 0$.

Recurrence Relation:

$$E(i, j, l) = \max(E1(i, j, l), E2(i, j, l))$$

where,

$$E1(i, j, l) = \frac{1}{j}(1 + E(i - 1, 30, l - 1)) + \frac{j - 1}{j}E(i - 1, j - 1, l),$$

$$E2(i, j, l) = \frac{1}{j}(1 + E(i, 30, l - 1)) + \frac{j - 1}{j}E(i, j - 1, l)$$

Algorithm:

- For $i$ in $[0, m]$:
  - For $j$ in $[0, 30]$:
    - For $l$ in $[0, k]$:
      * calculate $E(i, j, l)$ based on the equations above
  - Return $E(m, 30, k)$

Correctness: Since from the recurrence relation the $E(i, j, l)$ with $l \leq 0$ will not affect the final result, we can set them as 0. Also, If Michael has no month or vacation day left, he can certainly make no more hiking day so the base case is correct. For any other situation, $E1$ represents the greatest expected number of hiking days if Michael uses a vacation day in the next day; it consists of two parts: the expected hiking days with and without one happens the next day. If a hiking day happens the next day, then the expected total hiking day would be $1 +$ the expected hiking days Michael meets in the following months; if not, it will equal the expected hiking days after the next day with one fewer vacation day. Similarly, $E2$ is the greatest expected number of hiking days if Michael doesn’t take a break in the next day. Assuming Micheal will choose the optimal strategy $E = \max(E1, E2)$ so the recurrence relation is correct.

Complexity: Since there are $O(30km)$ subproblems and calculating any one takes $O(1)$ time, the whole algorithm has complexity $O(km)$. 