Subset Sums and Knapsacks

Dynamic Programming - II

Subset Sums

- **Problem:** Given n items {1, 2, … , n}, and each has a given non-negative weight w_i (for i={1, 2, … , n}). We also have a bound W. We would like to select a subset S if the items so that $\sum_{i \in S} w_i \leq W$, and subject to the restriction that $\sum_{i \in S} w_i$ is as large as possible.
- Informally: Consider, we have a single machine that we can run from 0 to W time to process jobs. If we have n jobs to process where each job requires w_i time, to keep the machine as busy as possible, which jobs will you choose? => you want maximum resource utilization.

Will greedy approach work?

Give counter examples to show that greedy will not for:

1) Sort the items by decreasing weight
2) Sort by increasing weight

+2 in Quiz1 for first 3 students to give correct answer (over mail or in class)

Aim: $\sum_i w_i \leq W$ and $\sum_i w_i$ is as large as possible.

Greedy strategy 1: Choose the least weight items first

Item	Weight
1	20
2	7
3	6
4	3

$W = 25$ kg

Chosen items = item4 + item3 + item2
= 3 + 6 + 7 = 16 kg ($\leq$ 25 kg ∴ item1 cannot be added)
↳ as per greedy strategy

Ideal solution: item1 + item4
= 20 + 3 = 23 kg ($\leq$ 25 kg ∴ none of the remaining items can be added).

— Doesn't work

Greedy strategy 2: Choose the maximum weight items first

Item	Weight
1	20
2	7
3	15
4	3

$W = 25$ kg.

Chosen items = item1 + item4
= 20 + 3 = 23 kg
as per greedy strategy

Ideal solution: item2 + item3 + item4
= 7 + 15 + 3 = 25 kg

— Doesn't work

Solution

- For a DP solution, we would like a *small* set of sub-problems that once solved using smaller sub-problems, will help us find the solution to the original problem.
- Important: What should the sub-problems be?

Let us try the approach used previously for WIS

- Two cases:
  - Do not include n-th request

    If $n \notin \mathcal{O}$, then $\text{OPT}(n) = \text{OPT}(n-1)$.

  - Include n-th request

    In WIS, we removed requests that conflicted with request n. Here, we do not have such a requirement. But we do understand that once we choose n-th request requiring w_n time, the time W for the remaining n-1 jobs is now reduced to $W-w_n$
- Requires a better solution.

Better Solution (Informal)

- To find OPT(n), we need
  - OPT(n-1) with W → option 1: item n ∉ O
  - OPT(n-1) with W – w_n → option 2: item n ∈ O

Remaining capacity

Better Solution (formal)

- Assume W is an integer and each request i ={1, 2, … , n} have integer weights w_i
- Using the informal def, we can see that we will have subproblems for each of i ={1, 2, … , n} and each integer 0 <= w <= W.
- We will therefore use OPT(i,w) to denote the value of the optimal solution using a subset of the items {1, …, n} with maximum allowed weight w, that is:

remaining capacity

$$\mathrm{OPT}(i, w) = \max_S \sum_{j \in S} w_j,$$

where $\sum_{j \in S} w_j \leq w$

Solution

- We are looking for OPT(n, W) which is defined as below:

- If $n \notin \mathcal{O}$, then $\text{OPT}(n, W) = \text{OPT}(n-1, W)$, since we can simply ignore item n.
- If $n \in \mathcal{O}$, then $\text{OPT}(n, W) = w_n + \text{OPT}(n-1, W-w_n)$, since we now seek to use the remaining capacity of $W - w_n$ in an optimal way across items $1, 2, \ldots, n-1$.

One further improvement can be made.
We will consider including n-th request only if $w_n <= W$

Solution - Generalised [Updated]

remaining capacity

weight of current item

CANNOT include

$$\text{If } w < w_i \text{ then } \mathrm{OPT}(i, w) = \mathrm{OPT}(i-1, w). \text{ Otherwise}$$

$$\mathrm{OPT}(i, w) = \max(\mathrm{OPT}(i-1, w), w_i + \mathrm{OPT}(i-1, w-w_i)).$$

May / May not include

Next step: Write the algorithm that builds up the array for all OPT(i,w) while computing each of them once.

Writing the algorithm

- Few questions before your begin:
  - What will be the dimensions for the array M storing optimal solution to all problems?
  - What recurrence will you use
  - What will you return?

```
m[0]=0
  For j=1,2,....,n
    m[j]=max (p_j + m[p(j)] ,  m[j-1])
  End for
```

Take help from this

Writing the algorithm

- Few questions before your begin:
  - What will be the dimensions for the array M storing optimal solution to all problems? → n+1 x W+1
  - What recurrence will you use
  - What will you return? → last cell of 2-D array M i.e M[n,W]

```
m[0]=0
  For j=1,2,....,n
    m[j]=max (p_j + m[p(j)] ,  m[j-1])
  End for
```

Take help from this

Algorithm

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

Example Subset Sum

- W = 6, n=3
- w_1=2, w_2=2, w_3=3
- Because the recursion is:

$$\textit{If } w < w_i \textit{ then } \text{OPT}(i, w) = \text{OPT}(i-1, w). \textit{ Otherwise}$$

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

- We need to find M[n.W]. Therefore, M will have n+1 (0 to n) rows and W+1 (0 to W) columns

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

→ Reason?

3							
2							
1							
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

i=1, w=0

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

3							
2							
1	**0**						
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=1, w=1

3							
2							
1	**0**	**0**					
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=1, w=2

3							
2							
1	**0**	**0**	**2**				
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

i=1, w=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

3							
2							
1	**0**	**0**	**2**	**2**			
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

i=1, w=4

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

3							
2							
1	**0**	**0**	**2**	**2**	**2**		
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=1, w=5

3							
2							
1	**0**	**0**	**2**	**2**	**2**	**2**	
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=1, w=6

3							
2							
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=0

3							
2	**0**						
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=1

3							
2	**0**	**0**					
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=2

3							
2	**0**	**0**	**2**				
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=3

3							
2	**0**	**0**	**2**	**2**			
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=4

3							
2	**0**	**0**	**2**	**2**	**4**		
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=5

3							
2	**0**	**0**	**2**	**2**	**4**	**4**	
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=2, w=6

3							
2	**0**	**0**	**2**	**2**	**4**	**4**	**4**
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=0

3	**0**						
2	**0**	**0**	**2**	**2**	**4**	**4**	**4**
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=1

3	**0**	**0**					
2	**0**	**0**	**2**	**2**	**4**	**4**	**4**
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=2

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**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=3

3	**0**	**0**	**2**	**3**			
2	**0**	**0**	**2**	**2**	**4**	**4**	**4**
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=4

3	**0**	**0**	**2**	**3**	**4**		
2	**0**	**0**	**2**	**2**	**4**	**4**	**4**
1	**0**	**0**	**2**	**2**	**2**	**2**	**2**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=5

3	**0**	**0**	**2**	**3**	**4**	**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**
	0	1	2	3	4	5	6

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w - w_i)).$$

W = 6, n=3

w_1=2, w_2=2, w_3=3

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

i=3, w=6

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**
	0	1	2	3	4	5	6

Time Complexity

```
Subset-Sum(n, W)
  Array M[0...n, 0...W]
  Initialize M[0, w] = 0 for each w = 0, 1, ..., W
  For i = 1, 2, ..., n
     For w = 0, ..., W
        Use the recurrence (6.8) to compute M[i, w]
     Endfor
  Endfor
  Return M[n, W]
```

Runs for n*W times

Another intuitive way to think about TC: no. of cells to be filled? (n+1*W+1) = O(nW)

Classwork Problem.

If $w < w_i$ *then* $\text{OPT}(i, w) = \text{OPT}(i-1, w)$. *Otherwise*

$$\text{OPT}(i, w) = \max(\text{OPT}(i-1, w), w_i + \text{OPT}(i-1, w-w_i)).$$

Try to create a table for W = 14 and n = 5 items of sizes/weights $w_1 = 7$, $w_2 = 3$, $w_3 = 2$, $w_4 = 5$, $w_5 = 8$.

Find optimal value that is OPT(5,14).

item	weight
1	2
2	3
3	1
4	4

W = 5 kg.

OPT(n,w) = 5

classwork

Solution to homework problem

W = 14, n = 5
w_1= 7, w_2 = 3, w_3 = 2, w_4 = 5, w_5 = 8.

5	**0**	**0**	**2**	**3**	**3**	**5**	**5**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
4	**0**	**0**	**2**	**3**	**3**	**5**	**5**	**7**	**8**	**9**	**10**	**10**	**12**	**12**	**14**
3	**0**	**0**	**2**	**3**	**3**	**5**	**5**	**7**	**7**	**9**	**10**	**10**	**12**	**12**	**12**
2	**0**	**0**	**0**	**3**	**3**	**3**	**3**	**7**	**7**	**7**	**10**	**10**	**10**	**10**	**10**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Find optimal solution

- Optimal value != optimal solution in this case.
- To find optimal solution, trace back through M as done in WIS.

```
Find-Solution(i,w)
     if i = 0 then
          return nothing
     else
          if w < w_i
               Find-Solution(i-1, w)
          else if w_i + M[i-1, w-w_i] > M[i-1,w]
               output i and Find-Solution(i-1, w-w_i)
          else
               Find-Solution(i-1, w)
          Endif
     Endif
```

Time complexity?

Scan the table. O(nW)

Programming Task.

1. Read input data from file
2. Find optimal value using the recurrence relations and stored values in M.
3. Display M
4. Find the optimal solution using M

Discussion → tomorrow (04-02-2022)