LeetCode Problem Solving

Week-4

Q. Add Two Numbers → Number 2

=> Given two non empty Linked List. Digits store in reverse order. Our Task Is To Add The Digits And Return the result as a linked list in same order. Since We Have to Return The Result As Linked List So We Don’t have linked list we will create a node name copyList and tail to track the last node. We will store 0 in copyList Linked list. Tail now pointing copyList Then we will add and and that node at the last of copyList using Tail. After that we will simply remove the first node which contain 0 and return the result. Here is the code:

```
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
struct ListNode* addTwoNumbers(struct ListNode* l1, struct ListNode* l2) {
        struct ListNode* copyList = (struct ListNode*)malloc(sizeof(struct ListNode));
        struct ListNode* tail = copyList;
        copyList->val = 0;
        copyList->next = NULL;

        int sum, carry = 0;
        while(l1!=NULL || l2!=NULL){
                sum = 0;
                if(!l1 && l2) sum = sum + l2->val;
                else if(l1 && !l2) sum = sum + l1->val;
                else sum = sum + l1->val + l2->val;
                sum = sum + carry;
                carry = sum/10;
                struct ListNode* temp = (struct ListNode*)malloc(sizeof(struct ListNode));
                temp->val = sum%10;
                temp->next = NULL;
                tail->next = temp;
                tail = temp;
                if(l1) l1 = l1->next;
                if(l2) l2 = l2->next;
        }
        if(carry){
                struct ListNode* temp = (struct ListNode*)malloc(sizeof(struct ListNode));
                temp->val = carry;
                temp->next = NULL;
                tail->next = temp;
                tail = temp;
        }
        struct ListNode* result = copyList->next;
        free(copyList);
        return result;
}
```

Q. Merge Two Sorted Lists → Number 21

=> Here given two Single Linked List(Sorted). Our Target is to simply merge this two single linked list and return the result as a single linked list. To Solve this observer the problem. Here Linked List Is Sorted So just simply we will add the node at the last. Since we have to return the result as a single linked list we will create a copy single linked list. Then create a tail to easily add the node into the last without O(n) time complexity. Then create a loop until List1 and List2 Not Equal Null. In Loop we will check if List1 Value greater than List2 Value then add list2 node in the last and move the list2. Then if List1 Value Lower than list2 then add list1 node in the last and move the list1. After loop end. If list1 not null then simply add remaining list1 into the last if list2 not null then simply add remaining list2 into the last. Since we have created the copy single linked list with 0 so now we will delete that and then return the remaining list. Here is the code:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode* list2) {
        struct ListNode* copyList = (struct ListNode*)malloc(sizeof(struct ListNode));
        struct ListNode* tail = copyList;
        copyList->val = 0;
        copyList->next = NULL;
        while(list1!=NULL && list2!=NULL){
                if(list1->val > list2->val){
                        tail->next = list2;
                        list2 = list2->next;
                }
                else{
                        tail->next = list1;
                        list1 = list1->next;
                }
                tail = tail->next;
        }
        if(list1!=NULL && list2==NULL){
                tail->next = list1;
        }
        if(list1==NULL && list2!=NULL){
                tail->next = list2;
        }
        struct ListNode* result = copyList->next;
        free(copyList);
        return result;
}
```

Q. Search Insert Position → Number 35

=> Given an array and an number. Array here is sorted. Our target is to find the given number into array and if found return it's position in array if not found then return the position where this number should be inserted. We can do this in two method. First Method Loop & check. Second Method Binary Search Tree.

First Method: Here We Will Do A Loop from 0 to last number of the array. If we found number in array nums[i]==target simply return index which is I. If not then check if number is greater than target simply return I. Else simply return numsSize which means number didn't found and its higher than all the number that's why return numsSize. Here is the code.

```
int searchInsert(int* nums, int numsSize, int target) {
        int i;
        for(i=0; i<numsSize;i++){
                if(nums[i]==target){
                        return i;
                }
                if(nums[i]>target){
                        return i;
                }
        }
        return numsSize;
}
```

But Here is a problem here time complexity is O(n). We can reduce this to O(log n) by this Binary Seach Tree Method.

Second Method: Here we will use left, right and mid variable. Left is the 1st number in Array and Right is the Last Number in Array and Mid is the Middle Number in Array. Here we will do a loop until left not greater than right. If left becomes greater than right immedietly stop the loop. In loop we will calculate the mid then check if the mid number is equal to target. If equal then simply return mid. If not equal then check if mid < target if then left = mid+1. Here target greater than mid that why we will simply make left which was previously 1st now becomes the mid+1. Else if mid > target which means target is lower than mid then simply we will make right which was previously last number now becomes the mid-1. After loop end Simply Return Left. Here is the code

```
int searchInsert(int* nums, int numsSize, int target) {
        int left = 0;
        int right = numsSize - 1;

        while (left <= right) {
                int mid = left + (right - left) / 2;
                if (nums[mid] == target) {
                        return mid;
                }
                else if (nums[mid] < target) {
                        left = mid + 1;
                }
                else {
                        right = mid - 1;
                }
        }
        return left;
}
```

Here Time Complexity Is O(log n) because we are reducing the loop into half. That’s why its best method to solve this problem.