```java
import java.util.ArrayList;
import java.util.List;

class Solution {
    public List<String> summaryRanges(int[] nums) {
        List<String> result = new ArrayList<>();

        if (nums == null || nums.length == 0) {
            return result;
        }

        int start = nums[0]; // Start of the range

        for (int i = 1; i < nums.length; i++) {
            // If nums[i] is not consecutive
            if (nums[i] != nums[i - 1] + 1) {
                // Add the range to the result
                if (start == nums[i - 1]) {
                    result.add(String.valueOf(start));
                } else {
                    result.add(start + "->" + nums[i - 1]);
                }
                // Update the start of the new range
                start = nums[i];
            }
        }

        // Add the last range
        if (start == nums[nums.length - 1]) {
            result.add(String.valueOf(start));
        } else {
            result.add(start + "->" + nums[nums.length - 1]);
        }

        return result;
    }

}
```

```
if (s == null || s.length() == 0) {
            return 0;
        }

        // Variables to track the last two states
        int prev1 = 1; // Ways to decode an empty string
        int prev2 = 0; // Ways to decode up to the previous character

        for (int i = 0; i < s.length(); i++) {
            int current = 0;

            // Check if the current single digit is valid (1-9)
            if (s.charAt(i) != '0') {
                current += prev1;
            }

            // Check if the last two digits form a valid number (10-26)
            if (i > 0) {
                int twoDigit = Integer.parseInt(s.substring(i - 1, i + 1));
                if (twoDigit >= 10 && twoDigit <= 26) {
                    current += prev2;
                }
            }

            // Update prev2 and prev1 for the next iteration
            prev2 = prev1;
            prev1 = current;
        }

        return prev1;
```

```
// Handle overflow cases
        if (dividend == Integer.MIN_VALUE && divisor == -1) {
            return Integer.MAX_VALUE; // Overflow case
        }
        if (dividend == Integer.MIN_VALUE && divisor == 1) {
            return Integer.MIN_VALUE;
        }
```

```
        // Determine the sign of the result
        boolean isNegative = (dividend < 0) ^ (divisor < 0); // XOR to
determine if the signs differ

        // Work with positive values
        long absDividend = Math.abs((long) dividend);
        long absDivisor = Math.abs((long) divisor);

        int result = 0;

        // Subtract divisor using bit manipulation
        while (absDividend >= absDivisor) {
            long tempDivisor = absDivisor;
            int multiple = 1;

            // Increase tempDivisor by powers of 2
            while (absDividend >= (tempDivisor << 1)) {
                tempDivisor <<= 1;
                multiple <<= 1;
            }

            // Subtract the largest shifted divisor from the dividend
            absDividend -= tempDivisor;
            result += multiple;
        }

        // Apply the sign to the result
        return isNegative ? -result : result;
```