461. Hamming Distance

[Description](https://leetcode.com/problems/hamming-distance/description/)[Hints](https://leetcode.com/problems/hamming-distance/hints/)[Submissions](https://leetcode.com/problems/hamming-distance/submissions/)[Discuss](https://leetcode.com/problems/hamming-distance/discuss/)[Solution](https://leetcode.com/problems/hamming-distance/solution/)

[Discuss](https://discuss.leetcode.com/category/590) [Pick One](https://leetcode.com/problems/random-one-question/)

The [Hamming distance](https://en.wikipedia.org/wiki/Hamming_distance) between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, calculate the Hamming distance.

**Note:**  
0 ≤ x, y < 231.

**Example:**

**Input:** x = 1, y = 4

**Output:** 2

**Explanation:**

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

[My C++ solution using bit manipulation](https://discuss.leetcode.com/topic/72236/my-c-solution-using-bit-manipulation)

**class** **Solution** {

**public**:

**int** **hammingDistance**(**int** x, **int** y) {

**int** dist = 0, n = x ^ y;

**while** (n) {

++dist;

n &= n - 1;

}

**return** dist;

}

};

657. Judge Route Circle

[Description](https://leetcode.com/problems/judge-route-circle/description/)[Hints](https://leetcode.com/problems/judge-route-circle/hints/)[Submissions](https://leetcode.com/problems/judge-route-circle/submissions/)[Discuss](https://leetcode.com/problems/judge-route-circle/discuss/)[Solution](https://leetcode.com/problems/judge-route-circle/solution/)

[Discuss](https://discuss.leetcode.com/category/1465) [Pick One](https://leetcode.com/problems/random-one-question/)

Initially, there is a Robot at position (0, 0). Given a sequence of its moves, judge if this robot makes a circle, which means it moves back to **the original place**.

The move sequence is represented by a string. And each move is represent by a character. The valid robot moves are R (Right), L (Left), U (Up) and D (down). The output should be true or false representing whether the robot makes a circle.

**Example 1:**

**Input:** "UD"

**Output:** true

**Example 2:**

**Input:** "LL"

**Output:** false

**Java**

public class Solution {

public boolean judgeCircle(String moves) {

int x = 0;

int y = 0;

for (char ch : moves.toCharArray()) {

if (ch == 'U') y++;

else if (ch == 'D') y--;

else if (ch == 'R') x++;

else if (ch == 'L') x--;

}

return x == 0 && y == 0;

}

}

617. Merge Two Binary Trees

[Description](https://leetcode.com/problems/merge-two-binary-trees/description/)[Hints](https://leetcode.com/problems/merge-two-binary-trees/hints/)[Submissions](https://leetcode.com/problems/merge-two-binary-trees/submissions/)[Discuss](https://leetcode.com/problems/merge-two-binary-trees/discuss/)[Solution](https://leetcode.com/problems/merge-two-binary-trees/solution/)

[Discuss](https://discuss.leetcode.com/category/782) [Pick One](https://leetcode.com/problems/random-one-question/)

Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.

**Example 1:**

**Input:**

Tree 1 Tree 2

1 2

/ \ / \

3 2 1 3

/ \ \

5 4 7

**Output:**

Merged tree:

3

/ \

4 5

/ \ \

5 4 7

**Note:** The merging process must start from the root nodes of both trees.

Approach #1 Using Recursion [Accepted]

We can traverse both the given trees in a preorder fashion. At every step, we check if the current node exists(isn't null) for both the trees. If so, we add the values in the current nodes of both the trees and update the value in the current node of the first tree to reflect this sum obtained. At every step, we also call the original function mergeTrees() with the left children and then with the right children of the current nodes of the two trees. If at any step, one of these children happens to be null, we return the child of the other tree(representing the corresponding child subtree) to be added as a child subtree to the calling parent node in the first tree. At the end, the first tree will represent the required resultant merged binary tree.

**Java**

*/\*\**

*\* Definition for a binary tree node.*

*\* public class TreeNode {*

*\* int val;*

*\* TreeNode left;*

*\* TreeNode right;*

*\* TreeNode(int x) { val = x; }*

*\* }*

*\*/*

**public** **class** **Solution** **{**

**public** TreeNode **mergeTrees(**TreeNode t1**,** TreeNode t2**)** **{**

**if** **(**t1 **==** **null)**

**return** t2**;**

**if** **(**t2 **==** **null)**

**return** t1**;**

t1**.**val **+=** t2**.**val**;**

t1**.**left **=** mergeTrees**(**t1**.**left**,** t2**.**left**);**

t1**.**right **=** mergeTrees**(**t1**.**right**,** t2**.**right**);**

**return** t1**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(m)O(m)O(m). A total of mmm nodes need to be traversed. Here, mmm represents the minimum number of nodes from the two given trees.
* Space complexity : O(m)O(m)O(m). The depth of the recursion tree can go upto mmm in the case of a skewed tree. In average case, depth will be O(logm)O(logm)O(logm).

Approach #2 Iterative Method [Accepted]

**Algorithm**

In the current approach, we again traverse the two trees, but this time we make use of a stackstackstack to do so instead of making use of recursion. Each entry in the stackstackstack strores data in the form [nodetree1,nodetree2][node\_{tree1}, node\_{tree2}][node​tree1​​,node​tree2​​]. Here, nodetree1node\_{tree1}node​tree1​​ and nodetree2node\_{tree2}node​tree2​​ are the nodes of the first tree and the second tree respectively.

We start off by pushing the root nodes of both the trees onto the stackstackstack. Then, at every step, we remove a node pair from the top of the stack. For every node pair removed, we add the values corresponding to the two nodes and update the value of the corresponding node in the first tree. Then, if the left child of the first tree exists, we push the left child(pair) of both the trees onto the stack. If the left child of the first tree doesn't exist, we append the left child(subtree) of the second tree to the current node of the first tree. We do the same for the right child pair as well.

If, at any step, both the current nodes are null, we continue with popping the next nodes from the stackstackstack.

**Java**

*/\*\**

*\* Definition for a binary tree node.*

*\* public class TreeNode {*

*\* int val;*

*\* TreeNode left;*

*\* TreeNode right;*

*\* TreeNode(int x) { val = x; }*

*\* }*

*\*/*

**public** **class** **Solution** **{**

**public** TreeNode **mergeTrees(**TreeNode t1**,** TreeNode t2**)** **{**

**if** **(**t1 **==** **null)**

**return** t2**;**

Stack **<** TreeNode**[]** **>** stack **=** **new** Stack **<** **>** **();**

stack**.**push**(new** TreeNode**[]** **{**t1**,** t2**});**

**while** **(!**stack**.**isEmpty**())** **{**

TreeNode**[]** t **=** stack**.**pop**();**

**if** **(**t**[**0**]** **==** **null** **||** t**[**1**]** **==** **null)** **{**

**continue;**

**}**

t**[**0**].**val **+=** t**[**1**].**val**;**

**if** **(**t**[**0**].**left **==** **null)** **{**

t**[**0**].**left **=** t**[**1**].**left**;**

**}** **else** **{**

stack**.**push**(new** TreeNode**[]** **{**t**[**0**].**left**,** t**[**1**].**left**});**

**}**

**if** **(**t**[**0**].**right **==** **null)** **{**

t**[**0**].**right **=** t**[**1**].**right**;**

**}** **else** **{**

stack**.**push**(new** TreeNode**[]** **{**t**[**0**].**right**,** t**[**1**].**right**});**

**}**

**}**

**return** t1**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n)O(n)O(n). We traverse over a total of nnn nodes. Here, nnn refers to the smaller of the number of nodes in the two trees.
* Space complexity : O(n)O(n)O(n). The depth of stack can grow upto nnn in case of a skewed tree.

561. Array Partition I

[Description](https://leetcode.com/problems/array-partition-i/description/)[Hints](https://leetcode.com/problems/array-partition-i/hints/)[Submissions](https://leetcode.com/problems/array-partition-i/submissions/)[Discuss](https://leetcode.com/problems/array-partition-i/discuss/)[Solution](https://leetcode.com/problems/array-partition-i/solution/)

[Discuss](https://discuss.leetcode.com/category/718) [Pick One](https://leetcode.com/problems/random-one-question/)

Given an array of **2n** integers, your task is to group these integers into **n** pairs of integer, say (a1, b1), (a2, b2), ..., (an, bn) which makes sum of min(ai, bi) for all i from 1 to n as large as possible.

**Example 1:**

**Input:** [1,4,3,2]

**Output:** 4

**Explanation:** n is 2, and the maximum sum of pairs is 4 = min(1, 2) + min(3, 4).

**Note:**

1. **n** is a positive integer, which is in the range of [1, 10000].
2. All the integers in the array will be in the range of [-10000, 10000].

SolutionApproach #1 Brute Force [Time Limit Exceeded] **Algorithm**

The simplest solution is to consider every possible set of pairings possible by using the elements of the numsnumsnums array. For generating all the possible pairings, we make use of a function permute(nums, current\_index). This function creates all the possible permutations of the elements of the given array.To do so, permute takes the index of the current element currentindexcurrent\_indexcurrent​i​​ndex as one of the arguments. Then, it swaps the current element with every other element in the array, lying towards its right, so as to generate a new ordering of the array elements. After the swapping has been done, it makes another call to permute but this time with the index of the next element in the array. While returning back, we reverse the swapping done in the current function call. Thus, when we reach the end of the array, a new ordering of the array's elements is generated. We consider the elements to be taken for the pairings such that the first element of every pair comes from the first half of the new array and the second element comes from the last half of the array. Thus, we sum up the minimum elements out of all these possible pairings and find out the maximum sum out of them.

**Java**

**public** **class** **Solution** **{**

**int** max\_sum **=** Integer**.**MIN\_VALUE**;**

**public** **int** **arrayPairSum(int[]** nums**)** **{**

permute**(**nums**,** 0**);**

**return** max\_sum**;**

**}**

**public** **void** **permute(int[]** nums**,** **int** l**)** **{**

**if** **(**l **==** nums**.**length **-** 1**)** **{**

**int** sum **=** 0**;**

**for** **(int** i **=** 0**;** i **<** nums**.**length **/** 2**;** i**++)** **{**

sum **+=** Math**.**min**(**nums**[**i**],** nums**[**nums**.**length **/** 2 **+** i**]);**

**}**

max\_sum **=** Math**.**max**(**max\_sum**,** sum**);**

**}**

**for** **(int** i **=** l**;** i **<** nums**.**length**;** i**++)** **{**

swap**(**nums**,** i**,** l**);**

permute**(**nums**,** l **+** 1**);**

swap**(**nums**,** i**,** l**);**

**}**

**}**

**public** **void** **swap(int[]** nums**,** **int** x**,** **int** y**)** **{**

**int** temp **=** nums**[**x**];**

nums**[**x**]** **=** nums**[**y**];**

nums**[**y**]** **=** temp**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n!)O(n!)O(n!). A total of n!n!n! permutations are possible for nnn elements in the array.
* Space complexity : O(1)O(1)O(1). Constant extra space is used.

Approach #2 Using Sorting [Accepted]

**Algorithm**

In order to understand this approach, let us look at the problem from a different perspective. We need to form the pairings of the array's elements such that the overall sum of the minimum out of such pairings is maximum. Thus, we can look at the operation of choosing the minimum out of the pairing, say (a,b)(a, b)(a,b) as incurring a loss of a−ba - ba−b(if a>ba> ba>b), in the maximum sum possible.

The total sum will now be maximum if the overall loss incurred from such pairings is minimized. This minimization of loss in every pairing is possible only if the numbers chosen for the pairings lie closer to each other than to the other elements of the array.

Taking this into consideration, we can sort the elements of the given array and form the pairings of the elements directly in the sorted order. This will lead to the pairings of elements with minimum difference between them leading to the maximization of the required sum.

**Java**

**public** **class** **Solution** **{**

**public** **int** **arrayPairSum(int[]** nums**)** **{**

Arrays**.**sort**(**nums**);**

**int** sum **=** 0**;**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i **+=** 2**)** **{**

sum **+=** nums**[**i**];**

**}**

**return** sum**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(nlog(n))O\big(nlog(n)\big)O(nlog(n)). Sorting takes O(nlog(n))O\big(nlog(n)\big)O(nlog(n)) time. We iterate over the array only once.
* Space complexity : O(1)O(1)O(1). Constant extra space is used.

Approach #3 Using Extra Array [Accepted]

**Algorithm**

This approach is somewhat related to the sorting approach. Since the range of elements in the given array is limited, we can make use of a hashmap arrarrarr, such that arr[i]arr[i]arr[i] stores the frequency of occurence of (i−10000)th(i-10000)^{th}(i−10000)​th​​ element. This subtraction is done so as to be able to map the numbers in the range −10000≤*i*≤−1 −10000≤i≤−1 onto the hashmap.

Thus, now instead of sorting the array's elements, we can directly traverse the hashmap in an ascending order. But, any element could also occur multiple times in the given array. We need to take this factor into account.

For this, consider an example: nums: [a, b, a, b, b, a]. The sorted order of this array will be nums\_sorted: [a, a, a, b, b, b]. (We aren't actually sorting the array in this approach, but the sorted array is taken just for demonstration). From the previous approach, we know that the required set of pairings is (a,a),(a,b),(b,b)(a,a), (a,b), (b,b)(a,a),(a,b),(b,b). Now, we can see that while choosing the minimum elements, aaa will be chosen twice and bbb will be chosen once only. This happens because the number of aaa's to be chosen has already been determined by the frequency of aaa, leaving the rest of the places to be filled by bbb. This is because, for the correct result we need to consider the elements in the ascending order. Thus, the lower number always gets priority to be added to the end result.

But, if the sorted elements take the form: nums\_sorted: [a, a, b, b, b, b], the correct pairing will be (a,a),(b,b),(b,b)(a,a), (b,b), (b,b)(a,a),(b,b),(b,b). Again, in this case the number of aaa's chosen is already predetermined, but since the number of aaa's is odd, it doesn't impact the choice of bbb in the final sum.

Thus, based on the above discussion, we traverse the hashmap arrarrarr. If the current element is occuring freqifreq\_ifreq​i​​ number of times, and one of the elements is left to be paired with other elements in the right region(considering a virtual sorted array), we consider the current element ⌈freqi2⌉\left \lceil\frac{freq\_i}{2}\right \rceil⌈​2​​freq​i​​​​⌉ number of times and the next element occuring in the array ⌊freqj2⌋\left \lfloor\frac{freq\_j}{2}\right \rfloor⌊​2​​freq​j​​​​⌋ number of times for the final sum. To propagate the impact of this left over chosen number, we make use of a flag ddd. This flag is set to 1 if there is a leftover element from the current set which will be considered one more time. The same extra element already considered is taken into account while choosing an element from the next set.

While traversing the hashmap, we determine the correct number of times each element needs to be considered as discussed above. Note that the flag ddd and the sumsumsum remains unchanged if the current element of the hashmap doesn't exist in the array.

Below code is inspired by [@fallcreek](https://leetcode.com/fallcreek)

**java**

**public** **class** **Solution** **{**

**public** **int** **arrayPairSum(int[]** nums**)** **{**

**int[]** arr **=** **new** **int[**20001**];**

**int** lim **=** 10000**;**

**for** **(int** num**:** nums**)**

arr**[**num **+** lim**]++;**

**int** d **=** 0**,** sum **=** 0**;**

**for** **(int** i **=** **-**10000**;** i **<=** 10000**;** i**++)** **{**

sum **+=** **(**arr**[**i **+** lim**]** **+** 1 **-** d**)** **/** 2 **\*** i**;**

d **=** **(**2 **+** arr**[**i **+** lim**]** **-** d**)** **%** 2**;**

**}**

**return** sum**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n)O(n)O(n). The whole hashmap arrarrarr of size nnn is traversed only once.
* Space complexity : O(n)O(n)O(n). A hashmap arrarrarr of size nnn is used.

339. Nested List Weight Sum

[Description](https://leetcode.com/problems/nested-list-weight-sum/description/)[Hints](https://leetcode.com/problems/nested-list-weight-sum/hints/)[Submissions](https://leetcode.com/problems/nested-list-weight-sum/submissions/)[Discuss](https://leetcode.com/problems/nested-list-weight-sum/discuss/)[Solution](https://leetcode.com/problems/nested-list-weight-sum/solution/)

[Discuss](https://discuss.leetcode.com/category/423) [Pick One](https://leetcode.com/problems/random-one-question/)

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

**Example 1:**  
Given the list [[1,1],2,[1,1]], return **10**. (four 1's at depth 2, one 2 at depth 1)

**Example 2:**  
Given the list [1,[4,[6]]], return **27**. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4\*2 + 6\*3 = 27)

Summary

This is a very simple recursion problem and is a nice introduction to Depth-first Search (DFS).

Solution

Depth-first Traversal [Accepted]

**Algorithm**

Because the input is nested, it is natural to think about the problem in a recursive way. We go through the list of nested integers one by one, keeping track of the current depth ddd. If a nested integer is an integer nnn, we calculate its sum as n×dn\times dn×d. If the nested integer is a list, we calculate the sum of this list recursively using the same process but with depth d+1d+1d+1.

**Java**

*/\*\**

*\* // This is the interface that allows for creating nested lists.*

*\* // You should not implement it, or speculate about its implementation*

*\* public interface NestedInteger {*

*\**

*\* // @return true if this NestedInteger holds a single integer,*

*\* // rather than a nested list.*

*\* public boolean isInteger();*

*\**

*\* // @return the single integer that this NestedInteger holds,*

*\* // if it holds a single integer*

*\* // Return null if this NestedInteger holds a nested list*

*\* public Integer getInteger();*

*\**

*\* // @return the nested list that this NestedInteger holds,*

*\* // if it holds a nested list*

*\* // Return null if this NestedInteger holds a single integer*

*\* public List<NestedInteger> getList();*

*\* }*

*\*/*

**public** **int** **depthSum(**List**<**NestedInteger**>** nestedList**)** **{**

**return** depthSum**(**nestedList**,** 1**);**

**}**

**public** **int** **depthSum(**List**<**NestedInteger**>** list**,** **int** depth**)** **{**

**int** sum **=** 0**;**

**for** **(**NestedInteger n **:** list**)** **{**

**if** **(**n**.**isInteger**())** **{**

sum **+=** n**.**getInteger**()** **\*** depth**;**

**}** **else** **{**

sum **+=** depthSum**(**n**.**getList**(),** depth **+** 1**);**

**}**

**}**

**return** sum**;**

**}**

**Complexity Analysis**

The algorithm takes O(N)O(N)O(N) time, where NNN is the total number of nested elements in the input list. For example, the list [ [[[[1]]]], 2 ] contains 444 nested lists and 222 nested integers (111 and 222), so N=6N=6N=6.

In terms of space, at most O(D)O(D)O(D) recursive calls are placed on the stack, where DDD is the maximum level of nesting in the input. For example, D=2D=2D=2 for the input [[1,1],2,[1,1]], and D=3D=3D=3 for the input [1,[4,[6]]].

### 476. Number Complement

[Description](https://leetcode.com/problems/number-complement/description/)[Hints](https://leetcode.com/problems/number-complement/hints/)[Submissions](https://leetcode.com/problems/number-complement/submissions/)[Discuss](https://leetcode.com/problems/number-complement/discuss/)[Solution](https://leetcode.com/problems/number-complement/solution/)

[Discuss](https://discuss.leetcode.com/category/608) [Pick One](https://leetcode.com/problems/random-one-question/)

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

**Note:**

1. The given integer is guaranteed to fit within the range of a 32-bit signed integer.
2. You could assume no leading zero bit in the integer’s binary representation.

**Example 1:**

**Input:** 5

**Output:** 2

**Explanation:** The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

**Example 2:**

**Input:** 1

**Output:** 0

**Explanation:** The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

Solution

class Solution(object):

def findComplement(self, num):

i = 1

while i <= num:

i = i << 1

return (i - 1) ^ num

### 500. Keyboard Row

[Description](https://leetcode.com/problems/keyboard-row/description/)[Hints](https://leetcode.com/problems/keyboard-row/hints/)[Submissions](https://leetcode.com/problems/keyboard-row/submissions/)[Discuss](https://leetcode.com/problems/keyboard-row/discuss/)[Solution](https://leetcode.com/problems/keyboard-row/solution/)

[Discuss](https://discuss.leetcode.com/category/649) [Pick One](https://leetcode.com/problems/random-one-question/)

Given a List of words, return the words that can be typed using letters of **alphabet** on only one row's of American keyboard like the image below.



**Example 1:**

**Input:** ["Hello", "Alaska", "Dad", "Peace"]

**Output:** ["Alaska", "Dad"]

**Note:**

1. You may use one character in the keyboard more than once.
2. You may assume the input string will only contain letters of alphabet.

[Easy understand solution in 7 lines for everyone](https://discuss.leetcode.com/topic/77759/easy-understand-solution-in-7-lines-for-everyone)

First of all,

**if** you are python user {

please upvote **if** **it** makes sense;

} **else** **if** you are C++/Java user {

please let **me** know **if** somewhere **is** **not** clear;

}

I have used set to check the word.  
I firstly make every line a set of letter.  
Then I check every word if this word set is the subset if any line set.

def findWords(self, **words**):

line1, line2, line3 = set('qwertyuiop'), set('asdfghjkl'), set('zxcvbnm')

ret = []

**for** word **in** **words**:

w = set(word.lower())

**if** w.issubset(line1) **or** w.issubset(line2) **or** w.issubset(line3):

ret.append(word)

return ret

359. Logger Rate Limiter

[Description](https://leetcode.com/problems/logger-rate-limiter/description/)[Hints](https://leetcode.com/problems/logger-rate-limiter/hints/)[Submissions](https://leetcode.com/problems/logger-rate-limiter/submissions/)[Discuss](https://leetcode.com/problems/logger-rate-limiter/discuss/)[Solution](https://leetcode.com/problems/logger-rate-limiter/solution/)

[Discuss](https://discuss.leetcode.com/category/443) [Pick One](https://leetcode.com/problems/random-one-question/)

Design a logger system that receive stream of messages along with its timestamps, each message should be printed if and only if it is **not printed in the last 10 seconds**.

Given a message and a timestamp (in seconds granularity), return true if the message should be printed in the given timestamp, otherwise returns false.

It is possible that several messages arrive roughly at the same time.

**Example:**

Logger logger = new Logger();

// logging string "foo" at timestamp 1

logger.shouldPrintMessage(1, "foo"); returns true;

// logging string "bar" at timestamp 2

logger.shouldPrintMessage(2,"bar"); returns true;

// logging string "foo" at timestamp 3

logger.shouldPrintMessage(3,"foo"); returns false;

// logging string "bar" at timestamp 8

logger.shouldPrintMessage(8,"bar"); returns false;

// logging string "foo" at timestamp 10

logger.shouldPrintMessage(10,"foo"); returns false;

// logging string "foo" at timestamp 11

logger.shouldPrintMessage(11,"foo"); returns true;

[Short C++/Java/Python, bit different](https://discuss.leetcode.com/topic/48359/short-c-java-python-bit-different)

Instead of logging print times, I store when it's ok for a message to be printed again. Should be slightly faster, because I don't always have to add or subtract (e.g., timestamp < log[message] + 10) but only do in the true case. Also, it leads to a shorter/simpler longest line of code. Finally, C++ has 0 as default, so I can just use ok[message].

**C++**

**class** **Logger** {

**public**:

map<string, **int**> ok;

**bool** **shouldPrintMessage**(**int** timestamp, string message) {

**if** (timestamp < ok[message])

**return** false;

ok[message] = timestamp + 10;

**return** true;

}

};

**Python**

**class** **Logger**(**object**):

**def** **\_\_init\_\_**(**self**):

**self**.ok = {}

**def** **shouldPrintMessage**(**self**, timestamp, message):

**if** timestamp < **self**.ok.get(message, 0):

**return** False

**self**.ok[message] = timestamp + 10

**return** True

**Java**

**public** **class** **Logger** {

**private** Map<**String**, Integer> ok = **new** HashMap<>();

**public** boolean shouldPrintMessage(int timestamp, **String** message) {

**if** (timestamp < ok.getOrDefault(message, 0))

**return** false;

ok.put(message, timestamp + 10);

**return** true;

}

}

557. Reverse Words in a String III

[Description](https://leetcode.com/problems/reverse-words-in-a-string-iii/description/)[Hints](https://leetcode.com/problems/reverse-words-in-a-string-iii/hints/)[Submissions](https://leetcode.com/problems/reverse-words-in-a-string-iii/submissions/)[Discuss](https://leetcode.com/problems/reverse-words-in-a-string-iii/discuss/)[Solution](https://leetcode.com/problems/reverse-words-in-a-string-iii/solution/)

[Discuss](https://discuss.leetcode.com/category/714) [Pick One](https://leetcode.com/problems/random-one-question/)

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Example 1:**

**Input:** "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

**Note:** In the string, each word is separated by single space and there will not be any extra space in the string.

Approach #1 Simple Solution[Accepted]

The first method is really simple. We simply split up the given string based on whitespaces and put the individual words in an array of strings. Then, we reverse each individual string and concatenate the result. We return the result after removing the additional whitespaces at the end.

**Java**

**public** **class** **Solution** **{**

**public** String **reverseWords(**String s**)** **{**

String words**[]** **=** s**.**split**(**" "**);**

StringBuilder res**=new** StringBuilder**();**

**for** **(**String word**:** words**)**

res**.**append**(new** StringBuffer**(**word**).**reverse**().**toString**()** **+** " "**);**

**return** res**.**toString**().**trim**();**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n)O(n)O(n). where nnn is the length of the string.
* Space complexity : O(n)O(n)O(n). resresres of size nnn is used.

Approach #2 Without using pre-defined split and reverse function [Accepted]

**Algorithm**

We can create our own split and reverse function. Split function splits the string based on the delimiter " "(space) and returns the array of words. Reverse function returns the string after reversing the characters.

**Java**

**public** **class** **Solution** **{**

**public** String **reverseWords(**String s**)** **{**

String words**[]** **=** split**(**s**);**

StringBuilder res**=new** StringBuilder**();**

**for** **(**String word**:** words**)**

res**.**append**(**reverse**(**word**)** **+** " "**);**

**return** res**.**toString**().**trim**();**

**}**

**public** String**[]** **split(**String s**)** **{**

ArrayList **<** String **>** words **=** **new** ArrayList **<** **>** **();**

StringBuilder word **=** **new** StringBuilder**();**

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

**if** **(**s**.**charAt**(**i**)** **==** ' '**)** **{**

words**.**add**(**word**.**toString**());**

word **=** **new** StringBuilder**();**

**}** **else**

word**.**append**(** s**.**charAt**(**i**));**

**}**

words**.**add**(**word**.**toString**());**

**return** words**.**toArray**(new** String**[**words**.**size**()]);**

**}**

**public** String **reverse(**String s**)** **{**

StringBuilder res**=new** StringBuilder**();**

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

res**.**insert**(**0**,**s**.**charAt**(**i**));**

**return** res**.**toString**();**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n)O(n)O(n). where nnn is the length of the string.
* Space complexity : O(n)O(n)O(n). resresres of size nnn is used.

Approach #3 Using StringBuilder and reverse method [Accepted]

**Algorithm**

Instead of using split method, we can use temporary string wordwordword to store the word. We simply append the characters to the wordwordword until ' ' character is not found. On getting ' ' we append the reverse of the wordwordword to the resultant string resultresultresult. Also after completion of loop , we still have to append the reversereversereverse of the wordwordword(last word) to the resultresultresult string.

Below code is inspired by [@ApolloX](http://leetcode.com/apolloX).

**Java**

**public** **class** **Solution** **{**

**public** String **reverseWords(**String input**)** **{**

**final** StringBuilder result **=** **new** StringBuilder**();**

**final** StringBuilder word **=** **new** StringBuilder**();**

**for** **(int** i **=** 0**;** i **<** input**.**length**();** i**++)** **{**

**if** **(**input**.**charAt**(**i**)** **!=** ' '**)** **{**

word**.**append**(**input**.**charAt**(**i**));**

**}** **else** **{**

result**.**append**(**word**.**reverse**());**

result**.**append**(**" "**);**

word**.**setLength**(**0**);**

**}**

**}**

result**.**append**(**word**.**reverse**());**

**return** result**.**toString**();**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n)O(n)O(n). Single loop upto nnn is there, where nnn is the length of the string.
* Space complexity : O(n)O(n)O(n). resultresultresult and wordwordword size will grow upto nnn.

344. Reverse String

[Description](https://leetcode.com/problems/reverse-string/description/)[Hints](https://leetcode.com/problems/reverse-string/hints/)[Submissions](https://leetcode.com/problems/reverse-string/submissions/)[Discuss](https://leetcode.com/problems/reverse-string/discuss/)[Solution](https://leetcode.com/problems/reverse-string/solution/)

[Discuss](https://discuss.leetcode.com/category/428) [Pick One](https://leetcode.com/problems/random-one-question/)

Write a function that takes a string as input and returns the string reversed.

**Example:**  
Given s = "hello", return "olleh".

[Python 3 solutions: Recursive, Classic, Pythonic](https://discuss.leetcode.com/topic/58719/python-3-solutions-recursive-classic-pythonic)

**class** **Solution**(**object**):

**def** **reverseString**(**self**, s):

l = len(s)

**if** l < 2:

**return** s

**return** **self**.reverseString(s[l/2:]) + **self**.reverseString(s[:l/2])

**class** **SolutionClassic**(**object**):

**def** **reverseString**(**self**, s):

r = list(s)

i, j = 0, len(r) - 1

**while** i < j:

r[i], r[j] = r[j], r[i]

i += 1

j -= 1

**return** "".join(r)

**class** **SolutionPythonic**(**object**):

**def** **reverseString**(**self**, s):

**return** s[::-1]

669. Trim a Binary Search Tree

[Description](https://leetcode.com/problems/trim-a-binary-search-tree/description/)[Hints](https://leetcode.com/problems/trim-a-binary-search-tree/hints/)[Submissions](https://leetcode.com/problems/trim-a-binary-search-tree/submissions/)[Discuss](https://leetcode.com/problems/trim-a-binary-search-tree/discuss/)[Solution](https://leetcode.com/problems/trim-a-binary-search-tree/solution/)

[Discuss](https://discuss.leetcode.com/category/1496) [Pick One](https://leetcode.com/problems/random-one-question/)

Given a binary search tree and the lowest and highest boundaries as L and R, trim the tree so that all its elements lies in [L, R] (R >= L). You might need to change the root of the tree, so the result should return the new root of the trimmed binary search tree.

**Example 1:**

**Input:**

1

/ \

0 2

L = 1

R = 2

**Output:**

1

\

2

**Example 2:**

**Input:**

3

/ \

0 4

\

2

/

1

L = 1

R = 3

**Output:**

3

/

2

/

1

Approach #1: Recursion [Accepted]

**Intuition**

Let trim(node) be the desired answer for the subtree at that node. We can construct the answer recursively.

**Algorithm**

When node.val > R node.val > R , we know that the trimmed binary tree must occur to the left of the node. Similarly, when node.val < L node.val < L , the trimmed binary tree occurs to the right of the node. Otherwise, we will trim both sides of the tree.

**Java**

**class** **Solution** **{**

**public** TreeNode **trimBST(**TreeNode root**,** **int** L**,** **int** R**)** **{**

**if** **(**root **==** **null)** **return** root**;**

**if** **(**root**.**val **>** R**)** **return** trimBST**(**root**.**left**,** L**,** R**);**

**if** **(**root**.**val **<** L**)** **return** trimBST**(**root**.**right**,** L**,** R**);**

root**.**left **=** trimBST**(**root**.**left**,** L**,** R**);**

root**.**right **=** trimBST**(**root**.**right**,** L**,** R**);**

**return** root**;**

**}**

**}**

**Python**

**class** **Solution**(object):

**def** **trimBST**(self, root, L, R):

**def** **trim**(node):

**if** **not** node:

**return** None

**elif** node**.**val **>** R:

**return** trim(node**.**left)

**elif** node**.**val **<** L:

**return** trim(node**.**right)

**else**:

node**.**left **=** trim(node**.**left)

node**.**right **=** trim(node**.**right)

**return** node

**return** trim(root)

**Complexity Analysis**

* Time Complexity: O(N)O(N)O(N), where NNN is the total number of nodes in the given tree. We visit each node at most once.
* Space Complexity: O(N)O(N)O(N). Even though we don't explicitly use any additional memory, the call stack of our recursion could be as large as the number of nodes in the worst case.

### 575. Distribute Candies

[Description](https://leetcode.com/problems/distribute-candies/description/)[Hints](https://leetcode.com/problems/distribute-candies/hints/)[Submissions](https://leetcode.com/problems/distribute-candies/submissions/)[Discuss](https://leetcode.com/problems/distribute-candies/discuss/)[Solution](https://leetcode.com/problems/distribute-candies/solution/)

[Discuss](https://discuss.leetcode.com/category/734) [Pick One](https://leetcode.com/problems/random-one-question/)

Given an integer array with **even** length, where different numbers in this array represent different **kinds** of candies. Each number means one candy of the corresponding kind. You need to distribute these candies **equally** in number to brother and sister. Return the maximum number of **kinds** of candies the sister could gain.

**Example 1:**

**Input:** candies = [1,1,2,2,3,3]

**Output:** 3

**Explanation:**

There are three different kinds of candies (1, 2 and 3), and two candies for each kind.

Optimal distribution: The sister has candies [1,2,3] and the brother has candies [1,2,3], too.

The sister has three different kinds of candies.

**Example 2:**

**Input:** candies = [1,1,2,3]

**Output:** 2

**Explanation:** For example, the sister has candies [2,3] and the brother has candies [1,1].

The sister has two different kinds of candies, the brother has only one kind of candies.

**Note:**

1. The length of the given array is in range [2, 10,000], and will be even.
2. The number in given array is in range [-100,000, 100,000].

Approach #1 Brute Force [Time Limit Exceeded]

**Algorithm**

The brute force approach is really simple. We can generate all the permutations of the given numsnumsnums array representing the candies and determine the number of unique elements in the first half of the generated array.

In order to determine the number of unique elements in the first half of the array, we put all the required elements in a set and count the number of elements in the set. We count such unique elements in the first half of the generated arrays for all the permutations possible and return the size of the largest set.

**Java**

**public** **class** **Solution** **{**

**int** max\_kind **=** 0**;**

**public** **int** **distributeCandies(int[]** nums**)** **{**

permute**(**nums**,** 0**);**

**return** max\_kind**;**

**}**

**public** **void** **permute(int[]** nums**,** **int** l**)** **{**

**if** **(**l **==** nums**.**length **-** 1**)** **{**

HashSet **<** Integer **>** set **=** **new** HashSet **<** **>** **();**

**for** **(int** i **=** 0**;** i **<** nums**.**length **/** 2**;** i**++)** **{**

set**.**add**(**nums**[**i**]);**

**}**

max\_kind **=** Math**.**max**(**max\_kind**,** set**.**size**());**

**}**

**for** **(int** i **=** l**;** i **<** nums**.**length**;** i**++)** **{**

swap**(**nums**,** i**,** l**);**

permute**(**nums**,** l **+** 1**);**

swap**(**nums**,** i**,** l**);**

**}**

**}**

**public** **void** **swap(int[]** nums**,** **int** x**,** **int** y**)** **{**

**int** temp **=** nums**[**x**];**

nums**[**x**]** **=** nums**[**y**];**

nums**[**y**]** **=** temp**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n!)O(n!)O(n!). A total of n!n!n! permutations are possible for numsnumsnums array of size nnn.
* Space complexity : O(n)O(n)O(n). The depth of the recursion tree can go upto nnn.

Approach #2 Better Brute Force [Time Limit Exceeded]:

**Algorithm**

Before looking into the idea behind this approach, firstly we need to observe one point. The maximum no. of unique candies which the girl can obtain could be atmost n/2n/2n/2, where nnn refers to the number of candies. Further, in case the number of unique candies are below n/2n/2n/2, to maximize the number of unique candies that the girl will obtain, we'll assign all the unique candies to the girl. Thus, in such a case, the number of unique candies the girl gets is equal to the total number of unique candies in the given candiescandiescandies array.

Now, let's look at the idea behind this approach. We need to find the total number of unique candies in the given candiescandiescandies array. One way to find the number of unique candies is to traverse over the given candiescandiescandies array. Whenever we encounter an element, say candies[j]candies[j]candies[j], we can mark all the elements which are the same as candies[j]candies[j]candies[j] as invalid and increment the count of unique elements by 1.

Thus, we need to do such markings for all the elements of candiescandiescandies array. At the end, countcountcount gives the required number of unique candies that can be given to the girl. Further, the value to be returned is given by: min(n2,count)\text{min}(\frac{n}{2}, count)min(​2​​n​​,count). Instead of finding the min\text{min}min, we can stop the traversal over the given candiescandiescandies array as soon as the countcountcount exceeds n2\frac{n}{2}​2​​n​​.

**Java**

**public** **class** **Solution** **{**

**public** **int** **distributeCandies(int[]** candies**)** **{**

**int** count **=** 0**;**

**for** **(int** i **=** 0**;** i **<** candies**.**length **&&** count **<** candies**.**length **/** 2**;** i**++)** **{**

**if** **(**candies**[**i**]** **!=** Integer**.**MIN\_VALUE**)** **{**

count**++;**

**for** **(int** j **=** i **+** 1**;** j **<** candies**.**length**;** j**++)** **{**

**if** **(**candies**[**j**]** **==** candies**[**i**])**

candies**[**j**]** **=** Integer**.**MIN\_VALUE**;**

**}**

**}**

**}**

**return** count**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n2)O(n^2)O(n​2​​). We traverse over all the elements of candiescandiescandies for every new element found. In the worst case, we do so for every element of candiescandiescandies array. nnn refers to the size of candiescandiescandies array.
* Space complexity : O(1)O(1)O(1). Constant space is used.

Approach #3 Using sorting[Accepted]

**Algorithm**

We can sort the given candiescandiescandies array and find out the elements which are unique by comparing the adjacent elements of the sorted array. For every new element found(which isn't the same as the previous element), we need to update the countcountcount. At the end, we can return the required result as min(n/2,count)\text{min}(n/2, count)min(n/2,count), as discussed in the previous approach.

**Java**

**public** **class** **Solution** **{**

**public** **int** **distributeCandies(int[]** candies**)** **{**

Arrays**.**sort**(**candies**);**

**int** count **=** 1**;**

**for** **(int** i **=** 1**;** i **<** candies**.**length **&&** count **<** candies**.**length **/** 2**;** i**++)**

**if** **(**candies**[**i**]** **>** candies**[**i **-** 1**])**

count**++;**

**return** count**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(nlogn)O(nlogn)O(nlogn). Sorting takes O(nlogn)O(nlogn)O(nlogn) time.
* Space complexity : O(1)O(1)O(1). Constant space is used.

Approach #4 Using set [Accepted]

**Algorithm**

Another way to find the number of unique elements is to traverse over all the elements of the given candiescandiescandies array and keep on putting the elements in a set. By the property of a set, it will contain only unique elements. At the end, we can count the number of elements in the set, given by, say countcountcount. The value to be returned will again be given by min(count,n/2)\text{min}(count, n/2)min(count,n/2), as discussed in previous approaches. Here, nnn refers to the size of the candiescandiescandies array.

**Java**

**public** **class** **Solution** **{**

**public** **int** **distributeCandies(int[]** candies**)** **{**

HashSet **<** Integer **>** set **=** **new** HashSet **<** **>** **();**

**for** **(int** candy**:** candies**)** **{**

set**.**add**(**candy**);**

**}**

**return** Math**.**min**(**set**.**size**(),** candies**.**length **/** 2**);**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(n)O(n)O(n). The entire candiescandiescandies array is traversed only once. Here, nnn refers to the size of candiescandiescandies array.
* Space complexity : O(n)O(n)O(n). setsetset will be of size nnn in the worst case.

346. Moving Average from Data Stream

[Description](https://leetcode.com/problems/moving-average-from-data-stream/description/)[Hints](https://leetcode.com/problems/moving-average-from-data-stream/hints/)[Submissions](https://leetcode.com/problems/moving-average-from-data-stream/submissions/)[Discuss](https://leetcode.com/problems/moving-average-from-data-stream/discuss/)[Solution](https://leetcode.com/problems/moving-average-from-data-stream/solution/)

[Discuss](https://discuss.leetcode.com/category/430) [Pick One](https://leetcode.com/problems/random-one-question/)

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

For example,

MovingAverage m = new MovingAverage(3);

m.next(1) = 1

m.next(10) = (1 + 10) / 2

m.next(3) = (1 + 10 + 3) / 3

m.next(5) = (10 + 3 + 5) / 3

[Simple Python solution based on Circular Array - real O(1) time next()](https://discuss.leetcode.com/topic/49382/simple-python-solution-based-on-circular-array-real-o-1-time-next)

**class** **MovingAverage**(**object**):

**def** **\_\_init\_\_**(**self**, size):

**self**.vect, **self**.sums, **self**.idx, **self**.size = [0] \* size, 0, 0, size

**def** **next**(**self**, val):

**self**.idx += 1

**self**.sums -= **self**.vect[**self**.idx % **self**.size]

**self**.vect[**self**.idx % **self**.size] = val

**self**.sums += val

**return** **self**.sums / float(min(**self**.idx, **self**.size))

My solution requires real O(1) time for next() operation as it is not needed to compute the sum every time. We just subtract the element that is exiting from the sliding window, and we're done. We need also to substitute that element with the new one.

[Java O(1) time solution.](https://discuss.leetcode.com/topic/44108/java-o-1-time-solution)

The idea is to keep the sum so far and update the sum just by replacing the oldest number with the new entry.

**public** **class** **MovingAverage** {

**private** **int** [] window;

**private** **int** n, insert;

**private** **long** **sum**;

/\*\* Initialize your data structure here. \*/

**public** MovingAverage(**int** size) {

window = **new** **int**[size];

insert = 0;

**sum** = 0;

}

**public** **double** next(**int** val) {

**if** (n < window.length) n++;

**sum** -= window[insert];

**sum** += val;

window[insert] = val;

insert = (insert + 1) % window.length;

**return** (**double**)**sum** / n;

}

}

### 566. Reshape the Matrix

[Description](https://leetcode.com/problems/reshape-the-matrix/description/)[Hints](https://leetcode.com/problems/reshape-the-matrix/hints/)[Submissions](https://leetcode.com/problems/reshape-the-matrix/submissions/)[Discuss](https://leetcode.com/problems/reshape-the-matrix/discuss/)[Solution](https://leetcode.com/problems/reshape-the-matrix/solution/)

[Discuss](https://discuss.leetcode.com/category/724) [Pick One](https://leetcode.com/problems/random-one-question/)

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.

You're given a matrix represented by a two-dimensional array, and two **positive** integers **r** and **c** representing the **row** number and **column** number of the wanted reshaped matrix, respectively.

The reshaped matrix need to be filled with all the elements of the original matrix in the same **row-traversing** order as they were.

If the 'reshape' operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

**Example 1:**

**Input:**

nums =

[[1,2],

[3,4]]

r = 1, c = 4

**Output:**

[[1,2,3,4]]

**Explanation:**  
The **row-traversing** of nums is [1,2,3,4]. The new reshaped matrix is a 1 \* 4 matrix, fill it row by row by using the previous list.

**Example 2:**

**Input:**

nums =

[[1,2],

[3,4]]

r = 2, c = 4

**Output:**

[[1,2],

[3,4]]

**Explanation:**  
There is no way to reshape a 2 \* 2 matrix to a 2 \* 4 matrix. So output the original matrix.

**Note:**

1. The height and width of the given matrix is in range [1, 100].
2. The given r and c are all positive.

Approach #1 Using queue [Accepted]

**Algorithm**

The simplest method is to extract all the elements of the given matrix by reading the elements in a row-wise fashion. In this implementation, we use a queue to put the extracted elements. Then, we can take out the elements of the queue formed in a serial order and arrange the elements in the resultant required matrix in a row-by-row order again.

The formation of the resultant matrix won't be possible if the number of elements in the original matrix isn't equal to the number of elements in the resultant matrix.

**Java**

**public** **class** **Solution** **{**

**public** **int[][]** **matrixReshape(int[][]** nums**,** **int** r**,** **int** c**)** **{**

**int[][]** res **=** **new** **int[**r**][**c**];**

**if** **(**nums**.**length **==** 0 **||** r **\*** c **!=** nums**.**length **\*** nums**[**0**].**length**)**

**return** nums**;**

**int** count **=** 0**;**

Queue **<** Integer **>** queue **=** **new** LinkedList **<** **>** **();**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**for** **(int** j **=** 0**;** j **<** nums**[**0**].**length**;** j**++)** **{**

queue**.**add**(**nums**[**i**][**j**]);**

**}**

**}**

**for** **(int** i **=** 0**;** i **<** r**;** i**++)** **{**

**for** **(int** j **=** 0**;** j **<** c**;** j**++)** **{**

res**[**i**][**j**]** **=** queue**.**remove**();**

**}**

**}**

**return** res**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(m∗n)O(m\*n)O(m∗n). We traverse over m∗nm\*nm∗n elements twice. Here, mmm and nnn refer to the number of rows and columns of the given matrix respectively.
* Space complexity : O(m∗n)O(m\*n)O(m∗n). The queue formed will be of size m∗nm\*nm∗n.

Approach #2 Without using extra Space [Accepted]

**Algorithm**

Instead of unnecessarily using the queue as in the brute force approach, we can keep putting the numbers in the resultant matrix directly while iterating over the given matrix in a row-by-row order. While putting the numbers in the resultant array, we fix a particular row and keep on incrementing the column numbers only till we reach the end of the required columns indicated by ccc. At this moment, we update the row index by incrementing it and reset the column index to start from 0 again. Thus, we can save the space consumed by the queue for storing the data that just needs to be copied into a new array.

**Java**

**public** **class** **Solution** **{**

**public** **int[][]** **matrixReshape(int[][]** nums**,** **int** r**,** **int** c**)** **{**

**int[][]** res **=** **new** **int[**r**][**c**];**

**if** **(**nums**.**length **==** 0 **||** r **\*** c **!=** nums**.**length **\*** nums**[**0**].**length**)**

**return** nums**;**

**int** rows **=** 0**,** cols **=** 0**;**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**for** **(int** j **=** 0**;** j **<** nums**[**0**].**length**;** j**++)** **{**

res**[**rows**][**cols**]** **=** nums**[**i**][**j**];**

cols**++;**

**if** **(**cols **==** c**)** **{**

rows**++;**

cols **=** 0**;**

**}**

**}**

**}**

**return** res**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(m∗n)O(m\*n)O(m∗n). We traverse the entire matrix of size m∗nm\*nm∗n once only. Here, mmm and nnn refers to the number of rows and columns in the given matrix.
* Space complexity : O(m∗n)O(m\*n)O(m∗n). The resultant matrix of size m∗nm\*nm∗n is used.

Approach #3 Using division and modulus [Accepted]

**Algorithm**

In the last approach, we needed to keep a track of when we reached the end of columns for the resultant matrix and needed to update the current row and column number for putting the extracted elements by checking the current indices every time. Instead of doing these limit checks at every step, we can make use of maths to help ease the situation.

The idea behind this approach is as follows. Do you know how a 2-D array is stored in the main memory(which is 1-D in nature)? It is internally represented as a 1-D array only. The element nums[i][j]nums[i][j]nums[i][j] of numsnumsnums array is represented in the form of a one dimensional array by using the index in the form: nums[n∗i+j]nums[n\*i + j]nums[n∗i+j], where mmm is the number of columns in the given matrix. Looking at the same in the reverse order, while putting the elements in the elements in the resultant matrix, we can make use of a countcountcount variable which gets incremented for every element traversed as if we are putting the elements in a 1-D resultant array. But, to convert the countcountcount back into 2-D matrix indices with a column count of ccc, we can obtain the indices as res[count/c][count%c]res[count/c][count\%c]res[count/c][count%c] where count/ccount/ccount/c is the row number and count%ccount\%ccount%c is the coloumn number. Thus, we can save the extra checking required at each step.

**Java**

**public** **class** **Solution** **{**

**public** **int[][]** **matrixReshape(int[][]** nums**,** **int** r**,** **int** c**)** **{**

**int[][]** res **=** **new** **int[**r**][**c**];**

**if** **(**nums**.**length **==** 0 **||** r **\*** c **!=** nums**.**length **\*** nums**[**0**].**length**)**

**return** nums**;**

**int** count **=** 0**;**

**for** **(int** i **=** 0**;** i **<** nums**.**length**;** i**++)** **{**

**for** **(int** j **=** 0**;** j **<** nums**[**0**].**length**;** j**++)** **{**

res**[**count **/** c**][**count **%** c**]** **=** nums**[**i**][**j**];**

count**++;**

**}**

**}**

**return** res**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(m∗n)O(m\*n)O(m∗n). We traverse the entire matrix of size m∗nm\*nm∗n once only. Here, mmm and nnn refers to the number of rows and columns in the given matrix.
* Space complexity : O(m∗n)O(m\*n)O(m∗n). The resultant matrix of size m∗nm\*nm∗n is used.

. Fizz Buzz

[Description](https://leetcode.com/problems/fizz-buzz/description/)[Hints](https://leetcode.com/problems/fizz-buzz/hints/)[Submissions](https://leetcode.com/problems/fizz-buzz/submissions/)[Discuss](https://leetcode.com/problems/fizz-buzz/discuss/)[Solution](https://leetcode.com/problems/fizz-buzz/solution/)

[Discuss](https://discuss.leetcode.com/category/540) [Pick One](https://leetcode.com/problems/random-one-question/)

Write a program that outputs the string representation of numbers from 1 to *n*.

But for multiples of three it should output “Fizz” instead of the number and for the multiples of five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”.

**Example:**

n = 15,

Return:

[

"1",

"2",

"Fizz",

"4",

"Buzz",

"Fizz",

"7",

"8",

"Fizz",

"Buzz",

"11",

"Fizz",

"13",

"14",

"FizzBuzz"

]

[C++ solution, 3ms](https://discuss.leetcode.com/topic/63774/c-solution-3ms)

**class** **Solution** {

**public**:

vector<string> fizzBuzz(**int** n) {

vector<string> ret\_vec(n);

**for**(**int** i=1; i<=n; ++i)

{

**if**(i%3 == 0)

{

ret\_vec[i-1] += "Fizz";

}

**if**(i%5 == 0)

{

ret\_vec[i-1] += "Buzz";

}

**if**(!ret\_vec[i-1].size())

{

ret\_vec[i-1] += to\_string(i);

}

}

**return** ret\_vec;

}

};

463. Island Perimeter

[Description](https://leetcode.com/problems/island-perimeter/description/)[Hints](https://leetcode.com/problems/island-perimeter/hints/)[Submissions](https://leetcode.com/problems/island-perimeter/submissions/)[Discuss](https://leetcode.com/problems/island-perimeter/discuss/)[Solution](https://leetcode.com/problems/island-perimeter/solution/)

[Discuss](https://discuss.leetcode.com/category/592) [Pick One](https://leetcode.com/problems/random-one-question/)

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water. Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells). The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

**Example:**

[[0,1,0,0],

[1,1,1,0],

[0,1,0,0],

[1,1,0,0]]

Answer: 16

Explanation: The perimeter is the 16 yellow stripes in the image below:



[Short Python](https://discuss.leetcode.com/topic/68778/short-python)

Since there are no lakes, every pair of neighbour cells with different values is part of the perimeter (more precisely, the edge between them is). So just count the differing pairs, both horizontally and vertically (for the latter I simply transpose the grid).

def islandPerimeter(self, grid):

return sum(sum(map(operator.ne, [0] + row, row + [0]))

**for** row **in** grid + map(list, zip(\*grid)))

[C++ solution with explanation](https://discuss.leetcode.com/topic/68845/c-solution-with-explanation)

1. find how many 1 in the map. If without the consideration of surrounding cells, the total perimeter should be the total amount of 1 times 4.
2. find how many cell walls that connect with both lands. We need to deduct twice of those lines from total perimeter

**int** islandPerimeter(**vector**<**vector**<**int**>>& **grid**) {

**int** count=0, repeat=0;

**for**(**int** i=0;i<**grid**.**size**();i++)

{

**for**(**int** j=0; j<**grid**[i].**size**();j++)

{

**if**(**grid**[i][j]==1)

{

count ++;

**if**(i!=0 && **grid**[i-1][j] == 1) repeat++;

**if**(j!=0 && **grid**[i][j-1] == 1) repeat++;

}

}

}

**return** 4\*count-repeat\*2;

}

[clear and easy java solution](https://discuss.leetcode.com/topic/68786/clear-and-easy-java-solution)

1. loop over the matrix and count the number of islands;
2. if the current dot is an island, count if it has any right neighbour or down neighbour;
3. the result is islands \* 4 - neighbours \* 2

public class Solution {

public **int** islandPerimeter(**int**[][] **grid**) {

**int** islands = 0, neighbours = 0;

**for** (**int** i = 0; i < **grid**.length; i++) {

**for** (**int** j = 0; j < **grid**[i].length; j++) {

**if** (**grid**[i][j] == 1) {

islands++; // count islands

**if** (i < **grid**.length - 1 && **grid**[i + 1][j] == 1) neighbours++; // count down neighbours

**if** (j < **grid**[i].length - 1 && **grid**[i][j + 1] == 1) neighbours++; // count right neighbours

}

}

}

**return** islands \* 4 - neighbours \* 2;

}

}