LeetCode Problem Solving

Week-2

Q. Rotate Array → Number 189

=> Given An Array and an int value k. Task is to rotate the array by k steps. Suppose k = 3. which means I have to reverse the array 3 times. Not whole array 3 means last 3 digit will come to front. Example:
Array = [1,2,3,4,5,6,7] k = 3
Output = [5,6,7,1,2,3,4]
So means Left shift the last number 3 times.
We can solve this using 2 Method. One is Brute Force Method And One Is Three Step Reverse Method.

First Method: Brute Force Method. In this method we will Left Shift The Last Number Till It Comes To First. After That Again We Will Left Shift The Last Number Till It Comes To First. We Will Do This Till k Times. Here Is The Code.

```
void rotate(int* nums, int numsSize, int k) {
        int i = numsSize-1;
        int temp;
        while(k>0){
                i = numsSize-1;
                while(i>0){
                        temp = nums[i];
                        nums[i] = nums[i-1];
                        nums[i-1] = temp;
                        i--;
                }
                k--;
        }
}
```

This Method Is Suitable If Array Size Is Few. But Its Not Perfect or Correct If Array Size is Larger than 10^5. Then we will face Time Exceed Problem. To Fix This We Have Three Step Reverse Method. Here is that Method.

Second Method: Three Step Reverse Method. In This Method We Will Reverse The Whole Array First. Then We Will Reverse The First Part Then Second Part. First Part Means Suppose K = 3 So According To The Question Logic Is That Last 3 Will Come First And First 4 will Come Last. Since We Reverse The Whole Array So Now We Will Reverse First 3 And Then Last 4 To Sort. Because Last 3 Comes In First With Reverse Because We Reverse The Whole Array That's why we will reverse these two parts again. And this is the Three Step Reverse Method. Here is the Code.

```
void reverse(int* nums, int i, int j){
        int temp;
        while(i<j){
                temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
                i++;
                j--;
        }
}
void rotate(int* nums, int numsSize, int k) {
        k = k % numsSize;
        if (k == 0 || numsSize <= 1) return;
        reverse(nums, 0, numsSize-1);
        reverse(nums, 0, k-1);
        reverse(nums, k, numsSize-1);
}
```

This is optimal, easiest and perfect solution.
K = k%numsSize reduce the time complexity

Suppose numsSize is 7 and k = 8. So we don't need to reverse 8 times. 8 time reverse = 1
and if (k == 0 || numsSize <= 1) return; this handle edge cases suppose if k = 0 then we don't need to reverse we will simply exit using return. And if numsSize<=1 means array element is 0 and 1 then we don't need to reverse we will exit using return.

Q. Best Time To Buy And Sell Stock → Number 121

=> Given An Array the value of the array is the prices of the stock. Our task is to find the maximum profits. Remember we can't sell before buying. There are some methods to do this. First Method is solve using functions. Here is the code.

```
int profitCount(int buy, int sell){
	int profit;
	profit = sell-buy;
	if(profit<=0){
		return 0;
	}
	else{
		return profit;
	}
}

int maxProfit(int* prices, int pricesSize) {
	int i = 0;
	int j = i+1;
	int profit;
	int high = 0;
	if(pricesSize==0 || pricesSize==1){
		return 0;
	}
	else{
		while(i!=pricesSize-1){
			profit = profitCount(prices[i], prices[j++]);
			if(profit>high){
				high = profit;
			}
			if(j==pricesSize){
				i++;
				j = i+1;
			}
		}
		return high;
	}
}
```

Our This Method Is Perfect for Smallest Input not perfect for larger input. For larger input our code will get time limit exceed. Because our Time Complexity is O(n^2); So this is not ideal for LeetCode. In LeetCode we always try to make Time Complexity O(n) and if possible then O(1). For solving this we have another method. Here is the code for that method.

```
int maxProfit(int* prices, int pricesSize) {
	int minPrice,i,profit,maxProfits;
	minPrice = prices[0];
	maxProfits = 0;
	for(i=0;i<pricesSize;i++){
		if(prices[i]<minPrice){
			minPrice = prices[i];
		}
		profit = prices[i]-minPrice;
		if(profit>maxProfits){
			maxProfits = profit;
		}
	}
	return maxProfits;
}
```

This is perfect and also time complexity is O(n). But it's runtime in LeetCode is 4ms. This is not so much problem but it's always good practice to keep runtime 0ms. So we just need a slight change in our current code to make this runtime 0ms. We just don't calculate profit in line we rather calculate in if condition and then its done. Here is the code.

```
int maxProfit(int* prices, int pricesSize) {
	int minPrice,i,profit,maxProfits;
	minPrice = prices[0];
	maxProfits = 0;
	for(i=0;i<pricesSize;i++){
		if(prices[i]<minPrice){
			minPrice = prices[i];
		}
		if(prices[i]-minPrice>maxProfits){
			maxProfits = prices[i]-minPrice;
		}
	}
	return maxProfits;
}
```

Q. Linked List Cycle → Number 141

=> Given A Singly Linked List. And Head Pointer. Our Target is to find the cycle in this singly linked list. So the problem is that we can't traverse since there is cycle so it will run infinite time. So we cant do traverse and check then tell is either Circle or not. But there is a solution. In our childhood, we learned Hare & Tortoise Story. We will use this now. We Will Use two pointer one will work like hare and one will work like tortoise. Hare pointer will move fast and tortoise pointer will move slow. Faster means move two step and slower means move one step. Here is the code.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool hasCycle(struct ListNode *head) {
        struct ListNode* p = head;
        struct ListNode* q = head;
        do{
                if(p==NULL){
                        return false;
                }
                else if(p->next==NULL){
                        return false;
                }
                else{
                        p = p->next->next;
                        q = q->next;
                }
        }while(p!=q);
        return true;
}
```

This is perfect but problem run time is high. Because we used too much if else we can reduce runtime by reducing them. Here is neat and clean code.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
bool hasCycle(struct ListNode *head) {
    if (!head || !head->next) return false;

    struct ListNode *slow = head;
    struct ListNode *fast = head->next;

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