Exercise 5.1 Reverse Integers

Array is a very common data structure in Java and loops are very often used to handle data stored in an array. Similar to the advice in Exercise 4.2, summarising an appropriate pattern is very important and difficult for beginner.

Problem Description:

Write a Java method void reverseInts(int[] nums) that reverse the values within the input array.

You must reverse the values in-place by swapping and not create a new array.

Key information:

Reverse values

Swapping instead of creating a new array

Solution design

The best way to find patterns is still through simulations. Remember that the exact values in array do not affect the design of solution in **this** problem.

- Simulation 1: int[] $nums = \{1\}$

We don't need to reverse it because it has only one value

- Simulation 2: int[] $nums = \{1,2\}$

The result should be $nums = \{2,1\}$. We might need some graphs to demonstrate it.

In below figure, we have an array $nums$ with a length of two. And nums[0]=1 and nums[1]=2.

Our goal is to let $nums[0]$ be 2 and let $nums[1]$ be 1. This task is very easy if we carry it out on paper with pens. But it takes some thoughts on programming.

The most direct way is to carry out the following lines of code.

$$nums[0] = 2;$$

$$nums[1] = 1;$$

However, we usually don't know the exact numerical values in an array. So we cannot assign numerical value straight value to it. But we can initialize an integer variable and store a value in it! For example

$$int\ value0 = nums[0];$$

$$int\ value1 = nums[1];$$

$$nums[0] = value1;$$

$$nums[1] = value0;$$

If you are imaginative (I would not say this is positive), you might find out we only need the next two lines of code to swap values. However, this might be conceptually correct in some ways (which I am not sure where we can find for now).

$$nums[0] = nums[1];$$

$$nums[1] = nums[0];$$

Above lines fail to reach our objective in the domain of programming. In java programming language, our programs run consecutive lines of code sequentially. Let us simulate the effect of above two lines of code.

Step 1: int[] $nums = \{1,2\}$

Step 2: $nums[0] = nums[1]$

Step 3:$nums[1] = nums[0]$

The problem lies in step 2. The new value assigned to $nums[0]$ covers its old value. Thus step 3 is only assigning the value of $nums[1]$ to itself.

Thus, we need one more variable to store the old value of $nums[0]$ before its value is replaced.

$$int\ temp = nums[0];$$

$$nums[0] = nums[1];$$

$$nums[1] = temp;$$

Step 1: int[] $nums = \{1,2\}$

Step 2: $int\ temp = nums[0];$

Step 3: $nums[0] = nums[1]$

Step 4:$nums[1] = nums[0]$

Before we move to last two examples, we can conclude the logic to swap the value of two variables. For any two integer variables $num1, num2$, we only need one more variable to swap the value of $num1, num2$.

$$int\ temp = num1;$$

$$num1 = 2;$$

$$num2 = temp;$$

Previous examples are trivial and things get complicated from now. Once the length of array is larger than 2, we have to consider the swap order and decide which two variables are about to swap.

- Simulation 3: $nums = \{1,2,3\}$

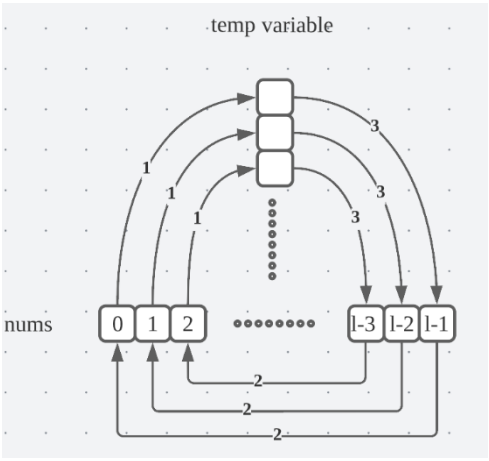
Expected results: $nums = \{3,2,1\}$

- Simulation 4: $nums = \{1,2,3,4\}$

Expected results: $nums = \{4,3,2,1\}$

To reverse the order, we need to swap the value of first variable and the last variable. Then, swap the value of second variable and the second last variable. And so on.

Generalize the process:



For an array with a considerable large number length, we draw a figure of process. The number inside each box is the index, not the actual value. l is the length of array.

From the process, we can find patterns easily. We can start our loop from swapping $nums[0]$ and $nums[l-1]$, then $nums[1]$ and $nums[l-2]$ ….., $nums[i]$ and $nums[l-1-i]$. But don't forget when to stop. Otherwise, if i$= l-1$,

then we are going to cancel the first swap.

For array with even number length, the index of the middle two value is $\frac{l}{2}$-1 and $\frac{l}{2}$ (index from 0). So, we stop when $i < \frac{l}{2}$.

For array with odd number length, the index of the middle value is $\frac{l-1}{2}$. Now, does above stop condition applies now? The answer is yes! i is an integer and the max value of i for $i < \frac{l}{2}$ is $\frac{l-1}{2}$. And, when $i = \frac{l-1}{2}$, we are swapping value of $nums\left[\frac{l-1}{2}\right]$ and $nums\left[\frac{l-1}{2}\right]$. This swap will not disturb the expected order and we just leave the program to do this swap(maybe meaningless but it is fine). Double check whether the pattern applies to example 1 and 2.

Now we have summerised the initial value, loop condition, increment and inner pattern. Start coding.

Check implementation in this repo

More exercises

1. Reverse values with even indices.
2. Reverse values with odd indices.