Cancellation and Surplus Patterns: A 2-Page Cheat Sheet

GOAL
Find exact or approximate frequent elements in a data stream with small memory and one pass. Central idea: maintain a minimal residue via online cancellation or a surplus (vote margin) potential.

TERMS
Residue: result of deleting “opposite” pairs without changing winners. Vote margin s: unpaired count of current candidate. Phase: segment between resets when s returns to 0.

PATTERN 1 — Boyer–Moore Majority (> n/2)
State: (candidate c, margin s>=0)
Update per x: if s==0 -> c:=x; s:=1   else if x==c -> s++   else -> s--
Invariant: online simulation of deleting unequal pairs. After any prefix, residue is empty or k copies of one value; (c,s) equals that residue. If a true majority exists, residue over full stream is non-empty and equals it.
Verify: second pass counts(c) > n//2
Merge/distribute: yes; reduce chunk residues with same rule, then verify globally.
Complexity: O(n) time, O(1) space.

Boyer–Moore pseudocode (reference)

```
s=0; for x in stream:
  if s==0: c=x; s=1
  elif x==c: s+=1
  else: s-=1
# verify c by counting
```

PATTERN 2 — Misra–Gries Heavy Hitters (> n/k)
Goal: candidates for all items with freq > n/k.
State: up to k-1 pairs (item, count).
Update: if x tracked -> ++; else if room -> insert(x,1); else decrement all counts and delete zeros.
Guarantee: every true heavy hitter appears among candidates; counts are underestimates by at most total decrements. Verify in second pass.
Notes: exact candidate set with O(k) space and one pass; mergeable across partitions.

PATTERN 3 — Space-Saving (approx top-k with errors)
State: k slots (item, count, error).
Update: hit -> count++; miss -> replace min-count slot with (x, min+1, min).
Guarantee: stored_count(x) − error(x) <= true_count(x) <= stored_count(x).
Use: fixed memory, good for streaming top-k dashboards.

PATTERN 4 — XOR Parity (Single Number)
State: a=0; Update: a ^= x
Invariant: pair duplicates cancel under XOR; a equals element with odd multiplicity.
Use: “find the unique among pairs/triples” variants (extend with bitwise FSM for triples).

PATTERN 5 — Monotone Stack/Deque (Next Greater, Stock Span, Sliding Window Min)
State: stack/queue keeping monotonicity.
Update: pop while invariant violated; push x.
Invariant: structure holds only candidates that can survive future dominance checks.
Use: O(n) time range extrema and spans.

POTENTIAL / RESET FAMILY

PATTERN 6 — Kadane (Max Subarray)
State: best_suffix, best. Update: best_suffix=max(x, best_suffix+x); best=max(best, best_suffix).
Idea: forget history when debt is positive. Merge via (sum, best, best_prefix, best_suffix).

PATTERN 7 — Gas Station (Circular Tour)
State: start, tank, total. Update: tank+=gas[i]-cost[i]; if tank<0: start=i+1; tank=0.
Invariant: last reset wins if total>=0. One pass, O(1).

SKETCHING / WINDOWED

PATTERN 8 — Count-Min Sketch (ε,δ approximate counts)
State: d hash rows × w counters.
Update: inc one cell per row at h_j(x). Query: min over rows.
Error: overestimates by ≤ εN with prob ≥ 1−δ. Merge: add counters.

PATTERN 9 — DGIM / Smooth Histograms (Sliding Windows)
State: timestamped buckets with exponentially scaled sizes; keep ≤2 per size.
Update: on each 1, create bucket; merge oldest pair when >2. Query by summing buckets.
Use: approximate counts in last W items.

COMPOSITION / QUERIES

PATTERN 10 — Segment Tree with Boyer–Moore Monoid
Node state: (candidate, margin). Merge: apply Boyer–Moore’s cancel/accumulate to children.
Use: range majority queries; verify candidates on demand.

WEIGHTED VARIANTS

PATTERN 11 — Weighted Boyer–Moore
State: (c,s). For item with weight w: if x==c -> s+=w else s-=w; if s≤0 set c:=x and s:=−s (or pick next with residual).
Use: majority under costs/weights; verify by weighted count.

SAMPLING

PATTERN 12 — Reservoir Sampling (k items)
State: sample S, t. Update: t+=1; with prob k/t replace a random element in S with x.
Invariant: each seen item in S with prob k/t. Merge: weighted reservoirs.

DESIGN CHECKLIST
1) Exact vs approximate? Threshold known (>n/2, >n/k)?
2) Stream constraints: one pass, memory cap, verification possible?
3) Mergeability: per-partition summaries must compose.
4) Windowed vs global? If windowed, prefer deque or DGIM-style summaries.
5) Adversarial order? Keep proofs independent of input order.
6) Weighted inputs? Use weighted cancellation or reweighting.

CHOOSER (RULE OF THUMB)
- Exact global majority: Boyer–Moore + verify.
- All exact > n/k: Misra–Gries + verify.
- Top-k under tight RAM, allow error: Space-Saving or Count-Min + heap.
- Unique by parity: XOR or bit-FSM.
- Range queries: segment tree with Boyer–Moore monoid + verify.
- Windowed extrema: monotone deque; windowed counts: DGIM.
- Running “best with resets”: Kadane, Gas-station.

PITFALLS
- Skipping verification when majority may not exist.
- Misinterpreting counts from Misra–Gries as exact.
- Using sketches where deletions or windows are required (needs specialized variants).
- Ignoring merge semantics in distributed settings.
- Allowing negative margins without reset in weighted variants.

MINIMAL REFERENCES
- Boyer & Moore, 1981. “MJRTY — A Fast Majority Vote Algorithm.”
- Misra & Gries, 1982. “Finding Repeated Elements.”
- Metwally et al., 2005. “Efficient Computation of Frequent and Top-k Elements...”
- Cormode & Muthukrishnan, 2005. “An Improved Data Stream Summary: the Count-Min Sketch.”