Problem Decomposition

This is a **Longest Increasing/Decreasing Subsequence variant**, adapted to train cars.

- Each car weight arrives in a fixed order.
- Erin can only attach cars to the **front** (heavier than current head of train) or to the **back** (lighter than current tail).
- The goal is to form the **longest possible valid train**, where cars are sorted in strictly decreasing order.

Subproblem Definition

We define two DP subproblems for each car *i;*

- **max_increasing[i]**: the longest subsequence of heavier cars that can be placed **in front** of car i.
- **max_decreasing[i]**: the longest subsequence of lighter cars that can be placed **behind** car i.
- The total length if pivoting at car i:
    - *train(i)* = *max_increasing*[i] + *max_decreasing*[i] − 1

Recursion & Order of Subproblems

Base Case = 1

- At least one car (itself) can always be placed, so there will always be a valid train length of 1.

max_increasing[i] = 1 and *max_decreasing[i]* = 1

Recursive Step

We compute values by traversing from **last car → first car** (right to left; bottom-up), comparing each car with later arrivals:

Case 1 – Increasing (add to front)

Front (heavier cars):

- Look at all later cars j > i.
- If weight[i] < weight[j], then car j can be placed in front.
- Extend the sequence:
    - *max_increasing [i] = max(max_increasing[i], 1 + max_increasing [j])*

If the current car's weight is less than a later car's weight, then we can extend the **increasing sequence length** at the current index by 1 + the sequence length at that later index.

Case 2 — Decreasing (add to back)

Train Sorting

Back (lighter cars):

- If weight[i] > weight[j], then car j can go behind.

- Extend the sequence:
    - *max_decreasing [i] = max(max_decreasing[i],1 + max_decreasing [j])*

Boundary and Termination

After you compute all subproblems, you will combine the two:

longest_train= max(max_increasing [i] + max_decreasing [i] − 1)

For each pivot car i, imagine it standing in the middle. In front, we attach the tallest sequence of heavier cars. Behind, we attach the longest sequence of lighter cars. We subtract 1 so the pivot isn't double-counted, then pick the pivot that gives the longest total train.

Summary of Notation
n - number of cars
weight [i]
max_increasing[i]
max_decreasing[i]
train(i)

Example:

Input:
3
1
2
3

So,
index: 0 1 2
weight: [1, 2, 3]

Step 0 - Base case
- Each car alone is length 1
    - max_increasing = [1, 1, 1] #by index
    - max_decreasing = [1, 1, 1]

Step 1 - Pivot at index 2 (weight = 3)
- There are no cars after index 2, so the sequence stays:
    - max_increasing[2] = 1
    - max_decreasing[2] = 1

Train Sorting

Visual:

- Front (heavier) → none
- Pivot → car at index 2 (weight 3)
- Back (lighter) → none

Step 2 — Pivot at index 1 (weight = 2)

- Compare with index 2 (weight 3):
  - weight 2 < weight 3 → can go in front → extend:
    - max_increasing[1] = 1 + max_increasing[2] = 1 + 1 = 2
- No lighter cars →
  - max_decreasing[1] = 1

Visual:

- Front (heavier) → 3
- Pivot → 2
- Back (lighter) → none

Step 3 — Pivot at index 0 (weight = 1)

- Compare with index 1 (weight 2):
  - 1 (weight) < 2 (weight) →
    - max_increasing[0] = 1 + max_increasing[1] = 1 + 2 = 3
- Compare with index 2 (weight 3):
  - 1 (weight) < 3 (weight) → max_increasing[0] = max(3, 1 + max_increasing[2])
    = max(3, 1 + 1)
    = 3

- No lighter cars after index 0 →
  - max_decreasing[0] = 1

Visual

- Front (heavier) → car at index 1 (weight 2) → car at index 2 (weight 3)
- Pivot → car at index 0 (weight 1)
- Back (lighter) → none

Step 4 - Combining at Each Pivot

train(i) = max_increasing[i] + max_decreasing[i] − 1

So from what we have done in the above steps:

train(0) = 3 + 1 - 1 = 3
train(1) = 2 + 1 - 1 = 2
train(2) = 1 + 1 - 1 = 1

(work backwards)

Time and Space Complexity

- **Time Complexity:**
  - Each car compares with all later cars → $O(n^2)$.
- **Space Complexity:**
  - We store two arrays (max_increasing, max_decreasing) of size n → $O(n)$.