Rules for every challenge

- **Do all setup (filling, seeding indices, printing) outside the timed block.**
- **Put only the target work between t0 and t1.**
- **Repeat ≥5 trials and average. Try multiple n (e.g., 1e3, 1e4, 1e5) where it makes sense.**
- **Hand in: your code, a small table of times, and a 2–4 sentence explanation per part.**

Part A — Circular & Doubly Linked List mastery

A1 — CSLL: tail-to-head wrap vs manual reset

- **Implement traversal of n nodes in two ways:**
  (i) CSLL with tail->next = head and a single loop of n steps;
  (ii) Non-circular SLL that restarts at head whenever you hit nullptr.
- **Predict which is faster and why (branching, cache/predictability).**
- **Measure and explain.**

A2 — CSLL deletion with/without predecessor

- **Case 1:** Delete a given node when you also have its predecessor (O(1)).
- **Case 2:** Delete the same node when you only have the node pointer (must find predecessor).
- **Predict cost difference; measure for random positions; explain the curve.**

A3 — Rotate-k on CSLL vs SLL

- **Implement "rotate right by k":** in CSLL it's pointer moves; in SLL it's find-break-relink.
- **Test for multiple k (small, n/2, n−1).**
- **Decide** which wins and under what k ranges.

A4 — DLL vs SLL: erase-given-node

- **Build DLL with prev.** Given a pointer to any node, erase it.
- **Compare to SLL** erase with known predecessor and SLL erase without predecessor.
- **Predict → measure → explain O(1) vs O(n) and constant-factor hits.**

A5 — Push/pop ends: head-only vs head+tail

- Implement push_front/pop_front and push_back/pop_back on:
  (i) SLL with head only, (ii) SLL with head+tail, (iii) DLL with head+tail.
- **Benchmark each op for random mixes (e.g., 70% push_back, 30% pop_front).**
- **Explain why tail changes the story.**

A6 — Memory overhead audit

- **For the same logical dataset size n, allocate SLL, CSLL, and DLL nodes.**
- **Report bytes per node (pointer count), total bytes, and measured allocation time.**
- **Discuss the time–space trade-off you'd choose for frequent middle deletions.**

Part B — Real-world use cases

B1 — Recent Items Tray (add/remove at the same end)

- **Build a "recent items"** tray where the most recently added item is always the next one removed.

- **Implement with:**
  A) Singly linked nodes adding/removing at the front.
  B) Doubly linked nodes adding/removing at the front.
- **Predict which is faster** (pointer count vs rewiring), measure adds/removes (mixed workload), explain whether the second pointer helps here.

B2 — Editor Undo History

- **Implement an Undo history where each new action is "placed on top," and Undo removes the last action added.**
- **Version A: Singly linked front-add/front-remove.**
- **Version B: Dynamic array (grow by doubling).**
- **Workload: 80% add actions, 20% undo actions.**
- **Predict throughput and memory spikes (reallocs) vs constant-time links; measure and justify which design you'd ship.**