Binary Knapsack

Binary Knapsack

Input

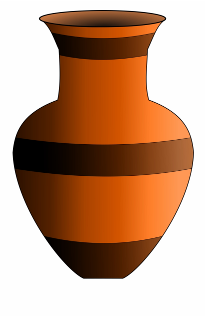
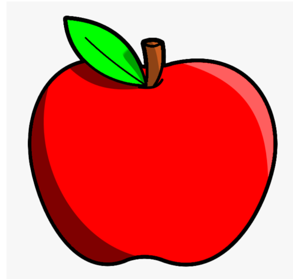
- You are given a collection $\mathcal{I}$ of n items indexed from 1 to n.
- Item i has a weight $w_i : \mathbb{N}^+$ and a value $v_i \in \mathbb{N}^+$.
- You can carry an overall weight of at most $W \in \mathbb{N}$.

Goal

Find a subset of $S \subset \mathcal{I}$ such that:

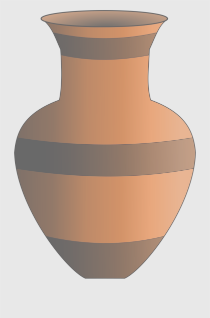
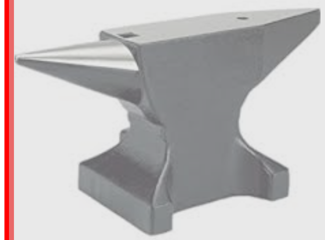
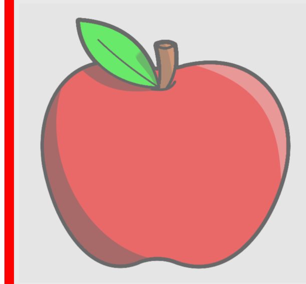
- Its overall weight $w(S) = \sum_{i \in \mathcal{I}} w_i$ is at most W; and
- Its overall value $v(S) = \sum_{i \in \mathcal{I}} v_i$ is maximized.

Example

1	2	3	4	5	6
					
$w_1 = 12$	$w_2 = 1$	$w_3 = 19$	$w_4 = 14$	$w_5 = 4$	$w_6 = 3$
$v_1 = 15$	$v_2 = 1$	$v_3 = 8$	$v_4 = 20$	$v_5 = 10$	$v_6 = 4$

Maximum Weight: 20

Example

1	2	3	4	5	6
					
$w_1 = 12$	$w_2 = 1$	$w_3 = 19$	$w_4 = 14$	$w_5 = 4$	$w_6 = 3$
$v_1 = 15$	$v_2 = 1$	$v_3 = 8$	$v_4 = 20$	$v_5 = 10$	$v_6 = 4$

Maximum Weight: 20

$w(S) = 19$

$v(S) = 31$

A Dynamic Programming Algorithm

Subproblem definition:

$OPT[i, x]$ = Maximum overall value $v(S)$ among all subsets S of $\{1, \ldots, i\}$ such that $w(S) \leq x$.

Base case:

For any $x \geq 0$, $OPT[0, x] = 0$.

A Dynamic Programming Algorithm

Recursive Formula

- Either we ignore item i...

$$OPT[i, x] = OPT[i-1, x]$$

A Dynamic Programming Algorithm

Recursive Formula

- Either we ignore item i...

$$OPT[i, x] = OPT[i-1, x]$$

- Or we select item i and we can still carry a weight of $x - w_i$

$$OPT[i, x] = v_i + OPT[i-1, x - w_i]$$

This is only viable if $x \geq w_i$!

A Dynamic Programming Algorithm

Recursive Formula

- Either we ignore item i...

$$OPT[i,x] = OPT[i-1,x]$$

- Or we select item i and we can still carry a weight of $x - w_i$

$$OPT[i,x] = v_i + OPT[i-1,x-w_i]$$

This is only viable if $x \geq w_i$!

$$OPT[i,x] = \begin{cases} OPT[i-1,x] & \text{if } x < w_i \\ \max\begin{cases} OPT[i-1,x] \\ v_i + OPT[i-1,x-w_i] \end{cases} & \text{if } x \geq w_i \end{cases}$$

Time Complexity

- $\Theta(n \cdot W)$ subproblems
- Optimal solution in $OPT[n, W]$
- Each problem can be solved in constant time
- **Overall time:** $\Theta(n \cdot W)$

Time Complexity

- $\Theta(n \cdot W)$ subproblems
- Optimal solution in $OPT[n, W]$
- Each problem can be solved in constant time
- **Overall time:** $\Theta(n \cdot W)$

Is this a polynomial-time algorithm?

Time Complexity

- $\Theta(n \cdot W)$ subproblems
- Optimal solution in $OPT[n, W]$
- Each problem can be solved in constant time
- **Overall time:** $\Theta(n \cdot W)$

Is this a polynomial-time algorithm?

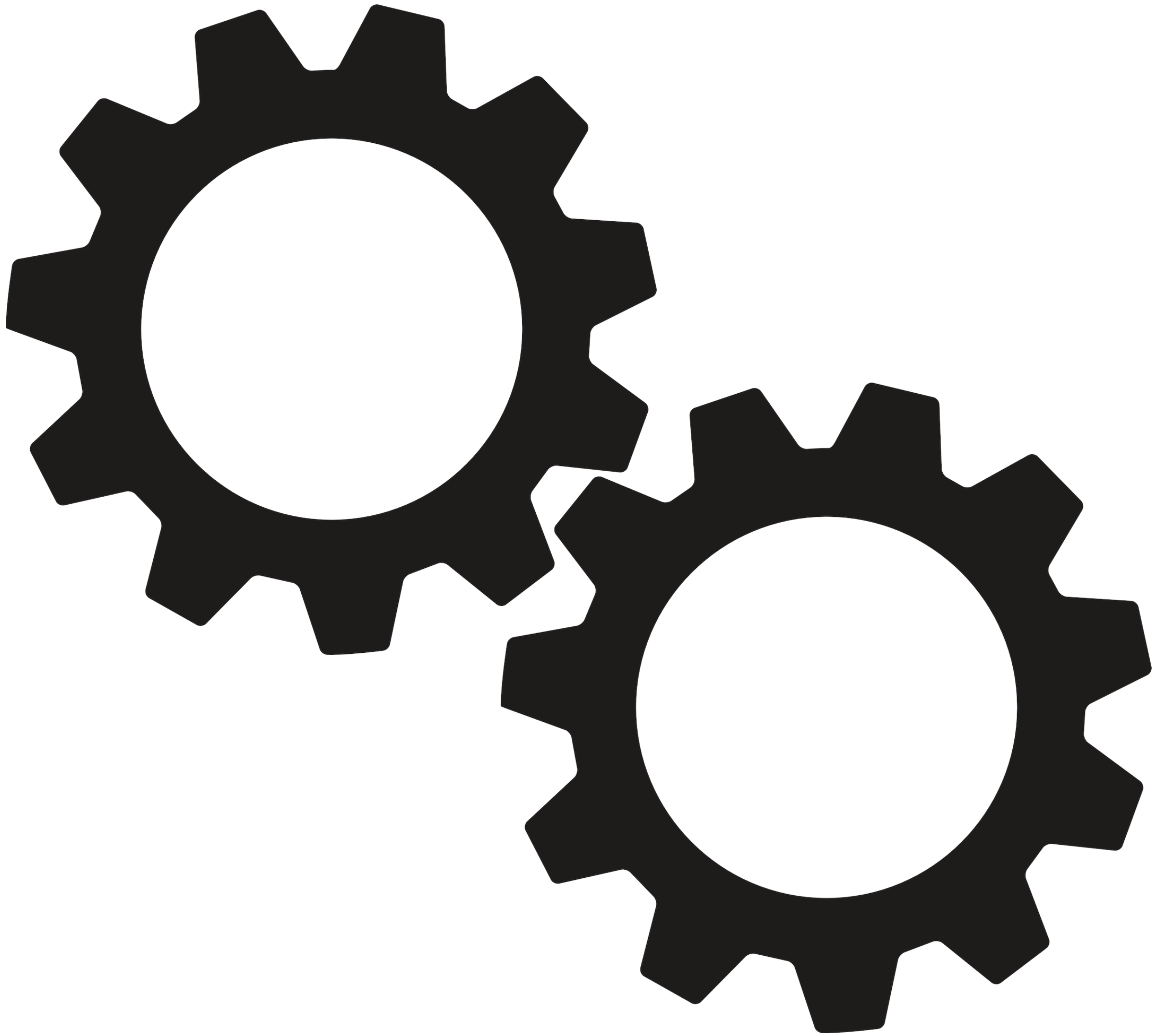
NO!

The input size is $O(n(\log W + \log V))$ where $V = \max_i v_i$

Choose, e.g., $W = 2^n$.

Small maximum value

Can we do better if W is large (e.g., 2^n) and $V = \max_i v_i$ is small?



Small maximum value

Can we do better if W is large (e.g., 2^n) and $V = \max_i v_i$ is small?

Subproblem definition (sketch):

$OPT[i, x] =$ Minimim overall weight $w(S)$ among all subsets S of $\{1, \ldots, i\}$ such that $v(S) \geq x$.

Base case:

$OPT[0, 0] = 0.$

For any $x > 0$, $OPT[0, x] = +\infty$.

Small maximum value

Can we do better if W is large (e.g., 2^n) and $V = \max_i v_i$ is small?

Subproblem definition (sketch):

$OPT[i, x] =$ Minimim overall weight $w(S)$ among all subsets S of $\{1, \ldots, i\}$ such that $v(S) \geq x$.

Base case:

$OPT[0, 0] = 0.$

For any $x > 0$, $OPT[0, x] = +\infty$.

Use "$+\infty$" to encode "not feasible"

Small maximum value

Recursive Formula

- Either we ignore item i...

$$OPT[i, x] = OPT[i-1, x]$$

Small maximum value

Recursive Formula

- Either we ignore item i...

$$OPT[i,x] = OPT[i-1,x]$$

- Or we select item i and we need to gain an additional value of $x - v_i$

$$OPT[i,x] = w_i + OPT[i-1, \max\{x - v_i, 0\}]$$

Small maximum value

Recursive Formula

- Either we ignore item i...

$$OPT[i,x] = OPT[i-1,x]$$

- Or we select item i and we need to gain an additional value of $x - v_i$

$$OPT[i,x] = w_i + OPT[i-1, \max\{x - v_i, 0\}]$$

$$OPT[i,x] = \min \begin{cases} OPT[i-1,x] \\ w_i + OPT[i-1, \max\{x - v_i, 0\}] \end{cases}$$

Optimal Solution: $V^* = \max_{x : OPT[n,x] \leq W} x$

Small maximum value

Optimal Solution: $V^* = \max_{x : OPT[n,x] \leq W} x$

Note: $OPT[n, x]$ is monotonically non-decreasing w.r.t. x

Small maximum value

Optimal Solution: $V^* = \max_{x : OPT[n,x] \leq W} x$

Note: $OPT[n, x]$ is monotonically non-decreasing w.r.t. x

Order of subproblems:

For each $x = 1, 2, \ldots$

  Compute $OPT[1, x], OPT[2, x], \ldots, OPT[n, x]$

Stop computing subproblems as a soon as $OPT[n, x] > W$.

Small maximum value

Optimal Solution: $V^* = \max_{x : OPT[n,x] \leq W} x$

Note: $OPT[n, x]$ is monotonically non-decreasing w.r.t. x

Order of subproblems:

For each $x = 1, 2, \ldots$

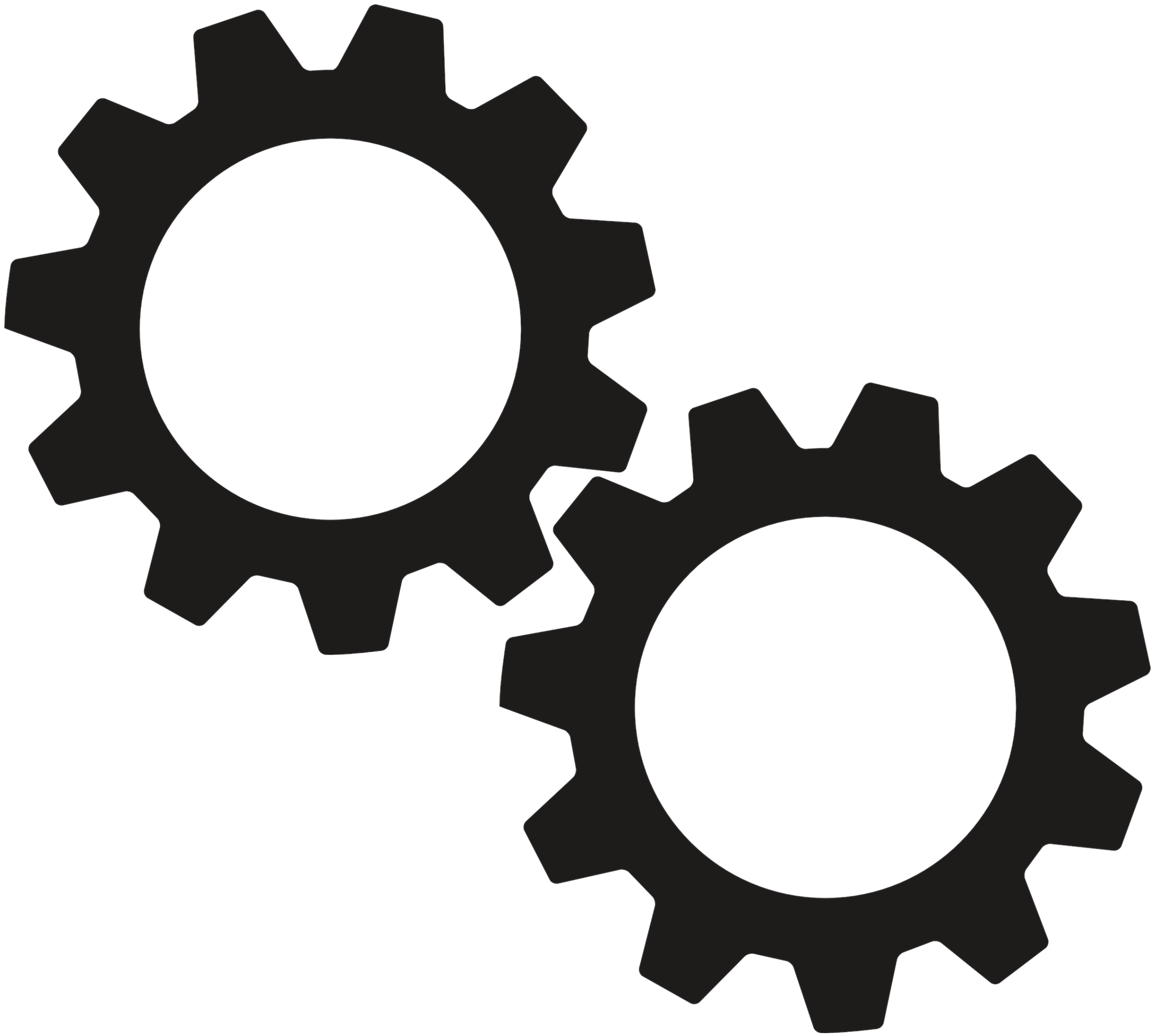
Compute $OPT[1, x], OPT[2, x], \ldots, OPT[n, x]$

Stop computing subproblems as a soon as $OPT[n, x] > W$.

Time complexity

- $\Theta(n \cdot V^*)$ subproblems
- Each problem can be solved in constant time
- **Overall time:** $\Theta(n \cdot V^*) = O(n^2 V)$ where $V = \max_i v_i$

What if there are few items?



A Split & List Algorithm

- **Split:** let $S_1 = \{1, \ldots, \lceil n/2 \rceil\}$ and $S_2 = S \setminus S_1$.

A Split & List Algorithm

- **Split:** let $S_1 = \{1, \ldots, \lceil n/2 \rceil\}$ and $S_2 = S \setminus S_1$.
- **List:** Let L_1 (resp. L_2) be the list of all the pairs $(w(X), v(X))$ for each $X \subseteq S_1$ (resp. $X \subseteq S_2$).

A Split & List Algorithm

- **Split:** let $S_1 = \{1, \ldots, \lceil n/2 \rceil\}$ and $S_2 = S \setminus S_1$.
- **List:** Let L_1 (resp. L_2) be the list of all the pairs $(w(X), v(X))$ for each $X \subseteq S_1$ (resp. $X \subseteq S_2$).
- Sort L_2 in lexicographic order.
- For each $(w, \cdot) \in L_2$, add a pair (w, v) in L'_2, where $v = \max\{v' \,:\, (w', v') \in L_2, w' \leq w\}$

A Split & List Algorithm

- **Split:** let $S_1 = \{1, \ldots, \lceil n/2 \rceil\}$ and $S_2 = S \setminus S_1$.
- **List:** Let L_1 (resp. L_2) be the list of all the pairs $(w(X), v(X))$ for each $X \subseteq S_1$ (resp. $X \subseteq S_2$).
- Sort L_2 in lexicographic order.
- For each $(w, \cdot) \in L_2$, add a pair (w, v) in L'_2, where $v = \max\{v' : (w', v') \in L_2, w' \leq w\}$
- For each pair $(w, v) \in L_1$ such that $w \leq W$:
  - Binary search for the last pair $(w', v') \in L'_2$ for which $w' \leq W - w$, if any.
  - If w' exists, $v + v'$ is the value of a candidate solution.

A Split & List Algorithm

- **Split:** let $S_1 = \{1, \ldots, \lceil n/2 \rceil\}$ and $S_2 = S \setminus S_1$.

- **List:** Let L_1 (resp. L_2) be the list of all the pairs $(w(X), v(X))$ for each $X \subseteq S_1$ (resp. $X \subseteq S_2$).

- Sort L_2 in lexicographic order.

- For each $(w, \cdot) \in L_2$, add a pair (w, v) in L_2', where $v = \max\{v' : (w', v') \in L_2, w' \leq w\}$

- For each pair $(w, v) \in L_1$ such that $w \leq W$:
  - Binary search for the last pair $(w', v') \in L_2'$ for which $w' \leq W - w$, if any.
  - If w' exists, $v + v'$ is the value of a candidate solution.

- **Return:** Best candidate solution, if any.

A Split & List Algorithm

- **Split:** let $S_1 = \{1, \ldots, \lceil n/2 \rceil\}$ and $S_2 = S \setminus S_1$. $O(n)$

- **List:** Let L_1 (resp. L_2) be the list of all the pairs $(w(X), v(X))$ for each $X \subseteq S_1$ (resp. $X \subseteq S_2$). $O(2^{\frac{n}{2}})$

- Sort L_2 in lexicographic order. $O(n \cdot 2^{\frac{n}{2}})$

- For each $(w, \cdot) \in L_2$, add a pair (w, v) in L_2', where $v = \max\{v' : (w', v') \in L_2, w' \leq w\}$ $O(2^{\frac{n}{2}})$

- For each pair $(w, v) \in L_1$ such that $w \leq W$: $O(2^{\frac{n}{2}})$

  - Binary search for the last pair $(w', v') \in L_2'$ for which $w' \leq W - w$, if any. $O(n)$

  - If w' exists, $v + v'$ is the value of a candidate solution.

- **Return:** Best candidate solution, if any.

Recap

Three Algorithms

- **Dynamic programming:** parameterize weights, store values.

$$O(nW)$$

- **Dynamic programming:** parameterize values, store weights.

$$O(nV^*) = O(n^2V)$$

- **Split & List:**

$$O(n2^{\frac{n}{2}})$$