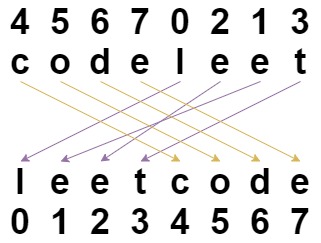
**Day 05-771, 1678, 2057, 2367, 1528.**

**25. Problem: 1528. Shuffle String**

You are given a string s and an integer array indices of the same length. The string s will be shuffled such that the character at the ith position moves to indices[i] in the shuffled string.

Return *the shuffled string*.

Example 1:



Input: s = "codeleet", indices = [4,5,6,7,0,2,1,3]

Output: "leetcode"

Explanation: As shown, "codeleet" becomes "leetcode" after shuffling.

Example 2:

Input: s = "abc", indices = [0,1,2]

Output: "abc"

Explanation: After shuffling, each character remains in its position.

Constraints:

s.length == indices.length == n

1 <= n <= 100

s consists of only lowercase English letters.

0 <= indices[i] < n

All values of indices are unique.

**Solution:**

**My Solution:**

class Solution\_1528 {  
 public String restoreString(String s, int[] in) {  
 char[] c = new char[in.length];  
 for (int i = 0; i < in.length; i++)  
 c[in[i]] = s.charAt(i);  
 return new String(c);  
 }  
}  
// O(n) - Time and Space  
  
// <https://leetcode.com/problems/shuffle-string/>

**Another Solution: 01**

*/\*\*  
 \* Cyclic Sort -O(n) Time , O(1) Space  
 \* The approach is to position the index found from the indices array at the correct location  
 \* loop through the elements of the indices array  
 \* continuously swap the characters at index and i and the index of the indices array  
 \*/*class Solution {  
 public String restoreString(String s, int[] indices) {  
 char[] chArray = s.toCharArray();  
 int i = 0;  
 while (i < indices.length) {  
 if (indices[i] != i) {  
 int index = indices[i];  
 // swap characters  
 char temp1 = chArray[index];  
 chArray[index] = chArray[i];  
 chArray[i] = temp1;  
 // swap indexes  
 int temp2 = indices[index];  
 indices[index] = indices[i];  
 indices[i] = temp2;  
 } else {  
 i++;  
 }  
 }  
 return new String(chArray);  
 }  
}

**Another Solution: 02**

class Solution {  
 public String restoreString(String s, int[] indices) {  
 StringBuilder res = new StringBuilder(s);  
 int l = s.length();  
 for (int i = 0; i < l; i++) {  
 res.setCharAt(indices[i], s.charAt(i));  
 }  
 return res.toString();  
 }  
}

**Another Solution: 03**

class Solution {  
 public String restoreString(String s, int[] indices) {  
 String answer = "";  
 int a = 0;  
 for (int i = 0; i < indices.length; i++) {  
 if (a < indices.length) {  
 if (indices[i] == a) {  
 answer += s.charAt(i);  
 a++;  
 i = -1;  
 }  
 }  
 }  
 return answer;  
 }  
}

**24. Problem: 2367. Number of Arithmetic triplets**

You are given a 0-indexed, strictly increasing integer array nums and a positive integer diff. A triplet (i, j, k) is an arithmetic triplet if the following conditions are met:

i < j < k,

nums[j] - nums[i] == diff, and

nums[k] - nums[j] == diff.

Return *the number of unique arithmetic triplets.*

Example 1:

Input: nums = [0,1,4,6,7,10], diff = 3

Output: 2

Explanation:

(1, 2, 4) is an arithmetic triplet because both 7 - 4 == 3 and 4 - 1 == 3.

(2, 4, 5) is an arithmetic triplet because both 10 - 7 == 3 and 7 - 4 == 3.

Example 2:

Input: nums = [4,5,6,7,8,9], diff = 2

Output: 2

Explanation:

(0, 2, 4) is an arithmetic triplet because both 8 - 6 == 2 and 6 - 4 == 2.

(1, 3, 5) is an arithmetic triplet because both 9 - 7 == 2 and 7 - 5 == 2.

Constraints:

3 <= nums.length <= 200

0 <= nums[i] <= 200

1 <= diff <= 50

nums is strictly increasing.

**Solution:**

**My Solution:**

class Solution\_2367 {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int t = 0;  
  
 for (int i = 0; i < nums.length; i++) {  
 for (int j = 0; j < nums.length; j++) {  
 for (int k = 0; k < nums.length; k++) {  
 if (i != j && j != k && i < j && j < k && nums[j] - nums[i] == diff && nums[k] - nums[j] == diff) {  
 t += 1;  
 }  
 }  
 }  
 }  
 return t;  
 }  
}  
  
// <https://leetcode.com/problems/number-of-arithmetic-triplets/>

**Another Solution: 01**

class Solution {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int cnt = 0;  
 Set<Integer> seen = new HashSet<>();  
 for (int num : nums) {  
 if (seen.contains(num - diff) && seen.contains(num - diff \* 2)) {  
 ++cnt;  
 }  
 seen.add(num);  
 }  
 return cnt;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int count = 0;  
 Map<Integer,Integer> map = new HashMap<>();  
 for(int i=0;i<nums.length;i++) map.put(nums[i],i);  
 for(int k : nums) count += (map.containsKey(k+diff) && map.containsKey(k+2\*diff)) ? 1 : 0;  
 return count;  
 }  
 // Time Complexity : O(log(n))  
 // Space Complexity : O(n)  
}

**Another Solution: 03**

class Solution {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int count = 0;  
 int pointer1 = 0 , pointer2 = 1 , pointer3 = 2;  
 while(pointer3 < nums.length){  
 int compare = nums[pointer2]-nums[pointer1];  
 if(compare==diff){  
 compare = nums[pointer3]-nums[pointer2];  
 if(compare==diff){  
 pointer1++;  
 pointer2++;  
 pointer3++;  
 count++;  
 }  
 else if(compare<diff) pointer3++;  
 else{  
 pointer1++;pointer2++;  
 pointer3 = Math.*max*(pointer3,pointer2+1);  
 }  
 }  
 else if(compare<diff){  
 pointer2++;  
 pointer3 = Math.*max*(pointer3,pointer2+1);  
 }  
 else{  
 pointer1++;  
 pointer2 = Math.*max*(pointer2,pointer1+1);  
 pointer3 = Math.*max*(pointer3,pointer2+1);  
 }  
 }  
 return count;  
 }  
 // Time Complexity : O(n)  
 // Space Complexity : O(1)  
}

**23. Problem: 2057. Smallest Index with Equal Value**

Given a 0-indexed integer array nums, return *the smallest index*i*of*nums*such that*i mod 10 == nums[i]*, or*-1*if such index does not exist*.

x mod y denotes the remainder when x is divided by y.

Example 1:

Input: nums = [0,1,2]

Output: 0

Explanation:

i=0: 0 mod 10 = 0 == nums[0].

i=1: 1 mod 10 = 1 == nums[1].

i=2: 2 mod 10 = 2 == nums[2].

All indices have i mod 10 == nums[i], so we return the smallest index 0.

Example 2:

Input: nums = [4,3,2,1]

Output: 2

Explanation:

i=0: 0 mod 10 = 0 != nums[0].

i=1: 1 mod 10 = 1 != nums[1].

i=2: 2 mod 10 = 2 == nums[2].

i=3: 3 mod 10 = 3 != nums[3].

2 is the only index which has i mod 10 == nums[i].

Example 3:

Input: nums = [1,2,3,4,5,6,7,8,9,0]

Output: -1

Explanation: No index satisfies i mod 10 == nums[i].

Constraints:

1 <= nums.length <= 100

0 <= nums[i] <= 9

**Solution:**

**My Solution:**

class Solution\_2057 {  
 public int smallestEqual(int[] nums) {  
 for (int i = 0; i < nums.length; i++) {  
 if (i % 10 == nums[i]) {  
 return i;  
 }  
 }  
 return -1;  
 }  
}  
  
// <https://leetcode.com/problems/smallest-index-with-equal-value/>

**Another Solution: 01**

class Solution {  
 public int smallestEqual(int[] nums) {  
 int min = 101;  
 for (int i = 0; i < nums.length; i++) {  
 if (i % 10 == nums[i]) {  
 min = Math.*min*(min, i);  
 }  
 }  
 return min == 101 ? -1 : min;  
 }  
}  
// TC: O(n), SC: O(1)

**Another Solution: 02**

class Solution {  
 public int smallestEqual(int[] nums) {  
 int index = 0;  
 for (int i = 0; i < nums.length; i++) {  
 if (index == nums[i]) {  
 return i;  
 }  
 if (++index == 10) {  
 index = 0;  
 }  
 }  
 return -1;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int smallestEqual(int[] nums) {  
 int n = nums.length; // Length of the array  
 int min = Integer.*MAX\_VALUE*; // Assign max value to min variable  
 int ans = 0;// Declare and initialize the ans variable  
 for (int i = 0; i < n; i++) { // Loop for iterating the mod operator throughout the elements  
 if (i % 10 == nums[i]) {  
 ans = i; // Index value is assigned to variable ans  
 min = Math.*min*(ans, min); // Find min out of the two variables  
 }  
 }  
 if (min == Integer.*MAX\_VALUE*) return -1; // If min value remains unchanged  
 return min; // Else return the min index  
 }  
}

**22. Problem: 1678. Goal Parser Interpretation**

You own a Goal Parser that can interpret a string command. The command consists of an alphabet of "G", "()" and/or "(al)" in some order. The Goal Parser will interpret "G" as the string "G", "()" as the string "o", and "(al)" as the string "al". The interpreted strings are then concatenated in the original order.

Given the string command, return *the Goal Parser's interpretation of*command.

Example 1:

Input: command = "G()(al)"

Output: "Goal"

Explanation: The Goal Parser interprets the command as follows:

G -> G

() -> o

(al) -> al

The final concatenated result is "Goal".

Example 2:

Input: command = "G()()()()(al)"

Output: "Gooooal"

Example 3:

Input: command = "(al)G(al)()()G"

Output: "alGalooG"

Constraints:

1 <= command.length <= 100

command consists of "G", "()", and/or "(al)" in some order.

**Solution:**

**My Solution:**

class Solution\_1678 {  
 public String interpret(String input) {  
 input = input.replaceAll("\\(\\)","o");  
 input = input.replaceAll("\\(al\\)","al");  
 return input;  
 }  
}  
  
// <https://leetcode.com/problems/goal-parser-interpretation/>

**Another Solution: 01**

class Solution {  
 public String interpret(String command) {  
 char[] str = command.toCharArray();  
 String fi="";  
 for(int i=0;i<str.length;i++){  
 if(str[i]=='G'){  
 fi=fi+str[i];  
 }  
 else if(str[i]=='(' && str[i+1]==')'){  
 fi=fi+'o';  
 }  
 else if(str[i]=='(' && str[i+1]=='a'){  
 fi= fi+"al";  
 }  
 }  
 return fi;  
 }  
}

**Another Solution: 02**

class Solution {  
 public String interpret(String command) {  
 StringBuilder S = new StringBuilder(command.length());  
 for(int i = 0 ; i < command.length() ; i++){  
 if(command.charAt(i)=='G') S.append('G');  
 if(command.charAt(i)=='('){  
 if(command.charAt(i+1)==')') {S.append('o'); i++;}  
 else{S.append("al"); i = i + 3;}  
 }  
 }  
 return S.toString();  
 }   
}

**Another Solution: 03**

class Solution {  
 public String interpret(String command) {  
 return command.replace("()", "o").replace("(al)", "al");  
 }   
}

**21. Problem: 771. Jewels and Stone**

You're given strings jewels representing the types of stones that are jewels, and stones representing the stones you have. Each character in stones is a type of stone you have. You want to know how many of the stones you have are also jewels.

Letters are case sensitive, so "a" is considered a different type of stone from "A".

Example 1:

Input: jewels = "aA", stones = "aAAbbbb"

Output: 3

Example 2:

Input: jewels = "z", stones = "ZZ"

Output: 0

Constraints:

1 <= jewels.length, stones.length <= 50

jewels and stones consist of only English letters.

All the characters of jewels are unique.

**Solution:**

**My Solution:**

class Solution\_771 {  
 public int numJewelsInStones(String jewels, String stones) {  
 int s = 0;  
 for(int i = 0 ; i < jewels.length() ; i++){  
 for(int j = 0 ; j < stones.length() ; j++){  
 if (jewels.charAt(i) == stones.charAt(j)){  
 s += 1;  
 //System.out.println(jewels.charAt(i) + " " + stones.charAt(j));  
 }  
 }  
 }  
 return s;  
 }  
}  
  
// <https://leetcode.com/problems/jewels-and-stones/submissions/>

**Another Solution: 01**

class Solution {  
 public int numJewelsInStones(String jewels, String stones) {  
 int num = 0;  
 for (int i = 0 ; i < stones.length(); i ++) {  
 if(jewels.indexOf(stones.charAt(i)) != -1) {  
 num++;  
 }  
 }  
 return num;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int numJewelsInStones(String J, String S) {  
 if(J.isEmpty()) return 0;  
 int cnt = 0;  
 for(int i = 0 ; i < S.length() ; i++) {  
 if(J.indexOf(S.charAt(i)) != -1) cnt++;  
 }  
 return cnt;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int numJewelsInStones(String jewels, String stones) {  
 int count=0;  
 for(char c:stones.toCharArray()){  
 if(jewels.indexOf(c)!=-1)count++;  
 }  
 return count;  
 }  
}

**Day 04-1470, 1672, 1431, 2367, 2057.**

**20. Problem: 2057. Smallest Index with equal value**

Given a **0-indexed** integer array nums, return *the****smallest****index*i*of*nums*such that*i mod 10 == nums[i]*, or*-1*if such index does not exist*.

x mod y denotes the **remainder** when x is divided by y.

**Example 1:**

**Input:** nums = [0,1,2]

**Output:** 0

**Explanation:**

i=0: 0 mod 10 = 0 == nums[0].

i=1: 1 mod 10 = 1 == nums[1].

i=2: 2 mod 10 = 2 == nums[2].

All indices have i mod 10 == nums[i], so we return the smallest index 0.

**Example 2:**

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

**Output:** 2

**Explanation:**

i=0: 0 mod 10 = 0 != nums[0].

i=1: 1 mod 10 = 1 != nums[1].

i=2: 2 mod 10 = 2 == nums[2].

i=3: 3 mod 10 = 3 != nums[3].

2 is the only index which has i mod 10 == nums[i].

**Example 3:**

**Input:** nums = [1,2,3,4,5,6,7,8,9,0]

**Output:** -1

**Explanation:** No index satisfies i mod 10 == nums[i].

**Constraints:**

1 <= nums.length <= 100

0 <= nums[i] <= 9

**Solution:**

**My Solution:**

package O1\_easy.day\_04;  
  
public class O20\_SmallestIndexWithEqualValue\_2057 {  
 public static void main(String[] args) {  
 Solution\_2057 solution\_2057 = new Solution\_2057();  
 int[] arr = {0, 1, 2};  
 int[] arr2 = {4, 3, 2, 1};  
 int[] arr3 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};  
 System.*out*.println(solution\_2057.smallestEqual(arr));  
 System.*out*.println(solution\_2057.smallestEqual(arr2));  
 System.*out*.println(solution\_2057.smallestEqual(arr3));  
 }  
}  
  
class Solution\_2057 {  
 public int smallestEqual(int[] nums) {  
 for (int i = 0; i < nums.length; i++) {  
 if (i % 10 == nums[i]) {  
 return i;  
 }  
 }  
 return -1;  
 }  
}  
  
// <https://leetcode.com/problems/smallest-index-with-equal-value/>

**Another Solution: 01**

class Solution {  
 public int smallestEqual(int[] a) {  
 for (int d1 = 0; d1 <= 9; d1++)  
 for (int d2 = 0; d2 <= 9 && 10 \* d1 + d2 < a.length; d2++)  
 if (d2 == a[d1 \* 10 + d2]) return d1 \* 10 + d2;  
 return -1;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int smallestEqual(int[] n) {  
 int k = n.length;  
 for (int i = 0; i < k; i++) {  
 if (i % 10 == n[i]) return i;  
 }  
 return -1;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int smallestEqual(int[] nums) {  
 for (int i = 0; i < nums.length; i++) {  
 if (i % 10 == nums[i]) {  
 return i;  
 }  
 }  
 return -1;  
 }  
}

**19. Problem: 2367. Number of Arithmetic Triplets**

You are given a **0-indexed**, **strictly increasing** integer array nums and a positive integer diff. A triplet (i, j, k) is an **arithmetic triplet** if the following conditions are met:

i < j < k,

nums[j] - nums[i] == diff, and

nums[k] - nums[j] == diff.

Return *the number of unique****arithmetic triplets****.*

**Example 1:**

**Input:** nums = [0,1,4,6,7,10], diff = 3

**Output:** 2

**Explanation:**

(1, 2, 4) is an arithmetic triplet because both 7 - 4 == 3 and 4 - 1 == 3.

(2, 4, 5) is an arithmetic triplet because both 10 - 7 == 3 and 7 - 4 == 3.

**Example 2:**

**Input:** nums = [4,5,6,7,8,9], diff = 2

**Output:** 2

**Explanation:**

(0, 2, 4) is an arithmetic triplet because both 8 - 6 == 2 and 6 - 4 == 2.

(1, 3, 5) is an arithmetic triplet because both 9 - 7 == 2 and 7 - 5 == 2.

**Constraints:**

3 <= nums.length <= 200

0 <= nums[i] <= 200

1 <= diff <= 50

nums is **strictly** increasing.

**Solution:**

**My Solution:**

import java.util.HashSet;  
import java.util.Set;  
import java.util.stream.IntStream;  
  
public class O19\_NumberOfArithmeticTriplets\_2367 {  
 public static void main(String[] args) {  
 Solution\_2367 solution\_2367 = new Solution\_2367();  
 int[] num = {0, 1, 4, 6, 7, 10};  
 int[] num2 = {4, 5, 6, 7, 8, 9};  
 System.*out*.println(solution\_2367.arithmeticTriplets(num, 3));  
 System.*out*.println(solution\_2367.arithmeticTriplets(num2, 2));  
 }  
}  
  
class Solution\_2367 {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int t = 0;  
 for (int i = 0; i < nums.length; i++) {  
 for (int j = 0; j < nums.length; j++) {  
 for (int k = 0; k < nums.length; k++) {  
 if (i != j && j != k && i < j && j < k && nums[j] - nums[i] == diff && nums[k] - nums[j] == diff) {  
 t += 1;  
 }  
 }  
 }  
 }  
 return t;  
 }  
}  
  
// <https://leetcode.com/problems/number-of-arithmetic-triplets/>

**Another Solution: 01**

class Solution {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int cnt = 0;  
 Set<Integer> seen = new HashSet<>();  
 for (int num : nums) {  
 if (seen.contains(num - diff) && seen.contains(num - diff \* 2)) {  
 ++cnt;  
 }  
 seen.add(num);  
 }  
 return cnt;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 int count = 0;  
 for (int i = 0; i < nums.length - 2; i++) {  
 for (int j = i + 1; j < nums.length - 1; j++) {  
 for (int k = j + 1; k < nums.length; k++) {  
 if (nums[j] - nums[i] == diff && nums[k] - nums[j] == diff) count++;  
 }  
 }  
 }  
 return count;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int arithmeticTriplets(int[] nums, int diff) {  
 Set<Integer> set = new HashSet<>();  
 for (int num : nums) {  
 set.add(num);  
 }  
 int ans = 0;  
 for (int num : nums) {  
 if ((set.contains(num + diff) && set.contains(num + 2 \* diff))) {  
 ans++;  
 }  
 }  
 return ans;  
 }  
}

**18. Problem: 1431. Kids with Greatest Number Of Candies**

There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have.

Return *a boolean array*result*of length*n*, where*result[i]*is*true*if, after giving the*ith*kid all the*extraCandies*, they will have the****greatest****number of candies among all the kids, or*false*otherwise*.

Note that **multiple** kids can have the **greatest** number of candies.

**Example 1:**

**Input:** candies = [2,3,5,1,3], extraCandies = 3

**Output:** [true,true,true,false,true]

**Explanation:** If you give all extraCandies to:

- Kid 1, they will have 2 + 3 = 5 candies, which is the greatest among the kids.

- Kid 2, they will have 3 + 3 = 6 candies, which is the greatest among the kids.

- Kid 3, they will have 5 + 3 = 8 candies, which is the greatest among the kids.

- Kid 4, they will have 1 + 3 = 4 candies, which is not the greatest among the kids.

- Kid 5, they will have 3 + 3 = 6 candies, which is the greatest among the kids.

**Example 2:**

**Input:** candies = [4,2,1,1,2], extraCandies = 1

**Output:** [true,false,false,false,false]

**Explanation:** There is only 1 extra candy.

Kid 1 will always have the greatest number of candies, even if a different kid is given the extra candy.

**Example 3:**

**Input:** candies = [12,1,12], extraCandies = 10

**Output:** [true,false,true]

**Constraints:**

n == candies.length

2 <= n <= 100

1 <= candies[i] <= 100

1 <= extraCandies <= 50

**Solution:**

**My Solution:**

import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class O18\_KidsWithGreatestNumberOfCandies\_1431 {  
 public static void main(String[] args) {  
 Solution\_1431 solution\_1431 = new Solution\_1431();  
 int[] arr = {2, 3, 5, 1, 3};  
 int[] arr2 = {4, 2, 1, 1, 2};  
 int[] arr3 = {12, 1, 12};  
 System.*out*.println(solution\_1431.kidsWithCandies(arr, 3));  
 System.*out*.println(solution\_1431.kidsWithCandies(arr2, 1));  
 System.*out*.println(solution\_1431.kidsWithCandies(arr3, 10));  
 }  
}  
  
class Solution\_1431 {  
 public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {  
 List<Boolean> result = new ArrayList<>();  
 for (int i = 0; i < candies.length; i++) {  
 boolean b = false;  
 for (int j = 0; j < candies.length; j++) {  
 if (i != j && candies[i] + extraCandies < candies[j]) {  
 b = false;  
 break;  
 } else {  
 b = true;  
 }  
 }  
 result.add(b);  
 }  
 return result;  
 }  
}  
  
// <https://leetcode.com/problems/kids-with-the-greatest-number-of-candies/>

**Another Solution: 01**

class Solution {  
 public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {  
 int max = Arrays.*stream*(candies).max().getAsInt();  
 return Arrays.*stream*(candies).mapToObj(candy -> candy + extraCandies >= max).collect(Collectors.*toList*());  
 }  
}

**Another Solution: 02**

class Solution {  
 public List<Boolean> kidsWithCandies(int[] cs, int e) {  
 int max = 0;  
 List<Boolean> res = new ArrayList<>();  
 for (int n : cs) max = Math.*max*(n, max);  
 for (int n : cs) res.add(n + e >= max);  
 return res;  
 }  
}

**Another Solution: 03**

class Solution {  
 public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {  
 List<Boolean> ans = new ArrayList<>(candies.length);  
 int max = 0;  
 for (int candy : candies) {  
 max = Math.*max*(candy, max);  
 }  
 for (int candy : candies) {  
 ans.add(candy + extraCandies >= max);  
 }  
 return ans;  
 }  
}

**17. Problem: 1672. Richest Customer Wealth**

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the i​​​​​​​​​​​th​​​​ customer has in the j​​​​​​​​​​​th​​​​ bank. Return*the****wealth****that the richest customer has.*

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

**Example 1:**

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

**Output:** 6

**Explanation:**

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

**Example 2:**

**Input:** accounts = [[1,5],[7,3],[3,5]]

**Output:** 10

**Explanation**:

1st customer has wealth = 6

2nd customer has wealth = 10

3rd customer has wealth = 8

The 2nd customer is the richest with a wealth of 10.

**Example 3:**

**Input:** accounts = [[2,8,7],[7,1,3],[1,9,5]]

**Output:** 17

**Constraints:**

m == accounts.length

n == accounts[i].length

1 <= m, n <= 50

1 <= accounts[i][j] <= 100

**Solution:**

**My Solution:**

import java.util.Arrays;  
  
public class O17\_RichestCustomerWealth\_1672 {  
 public static void main(String[] args) {  
 Solution\_1672 solution\_1672 = new Solution\_1672();  
 int[][] arr = {{1, 2, 3}, {3, 2, 1}};  
 int[][] arr2 = {{1, 5}, {7, 3}, {3, 5}};  
 int[][] arr3 = {{2, 8, 7}, {7, 1, 3}, {1, 9, 5}};  
 System.*out*.println(solution\_1672.maximumWealth(arr));  
 System.*out*.println(solution\_1672.maximumWealth(arr2));  
 System.*out*.println(solution\_1672.maximumWealth(arr3));  
 }  
}  
  
class Solution\_1672 {  
 public int maximumWealth(int[][] accounts) {  
 int max = 0;  
 for (int i = 0; i < accounts.length; i++) {  
 int a = wealth(accounts[i]);  
 if (a > max) {  
 max = a;  
 }  
 }  
 return max;  
 }  
  
 public int wealth(int[] a) {  
 int sum = 0;  
 for (int i = 0; i < a.length; i++) {  
 sum += a[i];  
 }  
 return sum;  
 }  
}  
  
// <https://leetcode.com/problems/richest-customer-wealth/submissions/>

**Another Solution: 01**

class Solution {  
 public int maximumWealth(int[][] accounts) {  
 int temp = 0;  
 for (int i = 0; i < accounts.length; i++) {  
 int sum = 0;  
 for (int j = 0; j < accounts[i].length; j++) {  
 sum = sum + accounts[i][j];  
 }  
 if (sum > temp) {  
 temp = sum;  
 }  
 }  
 return temp;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int maximumWealth(int[][] accounts) {  
 return Arrays.*stream*(accounts)  
 .mapToInt(i -> Arrays.*stream*(i).sum())  
 .max()  
 .getAsInt();  
 }  
}

**Another Solution: 03**

class Solution {  
 public int maximumWealth(int[][] accounts) {  
 var maxWealth = 0;  
 for (var customer : accounts) {  
 maxWealth = Math.*max*(maxWealth, Arrays.*stream*(customer).sum());  
 }  
 return maxWealth;  
 }  
}

**16. Problem: 1470. Shuffle the array**

Given the array nums consisting of 2n elements in the form [x1,x2,...,xn,y1,y2,...,yn].

*Return the array in the form* [x1,y1,x2,y2,...,xn,yn].

**Example 1:**

**Input:** nums = [2,5,1,3,4,7], n = 3

**Output:** [2,3,5,4,1,7]

**Explanation:** Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].

**Example 2:**

**Input:** nums = [1,2,3,4,4,3,2,1], n = 4

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

**Example 3:**

**Input:** nums = [1,1,2,2], n = 2

**Output:** [1,2,1,2]

**Constraints:**

1 <= n <= 500

nums.length == 2n

1 <= nums[i] <= 10^3

**Solution:**

**My Solution:**

import java.util.Arrays;  
import java.util.stream.IntStream;  
  
public class O16\_ShuffleTheString\_1470 {  
 public static void main(String[] args) {  
 Solution\_1470 solution\_1470 = new Solution\_1470();  
 int[] arr = {2, 5, 1, 3, 4, 7};  
 int[] arr2 = {1, 2, 3, 4, 4, 3, 2, 1};  
 int[] arr3 = {1, 1, 2, 2};  
 System.*out*.println(Arrays.*toString*(solution\_1470.shuffle(arr, 3)));  
 System.*out*.println(Arrays.*toString*(solution\_1470.shuffle(arr2, 4)));  
 System.*out*.println(Arrays.*toString*(solution\_1470.shuffle(arr3, 2)));  
 }  
}  
  
class Solution\_1470 {  
 public int[] shuffle(int[] nums, int n) {  
 int[] arrR = new int[nums.length];  
 int t = 0;  
 for (int k = 0; k < nums.length; k += 2) {  
 arrR[k] = nums[t];  
 t++;  
 }  
 int t2 = n;  
 for (int l = 1; l < nums.length; l += 2) {  
 arrR[l] = nums[t2];  
 t2++;  
 }  
 return arrR;  
 }  
}  
  
// <https://leetcode.com/problems/shuffle-the-array/submissions/>

**Another Solution: 01**

class Solution {  
 public int[] shuffle(int[] nums, int n) {  
 int len = nums.length;  
 for (int i = n; i < len; i++) {  
 nums[i] = (nums[i] << 10) | nums[i - n];  
 }  
 int index = 0;  
 for (int i = n; i < len; i++, index += 2) {  
 nums[index] = nums[i] & 1023;  
 nums[index + 1] = nums[i] >>> 10;  
 }  
 return nums;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int[] shuffle(int[] nums, int n) {  
 return IntStream.*range*(0, 2 \* n).map(i -> nums[i / 2 + i % 2 \* n]).toArray();  
 }  
}

**Another Solution: 03**

class Solution {  
 public int[] shuffle(int[] nums, int n) {  
 int[] res = new int[2 \* n];  
 for (int i = 0, j = n, idx = 0; idx < res.length; i++, j++) {  
 res[idx++] = nums[i];  
 res[idx++] = nums[j];  
 }  
 return res;  
 }  
}

**Day 03-2114, 1, 24, 415, 709.**

**15. Problem: 709. To Lower Case**

Given a string s, return the string after replacing every uppercase letter with the same lowercase letter.

**Example 1:**

**Input:** s = "Hello"

**Output:** "hello"

**Example 2:**

**Input:** s = "here"

**Output:** "here"

**Example 3:**

**Input:** s = "LOVELY"

**Output:** "lovely"

**Constraints:**

1 <= s.length <= 100

s consists of printable ASCII characters.

**Solution:**

**My Solution:**

public class O15\_ToLowerCase\_709 {  
 public static void main(String[] args) {  
 Solution\_709 solution\_709 = new Solution\_709();  
 String s = "Hello";  
 System.*out*.println(solution\_709.interpret(s));  
 }  
}  
  
class Solution\_709 {  
 public String interpret(String command) {  
 return command.toLowerCase();  
 }  
}  
  
// <https://leetcode.com/problems/to-lower-case/>

**Another Solution: 01**

class Solution\_709 {  
 public String toLowerCase(String str) {  
 char[] a = str.toCharArray();  
 for (int i = 0; i < a.length; i++)  
 if ('A' <= a[i] && a[i] <= 'Z')  
 a[i] = (char) (a[i] - 'A' + 'a');  
 return new String(a);  
 }  
}

**Another Solution: 02**

class Solution\_709 {  
 public String toLowerCase(String s) {  
 char[] a = s.toCharArray();  
 IntStream.*range*(0, a.length).filter(i -> 'A' <= a[i] && a[i] <= 'Z').forEach(i -> a[i] = (char) (a[i] - 'A' + 'a'));  
 return new String(a);  
 }  
}

**Another Solution: 03**

class Solution {  
 public String toLowerCase(String s) {  
 StringBuilder str = new StringBuilder();  
 for (int i = 0; i < s.length(); i++) {  
 if (s.charAt(i) >= 65 && s.charAt(i) <= 90) {  
 str.append((char) (s.charAt(i) + 32));  
 } else {  
 str.append(s.charAt(i));  
 }  
 }  
 return str.toString();  
 }  
}

**14. Problem: 415. Add String**

Given two non-negative integers, num1 and num2 represented as string, return *the sum of* num1 *and* num2 *as a string*.

You must solve the problem without using any built-in library for handling large integers (such as BigInteger). You must also not convert the inputs to integers directly.

**Example 1:**

**Input:** num1 = "11", num2 = "123"

**Output:** "134"

**Example 2:**

**Input:** num1 = "456", num2 = "77"

**Output:** "533"

**Example 3:**

**Input:** num1 = "0", num2 = "0"

**Output:** "0"

**Constraints:**

1 <= num1.length, num2.length <= 104

num1 and num2 consist of only digits.

num1 and num2 don't have any leading zeros except for the zero itself.

**Solution:**

**My Solution:**

import java.math.BigInteger;  
  
public class O14\_AddString\_415 {  
 public static void main(String[] args) {  
 Solution\_415 solution\_415 = new Solution\_415();  
 String s = "654615";  
 String s2 = "654615";  
 System.*out*.println(solution\_415.addStrings(s, s2));  
 }  
}  
  
class Solution\_415 {  
 public String addStrings(String num1, String num2) {  
 BigInteger bigInteger = new BigInteger(num1);  
 BigInteger bigInteger2 = new BigInteger(num2);  
 return String.*valueOf*(bigInteger2.add(bigInteger));  
 }  
}  
  
// <https://leetcode.com/problems/add-strings/submissions/>

**Another Solution: 01**

class Solution {  
 public String addStrings(String num1, String num2) {  
 int i = num1.length() - 1, j = num2.length() - 1, carry = 0;  
 StringBuilder sb = new StringBuilder();  
  
 while (i >= 0 || j >= 0 || carry != 0) {  
 if (i >= 0) carry += num1.charAt(i--) - '0';  
 if (j >= 0) carry += num2.charAt(j--) - '0';  
 sb.append(carry % 10);  
 carry /= 10;  
 }  
 return sb.reverse().toString();  
 }  
}

**Another Solution: 02**

class Solution {  
 public String addStrings(String num1, String num2) {  
 int carry = 0;  
 int i = num1.length() - 1, j = num2.length() - 1;  
 StringBuilder sb = new StringBuilder();  
 while (i >= 0 || j >= 0) {  
 int n1 = 0, n2 = 0;  
 if (i >= 0) {  
 n1 = num1.charAt(i) - '0';  
 }  
 if (j >= 0) {  
 n2 = num2.charAt(j) - '0';  
 }  
 int sum = n1 + n2 + carry;  
 carry = sum / 10;  
 sb.append(sum % 10);  
 i--;  
 j--;  
 }  
 if (carry != 0) {  
 sb.append(carry);  
 }  
 return sb.reverse().toString();  
 }  
}

**Another Solution: 03**

public class Solution {  
 public String addStrings(String num1, String num2) {  
 int i = num1.length() - 1;  
 int j = num2.length() - 1;  
 int carry = 0;  
 char[] num1Array = num1.toCharArray();  
 char[] num2Array = num2.toCharArray();  
 StringBuilder sb = new StringBuilder();  
 while (i >= 0 || j >= 0 || carry == 1) {  
 int a = i >= 0 ? (num1Array[i--] - '0') : 0;  
 int b = j >= 0 ? (num2Array[j--] - '0') : 0;  
 int sum = a + b + carry;  
 sb.insert(0, sum % 10);  
 carry = sum / 10;  
 }  
 return sb.toString();  
 }  
}  
  
*/\*\*  
 \* Complexity Analysis  
 \* Time Complexity: `O(m + n)` (Average Case) and `O(m + n)` (Worst Case) where `m` and `n` are the total number of characters  
 \* in the first and second input respectively. The algorithm evaluate each character for potential carry.  
 \*  
 \* Auxiliary Space: `O(m + n)` space is used where `m` and `n` are the total number of characters in the first and second input respectively.  
 \* Converting both input to character array required extra space.  
 \*  
 \* Algorithm  
 \* Approach: Iterative  
 \* \* The while loop will run as long as there are characters left in one of the strings or when there is a carry in remaining.  
 \* Starting from right to left, each character is converted to integer by the mean of offsetting its ASCII value.  
 \* If the shorter string is exhausted first, the value will be forced to `0` as default from there onwards.  
 \* Sum for that particular position is conveniently calculated and a modulus of `10` will extract the digit portion in case the sum is bigger than 10.  
 \* Carry in is extracted by flooring the number after division by `10`. StringBuilder is used due to its efficiently in inserting character to  
 \* existing StringBuilder object. If normal String is used then each insertion by + operation will have to copy over the immutable String object which is highly inefficient*

**13. Problem: 26. Remove Duplicates From sorted array**

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of nums.

Do **not** allocate extra space for another array. You must do this by **modifying the input array**[**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Custom Judge:**

The judge will test your solution with the following code:

int[] nums = [...]; // Input array

int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;

for (int i = 0; i < k; i++) {

assert nums[i] == expectedNums[i];

}

If all assertions pass, then your solution will be **accepted**.

**Example 1:**

**Input:** nums = [1,1,2]

**Output:** 2, nums = [1,2,\_]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Example 2:**

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]

**Output:** 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

**Explanation:** Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Constraints:**

1 <= nums.length <= 3 \* 104

-100 <= nums[i] <= 100

nums is sorted in **non-decreasing** order.

**Solution:**

**My Solution:**

import java.util.\*;  
  
public class O13\_RemoveDuplicateFromsortedArray\_26 {  
 public static void main(String[] args) {  
 Solution solution = new Solution();  
 int[] arr = {1, 1, 2};  
 int[] arr2 = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};  
 System.*out*.println(solution.removeDuplicates(arr));  
 System.*out*.println(solution.removeDuplicates(arr2));  
 }  
}  
  
class Solution {  
 public int removeDuplicates(int[] nums) {  
 int[] temp = new int[nums.length];  
 int j = 0;  
 for (int i = 0; i < nums.length - 1; i++) {  
 if (nums[i] != nums[i + 1]) {  
 temp[j++] = nums[i];  
 }  
 }  
  
 temp[j++] = nums[nums.length - 1];  
  
 // Changing original array  
 for (int i = 0; i < j; i++) {  
 nums[i] = temp[i];  
 }  
 return j;  
 }  
}  
  
// <https://leetcode.com/problems/remove-duplicates-from-sorted-array/>

**Another Solution: 01**

class Solution {  
 public int removeDuplicates(int[] nums) {  
 int i = 0;  
 for (int n : nums)  
 if (i == 0 || n > nums[i - 1])  
 nums[i++] = n;  
 return i;  
 }  
}  
  
// And to not need the i ==0check in the loop:  
  
class Solution {  
 public int removeDuplicates(int[] nums) {  
 int i = nums.length > 0 ? 1 : 0;  
 for (int n : nums)  
 if (n > nums[i - 1])  
 nums[i++] = n;  
 return i;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int removeDuplicates(int[] nums) {  
 int pos = 0;  
 for (int num : nums) {  
 if (nums[pos] != num) {  
 nums[++pos] = num;  
 }  
 }  
 return pos + 1;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int removeDuplicates(int[] A) {  
 if (A.length == 0) return 0;  
 int j = 0;  
 for (int i = 0; i < A.length; i++)  
 if (A[i] != A[j]) A[++j] = A[i];  
 return ++j;  
 }  
}

**12. Problem: 01. Two Sum**

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

**Input:** nums = [3,2,4], target = 6

**Output:** [1,2]

**Example 3:**

**Input:** nums = [3,3], target = 6

**Output:** [0,1]

**Constraints:**

2 <= nums.length <= 104

-109 <= nums[i] <= 109

-109 <= target <= 109

**Only one valid answer exists.**

Follow-up: Can you come up with an algorithm that is less than O(n2) time complexity?

**Solution:**

**My Solution:**

import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
  
public class O12\_TwoSum\_1 {  
 public static void main(String[] args) {  
 Solution\_1 solution\_1 = new Solution\_1();  
 int[] arr = {2, 7, 11, 15};  
 int[] arr2 = {3, 2, 4};  
 System.*out*.println(Arrays.*toString*(solution\_1.twoSum(arr, 9)));  
 System.*out*.println(Arrays.*toString*(solution\_1.twoSum(arr2, 6)));  
 }  
}  
  
class Solution\_1 {  
 public int[] twoSum(int[] nums, int target) {  
 int[] result = new int[2];  
 for (int i = 0; i < nums.length; i++) {  
 for (int j = 0; j < nums.length; j++) {  
 if (i != j && nums[i] + nums[j] == target) {  
 result[0] = i;  
 result[1] = j;  
 break;  
 }  
 }  
 }  
 return result;  
 }  
}  
  
// <https://leetcode.com/problems/two-sum/submissions/>

**Another Solution: 01**

class Solution {  
 public int[] twoSum(int[] numbers, int target) {  
 int[] result = new int[2];  
 Map<Integer, Integer> map = new HashMap<Integer, Integer>();  
 for (int i = 0; i < numbers.length; i++) {  
 if (map.containsKey(target - numbers[i])) {  
 result[1] = i;  
 result[0] = map.get(target - numbers[i]);  
 return result;  
 }  
 map.put(numbers[i], i);  
 }  
 return result;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int[] twoSum(int[] nums, int target) {  
 HashMap<Integer, Integer> seen = new HashMap<>();  
 for (int i = 0; i < nums.length; ++i) {  
 int b = nums[i], a = target - b;  
 if (seen.containsKey(a)) return new int[]{seen.get(a), i}; // Found pair of (a, b), so that a + b = target  
 seen.put(b, i);  
 }  
 return new int[]{};  
 }  
}

**Another Solution: 03**

class Solution {  
 //O(nlogn)  
 public int[] twoSum\_n2(int[] nums, int target) {  
 if (nums == null) return null;  
 int[] nums2 = Arrays.*copyOf*(nums, nums.length);  
 Arrays.*sort*(nums2);  
 int a = 0, b = 0;  
 int start = 0, end = nums2.length - 1;  
 //find two nums  
 while (start < end) {  
 int sum = nums2[start] + nums2[end];  
 if (sum < target) start++;  
 else if (sum > target) end--;  
 else {  
 a = nums2[start];  
 b = nums2[end];  
 break;  
 }  
 }  
 //find the index of two numbers  
 int[] res = new int[2];  
 for (int i = 0; i < nums.length; i++) {  
 if (nums[i] == a) {  
 res[0] = i;  
 break;  
 }  
 }  
 if (a != b) {  
 for (int i = 0; i < nums.length; i++) {  
 if (nums[i] == b) {  
 res[1] = i;  
 break;  
 }  
 }  
 } else {  
 for (int i = 0; i < nums.length; i++) {  
 if (nums[i] == b && i != res[0]) {  
 res[1] = i;  
 break;  
 }  
 }  
 }  
 return res;  
 }  
}

**11. Problem: 2114. Maximum Number of Words found in Sentence**

A sentence is a list of words that are separated by a single space with no leading or trailing spaces.

You are given an array of strings sentences, where each sentences[i] represents a single sentence.

Return *the maximum number of words that appear in a single sentence*.

Example 1:

Input: sentences = ["alice and bob love leetcode", "i think so too", "this is great thanks very much"]

Output: 6

Explanation:

- The first sentence, "alice and bob love leetcode", has 5 words in total.

- The second sentence, "i think so too", has 4 words in total.

- The third sentence, "this is great thanks very much", has 6 words in total.

Thus, the maximum number of words in a single sentence comes from the third sentence, which has 6 words.

Example 2:

Input: sentences = ["please wait", "continue to fight", "continue to win"]

Output: 3

Explanation: It is possible that multiple sentences contain the same number of words.

In this example, the second and third sentences (underlined) have the same number of words.

Constraints:

1 <= sentences.length <= 100

1 <= sentences[i].length <= 100

sentences[i] consists only of lowercase English letters and ' ' only.

sentences[i] does not have leading or trailing spaces.

All the words in sentences[i] are separated by a single space.

**Solution:**

**My Solution:**

import java.util.Arrays;  
import java.util.stream.Stream;  
  
public class O11\_MaxNumberOfWordInSentence\_2114 {  
 public static void main(String[] args) {  
 Solution\_2114 solution\_2114 = new Solution\_2114();  
 String[] arr = {"alice and bob love leetcode", "i think so too", "this is great thanks very much"};  
 String[] arr2 = {"please wait", "continue to fight", "continue to win"};  
 System.*out*.println((solution\_2114.mostWordsFound(arr)));  
 System.*out*.println((solution\_2114.mostWordsFound(arr2)));  
 }  
}  
  
class Solution\_2114 {  
 public int mostWordsFound(String[] sentences) {  
 int max = 0;  
 for (int i = 0; i < sentences.length; i++) {  
 int c = most(sentences[i]);  
 if (c > max) {  
 max = c;  
 }  
 }  
 return max;  
 }  
  
 public int most(String s) {  
 int m = 0;  
 String[] strings = s.split(" ");  
 m = strings.length;  
 return m;  
 }  
}  
  
// <https://leetcode.com/problems/maximum-number-of-words-found-in-sentences/>

**Another Solution: 01**

class Solution {  
 public int mostWordsFound(String[] sentences) {  
 return Stream.*of*(sentences).mapToInt(s -> s.split(" ").length).max().getAsInt();  
 }  
}

**Another Solution: 02**

class Solution {  
 public int mostWordsFound(String[] sentences) {  
 int max = 0;  
 for (String s : sentences) {  
 max = Math.*max*(max, s.split(" ").length);  
 }  
 return max;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int mostWordsFound(String[] sentences) {  
 int max = 0;  
 for (String s : sentences) {  
 int x = s.indexOf(" "), count = 1;  
 while (x != -1) {  
 count++;  
 x = s.indexOf(" ", x + 1);  
 }  
 max = Math.*max*(count, max);  
 }  
 return max;  
 }  
}

**Day 02-1929, 1920, 1512, 1365, 1816.**

**10. Problem: 1816. Truncate Sentence**

A **sentence** is a list of words that are separated by a single space with no leading or trailing spaces. Each of the words consists of **only** uppercase and lowercase English letters (no punctuation).

For example, "Hello World", "HELLO", and "hello world hello world" are all sentences.

You are given a sentence s​​​​​​ and an integer k​​​​​​. You want to **truncate** s​​​​​​ such that it contains only the **first** k​​​​​​ words. Return s​​​​*​​ after****truncating****it.*

**Example 1:**

**Input:** s = "Hello how are you Contestant", k = 4

**Output:** "Hello how are you"

**Explanation:**

The words in s are ["Hello", "how" "are", "you", "Contestant"].

The first 4 words are ["Hello", "how", "are", "you"].

Hence, you should return "Hello how are you".

**Example 2:**

**Input:** s = "What is the solution to this problem", k = 4

**Output:** "What is the solution"

**Explanation:**

The words in s are ["What", "is" "the", "solution", "to", "this", "problem"].

The first 4 words are ["What", "is", "the", "solution"].

Hence, you should return "What is the solution".

**Example 3:**

**Input:** s = "chopper is not a tanuki", k = 5

**Output:** "chopper is not a tanuki"

**Constraints:**

1 <= s.length <= 500

k is in the range [1, the number of words in s].

s consist of only lowercase and uppercase English letters and spaces.

The words in s are separated by a single space.

There are no leading or trailing spaces.

**Solution:**

**My Solution:**

public class O10\_TruncateSentence\_1816 {  
 public static void main(String[] args) {  
 Solution\_1816 solution\_1816 = new Solution\_1816();  
 System.*out*.println(solution\_11816.truncateSentence("Hello how are you Contestant", 4));  
 }  
}  
class Solution\_1816 {  
 public String truncateSentence(String s, int k) {  
 String[] na = s.split(" ");  
 String newString = "";  
 for (int i = 0; i < k; i++) {  
 if (i != k - 1) {  
 newString += na[i] + " ";  
 } else {  
 newString += na[i];  
 }  
 }  
 return newString;  
 }  
}  
  
// <https://leetcode.com/problems/truncate-sentence/>

**Another Solution: 01**

//Approch 1:[using split]  
class Solution {  
 public String truncateSentence(String s, int k) {  
 String[] str = s.split(" ");  
 StringBuilder truncatedSentence = new StringBuilder();  
 for (int i = 0; i < k - 1; i++) {  
 truncatedSentence.append(str[i] + " ");  
 }  
 truncatedSentence.append(str[k - 1]); //because we don't want space for last word  
 return truncatedSentence.toString();  
 }  
}  
//Complexity:  
//Time:O(n) and Space:O(n)  
//Note: Split function will iterate over the string to split on the basis of "". thats why time is O(n) not O(k)

//Approch 2:[without split]  
class Solution {  
 public String truncateSentence(String s, int k) {  
 int idx = 0;  
 int spaceCount = 0;  
  
 while (idx < s.length() && spaceCount < k) {  
 if (s.charAt(idx) == ' ') spaceCount++;  
 idx++;  
 }  
 // if(spaceCount<k) means we have to include whole string  
 return spaceCount == k ? s.substring(0, idx - 1) : s;  
 }  
}  
//Complexity:  
//Time:O(n) and O(1) [Big O notation average time complexity is better than approch 1]

**Another Solution: 02**

class Solution {  
 public String truncateSentence(String s, int k) {  
 String[] words = s.split(" ");  
 StringBuilder sb = new StringBuilder(words[0]);  
 for (int i = 1; i < k; ++i) {  
 sb.append(" " + words[i]);  
 }  
 return sb.toString();  
 }  
}

**Another Solution: 03**

class Solution {  
 public String truncateSentence(String s, int k) {  
 int n = s.length();  
 int count = 0;  
 int i = 0;  
 while (i < n) {  
 if (s.charAt(i) == ' ') {  
 count++;  
 if (count == k)  
 return s.substring(0, i);  
 }  
 i++;  
 }  
 return s;  
 }  
}

**09. Problem: 1365. How many numbers are smaller than the current number**

Given the array nums, for each nums[i] find out how many numbers in the array are smaller than it. That is, for each nums[i] you have to count the number of valid j's such that j != i and nums[j] < nums[i].

Return the answer in an array.

Example 1:

Input: nums = [8,1,2,2,3]

Output: [4,0,1,1,3]

Explanation:

For nums[0]=8 there exist four smaller numbers than it (1, 2, 2 and 3).

For nums[1]=1 does not exist any smaller number than it.

For nums[2]=2 there exist one smaller number than it (1).

For nums[3]=2 there exist one smaller number than it (1).

For nums[4]=3 there exist three smaller numbers than it (1, 2 and 2).

Example 2:

Input: nums = [6,5,4,8]

Output: [2,1,0,3]

Example 3:

Input: nums = [7,7,7,7]

Output: [0,0,0,0]

Constraints:

2 <= nums.length <= 500

0 <= nums[i] <= 100

**Solution:**

**My Solution:**

import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
  
public class O9\_Test {  
 public static void main(String[] args) {  
 Solution\_1365 solution\_1365 = new Solution\_1365();  
 int[] arr = {8, 1, 2, 2, 3};  
 int[] arr2 = {6, 5, 4, 8};  
 int[] arr3 = {7, 7, 7, 7};  
 System.*out*.println((Arrays.*toString*(solution\_1365.smallerNumbersThanCurrent(arr))));  
 System.*out*.println((Arrays.*toString*(solution\_1365.smallerNumbersThanCurrent(arr2))));  
 System.*out*.println((Arrays.*toString*(solution\_1365.smallerNumbersThanCurrent(arr3))));  
 }  
}  
  
class Solution\_1365 {  
 public int[] smallerNumbersThanCurrent(int[] nums) {  
 int[] result = new int[nums.length];  
 int count = 0;  
 for (int i = 0; i < nums.length; i++) {  
 for (int j = 0; j < nums.length; j++) {  
 if (nums[i] > nums[j]) {  
 count++;  
 }  
 }  
 result[i] = count;  
 count = 0;  
 }  
 return result;  
 }  
}  
  
// <https://leetcode.com/problems/how-many-numbers-are-smaller-than-the-current-number/submissions/>   
  
*/\*\*  
 \* Both i and j loop start from 0.  
 \* so that every element is compare to every element.  
 \*/*

**Another Solution: 01**

class Solution {  
 public int[] smallerNumbersThanCurrent(int[] nums) {  
 int[] answer = new int[nums.length];  
 int count;  
 for (int i = 0; i < nums.length; i++) {  
 count = 0;  
 for (int j = 0; j < nums.length; j++) {  
 if (i != j && nums[j] < nums[i])  
 count++;  
 }  
 answer[i] = count;  
 }  
 return answer;  
 }  
}

**Another Solution: 02**

class Solution {  
 public int[] smallerNumbersThanCurrent(int[] nums) {  
 int[] buck = new int[101];  
 for (int i = 0; i < nums.length; i++) {  
 buck[nums[i]] += 1;  
 }  
 for (int j = 1; j <= 100; j++) {  
 buck[j] += buck[j - 1];  
 }  
 for (int k = 0; k < nums.length; k++) {  
 int pos = nums[k];  
 nums[k] = pos == 0 ? 0 : buck[pos - 1];  
 }  
 return nums;  
 }  
}

**Another Solution: 03**

class Solution {  
 public int[] smallerNumbersThanCurrent(int[] nums) {  
 Map<Integer, Integer> map = new HashMap<>();  
 int[] copy = nums.clone();  
 Arrays.*sort*(copy);  
 for (int i = 0; i < nums.length; i++) {  
 map.putIfAbsent(copy[i], i);  
 }  
 for (int i = 0; i < nums.length; i++) {  
 copy[i] = map.get(nums[i]);  
 }  
 return copy;  
 }  
}

**08. Problem: 1512. Number of Good Pairs**

Given an array of integers nums, return *the number of****good pairs***.

A pair (i, j) is called *good* if nums[i] == nums[j] and i < j.

**Example 1:**

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

**Output:** 4

**Explanation:** There are 4 good pairs (0,3), (0,4), (3,4), (2,5) 0-indexed.

**Example 2:**

**Input:** nums = [1,1,1,1]

**Output:** 6

**Explanation:** Each pair in the array are *good*.

**Example 3:**

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

**Output:** 0

**Constraints:**

1 <= nums.length <= 100

1 <= nums[i] <= 100

Accepted

362,718

Submissions

411,274

**Solution:**

**My Solution:**

package O1\_easy.day\_02;  
  
import java.util.Arrays;  
import java.util.HashMap;  
  
public class O8\_NumberOfGoodPairs\_1512 {  
 public static void main(String[] args) {  
 Solution\_1470 solution\_1470 = new Solution\_1470();  
 int[] arr = {1, 2, 3, 1, 1, 3};  
 int[] arr2 = {1, 1, 1, 1};  
 int[] arr3 = {1, 2, 3};  
 System.*out*.println((solution\_1470.numIdenticalPairs(arr)));  
 System.*out*.println((solution\_1470.numIdenticalPairs(arr2)));  
 System.*out*.println((solution\_1470.numIdenticalPairs(arr3)));  
 }  
}  
  
class Solution\_1470 {  
 public int numIdenticalPairs(int[] nums) {  
 int count = 0;  
 for (int i = 0; i < nums.length; i++) {  
 for (int j = 1; j < nums.length; j++) {  
 if (nums[i] == nums[j] && i < j) {  
 count++;  
 }  
 }  
 }  
 return count;  
 }  
}

// <https://leetcode.com/problems/number-of-good-pairs/>

**Another Solution: 01**

class Solution1 {  
 public int numIdenticalPairs1(int[] A) {  
 int res = 0, count[] = new int[101];  
 for (int a : A) {  
 res += count[a]++;  
 }  
 return res;  
 }  
}

**Another Solution: 02**

class Solution2 {  
 public int numIdenticalPairs2(int[] guestList) {  
 HashMap<Integer, Integer> hm = new HashMap<>();  
 int ans = 0;  
 for (int friend : guestList) {  
 int friendCount = hm.getOrDefault(friend, 0);  
 ans += friendCount;  
 hm.put(friend, friendCount + 1);  
 }  
 return ans;  
 }  
}

**Another Solution: 03**

class Solution3 {  
 public int numIdenticalPairs3(int[] nums) {  
 int ans = 0;  
 int[] count = new int[101];  
 for (int n : nums)  
 count[n]++;  
 for (int n : count)  
 ans += (n \* (n - 1)) / 2;  
 return ans;  
 }  
}

**07. Problem: 2011. Final value of variable after performing operation**

There is a programming language with only **four** operations and **one** variable X:

++X and X++ **increments** the value of the variable X by 1.

--X and X-- **decrements** the value of the variable X by 1.

Initially, the value of X is 0.

Given an array of strings operations containing a list of operations, return *the****final****value of*X *after performing all the operations*.

**Example 1:**

**Input:** operations = ["--X","X++","X++"]

**Output:** 1

**Explanation:** The operations are performed as follows:

Initially, X = 0.

--X: X is decremented by 1, X = 0 - 1 = -1.

X++: X is incremented by 1, X = -1 + 1 = 0.

X++: X is incremented by 1, X = 0 + 1 = 1.

**Example 2:**

**Input:** operations = ["++X","++X","X++"]

**Output:** 3

**Explanation:** The operations are performed as follows:

Initially, X = 0.

++X: X is incremented by 1, X = 0 + 1 = 1.

++X: X is incremented by 1, X = 1 + 1 = 2.

X++: X is incremented by 1, X = 2 + 1 = 3.

**Example 3:**

**Input:** operations = ["X++","++X","--X","X--"]

**Output:** 0

**Explanation:** The operations are performed as follows:

Initially, X = 0.

X++: X is incremented by 1, X = 0 + 1 = 1.

++X: X is incremented by 1, X = 1 + 1 = 2.

--X: X is decremented by 1, X = 2 - 1 = 1.

X--: X is decremented by 1, X = 1 - 1 = 0.

**Constraints:**

1 <= operations.length <= 100

operations[i] will be either "++X", "X++", "--X", or "X--".

**Solution:**

**My Solution:**

import java.util.Arrays;  
  
public class O7\_Test {  
 public static void main(String[] args) {  
 Solution\_2011 solution\_2011 = new Solution\_2011();  
 String[] arr = {"--X", "X++", "X++"};  
 String[] arr2 = {"X++", "++X", "--X", "X--"};  
 String[] arr3 = {"++X", "++X", "X++"};  
 System.*out*.println(solution\_2011.finalValueAfterOperations(arr));  
 System.*out*.println(solution\_2011.finalValueAfterOperations(arr2));  
 System.*out*.println(solution\_2011.finalValueAfterOperations(arr3));  
  
 }  
}  
  
class Solution\_2011 {  
 public int finalValueAfterOperations(String[] operations) {  
  
 int result = 0;  
 for (int i = 0; i < operations.length; i++) {  
 if (operations[i].equals("++X")) {  
 result = 1 + result;  
 } else if (operations[i].equals("--X")) {  
 result = result - 1;  
 } else if (operations[i].equals("X++")) {  
 result = result + 1;  
 } else if (operations[i].equals("X--")) {  
 result = result - 1;  
 }  
 }  
 return result;  
 }  
}

// <https://leetcode.com/problems/build-array-from-permutation/>

**Another Solution: 01**

class Solution1 {  
 public int finalValueAfterOperations1(String[] operations) {  
 int val = 0;  
 for (int i = 0; i < operations.length; i++) {  
 if (operations[i].charAt(1) == '+') val++;  
 else val--;  
 }  
 return val;  
 }  
}

**Another Solution: 02**

class Solution2 {  
 public int finalValueAfterOperations2(String[] operations) {  
 int res = 0;  
 for (String operation : operations) {  
 res += operation.charAt(1) == '+' ? 1 : -1;  
 }  
 return res;  
 }  
}

**Another Solution: 03**

public static int finalValueAfterOperations3(String[] operations) {  
 return Arrays.*stream*(operations, 0, operations.length)  
 .mapToInt(operation -> operation.charAt(1) == '+' ? 1 : -1)  
 .sum();  
 }

**06. Problem: 1929. Concatenation of Array**

Given an integer array nums of length n, you want to create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (**0-indexed**).

Specifically, ans is the **concatenation** of two nums arrays.

Return *the array*ans.

**Example 1:**

**Input:** nums = [1,2,1]

**Output:** [1,2,1,1,2,1]

**Explanation:** The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]]

- ans = [1,2,1,1,2,1]

**Example 2:**

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

**Output:** [1,3,2,1,1,3,2,1]

**Explanation:** The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[3],nums[0],nums[1],nums[2],nums[3]]

- ans = [1,3,2,1,1,3,2,1]

**Constraints:**

n == nums.length

1 <= n <= 1000

1 <= nums[i] <= 1000

**Solution:**

**My Solution:**

public class O6\_ConcatenationOfArray\_1929 {  
 public static void main(String[] args) {  
 Solution\_1929 solution\_1929 = new Solution\_1929();  
 int[] arr = {1, 3, 2, 1};  
 System.*out*.println(Arrays.*toString*(solution\_1929.getConcatenation(arr)));  
 }  
}  
class Solution\_1929 {  
 public int[] getConcatenation(int[] nums) {  
 int[] newAnswer = new int[nums.length \* 2];  
 int givenArrayLength = nums.length;  
 for (int i = 0; i < nums.length; i++) {  
 newAnswer[i] = nums[i];  
 newAnswer[i + givenArrayLength] = nums[i];  
 }  
 return newAnswer;  
 }  
}  
  
// https://leetcode.com/problems/concatenation-of-array/  
  
*/\*\*  
 \* (int i = 0; i < nums.length; i++) --> must be i<nums.length or i<=nums.length-1  
 \*/*

**Another Solution: 01**

public int[] getConcatenation(int[] nums) {  
 int[] result = new int[nums.length \* 2];  
 for (int i = 0; i < nums.length; i++)  
 result[i + nums.length] = result[i] = nums[i];  
 return result;  
}

**Another Solution: 02**

public int[] getConcatenation(int[] nums) {  
 int[] ans = new int[nums.length \* 2];  
 for (int i = 0; i < nums.length; i++) {  
 ans[i] = nums[i];  
 ans[nums.length + i] = nums[i];  
 }  
 return ans;  
}

**Day 01- 9, 1480, 1108, 1662, 2235.**

**5. Problem: 1108. Defanging an IP Address**

Given a valid (IPv4) IP address, return a defanged version of that IP address.

A *defanged IP address* replaces every period "." with "[.]".

**Example 1:**

**Input:** address = "1.1.1.1"

**Output:** "1[.]1[.]1[.]1"

**Example 2:**

**Input:** address = "255.100.50.0"

**Output:** "255[.]100[.]50[.]0"

**Constraints:**

The given address is a valid IPv4 address.

**Solution:**

**My Solution:**

public class O5\_DefiningIpAddress\_1108 {  
 public static void main(String[] args) {  
 Solution\_1108 solution\_1108 = new Solution\_1108();  
 System.*out*.println(solution\_1108.defangIPaddr("1.1.1.1"));  
 }  
}  
  
class Solution\_1108 {  
 public String defangIPaddr(String address) {  
 return address.replace(".", "[.]");  
 }  
}

**Another Solution: 01**

// Another solution:  
class Solution {  
 public String defangIPaddr(String address) {  
 StringBuilder str = new StringBuilder();  
 for (int i = 0; i < address.length(); i++) {  
 if (address.charAt(i) == '.') {  
 str.append("[.]");  
 } else {  
 str.append(address.charAt(i));  
 }  
 }  
 return str.toString();  
 }  
}

**Another Solution: 02**

//Another Solution:  
class Solution {  
 public String defangIPaddr(String address) {  
 return address.replaceAll("\\.", "[.]");  
 }  
}

**Another Solution: 03**

**04. Problem: 1662. Check If Two String array are Equivalent**

Given two string arrays word1 and word2, return true if the two arrays **represent** the same string, and false otherwise.

A string is **represented** by an array if the array elements concatenated **in order** forms the string.

**Example 1:**

**Input:** word1 = ["ab", "c"], word2 = ["a", "bc"]

**Output:** true

**Explanation:**

word1 represents string "ab" + "c" -> "abc"

word2 represents string "a" + "bc" -> "abc"

The strings are the same, so return true.

**Example 2:**

**Input:** word1 = ["a", "cb"], word2 = ["ab", "c"]

**Output:** false

**Example 3:**

**Input:** word1 = ["abc", "d", "defg"], word2 = ["abcddefg"]

**Output:** true

**Constraints:**

1 <= word1.length, word2.length <= 103

1 <= word1[i].length, word2[i].length <= 103

1 <= sum(word1[i].length), sum(word2[i].length) <= 103

word1[i] and word2[i] consist of lowercase letters.

**Solution:**

**My Solution:**

public class O4\_StringEquivalent\_1662 {  
 public static void main(String[] args) {  
 Solution\_1662 solution\_1662 = new Solution\_1662();  
 String[] word1 = {"a", "cb"};  
 String[] word2 = {"ab", "c"};  
 System.*out*.println(solution\_1662.arrayStringsAreEqual(word1, word2));  
 }  
}  
  
class Solution\_1662 {  
 public boolean arrayStringsAreEqual(String[] word1, String[] word2) {  
 String r1 = "";  
 for (int i = 0; i < word1.length; i++) {  
 r1 += word1[i];  
 }  
 System.*out*.println(r1);  
 String r2 = "";  
 for (int i = 0; i < word2.length; i++) {  
 r2 += word2[i];  
 }  
 System.*out*.println(r2);  
  
 if (r1.equals(r2)) {  
 return true;  
 } else {  
 return false;  
 }  
 }  
}  
  
// <https://leetcode.com/problems/check-if-two-string-arrays-are-equivalent/>

**Another Solution: 01**

class Solution {  
 public boolean arrayStringsAreEqual(String[] word1, String[] word2) {  
 return String.*join*("", word1).equals(String.*join*("", word2));  
 }  
}

**Another Solution: 02**

class Solution {  
 public boolean arrayStringsAreEqual(String[] word1, String[] word2) {  
 StringBuilder sb1 = new StringBuilder(), sb2 = new StringBuilder();  
 for (String word : word1)  
 sb1.append(word);  
  
 for (String word : word2)  
 sb2.append(word);  
  
 return sb1.toString().equals(sb2.toString());  
 }  
}

**Another Solution: 03**

class Solution {  
 public boolean arrayStringsAreEqual(String[] word1, String[] word2) {  
 int idx1 = 0, idx2 = 0, arrIdx1 = 0, arrIdx2 = 0;  
 while (arrIdx1 < word1.length && arrIdx2 < word2.length) {  
 if (word1[arrIdx1].charAt(idx1) != word2[arrIdx2].charAt(idx2)) return false;  
 if (idx1 == word1[arrIdx1].length() - 1) {  
 idx1 = 0;  
 arrIdx1++;  
 } else idx1++;  
 if (idx2 == word2[arrIdx2].length() - 1) {  
 idx2 = 0;  
 arrIdx2++;  
 } else idx2++;  
 }  
 return arrIdx1 == word1.length && arrIdx2 == word2.length;  
 }  
}

**Another Solution: 04**

class Solution {  
 public boolean arrayStringsAreEqual(String[] word1, String[] word2) {  
 int idx1 = 0, idx2 = 0, arrIdx1 = 0, arrIdx2 = 0;  
 while (arrIdx1 < word1.length && arrIdx2 < word2.length) {  
 if (word1[arrIdx1].charAt(idx1) != word2[arrIdx2].charAt(idx2)) return false;  
 idx1 = (++idx1) % word1[arrIdx1].length();  
 idx2 = (++idx2) % word2[arrIdx2].length();  
 if (idx1 == 0) arrIdx1++;  
 if (idx2 == 0) arrIdx2++;  
 }  
 return arrIdx1 == word1.length && arrIdx2 == word2.length;  
 }  
}

**03. Problem: 1480. Running Sum of 1d array**

Given an array nums. We define a running sum of an array as runningSum[i] = sum(nums[0]…nums[i]).

Return the running sum of nums.

**Example 1:**

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

**Output:** [1,3,6,10]

**Explanation:** Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].

**Example 2:**

**Input:** nums = [1,1,1,1,1]

**Output:** [1,2,3,4,5]

**Explanation:** Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1].

**Example 3:**

**Input:** nums = [3,1,2,10,1]

**Output:** [3,4,6,16,17]

**Constraints:**

1 <= nums.length <= 1000

-10^6 <= nums[i] <= 10^6

**Solution:**

**My Solution:**

import java.util.Arrays;  
  
public class O3\_RunningSum1dArray\_1480 {  
 public static void main(String[] args) {  
 Solution\_1480 solution\_1480 = new Solution\_1480();  
 int[] nums = {1, 1, 1, 1, 1};  
 System.*out*.println(Arrays.toString(solution\_1480.runningSum(nums)));  
 int[] nums2 = {1, 2, 3, 4};  
 System.*out*.println(Arrays.toString(solution\_1480.runningSum(nums2)));  
 }  
}  
  
class Solution\_1480 {  
 public int[] runningSum(int[] nums) {  
 int[] result = new int[nums.length];  
 int sum = 0;  
 for (int i = 0; i < nums.length; i++) {  
 sum = sum + nums[i];  
 result[i] = sum;  
 }  
 return result;  
 }  
}  
  
 */\*\*  
 \* {1, 2, 3, 4}  
 \* <p>  
 \* Iteration: 1  
 \* sum = sum + num[i] | sum = 0 + 1 = 1  
 \* result[i] = sum | result[0] = 1  
 \* <p>  
 \* Iteration: 2  
 \* sum = 1 + 2 = 3 , result[1] = 3  
 \* <p>  
 \* Iteration: 3  
 \* sum = 3 + 3 = 6 , result[2] = 6  
 \* <p>  
 \* Iteration: 4  
 \* sum = 3 + 6 = 10 , result[3] = 10  
 \*/*// <https://leetcode.com/problems/running-sum-of-1d-array/>

**Another Solution: 01**

public int[] runningSum(int[] nums) {  
 // modify the input array, adding n[i] with n[i-1]  
 for (int i = 1; i < nums.length; i++) {  
 nums[i] += nums[i - 1];  
 }  
  
 // return the modified array  
 return nums;  
 }

**Another Solution: 02**

class Solution {  
 public int[] runningSum(int[] nums) {  
 return IntStream.range(0, nums.length).map(i -> i == 0 ? nums[i] : (nums[i] += nums[i - 1])).toArray();  
 }  
}

**Another Solution: 03**

**02. Problem: 9. Palindrome or not**

Given an integer x, return true if x is palindrome integer.

An integer is a **palindrome** when it reads the same backward as forward.

For example, 121 is a palindrome while 123 is not.

**Example 1:**

**Input:** x = 121

**Output:** true

**Explanation:** 121 reads as 121 from left to right and from right to left.

**Example 2:**

**Input:** x = -121

**Output:** false

**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

**Example 3:**

**Input:** x = 10

**Output:** false

**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

**Constraints:**

-231 <= x <= 231 - 1

**Follow up: Could you solve it without converting the integer to a string?**

**Solution:**

**My Solution:**

package O1\_easy;  
  
public class O2\_Palindrome\_9 {  
 public static void main(String args[]) {  
 Solution\_9 solution = new Solution\_9();  
 System.*out*.println(solution.isPalindrome(121));  
 System.*out*.println(solution.isPalindrome(123));  
 }  
}  
  
class Solution\_9 {  
 public boolean isPalindrome(int x) {  
 String s = String.*valueOf*(x);  
 StringBuffer sb = new StringBuffer(s);  
 String s1 = String.*valueOf*(sb.reverse());  
  
 if (s.equals(s1)) {  
 return true;  
 } else {  
 return false;  
 }  
 }  
}  
  
// <https://leetcode.com/problems/palindrome-number/>

**Another Solution: 01**

public class Solution {  
 public boolean isPalindrome(int x) {  
 if (x < 0) return false;  
 int y = x;  
 int res = 0;  
 while (y != 0) {  
 res = res \* 10 + y % 10;  
 y /= 10;  
 }  
 return x == res;  
 }  
}

**Another Solution: 02**

public boolean isPalindrome(int x){  
 int p=x,q=0;  
 while(p>=1){  
 q\*=10;  
 q+=p%10;  
 p/=10;  
 }  
 return q==x;  
 }

**Another Solution: 03**

**01. Problem: 2235. Add Two Integers**

**Given two integers num1 and num2, return the sum of the two integers.**

**Example 1:**

**Input:** num1 = 12, num2 = 5

**Output:** 17

**Explanation:** num1 is 12, num2 is 5, and their sum is 12 + 5 = 17, so 17 is returned.

**Example 2:**

**Input:** num1 = -10, num2 = 4

**Output:** -6

**Explanation:** num1 + num2 = -6, so -6 is returned.

**Constraints:**

-100 <= num1, num2 <= 100

**Solution:**

**My Solution:**

public class O1\_AddTwoIntegers\_2235 {  
 public static void main(String[] args) {  
 Solution s = new Solution();  
 System.*out*.println(s.sum(5, 8));  
 }  
}  
  
class Solution {  
 public int sum(int num1, int num2) {  
 return num1 + num2;  
 }  
}  
  
// <https://leetcode.com/problems/add-two-integers/>

**Another Solution: 01**

class Solution {  
 public int sum(int num1, int num2) {  
 if (num2 == 0) return num1;  
 int temp = (num1 & num2) << 1;  
 return sum(num1 ^ num2, temp);  
 }  
}

**Another Solution: 02**

**Another Solution: 03**

**Day 00-1929, , , , , .**

**5. Problem:**

**Solution:**

**My Solution:**

**Another Solution: 01**

**Another Solution: 02**

**Another Solution: 03**

**04. Problem:**

**Solution:**

**My Solution:**

**Another Solution: 01**

**Another Solution: 02**

**Another Solution: 03**

**03. Problem:**

**Solution:**

**My Solution:**

**Another Solution: 01**

**Another Solution: 02**

**Another Solution: 03**

**02. Problem:**

**Solution:**

**My Solution:**

**Another Solution: 01**

**Another Solution: 02**

**Another Solution: 03**

**01. Problem:**

**Solution:**

**My Solution:**

**Another Solution: 01**

**Another Solution: 02**

**Another Solution: 03**