167. Two Sum II - Input Array Is Sorted

Medium 5579 947 Add to List Share

Given a **1-indexed** array of integers `numbers` that is already ***sorted in non-decreasing order***, find two numbers such that they add up to a specific `target` number. Let these two numbers be `numbers[index₁]` and `numbers[index₂]` where `1 <= index₁ < index₂ <= numbers.length`.

Return *the indices of the two numbers,* `index₁` *and* `index₂`*,* ***added by one*** *as an integer array* `[index₁, index₂]` *of length 2.*

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

```
Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore, index₁ = 1,
index₂ = 2. We return [1, 2].
```

Example 2:

```
Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index₁ = 1, index₂
= 3. We return [1, 3].
```

Example 3:

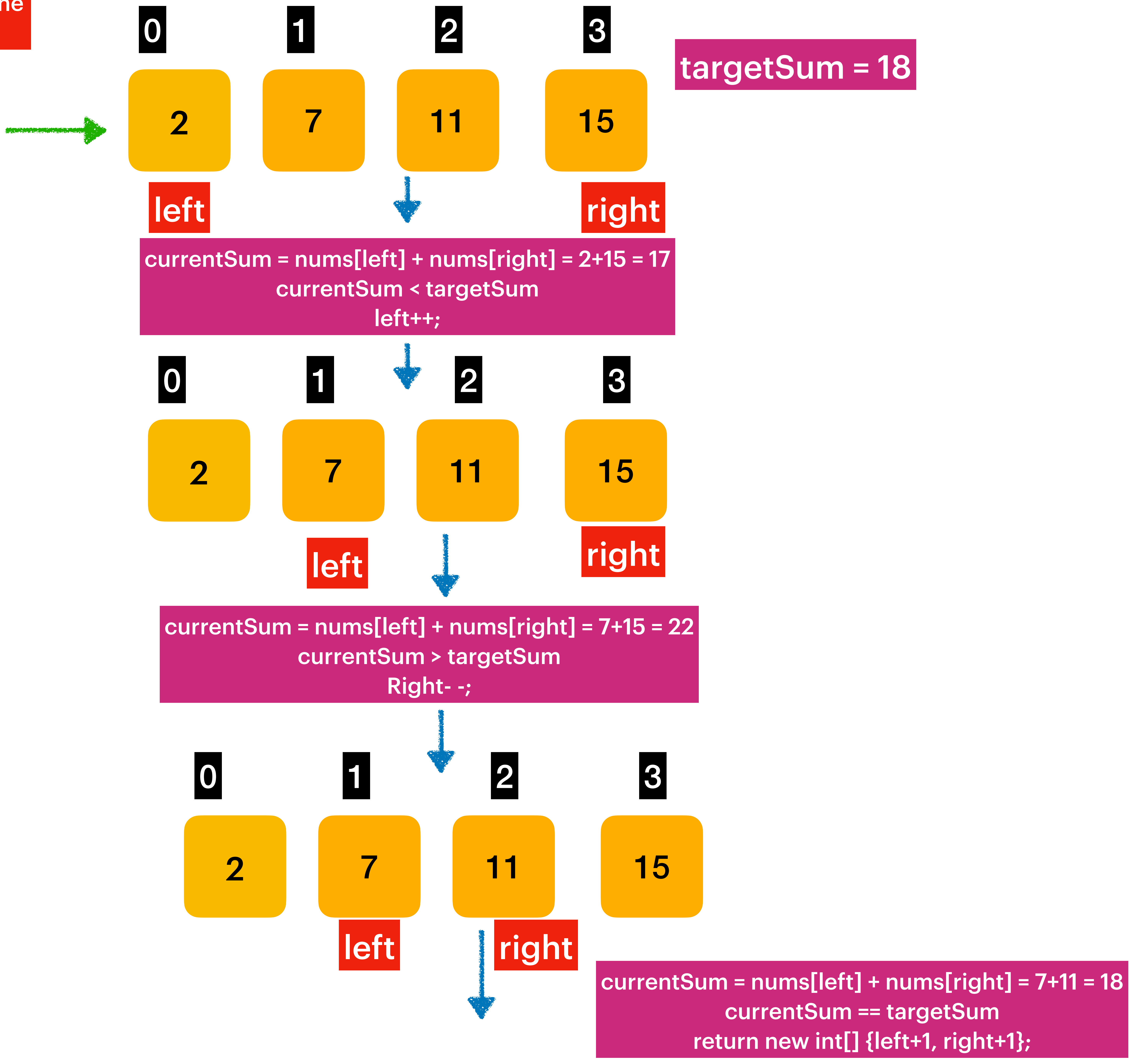
```
Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index₁ = 1,
index₂ = 2. We return [1, 2].
```

Constraints:

- `2 <= numbers.length <= 3 * 10⁴`
- `-1000 <= numbers[i] <= 1000`
- `numbers` is sorted in **non-decreasing order**.
- `-1000 <= target <= 1000`
- The tests are generated such that there is **exactly one solution**.

Solution 1: Use HashMap just like prev One
Takes Space : O(n)

Solution 2:
Two Pointer Technique
Time Complexity : O(n/2) = O(n)
Space Complexity : O(1)



currentSum = nums[left] + nums[right] = 2+15 = 17
currentSum < targetSum
left++;

currentSum = nums[left] + nums[right] = 7+15 = 22
currentSum > targetSum
Right- -;

currentSum = nums[left] + nums[right] = 7+11 = 18
currentSum == targetSum
return new int[] {left+1, right+1};

13. Roman to Integer

Easy 4335 303 Add to List Share

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 a roman numeral, convert it to an integer.

Example 1:

```
Input: s = "III"
Output: 3
Explanation: III = 3.
```

Example 2:

```
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 3:

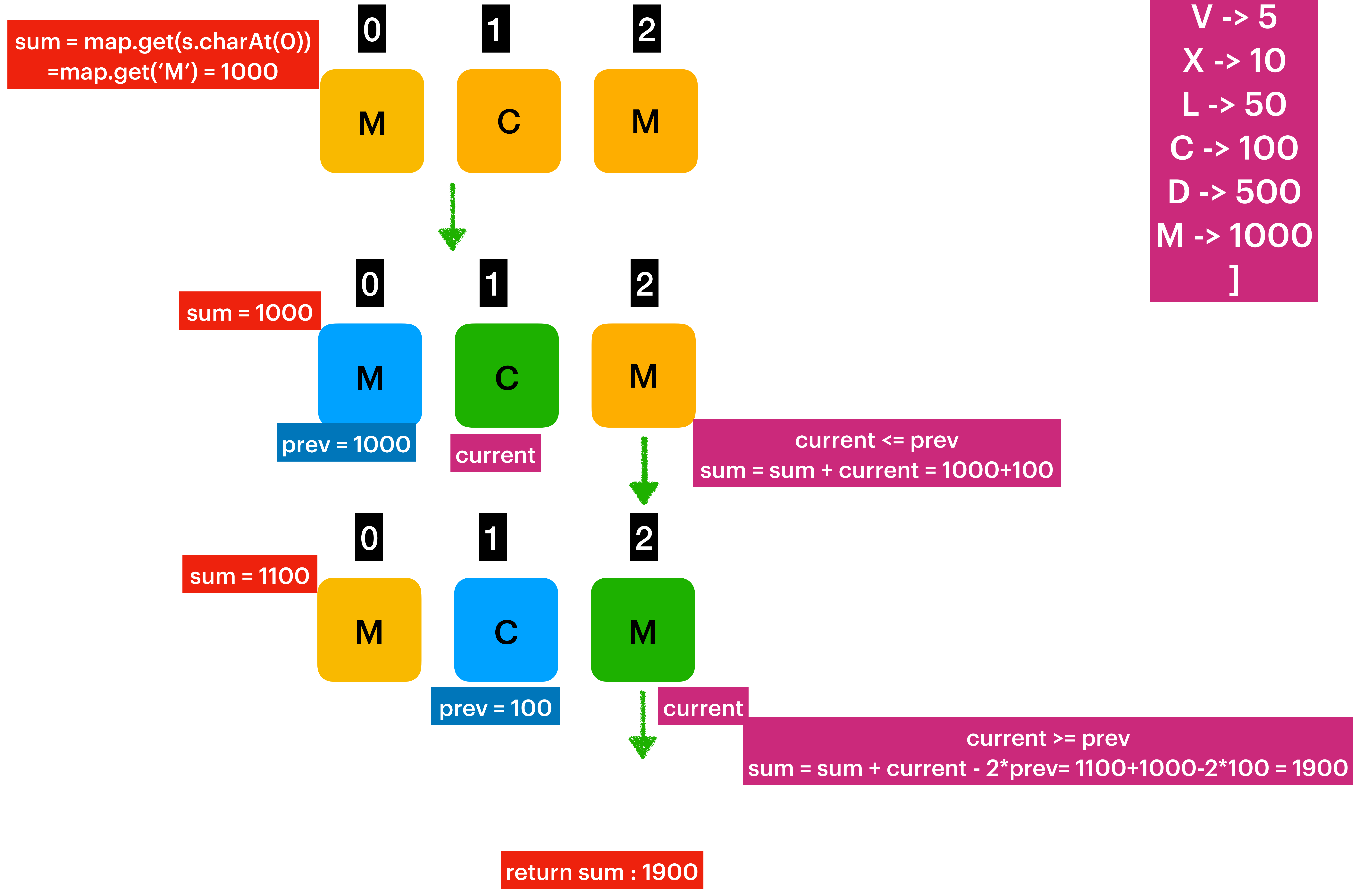
```
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Constraints:

- `1 <= s.length <= 15`
- `s` contains only the characters `('I', 'V', 'X', 'L', 'C', 'D', 'M')`.
- It is **guaranteed** that `s` is a valid roman numeral in the range `[1, 3999]`.

MCM -> M + CM = 1000 + 900 = 1900

Map [
I -> 1
V -> 5
X -> 10
L -> 50
C -> 100
D -> 500
M -> 1000
]


sum = map.get(s.charAt(0))
=map.get('M') = 1000
0
1
2
M
C
M
sum = 1000
0
1
2
M
C
M
prev = 1000
current
current <= prev
sum = sum + current = 1000+100
sum = 1100
0
1
2
M
C
M
prev = 100
current
current >= prev
sum = sum + current - 2*prev= 1100+1000-2*100 = 1900
return sum : 1900