1. Two Sum

1. Two Sum Given nums = [2, 7, 11, 15], target = 9, Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1].

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int[] res = new int[]{-1,-1};
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(target-nums[i])) {
                return new int[]{map.get(target-nums[i]),i};
            }else {
                map.put(nums[i],i);
            }
        }
        return res;
    }
}
```

2. Add Two Numbers

2. Add Two Numbers Input: (2 -> 4 -> 3) + (5 -> 6 -> 4) Output: 7 -> 0 -> 8 Explanation: 342 + 465 = 807.

```
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode p1 = l1;
        ListNode p2 = l2;
        ListNode dummyNode = new ListNode(-1);
        int carry = 0, sum = 0;
        ListNode cur = dummyNode;
        while (p1 != null || p2 != null) {
            sum = carry;
            if (p1 != null) {
                sum += p1.val;
                p1 = p1.next;
            }
            if (p2 != null) {
                sum += p2.val;
                p2 = p2.next;
            }
            cur.next = new ListNode(sum%10);
            carry = sum / 10;
            cur = cur.next;
        }
        if (carry != 0) {
            cur.next = new ListNode(carry);
        }
        return dummyNode.next;
    }
}
```

---

3. Longest Substring Without Repeating Characters

3. Longest Substring Without Repeating Characters Input: "abcabcbb" Output: 3 Explanation: The answer is "abc", with the length of 3.

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        if(s == null || s.length() == 0) return 0;
        int left = 0, right = 0;
        int max = 0, n = s.length();
        boolean[] used = new boolean[128];
        while (right < n) {
            if (used[s.charAt(right)] == false) {
                used[s.charAt(right)] = true;
                right++;
            }else {

                max = Math.max(max, right-left);
                while (left < right && s.charAt(left) != s.charAt(right)) {
                    used[s.charAt(left)] = false;
                    left++;
                }
                left++;
                right++;
            }
        }
        max = Math.max(max, right - left);
        return max;
    }
}
```

6. ZigZag Conversion

6. ZigZag Conversion Input: s = "PAYPALISHIRING", numRows = 3 Output: "PAHNAPLSIIGYIR"

```
class Solution {
 public String convert(String s, int numRows) {
     char[] c = s.toCharArray();
     int len = c.length;
     StringBuilder[] sb = new StringBuilder[numRows];
     for (int i = 0; i < numRows; i++) {
         sb[i] = new StringBuilder();
     }

     int i = 0;
     while( i < len) {
         for (int idx = 0; idx < numRows && i < len; idx++) {
             sb[idx].append(c[i++]);
         }
         for (int idx = numRows - 2; idx >= 1 && i < len; idx--) {
             sb[idx].append(c[i++]);
         }
     }
     for (int idx = 1; idx < numRows; idx++) {
         sb[0].append(sb[idx]);
     }
     return sb[0].toString();
 }
}
```

7. Reverse Integer

7. Reverse Integer Input: -123 Output: -321

tips: -123 / 100 = -1 -123 % 10 = -3

```
class Solution {
    public int reverse(int x) {
        long rev = 0;

        while (x != 0) {
            rev = rev * 10 + x % 10;
            x = x / 10;
        }
        if (rev > Integer.MAX_VALUE || rev < Integer.MIN_VALUE) return 0;
        return (int)rev;
    }
}
```

9. Palindrome Number

9. Palindrome Number Input: 121 Output: true

```
class Solution {
    public boolean isPalindrome(int num) {
        if(num < 0) return false;
        int div = 1;
        while (num / div >= 10) {
            div *= 10;
        }

        while(num != 0) {
            int right = num / div;
            int left = num % 10;
            if (right != left) return false;
            num = (num- right * div) / 10;
            div /= 100;
        }
        return true;
    }
}
```

13. Roman to Integer

13. Roman to Integer Input: "MCMXCIV" Output: 1994 Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

```
class Solution {
public int romanToInt(String s) {
    if(s == null || s.length() == 0) return 0;
    int res = 0;
    if (s.indexOf("CM") != -1) res -= 200;
    if (s.indexOf("CD") != -1) res -= 200;
    if (s.indexOf("XC") != -1) res -= 20;
    if (s.indexOf("XL") != -1) res -= 20;
    if (s.indexOf("IX") != -1) res -= 2;
    if (s.indexOf("IV") != -1) res -= 2;

    for (char c : s.toCharArray()) {
        if (c == 'M') res+= 1000;
        if (c == 'D') res+= 500;
        if (c == 'C') res+= 100;
        if (c == 'L') res+= 50;
        if (c == 'X') res+= 10;
        if (c == 'V') res+= 5;
        if (c == 'I') res+= 1;
    }
    return res;
}
}
```

19. Remove Nth Node From End of List

19. Remove Nth Node From End of List n == 1. then remove the last one Given linked list: 1->2->3->4->5, and n = 2. After removing the second node from the end, the linked list becomes 1->2->3->5.

```
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode dummyNode = new ListNode(-1);
        dummyNode.next = head;

        ListNode fast = dummyNode;
        ListNode slow = dummyNode;
        for (int i = 0; i < n; i++) {
            fast = fast.next;
        }
        while (fast.next != null) {
            fast = fast.next;
            slow = slow.next;
        }
        slow.next = slow.next.next;
        return dummyNode.next;
    }
}
```

74. Search a 2D Matrix

matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 50]] target = 3 Output: true

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0) return false;
        int row = matrix.length, col = matrix[0].length;
        int left = 0, right = row * col -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (matrix[mid/col][mid%col] == target)
                return true;
            else if (matrix[mid/col][mid%col] > target) {
                right = mid - 1;
            }else {
                left = mid + 1;
            }
        }
        return false;
    }
}
```