**Two-pointer Technique - Scenario II**

[Report Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

Sometimes, we can use two pointers with different steps to solve problems.

*An Example*

Let's start with another classic problem:

Given an array and a value, remove all instances of that value [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) and return the new length.

If we don't have the limitation of space complexity, it will be easier. We can initialize a new array to store the answer. Iterate the original array and add the element to the new array if the element is not equal to the given target value.

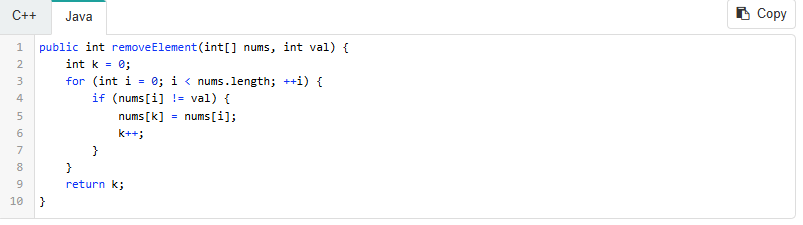
Actually, it is equivalent to using two pointers, one is used for the iteration of the original array and another one always points at the last position of the new array.

*Reconsider the Space Limitation*

Now let's reconsider the space limitation.

We can use a similar strategy. We still use two pointers: one is still used for the iteration while the second one always points at the position for next addition.

Here is the code for your reference:



We use two pointers, one faster-runner i and one slower-runner k, in the example above. i moves one step each time while k moves one step only if a new needed value is added.

*Summary*

This is a very common scenario of using the two-pointer technique when you need:

One slow-runner and one fast-runner at the same time.

The key to solving this kind of problems is to

Determine the movement strategy for both pointers.

Similar to the previous scenario, you might sometimes need to sort the array before using the two-pointer technique. And you might need a greedy thought to determine your movement strategy.