■ Module 1: Introduction (Q11–Q15)

11. Static vs Dynamic Data Structures

Static data structures have fixed size (e.g., arrays). Memory is allocated at compile time, which may cause wastage or overflow. Dynamic structures (linked lists, trees) allocate memory at runtime, allowing flexible size. Static structures provide faster access, while dynamic structures support efficient insertion/deletion. Choice depends on application needs. Static is suitable when size is known; dynamic is better for unpredictable data. This distinction is important in memory management and performance optimization.

---

12. Space Complexity

Space complexity measures total memory used by an algorithm, including input space and auxiliary space. It is expressed using asymptotic notation like O(n). Efficient algorithms minimize space usage. For example, merge sort uses extra space O(n), while quick sort uses O(log n). Space complexity is crucial in systems with limited memory. Optimizing space improves overall performance and scalability.

---

13. Time Complexity Examples

Different algorithms have different time complexities. Constant time O(1) is fastest, e.g., array access. Linear time O(n) occurs in traversal. Logarithmic O(log n) appears in binary search. Quadratic $O(n^2)$ is seen in bubble sort. Understanding these helps compare algorithms. Lower complexity means better performance for large inputs. Time complexity is a key factor in algorithm design.

---

14. Abstract Data Type (ADT)

ADT defines data structure behavior without specifying implementation. It focuses on operations like insert, delete, and search. For example, stack ADT defines push/pop but not how they are implemented. This abstraction improves modularity and flexibility. Programmers can change implementation without affecting usage. ADTs are widely used in software design.

Linear queue wastes space when front moves forward. Circular queue reuses space by wrapping around. Circular queue is more memory efficient. Both have O(1) operations. Circular queue is preferred in real-time systems.

---

15. Applications of Priority Queue

Priority queues are used in scheduling, shortest path algorithms, and simulations. They process elements based on priority. Implemented using heaps for efficiency (O(log n)). They are essential in operating systems and networking.

■ Module 3: Linked Lists (Q11–Q15)

11. Reverse a Linked List

Reversing a linked list involves changing pointer directions. Use three pointers: previous, current, next. Traverse list and reverse links. Time complexity O(n). It is a common interview and exam question.

---

12. Detect Loop in Linked List

Loop detection uses Floyd's cycle algorithm (slow and fast pointers). If pointers meet, loop exists. Time complexity O(n). Important for avoiding infinite loops.

---

13. Advantages of Doubly Linked List

Allows bidirectional traversal. Easier deletion as previous pointer is available. Useful in navigation systems. Requires extra memory. Improves efficiency in certain operations.

---

14. Circular vs Singly Linked List

Circular list has no NULL pointer; last node connects to first. It supports continuous traversal. Singly list ends with NULL. Circular lists are useful in scheduling algorithms.

---

15. Applications of Linked List

Used in dynamic memory allocation, stacks, queues, graphs, and hash tables. Efficient for insertion/deletion. Widely used in real-world systems.

---

15. Importance of Complexity Analysis

Complexity analysis helps choose efficient algorithms. It predicts performance without execution. Important for large-scale systems like databases and networks. It ensures optimal use of resources. Without analysis, programs may become slow or inefficient. It is essential for competitive programming and real-world applications.

■ Module 2: Stacks & Queues (Q11–Q15)

11. Prefix Expression Evaluation

Prefix expressions place operators before operands. Evaluation uses stack by scanning from right to left. Operands are pushed, operators apply on top elements. It avoids parentheses and simplifies computation. Time complexity is O(n). Used in compilers and expression evaluation systems.

---

12. Postfix Expression Evaluation

Postfix expressions place operators after operands. Evaluation uses stack by scanning left to right. Operands are pushed; operators pop two values and push result. It is efficient and avoids precedence rules. Time complexity O(n). Widely used in calculators and compilers.

---

13. Overflow and Underflow

Overflow occurs when inserting into full structure; underflow occurs when removing from empty structure. These conditions must be handled to avoid errors. In stacks and queues, checks are implemented before operations. Proper handling ensures program reliability.

---

14. Circular Queue vs Linear Queue

■ Module 4: Searching, Sorting & Hashing (Q11–Q15)

11. Binary Search Algorithm

Binary search divides sorted array into halves. Compare middle element and reduce search space. Time complexity O(log n). Efficient for large datasets. Requires sorted data.

---

12. Stability in Sorting

A sorting algorithm is stable if it preserves order of equal elements. Merge sort and insertion sort are stable; quick sort is not. Stability is important in multi-key sorting.

---

13. In-place Sorting

In-place sorting uses constant extra space. Examples: quick sort, heap sort. It reduces memory usage. Important for large datasets.

---

14. Hash Function

Hash function maps keys to indices. Good hash functions distribute data uniformly. It minimizes collisions. Essential for efficient hashing.

.