**Problem Name:** Pancake sorting

**Topics:** Array, Two-Pointers, Greedy, Sorting

**Companies:** Microsoft, Google

**Level:** Medium

**Language:** C++

**Problem Statement:** Given an array of integers arr, sort the array by performing a series of pancake flips.

In one pancake flip we do the following steps:

* Choose an integer k where 1 <= k <= arr.length.
* Reverse the sub-array arr[0...k-1] (**0-indexed**).

For example, if arr = [3,2,1,4] and we performed a pancake flip choosing k = 3, we reverse the sub-array [3,2,1], so arr = [1,2,3,4] after the pancake flip at k = 3.

Return *an array of the*k*-values corresponding to a sequence of pancake flips that sort*arr. Any valid answer that sorts the array within 10 \* arr.length flips will be judged as correct.

**Input Format:**

First line of the input contains integer n (length of array)

Second line contain n space separated integer.

Ex:

4

3 2 4 1

**Output Format:** Print an array of the k-values corresponding to a sequence of pancake flips that sort arr. Any valid answer that sorts the array within 10 \* arr.length flips will be judged as correct.

**Constraints:**

* 1 <= arr.length <= 100
* 1 <= arr[i] <= arr.length
* All integers in arr are unique (i.e. arr is a permutation of the integers from 1 to arr.length).

**Examples:**

**Input:** arr = [3,2,4,1]

**Output:** [4,2,4,3]

**Explanation:**

We perform 4 pancake flips, with k values 4, 2, 4, and 3.

Starting state: arr = [3, 2, 4, 1]

After 1st flip (k = 4): arr = [1, 4, 2, 3]

After 2nd flip (k = 2): arr = [4, 1, 2, 3]

After 3rd flip (k = 4): arr = [3, 2, 1, 4]

After 4th flip (k = 3): arr = [1, 2, 3, 4], which is sorted.

**Brute force Solution:**

Find the index i of the next maximum number x.  
Reverse i + 1 numbers, so that the x will be at A[0]  
Reverse x numbers, so that x will be at A[x - 1].  
Repeat this process N times.

**Update:**  
Actually, I didn't use the condition permutation of [1,2,..., A.length].  
I searched in the descending order of A.

**Explanation:**

**Code:**

**Time Complexity**: O(N2)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:**

**Code:**

**Time Complexity**: O(N)

**Space Complexity:** O(1)