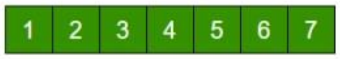
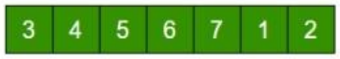
Array Rotation

As the term rotation signifies, array rotation means to rotate the elements of an array by given positions.

Consider the below array:



The above array is rotated counter-clockwise(towards left) by 2 elements. After rotation, the array will be:



Visually, the process of counter clock-wise array rotation(rotated by say K elements) looks like:

- Shift all elements after K-th element to the left by K positions.
- Fill the K blank spaces at the end of the array by first K elements from the original array.

Note: The similar approach can also be applied for clockwise array rotation.

Implementations

- **Simple Method**: The simplest way to rotate an array is to implement the above visually observed approach by using extra space.
    1. Store the first K elements in a temporary array say temp[].
    2. Shift all elements after K-th element to the left by K positions in the original array.
    3. Fill the K blank spaces at the end of the original array by the K elements from the temp array.

```
Say, arr[] = [1, 2, 3, 4, 5, 6, 7], K = 2
1) Store first K elements in a temp array
   temp[] = [1, 2]
2) Shift rest of the arr[]
   arr[] = [3, 4, 5, 6, 7, 6, 7]
3) Store back the K elements from temp
   arr[] = [3, 4, 5, 6, 7, 1, 2]
```

  Time Complexity: O(N), where N is the number of elements in the array.
  Auxiliary Space: O(K) where K is the number of places by which elements will be rotated.

- **Another Method (Without extra space)**: We can also rotate an array by avoiding the use of temporary array. The idea is to rotate the array one by one K times.

```
leftRotate(arr[], d, n)
start
  For i = 0 to i < d
    Left rotate all elements of arr[] by one
end
```

To rotate an array by 1 position to the left:

1. Store the first element in a temporary variable say temp.
2. Left shift all elements after the first element by 1 position. That is, move arr[1] to arr[0], arr[2] to arr[1] and so on.
3. Initialize arr[N-1] with temp.

To rotate an array by K position to the left, repeat the above process K times.

Take the same example,

```
arr[] = [1, 2, 3, 4, 5, 6, 7], K = 2

Rotate arr[] one by one 2 times.

After 1st rotation: [2, 3, 4, 5, 6, 7, 1]
After 2nd rotation: [ 3, 4, 5, 6, 7, 1, 2]
```

Time Complexity: O(N*K), where N is the number of elements in the array and K is the number of places by which elements will be rotated.
Auxiliary Space: O(1).

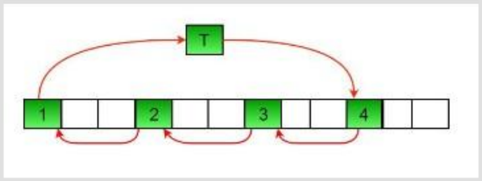
- **Juggling Algorithm**: This is an extension of the above method. Instead of moving one by one, divide the array in different sets, where number of sets is equal to GCD of N and K and move the elements within sets.

  If GCD is 1 as is for the above example array (N = 7 and K = 2), then elements will be moved within one set only, we just start with temp = arr[0] and keep moving arr[I+d] to arr[I] and finally store temp at the right place.

  Here is an example for N = 12 and K = 3. GCD of N and K is 3:

```
Let arr[] be {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

a) Elements are first moved in first set – (See below
   diagram for this movement)
```



```
      arr[] after this step --> {4 2 3 7 5 6 10 8 9 1 11 12}

b) Then in second set.
      arr[] after this step --> {4 5 3 7 8 6 10 11 9 1 2 12}

c) Finally in third set.
      arr[] after this step --> {4 5 6 7 8 9 10 11 12 1 2 3}
```

Time Complexity: O(N), where N is the number of elements in the array.
Auxiliary Space: O(1).