We will be implementing a mixing strategy that depends on the data itself. Given a sequence of integers (which may be positive, negative, or zero), we wish to process the numbers according to their **original order** but in the current configuration. Each number moves to the right or left in the container according to its value (right is positive, left is negative). A move of zero changes **nothing** in the container. Numbers cycle to the front or back of the container when necessary. When a number ends up between the last and first element, we place it as the **last** element. For example, given the starting values `{1, 0, 4, -1}`, we have the following steps:

1. Move 1 (which goes to the right by 1):
    - `{  0,  1,  4, -1 }`
2. Move 0 (which goes nowhere):
    - `{  0,  1,  4, -1 }`
3. Move 4 (which goes to the right by 4 which means it cycles around to the last position):
    - `{  0,  1, -1,  4 }`
4. Move -1 (which goes to the **left** by 1):
    - `{  0, -1,  1,  4 }`

Write a `mixNumbers` function in `solution.h` to accomplish this. This function takes two ForwardIterators and mixes the values **in place** (like STL's sort). Some things to note:

1. If a value ends up at the beginning or end of the list, the cyclic nature of the list means it is ambiguous which end of the list it should be placed at (the very beginning or the very end). Our rule is that the final position should never be the **first** entry but always the last. However, if `0` is at the beginning of the list, it does **not** move to the end. Thus mixing `{2, 0, 0}` should be `{0, 0, 2}`, not `{2, 0, 0}`, but mixing `{0, 1, 0}` becomes `{0, 0, 1}`, not `{0, 1, 0}` (the zero doesn't shift to the end of the list).
2. You cannot depend on the values themselves to tell you the correct position because there may be duplicates. Consider mixing `{3, 0, 3, 3, 0}`. The process goes `{3, 0, 3, 3, 0}` to `{0, 3, 3, 3, 0}` to `{0, 3, 3, 0, 3}` to `{0, 3, 0, 3, 3}` to `{0, 3, 0, 3, 3}`. However, it will not be clear which number is being considered at each step without keeping track of the original order with respect to the current positions.