CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 12: Greedy Algorithms: Unit-Weight Knapsack

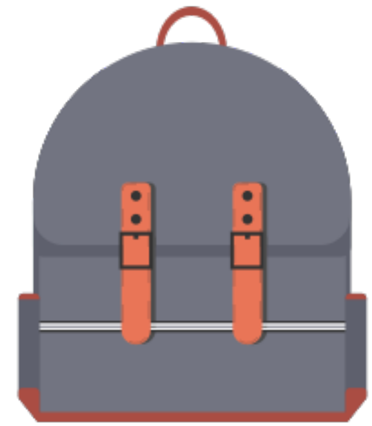
Greedy Algorithms: Going Beyond Checking All Options

Dynamic Programming vs Greedy Algorithms

- Dynamic programming:
  - Write a recursive formula to “check (almost) all” options
  - Store intermediate answers to speedup computation
- Greedy algorithms:
  - Only search a small set of options
  - Greedily pick a choice and ignore alternatives

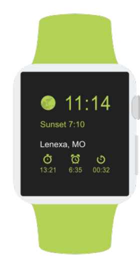
The Knapsack Problem

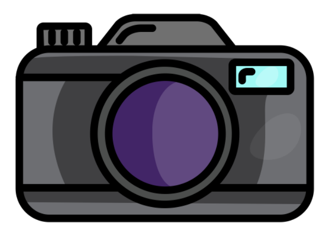
- **Input:**
  - A collection of n items with value v_i and weight w_i
  - A knapsack of size W



size: 35

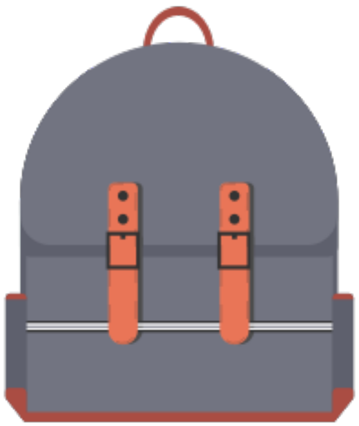
value:	**1000**	**5**	**50**	**5**	**100**
weight:	**25**	**5**	**8**	**5**	**15**





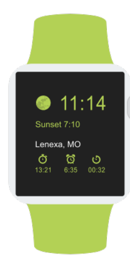
The Knapsack Problem

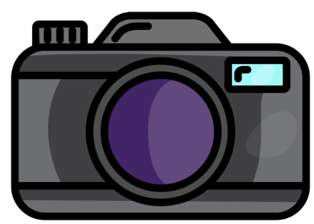
- **Output:**
  - The maximum value we can get by picking a subset S of items
  - The total weight of S should be less than Knapsack size



size: 35

value:	**1000**	**5**	**50**	**5**	**100**
weight:	**25**	**5**	**8**	**5**	**15**





The Unit-Weight Knapsack Problem

- **Input:**
  - and weight 1
  - A collection of n items with value v_i ~~and weight w_i~~
  - A knapsack of size W
- **Output:**
  - The maximum value we can get by picking a subset S of items
  - The total weight of S should be less than Knapsack size
  - Alternatively, pick W items with largest value

Dynamic Programming?

- IF $W \geq n$ pick all the items
- Otherwise run dynamic programming for original knapsack

Dynamic Programming?

- IF $W \geq n$ pick all the items
- Otherwise run dynamic programming for original knapsack
- Correctness?

Dynamic Programming?

- IF $W \geq n$ pick all the items
- Otherwise run dynamic programming for original knapsack

- Correctness?
- Runtime:
  - $O(nW)$ when $W < n$ so $O(n^2)$ time

Dynamic Programming?

- IF $W \geq n$ pick all the items
- Otherwise run dynamic programming for original knapsack
- Correctness?
- Runtime:
  - $O(nW)$ when $W < n$ so $O(n^2)$ time
  - Can we solve the problem faster?

Greedy Algorithm?

- A “Greedy” Observation:
  - An item of price 100 is definitely better than an item of price 10 in every possible way
  - Why? Both have the same weight

Greedy Algorithm?

- A "Greedy" Observation:
  - An item of price 100 is definitely better than an item of price 10 in every possible way
  - Why? Both have the same weight
- More generally,
  - If price of item i is more than item j, then we should first pick i before considering picking j

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- **Example:**

Price	3	5	2	1	7	8	4	9

$W = 4$

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- **Example:**

Price	3	5	2	1	7	8	4	9

$W = 3$

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- **Example:**

Price	3	5	2	1	7	8	4	9

W = 2

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- **Example:**

Price	3	5	2	1	7	8	4	9

W = 1

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- **Example:**

Price	3	5	2	1	7	8	4	9

$W = 0$

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- **Example:**

Price	3	5	2	1	7	8	4	9

We pick items 2,5,6, and 8 with a total price of 29

Proof of Correctness

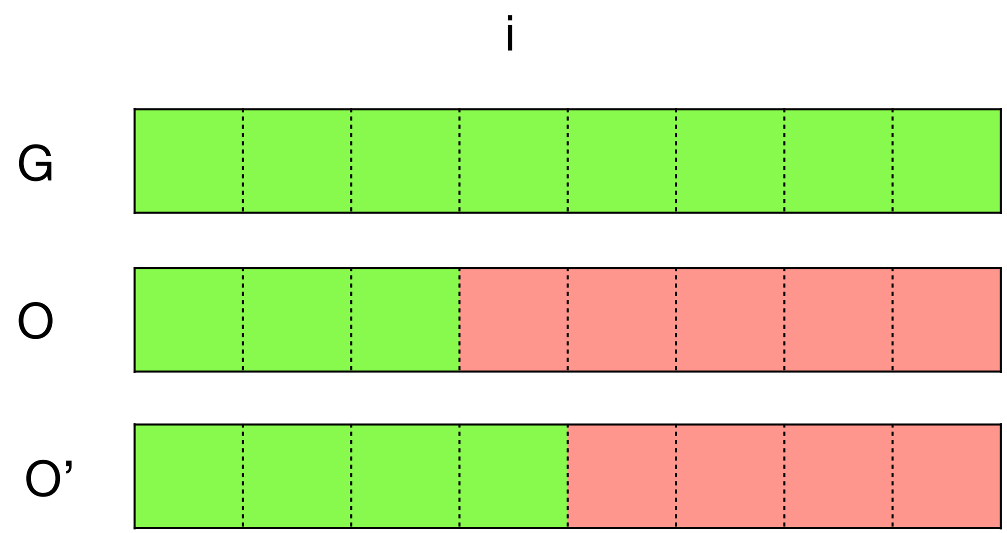
- Consider our greedy solution called $G = \{g_1, g_2, \ldots, g_W\}$
- Consider an optimal solution called $O = \{o_1, o_2, \ldots, o_W\}$
- We can assume:
  - $v_{g_1} \geq v_{g_2} \geq \cdots \geq v_{g_W}$
  - $v_{o_1} \geq v_{o_2} \geq \cdots \geq v_{o_W}$

Proof of Correctness

- If $O = G$ we are done: greedy is also an optimal solution
- If $O \neq G$ find the first index i such that $g_i \neq o_i$:
  - $g_1 = o_1$, $\dots$, $g_{i-1} = o_{i-1}$

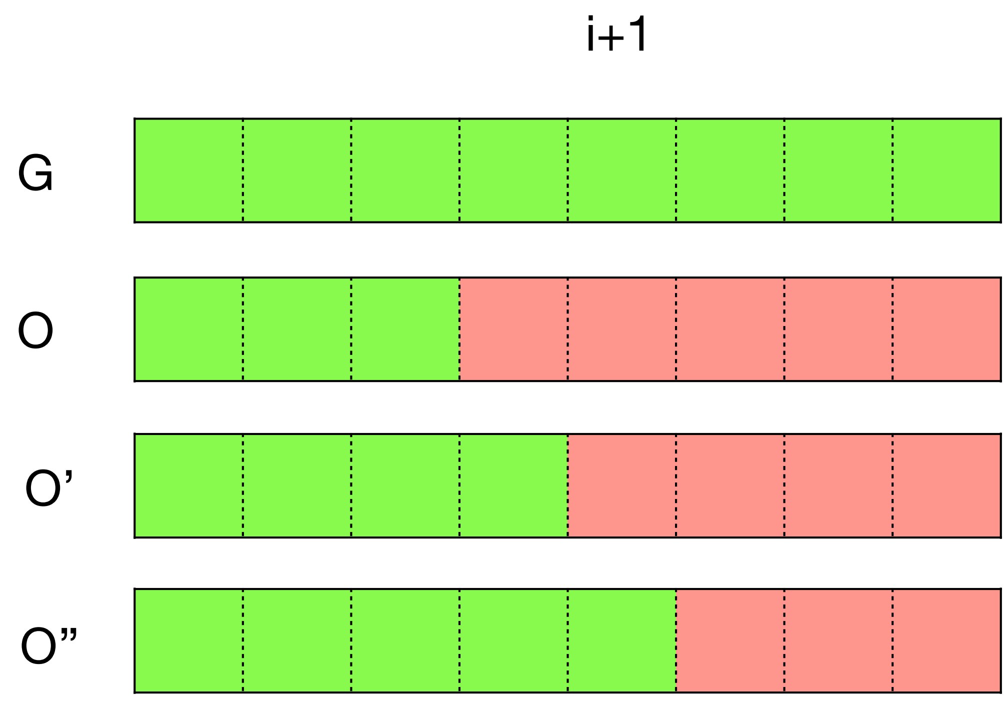
Proof of Correctness

- So we found another optimal solution "one step closer" to G
- We can repeat this step to i+1 until n to find an optimal solution which is identical to G



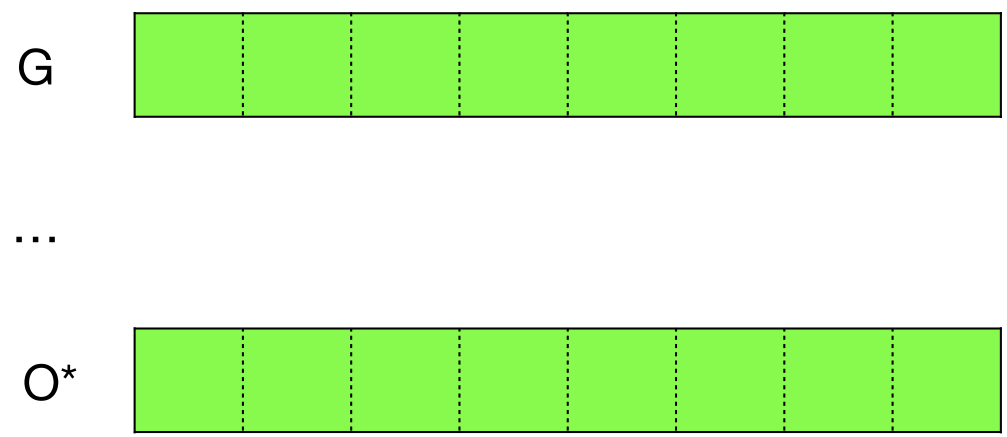
Proof of Correctness

- So we found another optimal solution "one step closer" to G
- We can repeat this step to i+1 until n to find an optimal solution which is identical to G



Proof of Correctness

- So we found another optimal solution “one step closer” to G
- We can repeat this step to i+1 until n to find an optimal solution which is identical to G



Exchange Argument

- We proved the correctness of our greedy algorithm
- This type of argument is called an exchange argument
  - Start from an optimal solution which can be "far from" greedy
  - Exchange decisions of the optimal solution, one at a time, to make it "closer to" greedy while keeping optimality
  - Once we changed all of optimal to greedy, we obtain that the greedy solution is also optimal

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1

- Runtime?

Greedy Algorithm

- Greedy algorithm:
  - While knapsack is not full:
    - Pick the item i with the largest price among remaining items
    - Remove item i from list of items and decrease the capacity of knapsack by 1
- Runtime?
  - We have to specify the algorithm more

Implementation

- While knapsack is not full:
  - Pick the item i with the largest price among remaining items
  - Remove item i from list of items and decrease the capacity of knapsack by 1

- Naive implementation:
- Add all items to a linked-list L
- For $i=1$ to W:
  - Find the maximum of L
  - Add this item to the solution and remove it from L

Implementation

- While knapsack is not full:
  - Pick the item i with the largest price among remaining items
  - Remove item i from list of items and decrease the capacity of knapsack by 1

- Naive implementation:
- Add all items to a linked-list L
- For i=1 to W:
  - Find the maximum of L
  - Add this item to the solution and remove it from L

Runtime is $O(nW)$ this way which can become $O(n^2)$ as before

Implementation

- While knapsack is not full:
  - Pick the item i with the largest price among remaining items
  - Remove item i from list of items and decrease the capacity of knapsack by 1

- Better implementation:
- Sort the items based on their prices in decreasing (non-increasing) order
- Return the first W items in the sorted array

Implementation

- While knapsack is not full:
  - Pick the item i with the largest price among remaining items
  - Remove item i from list of items and decrease the capacity of knapsack by 1

- Better implementation:
- Sort the items based on their prices in decreasing (non-increasing) order
- Return the first W items in the sorted array

Runtime is $O(n \log n)$ this way

Pattern of Greedy Algorithms

Pattern of Greedy Algorithms

1. See if you find a way of ignoring some of available options
   - Not a proof, just intuition
2. Design your greedy algorithm based on this observation
3. **Main step:** Prove the correctness of your algorithm
   - Typically (but not always) based on an exchange argument
4. What if you failed to prove the correctness?
   - Then your algorithm is (most likely) wrong and you should go back to step 1 or 2 to change the algorithm.
5. What if you keep failing? Not every problem has a greedy solution

The Job Scheduling Problem

The Job Scheduling Problem

- **Input:**
  - A collection of n computing jobs each with a length $L[i]$
  - A single processor that can compute job i in $L[i]$ time
- **Output:**
  - On ordering π of the jobs with minimal total delay
  - $$delay(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{i} L[\pi(j)]$$

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

- A possible ordering $\pi = (1,2,3,4,5,6,7,8)$
  - i.e., the same ordering as the input

6	1	2	3	2	1	1	4

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

- A possible ordering $\pi = (1,2,3,4,5,6,7,8)$
  - i.e., the same ordering as the input
- Job 1 has to wait 6 unit
- Job 2 has to wait 7 unit

6	1	2	3	2	1	1	4

- Job 3 has to wait 9 unit
- …
- Total delay is 6+7+9+12+14+15+16+20 = 99 units

The Job Scheduling Problem: Example

- **Input:**

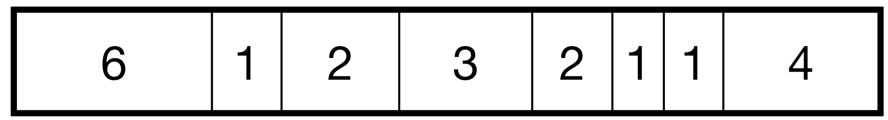
6	1	2	3	2	1	1	4

- Another possible ordering $\pi = (3,2,1,7,5,6,4,8)$

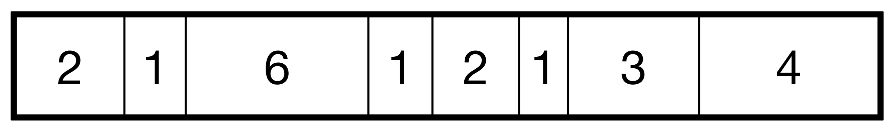
2	1	6	1	2	1	3	4

The Job Scheduling Problem: Example

- **Input:**



- Another possible ordering $\pi = (3,2,1,7,5,6,4,8)$
  - Job 3 has to wait 2 unit
  - Job 2 has to wait 3 unit
  - Job 1 has to wait 9 unit



  - …
  - Total delay is 2+3+9+10+12+13+16+20 = 85 units

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

- Yet another possible ordering $\pi = (2,6,7,5,3,4,8,1)$

1	1	1	2	2	3	4	6

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

- Yet another possible ordering $\pi = (2,6,7,5,3,4,8,1)$
  - Job 2 has to wait 1 unit
  - Job 6 has to wait 2 unit
  - Job 7 has to wait 3 unit

1	1	1	2	2	3	4	6

  - …
  - Total delay is 1+2+3+5+7+10+14+20 = 62 units

Greedy Algorithm?

- A “Greedy” Observation:
  - Suppose we have two jobs next to each other in the output ordering π
  - If we flip the order of these two jobs, the delay for remaining jobs does not change
  - Delay of first job increases by length of second, while delay of second job decreases by length of first