FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XII · CODE 308

Queue

CUET unit: Queue

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- A Queue is a linear ordered list that follows the First-In-First-Out (FIFO) principle, in contrast to the Stack (LIFO).
- Core Queue operations are ENQUEUE, DEQUEUE, IS EMPTY, IS FULL, PEEK; implement them in Python using the list data type.
- A Deque (Double-Ended Queue) permits insertion and deletion at both ends, so it can simulate both stack and queue behaviour.
- Queues combine conceptual understanding (FIFO vs. LIFO, overflow vs. underflow) with Python code tracing — two skill types that appear repeatedly in the exam, so CUET tests them heavily.
- Real-life and computer-science applications of both Queue and Deque are provided, making application-based MCQs straightforward to set.

📖 Detailed Notes

2.1 Core concepts

- **Queue as a data structure:** A Queue is an ordered linear list of elements having different ends for adding and removing elements. New elements are added at the REAR (also called TAIL) and removed from the FRONT (also called HEAD). (NCERT §4.1, p. 53)
- **FIFO principle:** Queue follows First-In-First-Out (FIFO), also called First Come First Served (FCFS). The element that enters first leaves first; the element longest in the queue is removed first. (NCERT §4.1.1, p. 54)
- **Real-life applications of Queue:** Students in morning assembly, customers at a bank cash counter, vehicles at a petrol pump fuel queue, train ticket waiting-list (W/L1 processed first), customer-service IVRS call queues, single-lane one-way roads, and highway toll-tax booths — all follow FIFO. (NCERT §4.1.2(A), p. 54)
- **Computer-science applications of Queue:** Web servers use a queue to handle thousands of concurrent result-view requests (up to 50 at a time processed serially); OS multitasking systems line up jobs in a queue for processor access on FIFO basis; shared-printer spooling sends print requests to a printer one by one in FIFO order. (NCERT §4.1.2(B), p. 55)
- **ENQUEUE operation:** Inserts a new element at the REAR of the queue. Inserting beyond capacity raises an Overflow exception. (NCERT §4.2, p. 55)

- **DEQUEUE operation:** Removes one element at a time from the FRONT of the queue. Deleting from an empty queue raises an Underflow exception. (NCERT §4.2, p. 55)
- **Supporting operations — IS EMPTY:** Checks whether the queue has any element; used before DEQUEUE to avoid Underflow. (NCERT §4.2, p. 55)
- **Supporting operations — PEEK:** Views (reads) the element at the FRONT without removing it from the queue. (NCERT §4.2, p. 56)
- **Supporting operations — IS FULL:** Checks whether any more elements can be added to avoid Overflow during ENQUEUE. (NCERT §4.2, p. 56)
- **Figure 4.3 — Queue state diagram:** Tracks FRONT (F) and REAR (R) after each enqueue/dequeue showing that FRONT moves right on each dequeue and REAR moves right on each enqueue. (NCERT §4.2, p. 56)
- **Python implementation of Queue using list:** A queue is created with `myQueue = list()`. The `append()` method adds to the end (REAR); `pop(0)` removes from index 0 (FRONT). IS FULL is not needed in Python because Python lists are dynamic. (NCERT §4.3, p. 56–57)
- **Python functions for Queue:** `enqueue(myQueue, element)` uses `myQueue.append(element)`; `dequeue(myQueue)` uses `myQueue.pop(0)`; `isempty(myQueue)` checks `len(myQueue)==0`; `size(myQueue)` returns `len(myQueue)`; `peek(myQueue)` returns `myQueue[0]` without removing it. (NCERT §4.3, p. 56–58)
- **Program 4-1 (bank counter simulation):** Demonstrates a full queue lifecycle: two enqueues, one dequeue, size check, three more enqueues, one more dequeue, then a while loop draining the remaining queue until Underflow (queue is empty). (NCERT §4.3, p. 58–59)
- **Deque (Double-Ended Queue):** Pronounced "deck". An arrangement where addition and removal of elements can happen from any end (head/front or tail/rear). It imposes no restriction on the side for insertion/removal, so it can simulate both a stack and a queue. Also called a Double-Ended Queue. (NCERT §4.4, p. 59)
- **Real-life applications of Deque:** Train ticket counter — a person who already purchased a ticket and returns can rejoin from the front; highway toll booth — vehicles from a fully-served booth can join the front of the adjacent vacant booth's queue. (NCERT §4.4.1, p. 59–60)
- **Computer-science applications of Deque:** Maintaining browser history (URL list) — a deque-like structure deletes least-visited URLs from the end; Do/Undo in text editors; palindrome checking using character-wise insertion and front/rear deletion matching. (NCERT §4.4.1, p. 60)
- **Operations on Deque:** INSERTFRONT (insert at front), INSERTREAR (insert at rear, same as ENQUEUE), DELETIONFRONT (remove from front, same as DEQUEUE), DELETIONREAR (remove from rear). Supporting operations: Is Empty, Peek, Size. (NCERT §4.4.2, p. 60)

- **Algorithm 4.1 — Palindrome check with Deque:** Insert each character using INSERTREAR; then repeatedly remove one character from FRONT and one from REAR using DELETIONFRONT and DELETIONREAR; if both match every iteration until one or zero characters remain, the string is a palindrome. (NCERT §4.4.2, p. 61)
- **Python implementation of Deque:** `myDeque = list()`; `insertFront()` uses `myDeque.insert(0, element)`; `insertRear()` uses `myDeque.append(element)`; `deletionRear()` uses `myDeque.pop()` (no argument); `deletionFront()` uses `myDeque.pop(0)`; `getFront()` returns `myDeque[0]`; `getRear()` returns `myDeque[len(myDeque)-1]`. (NCERT §4.5, p. 62–63)
- **Time complexity of queue operations on a Python list (NCERT §4.3).** `append()` is O(1), but `pop(0)` is O(n) because all remaining elements must shift left. This is why production Python code prefers `collections.deque` over a plain list for queues — although NCERT stays with `list`.
- **Real-world ticket-counter scenario for deque (NCERT §4.4.1, p. 59-60).** A passenger who has already bought a ticket and returns may be allowed to rejoin from the FRONT — this is precisely `insertFront()`, an operation impossible in a plain FIFO queue.
- **Palindrome algorithm step count.** For a string of n characters, Algorithm 4.1 performs at most $\lfloor n/2 \rfloor$ comparisons because each iteration removes one element from each end. Even-length strings end with a 0-element deque; odd-length strings end with a 1-element deque.
- **Program 4-2 (Deque in Python):** A main() function lets the user choose between using the deque as a normal queue (choice=1, insertRear/deletionFront) or as a reverse queue (choice=2, insertRear/deletionRear), demonstrating how a single deque structure supports both modes. (NCERT §4.5, p. 63–64)

2.2 Definitions to memorise

Term	Definition	Page
Queue	An ordered linear list of elements where insertion happens at REAR and deletion at FRONT	53
FIFO	First-In-First-Out — the element that entered first is the first to be removed	54
FCFS	First Come First Served — equivalent name for the FIFO principle used in Queue	54
REAR (TAIL)	The end of the queue where new elements are inserted (enqueued)	54
FRONT (HEAD)	The end of the queue from which elements are removed (dequeued)	54
ENQUEUE	Operation to insert a new element at the REAR of the queue	55

Term	Definition	Page
DEQUEUE	Operation to remove one element from the FRONT of the queue	55
Overflow	Exception raised when an element is inserted into a full queue	55
Underflow	Exception raised when a dequeue operation is attempted on an empty queue	55
IS EMPTY	Supporting operation that checks whether the queue has any element	55
IS FULL	Supporting operation that checks whether the queue has reached its capacity	56
PEEK	Supporting operation that reads the FRONT element without removing it	56
Deque	Double-Ended Queue — an ordered linear list where insertion/deletion can happen at both ends	59
INSERTFRONT	Deque operation to insert a new element at the front	60
INSERTREAR	Deque operation to insert a new element at the rear (same as ENQUEUE)	60
DELETIONFRONT	Deque operation to remove an element from the front (same as DEQUEUE)	60
DELETIONREAR	Deque operation to remove an element from the rear	60
`enqueue(q, x)`	NCERT helper that appends x at the rear of queue q	56-57
`dequeue(q)`	NCERT helper that returns and removes the front element	57
`isempty(q)`	Helper returning True if the queue has no elements	57
`peek(q)`	Helper returning the FRONT element without removing it	56
`size(q)`	Helper returning the number of elements in the queue	58
Print spooling	Use case for queue: print jobs processed in FIFO order	55
OS multitasking	Use case for queue: jobs lined up for CPU in FIFO order	55
Palindrome check (Deque)	Algorithm 4.1 — uses INSERTREAR then alternating DELETIONFRONT/DELETIONREAR to test if a string reads same forwards and backwards	61
URL history	Use case for deque: browser history with least-visited-removed-from-end	60
Undo / Redo (Deque)	Use case for deque: text-editor change stack	60
`insertFront()` / `insertRear()`	Deque helpers using `insert(0, x)` and `append(x)` respectively	62

Term	Definition	Page
`deletionFront()` / `deletionRear()`	Deque helpers using `pop(0)` and `pop()` respectively	62-63

2.3 Diagrams / processes to remember

- **Figure 4.1 (p. 53):** Queue of people at a bank cash counter — illustrates FRONT (Cashier side) and REAR (new arrivals side) in a real-world context.
- **Figure 4.2 (p. 54):** Queue of cars at a petrol pump — shows FIFO for vehicles; the first car in line gets served first.
- **Figure 4.3 (p. 56):** Table showing various stages of queue operations (enqueue z, x, c; dequeue; enqueue v; dequeue; dequeue) — tracks F and R positions after each step. Important for tracing MCQs.
- **Figure 4.4 (p. 59):** Basic deque structure — arrows show Push (insertion) and Pop (deletion) from BOTH the Front (left) and Rear (right) ends simultaneously, unlike a queue which has one-way flow.
- **Figure 4.5 (p. 61):** Status of Deque after 4th iteration of palindrome algorithm for "madam" — shows characters m, a, d, a inserted via INSERTREAR with FRONT on left.
- **Figure 4.6 (p. 61):** Status after removing one character from both ends — 'm' removed from FRONT (removefront) and 'm' from REAR (insertrear arrow shows remaining 'a d a'), used in matching step of palindrome check.

2.4 Common confusions / NTA trap points

- **REAR vs. FRONT direction:** Students confuse which end is REAR and which is FRONT. Remember: REAR = where you join the queue (insertion/ENQUEUE); FRONT = where you leave (deletion/DEQUEUE). NTA distractors often swap these labels.
- **pop(0) vs. pop():** In Python Queue implementation, `pop(0)` removes from FRONT (index 0); `pop()` with no argument removes from the end (REAR) — used in Deque's `deletionRear()`. Mixing these is a classic trap in code-trace questions.
- **Overflow vs. Underflow:** Overflow occurs on ENQUEUE into a full queue; Underflow occurs on DEQUEUE from an empty queue. NTA sometimes presents both conditions in a single scenario and asks which exception is raised.
- **Deque can act as Stack:** If insertion and deletion are done from the same end of a deque, it behaves as a Stack (LIFO). If done from opposite ends, it behaves as a Queue (FIFO). Activity 4.3 and 4.4 test exactly this concept.
- **IS FULL not needed in Python (NCERT §4.3, p. 57).** Because Python lists are dynamic, `IS FULL` is never needed for Python queue/deque implementations.
- **FRONT moves on dequeue, REAR moves on enqueue (NCERT Figure 4.3, p. 56).** Each enqueue increments REAR; each dequeue increments FRONT.

- **Print queue is FIFO (NCERT §4.1.2(B), p. 55).** Print spooling is the most-cited computer-science example. NTA distractor: claims it is LIFO.
- **Deque palindrome check (NCERT Algorithm 4.1, p. 61).** Insert all characters via insertRear, then keep popping one from each end and comparing.
- **Browser history is deque-like (NCERT §4.4.1, p. 60).** Old URLs are dropped from the end when capacity is exceeded.
- `insert(0, x)` **in Python (NCERT §4.5, p. 62).** Inserts x at the front of the list — O(n) operation due to shifting.
- **Single deque can act as either stack or queue (NCERT §4.5 Program 4-2, p. 63-64).** Same data structure, different usage.

🎯 Practice MCQs

Q1. Which of the following correctly describes the working principle of a Queue data structure?

A. Last-In-First-Out (LIFO)

B. First-In-First-Out (FIFO)

C. Random access of elements

D. Insertion and deletion from the same end

Q2. In a Queue, a new element is always inserted at the _____ and removed from the _____.

A. FRONT; REAR

B. REAR; REAR

C. REAR; FRONT

D. FRONT; FRONT

Q3. Consider the following sequence of operations on an initially empty queue: `enqueue(z), enqueue(x), enqueue(c), dequeue(), enqueue(v), dequeue()` What is the current FRONT element of the queue after all these operations?

A. z

B. x

C. c

D. v

📊 PYQ Alignment

Queue and Deque are a regular fixture in CUET Computer Science papers, typically contributing 1–2 direct questions per year on FIFO concept identification, Python code tracing for enqueue/dequeue, and Deque operation matching; students who practise Figure 4.3-style state-tracing exercises and know the Python `pop(0)` vs `pop()` distinction will handle these questions confidently. See PYQ archive for Computer Science.