**10. Explain Priority Queue**

A **Priority Queue (PQ)** is a special type of queue where each element is assigned a *priority*.

* In a **normal queue**, insertion happens at rear and deletion at front (FIFO).
* In a **priority queue**, elements are dequeued based on priority rather than order of arrival.

**Types:**

* **Ascending PQ**: element with *smallest* key removed first.
* **Descending PQ**: element with *largest* key removed first.

**Operations:**

* **Insert**: place element at proper position according to priority.
* **Delete**: remove the highest (or lowest) priority element at the front.

**Applications:**

* CPU job scheduling
* Simulation systems (bank counters, hospital patients)
* Network routing (packets with higher priority transmitted first).

**11. What is Stack? Applications**

**Definition:** A stack is a linear data structure in which insertion and deletion take place only at one end, called the **TOP**. It follows the **Last-In First-Out (LIFO)** principle.

**Applications of Stack:**

1. **Backtracking**: Undoing moves in puzzles or traversing mazes.
2. **Function calls**: Each call pushes local data onto the call stack; return pops it back.
3. **Recursion**: System uses stack to manage recursive function calls.
4. **Expression evaluation**: Conversion from infix to postfix/prefix and evaluation.
5. **String reversal**: Push each char, pop to reverse.
6. **Undo operation**: Editors use stack to revert recent actions.

**Why used:** Stack is efficient for situations where *reversing order* or *tracking nested operations* is needed.

**12. Algorithm to implement Stack using Array**

**PUSH Operation:**

Procedure PUSH(S, TOP, X)

1. If TOP ≥ N then

Print "STACK OVERFLOW"

Return

2. TOP ← TOP + 1

3. S[TOP] ← X

4. Return

**POP Operation:**

Function POP(S, TOP)

1. If TOP = 0 then

Print "STACK UNDERFLOW"

Exit

2. TOP ← TOP - 1

3. Return S[TOP+1]

**14. What is Queue ADT? Operations**

A **Queue** is a linear data structure that follows **FIFO (First In First Out)**.

* Insertion happens at the **rear**.
* Deletion happens at the **front**.

**Operations of Queue ADT:**

1. **Enqueue(x):** Insert element at rear.
2. **Dequeue():** Remove element from front.
3. **Peek():** Display front element without removing it.
4. **isEmpty():** Check if queue has no elements.
5. **isFull():** Check if queue is full (array implementation).

**16. Algorithm to implement Queue using Array**

**ENQUEUE:**

Procedure ENQUEUE(Q, F, R, N, Y)

1. If R = N then Print "Queue Full" and Return

2. If F = 0 then F ← 1

3. R ← R + 1

4. Q[R] ← Y

**DEQUEUE:**

Function DEQUEUE(Q, F, R, N)

1. If F = 0 then Print "Queue Empty" and Return

2. Y ← Q[F]

3. If F = R then F ← R ← 0

Else F ← F + 1

4. Return Y

**17. Issue with Simple Queue & Solution**

**Issue:** In array implementation, after many dequeues, empty slots form at the front but cannot be reused. Eventually, rear reaches the end → **false overflow** even if free space exists.

**Solution:** **Circular Queue** — treat array as circular. Front and rear indices wrap around, reusing empty slots. This avoids wastage and increases efficiency.

**18. Priority Queue: Definition, Algorithm**

**Definition:** A queue where each element has a priority. The element with higher priority is dequeued first.

**Array Implementation Algorithm:**

Procedure PQINSERT(Q, N, ITEM, PRIORITY)

1. If N = MAX then Print "Overflow" Return

2. i ← N

3. While i>0 and Q[i].priority < ITEM.priority

Q[i+1] ← Q[i]

i ← i - 1

4. Q[i+1] ← ITEM

5. N ← N+1

Function PQDELETE(Q, N)

1. If N=0 then Print "Underflow" Return

2. ITEM ← Q[1] // front has highest priority

3. Shift elements left

4. N ← N-1

5. Return ITEM

**Applications:** CPU scheduling, packet routing, hospital triage.

**19. Undo Mechanism in Word Editor**

* Requirement: Ability to undo the last change quickly.
* Suitable DS: **Stack** (LIFO) because last action undone first.
* Best-case Complexity: **O(1)** (simple push/pop).
* Worst-case Complexity: **O(1)** (still constant since only top is accessed).

Thus, stack is ideal.

**21. Circular Queue & Algorithm**

**Definition:** A queue in which last position connects back to first, forming a circle. Allows reuse of deleted slots.

**Algorithm:**

Procedure CENQUEUE(Q, F, R, N, X)

1. If ( (R+1) mod N ) = F then Print "Queue Full"

2. Else R ← (R+1) mod N

Q[R] ← X

If F = -1 then F ← 0

Function CDEQUEUE(Q, F, R, N)

1. If F = -1 then Print "Empty"

2. X ← Q[F]

3. If F = R then F = R = -1

Else F ← (F+1) mod N

4. Return X

**22. Double Ended Queue**

**Definition:** A queue allowing insertion and deletion at both ends.

**Variants:**

* Input Restricted Dequeue: Insertion only at rear, deletion both ends.
* Output Restricted Dequeue: Deletion only at front, insertion both ends.

**Applications:** Palindrome checking, real-time simulations, browsers’ forward/backward navigation.

**23. Algorithms of Dequeue Operations**

(As given in slides)

**Rear Insertion:** Increment rear, add item.  
**Rear Deletion:** Remove item at rear, decrement rear.  
**Front Insertion:** Decrement front, add item.  
**Front Deletion:** Remove item at front, increment front.

**24. Applications of Stack**

* Backtracking (maze solving)
* Function calls
* Recursion
* Expression conversion/evaluation
* String reversal
* Undo mechanism in editors

**25. Need of Circular Queue**

Solves the **false overflow** problem of linear queues by reusing freed slots using modular arithmetic. Improves memory utilization.

**26. PUSH Algorithm**

(Already written in Q12)

**27. Infix → Postfix: a + b \* c + d / b + a \* c + d**

Slides give result: **abc\*+db/+ac\*+d+**  
**Rank = 1** (valid).

**28. Convert (A + B) - C \* D / (E - F / G)**

**Postfix:** AB+CD\*EFG/-/-

**29. Variants of Dequeue**

* Input Restricted Dequeue
* Output Restricted Dequeue

**30. ENQUEUE Algorithm**

(Same as Q16 but only insertion)

**31. Infix → Postfix (A+B\*C/D-E+F/G/(H+I))**

**Postfix:** ABC\*D/+E-FG/HI+/+  
**Rank:** 1 → valid.

**32. Expression Evaluation (2 + 10 \* 4 / 5 – (9 – 3))**

Stack evaluation steps (as in slides) lead to result = **4**.

**33. Dequeue Insertion/Deletion + Palindrome**

* **Insertion/Deletion:** At both front & rear (algorithms given in Q23).
* **Palindrome check:** Insert string in deque; compare/delete front & rear. If all pairs match, string is palindrome.
* **Complexity:** Insert/delete = O(1); palindrome check = O(n) worst-case, O(1) best-case.