Multiplicity-Aware Lexicographic Scaling (MALS)

Multiplicity-Aware Lexicographic Scaling (MALS) is an algorithm that transforms relative priority values into **dominating weights**. The output weights are guaranteed to respect *lexicographic dominance*: any item at a higher priority level outweighs **all items** at lower levels combined.

Problem Statement

- **Input**: a vector of integer priorities
  $P = (p_1, p_2, \dots, p_n)$
  (positive and negative values allowed).

- **Output**: a vector of integer weights
  $W = (w_1, w_2, \dots, w_n)$
  such that higher priority items *dominate* all lower priority items.

Algorithm (Variant A — All-Lower Dominance)

1. **Normalize levels**
2. Compute absolute values $|p_i|$.
3. Sort the distinct values.
4. Assign each absolute value a *level index* $\ell \in \{0, 1, 2, \dots\}$.
5. Each item p_i is thus assigned a level L_i .

6. **Count multiplicities**

7. For each level ℓ , compute the count

$$c_\ell = \left|\{\, i : L_i = \ell \,\}\right|.$$

8. **Compute level weights**

9. Initialize:

$$w_0 = 0, \quad S_0 = c_0 \cdot w_0$$

10. For each level $\ell \geq 1$:

$$w_\ell = S_{\ell-1} + 1, \qquad S_\ell = S_{\ell-1} + c_\ell \cdot w_\ell.$$

This guarantees that a single item at level ℓ outweighs **all items from lower levels**.

1. **Assign signed weights**

2. For each input p_i :

$$f(p_i) = \{0, \quad p_i = 0, \ \mathrm{sign}(p_i) \cdot w_{L_i}, \ \text{otherwise.}$$

3. **Return**

4. The transformed vector $W = (f(p_1), f(p_2), \dots, f(p_n))$.

Example

Input

```
[0, 1, 1, 1, 2, 2, -2, 3, 3, -4, -4, 7]
```

Steps: - Normalize levels → `[0,1,1,1,2,2,-2,3,3,-4,-4,5]` - Counts: `{0:1, 1:3, 2:3, 3:2, 4:2, 5:1}` - Level weights: `[0, 1, 4, 13, 27, 55]`

Output

```
[0, 1, 1, 1, 4, 4, -4, 13, 13, -27, -27, 55]
```

Properties

- **Deterministic** and order-independent (depends only on priority magnitudes and their multiplicities).
- **Lexicographic dominance**: any single item at level ℓ outweighs all lower levels combined.
- Time complexity: $O(n \log n)$ (sorting distinct absolute values dominates); space $O(n)$.

Mathematical Definition

Let
- $P = (p_1, \dots, p_n)$ be input priorities,
- $L_i = \mathrm{rank}(|p_i|)$ the level index of $|p_i|$ in the sorted list of distinct absolute values,
- $c_\ell = |\{\, i : L_i = \ell \,\}|$ the multiplicity at level ℓ .

Define:

$$w_0 = 0,\ S_0 \quad = c_0\, w_0,\ w_\ell = S_{\ell-1} + 1 \quad (\ell \geq 1),\ S_\ell \quad = S_{\ell-1} + c_\ell\, w_\ell.$$

Then the output weight for item i is

$$f(p_i) = \Big\{ 0, \quad p_i = 0,\ \operatorname{sign}(p_i)\, w_{L_i},\ \text{otherwise}.$$

Python Reference Implementation

```
from typing import List

def mals_all_lower_dominance(priorities: List[int]) -> List[int]:
    """
    Multiplicity-Aware Lexicographic Scaling (Variant A).
    Each priority level dominates *all* lower levels combined.

    Parameters
    ----------
    priorities : list[int]
        Input vector of integers (can be positive or negative).

    Returns
    -------
    list[int]
        Transformed weights preserving sign and guaranteeing lexicographic
dominance.
    """
    if not priorities:
        return []

    # 1) Normalize priorities into consecutive levels (by abs value)
    abs_vals = [abs(x) for x in priorities]
    levels_sorted = sorted(set(abs_vals))  # unique abs values
    level_of = {v: i for i, v in enumerate(levels_sorted)}
    L = [level_of[abs(x)] for x in priorities]

    # 2) Count multiplicity per level
    K = max(L)
    counts = [0] * (K + 1)
    for lvl in L:
        counts[lvl] += 1

    # 3) Build weights so that one item at level k beats all lower levels
combined
    w = [0] * (K + 1)
```

```
    S = counts[0] * w[0]  # running dominated mass
    for k in range(1, K + 1):
        w[k] = S + 1
        S += counts[k] * w[k]

    # 4) Map back with original signs
    out = []
    for i, p in enumerate(priorities):
        if p == 0:
            out.append(0)
        else:
            out.append((1 if p > 0 else -1) * w[L[i]])
    return out

if __name__ == "__main__":
    # Example
    inp = [0, 1, 1, 1, 2, 2, -2, 3, 3, -4, -4, 7]
    print("input:  ", inp)
    print("output: ", mals_all_lower_dominance(inp))
```

Notes & Variants

- If you instead want each level to dominate **only the immediately lower level** (not all lower levels combined), set $w_\ell = c_{\ell-1} \cdot w_{\ell-1} + 1$ with suitable bases; this reproduces the sequence in the worked example and can be documented as a variant ("Adjacent-Level Dominance").
- You can adopt exponential bases (e.g., powers of a base B) to reach the same lexicographic effect without counting multiplicities; MALS is *multiplicity-aware*, producing minimal integer weights.