Introduction to Algorithms and Data Structures
Dynamic Programming - Subset Sum and Knapsack
The subset sum problem
The **subset sum** problem

We are given a set of $n$ items \{1, 2, \ldots, n\}. 
The **subset sum** problem

We are given a set of $n$ items \{1, 2, … , $n$\}.

Each item $i$ has a non-negative weight $w_i$. 
The **subset sum** problem

We are given a set of \( n \) items \( \{1, 2, \ldots, n\} \).

Each item \( i \) has a non-negative weight \( w_i \).

We are given a bound \( W \).
The subset sum problem

We are given a set of $n$ items $\{1, 2, \ldots, n\}$. Each item $i$ has a non-negative weight $w_i$. We are given a bound $W$.

**Goal:** Select a subset $S$ of the items such that

$$\sum_{i \in N} w_i \leq W$$

and

$$\sum_{i \in N} w_i$$

is maximised.
To be more precise

We are given a set of $n$ items $\{1, 2, \ldots, n\}$. Each item $i$ has a non-negative weight $w_i$. We are given a bound $W$. 
To be more precise

We are given a set of $n$ items \{1, 2, … , $n$\}.

Each item $i$ has a non-negative weight $w_i$.

We are given a bound $W$.

*How are these inputs represented by a computer?*
Representation
Representation

Let’s say that $w_3 = 5$. 
Representation

Let’s say that $w_3 = 5$.

How do we “save” 5 in a computer, using only 0 and 1?
Let's say that $w_3 = 5$.

How do we “save” 5 in a computer, using only 0 and 1?

Binary representation: $5_{10} = 101_{2} \rightarrow 101$
Let’s say that $w_3 = 5$.

How do we “save” 5 in a computer, using only 0 and 1?

Binary representation: $5_{10} = 101_2 \rightarrow 101$

Unary representation: $5_{10} \rightarrow 11111$
The **subset sum problem**

We are given a set of \( n \) items \( \{1, 2, \ldots, n\} \).

Each item \( i \) has a non-negative weight \( w_i \) *given in binary representation.*

We are given a bound \( W \) *given in binary representation.*

**Goal:** Select a subset \( S \) of the items such that

\[
\sum_{i \in N} w_i \leq W \quad \text{and} \quad \sum_{i \in N} w_i \text{ is maximised.}
\]
Our input

\[
\begin{align*}
\omega_{1}^{\text{bin}} & \quad \omega_{2}^{\text{bin}} & \quad \omega_{3}^{\text{bin}} & \quad \text{...} & \quad \omega_{n}^{\text{bin}} \\
\end{align*}
\]
The subset sum problem

We are given a set of $n$ items $\{1, 2, \ldots, n\}$.

Each item $i$ has a non-negative weight $w_i$.

We are given a bound $W$.

**Goal:** Select a subset $S$ of the items such that

$$\sum_{i \in N} w_i \leq W \quad \text{and} \quad \sum_{i \in N} w_i \text{ is maximised.}$$
Greedy Approaches
Greedy Approaches

• Ideas?
Greedy Approaches

• Ideas?

• Sort items in terms of *decreasing weight* and put them in \( S \) one by one.
Greedy Approaches

• Ideas?

• Sort items in terms of *decreasing weight* and put them in \( S \) one by one.

• Sort items in terms of *increasing weight* and put them in \( S \) one by one.
Greedy Approaches

- Sort items in terms of *decreasing weight* and put them in $S$ one by one.
Greedy Approaches

- Sort items in terms of *decreasing weight* and put them in $S$ one by one.

- Example where this fails?
Greedy Approaches

- Sort items in terms of *decreasing weight* and put them in $S$ one by one.

- Example where this fails?
  
  - $W = 4, w_1 = 3, w_2 = 2, w_3 = 2$. 

Greedy Approaches

• Sort items in terms of *increasing weight* and put them in $S$ one by one.
Greedy Approaches

• Sort items in terms of *increasing weight* and put them in $S$ one by one.

• Example where this fails?
Greedy Approaches

• Sort items in terms of *increasing weight* and put them in $S$ one by one.

• Example where this fails?

  • $W = 4$, $w_1 = 1$, $w_2 = 2$, $w_3 = 2$. 
Dynamic Programming
Dynamic Programming

- We need to identify the appropriate subproblems to use in order to solve the main problem.
Dynamic Programming

• We need to identify the appropriate subproblems to use in order to solve the main problem.

• Recall the weighted interval scheduling problem. Similar approach.
Dynamic Programming

• We need to identify the appropriate subproblems to use in order to solve the main problem.

• Recall the weighted interval scheduling problem. Similar approach.

• Let $O_i$ be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of \{1, 2, \ldots , i\}, and let $OPT(i)$ be its value.
Dynamic Programming

• We need to identify the appropriate subproblems to use in order to solve the main problem.

• Recall the weighted interval scheduling problem. Similar approach.

• Let $O_i$ be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of $\{1, 2, \ldots, i\}$, and let $OPT(i)$ be its value.

  • Hence $O$ is $O_n$, and $OPT = OPT(n)$
Dynamic Programming

• We need to identify the appropriate subproblems to use in order to solve the main problem.

• Recall the weighted interval scheduling problem. Similar approach.

• Let $O_i$ be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of \{1, 2, \ldots, i\}, and let $OPT(i)$ be its value.

  • Hence $O$ is $O_n$, and $OPT = OPT(n)$

• Should item $n$ be in the optimal solution $O$ or not?
Dynamic Programming

- We need to identify the appropriate subproblems to use in order to solve the main problem.

- Recall the weighted interval scheduling problem. Similar approach.

- Let $O_i$ be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of $\{1, 2, \ldots, i\}$, and let $OPT(i)$ be its value.

  - Hence $O$ is $O_n$, and $OPT = OPT(n)$

- Should item $n$ be in the optimal solution $O$ or not?

  - If no, then $OPT(n-1) = OPT(n)$
Dynamic Programming

• We need to identify the appropriate subproblems to use in order to solve the main problem.

• Recall the weighted interval scheduling problem. Similar approach.

• Let $O_i$ be the optimal solution to the subset be the optimal solution to the subset sum problem, using a subset of $\{1, 2, \ldots, i\}$, and let $\text{OPT}(i)$ be its value.

  • Hence $O$ is $O_n$, and $\text{OPT} = \text{OPT}(n)$

• Should item $n$ be in the optimal solution $O$ or not?

  • If no, then $\text{OPT}(n-1) = \text{OPT}(n)$

  • If yes, ?
If $n$ is in $O$
If \( n \) is in \( O \)

- What information do we get about the other items?
If $n$ is in $O$:

- What information do we get about the other items?
- In weighted interval scheduling, we could remove all intervals overlapping with $n$. 
If $n$ is in $O$:

- What information do we get about the other items?

- In weighted interval scheduling, we could remove all intervals overlapping with $n$.

- Can we do something similar here?
If $n$ is in $O$

- What information do we get about the other items?
- In weighted interval scheduling, we could remove all intervals overlapping with $n$.
- Can we do something similar here?
  - There is no reason to a-priori exclude any remaining item, unless adding it would exceed the weight.
If $n$ is in $O$

- What information do we get about the other items?

- In weighted interval scheduling, we could remove all intervals overlapping with $n$.

- Can we do something similar here?

  - There is no reason to a-priori exclude any remaining item, unless adding it would exceed the weight.

  - The only information that we really get is that we now have weight $W - w_n$ left.
What we really need
What we really need

• To find the optimal value $\text{OPT}(n)$, we need
What we really need

• To find the optimal value $\text{OPT}(n)$, we need

• The optimal value $\text{OPT}(n-1)$ if $n$ is not in $O$. 
What we really need

- To find the optimal value $\text{OPT}(n)$, we need
  - The optimal value $\text{OPT}(n-1)$ if $n$ is not in $O$.
  - The optimal value of the solution on input $\{1, 2, \ldots, n-1\}$ and $w = W - w_n$. 
What we really need

• To find the optimal value \( \text{OPT}(n) \), we need
  
  • The optimal value \( \text{OPT}(n-1) \) if \( n \) is not in \( O \).
  
  • The optimal value of the solution on input \( \{1, 2, \ldots, n-1\} \) and \( w = W - w_n \).

• How many subproblems do we need?
What we really need

• To find the optimal value $\text{OPT}(n)$, we need
  • The optimal value $\text{OPT}(n-1)$ if $n$ is not in $O$.
  • The optimal value of the solution on input \{1, 2, ..., n-1\} and $w = W - w_n$.

• How many subproblems do we need?
  • One for each initial set \{1, 2, ..., i\} of items and each possible value for the remaining weight $w$. 
Subproblems
Subproblems

- Assumptions:
Subproblems

• Assumptions:

  • $W$ is an integer.
Subproblems

• Assumptions:
  
  • $W$ is an integer.
  
  • Every $w_i$ is an integer.
Subproblems

• Assumptions:
  
  • $W$ is an integer.
  
  • Every $w_i$ is an integer.
  
  • We will have one subproblem for each $i=0, 1, \ldots, n$ and each integer $0 \leq w \leq W$. 
Subproblems

• Assumptions:
  • $W$ is an integer.
  • Every $w_i$ is an integer.
  • We will have one subproblem for each $i=0,1,\ldots,n$ and each integer $0 \leq w \leq W$.
  • Let $\text{OPT}(i,w)$ be the value of the optimal solution on subset $\{1, 2, \ldots, i\}$ and maximum allowed weight $w$. 
Subproblems
Subproblems

- Using this notation, what are we looking for?
Subproblems

• Using this notation, what are we looking for?
  • \( \text{OPT}(n,W) \)
Subproblems

• Using this notation, what are we looking for?
  • \text{OPT}(n,W)

• Should item \textit{n} be in the optimal solution \textit{O} or not?
Subproblems

• Using this notation, what are we looking for?

  • \( \text{OPT}(n, W) \)

• Should item \( n \) be in the optimal solution \( O \) or not?

  • If no, then \( \text{OPT}(n, W) = \text{OPT}(n-1, W) \).
Subproblems

• Using this notation, what are we looking for?
  • $\text{OPT}(n,W)$

• Should item $n$ be in the optimal solution $O$ or not?
  • If no, then $\text{OPT}(n,W) = \text{OPT}(n-1,W)$.
  • If yes, then $\text{OPT}(n,W) = w_n + \text{OPT}(n-1,W-w_n)$. 
Subproblems

Is $j$ in $O$?

yes

$OPT(j, w) = w_j + OPT(j-1, w-w_j)$.

no

$OPT(j, w) = OPT(j-1, w)$.
Subproblems

Is \( j \) in \( O \) ?

- **yes**
  \[
  \text{OPT}(j, w) = w_j + \text{OPT}(j - 1, w - w_j).
  \]

- **no**
  \[
  \text{OPT}(j, w) = \text{OPT}(j - 1, w).
  \]

\[
\text{OPT}(j, w) = \max\{ w_j + \text{OPT}(j - 1, w - w_j), \text{OPT}(j - 1, w) \}
\]
Subproblems

Is $j$ in $O$?

Unless $w_j > w$

- **yes**
  - $OPT(j, w) = w_j + OPT(j-1, w-w_j)$

- **no**
  - $OPT(j, w) = OPT(j-1, w)$

$OPT(j, w) = \max\{ w_j + OPT(j-1, w-w_j), OPT(j-1, w) \}$
Algorithm **SubsetSum**(*n*, *W*)

Array $M = [0 \ldots n, 0 \ldots W]$
Initialise $M[0, w] = 0$, for each $w = 0, 1, \ldots, W$

For $i = 1, 2, \ldots, n$
    For $w = 0, \ldots, W$
        If $(w_i > w)$ /* If the item does not fit */
            $M[i, w] = M[i-1, w]$
        Else
            $M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$
        EndIf

Return $M[n, W]$
Two dimensional array

\[
\begin{array}{cccccccccccccccc}
  & n & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  n-1 & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  \ldots & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  \ldots & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  i & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  i-1 & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  \ldots & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  2 & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  1 & 0 &  &  &  &  &  &  &  &  &  &  &  &  &  &  \\
  \hline
  0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  \hline
  0 & 1 & \ldots & w-w_i & w & & & & W & \\
\end{array}
\]
Two dimensional array

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>OPT(n,W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>n-1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>i-1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>...</td>
<td>w-w_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two dimensional array

\[
\begin{array}{cccccccccc}
 n & 0 & & & & & & & & \text{OPT}(n,W) \\
 n-1 & 0 & & & & & & & & \\
 ... & 0 & & & & & & & & \\
 ... & 0 & & & & & & & & \\
 ... & 0 & & & & & & & & \\
 i & 0 & & & & & & & & \\
 i-1 & 0 & & & & & & & & \\
 ... & 0 & & & & & & & & \\
 2 & 0 & & & & & & & & \\
 1 & 0 & & & & & & & & \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & ... & w-w_i & w & & & & W \\
\end{array}
\]
Two dimensional array

| n | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | OPT(n,W) |
| n-1| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | OPT(n-1,W) |
| ...| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| ...| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| ...| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| i | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| i-1| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| ...| 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 2 | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 1 | 0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | ... | w-w_i |   | w |   | W |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
Two dimensional array

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n-1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i-1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>w-w_i</td>
<td>w</td>
<td></td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OPT(n, W)

OPT(n-1, W)

OPT(n-2, W-w_{n-2})

OPT(n-2, W)
Two dimensional array

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>w-w_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>W</td>
</tr>
</tbody>
</table>

\[ \text{OPT}(n, W) \]
\[ \text{OPT}(n-1, W) \]
\[ \text{OPT}(n-2, W-w_{n-2}) \]
Two dimensional array

\[
\begin{array}{cccccccccccc}
\text{n} & 0 & & & & & & & & & & & \\
\text{n-1} & 0 & & & & & & & & & & & \\
\text{...} & 0 & & & & & & & & & & & \\
\text{...} & 0 & & & & & & & & & & & \\
\text{i} & 0 & & & & & & & & & & & \\
\text{i-1} & 0 & & & & & & & & & & & \\
\text{...} & 0 & & & & & & & & & & & \\
\text{2} & 0 & & & & & & & & & & & \\
\text{1} & 0 & & & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & \ldots & w-w_i & \text{w} & \text{w} & \text{W} \\
\end{array}
\]

\[
\text{OPT(}n\text{,W)} \quad \text{OPT(}n-1\text{,W)} \quad \text{OPT(}n\text{-2,}\text{W-w}\text{n-2)} \quad \text{OPT(}n\text{-2,}\text{W)}
\]
Two dimensional array

\[
\begin{array}{ccccccccccc}
  & n & 0 &  &  &  &  &  &  &  &  &  \\
  n-1 & 0 &  &  &  &  &  &  &  &  &  &  \\
  ... & 0 &  &  &  &  &  &  &  &  &  &  \\
  ... & 0 &  &  &  &  &  &  &  &  &  &  \\
  i & 0 &  &  &  &  &  &  &  &  &  &  \\
  i-1 & 0 &  &  &  &  &  &  &  &  &  &  \\
  ... & 0 &  &  &  &  &  &  &  &  &  &  \\
  2 & 0 &  &  &  &  &  &  &  &  &  &  \\
  1 & 0 &  &  &  &  &  &  &  &  &  &  \\
  0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  0 & 1 & ... & w-w_i & w &  &  &  &  &  \\
  &  &  &  &  &  &  &  &  &  &  & W
\end{array}
\]
Two dimensional array

\[ \text{OPT}(n, W) \]
\[ \text{OPT}(n-1, W) \]
\[ \text{OPT}(n-2, W - w_{n-2}) \]
Example

- \( n=3 \), \( W=6 \), \( w_1 = w_2 = 2 \) and \( w_3 = 3 \).

Array \( M=[0 \ldots n, 0 \ldots W] \)
Initialise \( M[0, w] = 0 \), for each \( w = 0, 1, \ldots, W \)
Example

- \( n=3 \), \( W=6 \), \( w_1 = w_2 = 2 \) and \( w_3 = 3 \).

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$. 

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3. \)

For \( w = 0 , \ldots , W \)

If \( (w_i > w) \)

\[
M[i, w] = M[i-1, w]
\]

Else

\[
M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
\]

EndIf
Example

- \( n=3 \), \( W=6 \), \( w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0 \), …, \( W \)

If \( (w_i > w) \) **doesn’t fit**

\[ M[i, w] = M[i-1, w] \]

Else

\[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \]

EndIf
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0, \ldots, \ W \)

If \( (w_i > w) \)

\[ M[i, \ w] = M[i-1, \ w] \]

Else

\[ M[i, \ w] = \max\{M[i-1, \ w], w_i + M[i-1, \ w-w_i]\} \]

EndIf
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0, \ldots, W \)

- If \( (w_i > w) \) **doesn't fit**
  
  \[
  M[i, w] = M[i-1, w]
  \]

- Else
  
  \[
  M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
  \]

EndIf
Example

- \( n=3 \), \( W=6 \), \( w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0, \ldots, W \)

If \( (w_i > w) \)

\[ M[i, w] = M[i-1, w] \]

Else

\[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \]

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

\[
\begin{array}{cccccccc}
3 & & & & & & & \\
2 & & & & & & & \\
1 & 0 & 0 & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & \\
\end{array}
\]

For $w = 0, \ldots, W$

- If ($w_i > w$) fits
  \[
  M[i, w] = M[i-1, w]
  \]
- Else
  \[
  M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
  \]

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

For $w = 0, \ldots, W$

If $(w_i > w)$ fits

\[ M[i, w] = M[i-1, w] \]

Else

\[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \]

EndIf
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0, \ldots, W \)

- If \((w_i > w)\) **fits**
  \[
  M[i, w] = M[i-1, w]
  \]
- Else
  \[
  M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
  \]

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

For $w = 0, \ldots, W$
If ($w_i > w$)
  \[ M[i, w] = M[i-1, w] \]
Else
  \[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \]
EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

For $w = 0, \ldots, W$

If ($w_i > w$)
   
   $M[i, w] = M[i-1, w]$

Else
   
   $M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

For $w = 0, \ldots, W$

If ($w_i > w$)

$M[i, w] = M[i-1, w]$

Else

$M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$

EndIf
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0, \ldots, W \)
If \( (w_i > w) \)
\[
M[i, w] = M[i-1, w]
\]
Else
\[
M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
\]
EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

\[
\begin{array}{cccccccc}
3 & & & & & & & \\
\hline
2 & 0 & 0 & & & & & \\
1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

For $w = 0, \ldots, W$
If ($w_i > w$)
\[M[i, w] = M[i-1, w]\]
Else
\[M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}\]
EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

For $w = 0, \ldots, W$

If ($w_i > w$)

\[ M[i, w] = M[i-1, w] \]

Else

\[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \]

EndIf
Example

- \( n = 3, W = 6, w_1 = w_2 = 2 \) and \( w_3 = 3 \).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

For \( w = 0, \ldots, W \)

- If \( (w_i > w) \)
  - \( M[i, w] = M[i-1, w] \)
- Else
  - \( M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \)

EndIf
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For \( w = 0, \ldots, W \)

If \( (w_i > w) \),

\[
M[i, w] = M[i-1, w]
\]

Else,

\[
M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
\]

EndIf
Example

- \( n=3, \ W=6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

For \( w = 0, \ldots, \ W \)

If \( (w_i > w) \)

\[ M[i, w] = M[i-1, w] \]

Else

\[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\} \]

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For $w = 0, \ldots, W$

If ($w_i > w$)

$M[i, w] = M[i-1, w]$

Else

$M[i, w] = \max(M[i-1, w], w_i + M[i-1, w-w_i])$

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

For $w = 0, \ldots, W$

- If ($w_i > w$)
  
  $M[i, w] = M[i-1, w]$

- Else
  
  $M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$

EndIf
Example

- \( n = 3, \ W = 6, \ w_1 = w_2 = 2 \) and \( w_3 = 3 \).

\[\begin{array}{cccccc}
3 & 0 & 0 & 2 & & & \\
2 & 0 & 0 & 2 & 2 & 4 & 4 & 4 \\
1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\]

For \( w = 0, \ldots, \ W \)

If \( (w_i > w) \)

\[ M[i, w] = M[i-1, w] \]

Else

\[ M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w - w_i]\} \]

EndIf
Example

- \( n=3 \), \( W=6 \), \( w_1 = w_2 = 2 \) and \( w_3 = 3 \).

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
```

For \( w = 0, \ldots, W \)

If \((w_i > w)\)

\[
M[i, w] = M[i-1, w]
\]

Else

\[
M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}
\]

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
```

For $w = 0, \ldots, W$

- If ($w_i > w$)
  - $M[i, w] = M[i-1, w]$
- Else
  - $M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$

EndIf
Example

- \( n=3 \), \( W=6 \), \( w_1 = w_2 = 2 \) and \( w_3 = 3 \).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

For \( w = 0, \ldots, W \)
- If \( (w_i > w) \)
  - \( M[i, w] = M[i-1, w] \)
- Else
  - \( M[i, w] = \text{max}\{M[i-1, w], w_i + M[i-1, w-w_i]\} \)
- EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

\[
\begin{array}{cccccccc}
3 & 0 & 0 & 2 & 3 & 4 & 5 & 5 \\
2 & 0 & 0 & 2 & 2 & 4 & 4 & 4 \\
1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6
\end{array}
\]

For $w = 0, \ldots, W$

If ($w_i > w$)

$$M[i, w] = M[i-1, w]$$

Else

$$M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$$

EndIf
Example

- $n=3$, $W=6$, $w_1 = w_2 = 2$ and $w_3 = 3$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

For $w = 0, \ldots, W$

If ($w_i > w$)

- $M[i, w] = M[i-1, w]$

Else

- $M[i, w] = \max\{M[i-1, w], w_i + M[i-1, w-w_i]\}$

EndIf
From values to solutions

- Very similar idea to weighted interval scheduling

\[
\text{OPT}(j,w) = w_j + \text{OPT}(j-1,w-w_j) \quad \text{if } j \text{ in } \mathbb{O} \\
\text{OPT}(j,w) = \text{OPT}(j-1,w) \quad \text{if } \text{not } j \text{ in } \mathbb{O}
\]
Running Time
Running Time

- Similar to weighted interval scheduling.
Running Time

• Similar to weighted interval scheduling.

• We are building up a table $M$ of solutions (instead of an array).
Running Time

• Similar to weighted interval scheduling.

• We are building up a table $M$ of solutions (instead of an array).

• We compute each value $M(i, w)$ of the table in $O(1)$ time using the previous values.
Running Time

- Similar to weighted interval scheduling.
- We are building up a table $M$ of solutions (instead of an array).
- We compute each value $M(i, w)$ of the table in $O(1)$ time using the previous values.
- What is the running time overall?
Running Time

• Similar to weighted interval scheduling.

• We are building up a table $M$ of solutions (instead of an array).

• We compute each value $M(i, w)$ of the table in $O(1)$ time using the previous values.

• What is the running time overall?

• How many entries does the table $M$ have?
Running Time
Running Time

- $\text{SubsetSum}(n, W)$ runs in time $O(nW)$. 
Running Time

- \textbf{SubsetSum}(n,W) runs in time \(O(nW)\).

- Is this a polynomial time algorithm?
Running Time

- $\text{SubsetSum}(n,W)$ runs in time $O(nW)$.

- Is this a polynomial time algorithm?
  - Not quite, because it depends on $W$. 
Our input

$w_{1}^{bin}$  $w_{2}^{bin}$  $w_{3}^{bin}$  $w_{n}^{bin}$

$n$
Running Time

- \textbf{SubsetSum}(n,W) \text{ runs in time } O(nW).

- Is this a polynomial time algorithm?
Running Time

- **SubsetSum**$(n,W)$ runs in time $O(nW)$.

- Is this a polynomial time algorithm?

  - It is *pseudopolynomial*, as it runs in time polynomial in $n$ and the unary representation of $W$. 

Running Time

- \texttt{SubsetSum}(n,W) \ runs \ in \ time \ O(nW).

- Is this a polynomial time algorithm?
  
  - It is \textit{pseudopolynomial}, as it runs in time polynomial in \( n \) and the unary representation of \( W \).

  - It is fairly efficient, if in the number involved in the input are reasonably small.
Should we be happy?
Should we be happy?

- Pseudopolynomial is good in some cases.
Should we be happy?

- Pseudopolynomial is good in some cases.
- But why not polynomial?
Should we be happy?

• Pseudopolynomial is good in some cases.

• But why not polynomial?

• How hard is it to design a polynomial time algorithm for subset sum?
Should we be happy?

- **Pseudopolynomial** is good in some cases.

- But why not polynomial?

- How hard is it to design a polynomial time algorithm for subset sum?
  - Hard enough to justify a reward of 1 million dollars!
Should we be happy?

• Pseudopolynomial is good in some cases.

• But why not polynomial?

• How hard is it to design a polynomial time algorithm for subset sum?

• Hard enough to justify a reward of 1 million dollars!
Should we be happy?

• Pseudopolynomial is good in some cases.

• But why not polynomial?

• How hard is it to design a polynomial time algorithm for subset sum?

  • Hard enough to justify a reward of 1 million dollars!

• Subset sum is NP-hard!
Should we be happy?

- Pseudopolynomial is good in some cases.
- But why not polynomial?
- How hard is it to design a polynomial time algorithm for subset sum?
  - Hard enough to justify a reward of 1 million dollars!
  - Subset sum is NP-hard!
- More about that later on in the course.
The **subset sum** problem
The subset sum problem

We are given a set of $n$ items \{1, 2, … , $n$\}. 
The **subset sum problem**

We are given a set of $n$ items \{1, 2, …, $n$\}.

Each item $i$ has a non-negative weight $w_i$. 
The **subset sum problem**

We are given a set of \( n \) items \( \{1, 2, \ldots, n\} \).

Each item \( i \) has a non-negative weight \( w_i \).

We are given a bound \( W \).
The subset sum problem

We are given a set of $n$ items $\{1, 2, \ldots, n\}$.

Each item $i$ has a non-negative weight $w_i$.

We are given a bound $W$.

**Goal**: Select a subset $S$ of the items such that

$$\sum_{i \in N} w_i \leq W \quad \text{and} \quad \sum_{i \in N} w_i \text{ is maximised.}$$
The (0/1) knapsack problem

We are given a set of $n$ items \{1, 2, … , $n$\}.

Each item $i$ has a non-negative weight $w_i$ and a non-negative value $v_i$.

We are given a bound $W$.

**Goal:** Select a subset $S$ of the items such that

$$\sum_{i \in N} w_i \leq W \quad \text{and} \quad \sum_{i \in N} v_i \text{ is maximised.}$$
The knapsack problem

- The subset sum problem is a specific instance of the knapsack problem (why?)
Design a dynamic programming algorithm for 0/1 knapsack.

**Algorithm SubetSum**($n, W$)

Array $M=[0 \ldots n, 0 \ldots W]$
Initialise $M[0, w] = 0$, for each $w = 0, 1, \ldots, W$

For $i = 1, 2, \ldots, n$
  For $w = 0, \ldots, W$
    If ($w_i > w$)
      $M[i, w] = M[i-1, w]$
    Else
      $M[i, w] = \max\{M[i-1, w] , w_i + M[i-1, w-w_i]\}$
  EndIf

Return $M[n, W]$
3 minute exercise

Design a dynamic programming algorithm for 0/1 knapsack.

Algorithm $\text{SubsetSum}(n,W)$

Array $M = [0 \ldots n, 0 \ldots W]$
Initialise $M[0, w] = 0$, for each $w = 0, 1, \ldots, W$

For $i = 1, 2, \ldots, n$
    For $w = 0, \ldots, W$
        If ($w_i > w$)
            $M[i, w] = M[i-1, w]$
        Else
            $M[i, w] = \max\{M[i-1, w], v_i + M[i-1, w-w_i]\}$
        EndIf

Return $M[n, W]$
Reading

- Kleinberg and Tardos 6.4.
- Roughgarden 16.5.