Experiment-3

Student Name: Abhinav Paswan **UID:** 22BET10332
Branch: BE-IT **Section/Group:** 22BET-IOT-701(A)
Semester: 6th **Date of Performance:**05-02-2025
Subject Name: AP LAB-II **Subject Code:** 22ITP-351

Problem 1:-

https://leetcode.com/problems/remove-duplicates-from-sorted-list.

Code:

```
class Solution {
public:
   ListNode* deleteDuplicates(ListNode* head) {
      ListNode* current = head;

      while (current && current->next) {
         if (current->val == current->next->val) {
            current->next = current->next->next;
         } else {
            current = current->next;
         }
      }

      return head;
   }
};
```

Problem 2:-

https://leetcode.com/problems/reverse-linked-list/

Code:

```
class Solution {

public:

   ListNode* reverseList(ListNode* head) {

      ListNode* prev = nullptr;

      ListNode* current = head;



      while (current) {

         ListNode* next = current->next;

         current->next = prev;

         prev = current;

         current = next;

      }



      return prev;

   }

};
```

Problem 3:-

https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list

Code:

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr;

        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;

        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }

        prev->next = slow->next;
        delete slow;

        return head;
    }
};
```

Problem 4:-

https://leetcode.com/problems/merge-k-sorted-lists/description/

Code:

```
#include <queue>

class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val; // Min-heap
        }
    };

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;

        for (auto list : lists) {
            if (list) minHeap.push(list);
        }

        ListNode dummy(0);
        ListNode* tail = &dummy;

        while (!minHeap.empty()) {
            ListNode* node = minHeap.top();
            minHeap.pop();
            tail->next = node;
            tail = tail->next;

            if (node->next) minHeap.push(node->next);
        }

        return dummy.next;
    }
};
```

Problem 5:-

https://leetcode.com/problems/sort-list/description/

Code:-

```
class Solution {
public:
    ListNode* merge(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = merge(l1->next, l2);
            return l1;
        } else {
            l2->next = merge(l1, l2->next);
            return l2;
        }
    }

    ListNode* findMiddle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;

        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next;
        }

        if (prev) prev->next = nullptr;
        return slow;
    }

    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;

        ListNode* mid = findMiddle(head);
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);
```

```
        return merge(left, right);
    }
};
```