Description | Editorial | Solutions (2.5K) | Submissions

1502. Can Make Arithmetic Progression From Sequence

Hint

Easy | 1.6K | 80

Companies

A sequence of numbers is called an **arithmetic progression** if the difference between any two consecutive elements is the same.

Given an array of numbers `arr`, return `true` *if the array can be rearranged to form an* ***arithmetic progression****. Otherwise, return* `false`.

Example 1:

```
Input: arr = [3,5,1]
Output: true
Explanation: We can reorder the elements as [1,3,5] or [5,3,1]
with differences 2 and -2 respectively, between each
consecutive elements.
```

Example 2:

```
Input: arr = [1,2,4]
Output: false
Explanation: There is no way to reorder the elements to obtain
an arithmetic progression.
```

Constraints:

- `2 <= arr.length <= 1000`
- `-10^6 <= arr[i] <= 10^6`

Java | Auto

```java
class Solution {
    public boolean canMakeArithmeticProgression(int[] arr) {
        if(arr.length==2)
            return true;

            Arrays.sort(arr);
            int diff = arr[1] - arr[0];
            for (int i=2; i<arr.length; i++){
                int currDiff=arr[i]-arr[i-1];
                if(currDiff!=diff)
                    return false;
            }
            return true;
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

arr =
[3,5,1]

Output

true

Expected

true

Console | Run | Submit

Description | Editorial | Solutions (12.9K) | Submissions

88. Merge Sorted Array

Hint

Easy 10.7K 1K

Companies

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to `0` and should be ignored. `nums2` has a length of `n`.

Example 1:

```
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined
elements coming from nums1.
```

Example 2:

```
Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and [].
The result of the merge is [1].
```

Example 3:

```
Input: nums1 = [0], m = 0, nums2 = [1], n = 1
Output: [1]
```

Java | Auto

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        for (int j = 0, i = m; j < n; j++) {
            nums1[i] = nums2[j];
            i++;
        }
        Arrays.sort(nums1);
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

nums1 =
[1,2,3,0,0,0]

m =
3

nums2 =
[2,5,6]

Console | Run | Submit

Description | Editorial | Solutions (7.1K) | Submissions

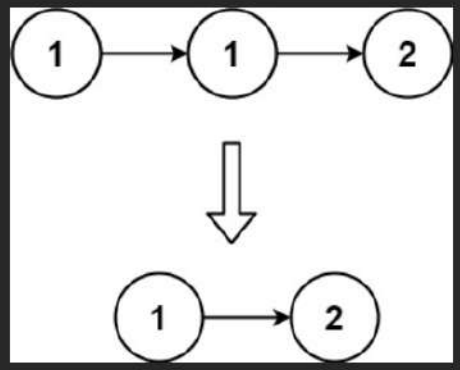
83. Remove Duplicates from Sorted List

Easy | 7.4K | 250

Companies

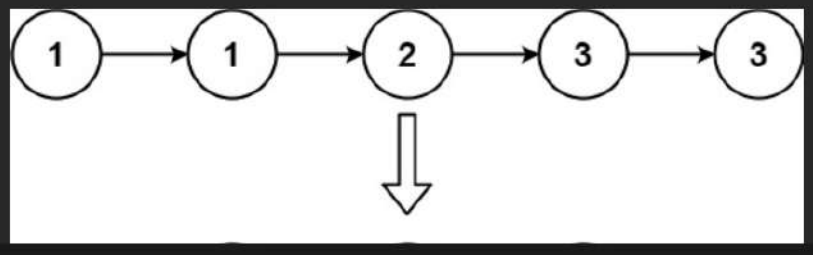
Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once.* Return *the linked list* ***sorted*** *as well.*

Example 1:




```
Input: head = [1,1,2]
Output: [1,2]
```

Example 2:



Java | Auto

```
 9  * }
10  */
11 class Solution {
12     public ListNode deleteDuplicates(ListNode head) {
13         ListNode temp = head;
14         while (temp != null && temp.next != null)
15         {
16             if (temp.next.val==temp.val)
17             {
18                 temp.next=temp.next.next;
19                 continue;
20             }
21             temp=temp.next;
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

head =
[1,1,2]

Output

[1,2]

Expected

[1,2]

Console | Run | Submit