Day 1: Greedy Algorithm

What is greedy?

Greedy Algorithm is a problem-solving approach that makes the locally optimal choice at each step, which aims for finding the global optimum solution.

Core:

- Makes local optimal choices: At each decision point, pick the option that looks best immediately
- Never reconsiders: Once a choice is made, it's final (no backtracking)
- Hopes for global optimum: The series of local best choices should lead to the overall best solution

What problem could be solved by a greedy algorithm?

- Optimization problem (maximize or minimize something)
- Making sequential choices (one after another)

Examples

A delivery worker is distributing packages inside a large apartment complex. To avoid blocking hallways and to prevent the cart from tipping over due to excessive weight:

- Each trip with the cart can carry at most two packages
- Each trip has a maximum weight capacity limit (in kg)

You are given an array of packages, where packages[i] represents the weight of the i-th package. All packages must be delivered (order doesn't matter; they are all scheduled for the same delivery window)

Goal: Return the minimum number of cart trips required to deliver all packages, without exceeding the cart's weight limit on any trip.

Solution to Example

Key idea: Always try to pair the heaviest remaining package with the lightest one that can go with it. If the heaviest and lightest together are within the weight limit, send them together. Otherwise, send the heaviest alone. This ensures we save space for remaining packages as much as possible.

Algorithm Step:

1. Sort the package weights in non-decreasing order
2. Set 2 pointers.

i = lightest package
j = heaviest package

3. While i <= j:
   If packages[i] + packages[j] <= limit, pair them (i++, j–)
   Else, send only the heaviest (j–)
   Increment trip count each time
4. Return the total trips

```python
def min_cart_trips(packages, limit):
    packages.sort()
    i, j = 0, len(packages) - 1
    trips = 0
    while i <= j:
        if packages[i] + packages[j] <= limit:
            i += 1
        j -= 1
        trips += 1
    return trips
```

Time Complexity

Part 1: Sorting

- Comparison-based sort (e.g. quicksort, mergesort). Worst case: O(nlogn)

Part 2: Two-pointer sweep

- We perform a single linear pass across the sorted array.
- Each iteration decreases j by 1; at most half of the iterations we also increase i by 1. Therefore, the loop performs at most n iterations and n comparisons of the form packages[i] + packages[j] <= limit.
- Cost Θ(n) time.

Combined: T(n) = O(nlogn) + O(n)
Final Result: T(n) = Θ(nlogn)

Proof for Greedy Algorithm

- Greedy Choice Property: A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.

    - How to prove: Show that there exists an optimal solution that includes the greedy choice. If not, you can modify any optimal solution to include the greedy choice without making it worse.
- Optimal Substructure: An optimal solution to the problem contains optimal solutions to subproblems.
    - How to prove: Show that after making the greedy choice, the remaining problem is a smaller instance of the same problem, and solving it optimally leads to an optimal solution overall.

Proof Method: Exchange Argument

a. Assume there exists an optimal solution O that differs from greedy solution G
b. Show you can "exchange" choices in O to match G's choices
c. Prove that after each exchange, the solution remains optimal (or improves)
d. Conclude that G must be optimal

1. Look at the heaviest package H.
2. Check the lightest package L. If H+L > limit, then H cannot pair with anyone, since everyone else is heavier. So sending H alone now is optimal.
3. If H+L <= limit, the greedy pairs them.
4. Consider any optimal solution that does something different. Suppose H is paired with some package X instead of L.
5. Swap partners. Pair H with L instead. Put X wherever L was (or alone if needed).
6. This swap never increases trips. L <= X, so placing L where X is always safe or easier.
7. Every time greedy pairs (H, L), we can transform an optimal solution to match it without increasing trips. Thus, greedy is optimal.

Proof Method: Induction

a. Base case: Prove the greedy algorithm works for the smallest instance
b. Inductive hypothesis: Assume it works for problems of size n
c. Inductive step: Prove it works for problems of size n+1
d. By induction, it works for all sizes

1. Base case: 0 packages (0 trips, trivially optimal); 1 package (1 trip, must take it).
2. Assume greedy works for any smaller number of packages (inductive hypothesis).
3. Consider the current set, look at the lightest L and heaviest H packages.
4. Case 1: H + L ≤ limit (greedy pairs them)
    - Claim: There exists an optimal solution that pairs H and L in the same trip

- Why: Suppose optimal solution O sends them separately. Since H + L ≤ limit, we can modify O to pair H and L together, potentially freeing up space or reducing trips. If H was paired with X and L was paired with Y, we can rearrange: pair (H,L) and check (X,Y) or send X,Y separately. This doesn't increase trips.
- Conclusion: After pairing H and L optimally, we have n-2 packages left. By IH, greedy solves the remaining optimally. Total: 1 + optimal(n-2) = optimal(n).

5. Case 2: H + L > limit (greedy sends H alone)
   - Claim: In any optimal solution, H must go alone
   - Why: If H cannot pair with L (the lightest), it cannot pair with any package. So H must travel alone in any solution.
   - Conclusion: After sending H alone optimally, we have n-1 packages left. By IH, greedy solves the remaining optimally. Total: 1 + optimal(n-1) = optimal(n).