1. What will be the output of the following code?

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class Main {
    public static void main(String[] args) {
        Node head = new Node(10);
        head.next = new Node(20);
        head.next.next = new Node(30);
        System.out.println(head.next.data);
    }
}
```

A) 10
B) 20
C) 30
D) Compilation error

Answer: B) 20

Explanation: head.next refers to the second node, which stores 20.

Explanation: You may have to check each node sequentially.

2. How do you remove the last element of a LinkedList in Java?

a) list.deleteLast();
b) list.remove(list.size() - 1);
c) list.removeLast();
d) list.popLast();

Answer: c) list.removeLast();

Explanation: Provided by Java's LinkedList class.

Easy Level (1–10)

3. What is the output of the following code?

```
class Node {
    int data;
    Node next;
    Node(int d) { data = d; next = null; }
}
class LinkedList {
    Node head;
    void printFirstNode() {
        System.out.println(head.data);
```

```
    }
}
public class Main {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.head = new Node(10);
        list.printFirstNode();
    }
}
```

a) 0
b) 10
c) null
d) Compile error
Answer: b) 10
Explanation: The head node is initialized with value 10, which gets printed.

4. What does this code print?

```
class Node {
    int data;
    Node next;
    Node(int data) { this.data = data; }
}
public class Main {
    public static void main(String[] args) {
        Node a = new Node(5);
        Node b = new Node(10);
        a.next = b;
        System.out.println(a.next.data);
    }
}
```

a) 5
b) 10
c) null
d) Compile-time error
Answer: b) 10
Explanation: a.next points to b, whose data is 10.

5. What does the following method do?

```
void printList(Node head) {
    while (head != null) {
        System.out.print(head.data + " ");
        head = head.next;
    }
}
```

a) Deletes the list
b) Prints the list
c) Counts nodes
d) Creates a new list
Answer: b) Prints the list
Explanation: This method prints all node data values.
6. What is the output?

```
Node head = new Node(1);
head.next = new Node(2);
head.next.next = new Node(3);
System.out.println(head.next.next.data);
```

a) 1
b) 2
c) 3
d) null
Answer: c) 3
Explanation: head.next.next points to the third node.
7. Which keyword is used to link one node to another?
a) data
b) next
c) prev
d) link
Answer: b) next
Explanation: next is the reference to the next node.
51. What does this method return?

```
int countNodes(Node head) {
    int count = 0;
    while (head != null) {
        count++;
        head = head.next;
    }
    return count;
}
```

a) Number of nodes
b) Node data
c) 0 always
d) Head value
Answer: a) Number of nodes
Explanation: It increments count while traversing.
8. What is the time complexity of counting nodes in a singly linked list?
a) O(1)
b) O(n)

```
c) O(log n)
d) O(n²)
Answer: b) O(n)
Explanation: Each node is visited once.
9. What is the output?
Node head = null;
System.out.println(head == null);
a) true
b) false
c) null
d) Error
Answer: a) true
Explanation: head is null, so the condition is true.
10. Identify the function's role:
Node insertAtBeginning(Node head, int data) {
    Node newNode = new Node(data);
    newNode.next = head;
    return newNode;
}
a) Deletes head
b) Appends at end
c) Inserts at beginning
d) Does nothing
Answer: c) Inserts at beginning
Explanation: It creates a new node and places it before the head.
11. What happens if we call head.next on a null head?
a) Returns null
b) Runtime error
c) 0
d) Compile error
Answer: b) Runtime error
Explanation: NullPointerException at runtime.
Medium Level (11–20)
12. Which of the following methods reverses the list?
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) Traverses list
b) Deletes last node
c) Reverses the list
d) Creates copy
Answer: c) Reverses the list
Explanation: Standard in-place reversal logic.
13. What is returned?

```
Node head = new Node(1);
head.next = new Node(2);
head = insertAtBeginning(head, 0);
System.out.println(head.data);
```

a) 1
b) 0
c) 2
d) null
Answer: b) 0
Explanation: 0 is inserted at the beginning.
14. What does this function do?

```
Node deleteHead(Node head) {
    return (head == null) ? null : head.next;
}
```

a) Deletes all nodes
b) Deletes head
c) Adds to head
d) No effect
Answer: b) Deletes head
Explanation: Head is updated to next node.
15. Identify the correct way to traverse:
a) Use while (head.next != null)
b) Use while (head != null)
c) Use for (int i = 0; i < head.size(); i++)
d) Use recursion only
Answer: b) while (head != null)
Explanation: Traverses till the end.
16. What is a memory efficient double linked list?
a) Each node has only one pointer to traverse the list back and forth
b) The list has breakpoints for faster traversal
c) An auxiliary singly linked list acts as a helper list to traverse through the doubly linked list
d) A doubly linked list that uses bitwise AND operator for storing

addresses

Answer: a
Explanation: Memory efficient doubly linked list has only one pointer to traverse the list back and forth. The implementation is based on pointer difference. It uses bitwise XOR operator to store the front and rear pointer addresses. Instead of storing actual memory address, every node store the XOR address of previous and next nodes.

17. Which of the following is false about a circular linked list?
a) Every node has a successor
b) Time complexity of inserting a new node at the head of the list is O(1)
c) Time complexity for deleting the last node is O(n)
d) We can traverse the whole circular linked list by starting from any point

Answer: b
Explanation: Time complexity of inserting a new node at the head of the list is O(n) because you have to traverse through the list to find the tail node.

18. Consider a small circular linked list. How to detect the presence of cycles in this list effectively?
a) Keep one node as head and traverse another temp node till the end to check if its 'next points to head
b) Have fast and slow pointers with the fast pointer advancing two nodes at a time and slow pointer advancing by one node at a time
c) Cannot determine, you have to pre-define if the list contains cycles
d) Circular linked list itself represents a cycle. So no new cycles cannot be generated

Answer: b
Explanation: Advance the pointers in such a way that the fast pointer advances two nodes at a time and slow pointer advances one node at a time and check to see if at any given instant of time if the fast pointer points to slow pointer or if the fast pointer's 'next' points to the slow pointer. This is applicable for smaller lists.

19. Which of the following is false about a doubly linked list?
a) We can navigate in both the directions
b) It requires more space than a singly linked list

c) The insertion and deletion of a node take a bit longer
d) Implementing a doubly linked list is easier than singly linked list

Answer: d
Explanation: A doubly linked list has two pointers 'left' and 'right' which enable it to traverse in either direction. Compared to singly liked list which has only a 'next' pointer, doubly linked list requires extra space to store this extra pointer. Every insertion and deletion requires manipulation of two pointers, hence it takes a bit longer time. Implementing doubly linked list involves setting both left and right pointers to correct nodes and takes more time than singly linked list.

20. Which of the following application makes use of a circular linked list?
a) Undo operation in a text editor
b) Recursive function calls
c) Allocating CPU to resources
d) Implement Hash Tables

Answer: c
Explanation: Generally, round robin fashion is employed to allocate CPU time to resources which makes use of the circular linked list data structure. Recursive function calls use stack data structure. Undo Operation in text editor uses doubly linked lists. Hash tables uses singly linked lists.

21. What is the time complexity of searching for an element in a circular linked list?
a) O(n)
b) O(nlogn)
c) O(1)
d) $O(n^2)$

Answer: a
Explanation: In the worst case, you have to traverse through the entire list of n elements.

22. What differentiates a circular linked list from a normal linked list?
a) You cannot have the 'next' pointer point to null in a circular linked list
b) It is faster to traverse the circular linked list
c) In a circular linked list, each node points to the previous node instead of the next node

d) Head node is known in circular linked list

Answer: a
Explanation: In a normal linked list, the 'next' pointer of the last node points to null. However, in a circular linked list, the 'next' pointer of the last node points to the head (first element) of the list. Every node in a circular linked list can be a starting point(head).

23. What is a disadvantage of a circular linked list?
a) Cannot be implemented in Java
b) Cannot traverse easily using a while loop
c) Takes more memory than a singly linked list
d) More complex insertion and deletion operations
Answer: d) More complex insertion and deletion operations.
Explanation: Maintaining links carefully is more involved.

24. What is the time complexity of deleting the last node in a circular linked list?
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)
Answer: b) O(n)
Explanation: You must find the node before the last node.

25. What is a data structure?
a) A way to store and organize data efficiently
b) A type of database
c) A programming language
d) A method for writing algorithms
Answer: a) A way to store and organize data efficiently.
Explanation:- Data Structure - Efficient storage and organization of data using constructs like arrays, lists, stacks, etc. Java provides in-built support for many of these via collections framework.

26. Which of the following is NOT a linear data structure?
a) Stack
b) Queue
c) Tree
d) Array
Answer: c) Tree
Explanation:- Tree is not linear - Trees have hierarchical (not sequential) relationships, unlike arrays or linked lists

27. Which of the following operations has O(1) time complexity in an array?

a) Insertion at the end
b) Searching an element
c) Insertion at the beginning
d) Deletion at the middle
Answer: a) Insertion at the end
Explanation:- Insertion at End (O(1)) - For fixed-size arrays, Java allows constant-time insertion at the last index, unless resizing is required (in case of ArrayList).

28 What does the following recursive function do?

```
void mystery(Node head) {
    if (head == null) return;
    mystery(head.next);
    System.out.print(head.data + " ");
}
```

a) Prints list in forward order
b) Reverses the list
c) Prints list in reverse order
d) Deletes every alternate node
Answer: c) Prints list in reverse order
Explanation: Due to the recursive call before printing, elements are printed in reverse order.
29. What is the space complexity of a singly linked list?
a) O(1)
b) O(n)
c) O(n log n)
d) O(n²)
Answer: b) O(n)
Explanation: One node stores one value.
30. Identify the bug:

```
Node head = null;
head.next = new Node(2);
```

a) Valid
b) NullPointerException
c) Prints 2
d) None
Answer: b) NullPointerException
Explanation: Cannot access next on null.
31. Choose correct insertion at end logic:

```
Node insertEnd(Node head, int data) {
    Node temp = new Node(data);
    if (head == null) return temp;
    Node curr = head;
```

```
    while (curr.next != null) curr = curr.next;
    curr.next = temp;
    return head;
}
```

a) Appends at head
b) Reverses
c) Appends at tail
d) Deletes node
Answer: c) Appends at tail
Explanation: It traverses till end and adds new node.
32. What is the result?

```
Node a = new Node(3);
Node b = new Node(4);
a.next = b;
b = null;
System.out.println(a.next.data);
```

a) 3
b) 4
c) null
d) Error
Answer: b) 4
Explanation: a.next still points to b, even if b = null.
33. What does this code do?

```
Node mid(Node head) {
    Node slow = head, fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}
```

a) Finds end
b) Deletes node
c) Finds middle
d) Error
Answer: c) Finds middle
Explanation: Slow-fast pointer technique.
34. What is the Output of given code?

```
Node head = new Node(10);
head.next = new Node(20);
head.next.next = null;
System.out.println(head.next.next);
```

a) 20
b) null
c) 10
d) Error
Answer: b) null
Explanation: .next of the last node is null.
35. What is the effect of this recursive function?

```
void printReverse(Node head) {
    if (head == null) return;
    printReverse(head.next);
    System.out.print(head.data + " ");
}
```

a) Reverse list
b) Print in reverse
c) Error
d) Null
Answer: b) Print in reverse
Explanation: Recursive post-order print.