DAY 26

INTERVIEW BIT PROBLEMS :

1. Sort List

Sort a linked list in **O(n log n)** time using constant space complexity.
Example :
Input : 1 -> 5 -> 4 -> 3

Returned list : 1 -> 3 -> 4 -> 5

CODE :

PYTHON

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # @param A : head node of linked list
    # @return the middle node in the linked list
    def find_middle(self,A):
        if A is None:
            return A
        slow_ptr=A
        fast_ptr=A
        while fast_ptr.next is not None and fast_ptr.next.next is not None:
            slow_ptr=slow_ptr.next
            fast_ptr=fast_ptr.next.next
        return slow_ptr

    def sort_list(self,l_ptr,r_ptr):
        out=None
        if l_ptr is None:
            return r_ptr
        if r_ptr is None:
            return l_ptr
        if l_ptr.val<=r_ptr.val:
            out=l_ptr
            out.next=self.sort_list(l_ptr.next,r_ptr)
        else:
            out=r_ptr
            out.next=self.sort_list(l_ptr,r_ptr.next)
        return out
```

```
    # @param A : head node of linked list
    # @return the head node in the linked list
    def sortList(self, A):
        if A is None or A.next is None:
            return A
        mid_ele=self.find_middle(A)
        next_mid=mid_ele.next
        mid_ele.next=None
        l=self.sortList(A)
        r=self.sortList(next_mid)
        result=self.sort_list(l,r)
        return result
```

(OR)

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # @param A : head node of linked list
    # @return the head node in the linked list
    def sortList(self, A):
        if not A:
            return None
        curr=A
        val_list=[]
        while curr:
            val_list.append(curr.val)
            curr=curr.next
        val_list.sort()
        curr=A
        for i in range(len(val_list)):
            curr.val=val_list[i]
            curr=curr.next
        return A
```

C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *      int val;
 *      ListNode *next;
 *      ListNode(int x) : val(x), next(NULL) {}
 * };
 */
pair<ListNode*,ListNode*> getMid(ListNode* A){
    ListNode* mid = A;
    ListNode* fast = A;
```

```
    ListNode* prev = NULL;

    while(fast!=NULL && fast->next!=NULL){
        prev = mid;
        mid = mid->next;
        fast = fast->next->next;
    }

    pair<ListNode*,ListNode*> result(prev, mid);
    return result;
}

ListNode* merge(ListNode* A, ListNode* B){

    ListNode* aNode = A;
    ListNode* bNode = B;
    ListNode* merged = NULL;
    ListNode* tail = NULL;

    while((aNode!=NULL) && (bNode!=NULL)){
        ListNode* insertedNode = NULL;

        if(aNode->val<bNode->val){
            insertedNode = aNode;
            aNode = aNode->next;
        }
        else {
            insertedNode = bNode;
            bNode = bNode->next;
        }

        if(merged){
            tail->next = insertedNode;
            tail = insertedNode;
        }
        else {
            merged = tail = insertedNode;
        }
    }

    //copy the remainder
    while(aNode!=NULL){
        tail->next = aNode;
        tail = aNode;
        aNode = aNode->next;
    }

    while(bNode!=NULL){
        tail->next = bNode;
        tail = bNode;
```

```
            bNode = bNode->next;
        }

        //Update the last node appropriately
        if(tail){
            tail->next = NULL;
        }

        return merged;
}

void mergeSort(ListNode*& A){
        if((A==NULL) || (A->next==NULL)) {
            return;
        }
        ListNode* mid = A;
        ListNode* fast = A;
        ListNode* prev = NULL;
        while(fast!=NULL && fast->next!=NULL){
            prev = mid;
            mid = mid->next;
            fast = fast->next->next;
        }
        if(prev){
            prev->next = NULL;
        }
        mergeSort(A);
        mergeSort(mid);
        A = merge(A,mid);
}

ListNode* Solution::sortList(ListNode* A) {
        // Do not write main() function.
        // Do not read input, instead use the arguments to the function.
        // Do not print the output, instead return values as specified
        mergeSort(A);
        return A;
}
```