

LeetCode Solutions

Program Creek

Version 0.0

Contents

1	Rotate Array in Java	7
2	Evaluate Reverse Polish Notation	9
3	Solution of Longest Palindromic Substring in Java	11
4	Solution Word Break	15
5	Word Break II <small>Shared by Ravit</small>	18
6	Word Ladder	20
7	Median of Two Sorted Arrays Java	23
8	Regular Expression Matching in Java	25
9	Merge Intervals	27
10	Insert Interval	29
11	Two Sum	31
12	Two Sum II Input array is sorted	32
13	Two Sum III Data structure design	33
14	3Sum	34
15	4Sum	36
16	3Sum Closest	38
17	String to Integer (atoi)	39
18	Merge Sorted Array	40
19	Valid Parentheses	42
20	Implement strStr()	43

21	Set Matrix Zeroes	44
22	Search Insert Position	46
23	Longest Consecutive Sequence Java	47
24	Valid Palindrome	49
25	Spiral Matrix	52
26	Search a 2D Matrix	55
27	Rotate Image	56
28	Triangle	58
29	Distinct Subsequences Total	60
30	Maximum Subarray	62
	Shared by Ravit	
31	Maximum Product Subarray	63
32	Remove Duplicates from Sorted Array	64
33	Remove Duplicates from Sorted Array II	67
34	Longest Substring Without Repeating Characters	69
35	Longest Substring Which Contains 2 Unique Characters	71
36	Palindrome Partitioning	73
37	Reverse Words in a String	75
38	Find Minimum in Rotated Sorted Array	76
39	Find Minimum in Rotated Sorted Array II	77
40	Find Peak Element	78
41	Min Stack	79
42	Majority Element	80
43	Combination Sum	82
44	Best Time to Buy and Sell Stock	83
45	Best Time to Buy and Sell Stock II	84

46	Best Time to Buy and Sell Stock III	85
47	Best Time to Buy and Sell Stock IV	86
48	Longest Common Prefix	88
49	Largest Number	89
50	Combinations	90
51	Compare Version Numbers	92
52	Gas Station	93
53	Candy	95
54	Jump Game	96
55	Pascal's Triangle	97
56	Container With Most Water	98
57	Count and Say	99
58	Repeated DNA Sequences	100
59	Add Two Numbers	101
60	Reorder List	105
61	Linked List Cycle	109
62	Copy List with Random Pointer	111
63	Merge Two Sorted Lists	114
64	Merge k Sorted Lists	116
65	Remove Duplicates from Sorted List	117
66	Partition List	119
67	LRU Cache	121
68	Intersection of Two Linked Lists	124
69	Java PriorityQueue Class Example	125
70	Solution for Binary Tree Preorder Traversal in Java	127

71	Solution of Binary Tree Inorder Traversal in Java	128
72	Solution of Iterative Binary Tree Postorder Traversal in Java	130
73	Validate Binary Search Tree	131
74	Flatten Binary Tree to Linked List	133
75	Path Sum	134
76	Construct Binary Tree from Inorder and Postorder Traversal	136
77	Convert Sorted Array to Binary Search Tree	137
78	Convert Sorted List to Binary Search Tree	138
79	Minimum Depth of Binary Tree	140
80	Binary Tree Maximum Path Sum	142
81	Balanced Binary Tree	143
82	Symmetric Tree	145
83	Clone Graph Java	146
84	How Developers Sort in Java?	149
85	Solution Merge Sort LinkedList in Java	151
86	Quicksort Array in Java	154
87	Solution Sort a linked list using insertion sort in Java	156
88	Maximum Gap	158
89	Iteration vs. Recursion in Java	160
90	Edit Distance in Java	163
91	Single Number	165
92	Single Number II	166
93	Twitter Codility Problem Max Binary Gap	166
94	Number of 1 Bits	167
95	Reverse Bits	168

96	Permutations	169
97	Permutations II	171
98	Permutation Sequence	173
99	Generate Parentheses	175
100	Reverse Integer	176
101	Palindrome Number	178
102	Pow(x, n)	179

1 Rotate Array in Java

You may have been using Java for a while. Do you think a simple Java array question can be a challenge? Let's use the following problem to test.

Problem: Rotate an array of n elements to the right by k steps. For example, with $n = 7$ and $k = 3$, the array $[1,2,3,4,5,6,7]$ is rotated to $[5,6,7,1,2,3,4]$.

How many different ways do you know to solve this problem?

1.1 Solution 1 - Intermediate Array

In a straightforward way, we can create a new array and then copy elements to the new array. Then change the original array by using `System.arraycopy()`.

```
public void rotate(int[] nums, int k) {
    if(k > nums.length)
        k=k%nums.length;

    int[] result = new int[nums.length];

    for(int i=0; i < k; i++){
        result[i] = nums[nums.length-k+i];
    }

    int j=0;
    for(int i=k; i<nums.length; i++){
        result[i] = nums[j];
        j++;
    }

    System.arraycopy( result, 0, nums, 0, nums.length );
}
```

Space is $O(n)$ and time is $O(n)$.

1.2 Solution 2 - Bubble Rotate

Can we do this in $O(1)$ space?

This solution is like a bubble sort.

```
public static void rotate(int[] arr, int order) {
    if (arr == null || order < 0) {
        throw new IllegalArgumentException("Illegal argument!");
    }
}
```

1 Rotate Array in Java

```
}

for (int i = 0; i < order; i++) {
    for (int j = arr.length - 1; j > 0; j--) {
        int temp = arr[j];
        arr[j] = arr[j - 1];
        arr[j - 1] = temp;
    }
}
}
```

However, the time is $O(n*k)$.

1.3 Solution 3 - Reversal

Can we do this in $O(1)$ space and in $O(n)$ time? The following solution does.

Assuming we are given 1,2,3,4,5,6 and order 2. The basic idea is:

-
1. Divide the array two parts: 1,2,3,4 and 5, 6
 2. Rotate first part: 4,3,2,1,5,6
 3. Rotate second part: 4,3,2,1,6,5
 4. Rotate the whole array: 5,6,1,2,3,4
-

```
public static void rotate(int[] arr, int order) {
    order = order % arr.length;

    if (arr == null || order < 0) {
        throw new IllegalArgumentException("Illegal argument!");
    }

    //length of first part
    int a = arr.length - order;

    reverse(arr, 0, a-1);
    reverse(arr, a, arr.length-1);
    reverse(arr, 0, arr.length-1);
}

public static void reverse(int[] arr, int left, int right){
    if(arr == null || arr.length == 1)
        return;

    while(left < right){
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;
        left++;
        right--;
    }
}
```