

# **LeetCode Solutions**

Sankalp Sangle

Last updated on May 4, 2020



# Contents

<b>I</b>	<b>LeetCode Top Interview Questions</b>	<b>1</b>
<b>1</b>	<b>Easy</b>	<b>3</b>
1.1	Arrays . . . . .	3
1.1.1	26. Remove Duplicates from Sorted Array . . . . .	3
1.1.2	122. Best Time to Buy and Sell Stock II . . . . .	4
1.1.3	189. Rotate Array . . . . .	4
1.1.4	217. Contains Duplicate . . . . .	5
1.1.5	136. Single Number . . . . .	5
1.1.6	350. Intersection of Two Arrays II . . . . .	6
1.1.7	66. Plus One . . . . .	6
1.1.8	283. Move Zeroes . . . . .	7
1.1.9	1. Two Sum . . . . .	7
1.1.10	36. Valid Sudoku . . . . .	8
1.1.11	48. Rotate Image . . . . .	9
1.2	Strings . . . . .	10
1.2.1	344. Reverse String . . . . .	10

1.2.2	7. Reverse Integer . . . . .	10
1.2.3	387. First Unique Character in a String . . . . .	11
1.2.4	242. Valid Anagram . . . . .	11
1.2.5	125. Valid Palindrome . . . . .	11
1.2.6	28. Implement strStr() . . . . .	12
1.2.7	38. Count and Say . . . . .	13
1.2.8	14. Longest Common Prefix . . . . .	13
1.3	Linked Lists . . . . .	14
1.3.1	237. Delete Node in a Linked List . . . . .	14
1.3.2	19. Remove Nth Node From End of List . . . . .	14
1.3.3	206. Reverse Linked List . . . . .	14

Part I

# LeetCode Top Interview Questions



# Chapter 1

## Easy

Link: [LeetCode Top Interview Questions: Easy section](#).

### 1.1 Arrays

Link: [Arrays](#)

#### 1.1.1 26. Remove Duplicates from Sorted Array

[Link to question](#), [Link to submission](#)

**Concepts** Two pointer

#### **Algorithm description**

- Maintain a read pointer and a write pointer, both starting from zero.
- Advance the write pointer until you see a new value or reach end of array.
- Write value at write location into read location.

- Return read.

### 1.1.2 122. Best Time to Buy and Sell Stock II

[Link to question](#), [Link to submission](#)

**Concepts** Greedy

#### Algorithm description

- Construct a consecutive elements difference array
- Return sum of all positive elements in difference array

### 1.1.3 189. Rotate Array

[Link to question](#), [Link to submission approach 1](#), [Link to submission approach 2](#)

**Concepts** Cyclic replacements, Implementation

#### Approach 1 description

- Maintain a visited array and a pointer initialized to 0
- while pointer + k is not visited, replace arr[pointer + k] with arr[pointer]. Update pointer to pointer + k. Set pointer + k to visited, increment a numberOfChanges variable.
- Increment pointer by 1
- Keep doing this while numberOfChanges less than size of array.



**Approach 2 description**

- Reverse the entire array
- Reverse from start to start + k
- Reverse from start + k to end

**1.1.4 217. Contains Duplicate**

[Link to question](#), [Link to submission](#)

**Concepts** Hash Table, Set

**Algorithm description**

- Initialize a Set
- For an element in array, if element in Set, return true
- else add element to Set
- If out of loop, return False

**1.1.5 136. Single Number**

[Link to question](#), [Link to submission](#)

**Concepts** Bit Manipulation, XOR

**Algorithm description**

- Initialize an answer variable to 0
- For every element, XOR it to answer. Elements appearing twice get XOR'd out to zero

- Return answer

### 1.1.6 350. Intersection of Two Arrays II

[Link to question](#), [Link to submission approach 1](#), [Link to submission approach 2](#)

**Concepts** Hash Table, Two Pointers

#### Approach 1 description

- Form an element:frequency mapping using map for smaller array (to save space)
- Traverse bigger array
- If frequency of element less than 0, add to answer. Decrement frequency

#### Approach 2 description

- If arrays are sorted, use two pointers p1 and p2
- If  $\text{nums1}[p1] == \text{nums2}[p2]$ , add to answer and increment both
- Else if  $\text{nums1}[p1]$  is smaller, increment p1. Else increment p2
- Keep doing until reach end of either array

### 1.1.7 66. Plus One

[Link to question](#), [Link to submission](#)

**Concepts** Array

**Algorithm description**

- Initialize a carry variable to 1
- Traverse array from the end.  
 $\text{digit}[i] = \text{carry} + \text{digit} \bmod 10$ ,  $\text{carry} = \text{carry} + \text{digit} \text{ div } 10$
- Finally, if carry is not zero, insert carry at start of array

**1.1.8 283. Move Zeroes**

[Link to question](#), [Link to submission](#)

**Concepts** Two Pointers

**Algorithm description**

- Maintain a read and a write pointer, both initialized to 0
- if read end has zero, increment read end
- else, copy read end to write end and increment both
- After read end reaches end, set all numbers from write end to end as 0

**1.1.9 1. Two Sum**

[Link to question](#), [Link to submission approach 1](#), [Link to submission approach 2](#)

**Concepts** Hash Table, Two Pointer

**Approach 1 description**

- Create an element:indices mapping
- Sort the array
- Use two pointers to search for a particular sum
- Once you find the sum, pop index from left pointer, and pop index from right pointer
- Return indices

**Approach 2 description**

- Create a hashmap of int, int
- Iterate the array with i as looping variable
- If element in hashmap, return (hashmap[element], i)
- Else insert hashmap[target - element] = i

**1.1.10 36. Valid Sudoku**

[Link to question](#), [Link to submission](#)

**Concepts** Hash Table, Set

**Algorithm description**

- Create sets to hold numbers for each row, col and square.
- Traverse the sudoku
- If a number is already in the row, col, square, return False
- Else, come out of loop and return true

### 1.1.11 48. Rotate Image

[Link to question](#), [Link to submission](#)

**Concepts** Array, Circular Permutation

#### Algorithm description

- Do a counterclockwise circular permutation as mentioned in solution
- Pure implementation problem. No algorithmic skill.

## 1.2 Strings

Link: [Strings](#)

### 1.2.1 344. Reverse String

[Link to question](#), [Link to submission](#)

**Concepts** Two Pointers

#### Algorithm description

- Set a left pointer to start of string, right pointer to end
- Swap left and right. Increment left, decrement right
- Do while l less than r

### 1.2.2 7. Reverse Integer

[Link to question](#), [Link to submission](#)

**Concepts** Two Pointers

#### Algorithm description

- Reverse the integer by converting to a string
- Store result in long
- If stored result is outside integer limits, return 0
- Else return the reversed number

### 1.2.3 387. First Unique Character in a String

[Link to question](#), [Link to submission](#)

**Concepts** Hash Map

#### Algorithm description

- Construct element frequency mapping
- Traverse the string from the start, if frequency of a char is 1, return index
- If reach end of string, return -1

### 1.2.4 242. Valid Anagram

[Link to question](#), [Link to submission](#)

**Concepts** Hash Map, Counting Sort

#### Algorithm description

- Traverse through s1, incrementing frequency counts
- Traverse through s2, decrementing frequency counts
- If all counts are zero, return true. Else false.

### 1.2.5 125. Valid Palindrome

[Link to question](#), [Link to submission](#)

**Concepts** Two Pointers

**Algorithm description**

- Maintain a left and a right pointer
- Before comparing the two, ensure left and right both are pointing to an alphanumeric character

**1.2.6 28. Implement strStr()**

[Link to question](#), [Link to Approach 1](#), [Link to Approach 2](#)

**Concepts** Two Pointers, Rabin-Karp Algorithm, Rolling Hash

**Approach 1 description**

- Traverse haystack until you find a character matching with first character of needle
- Once match is found, keep checking for further characters until either there's a mismatch or you reach end of arrays
- Return index accordingly

**Approach 2 description - Rabin-Karp**

- Hash the needle using a hash function that is easy to be "rolled", that is it is easy to compute hash for next window if hash for previous window is known
- Traverse the haystack using window of length `needle.length()`. Hash the window and compare with needle hash. If matched, return the index of start of window
- See implementation carefully, very interesting. Also see [LeetCode solution article](#).



**1.2.7 38. Count and Say**

[Link to question](#), [Link to submission](#)

**Concepts** Recursion, Two Pointers

**Algorithm description**

- Base case:  $n = 1$ , return "1"
- Get the answer for  $n-1$
- Traverse through answer of  $n-1$
- For each consecutive list of same elements, add the count, followed by the element
- Return answer

**1.2.8 14. Longest Common Prefix**

[Link to question](#), [Link to submission](#)

**Concepts** Implementation

**Algorithm description**

- Initialize answer string to ""
- Find length of smallest string
- For  $i$  from 0 to  $\text{min length} - 1$
- Traverse through all the characters at  $i$ th positions
- If different, return answer
- If same, add character to answer

## 1.3 Linked Lists

Link: [Linked Lists](#)

### 1.3.1 237. Delete Node in a Linked List

[Link to question](#), [Link to submission](#)

**Concepts** Trick

#### Algorithm description

- Copy value of next node into current node
- Set next ptr of current node to next ptr of next node

### 1.3.2 19. Remove Nth Node From End of List

[Link to question](#), [Link to submission](#)

**Concepts** Two Pointer

#### Algorithm description

- To do it in one pass, let a forward pointer advance n steps
- Then, start forwarding a slow pointer as well as the forward pointer one at a time until forward reaches the end
- delete the slow pointer node

### 1.3.3 206. Reverse Linked List

[Link to question](#), [Link to iterative approach](#), [Link to recursive approach](#)

**Concepts**   Implementation

**Approach 1 description**

- Initialize a `prev = NULL`, and a `curr = head`
- While `head` is not `NULL`, do a cyclic swap between `curr.next`, `prev`, and `curr`.
- Return `prev`

**Approach 2 description**

- If `head` is `NULL` or `head.next` is `NULL` return `head`
- `l = reversed list for head.next`
- `head.next.next = head`, `head.next = NULL`. Return `l`