1. Given a non-empty string s and a dictionary containing a list of unique words, design a dynamic programming algorithm to determine if s can be segmented into a space-separated sequence of one or more dictionary words. If *s* = "*algorithmdesign*" and your dictionary contains "*algorithm*" and "*design*". Your algorithm should answer Yes as *s* can be segmented as "*algorithm design*".

OPT(i) = 1, if s can be segmented into a space-separated sequence of one or more dictionary words.
0 otherwise.

Define $S_{i,k}$ as the substring of string s from the ith character to the kth character.

OPT(0) = 1.
for k = 1 to n
    OPT(k) = max(OPT(i)) where $0 < i < k$
                and $S_{i+1,k}$ is a dictionary word.
end for
Return OPT(n)

Time Complexity = $2+2+3+\cdots n$

$= \frac{n(n+1)}{2} = \Theta(n^2)$

2. Given n balloons, indexed from 0 to $n - 1$. Each balloon is painted with a number on it represented by array *nums*. You are asked to burst all the balloons. If you burst balloon *i* you will get `nums[left] * nums[i] * nums[right]` coins. Here *left* and *right* are adjacent indices of *i*. After bursting the balloon, the left and right then become adjacent. You may assume `nums[-1] = nums[n] = 1`, and they are not real, therefore, you can not burst them. Design a dynamic programming algorithm to find the maximum coins you can collect by bursting the balloons wisely. Analyze the running time of your algorithm.

OPT(l,r) = the maximum coins you can collect by bursting the balloons from l to r.

Assume balloon k is the last one in the array to burst.

if l > r:

  OPT(l,r) = 0

else:

  for any k from l to r:

    OPT(l,r) = max(OPT(l, k-1)
    + nums[l-1] × nums[k] × nums[r+1]
    + OPT(k+1, r))

  endfor

Return

Time complexity:

n^2 states for OPT(l,r), and for each (l,r) we are looking for the optimal 'k'. In the worst case, there is n operations. Thus total time is $O(n^3)$.

3. You are in Downtown of a city where all the streets are one-way streets. At any point, you may go right one block, down one block, or diagonally down and right one block. However, at each city block (i, j) you have to pay the entrance fees fee(i, j). The fees are arranged on a grid as shown below:

	0	1	2	3	⋯	n
0	$fee_{(0,0)}$	$fee_{(0,1)}$	$fee_{(0,2)}$	$fee_{(0,3)}$	⋯	$fee_{(0,n)}$
1	$fee_{(1,0)}$	$fee_{(1,1)}$	$fee_{(1,2)}$	$fee_{(1,3)}$	⋯	$fee_{(1,n)}$
2	$fee_{(2,0)}$	$fee_{(2,1)}$	$fee_{(2,2)}$	$fee_{(2,3)}$	⋯	$fee_{(2,n)}$
3	$fee_{(3,0)}$	$fee_{(3,1)}$	$fee_{(3,2)}$	$fee_{(3,3)}$	⋯	$fee_{(3,n)}$
⋮	⋮	⋮	⋮	⋮	⋱	⋮
n	$fee_{(n,0)}$	$fee_{(n,1)}$	$fee_{(n,2)}$	$fee_{(n,3)}$	⋯	$fee_{(n,n)}$

Your objective is to travel from the starting point at the city's entrance, located at block (0,0), to a specific destination block (n,n). The city is laid out in a grid, and at each intersection or block (i, j), you might either incur a cost (pay an entrance fee) or receive a reward (get a payback) for passing through. These transactions are captured in a grid, with positive values representing fees and negative values representing paybacks.

You would like to get to your destination with the least possible cost. Formulate the solution to this problem using dynamic programming.

a) Define (in plain English) subproblems to be solved.

b) Write the recurrence relation for subproblems.

(a) $OPT(i,j)$ = the minimum cost to reach a particular block (i,j) from the starting block $(0,0)$.

(b) For base case:

$OPT(0,0) = fee(0,0)$

Recurrence relation:

$$OPT(i,j) = fee(i,j) + \min(OPT(i-1,j), OPT(i,j-1), OPT(i-1,j-1))$$

```
dp[n][n] = fee[n][n]
for i from n down to 0:
  for j from n down to 0:
    if i<n:
      dp[i][j] = min(dp[i][j], dp[i+1][j]+fee[i][j])
    if j<n:
      dp[i][j] = min(dp[i][j], dp[i][j+1]+fee[i][j]).
    if i<n and j<n:
      dp[i][j] = min(dp[i][j], dp[i+1][j+1]+fee[i][j])
Return dp[0][0]
```

Time complexity is $O(n^2)$

4. Assume we have N workers. Each worker is assigned to work at one of M factories. For each of the M factories, they will produce a different profit depending on how many workers are assigned to that factory. We will denote the profits of factory i with j workers by P(i,j). Develop a dynamic programming solution to find the assignment of workers to factories that produce the maximum profit.(Mention the pseudocode).

DP[i,j] = the maximum profit obtained when allocating j workers to the first i factories.

Let dp = new table of size (M+1)×(N+1) filled with 0.

```
for i from 1 to M:
    for j from 1 to N:
        for k from 0 to j:
            dp[i][j] = max(dp[i][j], dp[i-1][j-k]+P(i,k))
return dp[m][n]
```

Time complexity: $O(n^2 m)$

5. You have two rooms to rent out. There are n customers interested in renting the rooms. The ith customer wishes to rent one room (either room you have) for d[i] days and is willing to pay bid[i] for the entire stay. Customer requests are non-negotiable in that they would not be willing to rent for a shorter or longer duration. Devise a dynamic programming algorithm to determine the maximum profit that you can make from the customers over a period of D days.
a) Define (in plain English) subproblems to be solved.
b) Write the recurrence relation for subproblems.

(a) $OPT(i, j, k)$ = the maximum profit achievable by considering the first i customers, ending on day j, and have renting out k rooms. ($k = 0, 1$ or 2)

(b) Recurrence Relation:

$$OPT(i,j,k) = \begin{cases} OPT(i-1, j, k), & \text{if not renting to customer } i. \\ \max(OPT(i,j,k), OPT(i-1, j-d[i], k-1) + bid[i]), & \text{if renting to customer } i \text{ is feasible} \end{cases}$$

The maximum profit achieved after considering all customers, ending on day D, and renting out 2 rooms: $OPT(n, D, 2)$

---

create $dp[n+1][D+1][3]$ initialized to 0

For i from 1 to n:
  For j from 0 to D:
    For k from 0 to 2:
      $dp[i][j][k] = dp[i-1][j][k]$
      If $j \geq d[i]$ and $K > 0$:
        $dp[i][j][k] = \max(dp[i][j][k], dp[i-1][j-d[i]][k-1] + bid[i])$

Return $\max(dp[n][D][0], dp[n][D][1], dp[n][D][2])$

Time complexity: $O(nD^2)$

6. You are given a sequence of n numbers (positive or negative): $x_1, x_2, \ldots, x_n$. Your job is to select a subset of these numbers of maximum total sum, subject to the constraint that you can't select two elements that are adjacent (that is, if you pick x_i then you cannot pick either x_{i-1} or x_i). On the boundaries, if x_1 is chosen, x_2 cannot be chosen; if x_n is chosen, then x_{n-1} cannot be chosen. Give a dynamic programming solution to find, in time polynomial in n, the subset of maximum total sum.

OPT(i): the maximum subsequence sum obtained by considering the first i elements without selecting adjacent elements.

Recurrence Relation:

OPT(i) = max (OPT(i-1), Not selecting current x_i,
OPT(i-2) + x_i) selecting the current x_i.

create dp[n+1] initialized to 0

if n ≥ 0:

dp[1] = max(0, x_1)

for i from 2 to n:

dp[i] = max(dp[i-1], x_i + dp[i-2])

Return dp[n]

The time complexity is O(n).

7. Suppose you have a rod of length N, and you want to cut up the rod and sell the pieces in a way that maximizes the total amount of money you get. A piece of length i is worth p_i dollars. Devise a Dynamic Programming algorithm to determine the maximum amount of money you can get by cutting the rod strategically and selling the cut pieces.

dp[j] is the maximum value of wood for each length from 0 to N.

prices array indices to wood block lengths and values to their respective prices.

```
let dp = array of length N+1, fill with 0
for j from 1 to N:
    for i from 1 to j:
        dp[j] = max(dp[j], prices[i] + dp[j-i])
return dp[N]
```

The time complexity is $O(n^2)$