Date / /
Page No.

Practice Test Solution.

Q1. Define a LinkedList. How does it differ from an array?

Ans. **Definition** :- A linkedlist is a linear data structure in which elements (called nodes) are connected using pointers. Each node contains
- Data
- A pointer (link) to the next node in list.

Difference from Array :

Feature	Array	LinkedList
Storage	Contiguous memory	Non-contiguous memory
Size	Fixed	Dynamic (grow/shrink)
Insertion/ Deletion	Costly (shifting elements)	Easy (adjusting pointers).
Access	Direct access by Index	Sequential (must traverse).

Date / /
Page No.

Q2. Difference between Circular & Doubly LinkedList :-

	Feature	Circular LL	Doubly LL
(1)	Links	Last node points to the first node	Each node has pointers to both previous & next
(2)	Traversal	Can traverse infinitely in one direction.	Can traverse in both directions.
(3)	Last node	Connects back to head	Points to NULL (if not circular doubly).

Q3. Advantage & Disadvantage of using LinkedList over an array :-

Advantage : Dynamic size – memory allocated as needed, no need to define size.

disadvantage : No random access – elements must be accessed sequentially making searches slower.

Q4.

```
class Node {
public:
    int data;          // stores data
    Node * next;       // pointer to next node

Node(int value) {      // constructor to initialize data &
    data = value;          next pointer.
    next = nullptr; }
};
```

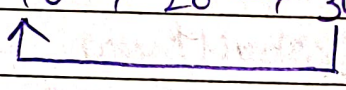
Date / /
Page No.

Q5. List types & linkedlist with examples :-

1. Singly linkedlist : Each node points to next
eg: 10 → 20 → 30 → NULL

2. Doubly linkedlist: Each node points to both previous & next
eg: NULL ← 10 ↔ 20 ↔ 30 → NULL

3. Circular linkedlist: Last node connects back to the first.
eg:



Q6. Time complexity of insertion at beginning and end in a linkedlist.

operation (Insertion)	—	Time complexity
at Beginning.	—	O(1)
at end	—	O(n) since traversal to the last node is needed.

Date / /
Page No.

Q7. Write algorithm to insert an element at the end of a LinkedList.

Algorithm :

1. Create new node
2. If list is empty (head == NULL), set head = newNode
3. Otherwise, traverse to last node
4. Set last -> next = newNode
5. Mark newNode -> next = NULL.

Code :

```
void insertAtEnd(Node* &head, int val) {
    Node * newNode = new Node(value);
    if (head == NULL) {
        head = newNode;
        return;
    }
    Node * temp = head
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = newNode;
}
```

Dry run : insert 30 at end of list 10 -> 20 -> NULL

① newNode = (30, NULL)
② Traverse: temp = 10 -> 20 (reaches end)
③ temp -> next = newNode.

Result : 10 -> 20 -> 30 -> NULL

→ Time complexity : O(n)
→ Space complexity : O(1).

Date / /
Page No.

08. logic to count total nodes in a LinkedList.

Algorithm :

1. initialize count = 0
2. start from head
3. Traverse each node & increment count.
4. Stop when current node becomes NULL.
5. Return count.

Code :

```
int countNodes(Node * head) {
    int count = 0;
    Node* temp = head;
    while (temp != NULL) {
        count++;
        temp = temp -> next;
    } return count;
}
```

DRY RUN : List 10 → 20 → 30 → NULL

- count 0
- visit 10 → count=1
- visit-20 → count=2
- visit-30 → count=3

Result : 3 Nodes.

→ Time complexity = O(n)
→ Space complexity = O(1)

Date / /
Page No.

Q9. Reverse a linkedlist : (iterative method).

Algorithm :

① Initialize 3 pointers :
prev = NULL, curr = head, next = NULL

② while (curr != NULL)
- store next node next = curr → next
- reverse the link curr → next = prev.
- move prev = curr, curr = next

③ After loop, set head = prev.

Code :

```
void reverseList(Node*& head) {
    Node *prev = NULL;
    Node *curr = head;
    Node *next = NULL;

    while(curr != NULL) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}
```

T(c) = O(n)
S(c) = O(1)

DRY RUN : 10 → 20 → 30 → NULL

	prev	curr	next	Result
①	NULL	10	20	10 → NULL
②	10	20	30	20 → 10 → NULL
③	20	30	NULL	30 → 20 → 10 → NULL

Result : 30 → 20 → 10 → NULL.

Date / /
Page No.

Q10. find middle element of a linkedlist using two pointer.

Algorithm :

① Initialize two pointers slow = head, fast = head.

② Move fast two steps & slow one step in each iteration.

③ when fast reaches end (NULL or fast->next == NULL), -> slow is at the middle.

code :

```
int findMiddle (Node* head) {
    Node * slow = head;
    Node * fast = head;

    while (fast != NULL && fast -> next != NULL) {
        slow = slow -> next;
        fast = fast -> next -> next;
    }
    return slow -> data;
}
```

DRY RUN :-

10 -> 20 -> 30 -> 40 -> 50 -> NULL

Iteration	slow	fast
1	20	30
2	30	50.

Result : middle = 30

Time complexity : O(n)
Space complexity : O(1)

Q11.

(5) → (10) → (15) → (20) → NULL

deleteAtEnd(), show resulting list & explain which node (5) get deleted.

Algorithm :

1. If list ∞ empty → return.
2. If only one node → delete it, set head == NULL.
3. Else, traverse until temp → next → next == NULL
4. delete temp → next (last node).
5. Set temp → next = NULL.

Code :-

```
Void deleteAtEnd (Node*& head) {
    if (head == NULL) return;
    if (head → next = NULL) {
        delete head;
        head = NULL;
        return;
    }
    Node * temp = head;
    while (temp → next → next != NULL) {
        temp = temp → next;
    }
    delete temp → next;
    temp → next = NULL;
}
```

TC : O(n)
SC : O(1)

DRY RUN. (5) → (10) → (15) → (20) → NULL

Traverse : temp stops at 15.
Delete node (20)
Resulting list : 5 → 10 → 15 → NULL
delete node : value = 20.

Date / /
Page No.

Q12 Merge two sorted list into single sorted list :-

Algorithm :-

1. Initialize pointers p1 (head1) and p2 (head2).
2. Compare values
   - Smaller one gets added to new list.
   - Move that list's pointer forward.
3. Continue until one list ends.
4. Append remaining nodes from the other list.
5. Return merged head.

Code :

```
Node* mergeSortedLists(Node* head1, Node* head2){
    if(!head1) return head2;
    if(!head2) return head1;

    Node* result = NULL;
    if(head1->data < head2->data){
        result = head1;
        result->next = mergeSortedLists(head1->next,
                                        head);
    }else{result = head2;
        result->next = mergeSortedLists(head1, head2->
                                        next);
    } return result;
}
```

Time complexity : O(n+m)
Space complexity : O(1)