DSA PRACTICE - 7

Name : Sidharth N
Dept : AI & DS - “C”

1.Next Permutation

```java
class Solution {
    public void nextPermutation(int[] nums) {
        int point_change = -1;
        int eleswap =  0;
        for(int i = nums.length - 1 ;i>0 ;i--){
            if(nums[i] > nums[i-1]){
                point_change = i-1;
                break;
            }
        }
        if(point_change == -1){
            Arrays.sort(nums);
            return;
        }
        for (int j = nums.length-1 ; j > point_change ;j--){
            if(nums[j] > nums[point_change]){
                eleswap = j;
                break;
            }
        }
        int temp = nums[eleswap];
        nums[eleswap] = nums[point_change];
        nums[point_change] = temp;

        int temp2 = point_change + 1;
        int temp1 = nums.length - 1;
        while (temp2 < temp1){
            int swap = nums[temp2];
            nums[temp2] = nums[temp1];
            nums[temp1] = swap;
            temp2 ++;
            temp1 --;
        }

    }

}
```

Time Complexity : O(n)

2.Longest SubString without Repeating Characters

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int ans = 0;
        Queue <Character> q = new LinkedList<>();
        for(char c : s.toCharArray()){
            while(q.contains(c)){
                q.poll();
            }
            q.offer(c);
            ans = Math.max(q.size(),ans);
        }
        return ans;
    }
}
```

Time Complexity : O(n)

3.Palindrome Linked List

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        ListNode slow = head , fast = head , prev , next;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        prev = slow;
        slow = slow.next;
        prev.next = null;

        while(slow != null){
            next = slow.next;
            slow.next = prev;
            prev = slow;
            slow = next;
        }
        slow = prev;
        fast = head;

        while(slow != null){
            if(fast.val != slow.val){
                return false;
            }
            fast = fast.next;
```

```
            slow = slow.next;
        }

        return true;
    }
}
```

Time Complexity : O(n)

4.Remove Linked List Elements

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        while(head != null && head.val == val){
            head = head.next;
        }
        ListNode curr = head;
        while (curr != null && curr.next != null){
            if(curr.next.val == val){
                curr.next = curr.next.next;
            }
            else{
                curr = curr.next;
            }
        }
        return head;
    }
}
```

Time Complexity : O(n)

5.Spiral Matrix

```
public class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List <Integer> ans = new ArrayList<Integer>();
        int top = 0;
        int left = 0;
        int right = matrix[0].length -1 ;
        int down = matrix.length - 1;
        while(true){
            for(int i=left ;i <= right;i++){
            ans.add(matrix[top][i]);
        }
            top++;
```

```
            if(left > right || top > down) break;

            for(int j = top; j <= down ;j++){
                ans.add(matrix[j][right]);
            }
            right --;
            if(left > right || top > down) break;

            for(int k = right ; k >= left ;k--){
                ans.add(matrix[down][k]);
            }
            down --;
            if(left > right || top > down) break;

            for(int l = down; l >= top ;l--){
                ans.add(matrix[l][left]);
            }
            left++;
            if(left > right || top > down) break;
        }
        return ans;

    }

}
```

Time Complexity : O(n)