Viva Q&A

Week 1 – Arrays

Q1. What's the time complexity of left rotating an array by *k* positions? Can you do it in O(n) without extra space?
Ans. Time complexity = O(n). Yes, using reversal algorithm (reverse 3 parts of array) → no extra space.

Q2. How can you find the element occurring odd number of times using XOR, and why does it work?
Ans. XOR of all elements gives the odd-occurring element because x^x=0 and x^0=x.

Q3. If I give you two numbers in an array, how do you find the minimum distance between them in a single traversal?
Ans. Keep track of last seen index of either number, update min distance whenever the other number is found.

Week 2 – Matrices

Q4. Why does searching in a row and column sorted matrix take O(n) time, not O(n^2)?
Ans. Start from top-right corner → move left if bigger, down if smaller → each move reduces row or column → O(m+n).

Q5. How do you find the row with maximum 1's in a boolean matrix efficiently?
Ans. Start from top-right, move left if 1, down if 0. O(m+n) approach.

Q6. Explain how you rotate a matrix 90° clockwise. Why use transpose + reverse?
Ans. Transpose converts rows → cols. Reversing each row then gives 90° rotation. Cleaner and in-place.

Week 3 – Stacks (Array Implementation)

Q7. Define stack ADT. What is the difference between stack and array?
Ans. Stack = LIFO ADT with push/pop. Array = data storage, random access possible. Stack restricts access.

Q8. What causes stack overflow and underflow?
Ans. Overflow → pushing in full stack. Underflow → popping from empty stack.

Q9. How can a stack be used to check balanced parentheses?
Ans. Push opening brackets, pop on closing → if matches fine, else unbalanced.

Q10. What's the logic behind longest valid parentheses using stack?
Ans. Push indices, pop when matching found, keep track of length from last unmatched index.

Week 4 – Advanced Stack Applications

Q11. Can you reverse a string without stack? Difference?
Ans. Yes, by swapping from both ends. Stack method uses LIFO property; swapping is in-place.

Q12. How to implement two stacks in one array without wasting space?
Ans. Start one stack from left, other from right, grow towards center.

Q13. Why use stack for evaluating postfix expressions?
Ans. Postfix removes precedence issues. Stack allows operand storage and operator application easily.

Week 5 – Queues

Q14. Define queue ADT. How is it different from stack?
Ans. Queue = FIFO ADT (enqueue, dequeue). Stack = LIFO.

Q15. What's the main problem with simple array queue? How does circular queue fix it?
Ans. Simple queue wastes space after dequeues. Circular queue reuses space using modulo.

Q16. How do you reverse a queue using recursion?
Ans. Dequeue element, recursively reverse rest, enqueue element back.

Q17. Difference between input-restricted and output-restricted deque?
Ans. Input-restricted: insert only at one end, delete both ends. Output-restricted: delete only one end, insert both ends.

Week 6 – Queue–Stack Hybrid

Q18. How to implement stack using two queues? Which is efficient?
Ans. Two methods: push costly (move elements every time) or pop costly. Push-costly usually better.

Q19. How to implement queue using two stacks? Which operation is costly?
Ans. Use two stacks: enqueue = push in stack1, dequeue = transfer stack1 → stack2 when needed. Dequeue is costly.

Q20. Why use modular arithmetic in circular queues?
Ans. To wrap indices around array → (front+1)%size.

Week 7 – Linked Lists

Q21. Advantages of linked list over arrays?
Ans. Dynamic size, easy insertion/deletion. No shifting needed.

Q22. Find middle element without size?
Ans. Slow–fast pointer method (slow moves 1 step, fast 2 steps).

Q23. Difference between singly and doubly linked list?
Ans. Singly: one pointer (next). Doubly: two pointers (next, prev), allows bidirectional traversal.

Q24. Time complexity of insertion at beginning vs end?
Ans. Beginning = O(1). End = O(n) unless tail pointer is maintained (then O(1)).

Q25. Can you implement stack/queue using linked list?
Ans. Yes. Stack: push/pop at head. Queue: enqueue at tail, dequeue at head.

Q26. What happens to memory when node deleted?
Ans. Freed (using free() in C), pointer lost if not managed.

---

small **Tricky questions**

Q. Why is return type struct node* in linked list functions?
Ans. Because functions often return new head pointer after modification.

Q. Can you detect a cycle in linked list? How?
Ans. Yes, Floyd's cycle detection (slow–fast pointers).

Q. If you had to implement undo/redo in text editor, which DS would you use?
Ans. Two stacks → one for undo, one for redo.

Definition

ADT (Abstract Data Type) = A *mathematical model* + a *set of operations* defined on that model, without worrying about how it's implemented.

It tells you **what operations** you can do, not **how** they're done internally.

Examples

- **Stack ADT** → operations: push, pop, peek, isEmpty.
- **Queue ADT** → operations: enqueue, dequeue, isFull, isEmpty.

Implementation can be different (arrays, linked lists), but the *interface* stays the same.

Viva Revision Notes (Short Theory)

1. Arrays

- Fixed size, contiguous memory, random access in O(1).
- Left rotation: reversal algorithm in O(n).
- XOR trick finds odd occurring element.

2. Matrices

- Row+col sorted search = O(m+n).
- Rotate 90° = transpose + reverse.
- Row with max 1's: top-right corner method.

3. Stack (LIFO)

- ADT with push, pop, peek.
- Overflow = push in full stack, Underflow = pop from empty.
- Applications: balanced parentheses, undo/redo, recursion.
- Infix → Postfix evaluation uses stack.

4. Queue (FIFO)

- ADT with enqueue, dequeue.
- Simple queue wastes space → fix with circular queue (modulo).
- Variants:
  - Deque (double ended).
  - Priority queue.
- Applications: scheduling, buffers, resource management.

5. Stack–Queue Hybrids

- Stack using 2 queues → push or pop costly.
- Queue using 2 stacks → dequeue costly.

6. Linked List

- Dynamic size, insertion/deletion easy, no shifting.
- Singly: next pointer.
- Doubly: next + prev.
- Middle element: slow–fast pointer.
- Stack/queue can be built using linked list.
- Cycle detection: Floyd’s algo.

7. ADT (Abstract Data Type)

- Defines *what operations* a DS supports, not *how implemented*.
- Eg: Stack ADT → push, pop; Queue ADT → enqueue, dequeue.