**Pseudo Code [Handwritten]**

**Code: [Paste your code here]**

#include<iostream>

using namespace **std**;

class **Node**{

    public:

    int data;

**Node**\* next;

**Node**(int val)

    {

        data = val;

        next = nullptr;

    }

};

class **LinkedList**{

    private:

**Node**\* head;

    public:

**LinkedList**(){

        head = nullptr;

    }

    void **insert\_at\_beginning**(int data)

    {

**Node**\* temp = new **Node**(data);

        temp->next = head;

        head = temp;

    }

    void **insert\_at\_end**(int data)

    {

**Node**\* new\_node = new **Node**(data);

        if (head == nullptr){

            head = new\_node;

            return;

        }

**Node**\* temp = head;

        while (temp->next != nullptr){

            temp = temp->next;

        }

        temp->next = new\_node;

    }

    void **display**()

    {

        if (head == nullptr){

            cout **<<** "Linked List does not exist" **<<** **endl**;

            return;

        }

**Node**\* temp = head;

        while (temp != nullptr){

            cout **<<** temp->data **<<** " ";

            temp = temp->next;

        }

    }

**Node**\* **getHead**() {

        return head;

    }

**Node**\* **reverseList**(**Node**\* head) {

**Node**\* prev = nullptr;

**Node**\* current = head;

**Node**\* next = nullptr;

        while (current != nullptr) {

            next = current->next;

            current->next = prev;

            prev = current;

            current = next;

        }

        return prev;

    }

    void **splitList**(**Node**\* source, **Node**\*\* front, **Node**\*\* back) {

**Node**\* slow = source;

**Node**\* fast = source->next;

        while (fast != nullptr) {

            fast = fast->next;

            if (fast != nullptr) {

                slow = slow->next;

                fast = fast->next;

            }

        }

        \*front = source;

        \*back = slow->next;

        slow->next = nullptr;

    }

*// Reorder the list in wave pattern*

    void **NewLL**() {

        if (head == nullptr || head->next == nullptr)

            return;

**Node**\* firstHalf = nullptr;

**Node**\* secondHalf = nullptr;

**splitList**(head, &firstHalf, &secondHalf);

*// Reverse the second half*

        secondHalf = **reverseList**(secondHalf);

*// Merge the two halves in alternating order*

**Node**\* temp = new **Node**(0);

**Node**\* current = temp;

        bool alternate = true;

*// alternate between first and second halves*

        while (firstHalf != nullptr && secondHalf != nullptr) {

            if (alternate) {

                current->next = firstHalf;

                firstHalf = firstHalf->next;

            } else {

                current->next = secondHalf;

                secondHalf = secondHalf->next;

            }

            alternate = !alternate;

            current = current->next;

        }

*// If there are remaining elements in the first or second half*

        if (firstHalf != nullptr) {

            current->next = firstHalf;

        } else if (secondHalf != nullptr) {

            current->next = secondHalf;

        }

        head = temp->next;

        delete temp;

    }

    int **Kth\_smallest\_ele**(int k){

        if(head == nullptr){

            return -1;

        }

        int count = 1;

**Node**\* temp = head;

        while (count <= k && temp != nullptr)

        {

            if(count == k){

                return temp->data;

            }

            count++;

            temp = temp->next;

        }

        return -1; *// If k is larger than the number of nodes*

    }

    void **Remove\_Duplicate**(){

        if(head == nullptr){

            cout **<<** "Linked List does not exist" **<<** **endl**;

            return;

        }

**Node**\* temp = head;

        while(temp != nullptr)

        {

            int d = temp->data;

**Node**\* temp2 = temp;

            while (temp2->next != nullptr)

            {

                if(temp2->next->data == d){

**Node**\* toDelete = temp2->next;

                    temp2->next = temp2->next->next;

                    delete toDelete; *// Free the memory of the removed node*

                } else {

                    temp2 = temp2->next;

                }

            }

            temp = temp->next;

        }

    }

};

int **main**() {

**LinkedList** list;

    list.**insert\_at\_end**(10);

    list.**insert\_at\_end**(20);

    list.**insert\_at\_end**(30);

    list.**insert\_at\_end**(30);

    list.**insert\_at\_end**(50);

    list.**insert\_at\_end**(60);

    list.**insert\_at\_end**(70);

    cout **<<** "Original Sorted List: ";

    list.**display**();

    cout **<<** **endl**;

    list.**NewLL**();

    cout **<<** "New Linked List in Wave Pattern: ";

    list.**display**();

    cout **<<** **endl**;

    int k = 2;

    int result = list.**Kth\_smallest\_ele**(k);

    if (result != -1) {

        cout **<<** "The " **<<** k **<<** "-th smallest element is: " **<<** result **<<** **endl**;

    } else {

        cout **<<** "The " **<<** k **<<** "-th element does not exist in the list." **<<** **endl**;

    }

    cout **<<** "List after removing duplicates: ";

    list.**Remove\_Duplicate**();

    list.**display**();

    cout **<<** **endl**;

    return 0;

}

**Output: [Paste your output here]**

**POST-LAB Exercise [Handwritten-Write the code]**