COMP 382: Reasoning about Algorithms

Greedy Algorithms: Car Refueling Problem, Job Scheduling

Prof. Maryam Aliakbarpour

co-instructors: Prof. Anjum Chida & Prof. Konstantinos Mamouras

October 16, 2025

Today's Lecture

1. **Introduction to Greedy Algorithms**
2. **Car Refueling Problem**
3. **Job Scheduling Problem**
4. **Huffman Encoding**

Reading:

- Chapter 5 of the Algorithms book [Dasgupta et al., 2006]
- Chapter 13 of [Roughgarden, 2022]
- Chapter 14 of [Roughgarden, 2022]

1. Introduction to Greedy Algorithms

What is a Greedy Algorithm?

- Solves an **optimization problem**:
  - Maximize or minimize an objective
  - Subject to constraints
- Builds the solution **step by step**
- At each step:
  - Choose what looks **best locally**
  - Once a choice is made, it is not changed later (**irrevocable**).

Key question: *Do locally optimal choices lead to a global optimum?*

A Classic Problem: Making Change

The Problem

How do you make change for a customer using the **fewest** possible coins? The coin denominations are: 25¢, 10¢, 5¢, 1¢.

A Classic Problem: Making Change

The Problem

How do you make change for a customer using the **fewest** possible coins? The coin denominations are: 25¢, 10¢, 5¢, 1¢.

The Greedy Strategy

At each step, pick the largest denomination coin that is less than or equal to the remaining amount owed.

Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.

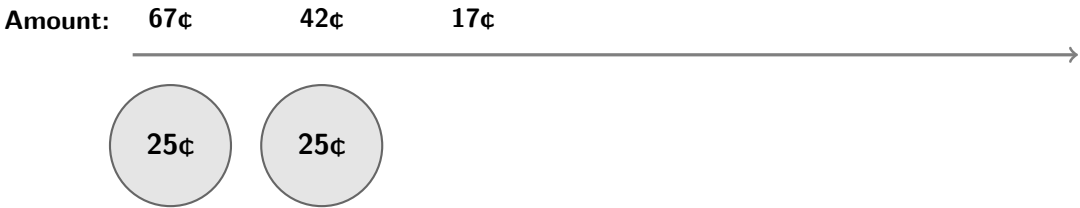
Amount: **67¢**

Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.

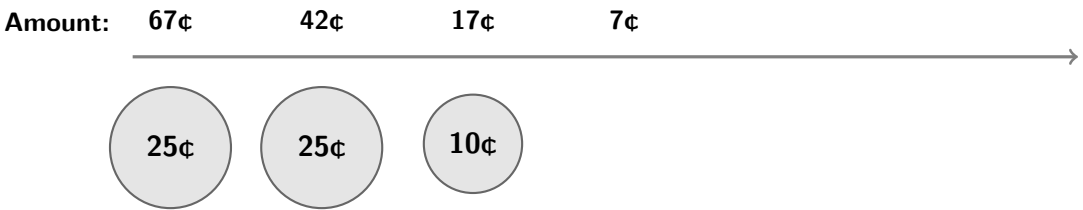
Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.



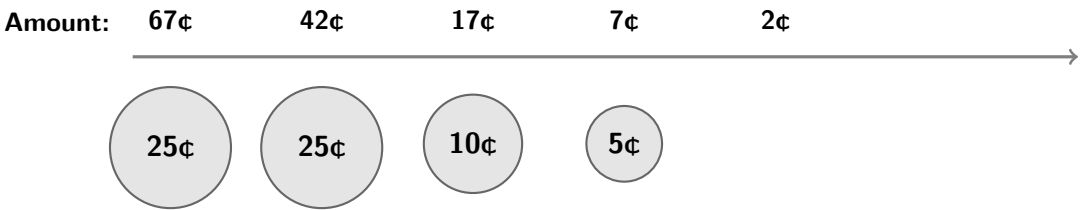
Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.



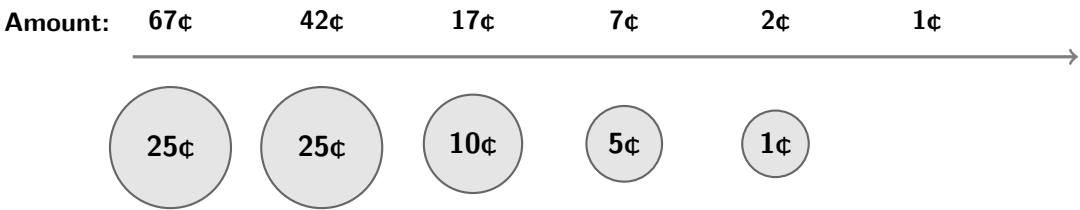
Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.



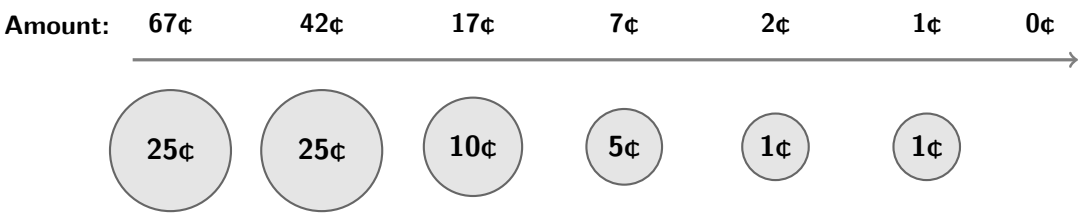
Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.



Example: Change for 67¢

The Goal: Make change for **67¢** by always choosing the largest possible coin at each step.



Is the Greedy Approach Always Optimal?

For Standard U.S. Coins: Yes.

The greedy strategy of always picking the largest coin denomination (25¢, 10¢, 5¢, 1¢) is proven to yield the optimal solution (the minimum number of coins).

Is the Greedy Approach Always Optimal?

For Standard U.S. Coins: Yes.

The greedy strategy of always picking the largest coin denomination (25¢, 10¢, 5¢, 1¢) is proven to yield the optimal solution (the minimum number of coins).

Exercise: Prove this optimality for the U.S. coin system.

Is the Greedy Approach Always Optimal?

For Arbitrary Coin Systems: No.

A greedy approach does not guarantee a globally optimal solution for all sets of coin denominations. The "locally best" choice can prevent finding the "globally best" solution.

Is the Greedy Approach Always Optimal?

For Arbitrary Coin Systems: No.

A greedy approach does not guarantee a globally optimal solution for all sets of coin denominations. The "locally best" choice can prevent finding the "globally best" solution.

Counterexample: Consider a currency system with coins valued at **4¢, 3¢, and 1¢**. We want to make change for an amount of = 6¢.

Is the Greedy Approach Always Optimal?

For Arbitrary Coin Systems: No.

A greedy approach does not guarantee a globally optimal solution for all sets of coin denominations. The "locally best" choice can prevent finding the "globally best" solution.

Counterexample: Consider a currency system with coins valued at **4¢, 3¢, and 1¢**. We want to make change for an amount of = 6¢.

Greedy Solution

- Take **4¢** (Remaining: 2¢)
- Take **1¢** (Remaining: 1¢)
- Take **1¢** (Remaining: 0¢)

Total: 3 coins

Is the Greedy Approach Always Optimal?

For Arbitrary Coin Systems: No.

A greedy approach does not guarantee a globally optimal solution for all sets of coin denominations. The "locally best" choice can prevent finding the "globally best" solution.

Counterexample: Consider a currency system with coins valued at **4¢, 3¢, and 1¢**. We want to make change for an amount of = 6¢.

Greedy Solution

- Take **4¢** (Remaining: 2¢)
- Take **1¢** (Remaining: 1¢)
- Take **1¢** (Remaining: 0¢)

Total: 3 coins

Optimal Solution

- Take **3¢** (Remaining: 3¢)
- Take **3¢** (Remaining: 0¢)

Total: 2 coins

The Power and the Pitfall

The Power

- Often simple to design and understand.

The Power and the Pitfall

The Power

- Often simple to design and understand.
- Usually very efficient and fast.

The Power and the Pitfall

The Power

- Often simple to design and understand.
- Usually very efficient and fast.

The Pitfall

- The "obvious" greedy choice can fail to produce a globally optimal solution.

The Power and the Pitfall

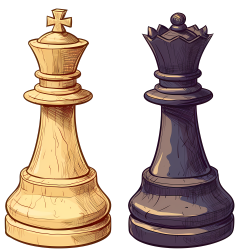
The Power

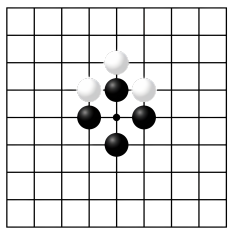
- Often simple to design and understand.
- Usually very efficient and fast.

The Pitfall

- The “obvious” greedy choice can fail to produce a globally optimal solution.

  Greedy algorithms are poorly suited for complex strategic games, e.g., chess and go.

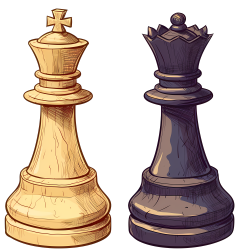


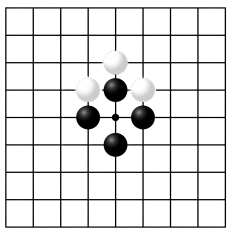


The Power and the Pitfall

The Power

- Often simple to design and understand.
- Usually very efficient and fast.





The Pitfall

- The "obvious" greedy choice can fail to produce a globally optimal solution.

  Greedy algorithms are poorly suited for complex strategic games, e.g., chess and go.

- The most difficult part is **proving** that the algorithm is actually producing an optimal solution.

2. Car Refueling Problem

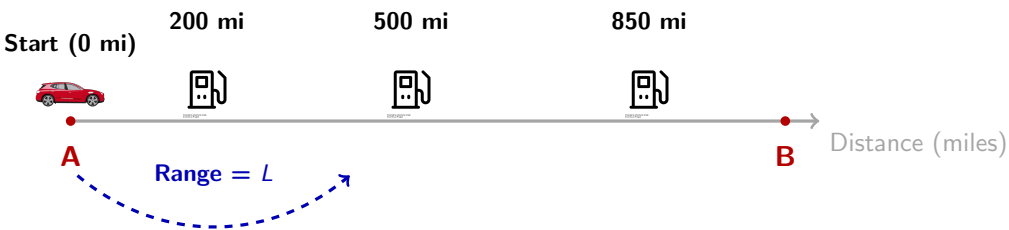
A greedy algorithm for car refueling

The Refueling Problem

The Setup:

- You are driving from A (position 0) to B.
- Your car's range on a full tank is L miles.
- There are gas stations at known positions $x_1, x_2, \ldots, x_n$.

Goal: Find a refueling strategy that minimizes the **total number of stops**.



A Greedy Strategy: Farthest-First

When you need to refuel, how should you choose the next stop?

- Should you stop at the closest possible station?
- Should you stop at a station in the middle of your range?

A Greedy Strategy: Farthest-First

When you need to refuel, how should you choose the next stop?

- Should you stop at the closest possible station?
- Should you stop at a station in the middle of your range?

Optimal Greedy Strategy

From your current position, drive to the **farthest reachable gas station** within your range L. Refuel there, and repeat the process.

This strategy maximizes your progress towards the destination with each stop. But is it optimal?

Proof of Correctness: General Recipe

The proof of correctness for a greedy algorithm has two main steps:

- The produced solution is **valid** given the constraints of the problem.
- The final solution is **optimal**.

Proof of Correctness: General Recipe

The proof of correctness for a greedy algorithm has two main steps:

- The produced solution is **valid** given the constraints of the problem. Usually easy.
- The final solution is **optimal**. This is the main hurdle.

Proof of Correctness: General Recipe

The proof of correctness for a greedy algorithm has two main steps:

- The produced solution is **valid** given the constraints of the problem. Usually easy.
- The final solution is **optimal**. This is the main hurdle.

Common Techniques for Proving Optimality: The core idea is a **comparison with an optimal solution**.

- **Greedy Stays Ahead:** Show that the greedy choice is always "better" than or equal to the optimal choice at every step, leading to an equally good final result.
- **Exchange Argument:** Show that any differences between a supposed optimal solution and the greedy solution can be "exchanged" to make the optimal solution more like the greedy one, without making it worse.

Proving Optimality

To prove our "Farthest-First" strategy is optimal, we'll use a classic technique: **Greedy Stays Ahead**.

1. **Setup:** Let's define two refueling strategies:
   - $G = \{g_1, g_2, \ldots, g_k\}$: The sequence of stops chosen by our **Greedy** algorithm.
   - $O = \{o_1, o_2, \ldots, o_m\}$: The sequence of stops in any **Optimal** solution.

Proving Optimality

To prove our "Farthest-First" strategy is optimal, we'll use a classic technique: **Greedy Stays Ahead**.

1. **Setup:** Let's define two refueling strategies:
   - $G = \{g_1, g_2, \ldots, g_k\}$: The sequence of stops chosen by our **Greedy** algorithm.
   - $O = \{o_1, o_2, \ldots, o_m\}$: The sequence of stops in any **Optimal** solution.

2. **Goal:** We want to prove that our greedy solution makes the minimum number of stops, i.e., $k = m$.

Proving Optimality

To prove our "Farthest-First" strategy is optimal, we'll use a classic technique: **Greedy Stays Ahead**.

1. **Setup:** Let's define two refueling strategies:
   - $G = \{g_1, g_2, \ldots, g_k\}$: The sequence of stops chosen by our **Greedy** algorithm.
   - $O = \{o_1, o_2, \ldots, o_m\}$: The sequence of stops in any **Optimal** solution.

2. **Goal:** We want to prove that our greedy solution makes the minimum number of stops, i.e., $k = m$.

3. **Method: Greedy Stays Ahead.** We will show that the greedy choice is always at least as close to the destination as the optimal choice at every step.

Proof Part 1: "Greedy Stays Ahead" Lemma

Lemma

For every stop i, the greedy choice g_i takes us at least as far as the optimal choice o_i. Formally:

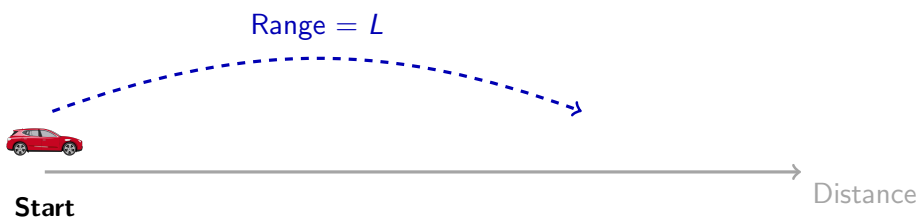
$$g_i \geq o_i \quad \text{for all } i \geq 1$$

Proof by Induction:

Created by Jamie Dickinson
from Noun Project

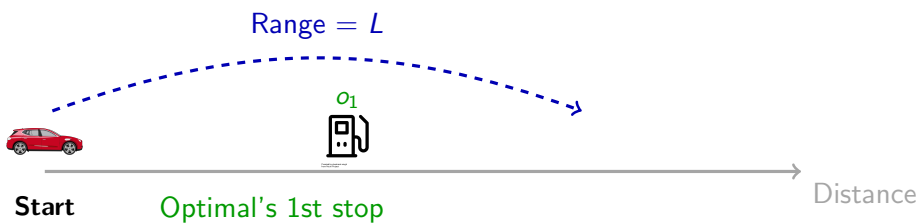
Proof Part 1: "Greedy Stays Ahead" Lemma

- **Base Case (i=1):** From the start, the greedy algorithm picks the farthest reachable station, g_1. Any other choice, including the optimal first stop o_1, must be reachable. By definition of our greedy choice, $g_1 \geq o_1$.



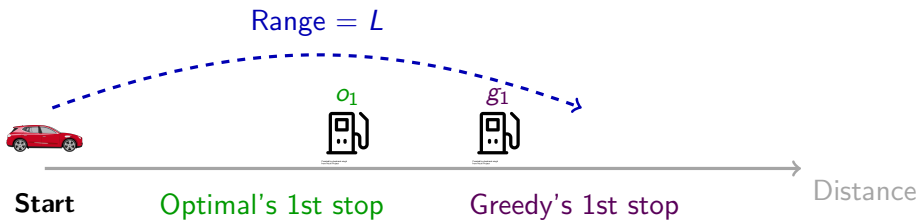
Proof Part 1: "Greedy Stays Ahead" Lemma

- **Base Case (i=1):** From the start, the greedy algorithm picks the farthest reachable station, g_1. Any other choice, including the optimal first stop o_1, must be reachable. By definition of our greedy choice, $g_1 \geq o_1$.



Proof Part 1: "Greedy Stays Ahead" Lemma

- **Base Case (i=1):** From the start, the greedy algorithm picks the farthest reachable station, g_1. Any other choice, including the optimal first stop o_1, must be reachable. By definition of our greedy choice, $g_1 \geq o_1$.



Proof Part 1: "Greedy Stays Ahead" (Inductive Step)

Inductive Hypothesis: Assume the lemma holds for stop $i-1$, so $g_{i-1} \geq o_{i-1}$.

Proof Part 1: "Greedy Stays Ahead" (Inductive Step)

Inductive Hypothesis: Assume the lemma holds for stop $i-1$, so $g_{i-1} \geq o_{i-1}$.

Inductive Step (for stop i):

Proof Part 1: "Greedy Stays Ahead" (Inductive Step)

Inductive Hypothesis: Assume the lemma holds for stop $i-1$, so $g_{i-1} \geq o_{i-1}$.

Inductive Step (for stop i):

- The optimal solution stops at o_i, which must be reachable from o_{i-1}.

$$o_i \leq o_{i-1} + L$$

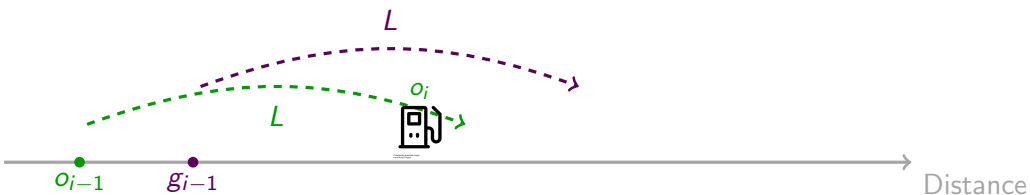
Proof Part 1: "Greedy Stays Ahead" (Inductive Step)

Inductive Hypothesis: Assume the lemma holds for stop $i-1$, so $g_{i-1} \geq o_{i-1}$.

Inductive Step (for stop i):

- Since $g_{i-1} \geq o_{i-1}$, our car at g_{i-1} can reach farther than the car at o_{i-1}.

$$o_i \leq o_{i-1} + L \leq g_{i-1} + L$$



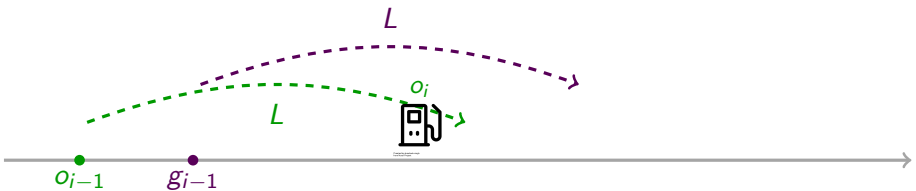
Proof Part 1: "Greedy Stays Ahead" (Inductive Step)

Inductive Hypothesis: Assume the lemma holds for stop $i-1$, so $g_{i-1} \geq o_{i-1}$.

Inductive Step (for stop i):

- This means station o_i is also reachable from our position at g_{i-1}.

$$o_i \leq o_{i-1} + L \leq g_{i-1} + L \qquad \text{and} \qquad g_i := \max_{j : x_j \leq g_{i-1}+L} x_j$$

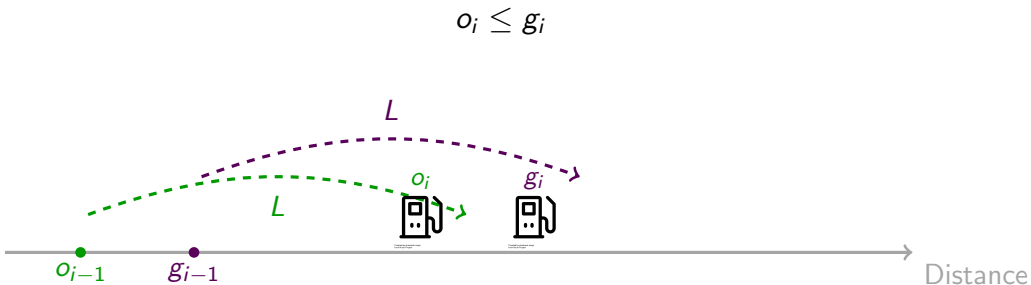


Proof Part 1: "Greedy Stays Ahead" (Inductive Step)

Inductive Hypothesis: Assume the lemma holds for stop $i-1$, so $g_{i-1} \geq o_{i-1}$.

Inductive Step (for stop i):

- The greedy algorithm chooses the *farthest* station, g_i. By definition, $g_i \geq o_i$.



Proof Part 2: Proving Optimality

Now we use the "Greedy Stays Ahead" lemma ($g_i \geq o_i$) to show the greedy solution is optimal.

Proof by Contradiction

Assume the greedy solution G is **not** optimal. This would mean that the greedy solution makes more stops than the optimal solution, i.e., $k > m$.

If $k > m$, then there is a stop g_{m+1} in the greedy solution. This stop was chosen because the destination B was not reachable from the previous greedy stop, g_m.

$$B - g_m > L$$

However, let's consider the optimal solution. After its final stop, o_m, it must be able to reach the destination.

$$B - o_m \leq L$$

Proof Part 2: Proving Optimality

From our lemma, we know that $g_m \geq o_m$. This implies:

$$B - g_m \leq B - o_m$$

Combining these inequalities, we get:

$$L < B - g_m \leq B - o_m \leq L$$

This simplifies to $L < L$, which is a **contradiction**!

Conclusion

Our assumption that the greedy solution is not optimal must be false. Therefore, the greedy solution makes the same number of stops as the optimal solution ($k = m$) and is indeed optimal.

3. Job Scheduling Problem

A greedy algorithm for job scheduling

A Scheduling Problem

The Setup: We are given n jobs to be processed on a single machine. Each job j has:

- A length (or processing time) $l_j > 0$.
- A weight (or priority) $w_j > 0$.

The Goal: Find an ordering of jobs that minimizes the **sum of weighted completion times**:

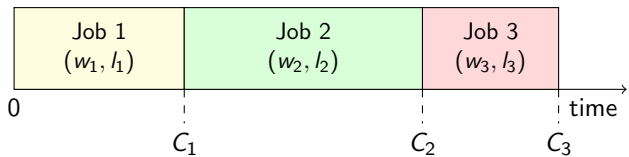
$$\min_{\sigma} \sum_{j=1}^{n} w_j C_j(\sigma)$$

Where:

- σ (sigma) represents a **schedule**, which is an ordering of the jobs.
- $C_j(\sigma)$ is the **completion time** of job j in that specific schedule.

How Completion Times Work

Completion Time C_j: The time at which job j finishes processing.



The completion times are calculated as follows:

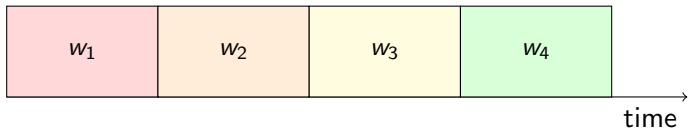
- $C_1 = l_1$
- $C_2 = l_1 + l_2$
- $C_3 = l_1 + l_2 + l_3$

The cost is: $w_1 l_1 + w_2(l_1 + l_2) + w_3(l_1 + l_2 + l_3)$.

Special Case 1: Equal Lengths

If all job lengths are the same, we should schedule jobs in **decreasing order of weight**.

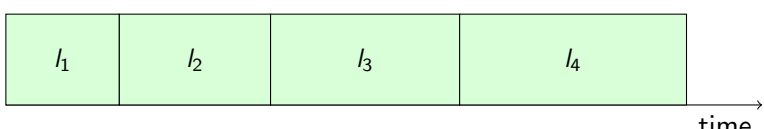
Give high-priority jobs smaller completion times.



Special Case 2: Equal Weights

If all job weights are the same, we should schedule jobs in **increasing order of length**.

Get short jobs out of the way to minimize impact on later jobs.



Potential Scores for General Case

Our special cases give conflicting advice for a short, low-weight job vs. a long, high-weight job.

Potential Scores for General Case

Our special cases give conflicting advice for a short, low-weight job vs. a long, high-weight job.

We need a "score" that combines length and weight.
Then, we schedule jobs in decreasing order of this score.

Potential Scores for General Case

Our special cases give conflicting advice for a short, low-weight job vs. a long, high-weight job.

We need a "score" that combines length and weight.
Then, we schedule jobs in decreasing order of this score.

Proposal 1: The Difference

Score $= w_j - l_j$

(GreedyDiff)

Proposal 2: The Ratio

Score $= w_j / l_j$

(GreedyRatio)

Comparing the Greedy Strategies

Let's test our two algorithms on a simple instance. Which job should go first?

Job 1
$l_1 = 5, w_1 = 3$

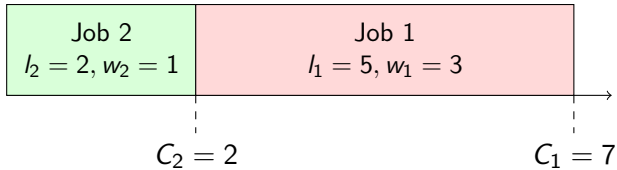
Job 2
$l_2 = 2, w_2 = 1$

	Job 1	**Job 2**
$w_j - l_j$	-2	**-1**
w_j / l_j	**0.6**	0.5

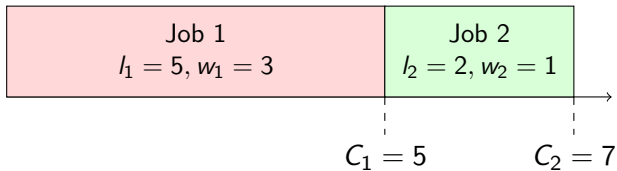
- **GreedyDiff** prefers Job 2 (higher difference).
- **GreedyRatio** prefers Job 1 (higher ratio).

The Winning Strategy

GreedyDiff Schedule (Job 2, Job 1) Cost $= (1 \cdot 2) + (3 \cdot 7) = \mathbf{23}$



GreedyRatio Schedule (Job 1, Job 2) Cost $= (3 \cdot 5) + (1 \cdot 7) = \mathbf{22}$



The Winning Algorithm: GreedyRatio

GreedyDiff is not optimal.

GreedyRatio is better for this instance. $\rightarrow$ Let's design an algorithm!

Greedy Strategy Based on GreedyRatio

1. For each job j, compute the score w_j/l_j.
2. Sort the jobs in descending order of their scores.
3. Schedule the jobs in this sorted order.

Running Time: The algorithm is dominated by the sorting step, so its running time is $\mathbf{O(n \log n)}$.

The Main Theorem

Theorem

For any set of jobs with positive lengths and weights, the **GreedyRatio** *algorithm produces a schedule with the minimum possible sum of weighted completion times.*

The Exchange Argument: A General Technique

This is a powerful method for proving a greedy algorithm is correct. The core idea is to show that any other solution can be transformed into the greedy one without increasing the cost.

1. Start with an assumed optimal solution, σ^*, that is different from the greedy one, σ.

The Exchange Argument: A General Technique

This is a powerful method for proving a greedy algorithm is correct. The core idea is to show that any other solution can be transformed into the greedy one without increasing the cost.

1. Start with an assumed optimal solution, σ^*, that is different from the greedy one, σ.

2. Find a place where σ^* differs from the greedy choice. This is an "inversion".

The Exchange Argument: A General Technique

This is a powerful method for proving a greedy algorithm is correct. The core idea is to show that any other solution can be transformed into the greedy one without increasing the cost.

1. Start with an assumed optimal solution, σ^*, that is different from the greedy one, σ.
2. Find a place where σ^* differs from the greedy choice. This is an "inversion".
3. Perform a small, local **exchange** to make σ^* look more like σ.

The Exchange Argument: A General Technique

This is a powerful method for proving a greedy algorithm is correct. The core idea is to show that any other solution can be transformed into the greedy one without increasing the cost.

1. Start with an assumed optimal solution, σ^*, that is different from the greedy one, σ.
2. Find a place where σ^* differs from the greedy choice. This is an "inversion".
3. Perform a small, local **exchange** to make σ^* look more like σ.
4. Show that this exchange does not increase the cost (and often improves it).

The Exchange Argument: A General Technique

This is a powerful method for proving a greedy algorithm is correct. The core idea is to show that any other solution can be transformed into the greedy one without increasing the cost.

1. Start with an assumed optimal solution, σ^*, that is different from the greedy one, σ.
2. Find a place where σ^* differs from the greedy choice. This is an "inversion".
3. Perform a small, local **exchange** to make σ^* look more like σ.
4. Show that this exchange does not increase the cost (and often improves it).
5. Argue that the greedy solution is at least as good as any optimal solution.

Proof Sketch: The Exchange Argument

[Assumption: for now suppose scores are unique.]

The proof works by showing that any schedule that is not the greedy schedule can be improved.

1. Assume for contradiction that there is an optimal schedule σ^* that is different from the greedy schedule σ.

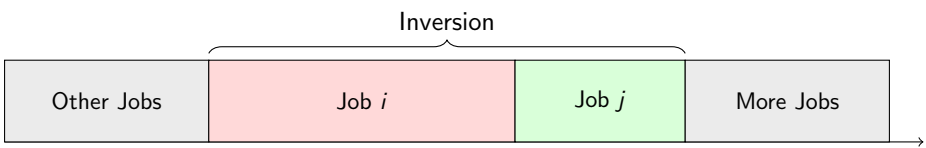
2. Since $\sigma^* \neq \sigma$, there must be two adjacent jobs i and j in σ^* that are "out of order" according to the greedy rule.
   This is not necessarily the first step where the solutions differ.

3. We will swap these two jobs to create a new, better schedule, which contradicts the assumption that σ^* was optimal.

Proof Sketch: The Inversion

An "out of order" pair is an **inversion**. In σ^*, we find adjacent jobs i, j where i is scheduled before j, but:

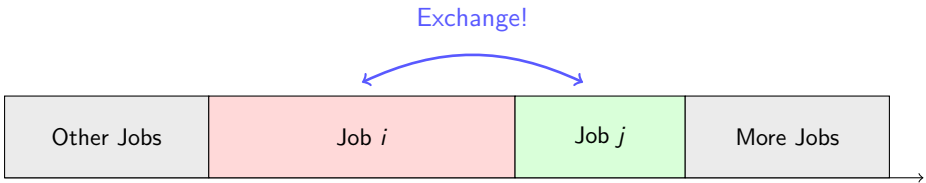
$$\frac{w_i}{l_i} < \frac{w_j}{l_j}$$



Optimal Schedule σ^*

Proof Sketch: The Swap

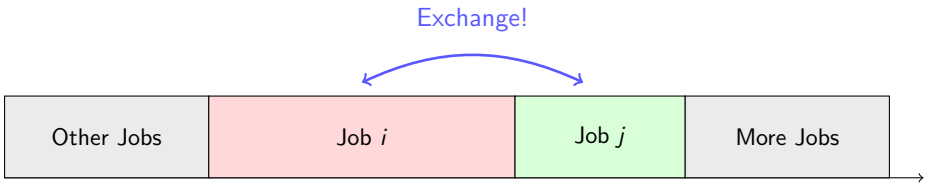
We create a new schedule σ' by swapping jobs i and j.



Optimal Schedule σ^*

Proof Sketch: The Swap

We create a new schedule σ' by swapping jobs i and j.



Optimal Schedule σ^*

After Swap Schedule σ'

Proof Sketch: Cost-Benefit of the Swap

How does the swap affect the total cost?

- Completion times of "Other Jobs" are **unchanged**.
- Completion time of j *decreases* by l_i. Cost change: $-w_j l_i$.
- Completion time of i *increases* by l_j. Cost change: $+w_i l_j$.

The total change in cost is:

$$\Delta(\text{Cost}) = w_i l_j - w_j l_i$$

Proof Sketch: The Contradiction

From the previous slide, we know the change in cost is $w_i l_j - w_j l_i$.

From our initial assumption about the inversion, we know: $\frac{w_i}{l_i} < \frac{w_j}{l_j}$

Proof Sketch: The Contradiction

From the previous slide, we know the change in cost is $w_i l_j - w_j l_i$.

From our initial assumption about the inversion, we know: $\frac{w_i}{l_i} < \frac{w_j}{l_j}$

Multiplying by the positive values $l_i l_j$ gives:

$$w_i l_j < w_j l_i \implies w_i l_j - w_j l_i < 0$$

The change in cost is negative! The new schedule σ' is strictly better than σ^*.

Contradiction!

This contradicts our assumption that σ^* was optimal. Therefore, no such σ^* can exist, and the greedy schedule must be optimal.

Why Is This Sufficient?

Key idea

If any non-greedy schedule can be improved by a swap, then the greedy schedule must already be optimal.

- We showed that whenever two adjacent jobs are out of greedy order, swapping them **strictly decreases** the total cost.
- Therefore, any schedule that is not in greedy order can be transformed into a strictly better one.
- Hence, the only schedule that cannot be improved is the greedy schedule itself.

Note that the *uniqueness of the score assumption* plays a central role here. Since $\frac{w_i}{l_i} < \frac{w_j}{l_j}$, the change in cost is strictly negative (not zero). Therefore, the optimal schedule is unique — precisely the one produced by the greedy rule.

Possible Concluding Proofs

Strictly improving the optimal solution. The exchange step strictly improves any non-greedy "optimal" schedule, which is impossible — an optimal schedule cannot be improved. Therefore, no optimal schedule can differ from the greedy one.

Possible Concluding Proofs

Strictly improving the optimal solution. The exchange step strictly improves any non-greedy "optimal" schedule, which is impossible — an optimal schedule cannot be improved. Therefore, no optimal schedule can differ from the greedy one.

Proof by contradiction. Assume there exists an optimal schedule O that differs from the greedy schedule G. Among all such optimal schedules, choose the one that is **most similar** to G (i.e., matches G on the largest prefix of jobs).
Now, apply the exchange argument to O to obtain a new schedule O' that is **even closer** to G and has no higher cost. This contradicts our choice of O as the most similar optimal schedule.

Possible Concluding Proofs

Strictly improving the optimal solution. The exchange step strictly improves any non-greedy "optimal" schedule, which is impossible — an optimal schedule cannot be improved. Therefore, no optimal schedule can differ from the greedy one.

Proof by contradiction. Assume there exists an optimal schedule O that differs from the greedy schedule G. Among all such optimal schedules, choose the one that is **most similar** to G (i.e., matches G on the largest prefix of jobs).
Now, apply the exchange argument to O to obtain a new schedule O' that is **even closer** to G and has no higher cost. This contradicts our choice of O as the most similar optimal schedule.

Inductive completion. By repeatedly applying this exchange argument, we can transform any valid schedule into the greedy one without increasing cost. Hence, the **greedy schedule is optimal.**

The Upshot

- Greedy algorithms build solutions via a sequence of myopic, locally optimal decisions.
- It is often easy to propose one or more greedy algorithms for a problem.
- **Most greedy algorithms are not correct.** Always be skeptical!
- When a greedy algorithm is correct, proving it can be difficult. The **exchange argument** and **greedy stays ahead** are two powerful proof techniques.

References


Dasgupta, S., Papadimitriou, C. H., and Vazirani, U. (2006).
Algorithms.
McGraw-Hill, Inc., USA, 1 edition.


Roughgarden, T. (2022).
Algorithms Illuminated: Omnibus Edition.
Soundlikeyourself Publishing, LLC.