Remove Duplicates (easy)

We'll cover the following

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++

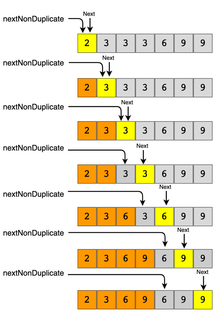
```
class RemoveDuplicates {

  public static int remove(int[] arr) {
    // TODO: Write your code here
    return -1;
  }
}

```

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:



Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
class RemoveDuplicates {

  public static int remove(int[] arr) {
    int nextNonDuplicate = 1; // index of the next non-duplicate element
    for (int i = 1; i < arr.length; i++) {
      if (arr[nextNonDuplicate - 1] != arr[i]) {
        arr[nextNonDuplicate] = arr[i];
        nextNonDuplicate++;
      }
    }

    return nextNonDuplicate;
  }

  public static void main(String[] args) {
    int[] arr = new int[] { 2, 3, 3, 3, 6, 9, 9 };
    System.out.println(RemoveDuplicates.remove(arr));

    arr = new int[] { 2, 2, 2, 11 };
    System.out.println(RemoveDuplicates.remove(arr));
  }
}

```

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:

```
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

```
class RemoveElement {

  public static int remove(int[] arr, int key) {
    int nextElement = 0; // index of the next element which is not 'key'
    for (int i = 0; i < arr.length; i++) {
      if (arr[i] != key) {
        arr[nextElement] = arr[i];
        nextElement++;
      }
    }

    return nextElement;
  }

  public static void main(String[] args) {
    int[] arr = new int[] { 3, 2, 3, 6, 3, 10, 9, 3 };
    System.out.println(RemoveElement.remove(arr, 3));

    arr = new int[] { 2, 11, 2, 2, 1 };
    System.out.println(RemoveElement.remove(arr, 2));
  }
}

```

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 $O(1)$.