Part I — STACK

A. Basics

1. Operation: Push / Pop (LIFO)

Scenario: In the MTN MoMo app, you fill payment details step-by-step; pressing Back removes the last step.

Q1: How does this show the LIFO nature of stacks?

- Each step (e.g., enter amount, choose recipient, confirm) is *pushed* onto the stack as you progress.
- Pressing Back *pops* the most recent step — the last thing you did is the first thing removed. That is exactly LIFO (Last-In, First-Out).
- Intuition: the most recent screen or change sits on top of the stack and is removed first when you go back.

2. Operation: Pop (Undo)

Scenario: In UR Canvas, navigating course modules — pressing Back undoes the last navigation.

Q2: Why is this action similar to popping from a stack?

- Each navigation is pushed onto a navigation stack. Hitting Back pops the top navigation state and returns you to the previous state.
- This mirrors the stack pop operation: undoing the last action restores the prior state.

B. Application

3. Operation: Push (Add to stack)

Scenario: In BK Mobile Banking, transactions appear in history as they occur.

Q3: How could a stack enable the undo function when correcting mistakes?

- When the user performs each editable action (e.g., transfer details, filled form fields), push a record of the action (and its inverse) onto an undo stack.
- To undo, pop the top record and apply its inverse (e.g., revert a changed field or cancel the staged transaction).

- This gives a deterministic way to step backwards action-by-action until the stack is empty or a checkpoint is reached.

4. Operation: Balanced Parentheses Check

Scenario: Irembo registration forms — data entry must be correctly matched (e.g., opening vs closing sections, nested inputs).

Q4: How can stacks ensure forms are correctly balanced?

- Treat each opening element (start of a section, sub-form, or grouped input) as a push onto a stack.
- On encountering a closing element (end of section, submit of a grouped input), check the top of the stack for the matching opener; if it matches, pop it. If not, the form structure is unbalanced.
- After processing all fields, if the stack is empty the form structure is balanced; otherwise there are unmatched openings.
- Practical use: validate nested dynamic fields, conditional subforms, or multi-part inputs before final submission.

C. Logical

5. Operation: Push and Pop sequence

Scenario: Student records tasks in a stack:

Push("CBE notes"), Push("Math revision"), Push("Debate"), Pop(), Push("Group assignment")

Q5: Which task is next (top of stack)?

- Trace: after Push(CBE notes) -> [CBE notes]
- Push(Math revision) -> [CBE notes, Math revision]
- Push(Debate) -> [CBE notes, Math revision, Debate]
- Pop() removes Debate -> [CBE notes, Math revision]
- Push(Group assignment) -> [CBE notes, Math revision, Group assignment] **Answer:**

  Group assignment is at the top and is next.

6. Operation: Undo with multiple Pops

Scenario: During ICT exams, a student undoes 3 recent actions (three pops).

Q6: Which answers remain in the stack after undoing?

- Without the original full push sequence we show the method: remove (pop) the top three items; the remaining items are the earlier actions that were pushed before those three.
- Example: If stack was [A, B, C, D, E] (A bottom, E top) and student undoes 3 actions (3 pops), we pop E, D, C -> remaining [A, B].

Answer (method): After three pops, only the earlier (older) answers remain — those below the top three. (If provided with a particular sequence we can list them exactly.)

D. Advanced Thinking

7. Operation: Pop to backtrack

Scenario: In RwandAir booking, a passenger goes back step-by-step in the form.

Q7: How does a stack enable this retracing process?

- Each form step (passenger info, seat selection, extras, payment) is pushed onto a stack as the passenger proceeds.
- Pressing Back pops the stack to reveal the previous step and its state, enabling exact retracing and restoration of previously-entered data.
- The stack approach makes it easy to restore form state, maintain return points, and implement multi-step undo with minimal state management.

8. Operation: Push words, then Pop to reverse Scenario: Reverse the proverb: "Umwana ni umutware"

Q8: Show how a stack algorithm reverses the proverb.

- Split the phrase into words: ["Umwana", "ni", "umutware"].
- Push each word in order onto the stack: top after pushes -> [Umwana, ni, umutware] (umutware is top).
- Pop repeatedly to get words in reverse order: Pop() -> "umutware", Pop() -> "ni", Pop() > "Umwana".
- Reconstructed reversed phrase: **"umutware ni Umwana"**.

9. Operation: DFS using a stack

Scenario: Student searches shelves in Kigali Public Library (deep search through connected sections).

Q9: Why does a stack suit this case better than a queue?

- DFS explores as far down one branch before backtracking — this is naturally implemented with a stack (explicit or call stack).
- A stack keeps the current path and allows backtracking by popping to the previous branch point; a queue (BFS) would spread search evenly and not dive deep quickly.
- For searching deep, nested shelf structures or following references within a section, DFS (stack) finds deep items faster and uses memory proportional to the search depth rather than the breadth.

10. Operation: Push/Pop for navigation

Scenario: BK Mobile app — moving through transaction history uses push and pop.

Q10: Suggest a feature using stacks for transaction navigation. Suggested feature — "Stepback & Replay" using an action stack:

- Push each viewed transaction or action (view details, filter change, sort) onto a navigation stack with timestamp and view state.
- Add UI buttons: Back (pop single), Fast-Back (pop to previous bookmarked checkpoint), and Replay (pop items into a replay queue to replay navigation in order).
- Use a parallel redo stack: when the user pops (Back), push the popped state onto a

redo stack so they can Redo if desired.

- Benefits: intuitive back/forward navigation, quick restore of filters/sorting, and a replay feature for auditing or tutorial playback.

Part II — QUEUE

A. Basics

1. **Operation: Enqueue / Dequeue (FIFO)**

Scenario: At a restaurant in Kigali, customers are served in order.

Q1: How does this show FIFO behavior?

- Customers enter the line and are *enqueued* at the rear.
- The first customer to arrive is the first to be served (dequeued) from the front.
- This is First-In, First-Out (FIFO) — the earliest arrival gets served first.

2. **Operation: Dequeue (next item leaves first)**

Scenario: In a YouTube playlist, the next video plays automatically.

Q2: Why is this like a dequeue operation?

- Videos are lined up in the playlist order.
- The first video in the queue plays, then is removed (dequeued).
- The next video becomes the new front and plays next — just like a queue's dequeue process.

B. Application

3. **Operation: Enqueue (job submission)**

Scenario: At RRA offices, people waiting to pay taxes form a line.

Q3: How is this a real-life queue?

- Each taxpayer arrives and joins the back of the line (enqueue).
- The front person is served first (dequeue).
- This ensures fairness: arrival time determines service order.

4. **Operation: Queue management**

Scenario: At MTN/Airtel service centers, SIM replacement requests are processed in order.

Q4: How do queues improve customer service?

- Queues prevent confusion and disputes by giving everyone a clear order.
- They reduce waiting stress, as customers know they'll be served when their turn comes.
- Staff can process requests sequentially, improving efficiency and fairness.

Page 3 — Logical (C)

C. Logical

5. **Operation: Sequence of Enqueue/Dequeue Scenario:** Equity Bank operations:

Enqueue("Alice"), Enqueue("Eric"), Enqueue("Chantal"), Dequeue(), Enqueue("Jean") **Q5:**

Who is at the front now?

- Trace: Start → []
- Enqueue(Alice) → [Alice]
- Enqueue(Eric) → [Alice, Eric]
- Enqueue(Chantal) → [Alice, Eric, Chantal]
- Dequeue() removes Alice → [Eric, Chantal]
- Enqueue(Jean) → [Eric, Chantal, Jean] **Answer:** *Eric* is now at the front.

6. **Operation: FIFO message handling**

Scenario: RSSB pension applications handled by arrival order.

Q6: Explain how a queue ensures fairness.

- Each application is enqueued in order of arrival.
- Applications are processed by dequeuing from the front.
- No one can skip ahead — FIFO ensures everyone is treated fairly according to arrival time.

D. Advanced Thinking

7. **Operation: Different queue types** Examples:

- **Linear queue:** People lining up at a wedding buffet — once food runs out, the queue empties.
- **Circular queue:** Buses looping at Nyabugogo — once a bus finishes, it rejoins the loop at the rear.
- **Deque:** Boarding a bus from both front and rear doors — passengers can enter/exit from both ends.

Q7: Explain how each maps to real Rwandan life.

- Wedding buffet: sequential, one-direction service (linear).
- Nyabugogo buses: continuous rotation, like a circular queue.
- Bus boarding from front/rear: access from both ends, like a deque.

8. **Operation: Enqueue orders, Dequeue when ready**

Scenario: Kigali restaurant customers order food and are called when ready.

Q8: How can queues model this process?

- Orders are enqueued in the order they are received.
- The kitchen dequeues the next order to prepare.
- When an order is ready, it is removed from the queue and served to the correct customer.

9. **Operation: Priority queue**

Scenario: At CHUK hospital, emergencies jump the line.

Q9: Why is this a priority queue, not a normal queue?

- In a normal queue, patients are served by arrival order only.
- In a priority queue, emergencies are given higher priority and can bypass non-urgent cases.
- Service order depends on *priority level* rather than just arrival time.

10. **Operation: Enqueue/Dequeue matching system**

Scenario: In a moto/e-bike taxi app, riders wait for passengers.

Q10: How would queues fairly match drivers and passengers?

- Passengers are enqueued when requesting rides.

- Available drivers dequeue the next passenger in line.
- This ensures first-come, first-served matching, preventing unfair skipping.