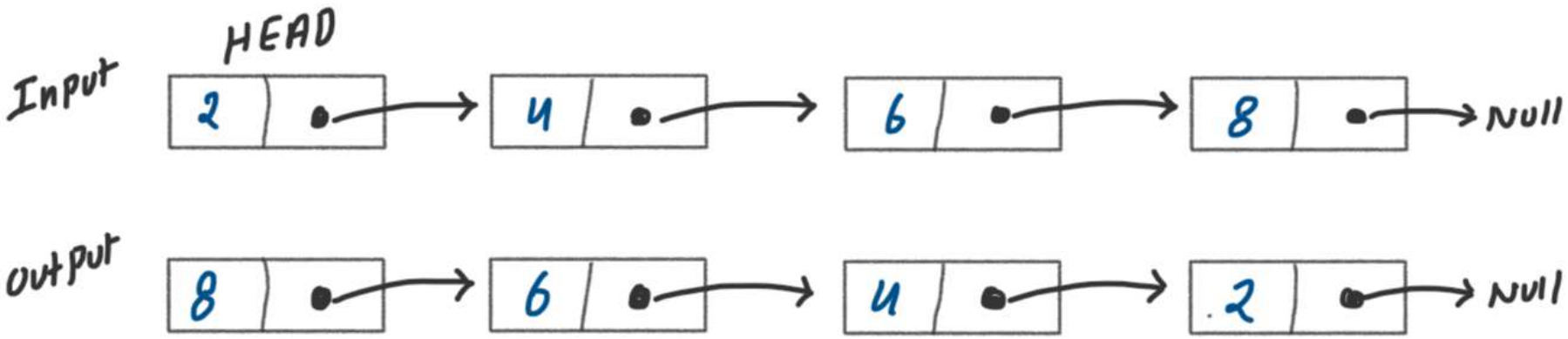
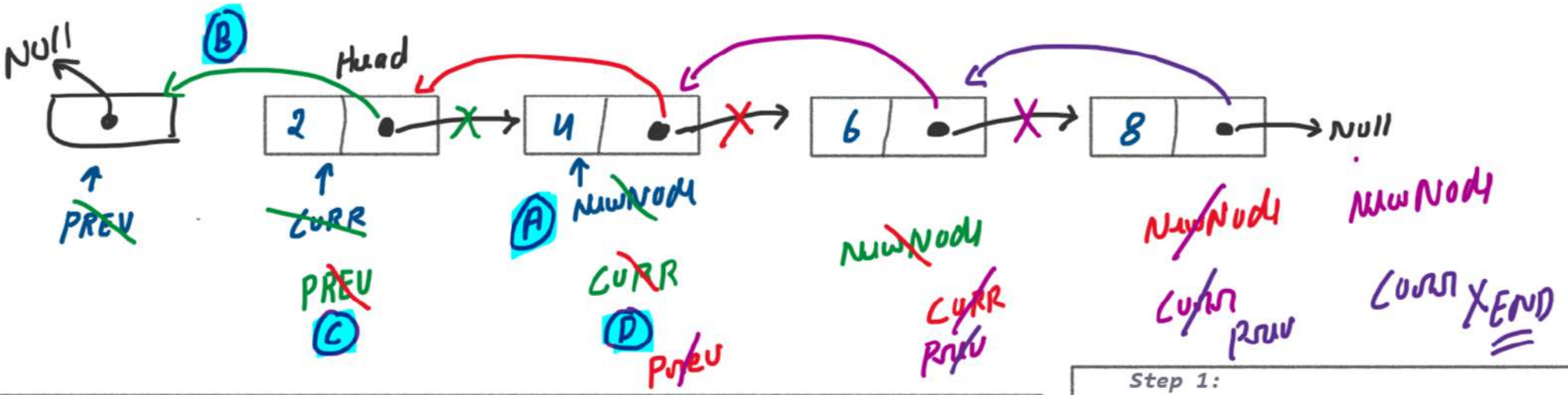
05/11/2023

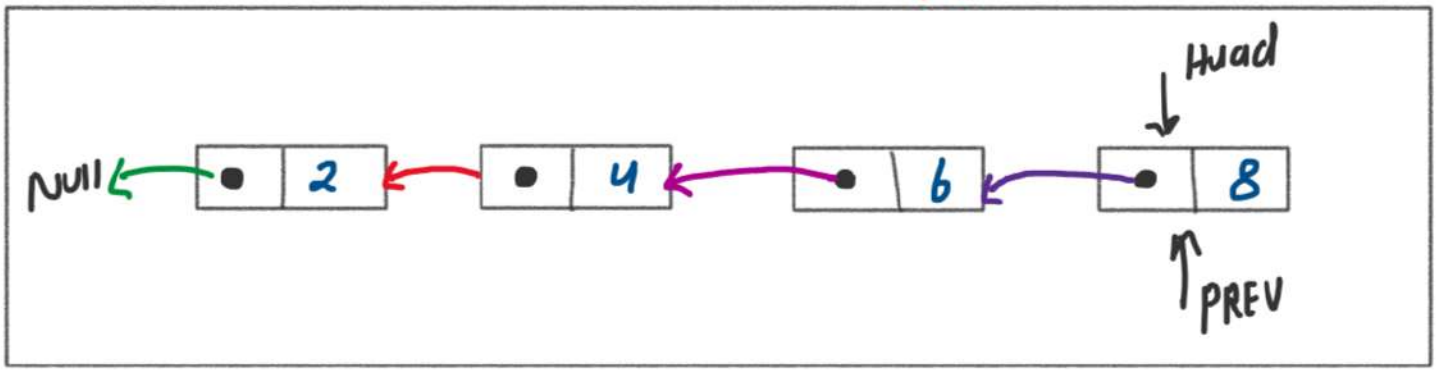
LINKED LIST CLASS - 3

Program 1: Reverse Linked List (Leetcode-206)



Approach 1: Iterative approach





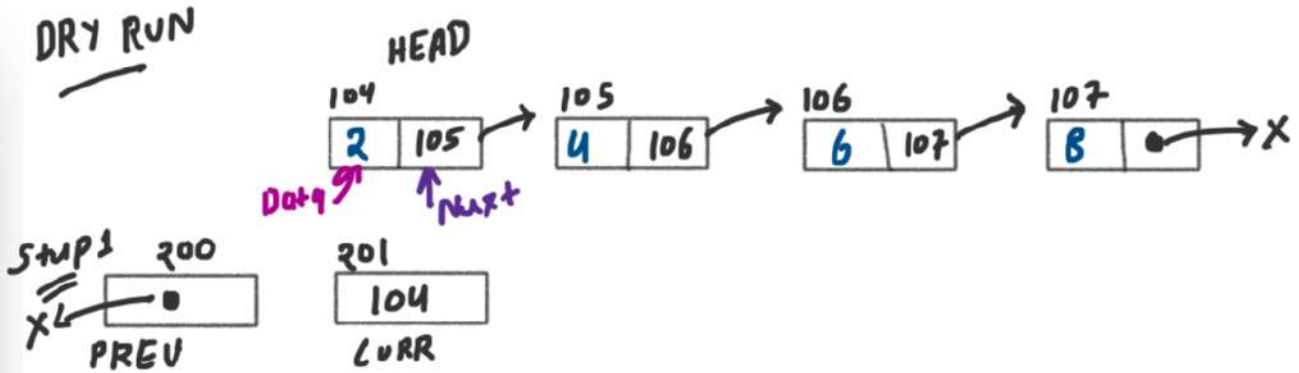

```
Step 1:
Node* PREV = NULL
Node* CURR = Head

Step 2: While (CURR != NULL)
A Node* nextNode = CURR->NEXT
B CURR->NEXT = PREV
C PREV = CURR
D CURR = nextNode

Step 3: Head = PREV or return PREV
```

```cpp
// 📂 Program 1: Reverse Linked List (Leetcode-206)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        // Step 1
        ListNode* prev = NULL;
        ListNode* curr = head;

        // Step 2
        while(curr != NULL){
            // Now no track lost due to curr->next = prev
            ListNode* nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }

        // Step 3
        // head = prev;
        // return head;

        return prev;
    }
};
```

T.C. = O(N)

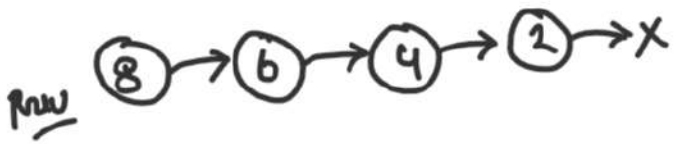
S.C. = O(1)

DRY RUN

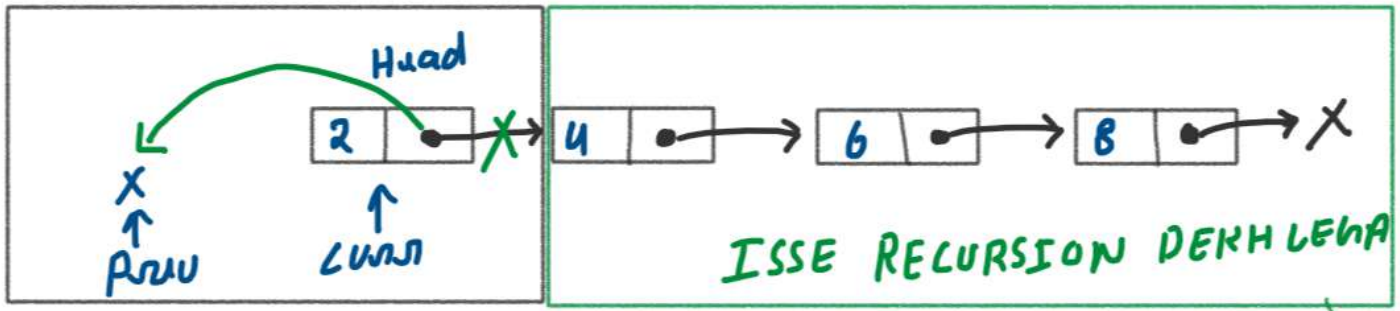


STEP 2

Prev	Curr	while	NextNode	Curr→Next	Prev	Curr
Null	104	✓	105 (300)	Null	104	105
104	105	✓	106 (300)	104	105	106
105	106	✓	107 (300)	105	106	107
106	107	✓	Null (300)	106	107	Null
107	Null	X END				



Approach 2: Recursive approach



EK STEP HUM CHALUNGA

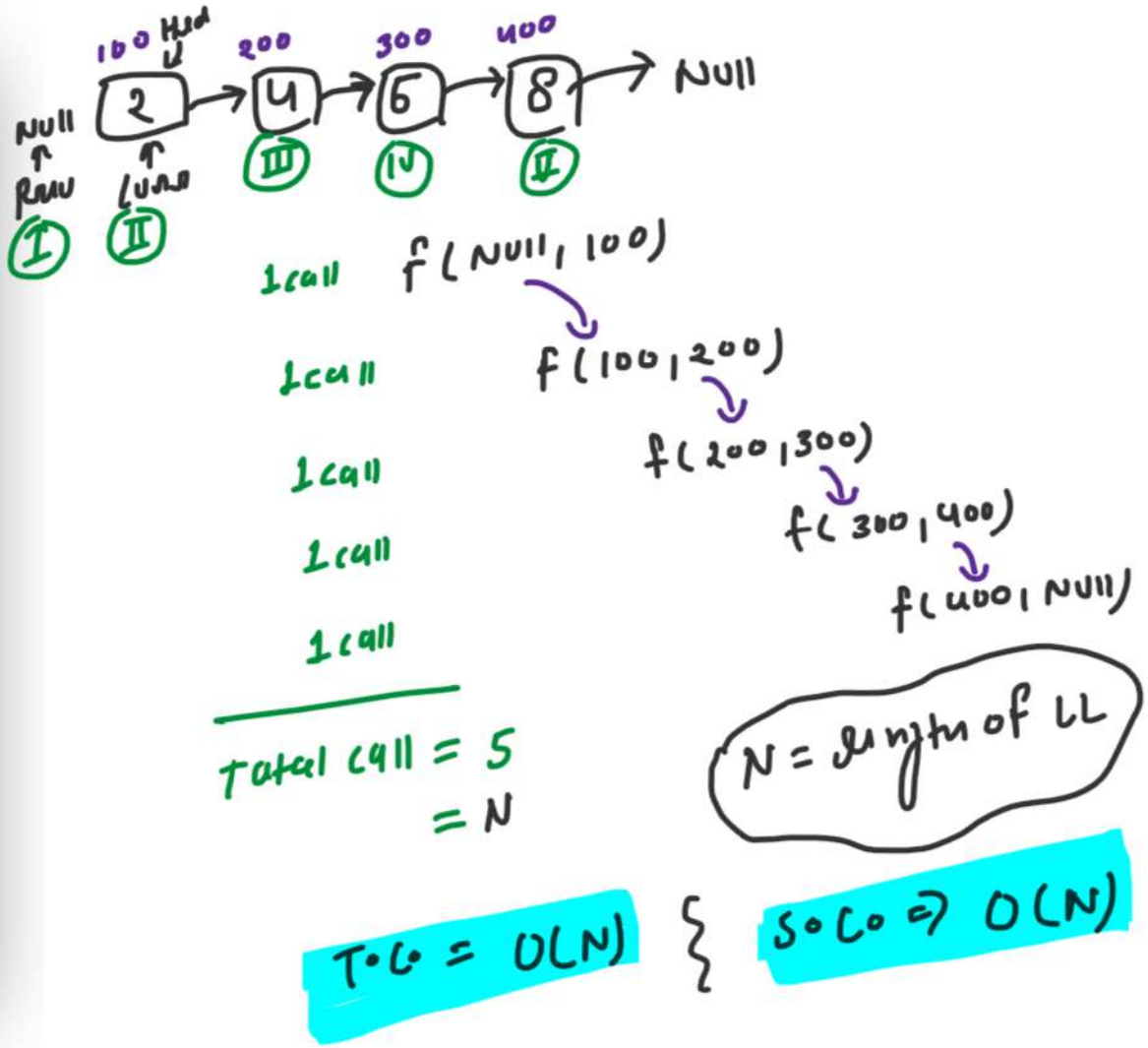
```
Node* prev = NULL;
Node* curr = Head;
Node* nextNode = curr->next;
curr->next = prev;
prev = curr;
curr = nextNode;
```

```
f( prev, curr)
```

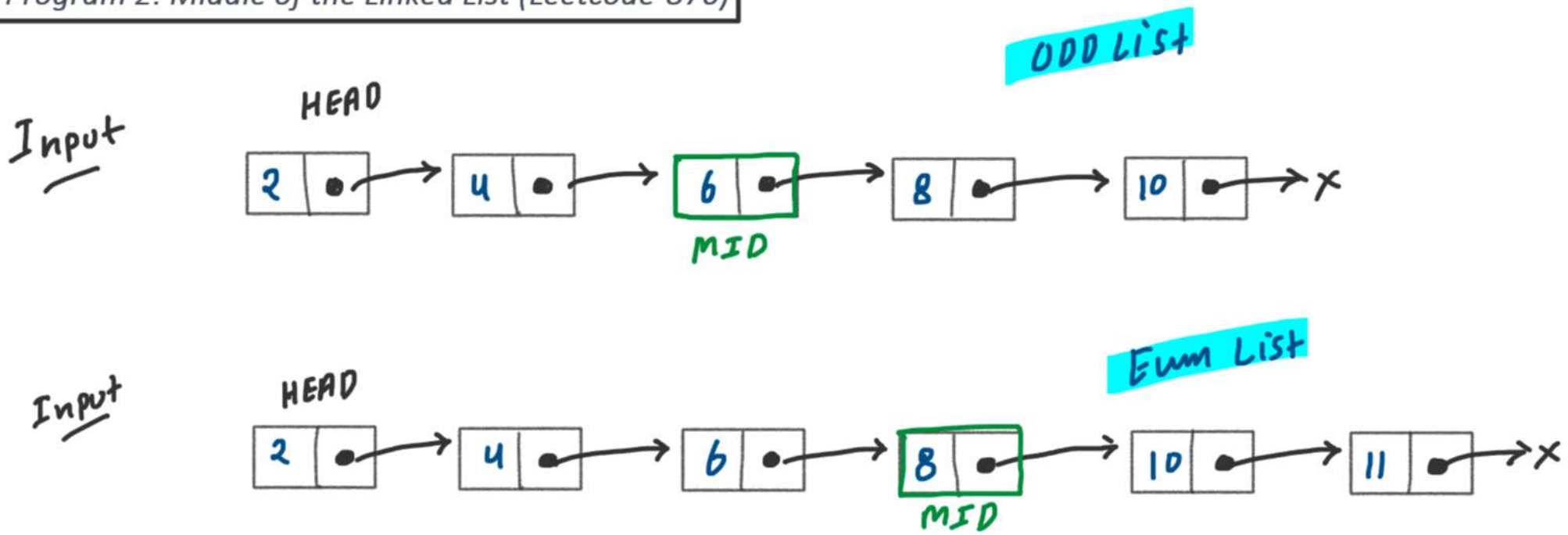
RUKNA KAB HAI

```
if( curr == NULL)
    return prev
```

```cpp
// 📂 Program 1: Reverse Linked List (Leetcode-206)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* reverseUsingRecursion(ListNode* prev, ListNode* curr){
        // Base case
        if(curr == NULL){
            return prev;
        }

        // Ek case hum solve krenge
        ListNode* nextNode = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextNode;

        // Baki recursion sambhal lega
        ListNode* recursionKaAns = reverseUsingRecursion(prev,curr);
        return recursionKaAns;
    }

    ListNode* reverseList(ListNode* head) {
        ListNode* prev = NULL;
        ListNode* curr = head;

        return reverseUsingRecursion(prev, curr);
    }
};
```

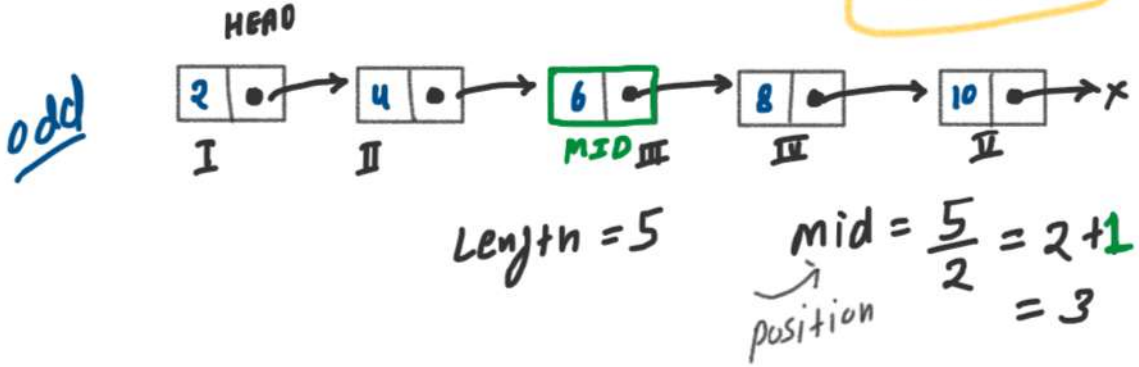


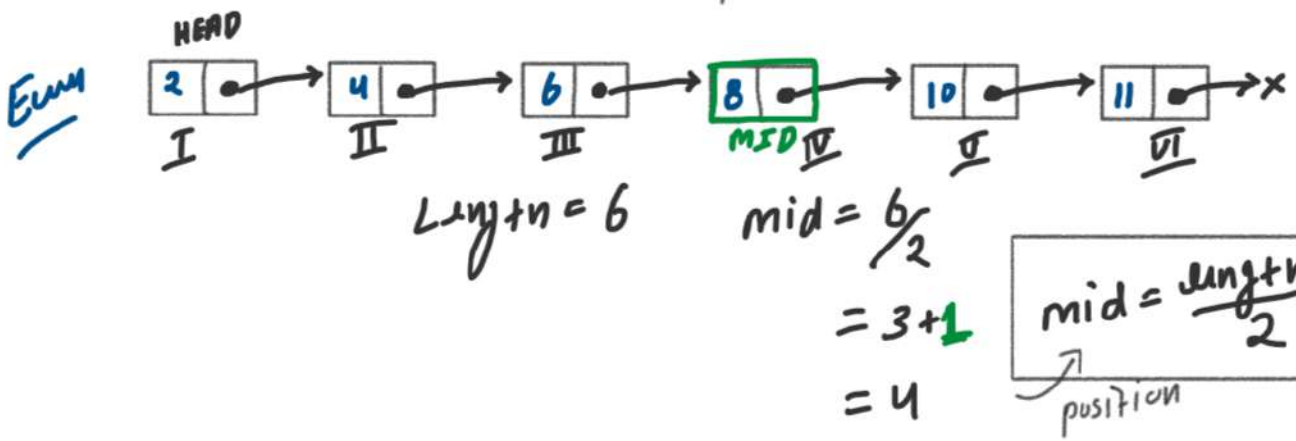
Program 2: Middle of the Linked List (Leetcode-876)



Approach 1:
Step 1: Get Length of LL
Step 2: Get Mid of LL

T.C. = O(N) + O(N)
= O(N)





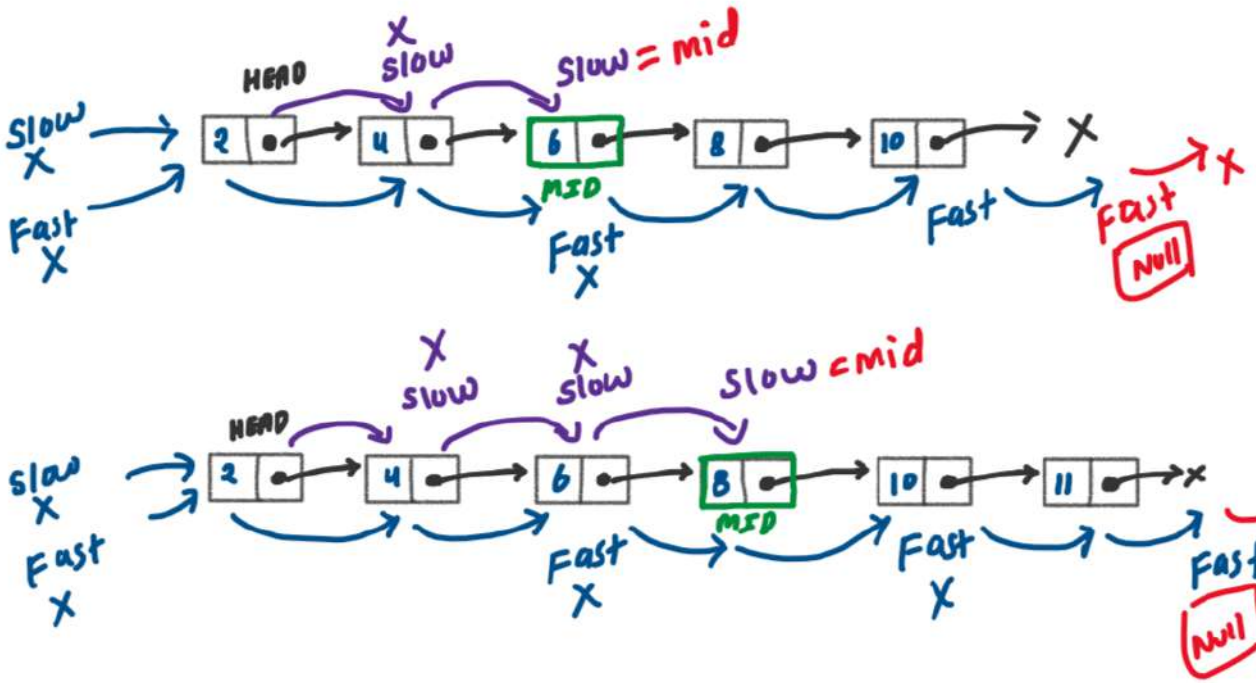

```cpp
// 📁 Program 2: Middle of the Linked List (Leetcode-876)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    int getLength(ListNode* head){
        ListNode* temp = head;
        int count = 0;

        while(temp != NULL){
            count++;
            temp = temp->next;
        }
        return count;
    }
    ListNode* middleNode(ListNode* head) {
        // Step 1: Get length
        int length = getLength(head);

        // Step 2: Get mid node
        int position = length/2 + 1;
        int currPos = 1;
        ListNode* temp = head;

        while(currPos != position){
            currPos++;
            temp = temp->next;
        }
        return temp;
    }
};
```

T.C. = O(N)

T.C. = O(N)

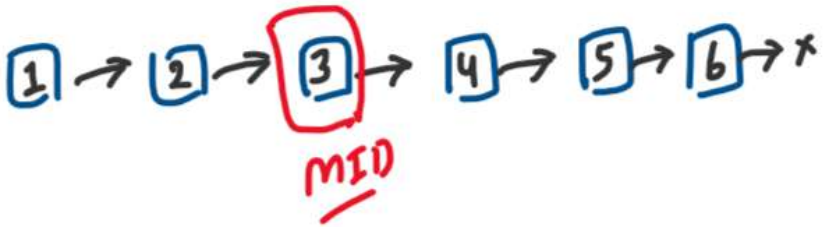
Approach 2: Slow and fast pointer "Hare & Tortoise" algorithm

Note: Slow ek step tabhi chalega jab Fast two step chal skta ho




```cpp
// 📁 Program 2: Middle of the Linked List (Leetcode-876)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

        while(fast != NULL){
            fast = fast->next;
            if(fast != NULL){
                fast = fast->next;
                // Ab fast two step chal chuka hai
                // Ab hum slow ko one step chla skte hai
                slow = slow->next;
            }
        }
        return slow;
    }
};
```

T.C. = O(N)




```cpp
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

        while(fast->next != NULL){
            fast = fast->next;
            if(fast->next != NULL){
                fast = fast->next;
                // Ab fast two step chal chuka hai
                // Ab hum slow ko one step chla skte hai
                slow = slow->next;
            }
        }
        return slow;
    }
};
```



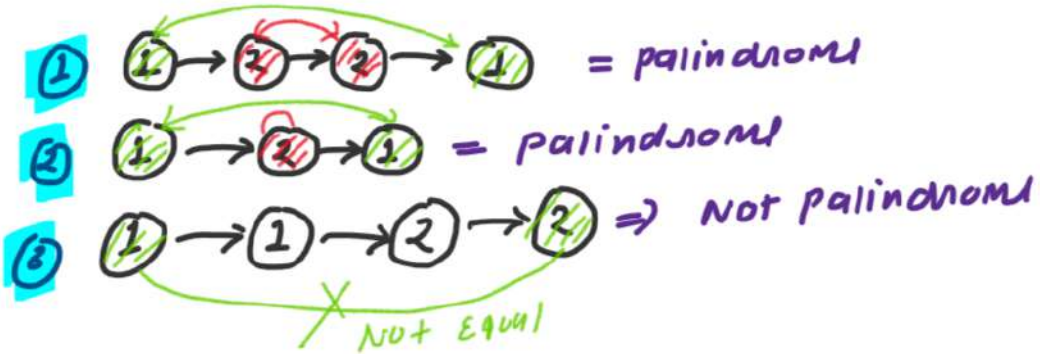
```cpp
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

        while(fast != NULL){
            fast = fast->next;
            if(fast != NULL){
                fast = fast->next;
                // Ab fast two step chal chuka hai
                // Ab hum slow ko one step chla skte hai
                slow = slow->next;
            }
        }
        return slow;
    }
};
```

Program 3: Middle of the Linked List (Leetcode-876)

Approach 1: Optimal

NOTE: Problem in this approach ki hum original linked list ko change kar rhe hai jo ek bad practice hoti hai

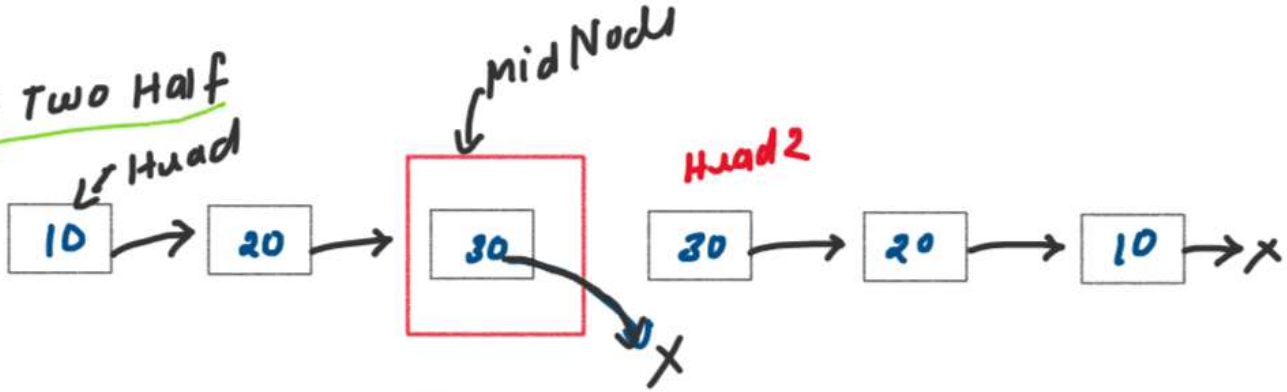
Step 1: Break linked list into 2 half
Step 2: Reverse second half of linked list
Step 3: Compare first half and reversed second half



Example 1

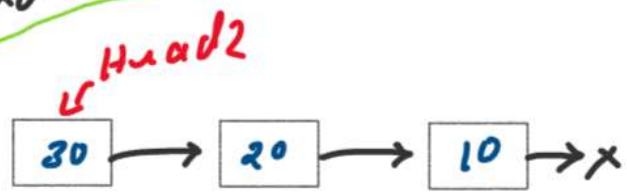
Example 1

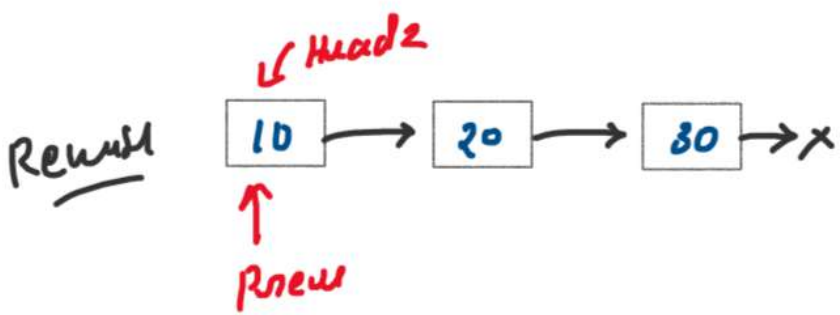
(1) Break into Two Half



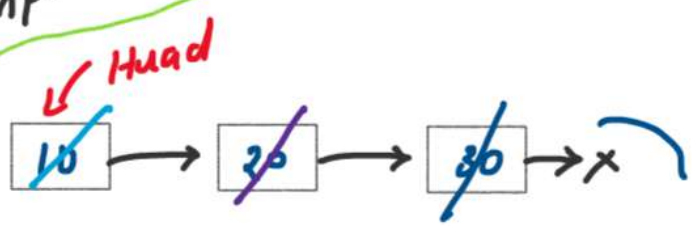
- First ⇒ Find midNode
- Second ⇒ Set Head 2 = midNode → Next
- third ⇒ midNode → Next = NULL

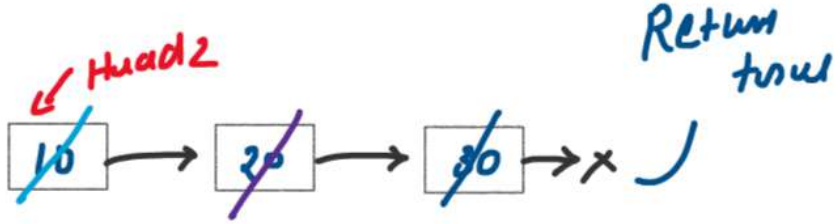
② Reverse second Half





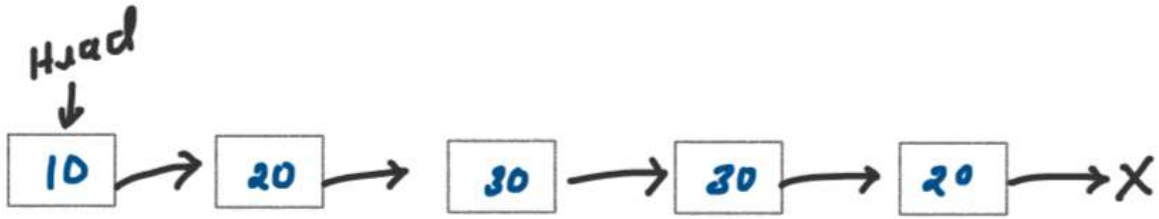
③ Compare both



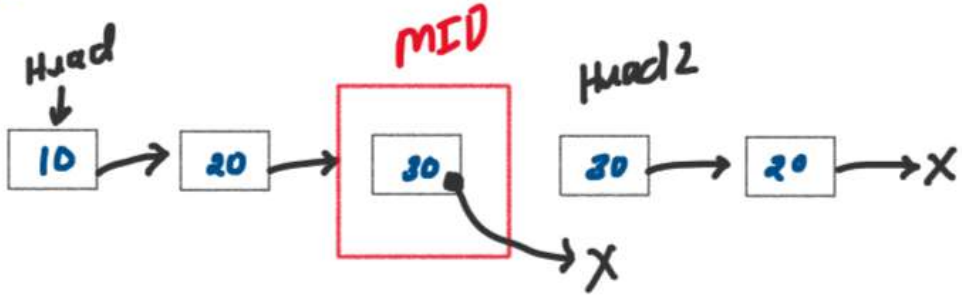


Ex=>2

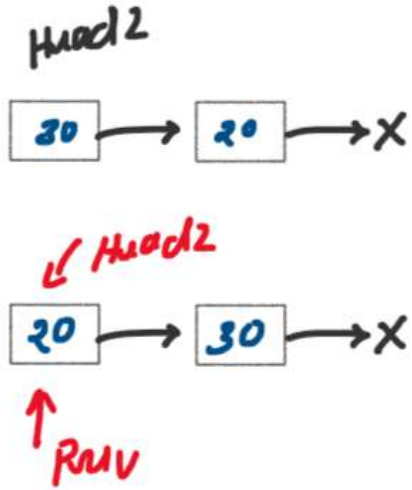
ODD case



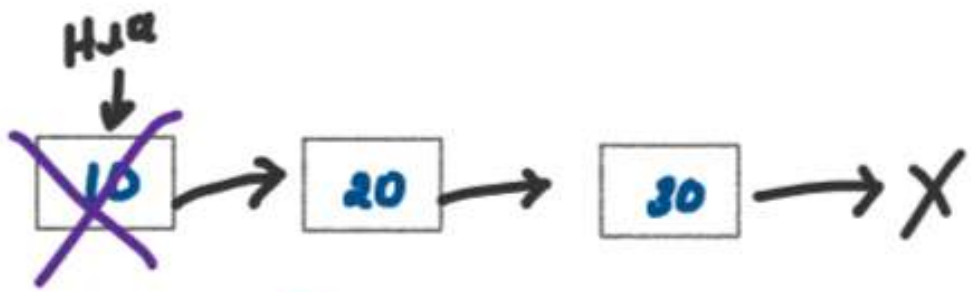
① Break into two Half

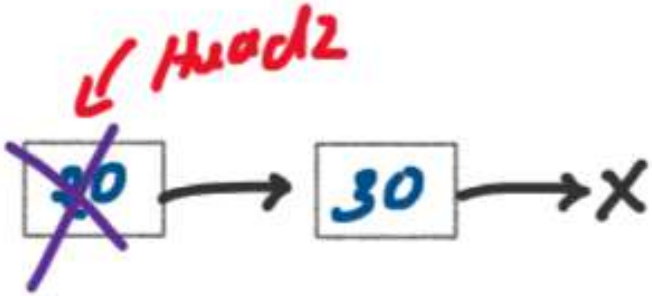


② Rev second Half



(3) Compare both





Return false

```cpp
// 📂 Program 3: Palindrome Linked List (Leetcode-234)
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* middleNode(ListNode* head){...}

    ListNode* reverseUsingRecursion(ListNode* prev, ListNode* curr){...}

    bool compareList(ListNode* head, ListNode* head2){...}

    bool isPalindrome(ListNode* head) {
        // Step 1: Break linked list into 2 halfs
        ListNode* midNode = middleNode(head);
        ListNode* head2 = midNode->next;
        midNode->next = NULL;

        // Step 2: Reverse second half of linked list
        ListNode* prev = NULL;
        ListNode* curr = head2;
        head2 = reverseUsingRecursion(prev, curr);

        // Step 3: Compare first half with reversed second half
        bool ans = compareList(head, head2);
        return ans;
    }
};
```

1

2

3

T.C. ⇒ O(N) + O(N) + O(N) = O(N)

```cpp
ListNode* middleNode(ListNode* head){
    ListNode* slow = head;
    ListNode* fast = head;

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

1 → T.C. ⇒ O(N)

```cpp
bool compareList(ListNode* head, ListNode* head2){
    while(head2 != NULL){
        if(head->val != head2->val){
            return false;
        }
        else{
            head = head->next;
            head2 = head2->next;
        }
    }
    return true;
}
```

3

```cpp
ListNode* reverseUsingRecursion(ListNode* prev, ListNode* curr){
    // Base case
    if(curr == NULL){
        return prev;
    }

    // Ek case hum solve kar lete hai
    ListNode* nextNode = curr->next;
    curr->next = prev;
    prev = curr;
    curr = nextNode;

    // Ab bakl ka recursion solve kr lega
    return reverseUsingRecursion(prev, curr);
}
```

2

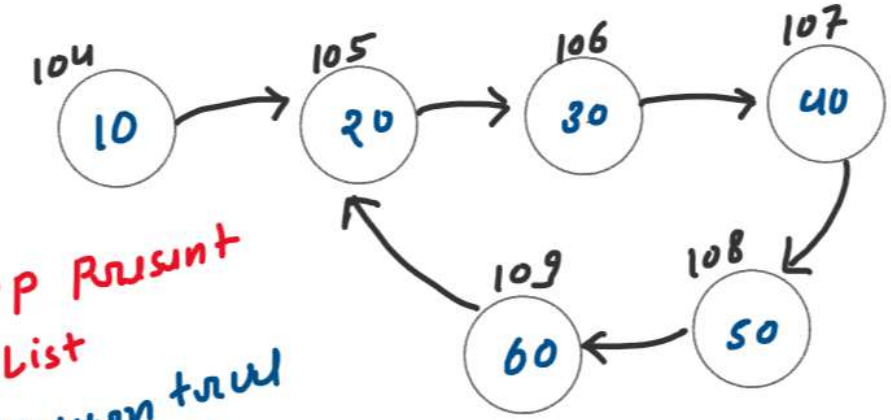
↳ T.C. ⇒ O(N)

T.C ⇒ O(N)

Program 4: Linked List Cycle (Leetcode-141)

COMPARE WITH ADDRESS OF NODE

EX-1



Loop Present in List

↳ Return true

map<Node*, bool> table;

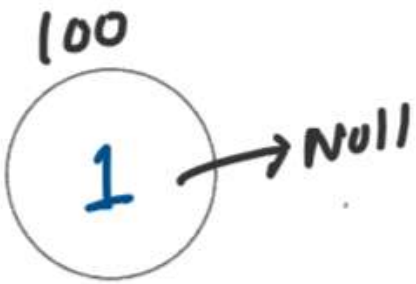
table

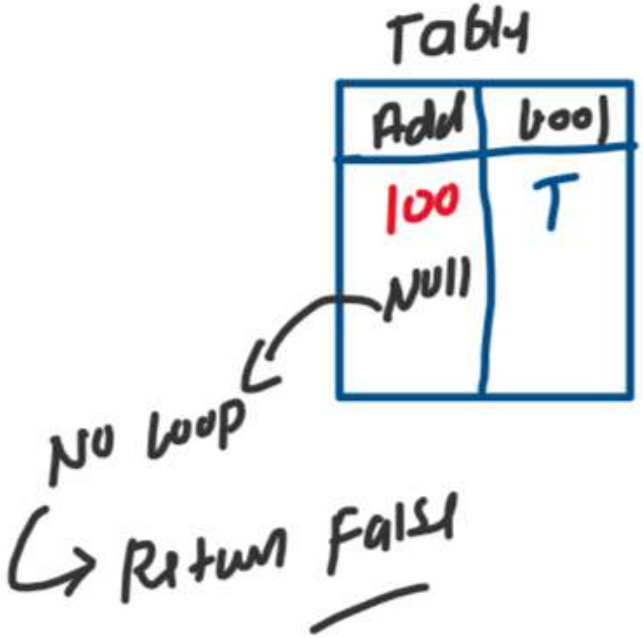
Add	bool
104	T
105	T
106	T
107	T
108	T
109	T

Repeat 105 Add

↳ Return true

Ex-2





```cpp
// 📂 Program 4: Linked List Cycle (Leetcode-141)

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        // Create map table
        map<ListNode*, bool> table;

        // Store head address in temp
        ListNode* temp = head;

        while(temp != NULL){
            if(table[temp] == false){
                table[temp] = true;
            }
            else{
                // Cycle present hai
                return true;
            }
            temp = temp->next;
        }

        // Loop not present
        return false;
    }
};
```

To C and S C = ?