Faculty of Computer &
Information Sciences
Ain Shams University

Algorithms Design and Analysis
DP Sheet

3rd Year — 2nd Semester - 2024

Table of Contents

FIRST: DESIGN PROBLEMS [with solution]

QUESTION1: Probability of Failure

A student is currently taking three courses. It is important that he does not fail all of them. If the probability of failing Graphics is P1, the probability of failing English is P2, and the probability of failing Algorithms is P3, then the probability of failing all of them is P1P2P3. He has left himself with four hours to study. How should he minimize his probability of failing all his courses? (use dynamic programming)

The following gives the probability of failing each course given he studies for a certain number of hours on that subject, as shown in the following table

Table: Student failure probabilities.

Hours	Graphics	English	Algorithms
0	0.8	0.75	0.9
1	0.7	0.7	0.7
2	0.65	0.67	0.6
3	0.62	0.65	0.55
4	0.6	0.62	0.5

Answer:

Probability Table → **P (H, N)**:-

<u>Where **H** means *Number of hours* and **N** means *Subject Number:*</u>

Hours/Subj. No	1	2	3
0	0.8	0.75	0.9
1	0.7	0.7	0.7
2	0.65	0.67	0.6
3	0.62	0.65	0.55
4	0.6	0.62	0.5

I.e. P (2, 1) = 0.65

P (0, 3) = 0.9

Minimized Failure Table (*Dynamic Programming*) → M (H, S):-

Where **H** means *Number of hours* and **S** means Number of *Subjects to study.*

We are going to check the probability of failure when studying certain number of subjects respectively in a time limit.

I.e. if we want to know the minimum probability of failure when studying the 3 subjects in 4 hours we use → ***M (4, 3)***

Means: Probability to study 2 subjects in 1 hours (i.e. subject 1 & 2)

Means: Probability to study one subject in 0 hours (i.e. subject 1)

Hours\No. of subject To study	1	2	3
0	P (0,1)	P (0,2) * M (0,1)	P (0,3) * M (0,2)
1	P (1,1)	Min: P (0,2) * M (1,1) P (1,2) * M (0,1)	Min: P (0,3) * M (1,2) P (1,3) * M (0,2)
2	P (2,1)	Min: P (0,2) * M (2,1) P (1,2) * M (1,1) P (2,2) * M (0,1)	Min: P (0,3) * M (2,2) P (1,3) * M (1,2) P (2,3) * M (0,2)
3	P (3,1)	Min: P (0,2) * M (3,1) P (1,2) * M (2,1) P (2,2) * M (1,1) P (3,2) * M (0,1)	Min: P (0,3) * M (3,2) P (1,3) * M (2,2) P (2,3) * M (1,2) P (3,3) * M (0,2)
4	P (4,1)	Min: P (0,2) * M (4,1) P (1,2) * M (3,1) P (2,2) * M (2,1) P (3,2) * M (1,1) P (4,2) * M (0,1)	Min: P (0,3) * M (4,2) P (1,3) * M (3,2) P (2,3) * M (2,2) P (3,3) * M (1,2) P (4,3) * M (0,2)

Notice every time Hours increment, we get H + 1 equations.

Means: Probability to study 3 subjects in 4 hours (i.e. subject 1, 2 & 3)

Explanation

Let’s consider *M (2, 3)*. We want to get the minimum probability of failure when studying 3 subjects in 2 hours. So we got 3 cases:-

1- Don’t study subject 3 and study other 2 subjects in 2 hours.
2- Study subject 3 for 1 hour then study 2 subjects in 1 hour.
3- Study subject 3 for 2 hours and don’t study other 2 subjects.

We get probability of the 3 cases then choose the minimum.

How to (*Dynamic Programming*):-

$$M\ (H, N) \begin{cases} For\ (I = 1 \rightarrow N) \quad T *= P\ (0, I) & If\ H = 0 \\ P\ (H, 1) & If\ N = 1 \\ For\ (I = 0 \rightarrow H) \quad T = P\ (I, N) * M\ (H-I, N-1) & If\ H > 0 \end{cases}$$

Minimum
T ( )

Complexity

- Creating the **Minimized Failure Table** takes H * N times
- Filling each cell takes H + 1 times
- Therefore total complexity → O (H*N * H) → O (H^2 * N)

Filling Values

Hours\No. of subject To study	1	2	3
0	0.8	Min: .75 * .8 = .6	Min: .9 * .6 = .54
1	0.7	Min: .75 * .7 = .525 ← .7 * .8 = .56	Min: .9 * .525 = .4725 .7 * .6 = .42 ←
2	0.65	Min: .75 * .65 = .4875 ← .7 * .7 = .49 .67 * .8 = .536	Min: .9 * .4875 = .4388 .7 * .525 = .3675 .6 * .6 = .36 ←
3	0.62	Min: .75 * .62 = .465 .7 * .65 = .455 ← .67 * .7 = .469 .65 * .8 = .52	Min: .9 * .465 = .4185 .7 * .4875 = .3413 .6 * .525 = .315 ← .55 * .6 = .33
4	0.6	Min: .75 * .6 = .45 .7 * .62 = .434 ← .67 * .65 = .4355 .65 * .7 = .455 .62 * .8 = .496	Min: .9 * .434 = .3906 .7 * .465 = .3255 .6 * .4875 = .2925 .55 * .525 = .2888 ← .5 * .6 = .3

Minimum Probability of Failure = **0.2888**

What Subjects to study and for how long??

We need to backtrack through the **Minimized Failure Table** and check for the minimum probability we have chosen earlier.

I.e. Last element we got → **P (3, 3)** * M (1, 2)

Then when we go to M (1, 2) we find → **P (0, 2)** * M (1, 1)

Then when we go to M (1, 1) we find → **P (1, 1)**

*Therefore to minimize the probability of failure that student should Study **Subject 3 for 3 hours, subject 1 for 1 hour** and **don't study subject 2** at all!!*

N.B. To achieve this we could create another table while filling the **Minimized Failure Table.**

That table would contain the minimum item we have chosen.

Complexity then would be O (N) for the process of choosing subjects and study time of each.

QUESTION2: Coin Change

Suppose A ={$a_1,a_2,a_3,\ldots.a_k$} is a set of distinct coin values (all the a_i are positive integers) available in a particular country. The coin changing problem is defined as follows: Given integer n find the minimum number of coins from A that add up to n assuming that all coins have value in A. You should assume that a_1=1 so that it is always possible to find some set that adds up to n.Design a O(nk) dynamic programming algorithm for solving the coin changing problem, that is, given inputs A and n it outputs the minimum number of coins in A to add up to n.(It is not necessary to say what the coins are)

Answer

- Let's consider "A" array that consists of the coin values >= 1:-

10	2	5	1

- Let "C [P]" be an array that contains *minimum number* of coins of dominations needed to make change for "P" cents.
- Let "S [P]" be an array that contains index of the optimal coin (In array "A") needed to make change for "P" cents.
- By using the following recursive formula we guarantee that we calculate the *minimum number* of coins of dominations needed to make change for "P" cents:-

$$C[P] = \begin{cases} 0 & \text{If } p = 0 \\ \min_{i:d_i<=p}\{1 + C[p - d_i]\} & \text{If } p > 0 \end{cases}$$

❖ Use this algorithm to fill in the two arrays:-

```
Change (d, k, n)
        C[0] ← 0
        for p ← 1 to n
                min ← ∞
                for i ← 1 to k
                        if A[i] <= p then
                                if 1 + C[p – d[i]] < min then
                                min ← 1 + C[p – d[i]]
                                coin ← i
                C[p] ← min
                S[p] ← coin
        return C and S
```

$\boxed{O(n * k)}$

❖ Testing with i.e. **Change(A, 4, 8)**
❖ Filling Arrays:-

	0	1	2	3	4	5	6	7	8
C	0	1	1	2	2	1	2	2	3
S	0	4	2	2	2	3	3	3	2

A	0	10	2	5	1

❖ Use this algorithm to extract minimum coins needed for the change (*Which is not actually needed in the question*):-

```
Make-Change (S, A, n)
        while n > 0
                Print A[S[n]]
                        n ← n – A[S[n]]
```

$\boxed{O(n)}$

❖ Output of **Make-Change(S,A, 8)**:-

n = 8
>> 2
n = 6
>> 5
n = 1
>> 1

QUESTION3: Knapsack Variation

(a) Write the recurrence relation and solve the 0-1 knapsack problem given that you have four items having weights of 2,3,4,5 and values of 3,4,5,6 respectively. The total weight capacity equals 5, what is the complexity of your algorithm? Is it polynomial in input size?

(b) Suppose you are given a set of n objects. The weights of the objects are w1, w2, . . . ,wn, and the values of the objects are v1, v2, . . . , vn. You are given two knapsacks each of weight capacity C. Write a recurrence relation to determine the maximum value of objects that can be placed into the two knapsacks. What will be the complexity in this case?

Example:

N = 4, C = 20

Item	Weight	Profit
1	10	20
2	5	15
3	10	10
4	15	10

Max Profit = 55 (**KS1:** item1 + item3, **KS2:** item2 + item4)

Answer

(a) Recurrence Relation:-

$$\text{Knap}(W, N) = \begin{cases} 0 & \text{If } N = 0 \quad \textit{No Items or space} \\ \text{Knap}(W, N-1) & \text{If } w[N] > W \quad \textit{Leave it} \\ \text{Max}\{\text{Knap}(W, N-1), \text{Knap}(W - w[N], N-1) + b[N]\} & \end{cases}$$

0-1 Knapsack Solution:-

Weight\Item	0	$\textcircled{1}$	$\textcircled{2}$	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	$\textcircled{3}$ ↖	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	$\textcircled{7}$ ←	7 ←	7

_-_-Items-_-_

(Weight, Value)

1- (2, 3)
2- (3, 4)
3- (4, 5)
4- (5, 6)

So we take items 1 and 2, to have maximum total value of 7

Complexity:-

O (n * W) →where "n" is the number of items

Polynomial in input size?? ← **I** suppose *yes* ^o) ??

(b) Recurrence Relation:-

$$\text{Knap}(W_1, W_2, N) = \begin{cases} 0 & \text{If } N = 0 \quad \textit{No Items or space} \\ \text{Knap}(W_1, W_2, N-1) & \text{If } w[N] > W_1 \;\&\; w[N] > W_2 \quad \textit{Leave it} \\ \text{Max}\{\text{Knap}(W_1, W_2, N-1),\ \text{Knap}(W_1 - w[N], W_2, N-1) + b[N],\ \text{Knap}(W_1, W_2 - w[N], N-1) + b[N]\} & \end{cases}$$

Complexity:

O (n * W_1 * W_2) →where "n" is the number of items

QUESTION4: Exact Sum

You are given a sequence of positive integers $a_1; a_2; :::; a_n$, and a positive integer B. Your goal is to determine if some subsequence of $a_1; a_2; :::; a_n$ sums to exactly B.

Answer

D&C Pseudo Code

```
Check (A, N, B):-
     If A [N] == B Then
          Return true
     End If
     If N == 0 ~~OR B == 0~~ Then
          Return false
     End If
     If A [N] < B Then
          B1 = check (A, N - 1, B - A [N])
          B2 = check (A, N - 1, B)
          Return (B1 OR B2)
     Else
          Return check (A, N - 1, B)
     End If
```

D&C Complexity:-

This algorithm uses recursion, so find its recurrence and solve it:

$T(N) = 2 \times T(N - 1) + \Theta(1)$

Solve by iteration method ➔ $\Theta(2^N)$

Side Note: Example on How Overlapping Can Occur

```
A = {1, 5, 10, 20, 30, 50, 100}, B = 80
```

We can reach to a common sub-problem: "**`exactlySum(3, 30)`**", which try to find combination that sum to 30 using first 3 numbers, by two ways:

1. Use 50 and bypass 30 & 20
2. Bypass 50 and use 30 & 20

Both problems need to solve **`exactlySum(3, 30)`**

Dynamic Programming Solution:-

Extra storage: 2D Boolean array of size N × B,

```
For I = 0 to N
      For J = 0 to B
            If A [I] == J Then
                  Check[I, J] = true
            Else if I == 0 Then
                  Check[I, J] = false
            Else if A [I] < J Then
                  B1 = Check[I - 1, J - A[I]]
                  B2 = Check[I - 1, J]
                  Check[I, J] = (B1 OR B2)
            Else
                  Check[I, J] = Check[I - 1, J]
            End If
      End For
End For
```

DP Complexity: $\Theta(N \times B)$

QUESTION5: N Stairs

Suppose we are walking up n stairs. At each step, you can go up 1 stair, 2 stairs or 3 stairs. Our goal is to compute how many different ways there are to get to the top (level n) starting at level 0. We can't go down and we want to stop exactly at level n.

1. Design an efficient algorithm to solve this problem?

2. What's the complexity of your algorithm?

Answer

Design	Answer
choices (trials) per sub/problem? Subproblem of each?	1. go up 1 stair: W(N – 1) 2. go up 2 stairs: W(N – 2) 3. go up 3 stairs: W(N – 3)
most suitable solution direction? Why?	1. Top-down 2. Bottom-up Reason: all subproblems are calculated (get rid of recursion)
In case of **bottom-up**, size of the extra storage?	Size = O(N)
equation to fill this extra storage (including the base case(s))?	$W[i] = \begin{cases} 1 & if\ i = 0 \\ i & if\ 0 < i \leq 2 \\ W[i-1] + W[i-2] + W[i-3] & else \end{cases}$
Complexity?	Θ(N)

QUESTION6: Restaurant Placement II

Smiley Donuts, a new donuts restaurant chain, wants to build restaurants on many street corners with the goal of maximizing their total profit.

The street network is described as an undirected graph *G* = (*V*, *E*), where the potential restaurant sites are the vertices of the graph. Each vertex *u* has a nonnegative integer value p_u, which describes the potential ***profit*** of site *u*. Two restaurants cannot be built on adjacent vertices (to avoid self-competition).

You are supposed to design an algorithm that outputs the chosen set $U \subseteq V$ of sites that maximizes the total profit $\sum_{u \in U} p_u$. Suppose that the street network *G* is acyclic, i.e., a tree. Give an efficient algorithm to determine a placement with maximum profit?

Answer:

1. pick any node u_0 as the root of the tree and sort all the nodes following a depth-first search (DFS). Store the sorted nodes in array N. Due to the definition of DFS, a parent node appears earlier than all of its children in N.
2. For each node v, define:
   a. A(v), the best cost of a placement in the subtree rooted at v if v is included,
   b. B(v), the best cost of a placement in the subtree rooted at v if v is not included.
3. The following recursion equations can be developed for A() and B():

a. If *v* is a leaf, then $A(v) = p_v$ and $B(v) = 0$.
b. If *v* is not a leaf, then:

$$A(v) = p_v + \sum_{u \in v.children} B(u)$$

$$B(v) = \sum_{u \in v.children} \max(A(u), B(u))$$

4. For each node v in N, in the reverse order, compute A(v) and B(v). Finally the max($A(u_0)$, $B(u_0)$) is the maximum profit.

The placement achieving this maximum profit can be derived by recursively comparing A() and B() starting from the root. The root u0 should be included if $A(u_0) > B(u_0)$ and excluded otherwise. If u_0 is excluded, we go to all of u_0's children and repeat the step; if u_0 is included, we go to all of u_0's grandchildren and repeat the step. This algorithm outputs an optimal placement by making one pass of the tree.

Analysis

Sorting all nodes using DFS takes $O(n)$ time. The time to compute $A(v)$ and $B(v)$, given the values for the children, is proportional to *degree*(v). So the total time is the sum of all the degrees of all the nodes, which is $O(|E|)$ where $|E|$ is the total number of edges. For a tree structure, $|E| = n - 1$ so the time for finding all *A*() and *B*() is $O(n)$. Finally, using the derived *A*() and *B*() to find the optimal placement visits each node once and thus is $O(n)$. Overall, the algorithm has $O(n)$ complexity.

SECOND: DESIGN PROBLEMS [without solution]

QUESTION1: longest oscillating subsequence

A sequence of numbers $a_1, a_2, a_3, \ldots, a_n$ is oscillating if $a_i < a_{i+1}$ for every odd index i and $a_i > a_{i+1}$ for every even index i. For example, the sequence 2, 7, 1, 8, 2, 8, 1, 8, 3 is oscillating. Describe and analyze an efficient algorithm to find a longest oscillating subsequence in a sequence of n integers. Your algorithm only needs to output the length of the oscillating subsequence. For example if the input sequence is 2, 4, 5, 1, 4, 2, 1, your algorithm should output 5, corresponding to the subsequence 2, 4, 1, 4, 1, or 2, 4, 1, 4, 2, or any other such subsequence. For full credits, your algorithm should run in $O(n2)$ time.

QUESTION2: Super Thief

You are an amazing super thief. You have scoped out a circle of ***n***, houses that you would like to rob tonight. However, as the super thief that you are, you've done your homework and realized that you want to **avoid robbing two adjacent houses** because this will minimize the chance that you get caught. You've also scouted out the hood, so for every house ***i***, you know that the net worth you will gain from robbing that house is $\boldsymbol{v_i}$. Thus, You want to try to **maximize the amount of money** that you can rob.

1. Explain why this problem is a good candidate for a DP algorithm?
2. Give a DP algorithm to find the maximum amount of money that you can steal given that you never rob two houses that are adjacent? state whether your are using the bottom up approach or the top down approach?
3. Modify your algorithm so that it also returns the best set of houses to rob instead of the maximum value you can steal?

QUESTION3: Subset Sum

In computer science, the **subset sum problem** is an important problem in complexity theory and cryptography. Given a set (or multiset) of integers, is there a non-empty subset whose sum is zero?

For example, given the set {−7, −3, −2, 5, 8}, the answer is *yes* because the subset {−3, −2, 5} sums to zero.

1. Write a recurrence relation?
2. Write a pseudocode of a DP solution for this problem?
3. What will be the complexity of the solution?

QUESTION4: Partitioning Problem

Given an array of positive integers, decide **whether or not** it can be partitioned into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2

1. Write a recurrence relation?
2. Write a pseudocode of a DP solution for this problem?

3. What will be the complexity of the solution?

QUESTION5: String Alignment

Two strings are aligned together if we can change one of them to the other. Moreover, same character may be appeared several consecutive times in one or both strings with different number of repetition, as shown below:

S1	A	L	L	L	G	O	O	R	I	T	H	M

S2	A	L	G	O	O	O	R	I	T	H	M

In this problem, it is allowed ONLY to change the 2nd string to be aligned with the 1st one.

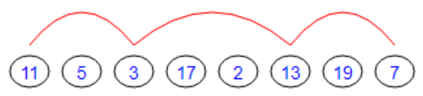
Three possible changes are allowed:

1. Replace a character from the 2nd string by the one from the 1st string
2. Repeat one character in 2nd string (only if it's matched with the one from 1st string). Allowing it to be stretched. (e.g. **red** case in the above example)
3. Skip one character from 2nd string (only if it's matched with the one from 1st string). Allowing it to be shrunk. (e.g. **green** case in the above example)

Thus, given two strings, s1 and s2, it's required to find the min number of changes that should be applied to s2 in order to align it with s1?

QUESTION6: Rabbit Crossing a River

The Rabbit is going to cross the river. There is a straight chain of tiny isles across the flow and the animal should jump from one to another. At each of the isles there are some candies. When the Rabbit arrives to the new isle, it collects all the candies here.
However, the Rabbit could not jump directly to the next isle in the chain - it just is too strong to make short jumps. So, instead, it can jump over the one or two isles (i.e. from the 1st for example to 3rd or 4th but not to 2nd or 5th and further). Also the Rabbit could not jump back.



The amount of 34 candies. Obviously he could do better if the path is chosen more wisely.
Your task is to choose the best path for Rabbit over the given chain of isles - i.e. to maximize the amount of the candies collected. Note that Rabbit starts from 1st isle and finishes either on the Nth or (N-1)th where N is the total number of isles (because from these two it will necessarily jump immediately to the other bank).

Examples

Input	Output
11 5 3 17 2 13 19 7	48
9 7 12 7 16 3 7 17 14 13 4 6 11 6 3 3 5 4 11 3 15 12 14 2 15 19 11 12	157

QUESTION7: Ali Baba in a Cave

Ali Baba and his thieves enter a cave that contains a set of *n* **types of items** from which they are to select some number of items to be theft. Each item type has a *weight*, a *profit* and a *number of available instances*. Their objective is to choose the set of items that fits the possible load of their Camels and maximizes the profit. Note the following:

1- Each item should be **taken as a whole** (i.e. they can't take part of it)

2- They can take the same item **one or more time** (according to its number of instances).

Given ***n*** **triples** where each triple represents the (**weight, profit, number of instances**) of an **item type** and the **Camels possible load**, find maximum profit that can be loaded on the Camels by the **OPTIMAL WAY**.

Complexity

Your algorithm should take **polynomial time**

Example

N = 4 Load = 10

Weight	Profit	# instances
2	1	2
4	8	2
3	6	2
4	5	2

Max Profit = 20$, as follows:

1. one instance from item2 (profit 8$)

2. two instances from item3 (profit 6$ × 2 = 12$)

THIRD: ONLINE PROBLEMS

1. Diving for Gold @ UVa

2. Vacations @codeforces

3. Dividing coins @ UVa

FOURTH: OTHER QUESTIONS

QUESTION1: 0-1 Knapsack Tracing

In a 0-1 Knapsack problem, it's required to fill a knapsack of weight **W** with the maximum total cost from **N** items. Each item has only one instance with weight w[i] and cost c[i]. Following is an example:

Knapsack weight (W) = 5; Items (weight, cost): (1,2), (2,3), (3, 4), (4, 5).

Answer: maximum total cost is 7. This is achieved by taking 2nd and 3rd item.

Given a knapsack of weight **W = 3**, and the following **4 items**, **fill-up** the DP solution table and **answer** the questions:

item index (i)	**1**	**2**	**3**	**4**
weight (w[i])	1	2	4	1
cost (c[i])	30	20	80	30

1. What's the final total cost in this case? What's the remaining weight in the knapsack (if any)?
2. From the filled DP table, find the optimal total cost in each of the following cases:

(knapsack weight, items count)	**(0,0)**	**(0,3)**	**(3,0)**	**(2,1)**	**(3,2)**	**(3,3)**	**(2,4)**
Optimal total cost	0						

Answer

1. 60, 1
2.

(knapsack weight, items count)	**(0,0)**	**(0,3)**	**(3,0)**	**(2,1)**	**(3,2)**	**(3,3)**	**(2,4)**
Optimal total cost	0	0	0	30	50	50	60

QUESTION2: Rod-Cut Tracing

In a rod-cutting problem, it's required to cut a steel rod into piece(s) to maximize the total revenue. Thus, given a steel rod of length N and a price table containing the price of each length; it's required to find the optimal cuts that result in maximum total revenue. Following is an example:

Rod length (N) = 4; **Price table (length, price):** (1,1), (2,5), (3, 8), (4, 9).

Answer: Optimal cut is two pieces of length two. This results total revenue of 5 + 5 = 10

Given the following price table:

Length	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Price	3	5	8	10	16	17	19	24	29	30

Find the optimal solutions (i.e. max revenues) to cut a rod of each of the following lengths: 3, 4, 5, 6, 7, 8, 9, 10? (i.e. Fill up the following dynamic-programming solution table)

Length	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Optimal Revenue	0	3	6								

Answer

Length	0	1	2	3	4	5	6	7	8	9	10
Optimal Revenue	0	3	6	9	12	16	19	22	25	29	32

QUESTION3: Convert D&C to DP (bottom-up)

1. Convert the following D&C solution to a **DP with building table (bottom-up)**?
2. What's the complexity of the DP solution?

```
FUN(I)
   if I = N + 1 return 0
   m = ∞
   for J = I to N
        m = Min(m, AUX(I,J) + FUN(J+1))
   return m

main()
     return FUN(1)
```

`AUX(i,j)`: **is an auxiliary function that calculate a specific value in O(1)**

Answer

```
FUN(N)

     for I = N + 1 downto 1

          if (I = N + 1) R[I] = 0
          else
               m = ∞
               for J = I to N
                    m = Min(m, AUX(I,J) + R[J+1])
               R[I] = m
return R[1]
```

$\theta(N^2)$

QUESTION4: Convert D&C to DP (top-down)

1. Convert the following D&C solution to a **DP with memoization (top-down)**?
2. What's the complexity of the DP solution?

```
//V is an input 1D array of size N (1-based)
ST(N)
    if (N = 0)
        return 0
    if (N = 1)
        return V[1]
   return max(V[N] + ST(N-2), ST(N-1))
```

Answer

```
ST(N)
    If (R[N] ≠ NiL)      return R[N]
    if (N = 0)
        return R[N] = 0
    if (N = 1)
        return R[N] = V[1]

return R[N] = max(V[N]+ST(N-2), ST(N-1))
```

$\theta(N)$

QUESTION5: Convert D&C to DP (bottom-up)

3. Convert the following D&C solution to a **DP with building table (bottom-up)**?
4. What's the complexity of the DP solution?

```
M(H, N)      //P is an input 2D array of size H×N
    if (H = 0)
        T = 1
        for i = 1 to N
            T = T × P[0, i]
        return T
    if (N = 1)
        return P[H, 1]
    T = ∞
    for i = 0 to H
        T = min(T, P[i, N] × M(H − i, N − 1))
    return T
```

```
M(H,N)
 dp[MAX_H_VALUE]
[MAX_N_VALUE];
 for h = 0 to H
  for n = 0 to N
   if(h == 0)
     T = 1
     for i = 1 to N
         T *= P[0, i];
    dp[h][n] = T;
     continue;

   if(n == 1)
  dp[h][n] = P[h,1];
  continue;


 T = INT_MAX;
 for i = 0 to H
   T = min(T, P[i,n] * dp[h-i][n-i]);

 dp[h][n] = T;

 return dp[H][N];
```