1.Add Two Numbers

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.
You may assume the two numbers do not contain any leading zero, except the number 0 itself.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummyHead = new ListNode(0); // Dummy node to simplify appending
        ListNode curr = dummyHead;
        int carry = 0;

        // Traverse both lists
        while (l1 != null || l2 != null || carry != 0) {
            int x = (l1 != null) ? l1.val : 0; // value from l1 or 0
            int y = (l2 != null) ? l2.val : 0; // value from l2 or 0

            int sum = x + y + carry;
            carry = sum / 10; // calculate carry for next iteration
            curr.next = new ListNode(sum % 10); // append digit to result list
            curr = curr.next;

            // move to next nodes
            if (l1 != null) l1 = l1.next;
            if (l2 != null) l2 = l2.next;
        }

        return dummyHead.next; // return the actual result list
    }
}
```

Output:
l1 = [2,4,3]
l2 = [5,6,4]

[7,0,8]

2.Longest Substring without repeating characters

Given a string s, find the length of the longest substring without duplicate characters.

```
class Solution {
   public int lengthOfLongestSubstring(String s)   {
      int start =0,end = 0,result=0;
      List<Character> list = new ArrayList<Character>();
      while(end<s.length()) {
         if(!list.contains(s.charAt(end))) {
            list.add(s.charAt(end));
            end++;
            result = Math.max(result,list.size());
         }
         else {
            list.remove(Character.valueOf(s.charAt(start)));
            start++;
         }
      }
      return result;
   }
}
```

Output:
"abcabcbb"

3.Merge sort algorithm

```
public class MergeSorting {
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;

            mergeSort(arr, left, mid);

            mergeSort(arr, mid + 1, right);

            merge(arr, left, mid, right);
```

```
    }
}

public static void merge(int[] arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int[] L = new int[n1];
    int[] R = new int[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

  public static void main(String[] args) {
    int[] arr = {38, 27, 43, 3, 9, 82, 10};
    mergeSort(arr, 0, arr.length - 1);

    System.out.println("\nSorted Array :");
    for (int num : arr) {
        System.out.print(num + " ");
    }
}
```

```
}
```

4.Median of two Arrays

```
public class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1);
        }

        int x = nums1.length;
        int y = nums2.length;
        int low = 0, high = x;

        while (low <= high) {
            int partitionX = (low + high) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxLeftX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minRightX = (partitionX == x) ? Integer.MAX_VALUE : nums1[partitionX];

            int maxLeftY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minRightY = (partitionY == y) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxLeftX <= minRightY && maxLeftY <= minRightX) {
                if ((x + y) % 2 == 0) {
                    return (Math.max(maxLeftX, maxLeftY) + Math.min(minRightX, minRightY)) / 2.0;
                } else {
                    return Math.max(maxLeftX, maxLeftY);
                }
            } else if (maxLeftX > minRightY) {
                high = partitionX - 1;
            } else {
                low = partitionX + 1;
            }
        }

        throw new IllegalArgumentException("Input arrays are not sorted or invalid.");
    }
}
```

5.longest palindromic substring

```
public class Solution {
    public String longestPalindrome(String s) {
        if (s == null || s.length() < 1) return ""; //if input is null
```

```
        int start = 0, end = 0;

        for (int i = 0; i < s.length(); i++) {
            int len1 = expandFromCenter(s, i, i);     // Odd length palindrome
            int len2 = expandFromCenter(s, i, i + 1); // Even length palindrome
            int len = Math.max(len1, len2);

            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + len / 2;
            }
        }

        return s.substring(start, end + 1);
    }

    private int expandFromCenter(String s, int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1; // Length of palindrome
    }
}
```

6.Reverse an integer

```
class Solution {
    public int reverse(int x) {
        int r = 0;
        while(x!=0) {
            int d = x%10;
            x = x/10;
            if (r > Integer.MAX_VALUE / 10 || (r == Integer.MAX_VALUE / 10 && d > 7)) return 0;
            if (r < Integer.MIN_VALUE / 10 || (r == Integer.MIN_VALUE / 10 && d < -8)) return 0;
             r = r*10+d;
        }
        return r;
    }
}
```