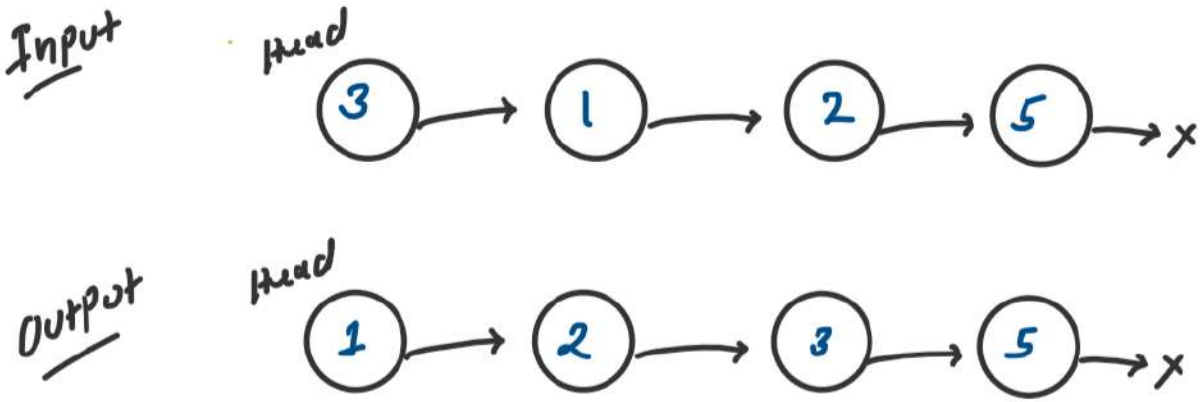
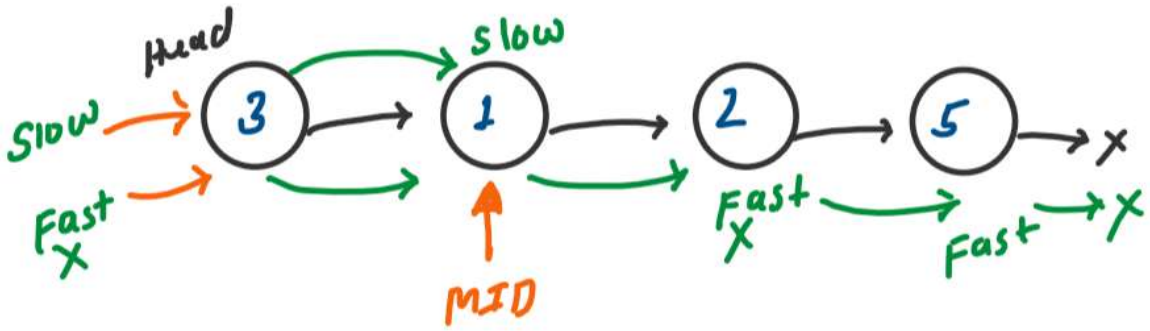
**HW 02: Sort Lists using Merge Sort (Leetcode-148)**




**MERGE SORT ALGORITH**

**Step 1:** Find mid position of the list
**Step 2:** Divide list into two half using mid
**Step 3:** Sort RE
**Step 4:** Merge both sorted list left and right

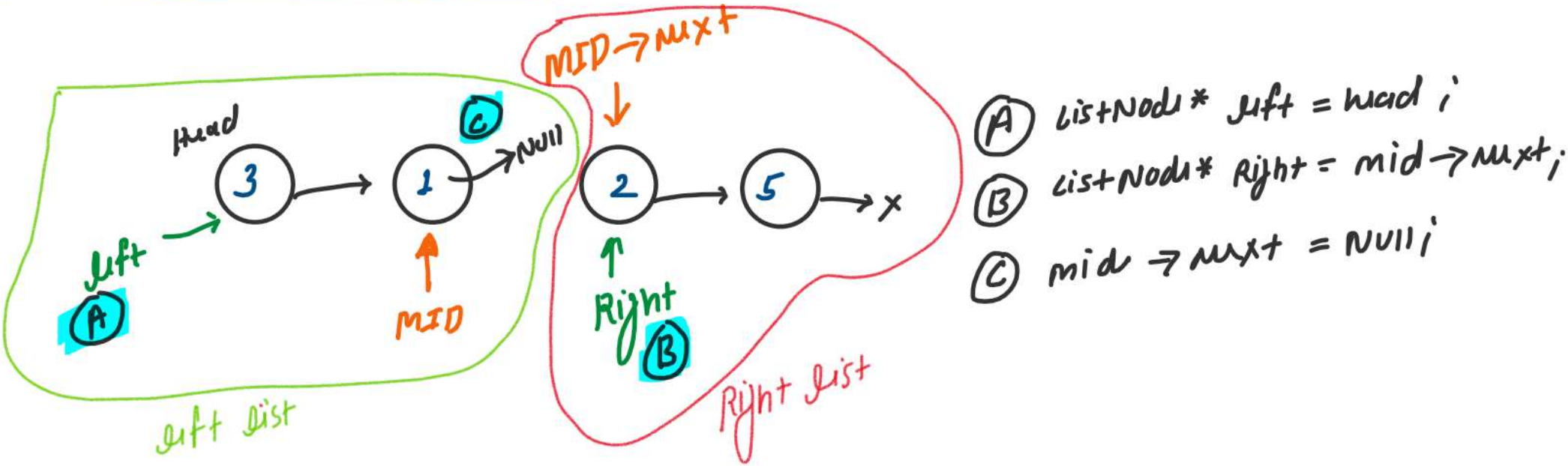
**Step 1:** *Find mid position of the list* using slow & Fast Algorithm



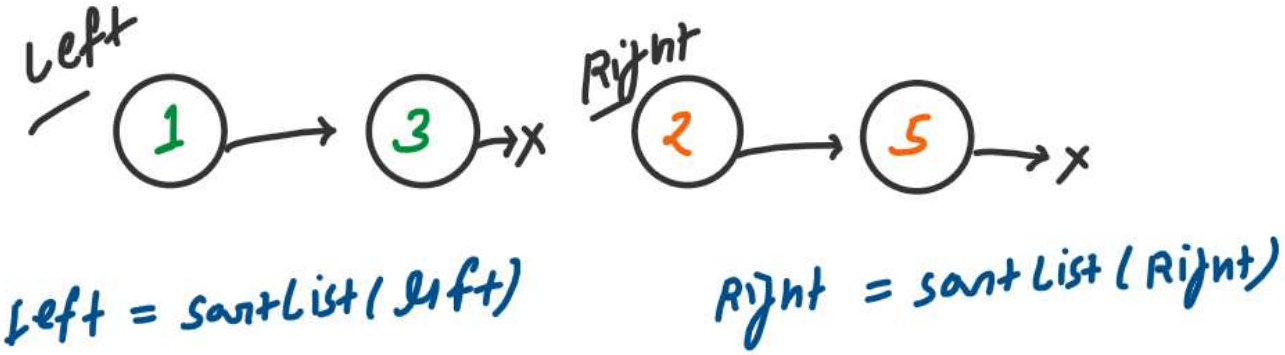

```
ListNode* getMid(ListNode* head){
    ListNode* slow = head;
    ListNode* fast = head;

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

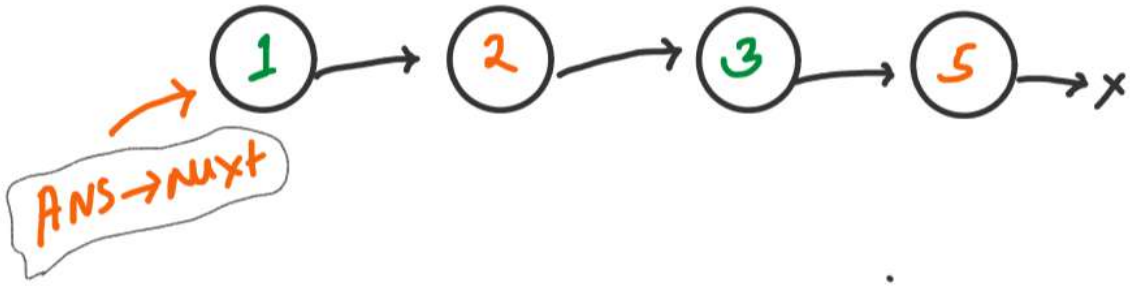
**Step 2:** *Divide list into two half using mid*



**Step 3:** Sort both list left and right RE



***Step 4:*** *Merge both sorted list left and right*




```cpp
ListNode* merge(ListNode* left, ListNode* right) {
    if(left == NULL) return right;
    if(right == NULL) return left;

    ListNode* ans = new ListNode(-1);
    ListNode* mptr = ans;

    while(left != NULL && right != NULL){
        if(left->val <= right->val){
            mptr->next = left;
            mptr = left;
            left = left->next;
        }
        else{
            mptr->next = right;
            mptr = right;
            right = right->next;
        }
    }

    if(left != NULL){
        mptr->next = left;
    }

    if(right != NULL){
        mptr->next = right;
    }

    return ans->next;
}
```

## COMPLETE CODE

```cpp
// HW 02: Sort Lists using Merge Sort (Leetcode-148)
/**
 * 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* getMid(ListNode* head){...}

    ListNode* merge(ListNode* left, ListNode* right){...}

    ListNode* sortList(ListNode* head) {
        // Base case
        if(head == NULL || head->next == NULL){
            return head;
        }

        // Step 1: Find mid position of the list
        ListNode* mid = getMid(head);

        // Step 2: Divede list into two half using mid
        ListNode* left = head;
        ListNode* right = mid->next;
        mid->next = NULL;

        // Step 3: Sort RE
        left = sortList(left);
        right = sortList(right);

        // Step 4: Merge both sorted list left and right
        ListNode* mergeLR = merge(left, right);
        return mergeLR;
    }
};
```

T.C.

getmid => T.C. = O(N)

merge => T.C. = O(N)

sortList => T.C. => O(logN)

Overall T.C. = O( O(getmid) + O(merge) ) * ( O(sortlist) )

= O( ( O(N) + O(N) ) * ( O(logN) ) )

= O( O(N) * O(logN) )

= O(NlogN)