1. Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

```
Input: s = "babad"

Output: "bab"

Note: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"

Output: "bb"
```

Example 3:

```
Input: s = "a"

Output: "a"
```

Example 4:

```
Input: s = "ac"

Output: "a"
```

2. Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

```
Symbol       Value
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral.

Example 1:

```
Input: num = 3
Output: "III"
```

Example 2:

```
Input: num = 4
Output: "IV"
```

Example 3:

```
Input: num = 9
Output: "IX"
```

Example 4:

```
Input: num = 58

Output: "LVIII"

Explanation: L = 50, V = 5, III = 3.
```

Example 5:

```
Input: num = 1994

Output: "MCMXCIV"

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Constraints: `1 <= num <= 3999`

3. Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

Example 3:

```
Input: digits = "2"
Output: ["a","b","c"]
```

Constraints:

- `0 <= digits.length <= 4`
- `digits[i]` is a digit in the range `['2', '9']`.

4. Given two integers `dividend` and `divisor`, divide two integers without using multiplication, division, and mod operator.

Return the quotient after dividing `dividend` by `divisor`.

The integer division should truncate toward zero, which means losing its fractional part. For example, `truncate(8.345) = 8` and `truncate(-2.7335) = -2`.

Note:

- Assume we are dealing with an environment that could only store integers within the 32-bit signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, assume that your function **returns $2^{31} - 1$ when the division result overflows**.

Example 1:

```
Input: dividend = 10, divisor = 3

Output: 3

Explanation: 10/3 = truncate(3.33333..) = 3.
```

Example 2:

```
Input: dividend = 7, divisor = -3

Output: -2

Explanation: 7/-3 = truncate(-2.33333..) = -2.
```

Example 3:

```
Input: dividend = 0, divisor = 1

Output: 0
```

Example 4:

```
Input: dividend = 1, divisor = 1

Output: 1
```

Constraints:

- -2^{31} `<= dividend, divisor <=` $2^{31} - 1$
- `divisor != 0`

5.

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

Follow up: Could you write an algorithm with `O(log n)` runtime complexity?

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]
```

Example 3:

```
Input: nums = [], target = 0

Output: [-1,-1]
```

Constraints:

- $0 <= nums.length <= 10^5$
- $-10^9 <= nums[i] <= 10^9$
- `nums` is a non-decreasing array.
- $-10^9 <= target <= 10^9$

6.

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`*.* You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7

Output: [[2,2,3],[7]]

Explanation:

2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.

7 is a candidate, and 7 = 7.

These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8

Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1

Output: []
```

Example 4:

```
Input: candidates = [1], target = 1

Output: [[1]]
```

Example 5:

```
Input: candidates = [1], target = 2

Output: [[1,1]]
```

Constraints:

- `1 <= candidates.length <= 30`
- `1 <= candidates[i] <= 200`
- All elements of `candidates` are **distinct**.
- `1 <= target <= 500`