Array Rotation Solution with formula

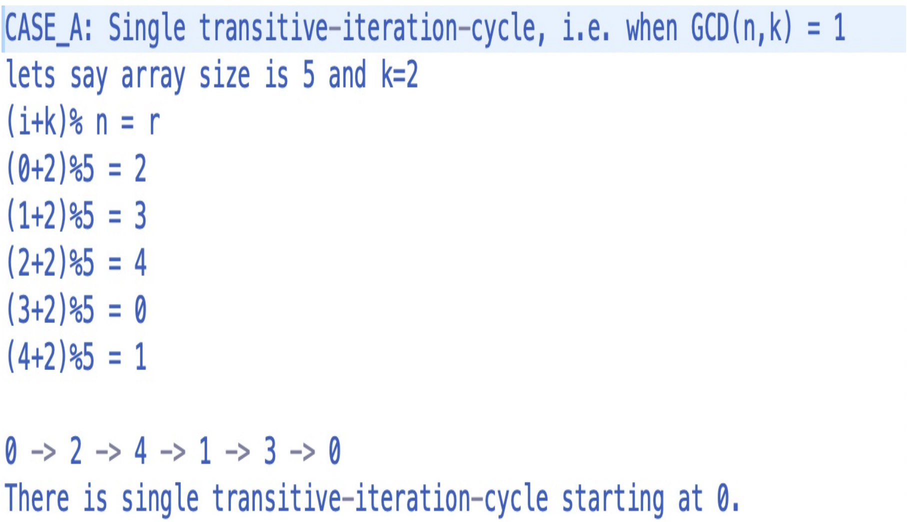
As we know, how we translate 1-d array index into 2-d array index

row_index = (1d_index) / 2d_arr[0].length
col_index = (1d_index) % 2d_arr[0].length

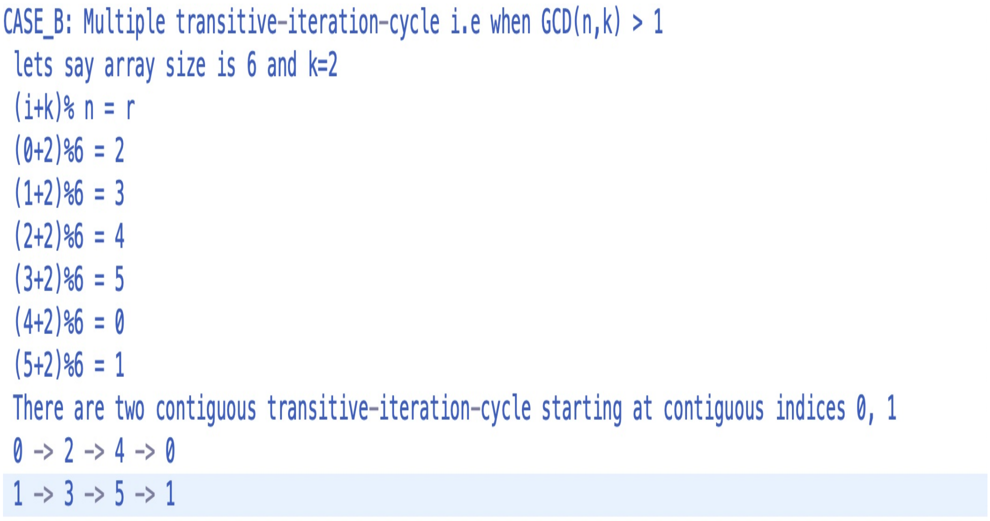
And vice versa: 1d_index = row_index * 2d_arr[0].length + col_index

```
Post right rotation array will follow the col_index formula as it is similar to occupy space in
next row at kth colum.

        (i+k) % n = col_index
         (i+k) % n = r

        (0+k)%n  = k
        (1+k)%n  = k+1
        (2+k)%n  = k+2
        . .. ... ..
        (x+k)%n  = 0; when x+k = n
        . .. ... ..
        (n-1+k)%n  = k-1;
```

```
formula : r =  (i+k)%n
i - represents index of array
k - represents rotation count
n - represents size of the array
r - represents index post rotation
```



```
CASE_A: Single transitive-iteration-cycle, i.e. when GCD(n,k) = 1
lets say array size is 5 and k=2
(i+k)% n = r
(0+2)%5 = 2
(1+2)%5 = 3
(2+2)%5 = 4
(3+2)%5 = 0
(4+2)%5 = 1

0 -> 2 -> 4 -> 1 -> 3 -> 0
There is single transitive-iteration-cycle starting at 0.
```



```
CASE_B: Multiple transitive-iteration-cycle i.e when GCD(n,k) > 1
 lets say array size is 6 and k=2
 (i+k)% n = r
 (0+2)%6 = 2
 (1+2)%6 = 3
 (2+2)%6 = 4
 (3+2)%6 = 5
 (4+2)%6 = 0
 (5+2)%6 = 1
 There are two contiguous transitive-iteration-cycle starting at contiguous indices 0, 1
 0 -> 2 -> 4 -> 0
 1 -> 3 -> 5 -> 1
```

This method works by rotating an array in-place by using cyclic replacements. This approach involves repeatedly moving elements to their correct positions one step at a time until all elements are in their correct positions.

Juggling algorithm for array-rotation

Total number of transitive-iteration-cycles : GCD(n,k)
Total elements in a transitive-iteration-cycle : n/GCD(n,k)
Each transitive-iteration-cycle is mutually exclusive.

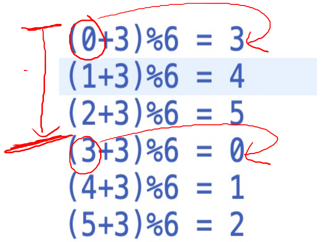
Why is it that once we finish a cycle, we start the new cycle from the next element i.e. can't the next element be already a part of a processed cycle?
https://leetcode.com/problems/rotate-array/solutions/259418/clear-cyclic-replacement-java-solution-with-proof-o-n-in-time-o-1-in-space/

Time complexity: O(n), because loop is getting iterated only once, that is we scan each element once combined inner+outer loop.

```
formula : r =  (i+k)%n
i - represents index of array
k - represents rotation count
n - represents size of the array
r - represents index post rotation
```

Let's try to figure-out iteration pattern for case GCD(n,k) > 1

3 iterations cycles starting at 0,1,2

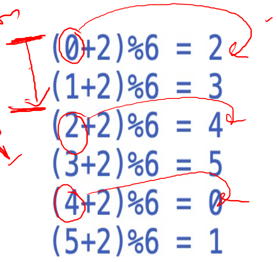


```
(0+3)%6 = 3
(1+3)%6 = 4
(2+3)%6 = 5
(3+3)%6 = 0
(4+3)%6 = 1
(5+3)%6 = 2
```

Here, there will be 3 transitive-iteration cycles which covers all the elements:
1. 0 -> 3 : 3 -> 0
2. 1 -> 4 : 4 -> 1
3. 2 - >5 : 5 -> 2

Note: Iteration cycle start nodes are sequential. 0,1,2

Two iteration cycles starting at 0,1

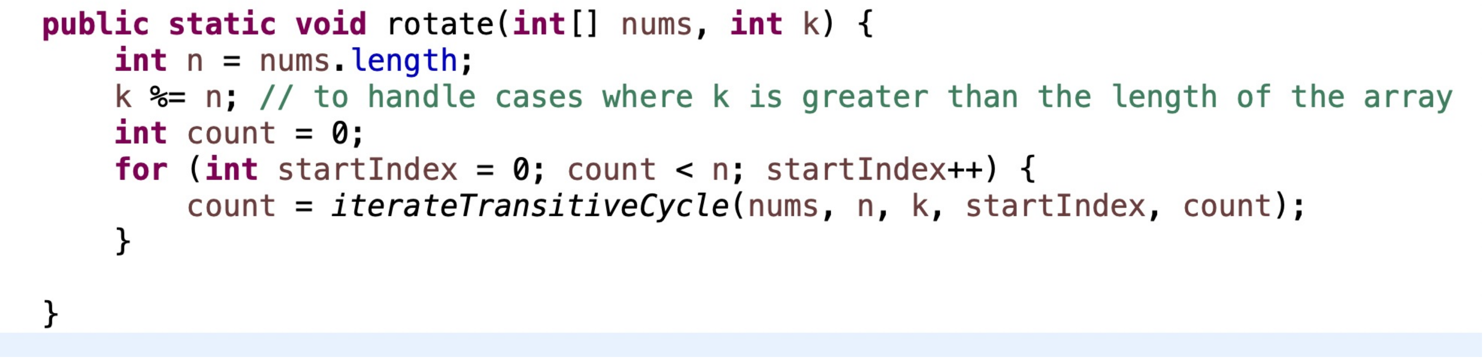


```
(0+2)%6 = 2
(1+2)%6 = 3
(2+2)%6 = 4
(3+2)%6 = 5
(4+2)%6 = 0
(5+2)%6 = 1
```

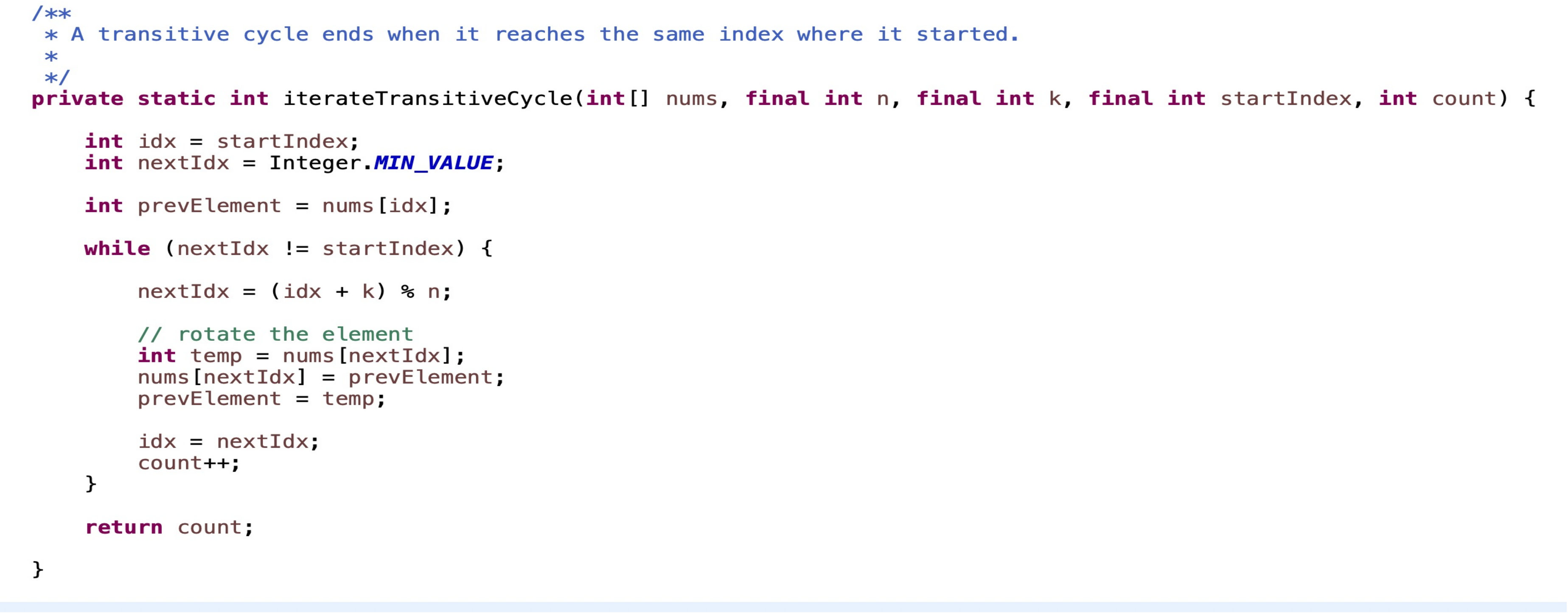
Here, there will be 2 transitive-iteration cycles which covers all the elements:
1. 0 -> 2 : 2 -> 4 : 4 -> 0
2. 1 -> 3 : 3 -> 5 : 5 -> 1

Note: Iteration cycle start nodes are sequential. 0,1

So, finally for case when GCD(n,k) > 1, outer loop needs to start the transitive-iteration clycles in sequence until we reach to the total count of elements. Since, for case when GCD(n,k) = 1 there will be only single transitive-iteration cycle, means we need to put the total count calculation for inner loop as well.



```java
  public static void rotate(int[] nums, int k) {
      int n = nums.length;
      k %= n; // to handle cases where k is greater than the length of the array
      int count = 0;
      for (int startIndex = 0; count < n; startIndex++) {
          count = iterateTransitiveCycle(nums, n, k, startIndex, count);
      }

  }
```



```java
/**
 * A transitive cycle ends when it reaches the same index where it started.
 *
 */
private static int iterateTransitiveCycle(int[] nums, final int n, final int k, final int startIndex, int count) {

    int idx = startIndex;
    int nextIdx = Integer.MIN_VALUE;

    int prevElement = nums[idx];

    while (nextIdx != startIndex) {

        nextIdx = (idx + k) % n;

        // rotate the element
        int temp = nums[nextIdx];
        nums[nextIdx] = prevElement;
        prevElement = temp;

        idx = nextIdx;
        count++;
    }

    return count;

}
```