Q1. [Definition] What is the time complexity to insert a node at the beginning of a singly linked list?
A. O(n)
B. O(1)
C. O(log n)
D. O(n log n)
Answer: B
Explanation: Insertion at the head takes constant time since no traversal is needed.

Q2. [Definition] What does the 'next' pointer in a singly linked list node typically store?
A. Data value
B. Index of the next node
C. Reference to the next node
D. Reference to the previous node
Answer: C
Explanation: It holds the reference to the next node in the list.

Q3. [Definition] What is the key advantage of using a linked list over an array?
A. Constant time random access
B. Better sorting algorithms
C. Dynamic size and efficient insertion/deletion
D. Lower memory usage
Answer: C
Explanation: Linked lists grow dynamically and allow O(1) insertions/deletions at the head/tail.

Q4. [Definition] Which condition indicates the end of a singly linked list?
A. node.data == -1
B. node.next == null
C. node.index == 0
D. node.next == node
Answer: B
Explanation: When the next pointer of a node is null, the list has ended.

Q5. [Definition] Which of the following operations is costly in a singly linked list?
A. Insertion at beginning
B. Deletion at beginning
C. Insertion at end
D. Traversal
Answer: C
Explanation: Insertion at the end of a singly linked list requires traversal, which takes O(n) time.

Q6. [Moderate] What will the following iterative function return?

```
Node reverse(Node head) {
```

```
    Node prev = null;
    Node curr = head;
    while (curr != null) {
        Node next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

A. Returns null
B. Returns reversed linked list's head
C. Causes infinite loop
D. Returns original head

Answer: B

Explanation: The function iteratively reverses the linked list and returns the new head (last original node).

Q7. [Moderate] Which line is incorrect in this recursive reversal implementation?

```
Node reverse(Node head) {
    if (head == null || head.next == null) return head;
    Node newHead = reverse(head.next);
    head.next.next = head;
    head.next = null;
    return newHead;
}
```

A. Line 1
B. Line 2
C. Line 3
D. No line is incorrect

Answer: D

Explanation: This is a correct and classic recursive approach for reversing a linked list.

Q8. [Moderate] What is the base case in a recursive reversal of a linked list?

A. When head == head.next
B. When head == null or head.next == null
C. When tail == null
D. When list size is even

Answer: B

Explanation: Base case is when the list is empty or has only one node.

Q9. [Moderate] In iterative reversal, why do we set curr.next = prev?

A. To move to the next node
B. To reverse the direction of the link
C. To assign null to next

D. To skip current node
Answer: B
Explanation: This changes the direction of the link for reversal.

Q10. [Moderate] What is the time complexity of both iterative and recursive reversal of a singly linked list?
A. O(n log n)
B. O(log n)
C. O(n)
D. O(1)
Answer: C
Explanation: Every node is visited once in both approaches, so time complexity is O(n).

Q1. [Definition] What is the time complexity to insert a node at the beginning of a singly linked list?
A. O(n)
B. O(1)
C. O(log n)
D. O(n log n)
Answer: B
Explanation: Insertion at the head takes constant time since no traversal is needed.

Q2. [Definition] What does the 'next' pointer in a singly linked list node typically store?
A. Data value
B. Index of the next node
C. Reference to the next node
D. Reference to the previous node
Answer: C
Explanation: It holds the reference to the next node in the list.

Q3. [Definition] What is the key advantage of using a linked list over an array?
A. Constant time random access
B. Better sorting algorithms
C. Dynamic size and efficient insertion/deletion
D. Lower memory usage
Answer: C
Explanation: Linked lists grow dynamically and allow O(1) insertions/deletions at the head/tail.

Q4. [Definition] Which condition indicates the end of a singly linked list?
A. node.data == -1
B. node.next == null
C. node.index == 0
D. node.next == node
Answer: B
Explanation: When the next pointer of a node is null, the list has ended.

Q5. [Definition] Which of the following operations is costly in a singly linked list?

A. Insertion at beginning
B. Deletion at beginning
C. Insertion at end
D. Traversal

Answer: C

Explanation: Insertion at the end of a singly linked list requires traversal, which takes O(n) time.

Q6. [Moderate] What will the following iterative function return?

```
Node reverse(Node head) {
   Node prev = null;
   Node curr = head;
   while (curr != null) {
      Node next = curr.next;
      curr.next = prev;
      prev = curr;
      curr = next;
   }
   return prev;
}
```

A. Returns null
B. Returns reversed linked list's head
C. Causes infinite loop
D. Returns original head

Answer: B

Explanation: The function iteratively reverses the linked list and returns the new head (last original node).

Q7. [Moderate] Which line is incorrect in this recursive reversal implementation?

```
Node reverse(Node head) {

   if (head == null || head.next == null) return head;

   Node newHead = reverse(head.next);

   head.next.next = head;

   head.next = null;

   return newHead;

}
```

A. Line 1
B. Line 2

C. Line 3
D. No line is incorrect
Answer: D
Explanation: This is a correct and classic recursive approach for reversing a linked list.

Q8. [Moderate] What is the base case in a recursive reversal of a linked list?
A. When head == head.next
B. When head == null or head.next == null
C. When tail == null
D. When list size is even
Answer: B
Explanation: Base case is when the list is empty or has only one node.

Q9. [Moderate] In iterative reversal, why do we set curr.next = prev?
A. To move to the next node
B. To reverse the direction of the link
C. To assign null to next
D. To skip current node
Answer: B
Explanation: This changes the direction of the link for reversal.

Q10. [Moderate] What is the time complexity of both iterative and recursive reversal of a singly linked list?
A. O(n log n)
B. O(log n)
C. O(n)
D. O(1)
Answer: C
Explanation: Every node is visited once in both approaches, so time complexity is O(n).

Q1. [Definition] What is the main idea behind Floyd's Cycle Detection Algorithm?
A. Using hashing to store visited nodes
B. Using recursion
C. Two pointers moving at different speeds
D. Using array index for cycle tracking
Answer: C
Explanation: Floyd's algorithm uses a slow and a fast pointer to detect a cycle by checking if they meet.

Q2. [Definition] What does it indicate if two pointers (slow and fast) meet in Floyd's algorithm?
A. Linked list is empty
B. No cycle exists
C. A cycle is present in the linked list
D. Head is null
Answer: C
Explanation: Meeting of slow and fast pointers confirms the presence of a loop in the linked list.

Q3. [Definition] What is the time complexity of Floyd's Cycle Detection Algorithm?
A. $O(n^2)$
B. O(log n)
C. O(n)
D. O(1)
Answer: C
Explanation: Floyd's algorithm visits each node at most twice, hence linear time.

Q4. [Definition] What happens if a loop exists but only slow pointer is used?
A. Always detects the loop
B. Detects loop but takes $O(n^2)$ time
C. May enter infinite loop
D. Can't detect the loop
Answer: C
Explanation: Without a fast pointer, a slow pointer alone may enter an infinite loop inside the cycle.

Q5. [Definition] Which of the following is the correct pointer movement for Floyd's algorithm?
A. Slow moves 2 steps, Fast 1 step
B. Both move 1 step
C. Slow moves 1 step, Fast 2 steps
D. Both move 2 steps
Answer: C
Explanation: The slow pointer moves by 1 step and fast by 2 steps per iteration.

Q6. [Moderate] What is the purpose of resetting one pointer to head after detection in Floyd's algorithm?
A. To find the loop's entry point
B. To exit the loop
C. To delete the last node
D. To free memory
Answer: A
Explanation: After detection, resetting one pointer helps locate the starting node of the loop.

Q7. [Moderate] What will this code output if a loop exists?

```
Node slow = head, fast = head;
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
    if (slow == fast) {
        System.out.println("Loop detected");
        break;
    }
}
```

A. Compilation error
B. Infinite loop
C. Loop detected
D. NullPointerException
Answer: C
Explanation: The code is a correct implementation of Floyd's cycle detection.

Q8. [Moderate] Which line is missing for loop removal after detecting the starting node of the loop?

```
Node ptr1 = head;
Node ptr2 = meetingPoint;
while (ptr1.next != ptr2.next) {
    ptr1 = ptr1.next;
    ptr2 = ptr2.next;
}
// Missing Line?
```

A. ptr1 = null;
B. ptr2.next = null;
C. ptr2 = ptr2.next;
D. head = ptr1;
Answer: B
Explanation: Setting ptr2.next = null removes the loop by breaking the cycle.

Q9. [Moderate] What does the following function do?

```
boolean detectLoop(Node head) {
    Node slow = head, fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow == fast) return true;
    }
    return false;
}
```

A. Detects if linked list has even length
B. Detects a loop in the linked list
C. Reverses the linked list
D. Deletes duplicate nodes
Answer: B
Explanation: This is the standard implementation of Floyd's cycle detection.

Q10. [Moderate] After detecting a loop, how do we find the number of nodes in the loop?
A. Count nodes while traversing from head
B. Count iterations between first and second meet
C. Keep one pointer at meeting point and move other till it meets again
D. Loop count = total nodes – 1
Answer: C
Explanation: Keeping one pointer fixed and moving the other until they meet again counts the loop length.

Q1. [Definition] What is the goal of merging two sorted linked lists?
A. Sort both lists independently
B. Combine them into a new sorted list
C. Reverse both lists
D. Remove duplicates only
Answer: B
Explanation: Merging two sorted lists means combining them into a single sorted list.

Q2. [Definition] Which technique is commonly used to merge two sorted lists in-place?
A. Brute-force copying
B. Recursion or iterative traversal
C. Bubble sort
D. Inversion count

Answer: B
Explanation: Merging sorted lists is commonly done using recursion or a two-pointer iterative approach.

Q3. [Definition] What is the time complexity of merging two sorted linked lists with lengths m and n?
A. O(1)
B. O(m * n)
C. O(m + n)
D. O(m log n)
Answer: C
Explanation: Each node is visited once during the merge, resulting in linear time.

Q4. [Definition] In the context of linked lists, what does "intersection" mean?
A. Same data at a node
B. Same index value
C. Two lists share a common node by reference
D. Two lists start with the same value
Answer: C
Explanation: Intersection means both lists point to the same node (same memory reference), not just same value.

Q5. [Definition] What must be true for two singly linked lists to intersect?
A. They must be of the same length
B. They must have the same head
C. Their last node must be the same (by reference)
D. They must have same data
Answer: C
Explanation: If the tail node of both lists is the same object, an intersection exists.

Q6. [Moderate] What is the return of this code snippet for merging sorted lists?

```
Node merge(Node a, Node b) {
    if (a == null) return b;
    if (b == null) return a;
    if (a.data < b.data) {
        a.next = merge(a.next, b);
        return a;
    } else {
        b.next = merge(a, b.next);
```

```
        return b;
    }
}
```

A. Null
B. Unsorted list
C. Merged sorted list
D. Infinite loop
Answer: C
Explanation: This is the correct recursive method to merge two sorted linked lists.

Q7. [Moderate] Which step is essential before checking for intersection using two pointers?
A. Reverse both lists
B. Count the lengths and align the starting points
C. Sort both lists
D. Set one pointer to null
Answer: B
Explanation: Aligning the start of the longer list helps both pointers reach the intersection node simultaneously.

Q8. [Moderate] What will this code output if two lists intersect at node with data = 30?

```
Node getIntersection(Node a, Node b) {
    while (a != b) {
        a = (a == null) ? b : a.next;
        b = (b == null) ? a : b.next;
    }
    return a;
}
```

A. null
B. 30
C. Compilation error
D. Infinite loop
Answer: B
Explanation: This is a classic two-pointer technique to find the intersection node.

Q9. [Moderate] What is the time complexity of finding the intersection point using two pointers?
A. $O(n^2)$
B. $O(n + m)$
C. $O(n \log m)$
D. $O(1)$
Answer: B
Explanation: Each pointer travels the total length of both lists once.

Q10. [Moderate] What is the base condition for stopping recursive merging of two sorted lists?
A. When both lists are equal
B. When list size is even
C. When either list is null
D. When values are repeated
Answer: C
Explanation: The merge ends when either list is fully traversed (null).

Q1. [Definition] What is the primary goal of LRU Cache?
A. Store sorted data
B. Minimize memory usage
C. Remove most recently used data
D. Remove least recently used data when capacity is exceeded
Answer: D
Explanation: LRU Cache evicts the least recently used item when the capacity limit is reached.

Q2. [Definition] What data structures are commonly used to implement an efficient LRU Cache?
A. Stack and Array
B. Queue and Set
C. HashMap and Doubly Linked List
D. Heap and TreeMap
Answer: C
Explanation: HashMap provides O(1) lookup, and doubly linked list maintains usage order.

Q3. [Definition] Why is a doubly linked list preferred over singly linked list in LRU cache design?
A. To sort elements
B. To allow backward traversal during removal
C. To save space
D. To reverse the list efficiently
Answer: B

Explanation: Doubly linked lists allow O(1) removal of nodes from the middle using both next and prev pointers.

Q4. [Definition] What is the time complexity of get() and put() operations in an LRU Cache using LinkedList and HashMap?
A. O(n)
B. O(log n)
C. O(1)
D. O(n log n)
Answer: C
Explanation: Both operations take constant time due to the combination of HashMap (O(1) lookup) and Linked List (O(1) node manipulation).

Q5. [Definition] What is stored in the HashMap of an LRU Cache?
A. Only keys
B. Only values
C. Keys mapped to linked list nodes
D. Indexes
Answer: C
Explanation: HashMap stores key → Node mappings to allow O(1) access and updates.

Q6. [Moderate] Which operation should be performed when a cache hit occurs in LRU cache?
A. Do nothing
B. Move accessed node to front of the list
C. Remove the tail node
D. Increase cache size
Answer: B
Explanation: On a hit, the node is moved to the front (most recently used) to maintain LRU order.

Q7. [Moderate] Which part of the linked list represents the least recently used item?
A. Head
B. Middle
C. Tail
D. None
Answer: C
Explanation: The tail represents the least recently used node in the doubly linked list.

Q8. [Moderate] What happens during a cache miss when the cache is full?
A. Do nothing
B. Remove tail node and insert new node at head
C. Throw exception

D. Overwrite the last node
Answer: B
Explanation: LRU cache removes the least recently used node (tail) and inserts the new one at the front.

Q9. [Moderate] What is the role of dummy head and dummy tail in LRU design?
A. To initialize cache
B. To simplify insertion/removal logic
C. To mark memory limits
D. To reduce cache size
Answer: B
Explanation: Dummy nodes simplify edge case handling during node insertion/removal.

Q10. [Moderate] Which line is incorrect in the following code for removing a node?

```
void remove(Node node) {
    node.prev.next = node.next;
    node.next.prev = node.prev;
}
```

A. Line 1
B. Line 2
C. Line 3
D. No line is incorrect
Answer: D
Explanation: The code correctly removes a node from a doubly linked list in O(1) time.

Q1. [Definition] A train is moving with coaches linked one after another. If each coach has a reference to the next coach, what data structure does this represent?
A. Stack
B. Queue
C. Array
D. Singly Linked List
Answer: D
Explanation: Each coach having a reference to the next is like a singly linked list, where each node points to the next.

Q2. [Definition] A snake game maintains the body of the snake as it grows. Each segment connects to the next. What is the best data structure to represent this behavior?
A. Array
B. Graph
C. Linked List
D. Matrix
Answer: C

Explanation: As the snake grows or shrinks dynamically, a linked list efficiently handles such dynamic changes.

Q3. [Definition] In a treasure hunt game, each clue leads to the next. Which data structure does this mimic?
A. HashMap
B. Tree
C. Linked List
D. Stack
Answer: C
Explanation: Each clue points to the next, just like each node in a linked list points to the next node.

Q4. [Definition] A photo slideshow app allows viewing photos in sequence and then reversing back one step. Which linked list type is best suited here?
A. Singly Linked List
B. Doubly Linked List
C. Circular Linked List
D. Array
Answer: B
Explanation: Doubly linked lists allow traversal in both forward and backward directions, ideal for a slideshow with next and previous buttons.

Q5. [Definition] A clock has 12 hours and keeps rotating in a loop. If we model this using a linked list where the last node points back to the first, which type is it?
A. Doubly Linked List
B. Singly Linked List
C. Circular Linked List
D. Queue
Answer: C
Explanation: In a circular linked list, the last node points to the head, forming a loop like a clock.

Q6. [Moderate] Alice is building a playlist. She can add songs at the beginning and remove from the end. Which data structure will make end deletion slower?
A. Stack
B. Queue
C. Singly Linked List
D. Doubly Linked List
Answer: C
Explanation: Deleting from the end of a singly linked list requires full traversal, making it slower.

Q7. [Moderate] In a museum exhibit, people move through connected rooms. A guard wants to detect if a visitor goes in circles. Which algorithm helps detect such loops?
A. Binary Search
B. DFS
C. Floyd's Cycle Detection
D. Kadane's Algorithm
Answer: C
Explanation: Floyd's algorithm (slow and fast pointers) can detect cycles, like someone looping through rooms.

Q8. [Moderate] A person is trying to read a book backward. Each page has a reference to the previous page. What linked list is most suitable here?
A. Singly Linked List
B. Doubly Linked List
C. Circular Linked List
D. Stack
Answer: B
Explanation: Doubly linked list allows backward traversal using the prev pointer.

Q9. [Moderate] In a food delivery chain, each kitchen passes an order to the next branch. If one kitchen forwards the order back to a previous one by mistake, what problem occurs?
A. Stack overflow
B. Array out of bounds
C. Infinite loop in linked list
D. Null pointer exception
Answer: C
Explanation: If a node (kitchen) points back to a previous one, it creates a loop, leading to infinite traversal.

Q10. [Moderate] A magician shuffles cards and creates a circular arrangement. To find the first card again after multiple passes, which traversal technique helps?
A. Linear search
B. BFS
C. Circular traversal
D. Stack traversal
Answer: C
Explanation: Circular traversal loops through the list continuously, eventually reaching the first card again.

Q1. [Definition] In a singly linked list, what condition must be true to reverse the list iteratively?
A. The list must be sorted
B. At least two nodes exist
C. Head should not be null

D. Nodes must be in descending order
Answer: C
Explanation: If the head is null, reversal is not required.

Q2. [Definition] Which of the following can be used to detect a loop with O(n) space?
A. Floyd's algorithm
B. Recursion
C. HashSet to track visited nodes
D. Sorting
Answer: C
Explanation: HashSet can track visited nodes and detect cycles using extra space.

Q3. [Definition] What is the first step in removing a loop in a linked list using Floyd's algorithm?
A. Reverse the list
B. Find the loop's start node
C. Set tail node to null
D. Convert list to array
Answer: B
Explanation: After detecting the cycle, we must find the starting node of the loop.

Q4. [Definition] How does merging two sorted lists help in merge sort?
A. It tracks duplicates
B. It reduces recursion
C. It's a key step in the combine phase
D. It minimizes loop detection
Answer: C
Explanation: Merging sorted halves is essential in the combine step of merge sort.

Q5. [Definition] In LRU Cache, why must we move the accessed node to the front?
A. To keep the list sorted
B. To mark it as least used
C. To avoid collisions
D. To mark it as recently used
Answer: D
Explanation: Accessed items must move to the front to indicate recent usage.

Q6. [Definition] If two linked lists intersect, how many nodes will they share?
A. Same length
B. Exactly one
C. From intersection point till end
D. All nodes
Answer: C
Explanation: Once they intersect, they share all remaining nodes.

Q7. [Definition] Why is a dummy node used in LRU Cache implementation?
A. For storing key-value pairs
B. To simplify edge case handling
C. To track capacity
D. To speed up iteration
Answer: B
Explanation: Dummy head/tail nodes simplify adding/removing nodes without special cases.

Q8. [Definition] Which statement is true about the head of a reversed linked list?
A. It stays the same
B. It becomes null
C. It points to the last original node
D. It remains fixed
Answer: C
Explanation: After reversal, the new head is the last node of the original list.

Q9. [Definition] What causes a NullPointerException in linked list traversal?
A. Calling next on a null node
B. Having only one node
C. List being sorted
D. Loop with more than one node
Answer: A
Explanation: Accessing .next on a null node throws NullPointerException.

Q10. [Definition] In a doubly linked list, how many pointers are modified during insertion between two nodes?
A. 1
B. 2
C. 3
D. 4
Answer: D
Explanation: Two for the new node, and one each for prev and next nodes.

Q11. [Moderate] Which of the following is NOT required for reversing a linked list iteratively?
A. A previous pointer
B. A temporary pointer
C. A fast pointer
D. A current pointer
Answer: C
Explanation: Fast pointer is used in cycle detection, not in reversal.

Q12. [Moderate] What will be the output of the following code if the list contains a loop?

```
Node temp = head;

while (temp != null) {

   temp = temp.next;

}

System.out.println("Done");
```

A. Done
B. Loop detected
C. Infinite loop
D. Compilation error
Answer: C
Explanation: Without loop detection, this will go into an infinite loop.

Q13. [Moderate] In recursive reversal, what happens after the base case returns?
A. Function ends
B. Pointers are realigned recursively
C. List is cleared
D. Loop is created
Answer: B
Explanation: As the stack unwinds, the links are adjusted.

Q14. [Moderate] Which function best represents intersection detection with O(1) space?
A. Recursive length check
B. HashMap storage
C. Two-pointer technique with tail alignment
D. Nested iteration
Answer: C
Explanation: Two-pointer with alignment offers O(n) time and O(1) space.

Q15. [Moderate] What is a sign of a well-implemented LRU cache?
A. Least used item remains at head
B. Items are removed from middle
C. Most used item is never deleted
D. Recently accessed items are always at front
Answer: D
Explanation: The front represents the most recently accessed items.

Q16. [Moderate] In merging two sorted lists, when is a recursive call made?
A. When both lists are null
B. When one list ends
C. At each comparison
D. Never

Answer: C
Explanation: Recursive calls occur for each comparison between nodes.

Q17. [Moderate] In Floyd's algorithm, which loop ensures detection?
A. for loop
B. Infinite loop with break
C. while (fast != null && fast.next != null)
D. do-while
Answer: C
Explanation: It ensures the loop ends safely without null pointer exception.

Q18. [Moderate] How to identify loop entry point after detection in Floyd's algorithm?
A. Start one pointer from head, other from meeting point
B. Use HashMap
C. Reverse list
D. Compare values
Answer: A
Explanation: When both pointers meet again, that's the start of the loop.

Q19. [Moderate] What issue occurs in LRU if hashmap is not updated during put()?
A. Memory leak
B. Wrong node accessed
C. Node not found in O(1)
D. Loop is formed
Answer: C
Explanation: HashMap must be updated to ensure constant-time access.

Q20. [Moderate] What will be printed?

```
Node n = new Node(10);

n.next = n;

System.out.println(n.next.data);
```

A. 10
B. null
C. 0
D. Runtime error
Answer: A
Explanation: n.next points to itself, so n.next.data is 10.

Q21. [Definition] In a library, books are placed one after another. You can only access the next one from the current. What structure fits?
A. Array
B. Stack
C. Singly Linked List
D. Graph
Answer: C
Explanation: You can only move forward like in a singly linked list.

Q22. [Definition] In a circular shuttle service, each bus station leads to the next, and the last leads to the first. Which linked list structure fits?
A. Doubly Linked List
B. Circular Linked List
C. Graph
D. Tree
Answer: B
Explanation: A circular linked list connects the last node to the first.

Q23. [Moderate] In a journal, each entry links to the next and the previous. You want to go back and forth. Which structure is ideal?
A. Queue
B. Singly Linked List
C. Doubly Linked List
D. Stack
Answer: C
Explanation: Doubly linked list allows two-way traversal.

Q24. [Moderate] In a detective story, each clue leads to the next. But one clue creates a loop. How can the detective detect infinite tracking?
A. Count number of clues
B. Use Floyd's algorithm
C. Track clue values
D. Ask for help
Answer: B
Explanation: Cycle detection is best done using Floyd's algorithm.

Q25. [Moderate] An LRU-based fridge removes the item not touched recently. What part of the structure will be removed when full?
A. Head
B. Middle
C. Tail
D. Random
Answer: C
Explanation: Tail holds the least recently used item.