

Remove Duplicates (easy)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
 - Code
 - Time Complexity
 - Space Complexity
- Similar Questions

Problem Statement

Given an array of sorted numbers, **remove all duplicates** from it. You should **not use any extra space**; after removing the duplicates in-place return the length of the subarray that has no duplicate in it.

Example 1:


```
Input: [2, 3, 3, 3, 6, 9, 9]
Output: 4
Explanation: The first four elements after removing the duplicates will be [2, 3, 6, 9].
```


Example 2:


```
Input: [2, 2, 2, 11]
Output: 2
Explanation: The first two elements after removing the duplicates will be [2, 11].
```


Try it yourself

Try solving this question here:

 Java

 Python3

 JS


 C++

```
1 def remove_duplicates(arr):
2     # TODO: Write your code here
3     return -1
4
```

Test

Save


Reset

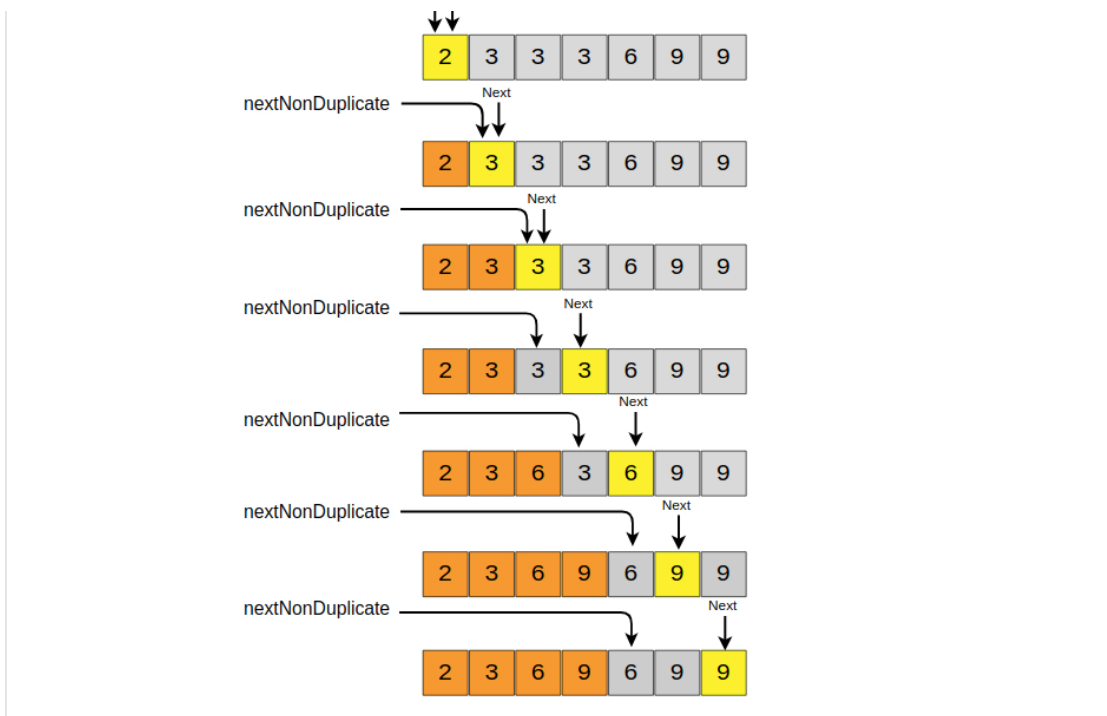


Solution

In this problem, we need to remove the duplicates in-place such that the resultant length of the array remains sorted. As the input array is sorted, therefore, one way to do this is to shift the elements left whenever we encounter duplicates. In other words, we will keep one pointer for iterating the array and one pointer for placing the next non-duplicate number. So our algorithm will be to iterate the array and whenever we see a non-duplicate number we move it next to the last non-duplicate number we've seen.

Here is the visual representation of this algorithm for Example-1:

nextNonDuplicate  Next



Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```

1 def remove_duplicates(arr):
2     # index of the next non-duplicate element
3     next_non_duplicate = 1
4
5     i = 1
6     while(i < len(arr)):
7         if arr[next_non_duplicate - 1] != arr[i]:
8             arr[next_non_duplicate] = arr[i]
9             next_non_duplicate += 1
10        i += 1
11
12    return next_non_duplicate
13
14
15 def main():
16     print(remove_duplicates([2, 3, 3, 3, 6, 9, 9]))
17     print(remove_duplicates([2, 2, 2, 11]))
18
19
20 main()
21

```

Run

Save

Reset

Time Complexity

The time complexity of the above algorithm will be $O(N)$, where 'N' is the total number of elements in the given array.

Space Complexity

The algorithm runs in constant space $O(1)$.

Similar Questions

Problem 1: Given an unsorted array of numbers and a target 'key', remove all instances of 'key' in-place and return the new length of the array.

Example 1•

Example 1:

Input: [3, 2, 3, 6, 3, 10, 9, 3], Key=3
Output: 4
Explanation: The first four elements after removing every 'Key' will be [2, 6, 10, 9].

Example 2:

Input: [2, 11, 2, 2, 1], Key=2
Output: 2
Explanation: The first two elements after removing every 'Key' will be [11, 1].

Solution: This problem is quite similar to our parent problem. We can follow a two-pointer approach and shift numbers left upon encountering the 'key'. Here is what the code will look like:

Java Python3 C++ JS

```
1 def remove_element(arr, key):
2     nextElement = 0 # index of the next element which is not 'key'
3     for i in range(len(arr)):
4         if arr[i] != key:
5             arr[nextElement] = arr[i]
6             nextElement += 1
7
8     return nextElement
9
10
11 def main():
12     print("Array new length: " +
13           str(remove_element([3, 2, 3, 6, 3, 10, 9, 3], 3)))
14     print("Array new length: " +
15           str(remove_element([2, 11, 2, 2, 1], 2)))
16
17
18 main()
19
```

Run Save Reset

Time and Space Complexity: The time complexity of the above algorithm will be $O(N)$, where 'N' is the total number of elements in the given array.

The algorithm runs in constant space $O(1)$.