Dynamic Programming

(Algorithmic Problems)

Dynamic Programming Paradigm

- **dynamic programming** is both an optimization technique and a computer programming method
- it was introduced by **Richard Bellman** in **1953**
- the main idea is that **we can break down complicated problems into smaller subproblems** in a recursive manner
- then we find the solutions for these subproblems and finally we combine the subresults to find the final solution

Dynamic Programming Paradigm

- **dynamic programming** is a method for solving a complex problem by breaking it down into a collection of simpler subproblems
- it is applicable to problems exhibiting the properties of overlapping subproblems
- dynamic programming takes far less time than other methods that don't take advantage of the subproblem overlap
- we need to solve different parts of the problem (subproblems) + combine the solutions of the subproblems to reach an overall solution
- we solve each subproblems only once - we reduce the number of computations
- subproblems can be stored in memory - **memoization** and **tabulation**

Dynamic Programming Paradigm

OPTIMAL SUBSTRUCTURE

In computer science, a problem is said to have **optimal substructure** if an **optimal** solution can be constructed from **optimal** solutions of its subproblems

BELLMAN-EQUATION

Of course there is a relationship between the subresults and the final result – this is what the **Bellman-equation** defines

Dynamic Programming Paradigm

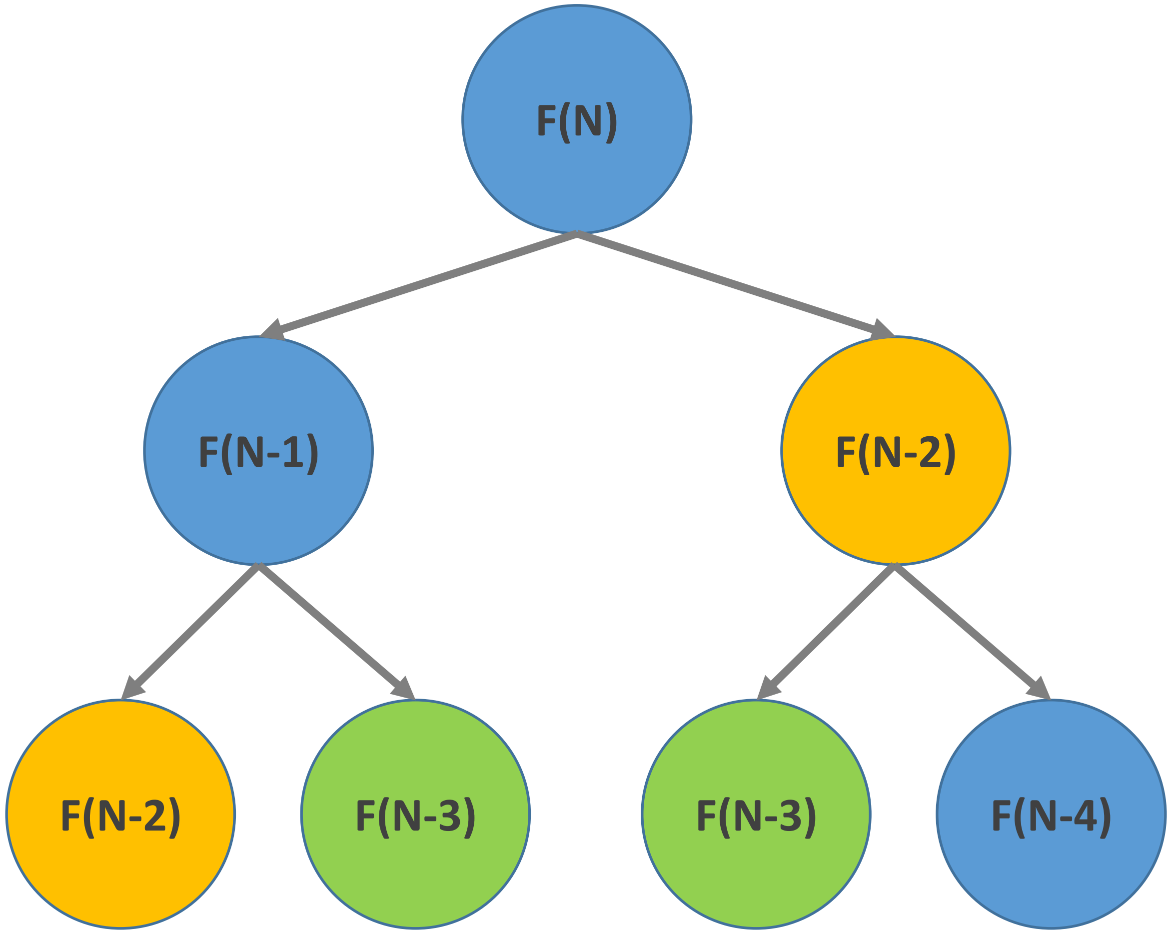
„If a given problem has optimal substructure and overlapping subproblems then we can use dynamic programming appraoch”

Dynamic Programming Paradigm

F(N) = F(N-1) + F(N-2)

the formula for calculating the ***N****-th*
Fibonacci-number *with recursion*
(note that there are several overlapping subproblems we have to solve several times)

dynamic programming *use*
memoization *or* ***tabulation***
to store these values
(so there is no need for recalculating them)



Memoization and Tabulation

The problem of recursion is that we may solve the same subproblems multiple times. This can be eliminated by:

1.) TOP-DOWN APPROACH „MEMOIZATION"

We can store the solutions of the subproblems in a table (priority queue for example)

Whenever we try to solve a new subproblem we first check whether it is present in the table (so we have already solved that problem)

Memoization and Tabulation

The problem of recursion is that we may solve the same subproblems multiple times. This can be eliminated by:

2.) BOTTOM-UP APPROACH „TABULATION”

We reformulate the original problem in a bottom-up fashion. We iteratively generate the subresults for larger and larger subproblems

Memoization and Tabulation

„Dynamic programming approach sacrifices extra memory
in exchange for faster running time – common technique in computer science"

Dynamic Programming and Divide and Conquer Approaches

- several problems can be solved by combining optimal solutions to *non-overlapping* subproblems
- this strategy is called **divide and conquer** method
- this is why merge sort (or quicksort) are not classified as dynamic programming problems
- overlapping subproblems - dynamic programming
- non-overlapping subproblems - divide and conquer method

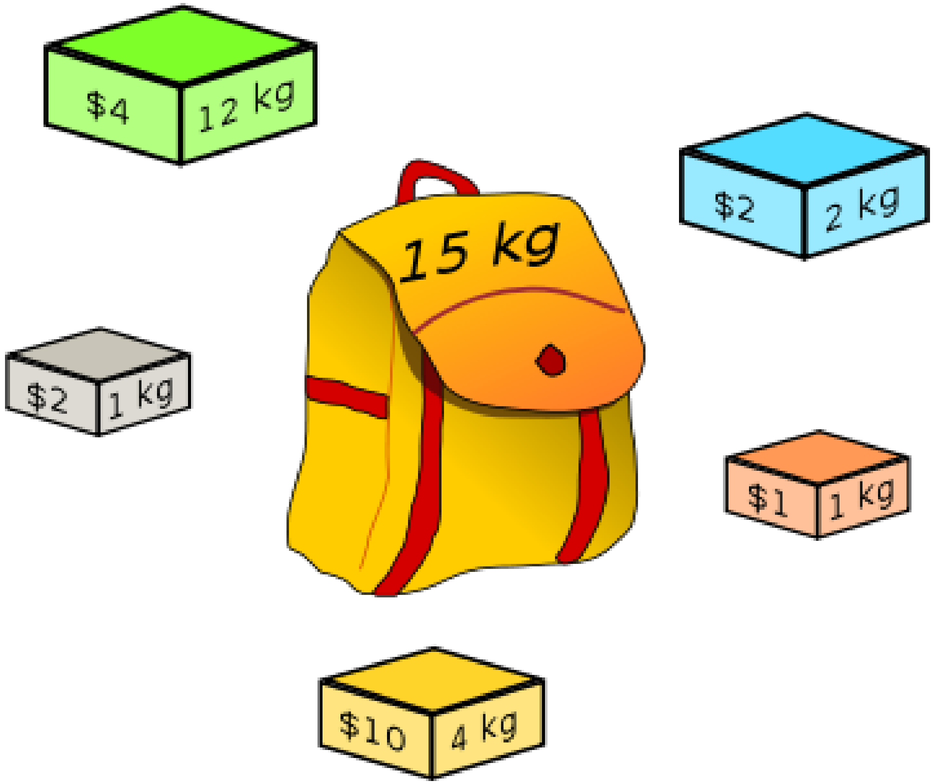
Knapsack Problem

(Algorithmic Problems)

Knapsack Problem

- it is a **combinatorial optimization** related problem
- given a set of **n** items – usually numbered from **1** to **n**
- each of these item has a mass $\mathbf{w_i}$ and a value $\mathbf{v_i}$
- determine the number of each item to include in a collection so that the total weight **M** is less than or equal to a given limit and the total value is as large as possible
- the problem often arises in resource allocation where there are financial constraints

Knapsack Problem



Knapsack Problem

- knapsack problem has several **applications** of course
- finding the least wasteful way to cut raw materials
- selection of investments and portfolios
- selection of assets for asset-backed securitization
- construction and scoring of tests in which the test-takers have a choice as to which questions they answer

Knapsack Problem

DIVISIBLE KNAPSACK PROBLEM
(we can take fractions of the given items – fast algorithm)

0-1 KNAPSACK PROBLEM
(either we take a given item or do not take – complex solution)

Divisible Knapsack Problem

- if we can take fractions of the given items then the greedy approach can be used
- sort the items according to their values - can be done in **O(NlogN)**
- start with the item that is the most valuable and take as much as possible – starting with highest $\mathbf{v_i}$ item
- than try with the next item from our sorted list – we make a linear search in **O(N)** time complexity
- overall running time is **O(NlogN) + O(N) = O(NlogN)**
- so we can solve the divisible knapsack problem quite fast

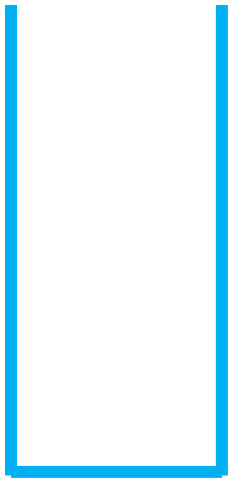
0-1 Knapsack Problem

- in this case we are not able to take fractions - we have to decide whether to take an item (**x=1**) or not (**x=0**)
- the greedy algorithm will not provide the optimal result
- another approach would be to sort by cost per unit weight and include from highest on down until knapsack is full but again not a good solution
- how many possible solutions are there with **N** items? The **brute-force approach** has **$O(2^N)$** exponential running time
- we should use **dynamic programming** instead

0-1 Knapsack Problem

- solves larger problem by relating it to overlapping subproblems and then solves the subproblems
- it works through the exponential set of solutions, but does not examine them all explicitly
- stores intermediate results so that they are not recomputed – this is called **memoization**
- solution to original problem is easily computed from the solutions to the subproblems

0-1 Knapsack Problem

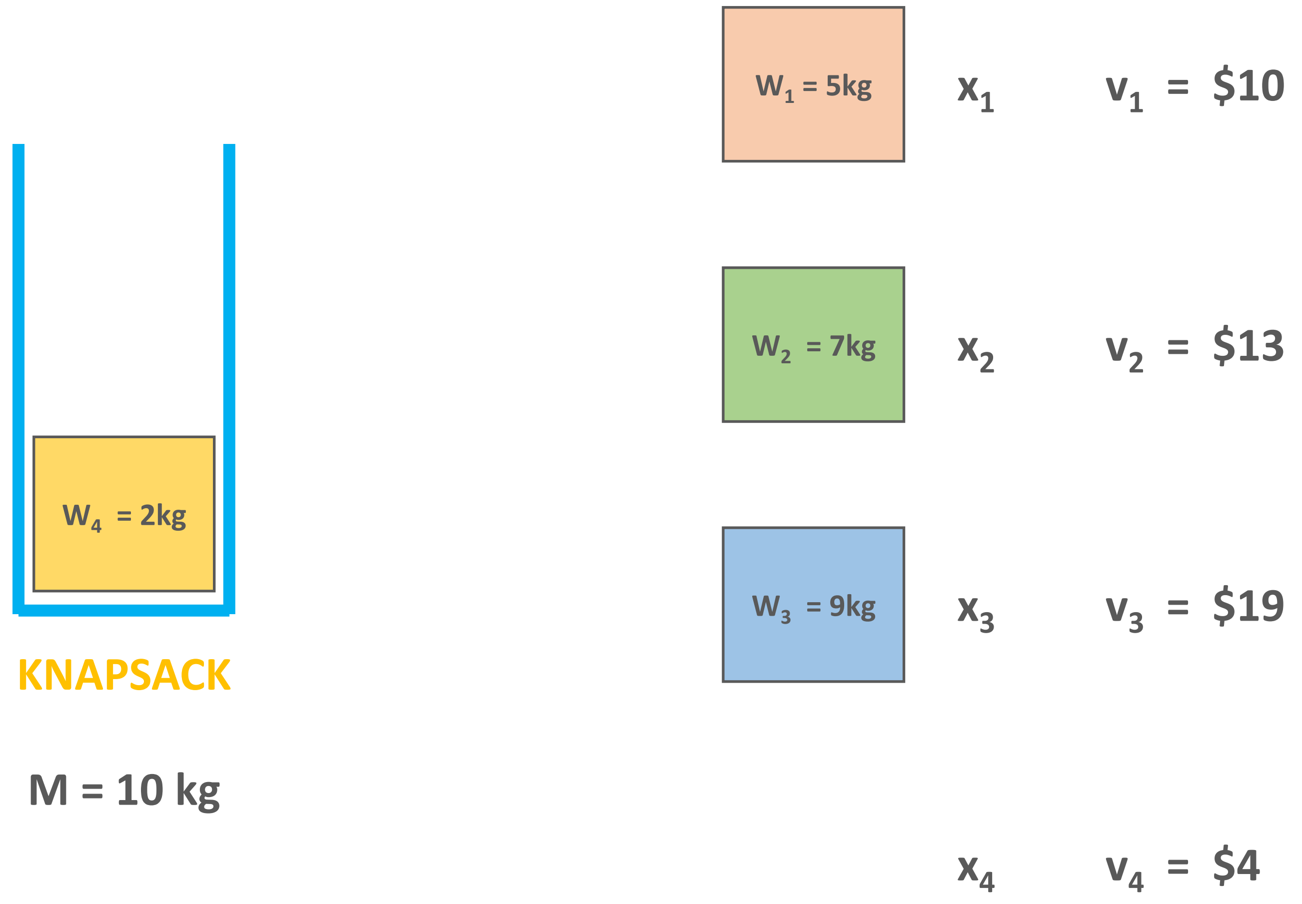


KNAPSACK

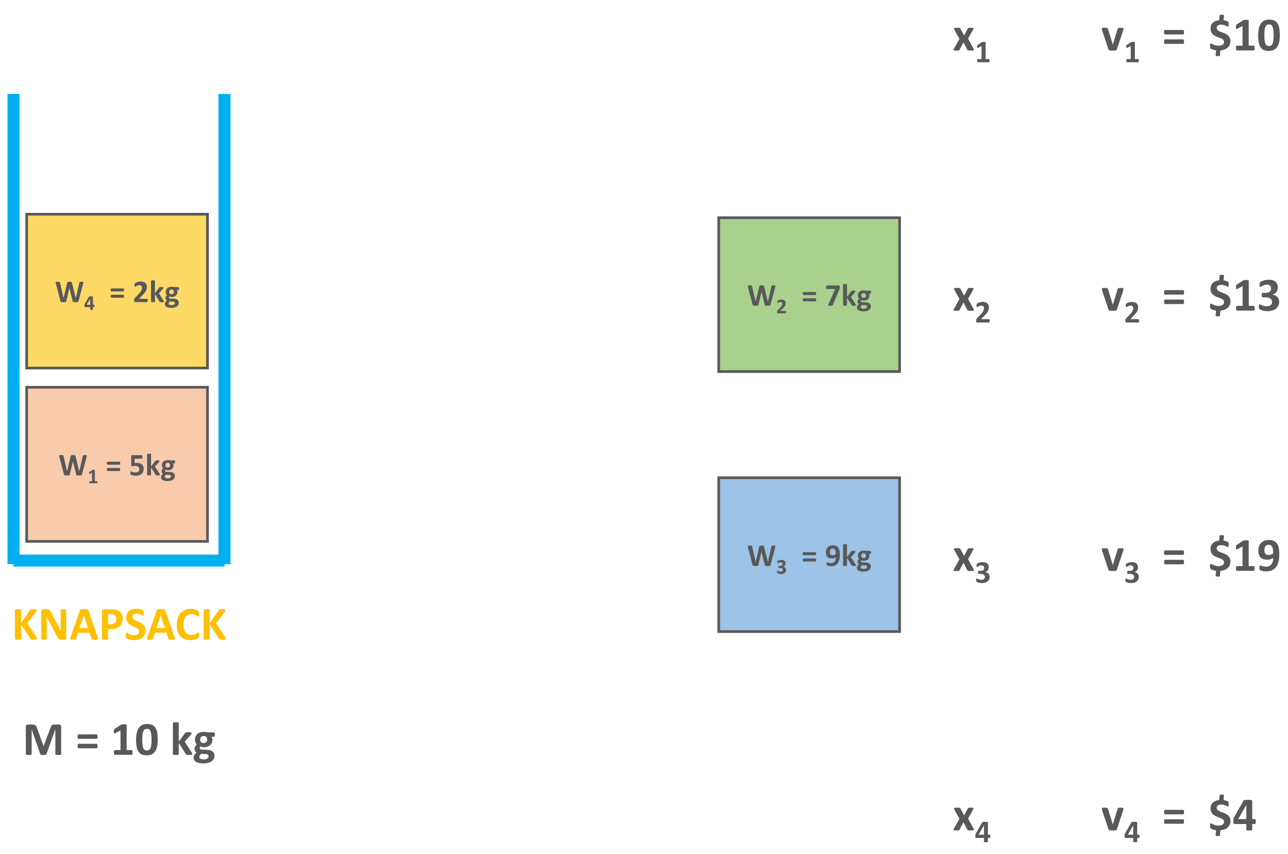
M = 10 kg

$W_1 = 5kg$	x_1	$v_1 = \$10$
$W_2 = 7kg$	x_2	$v_2 = \$13$
$W_3 = 9kg$	x_3	$v_3 = \$19$
$W_4 = 2kg$	x_4	$v_4 = \$4$

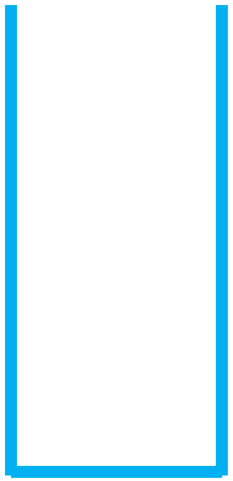
0-1 Knapsack Problem



0-1 Knapsack Problem



0-1 Knapsack Problem

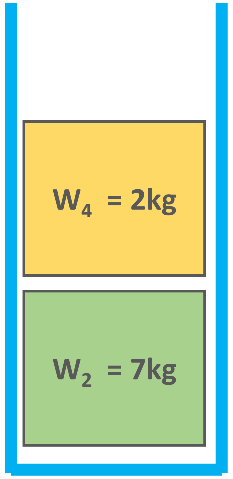


KNAPSACK

M = 10 kg

W_1 = 5kg	x_1	$v_1 = \$10$
W_2 = 7kg	x_2	$v_2 = \$13$
W_3 = 9kg	x_3	$v_3 = \$19$
W_4 = 2kg	x_4	$v_4 = \$4$

0-1 Knapsack Problem

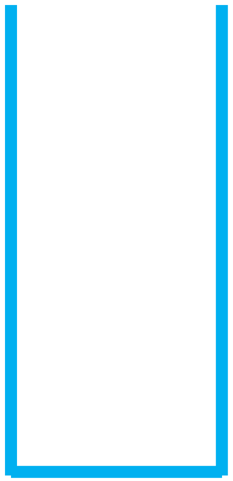


KNAPSACK

M = 10 kg

$W_1 = 5kg$	x_1	$v_1 = \$10$
	x_2	$v_2 = \$13$
$W_3 = 9kg$	x_3	$v_3 = \$19$
	x_4	$v_4 = \$4$

0-1 Knapsack Problem



KNAPSACK

M = 10 kg

W_1 = 5kg	x_1	v_1 = \$10
W_2 = 7kg	x_2	v_2 = \$13
W_3 = 9kg	x_3	v_3 = \$19
W_4 = 2kg	x_4	v_4 = \$4

0-1 Knapsack Problem

0-1 Knapsack Problem

x_i whether we take item **i** or not (**1** or **0** accordingly)

v_i value of the **i**-th item

w_i weight of the **i**-th item

M maximum capacity of knapsack

---

maximize $\sum_{i=1}^{N} v_i * x_i$ **subject to** $\sum_{i=1}^{N} w_i * x_i \leq M$

0-1 Knapsack Problem

- we have to define subproblems: we have **N** items so we have to make **N** decisions whether to take the item with given index or not
- subproblems: the solution considering every possible combination of remaining items and remaining weight
- **S[i][w]** the solution to the subproblem corresponding to the first **i** items and available weight **w**
- **S[i][w]** is the maximum cost of items that fit inside a knapsack of size (weight) **w**, choosing from the first **i** items
- we have to decide whether to take the item or not

0-1 Knapsack Problem

If we consider all subsets of the items – there may be **2** cases:

1.) the given item is **included** in the solution (optimal subset)
2.) the given item is **not included**

So the maximum value (solution) can be reduced to smaller and smaller subproblems – and these subproblems overlap

1.) the **i**-th item is not included which means that the max value is obtained by the previous **N-1** items (and M total weights)
2.) the **i**-th item is included – max value is $\mathbf{v_i}$ plus the values obtained by the previous **N-1** items (and $\mathbf{M\text{-}w_i}$ total weights)

0-1 Knapsack Problem

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

the maximum profit that fit inside a knapsack of weight **w**, choosing from the first **i** items

do not take ***i****-th item*

we take ***i****-th item*

- we have to use this **S[][]** two-dimensional array (list)
- we are only considering **S[i-1][w-w_i]** if it can fit **$w > w_i$**
- if there is not room for it: the answer is just **S[i-1][w]** !!!

0-1 Knapsack Problem

- running time of Knapsack: **O(nM)**
- but it is not polynomial – it is so-called **pseudo-polynomial**
- numeric algorithm runs in **pseudo-polynomial time** if its running time is polynomial in the *numeric value* of the input - but is exponential in the *length* of the input (so the number of bits required to represent it)

Knapsack Problem Example

(Algorithmic Problems)

0-1 Knapsack Problem

N = 3 items   **M = 5kg** capacity of knapsack

item #1	$\mathbf{w_1}$ = 4kg	$\mathbf{v_1}$ = \$10
item #2	$\mathbf{w_2}$ = 2kg	$\mathbf{v_2}$ = \$4
item #3	$\mathbf{w_3}$ = 3kg	$\mathbf{v_3}$ = \$7

		0	1	2	3	4	5
no items	**0**						
[item #1]	**1**						
[item #1, item #2]	**2**						
all items	**3**						

weights [kg]

$$S[i][w] = Math.max(S[i-1][w] \; ; \; v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**						
[item #1, item #2]	**2**						
all items	**3**						

weights [kg]

$S[i][w] = \text{Math.max}(S[i-1][w] ; v_i + S[i-1][w-w_i])$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5	weights [kg]
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
[item #1]	**1**	**0**						
[item #1, item #2]	**2**	**0**						
all items	**3**	**0**						

$$S[i][w] = Math.max(S[i-1][w]\ ;\ v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**				
[item #1, item #2]	**2**	**0**					
all items	**3**	**0**					

weights [kg]

$$S[i][w] = \text{Math.max}(S[i-1][w]\ ;\ v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	0	0	0	0	0	0	0
[item #1]	1	0	0	0			
[item #1, item #2]	2	0					
all items	3	0					

weights [kg]

$$S[i][w] = Math.max(S[i-1][w] \; ; \; v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = $10
item #2	w_2 = 2kg	v_2 = $4
item #3	w_3 = 3kg	v_3 = $7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**		
[item #1, item #2]	**2**	**0**					
all items	**3**	**0**					

weights [kg]

$$S[i][w] = Math.max(S[i-1][w]\ ;\ v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5	weights [kg]
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**		
[item #1, item #2]	**2**	**0**						
all items	**3**	**0**						

$$S[i][w] = Math.max(S[i-1][w] \; ; \; v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1 $\mathbf{w_1} = 4kg$ $\mathbf{v_1} = \$10$
item #2 $\mathbf{w_2} = 2kg$ $\mathbf{v_2} = \$4$
item #3 $\mathbf{w_3} = 3kg$ $\mathbf{v_3} = \$7$

		0	**1**	**2**	**3**	**4**	**5**
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**					
all items	**3**	**0**					

weights [kg]

$$S[i][w] = Math.max(S[i-1][w] \; ; \; v_i + S[i-1][w-w_i])$$

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = $10
item #2	w_2 = 2kg	v_2 = $4
item #3	w_3 = 3kg	v_3 = $7

		0	1	2	3	4	5	weights [kg]
no items	0	0	0	0	0	0	0	
[item #1]	1	0	0	0	0	10	10	
[item #1, item #2]	2	0	0					
all items	3	0						

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[2][1] = Math.max(S[1][1] ; $4 + S[1][1-2]) = max(0,0)

0-1 Knapsack Problem

N = 3 items　　**M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = $10
item #2	w_2 = 2kg	v_2 = $4
item #3	w_3 = 3kg	v_3 = $7

		0	1	2	3	4	5
no items	0	0	0	0	0	0	0
[item #1]	1	0	0	0	0	10	10
[item #1, item #2]	2	0	0	4			
all items	3	0					

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[2][2] = Math.max(S[1][2] ; $4 + S[1][2-2]) = max(0,4)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**		
all items	**3**	**0**					

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[2][3] = Math.max(S[1][3] ; \$4 + S[1][3-2]) = max(0,4)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1 $\mathbf{w_1}$ = 4kg $\mathbf{v_1}$ = \$10
item #2 $\mathbf{w_2}$ = 2kg $\mathbf{v_2}$ = \$4
item #3 $\mathbf{w_3}$ = 3kg $\mathbf{v_3}$ = \$7

		0	1	2	3	4	5
no items	0	0	0	0	0	0	0
[item #1]	1	0	0	0	0	10	10
[item #1, item #2]	2	0	0	4	4	10	
all items	3	0					

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[2][4] = Math.max(S[1][4] ; \$4 + S[1][4-2]) = max(10,4)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	0	0	0	0	0	0	0
[item #1]	1	0	0	0	0	10	10
[item #1, item #2]	2	0	0	4	4	10	10
all items	3	0					

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[2][5] = Math.max(S[1][5] ; \$4 + S[1][5-2]) = max(10,4)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**	**10**	**10**
all items	**3**	**0**	**0**				

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[3][1] = Math.max(S[2][1] ; \$7 + S[2][1-3]) = max(0,0)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1 $\mathbf{w_1}$ = 4kg $\mathbf{v_1}$ = \$10
item #2 $\mathbf{w_2}$ = 2kg $\mathbf{v_2}$ = \$4
item #3 $\mathbf{w_3}$ = 3kg $\mathbf{v_3}$ = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**	**10**	**10**
all items	**3**	**0**	**0**	**4**			

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[3][2] = Math.max(S[2][2] ; \$7 + S[2][2-3]) = max(4,0)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5	weights [kg]
no items	0	0	0	0	0	0	0	
[item #1]	1	0	0	0	0	10	10	
[item #1, item #2]	2	0	0	4	4	10	10	
all items	3	0	0	4	7			

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[3][3] = Math.max(S[2][3] ; \$7 + S[2][3-3]) = max(4,7)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**	**10**	**10**
all items	**3**	**0**	**0**	**4**	**7**	**10**	

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[3][4] = Math.max(S[2][4] ; \$7 + S[2][4-3]) = max(10,7)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1 $\mathbf{w_1}$ = 4kg $\mathbf{v_1}$ = $10
item #2 $\mathbf{w_2}$ = 2kg $\mathbf{v_2}$ = $4
item #3 $\mathbf{w_3}$ = 3kg $\mathbf{v_3}$ = $7

		0	1	2	3	4	5
no items	0	0	0	0	0	0	0
[item #1]	1	0	0	0	0	10	10
[item #1, item #2]	2	0	0	4	4	10	10
all items	3	0	0	4	7	10	11

weights [kg]

S[i][w] = Math.max(S[i-1][w] ; v_i + S[i-1][w-w_i])

S[3][5] = Math.max(S[2][5] ; $7 + S[2][5-3]) = max(10,7)

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**	**10**	**10**
all items	**3**	**0**	**0**	**4**	**7**	**10**	**$11**

weights [kg]

*we know that the maximum profit is **$11** but how wo achieve this result – what items to include? we have to start with the result and has to check the rows above with the decremented weights accordingly*

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = \$10
item #2	w_2 = 2kg	v_2 = \$4
item #3	w_3 = 3kg	v_3 = \$7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**	**10**	**10**
all items	**3**	**0**	**0**	**4**	**7**	**10**	**$11**

weights [kg]

0-1 Knapsack Problem

- what items to include?
- we start with the last item (last row and last column) and we keep comparing the items right above (below) each other
- if the **2** values are the same: it means we have not included the given item in the knapsack (so we take **1** step upwards in the **S** table)
- otherwise we take **1** step upwards and take as many steps to the left as the **w** weight of that item

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	w_1 = 4kg	v_1 = $10
item #2	w_2 = 2kg	v_2 = $4
item #3	w_3 = 3kg	v_3 = $7

		0	1	2	3	4	5	weights [kg]
no items	0	0	0	0	0	0	0	
[item #1]	1	0	0	0	0	10	10	
[item #1, item #2]	2	0	0	4	4	10	10	
all items	3	0	0	4	7	10	11	

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1 w_1 = 4kg v_1 = \$10
item #2 w_2 = 2kg v_2 = \$4
item #3 w_3 = 3kg v_3 = \$7

		0	1	2	3	4	5
no items	0	0	0	0	0	0	0
[item #1]	1	0	0	0	0	10	10
[item #1, item #2]	2	0	0	4	4	10	10
all items	3	0	0	4	7	10	11

weights [kg]

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1	$\mathbf{w_1}$ = 4kg	$\mathbf{v_1}$ = $10
item #2	$\mathbf{w_2}$ = 2kg	$\mathbf{v_2}$ = $4
item #3	$\mathbf{w_3}$ = 3kg	$\mathbf{v_3}$ = $7

		0	1	2	3	4	5
no items	**0**	**0**	**0**	**0**	**0**	**0**	**0**
[item #1]	**1**	**0**	**0**	**0**	**0**	**10**	**10**
[item #1, item #2]	**2**	**0**	**0**	**4**	**4**	**10**	**10**
all items	**3**	**0**	**0**	**4**	**7**	**10**	**11**

weights [kg]

0-1 Knapsack Problem

N = 3 items **M = 5kg** capacity of knapsack

item #1 $\mathbf{w_1}$ = 4kg $\mathbf{v_1}$ = \$10
item #2 $\mathbf{w_2}$ = 2kg $\mathbf{v_2}$ = \$4
item #3 $\mathbf{w_3}$ = 3kg $\mathbf{v_3}$ = \$7

		0	1	2	3	4	5	weights [kg]
no items	0	0	0	0	0	0	0	
[item #1]	1	0	0	0	0	10	10	
[item #1, item #2]	2	0	0	4	4	10	10	
all items	3	0	0	4	7	10	11	

0-1 Knapsack Problem

N = 3 items | **M = 5kg** capacity of knapsack

item #1 $\mathbf{w_1}$ = 4kg $\mathbf{v_1}$ = $10

item #2 $\mathbf{w_2}$ = 2kg $\mathbf{v_2}$ = $4

item #3 $\mathbf{w_3}$ = 3kg $\mathbf{v_3}$ = $7

		0	1	2	3	4	5	weights [kg]
no items	0	0	0	0	0	0	0	
[item #1]	1	0	0	0	0	10	10	
[item #1, item #2]	2	0	0	4	4	10	10	
all items	3	0	0	4	7	10	11	

Rod Cutting Problem
(Algorithmic Problems)

Rod Cutting Problem

- given a rod with certain length **N**
- given the $\mathbf{p_i}$ prices for rods of length **i** where $\mathbf{1 \leq i \leq N}$
- each cut is integer length
- what is the optimal way of cutting the rod into smaller parts in order to **maximize profit**?

Rod Cutting Problem

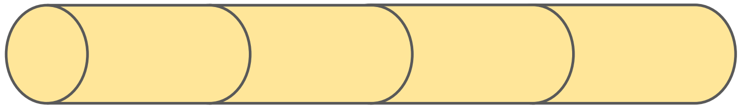
Given a rod with length **N = 4**

1m	2m	3m	4m
$2	$5	$7	$3

Rod Cutting Problem

Given a rod with length **N = 4**

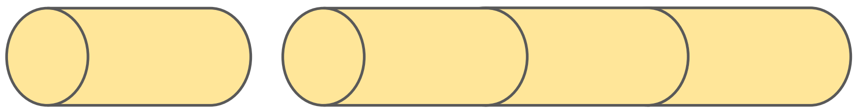
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

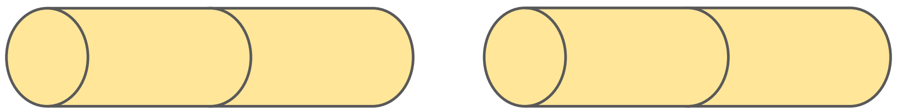
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

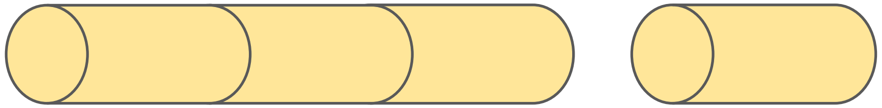
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

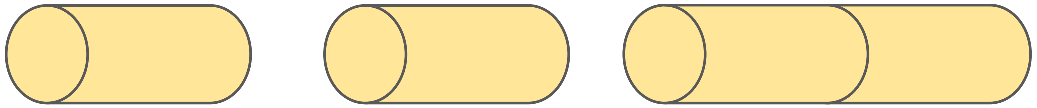
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

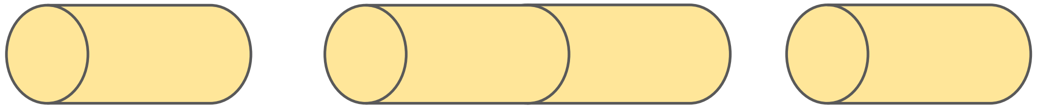
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

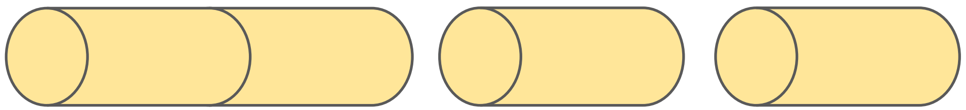
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

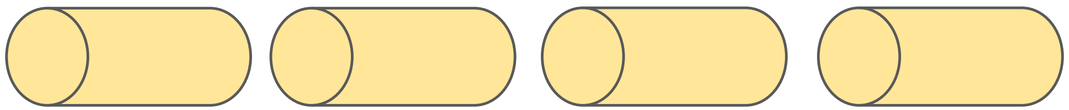
1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

Given a rod with length **N = 4**

1m	2m	3m	4m
$2	$5	$7	$3



Rod Cutting Problem

- the naive approach (brute-force method) is to use a simple recursion
- **N-1** cuts can be made in the rod of length **N**
- which means there are $\mathbf{2^{N-1}}$ ways to cut the rod
- the problem that there are a huge number of overlapping subproblems (as usual with reucurison)
- it has **O(**2^{N}) exponential time complexity - where **N** is the length of the rod in units
- for every length we have **2** options whether to cut or not

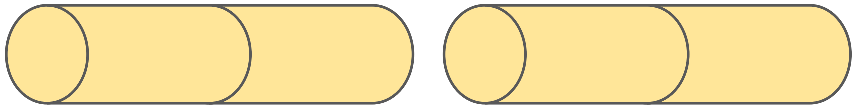
Rod Cutting Problem

- the problem is that we do not know in advance where to cut
- let $\mathbf{r_i}$ be the max (optimal) revenue for rod size **i**

if we cut the rod when ***i=1***
then the total revenue is $\boldsymbol{p[1] + r_{n-1}}$

Rod Cutting Problem

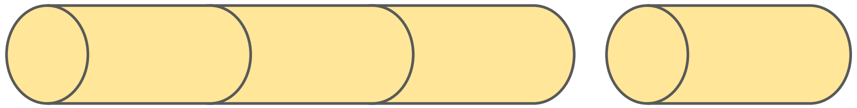
- the problem is that we do not know in advance where to cut
- let $\mathbf{r_i}$ be the max (optimal) revenue for rod size **i**



if we cut the rod when ***i=2***
then the total revenue is $\boldsymbol{p[2] + r_{n-2}}$

Rod Cutting Problem

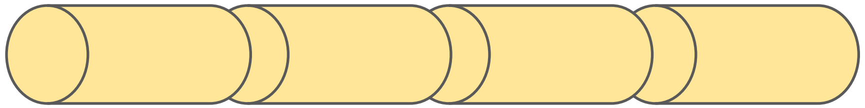
- the problem is that we do not know in advance where to cut
- let $\mathbf{r_i}$ be the max (optimal) revenue for rod size **i**



if we cut the rod when ***i=3***
then the total revenue is $\boldsymbol{p[3] + r_{n-3}}$

Rod Cutting Problem

- the problem is that we do not know in advance where to cut
- let $\mathbf{r_i}$ be the max (optimal) revenue for rod size **i**



$$MAX\ \{\ p[n]\ ,\ p[1] + r_{n-1}\ \ ...\ p[n-1] + r_1\ \}$$

Rod Cutting Problem

- we keep spitting the original problem into overlapping subproblems and finally we combine the subresults
- what is the maximum profit if we have just a single piece of the rod?
- what is the maximum profit if we have two pieces of the rod?
- we keep considering more and more complex subproblems

Rod Cutting Problem

we have to calculate what is the better:
not to make the cut or make the cut

$$S[i][j] = \begin{cases} \text{if } j = 0 \text{ then } 0 \\ \text{if } i = 0 \text{ then } 0 \\ \max\{\, S[i-1][j] \,;\, p[i] + S[i][j-i] \,\} \text{ if } i \leq j \\ S[i-1][j] \text{ if } i > j \end{cases}$$

the total value when
*we have the first **i** pieces*
*and the total length is **j***

if the piece is greater than the length
of the rod of course we skip it
(we do not make the cut- so we try to get the
*max revenue with **i-1** cuts and the **j** length is unchanged)*

Rod Cutting Problem Example

(Algorithmic Problems)

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	**0**							
[piece #1]	**1**							
[piece #1 and #2]	**2**							
[piece #1, #2, #3]	**3**							
[piece #1, #2, #3, #4]	**4**							
all pieces	**5**							

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	**0**	0	0	0	0	0	0	
[piece #1]	**1**							
[piece #1 and #2]	**2**							
[piece #1, #2, #3]	**3**							
[piece #1, #2, #3, #4]	**4**							
all pieces	**5**							

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	**0**	0	0	0	0	0	0	
[piece #1]	**1**	0						
[piece #1 and #2]	**2**	0						
[piece #1, #2, #3]	**3**	0						
[piece #1, #2, #3, #4]	**4**	0						
all pieces	**5**	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2					
[piece #1 and #2]	2	0						
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	**0**	**0**	**0**	**0**	**0**	**0**	**0**	
[piece #1]	**1**	**0**	**2**	**4**				
[piece #1 and #2]	**2**	**0**						
[piece #1, #2, #3]	**3**	**0**						
[piece #1, #2, #3, #4]	**4**	**0**						
all pieces	**5**	**0**						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6			
[piece #1 and #2]	2	0						
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8		
[piece #1 and #2]	2	0						
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0						
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

$S[i][j] = \max(S[i-1][j] \; ; \; p_i + S[i][j-i])$

$S[2][1] = \max(S[1][1] \; ; \; \$5 + S[2][-1]) = \max(2,0)$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2					
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[2][2] = max(S[1][2] ; \$5 + S[2][0]) = max(4,5)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5				
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = $2
piece #2	l_2 = 2m	p_2 = $5
piece #3	l_3 = 3m	p_3 = $7
piece #4	l_4 = 4m	p_4 = $3
Piece #5	l_5 = 5m	p_5 = $9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[2][3] = max(S[1][3] ; $5 + S[2][1]) = max(6,7)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7			
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[2][4] = max(S[1][4] ; \$5 + S[2][2]) = max(8,10)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10		
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

$S[i][j] = \max(S[i-1][j]\ ;\ p_i + S[i][j-i])$

$S[2][5] = \max(S[1][5]\ ;\ \$5 + S[2][3]) = \max(10,12)$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0						
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[3][1] = max(S[2][1] ; \$7 + S[3][-2]) = max(2,0)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2					
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[3][2] = max(S[2][2] ; \$7 + S[3][-1]) = max(5,0)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5				
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[3][3] = max(S[2][3] ; \$7 + S[3][0]) = max(7,7)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7			
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

$S[i][j] = \max(S[i-1][j] \,;\, p_i + S[i][j-i])$

$S[3][4] = \max(S[2][4] \,;\, \$7 + S[3][1]) = \max(10,9)$

		0	1	2	3	4	5
no pieces	0	0	0	0	0	0	0
[piece #1]	1	0	2	4	6	8	10
[piece #1 and #2]	2	0	2	5	7	10	12
[piece #1, #2, #3]	3	0	2	5	7	10	
[piece #1, #2, #3, #4]	4	0					
all pieces	5	0					

length [m]

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	$\mathbf{p_1}$ = \$2
piece #2	l_2 = 2m	$\mathbf{p_2}$ = \$5
piece #3	l_3 = 3m	$\mathbf{p_3}$ = \$7
piece #4	l_4 = 4m	$\mathbf{p_4}$ = \$3
Piece #5	l_5 = 5m	$\mathbf{p_5}$ = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[3][5] = max(S[2][5] ; \$7 + S[3][2]) = max(12,12)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0						
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1 l_1 = 1m p_1 = \$2
piece #2 l_2 = 2m p_2 = \$5
piece #3 l_3 = 3m p_3 = \$7
piece #4 l_4 = 4m p_4 = \$3
Piece #5 l_5 = 5m p_5 = \$9

$S[i][j] = \max(S[i-1][j]\ ;\ p_i + S[i][j-i])$

$S[4][1] = \max(S[3][1]\ ;\ \$3 + S[4][-3]) = \max(2,0)$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2					
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1 l_1 = 1m **p_1** = \$2
piece #2 l_2 = 2m **p_2** = \$5
piece #3 l_3 = 3m **p_3** = \$7
piece #4 l_4 = 4m **p_4** = \$3
Piece #5 l_5 = 5m **p_5** = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[4][2] = max(S[3][2] ; \$3 + S[4][-2]) = max(5,0)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5				
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

$S[i][j] = \max(S[i-1][j] \; ; \; p_i + S[i][j-i])$

$S[4][3] = \max(S[3][3] \; ; \; \$3 + S[4][-1]) = \max(7,0)$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7			
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[4][4] = max(S[3][4] ; \$3 + S[4][0]) = max(10,3)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10		
all pieces	5	0						

Rod Cutting Problem

N = 5m

piece #1 l_1 = 1m p_1 = \$2
piece #2 l_2 = 2m p_2 = \$5
piece #3 l_3 = 3m p_3 = \$7
piece #4 l_4 = 4m p_4 = \$3
Piece #5 l_5 = 5m p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[4][5] = max(S[3][5] ; \$3 + S[4][1]) = max(12,5)

		0	1	2	3	4	5
no pieces	0	0	0	0	0	0	0
[piece #1]	1	0	2	4	6	8	10
[piece #1 and #2]	2	0	2	5	7	10	12
[piece #1, #2, #3]	3	0	2	5	7	10	12
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12
all pieces	5	0					

length [m]

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

$S[i][j] = \max(S[i-1][j] \,;\, p_i + S[i][j-i])$

$S[5][1] = \max(S[4][1] \,;\, \$9 + S[5][-4]) = \max(2,0)$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2					

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

$S[i][j] = \max(S[i-1][j] \; ; \; p_i + S[i][j-i])$

$S[5][2] = \max(S[4][2] \; ; \; \$9 + S[5][-3]) = \max(5,0)$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5				

Rod Cutting Problem

N = 5m

piece #1 l_1 = 1m p_1 = \$2
piece #2 l_2 = 2m p_2 = \$5
piece #3 l_3 = 3m p_3 = \$7
piece #4 l_4 = 4m p_4 = \$3
Piece #5 l_5 = 5m p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[5][3] = max(S[4][3] ; \$9 + S[5][-2]) = max(7,0)

		0	1	2	3	4	5
no pieces	0	0	0	0	0	0	0
[piece #1]	1	0	2	4	6	8	10
[piece #1 and #2]	2	0	2	5	7	10	12
[piece #1, #2, #3]	3	0	2	5	7	10	12
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12
all pieces	5	0	2	5	7		

length [m]

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[5][4] = max(S[4][4] ; \$9 + S[5][-1]) = max(10,0)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10		

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

S[i][j] = max(S[i-1][j] ; p_i + S[i][j-i])

S[5][5] = max(S[4][5] ; \$9 + S[5][0]) = max(12,9)

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	$l_1 = 1m$	$p_1 = \$2$
piece #2	$l_2 = 2m$	$p_2 = \$5$
piece #3	$l_3 = 3m$	$p_3 = \$7$
piece #4	$l_4 = 4m$	$p_4 = \$3$
Piece #5	$l_5 = 5m$	$p_5 = \$9$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	$\mathbf{l_1}$ = 1m	$\mathbf{p_1}$ = \$2
piece #2	$\mathbf{l_2}$ = 2m	$\mathbf{p_2}$ = \$5
piece #3	$\mathbf{l_3}$ = 3m	$\mathbf{p_3}$ = \$7
piece #4	$\mathbf{l_4}$ = 4m	$\mathbf{p_4}$ = \$3
Piece #5	$\mathbf{l_5}$ = 5m	$\mathbf{p_5}$ = \$9

		0	1	2	3	4	5	**length [m]**
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	$l_1 = 1m$	$p_1 = \$2$
piece #2	$l_2 = 2m$	$p_2 = \$5$
piece #3	$l_3 = 3m$	$p_3 = \$7$
piece #4	$l_4 = 4m$	$p_4 = \$3$
Piece #5	$l_5 = 5m$	$p_5 = \$9$

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Rod Cutting Problem

N = 5m

piece #1	l_1 = 1m	p_1 = \$2
piece #2	l_2 = 2m	p_2 = \$5
piece #3	l_3 = 3m	p_3 = \$7
piece #4	l_4 = 4m	p_4 = \$3
Piece #5	l_5 = 5m	p_5 = \$9

		0	1	2	3	4	5	length [m]
no pieces	0	0	0	0	0	0	0	
[piece #1]	1	0	2	4	6	8	10	
[piece #1 and #2]	2	0	2	5	7	10	12	
[piece #1, #2, #3]	3	0	2	5	7	10	12	
[piece #1, #2, #3, #4]	4	0	2	5	7	10	12	
all pieces	5	0	2	5	7	10	12	

Subset Sum Problem

(Algorithmic Problems)

Subset Sum Problem

- one of the most important problems in complexity theory
- the problem is that given an **A** set of integers $\mathbf{a_1}$, $\mathbf{a_2}$ ... $\mathbf{a_N}$
- is there a non-empty subset such that the sum of the subset is a given **M** integer?
- for example: given the set **[5, 2, 1, 3]** and **s=9** the answer is **YES** because the subset **[5, 3, 1]** sums to **9**
- this is an **NP-complete** problem again
- by the way it is the special case of **knapsack problem**

Subset Sum Problem

4	-2	2	3	1

Subset Sum Problem

4	-2	2	3	1

Subset Sum Problem

Subset Sum Problem

- the naive approach (brute-force search) generates all the possible subsets of the original array – there are $\mathbf{2^N}$ possible states
- then considers all these subsets in **O(N)** linear running time and checks whether the sum of the items is **M** or not
- the **dynamic programming** approach has **pseudo-polynomial** running time again

Subset Sum Problem

Again as usual with dynamic programming approaches we have **2** options for every single item – we may include that $\mathbf{a}_i$ item or exclude $\mathbf{a}_i$

solve(A, M, i)

*we try to solve the problem when we have the **A** array with the **M** sum and using the first **i** items*

Subset Sum Problem

4	-2	2	3	1

Let’s consider the last $\mathbf{a_N}$ item (value **1**). We can include that item in the final solution or exclude that value (we do not know in advance)

IF WE INCLUDE THE LAST a_N ITEM

solve(A , M-a_N, i-1)

Subset Sum Problem

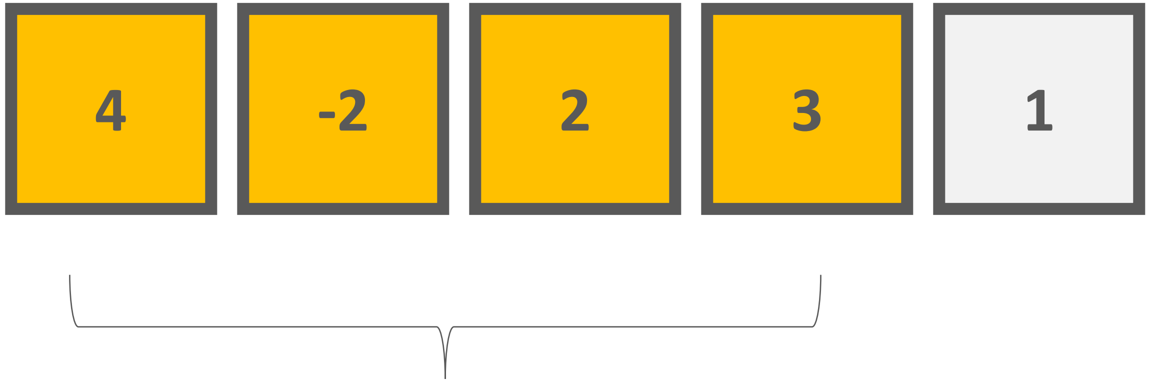
4	-2	2	3	1

Let's consider the last $\mathbf{a_N}$ item (value **1**). We can include that item in the final solution or exclude that value (we do not know in advance)

IF WE INCLUDE THE LAST a_N ITEM

solve(A , M , i-1)

Subset Sum Problem



and of course we can use the exact
*same approach for the previous **i-1** items*

Subset Sum Problem

$$S[i][j] = \begin{cases} \text{if } j = 0 \text{ then true} \\ \text{if } i = 0 \text{ then false} \\ S[i][j] = S[i-1][j] \text{ if } S[i-1][j] \text{ is true} \\ S[i][j] = S[i-1][j-A[i-1]] \quad \text{else} \end{cases}$$

*there is a non-empty subset of the first **i** integers that sums to **j***

Subset Sum Problem

we can always make the empty subset
*to end up with sum **0***

$$
S[i][j] = \begin{cases} \textbf{if } j = 0 \textbf{ then true} \\ \textbf{if } i = 0 \textbf{ then false} \\ S[i][j] = S[i-1][j] \textbf{ if } S[i-1][j] \textbf{ is true} \\ S[i][j] = S[i-1][j - A[i-1]] \textbf{ else} \end{cases}
$$

there is a non-empty subset of
*the first **i** integers that sums to **j***

Subset Sum Problem

*of course if **j** can be constructed with **i-1** integers then there must be a valid subset with **i** included as well (**EXCLUDE i ITEM**)*

$$S[i][j] = \begin{cases} \textbf{if } j = 0 \textbf{ then true} \\ \textbf{if } i = 0 \textbf{ then false} \\ S[i][j] = S[i-1][j] \textbf{ if } S[i-1][j] \textbf{ is true} \\ S[i][j] = S[i-1][j - A[i-1]] \quad \textbf{else} \end{cases}$$

*there is a non-empty subset of the first **i** integers that sums to **j***

Subset Sum Problem

if we can solve the problem with ***S: [1,2,3]*** *and* ***M = 5***
and there is a solution ***[2, 3]***
then we can solve the problem with other vales
as well such as ***[1, 2, 3, 4]***

$$S[i][j] = \begin{cases} \text{if } j = 0 \text{ then true} \\ \text{if } i = 0 \text{ then false} \\ S[i][j] = S[i-1][j] \text{ if } S[i-1][j] \text{ is true} \\ S[i][j] = S[i-1][j-A[i-1]] \text{ else} \end{cases}$$

there is a non-empty subset of
the first ***i*** *integers that sums to* ***j***

Subset Sum Problem

*of course if **j** can be constructed with **i-1** integers then there must be a valid subset with **i** included as well (**EXCLUDE i ITEM**)*

$$S[i][j] = \begin{cases} \textbf{if } j = 0 \textbf{ then true} \\ \textbf{if } i = 0 \textbf{ then false} \\ S[i][j] = S[i-1][j] \textbf{ if } S[i-1][j] \textbf{ is true} \\ S[i][j] = S[i-1][j-A[i-1]] \textbf{ else} \end{cases}$$

*there is a non-empty subset of the first **i** integers that sums to **j***

*the previous **i-1** items does not sum up to **j** so we try to solve the problem by including item **i** (**INCLUDE i ITEM**)*

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**										
[5]	**1**										
[5, 2]	**2**										
[5, 2, 1]	**3**										
[5, 2, 1, 3]	**4**										

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	F	F	F	F	F	F	F	F	F	F
[5]	**1**										
[5, 2]	**2**										
[5, 2, 1]	**3**										
[5, 2, 1, 3]	**4**										

*j index **[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
[5]	**1**	**T**										
[5, 2]	**2**	**T**										
[5, 2, 1]	**3**	**T**										
[5, 2, 1, 3]	**4**	**T**										

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F								
[5, 2]	2	T									
[5, 2, 1]	3	T									
[5, 2, 1, 3]	4	T									

*j index **[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**							
[5, 2]	**2**	**T**									
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**						
[5, 2]	**2**	**T**									
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F					
[5, 2]	2	T									
[5, 2, 1]	3	T									
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**					
[5, 2]	**2**	**T**										
[5, 2, 1]	**3**	**T**										
[5, 2, 1, 3]	**4**	**T**										

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F			
[5, 2]	2	T									
[5, 2, 1]	3	T									
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**		
[5, 2]	**2**	**T**									
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	
[5, 2]	**2**	**T**									
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T										
[5, 2, 1]	3	T										
[5, 2, 1, 3]	4	T										

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T										
[5, 2, 1]	3	T										
[5, 2, 1, 3]	4	T										

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**								
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

*j index **[0-9]***

dpTable[2][1] = dpTable[1][1-2] = dpTable[1][-1] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T								
[5, 2, 1]	3	T										
[5, 2, 1, 3]	4	T										

dpTable[2][2] = dpTable[1][2-2] = dpTable[1][0] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**						
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

dpTable[2][3] = dpTable[1][3-2] = dpTable[1][1] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**					
[5, 2, 1]	**3**	**T**									
[5, 2, 1, 3]	**4**	**T**									

*j index **[0-9]***

dpTable[2][4] = dpTable[1][4-2] = dpTable[1][2] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T					
[5, 2, 1]	3	T										
[5, 2, 1, 3]	4	T										

dpTable[2][5] = dpTable[1][5] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F			
[5, 2, 1]	3	T									
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[2][6] = dpTable[1][6-2] = dpTable[1][4] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T		
[5, 2, 1]	3	T									
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[2][7] = dpTable[1][7-2] = dpTable[1][5] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	
[5, 2, 1]	**3**	T									
[5, 2, 1, 3]	**4**	T									

*j index **[0-9]***

dpTable[2][8] = dpTable[1][8-2] = dpTable[1][6] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T									
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[2][9] = dpTable[1][9-2] = dpTable[1][7] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	**3**	T	T								
[5, 2, 1, 3]	**4**	T									

*j index **[0-9]***

dpTable[3][1] = dpTable[2][1-1] = dpTable[2][0] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T							
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[3][2] = dpTable[2][2] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T	T						
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[3][3] = dpTable[2][3-1] = dpTable[2][2] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**					
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

dpTable[3][4] = dpTable[2][4-1] = dpTable[2][3] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T	T	F	T				
[5, 2, 1, 3]	4	T									

*j index **[0-9]***

dpTable[3][5] = dpTable[2][5]

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T	T	F	T	T			
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[3][6] = dpTable[2][6-1] = dpTable[2][5] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**		
[5, 2, 1, 3]	**4**	**T**									

j index ***[0-9]***

dpTable[3][7] = dpTable[2][7]

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[3][8] = dpTable[2][8-1] = dpTable[2][7] = T

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	4	T									

j index ***[0-9]***

dpTable[3][9] = dpTable[2][9-1] = dpTable[2][8] = F

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**
[5, 2, 1, 3]	**4**	**T**	**T**								

*j index **[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T	F	T	F	F	
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F	
[5, 2, 1, 3]	4	T	T	T								

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T	F	T	F	F	
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F	
[5, 2, 1, 3]	4	T	T	T	T							

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T	F	T	F	F	
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F	
[5, 2, 1, 3]	4	T	T	T	T	T						

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T	F	T	F	F	
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F	
[5, 2, 1, 3]	4	T	T	T	T	T	T					

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	**3**	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	**4**	T	T	T	T	T	T	T			

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**	
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**	
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**	
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**			

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	**3**	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	**4**	T	T	T	T	T	T	T	T	T	

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**

*j index **[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**	
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**	
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**	
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	**3**	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	**4**	T	T	T	T	T	T	T	T	T	T

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**	
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**	
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**	
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	0	T	F	F	F	F	F	F	F	F	F
[5]	1	T	F	F	F	F	T	F	F	F	F
[5, 2]	2	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	4	T	T	T	T	T	T	T	T	T	T

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**	
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**	
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**	
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index* ***[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T	F	T	F	F	
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F	
[5, 2, 1, 3]	4	T	T	T	T	T	T	T	T	T	T	

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	**3**	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	**4**	T	T	T	T	T	T	T	T	T	T

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**

j index ***[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	T	F	F	F	F	F	F	F	F	F
[5]	**1**	T	F	F	F	F	T	F	F	F	F
[5, 2]	**2**	T	F	T	F	F	T	F	T	F	F
[5, 2, 1]	**3**	T	T	T	T	F	T	T	T	T	F
[5, 2, 1, 3]	**4**	T	T	T	T	T	T	T	T	T	T

*j index **[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9
[]	**0**	**T**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**	**F**
[5]	**1**	**T**	**F**	**F**	**F**	**F**	**T**	**F**	**F**	**F**	**F**
[5, 2]	**2**	**T**	**F**	**T**	**F**	**F**	**T**	**F**	**T**	**F**	**F**
[5, 2, 1]	**3**	**T**	**T**	**T**	**T**	**F**	**T**	**T**	**T**	**T**	**F**
[5, 2, 1, 3]	**4**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**

*j index **[0-9]***

Subset Sum Problem

A set of integers: **[5, 2, 1, 3]**
M = 9

		0	1	2	3	4	5	6	7	8	9	*j index **[0-9]***
[]	0	T	F	F	F	F	F	F	F	F	F	
[5]	1	T	F	F	F	F	T	F	F	F	F	
[5, 2]	2	T	F	T	F	F	T	F	T	F	F	
[5, 2, 1]	3	T	T	T	T	F	T	T	T	T	F	
[5, 2, 1, 3]	4	T	T	T	T	T	T	T	T	T	T	