# **Pre-Coding Essentials (Component: crates/vm\_algo/src/allocation/largest\_remainder.rs, Version/FormulaID: VM-ENGINE v0) — 45/89**

## **1) Goal & Success**

Goal: Implement **Largest Remainder (LR)** seat allocation with selectable quota (**Hare, Droop, Imperiali**), after applying the **PR entry threshold**. Integer-only math; deterministic/reproducible ties.

Success: Floors + remainder distribution sums to m; below-threshold options excluded; over-allocation handled (trim from **smallest remainder**) for Imperiali edge cases; convergence case matches tests (A/B/C 34/33/33, m=7 → **3/2/2**).

## **2) Scope**

In scope: Threshold filter; quota computation (Hare/Droop/Imperiali); floor seats; remainder ranking; deterministic tie-breaking; over-allocation trim path.

Out of scope: Tabulation, aggregation, gates/frontier, any I/O.

## **3) Inputs → Outputs**

Inputs:

seats: u32 (m ≥ 1)

scores: &BTreeMap<OptionId, u64> (natural tallies from tabulation)

options: &[OptionItem] (for (order\_index, id) order and status-quo flag)

threshold\_pct: u8 (**VM-VAR-012**)

quota: QuotaKind (Hare|Droop|Imperiali)

tie\_policy: TiePolicy, optional rng: &mut TieRng

Output: BTreeMap<OptionId, u32> seats per option (sum = seats)

## **4) Entities/Tables (minimal)**

## **5) Variables (used here)**

## **6) Functions (signatures only)**

rust

CopyEdit

use std::collections::BTreeMap;

use vm\_core::{

ids::OptionId, entities::OptionItem,

rng::TieRng, variables::TiePolicy,

};

#[derive(Copy, Clone, Eq, PartialEq)]

pub enum QuotaKind { Hare, Droop, Imperiali }

pub fn allocate\_largest\_remainder(

seats: u32,

scores: &BTreeMap<OptionId, u64>,

options: &[OptionItem],

threshold\_pct: u8,

quota: QuotaKind,

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

) -> Result<BTreeMap<OptionId, u32>, AllocError>;

// helpers

fn filter\_by\_threshold(scores: &BTreeMap<OptionId,u64>, threshold\_pct: u8) -> BTreeMap<OptionId,u64>;

fn compute\_quota(total: u128, seats: u128, quota: QuotaKind) -> u128; // integer-only

fn floors\_and\_remainders(

eligible: &BTreeMap<OptionId,u64>,

quota: u128

) -> (BTreeMap<OptionId,u32>, BTreeMap<OptionId,u128>); // floors + fractional leftovers

fn distribute\_leftovers(

seats: u32,

alloc: &mut BTreeMap<OptionId,u32>,

remainders: &BTreeMap<OptionId,u128>,

options: &[OptionItem],

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

);

fn trim\_over\_allocation\_if\_needed(

seats: u32,

alloc: &mut BTreeMap<OptionId,u32>,

remainders: &BTreeMap<OptionId,u128>,

options: &[OptionItem],

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

) -> bool; // Imperiali edge-case

## **7) Algorithm Outline (implementation plan)**

**Threshold filter**

Compute each option’s share using the ballot’s **natural totals**; drop options **strictly below** threshold\_pct.

**Quota**

Let V = sum(scores) and m = seats.

**Hare:** q = floor(V / m)

**Droop:** q = floor(V / (m + 1)) + 1 (example: V=90,m=4 → q=19).

**Imperiali:** q = floor(V / (m + 2)) (example: V=3,m=1 → q=1).

Use u128 for the division to avoid overflow.

**Floors**

For each eligible option: floor\_i = scores[i] / q (clamp to m if q==0, but in practice q>=1 once m≥1 and V>0). Sum floors.

**Remainders**

rem\_i = scores[i] % q (store as u128).

**Distribute leftovers**

If sum(floor\_i) < m: assign remaining seats one by one to **largest remainders**; ties broken by **higher raw score**, then by canonical (order\_index, id); if tie\_policy=random, draw via seeded RNG.

**Trim (Imperiali edge)**

If sum(floor\_i) > m (can happen under **Imperiali**, tiny totals), **trim** starting from **smallest remainder** until total equals m; equal-remainder trims use deterministic order (or seeded RNG if requested).

**Return**

Deterministic BTreeMap<OptionId,u32>; sum equals m.

LR definition & steps per spec; threshold applies **beforehand**; “score” means the ballot’s **natural** tally (approval=approvals, plurality=votes, score=sums).

## **8) State Flow**

Called by **AllocateUnit** after Tabulate; before aggregation; respects threshold and tie rules from Doc 4B/4C. Convergence test shared with highest-averages.

## **9) Determinism & Numeric Rules**

Integer-only math; compare remainders/scores via integers.

Stable option ordering by (order\_index, OptionId) for deterministic ties; RNG path uses only provided seeded generator for reproducibility.

## **10) Edge Cases & Failure Policy**

seats == 0 ⇒ empty allocation.

After threshold, **no eligible options** ⇒ AllocError::NoEligibleOptions.

V == 0 with seats > 0 ⇒ allocate entirely by tie policy (deterministic order unless random).

**Imperiali over-allocation** ⇒ trim from **smallest remainder** (deterministically or via RNG if requested).

## **11) Test Checklist (must pass)**

**Convergence (VM-TST-003)**: A/B/C = 34/33/33, m=7 ⇒ **3/2/2**.

**Droop boundary**: V=90, m=4 → q=19; votes {A:50,B:28,C:12}; floors+remainders yield total 4 with deterministic selection.

**Imperiali trim**: V=3, m=1 → q=1; floors 1,1,1 (sum 3) → **trim** from smallest remainder (all equal → canonical order).

Determinism: shuffled map insertion and equal remainders follow canonical order; with random + fixed seed, selection is reproducible.