Algorithms

Divide and Conquer

Dr. Mudassir Shabbir

LUMS University

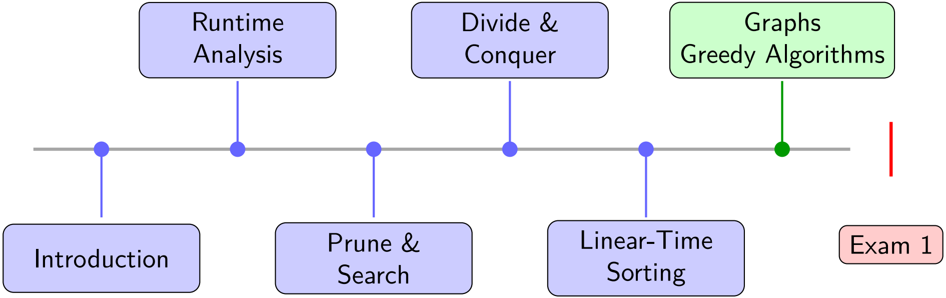
February 18, 2026

Announcements

- Midterm Exam/Long Quiz 1 on Sun 02/22, 2026 noon - 1:45p.
- Homework 2 (no-submission practice problems) is available - no submission required.

My Office Hours: Mon/Wed 12-1 PM.

Course Recap & What's Next



Recap: How to Write an Algorithm in the Exam

- *Input:* Describe the input format and what the algorithm receives.
- *Output:* Describe the output format and what the algorithm should produce.
- **Algorithm:** Write your algorithm in **plain English**, bullet points, or pseudocode.
- **Correctness:** Explain why your algorithm is correct, with a proof or argument.
- **Runtime Analysis:** Analyze the runtime of your algorithm.

Greedy Algorithms: The Good, the Bad, and the Ugly

How This Works

I'll show you a problem.

You vote: **Greedy Optimal** or **Greedy Fails**

(Sometimes: **It Depends**)

Rules:

No phones, no notes, just vibes and intuition.
If you get it right, you get... *nothing*.
But you'll *feel* smart. And that's what matters.

None of this is on the midterm. Relax.

Problem 1: Making Change at PDC

Scenario

You just had lunch at PDC. Your bill is 375 PKR and you pay with a 500-rupee note. The cashier needs to give you **125 PKR** back, but the register only has notes of **{5, 10, 20, 50, 75, 100}** PKR. You want the **fewest notes** possible.

Problem 1: Making Change at PDC

Scenario

You just had lunch at PDC. Your bill is 375 PKR and you pay with a 500-rupee note. The cashier needs to give you **125 PKR** back, but the register only has notes of **{5, 10, 20, 50, 75, 100}** PKR. You want the **fewest notes** possible.

The obvious greedy: Always pick the largest note that fits.

Vote Now!

Does greedy give the optimal (fewest notes) answer?

Problem 1: Making Change at PDC — Answer

× Greedy Fails!

Greedy says: $100 + 20 + 5 = 125$ PKR (3 notes)
Optimal: $75 + 50 = 125$ PKR (2 notes)

Lesson

Greedy coin change works for standard PKR denominations {10, 20, 50, 100, 500, 1000} by design. But for arbitrary denominations? You need dynamic programming.

Fun fact: Denominations where greedy always works are called "canonical" coin systems. Real-world currencies (PKR, USD, EUR) are canonical by design!

Problem 2: Road Trip on Empty

Scenario

You're driving from Lahore to Astore. Your tank holds enough fuel for **400 miles**. Gas stations are at various points along the highway. You want to make the **fewest stops** possible.

Problem 2: Road Trip on Empty

Scenario

You're driving from Lahore to Astore. Your tank holds enough fuel for **400 miles**. Gas stations are at various points along the highway. You want to make the **fewest stops** possible.

The obvious greedy: At each station, only stop if you can't reach the next one.

Vote Now!

Does greedy minimize the number of gas stops?

Problem 2: Road Trip on Empty — Answer

✓ **Greedy Wins!**

Strategy: At each station, only stop if you can't reach the next one.

Why it works: Skipping a stop never hurts — you can always stop later. So delaying stops as long as possible is optimal.

Alternative greedy (also works): Among reachable stations from your current position, pick the *farthest* one. Same result, different framing.

Proof Sketch

Exchange argument: if an optimal solution stops earlier than greedy at some point, you can "exchange" that stop for a later one without increasing total stops.

Problem 3: The Pizza Delivery Driver

Scenario

You have **5 deliveries** to make across town, then return to the shop. You want the **shortest total route**. Your GPS suggests: "always go to the closest house next."

Problem 3: The Pizza Delivery Driver

Scenario

You have **5 deliveries** to make across town, then return to the shop. You want the **shortest total route**. Your GPS suggests: “always go to the closest house next.”

The obvious greedy: Nearest neighbor — always visit the closest unvisited location.

Vote Now!

Does this always give the shortest delivery route?

Problem 3: The Pizza Delivery Driver — Answer

× **Greedy Fails!**

Classic counterexample:
4 points in a line: A—B—C—D
Starting at A, nearest neighbor visits B, C, D.
Total: $AB + BC + CD + DA$.
But if you go A $\rightarrow$ C $\rightarrow$ B $\rightarrow$ D, you might get a shorter total.

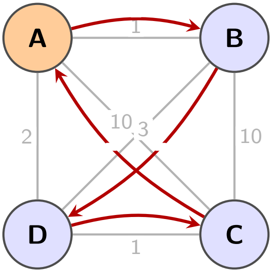
The bad news

This is the **Traveling Salesman Problem** (TSP). It's NP-hard. No known polynomial algorithm finds the optimal tour.

Nearest neighbor can be up to $O(\log n)$ times worse than optimal. Your pizza is getting cold.

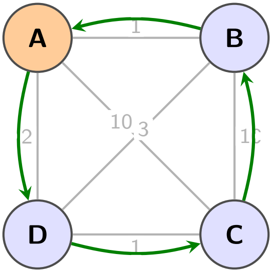
Problem 3: Greedy Failure — Concrete Example

Starting at **A**, greedy always picks the nearest unvisited city.



Greedy: $A \to B \to D \to C \to A$

Cost: $1 + 3 + 1 + 10 = \mathbf{15}$



Optimal: $A \to D \to C \to B \to A$

Cost: $2 + 1 + 10 + 1 = \mathbf{14}$

Greedy picks $AB = 1$ first (great!), but gets **trapped** needing the expensive $CA = 10$ edge. Optimal skips the "closest" neighbour and saves **1 unit** overall.

Problem 4: Netflix Binge Optimization

Scenario

You have a list of TV shows, each with a start and end time. You can only watch **one show at a time** (no second screen!). You want to watch the **maximum number** of complete shows today.

Problem 4: Netflix Binge Optimization

Scenario

You have a list of TV shows, each with a start and end time. You can only watch **one show at a time** (no second screen!). You want to watch the **maximum number** of complete shows today.

Some possible greedy strategies:

1. Pick the **shortest** show first
2. Pick the show that **starts earliest**
3. Pick the show that **ends earliest**

Vote Now!

Which (if any) greedy strategy is optimal?

Problem 4: Netflix Binge Optimization — Answer

✓ **Greedy Wins!** (Strategy 3: Earliest finish time)

Winning greedy: Sort by end time. Pick the first show. Skip everything that overlaps. Repeat.

Why the others fail:

- **Shortest first:** A short show in the middle can block two non-overlapping shows.
- **Earliest start:** A show starting at 6am but lasting 14 hours blocks everything.

Why earliest finish works

By finishing as early as possible, you leave the maximum room for future shows. Formally proven via exchange argument.

Problem 5: The Awkward Seating Chart

Scenario

You're planning a wedding. Some guests **hate each other** (modeled as edges in a graph). People at the same table must all get along. You want the **fewest tables** possible.

Problem 5: The Awkward Seating Chart

Scenario

You're planning a wedding. Some guests **hate each other** (modeled as edges in a graph). People at the same table must all get along. You want the **fewest tables** possible.

The obvious greedy: Go through guests one by one. Assign each to an existing table if possible, otherwise open a new table.

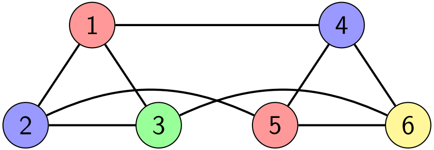
Vote Now!

Does this always minimize the number of tables?

Problem 5: The Awkward Seating Chart — Answer

× Greedy Fails!

Counterexample: Six people, each hating three others. Optimal seating uses **3 tables** (pairs of guests at the same tables needed), but greedy uses **4 tables** with a bad ordering.



Colors shown are greedy order $1, 2, \ldots, 6$: vertex 6 sees colors 1, 2, 3 $\Rightarrow$ forced to use color **4**.

Optimal: 3 Table — pair $(1, 5)$, $(2, 6)$, $(3, 4)$ at the same tables.

Greedy order $1, \ldots, 6$:

1 $\rightarrow$ red
2 $\rightarrow$ blue
3 $\rightarrow$ green
4 $\rightarrow$ blue
5 $\rightarrow$ red
6 $\rightarrow$ **yellow (4th!)**

Problem 6: The Bulk Candy Store

Scenario

You're at a candy store with a bag that holds **10 kg**. There are bins of candy with different prices per kg. You can scoop **any fraction** of a bin. You want to maximize the total value in your bag.

Problem 6: The Bulk Candy Store

Scenario

You're at a candy store with a bag that holds **10 kg**. There are bins of candy with different prices per kg. You can scoop **any fraction** of a bin. You want to maximize the total value in your bag.

The obvious greedy: Always scoop from the most expensive candy (highest \$/kg) first.

Vote Now!

Does greedy maximize the value of your candy bag?

Problem 6: The Bulk Candy Store — Answer

✓ **Greedy Wins!**

Strategy: Sort items by value-per-unit-weight (value density). Fill greedily from highest density.

Example:

Candy	Weight	Value	\$/kg
Truffles	3 kg	\$30	\$10/kg
Gummies	4 kg	\$28	\$7/kg
Mints	5 kg	\$25	\$5/kg

Bag = 10 kg: Take all 3 kg truffles (\$30) + all 4 kg gummies (\$28) + 3 kg mints (\$15) = **\$73**

Problem 7: The Bulk Candy Store... But Bags Are Sealed

Scenario

Same candy store, same 10 kg bag. But now the candy comes in **sealed packages** — you take the whole thing or leave it. No partial scooping!

Problem 7: The Bulk Candy Store... But Bags Are Sealed

Scenario

Same candy store, same 10 kg bag. But now the candy comes in **sealed packages** — you take the whole thing or leave it. No partial scooping!

The obvious greedy: Same strategy — pick by highest value/weight ratio.

Vote Now!

Does the same greedy strategy still work when you can't split items?

Problem 7: Sealed Bags — Answer

× Greedy Fails!

Counterexample: Capacity = 50

Item	Weight	Value	Value/Weight	Greedy picks?
A	10	60	6.0	✓(first)
B	20	100	5.0	✓(second)
C	30	120	4.0	× (won't fit!)

Greedy: A + B = weight 30, value **160**
Optimal: B + C = weight 50, value **220**

Lesson

Removing the ability to take fractions fundamentally changes the problem. This is the **0/1 Knapsack** problem (NP-hard). Needs DP.

Problem 8: The GPS with Toll Roads and Rebates

Scenario

You're driving through a network of roads. Most have tolls (positive edge weights). But some roads have **rebates** — they actually *pay you* to drive on them (negative weights). You want the cheapest route from A to B.

Problem 8: The GPS with Toll Roads and Rebates

Scenario

You're driving through a network of roads. Most have tolls (positive edge weights). But some roads have **rebates** — they actually *pay you* to drive on them (negative weights). You want the cheapest route from A to B.

The obvious greedy: Use Dijkstra's algorithm — it's greedy and finds shortest paths, right?

Vote Now!

Does Dijkstra's greedy approach still find the cheapest route?

Problem 8: GPS with Rebates — Answer

? It Depends...

Non-negative edges only? Dijkstra is **optimal**.
Negative edges present? Dijkstra **fails**.

Why it fails: Dijkstra "locks in" shortest distances greedily. But a negative edge discovered later could create a shorter path to an already-finalized node.

Fix: Use Bellman-Ford (not greedy — relaxes all edges $|V| - 1$ times).

Takeaway

Dijkstra is a greedy algorithm whose correctness *depends on a precondition* (no negative weights). If you violate the precondition, the greedy invariant breaks. Always check your assumptions!

Problem 9: Texting with a Bad Data Plan

Scenario

You want to compress text messages to save data. Common letters (e, t, a) should use **fewer bits**. Rare letters (z, q, x) can use more bits. You want the **shortest average encoding** with no ambiguity.

Problem 9: Texting with a Bad Data Plan

Scenario

You want to compress text messages to save data. Common letters (e, t, a) should use **fewer bits**. Rare letters (z, q, x) can use more bits. You want the **shortest average encoding** with no ambiguity.

The greedy idea: Repeatedly merge the two *least frequent* characters into a tree node.

Vote Now!

Does this greedy tree-building produce the optimal prefix code?

Problem 9: Texting with a Bad Data Plan — Answer

✓ Greedy Wins!

Huffman's Algorithm:

1. Create a leaf for each character with its frequency
2. Merge two nodes with smallest frequencies into a new node
3. Repeat until one tree remains

Example: Characters with frequencies A:5, B:9, C:12, D:13, E:16, F:45
The greedy merges produce an optimal prefix-free binary code!

Why it works

The two least frequent symbols should be deepest in the tree (longest codes). Greedy choice property: merging them first is always safe. Proven via contradiction.

Fun fact: Huffman invented this as a term paper in 1952. His professor couldn't solve!

Problem 10: Black Friday Shopping Spree

Scenario

It's Black Friday. You have a **$100 budget**. The store has dozens of items at various prices. You don't care *what* you buy — you just want to walk out with the **maximum number of items**.

Problem 10: Black Friday Shopping Spree

Scenario

It's Black Friday. You have a **$100 budget**. The store has dozens of items at various prices. You don't care *what* you buy — you just want to walk out with the **maximum number of items**.

The obvious greedy: Buy the cheapest item first. Then the next cheapest. Repeat until broke.

Vote Now!

Does greedy maximize the number of items?

Problem 10: Black Friday Shopping Spree — Answer

✓ **Greedy Wins!**

Strategy: Sort by price ascending. Buy items in order until budget runs out.

Why it works: Every cheap item you skip in favor of an expensive item can only decrease the count. There's no benefit to buying a $20 item over a $5 item if you want maximum quantity.

Proof (one line)

If solution S^* skips a cheaper item i and takes a more expensive item j, swapping j for i frees up budget ≥ 0, never decreasing the count. □

Note: This only works because all items have "equal value" (1 item = 1 item). The moment items have different values, you're back to knapsack territory.

Problem 11: The Database Join Nightmare

Scenario

You have 4 database tables and need to join them all together: $T_1 \bowtie T_2 \bowtie T_3 \bowtie T_4$. Joins are associative — the final result is the same, but the **order of joins** massively changes the cost. A join between a table of 1M rows and one of 500K rows is far more expensive than joining two small intermediate results. You want the **cheapest join order**.

Problem 11: The Database Join Nightmare

Scenario

You have 4 database tables and need to join them all together: $T_1 \bowtie T_2 \bowtie T_3 \bowtie T_4$. Joins are associative — the final result is the same, but the **order of joins** massively changes the cost. A join between a table of 1M rows and one of 500K rows is far more expensive than joining two small intermediate results. You want the **cheapest join order**.

The obvious greedy: Always join the pair of tables with the smallest combined size first.

Vote Now!

Does greedy find the optimal join order?

Problem 11: The Database Join Nightmare — Answer

× Greedy Fails!

Counterexample: Row counts: T_1: 10 rows, T_2: 100 rows, T_3: 5 rows, T_4: 50 rows (joining $T_i \bowtie T_{i+1}$ produces an intermediate with $|T_i|$ rows).

Option 1: $(T_1 \bowtie T_2) \bowtie T_3 = 10 \cdot 100 + 10 \cdot 5 = 1000 + 50 = \mathbf{1050}$ operations
Option 2: $T_1 \bowtie (T_2 \bowtie T_3) = 100 \cdot 5 + 10 \cdot 5 = 500 + 50 = \mathbf{550}$ operations

Greedy picks the cheapest *local* join, but ignores how the resulting intermediate table size explodes for *future* joins.

Takeaway

The number of join orderings grows as Catalan numbers: $C_n = \frac{1}{n+1}\binom{2n}{n}$. Every major query optimizer (Postgres, MySQL, Oracle) uses DP or heuristics — brute force is exponential; DP solves it in $O(n^3)$.

Problem 11: Wait — This is Matrix Chain Multiplication!

The Connection

The database join ordering problem is **exactly** the classical *Matrix Chain Multiplication* problem in disguise.

Formally: Given matrices $A_1 \times A_2 \times A_3 \times A_4$ with dims 10×100, 100×5, 5×50:
Option 1: $(A_1 \cdot A_2) \cdot A_3 = 10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 5000 + 2500 = \mathbf{7500}$
Option 2: $A_1 \cdot (A_2 \cdot A_3) = 100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50 = 25000 + 50000 = \mathbf{75000}$

Key Insight

Same structure, same DP solution. Recognizing that a *new* problem reduces to a *known* one is one of the most powerful tools in algorithm design. The $O(n^3)$ DP for matrix chain multiplication is your query optimizer's best friend.

Problem 11: A Concrete Join Example

Our Four Tables

Table	Rows	Cols
T_1: `Users`	10	4
T_2: `Orders`	100	3
T_3: `Items`	5	6
T_4: `Shipments`	50	3

Query: Find all users, their orders, items ordered, and shipment status.

```
SELECT * FROM Users
  JOIN Orders ON ...
  JOIN Items ON ...
  JOIN Shipments ON ...
```

Two Possible Join Orders

Order A: $((T_1 \bowtie T_2) \bowtie T_3) \bowtie T_4$

- $T_1 \bowtie T_2$: $10 \times 100 =$ **1000** ops → 10 rows
- $\cdot \bowtie T_3$: $10 \times 5 =$ **50** ops → 10 rows
- $\cdot \bowtie T_4$: $10 \times 50 =$ **500** ops
- **Total: 1550 ops** ✓

Order B: $T_1 \bowtie ((T_2 \bowtie T_3) \bowtie T_4)$

- $T_2 \bowtie T_3$: $100 \times 5 =$ **500** ops → 100 rows
- $\cdot \bowtie T_4$: $100 \times 50 =$ **5000** ops → 100 rows
- $T_1 \bowtie \cdot$: $10 \times 100 =$ **1000** ops
- **Total: 6500 ops**

Problem 12: The Overbooked Airport

Scenario

You manage an airport with limited gates. You have the arrival and departure times for every flight today. You need to find the **minimum number of gates** required so no two flights at a gate overlap.

Problem 12: The Overbooked Airport

Scenario

You manage an airport with limited gates. You have the arrival and departure times for every flight today. You need to find the **minimum number of gates** required so no two flights at a gate overlap.

The greedy idea: Sort all events (arrivals and departures) by time. Sweep through: $+1$ for arrivals, -1 for departures. Track the maximum.

Vote Now!

Does this greedy sweep find the minimum number of gates?

Problem 12: The Overbooked Airport — Answer

✓ Greedy Wins!

Strategy: Put all arrivals and departures in one sorted list. Sweep through with a counter. The peak value = minimum gates needed.

Example:

Flight 1: 9:00 – 11:30
Flight 2: 9:30 – 12:00
Flight 3: 11:00 – 13:00
Flight 4: 12:15 – 14:00

Events: +1(9:00), +1(9:30), +1(11:00), −1(11:30), −1(12:00), +1(12:15), −1(13:00)...
Peak concurrent = **3 gates**. That's optimal!

Why it works

At the moment of peak overlap, you *need* that many gates simultaneously.

Problem 13: The Security Camera Problem

Scenario

You need to place security cameras at intersections of a road network so that **every road** is monitored by at least one camera at either end. You want to use the **fewest cameras**.

Problem 13: The Security Camera Problem

Scenario

You need to place security cameras at intersections of a road network so that **every road** is monitored by at least one camera at either end. You want to use the **fewest cameras**.

The obvious greedy: Place a camera at the intersection with the most roads. Remove those roads. Repeat.

Vote Now!

Does greedily covering the most roads first minimize cameras?

Problem 13: The Security Camera Problem — Answer

× Greedy Fails!

Counterexample: A star graph with a center connected to many leaves, plus an isolated edge. Greedy picks the center (covers many edges), but the isolated edge still needs its own camera. A different solution might use fewer cameras overall.

This is the **Minimum Vertex Cover** problem. It's NP-hard.

But there's a simple 2-approximation: pick any edge, add *both* endpoints, remove covered edges. This always uses $\leq 2\times$ optimal cameras!

Fun fact

Vertex Cover was one of Karp's original 21 NP-complete problems (1972). The greedy "max degree" heuristic can be $O(\log n)$ times worse than optimal.

Problem 14: Wiring a College Campus on a Budget

Scenario

Your university needs to connect all campus buildings with fiber optic cable. Each possible connection has a cost. You want **all buildings connected** at **minimum total cable cost**. No cycles needed — just connectivity.

Problem 14: Wiring a College Campus on a Budget

Scenario

Your university needs to connect all campus buildings with fiber optic cable. Each possible connection has a cost. You want **all buildings connected** at **minimum total cable cost**. No cycles needed — just connectivity.

The greedy ideas:

1. **Kruskal's:** Sort all cables by cost. Add cheapest cable that doesn't create a cycle.
2. **Prim's:** Start at one building. Always extend to the cheapest reachable building.

Vote Now!

Do these greedy strategies find the minimum spanning tree?

Problem 14: Wiring a College Campus — Answer

✓ **Greedy Wins!** (Both strategies!)

Kruskal's: Greedy by global cheapest edge (+ union-find for cycle detection).
Prim's: Greedy by cheapest edge leaving the current tree.

Both are optimal! They may produce different MSTs (when edge weights tie), but always with the same total cost.

Why greedy works here

Cut property: For any cut of the graph, the minimum weight edge crossing the cut is safe to include in the MST. Both algorithms exploit this.

MST is one of the great success stories of greedy algorithms. Kruskal's appeared in 1956, Prim's in 1957 (though Jarník found it in 1930!).

Problem 15: Splitting the Chores Fairly

Scenario

You and your roommate need to split household chores. Each chore takes a different amount of time: **{7, 5, 5, 4, 4, 3}** minutes. You want to divide them so both people spend **as close to equal time** as possible.

Problem 15: Splitting the Chores Fairly

Scenario

You and your roommate need to split household chores. Each chore takes a different amount of time: $\{\mathbf{7, 5, 5, 4, 4, 3}\}$ minutes. You want to divide them so both people spend **as close to equal time** as possible.

The obvious greedy: Sort chores by time (descending). Assign each chore to the person with less total time so far.

Vote Now!

Does this greedy approach always find the most balanced split?

Problem 15: Splitting the Chores Fairly — Answer

× **Greedy Fails!** (but it's surprisingly close!)

Total $= 28$, target $= 14$.
Greedy: 7→A, 5→B, 5→B(10), 4→A(11), 4→A(15), 3→B(13). Split: **15 vs 13**.
Optimal: $\{7, 4, 3\}$ and $\{5, 5, 4\}$ $\Rightarrow$ **14 vs 14**. Perfect!

Takeaway

This is the **Partition Problem** — NP-hard! Greedy LPT gives a decent approximation (within $7/6$ of optimal) but isn't exact.

Problem 16: Assigning Ubers to Passengers

Scenario

There are n Uber drivers and n passengers on a number line (a long straight road). Each driver picks up exactly one passenger. You want to minimize the **maximum distance** any driver has to travel.

Problem 16: Assigning Ubers to Passengers

Scenario

There are n Uber drivers and n passengers on a number line (a long straight road). Each driver picks up exactly one passenger. You want to minimize the **maximum distance** any driver has to travel.

The obvious greedy: Sort both drivers and passengers by position. Match them in order: first driver to first passenger, second to second, etc.

Vote Now!

Does this sorted matching minimize the maximum distance?

Problem 16: Assigning Ubers to Passengers — Answer

✓ **Greedy Wins!**

Strategy: Sort drivers by position. Sort passengers by position. Match them pairwise: i-th driver $\rightarrow$ i-th passenger.

Example:
Drivers at positions: 1, 4, 9
Passengers at positions: 3, 5, 7
Matching: $1 \rightarrow 3$ (dist 2), $4 \rightarrow 5$ (dist 1), $9 \rightarrow 7$ (dist 2). Max = **2**.
Crossing matches (e.g., $1 \rightarrow 5$, $4 \rightarrow 3$) can only increase the max.

Why it works

Exchange argument: Any matching with "crossing" assignments (driver i goes right past driver j's passenger) can be uncrossed without increasing the max distance.

Problem 17: Placing Cell Towers

Scenario

A telecom company has identified 20 possible locations for cell towers. Each tower covers a different set of neighborhoods. You want **all neighborhoods covered** with the **fewest towers**.

Problem 17: Placing Cell Towers

Scenario

A telecom company has identified 20 possible locations for cell towers. Each tower covers a different set of neighborhoods. You want **all neighborhoods covered** with the **fewest towers**.

The obvious greedy: Always build the tower that covers the most *uncovered* neighborhoods.

Vote Now!

Does greedy always find the minimum number of towers?

Problem 17: Placing Cell Towers — Answer

? It Depends...

Not optimal, but remarkably good.
Counterexample: Universe $= \{1, 2, 3, 4, 5, 6\}$
$S_1 = \{1, 2, 3\}$, $S_2 = \{4, 5, 6\}$, $S_3 = \{1, 4\}$, $S_4 = \{2, 5\}$, $S_5 = \{3, 6\}$
Optimal: $S_1 + S_2$ (2 sets). Greedy might pick S_1, then S_2 (2 sets) — lucky!
But in general, greedy can use $O(\ln n)$ times more sets than optimal.

The silver lining

This is the **Set Cover Problem** (NP-hard). The greedy algorithm achieves a $\ln n$ approximation ratio, and this is *provably the best* any polynomial algorithm can do (assuming $P \neq NP$)!

Greedy isn't perfect, but nothing polynomial can do better. That's kind of beautiful.

Problem 18: The Fire Inspector

Scenario

A fire inspector must visit every building on a street. Each building has a **time window** during which it's available for inspection (an interval). The inspector is fast — each visit is instantaneous. She wants to pick the **fewest time points** to visit such that every building is inspected during its window.

Problem 18: The Fire Inspector

Scenario

A fire inspector must visit every building on a street. Each building has a **time window** during which it's available for inspection (an interval). The inspector is fast — each visit is instantaneous. She wants to pick the **fewest time points** to visit such that every building is inspected during its window.

The obvious greedy: Sort intervals by right endpoint. Pick the rightmost point of the first interval. Skip all intervals that contain this point. Repeat.

Vote Now!

Does this greedy minimize the number of visits?

Problem 18: The Fire Inspector — Answer

✓ Greedy Wins!

Strategy: Sort intervals by right endpoint. Place a point at the right end of the first uncovered interval. This "hits" as many overlapping intervals as possible.

Example:
Intervals: $[1, 4], [2, 6], [5, 7], [3, 5], [6, 9]$
Sorted by right end: $[1, 4], [3, 5], [2, 6], [5, 7], [6, 9]$
Pick point $t = 4$: covers $[1, 4], [3, 5], [2, 6]$
Pick point $t = 7$: covers $[5, 7], [6, 9]$
2 points total. Optimal!

Why it works

Delaying the point to the right end of the interval maximizes the chance of overlapping with subsequent intervals. Classic greedy-stays-ahead argument.

Problem 19: Procrastinator's Optimal Strategy

Scenario

You have n homework assignments, each with a deadline. All assignments take different amounts of time. You can only work on **one at a time**. You **will** finish all of them, but you want to minimize the **maximum lateness** (how late the worst assignment is).

Problem 19: Procrastinator's Optimal Strategy

Scenario

You have n homework assignments, each with a deadline. All assignments take different amounts of time. You can only work on **one at a time**. You **will** finish all of them, but you want to minimize the **maximum lateness** (how late the worst assignment is).

The greedy idea: Sort by deadline. Do the earliest-deadline assignment first.

Vote Now!

Does "earliest deadline first" minimize maximum lateness?

Problem 19: Procrastinator's Optimal Strategy — Answer

✓ **Greedy Wins!**

Strategy (EDD Rule): Sort all jobs by deadline. Process in that order.

Example:

Assignment	Duration	Deadline
Algorithms HW	3 hrs	5 PM
English Essay	2 hrs	4 PM
Lab Report	4 hrs	11 PM

EDD order: English (due 4PM), Algorithms (due 5PM), Lab (due 11PM).
Start at noon: English done 2PM ✓, Algo done 5PM ✓, Lab done 9PM ✓.

Why it works

Exchange argument: If two adjacent jobs are out of order, swapping them can only help.

Problem 20: Ancient Egyptian Math Homework

Scenario

Ancient Egyptians only used **unit fractions** (fractions with numerator 1). To represent $\frac{5}{7}$, they would decompose it as a sum of distinct unit fractions, like $\frac{1}{2} + \frac{1}{5} + \frac{1}{70}$.

Problem 20: Ancient Egyptian Math Homework

Scenario

Ancient Egyptians only used **unit fractions** (fractions with numerator 1). To represent $\frac{5}{7}$, they would decompose it as a sum of distinct unit fractions, like $\frac{1}{2} + \frac{1}{5} + \frac{1}{70}$.

The greedy (Fibonacci's method, 1202 AD): Always subtract the largest unit fraction $\frac{1}{k}$ such that $\frac{1}{k} \leq$ remainder.

Vote Now!

Does this greedy method always produce a valid decomposition?

Problem 20: Ancient Egyptian Math Homework — Answer

✓ **Greedy Wins!** (always terminates and is valid!)

Example: $\frac{5}{7}$

$\frac{5}{7} - \frac{1}{2} = \frac{3}{14}$ (since $\lceil 7/5 \rceil = 2$)

$\frac{3}{14} - \frac{1}{5} = \frac{1}{70}$ (since $\lceil 14/3 \rceil = 5$)

So $\frac{5}{7} = \frac{1}{2} + \frac{1}{5} + \frac{1}{70}$. ✓

Why it works

At each step, the numerator of the remainder strictly decreases (can be shown by algebra). Since numerators are positive integers, this must terminate.

*Caveat: The greedy decomposition isn't always the "best" (fewest terms or smallest denominators). For $\frac{5}{7}$, the decomposition $\frac{1}{2} + \frac{1}{4} + \frac{1}{28}$ also works and has smaller denominators! But greedy always produces **a** valid answer.*

Final Score

How did you do?

9 problems where greedy works
7 problems where greedy fails
2 problems where it depends

The meta-lesson:

Greedy algorithms are seductive. They're simple, fast, and intuitive.
The hard part is knowing *when* they work.

The key tools: **exchange arguments**, **greedy-stays-ahead proofs**,
and **counterexamples** when they don't.

Good luck on the midterm — which has nothing to do with any of this.