969. Pancake Sorting

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.

Example 1:

```
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.
```

Example 2:

```
Input: arr = [1,2,3]
Output: []
Explanation: The input is already sorted, so there is no need to flip
anything.
Note that other answers, such as [3, 3], would also be accepted.
```

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`).

The idea is the following: first, put pancake with the biggest index on its place, then we never need to move it! Let us go through example:

`[3,2,4,6,5,1]`.

1. We want to put pancake number `6` to the end, we can no do it immediatly, so let us put it to the beginning first: we need to flip first 4 pancakes: `A = [6,4,2,3,5,1]`. On the next step we can flip first 6 pancakes, so we have `A = [1,5,3,2,4,6]`.
2. Now, we want to put pancake number `5` to its place, we first flip first 2 pancakes to have `[5,1,3,2,4,6]` and then first 5, to have `[4,2,3,1,5,6]`.
3. Similar logic with number `4`, but it is already in the beginning, so one step is enough: we flip first 4 pancakes to have `[1,3,2,4,5,6]`.
4. Put `3` to its place: flip first 2 to have `[3,1,2,4,5,6]` and then flip first 3 to have `[2,1,3,4,5,6]`.
5. Finally, put `2` on its place, flit first 2: we have `[1,2,3,4,5,6]`.

So, our filps are the following: [4,6,2,5,4,2,3,2].

Comlexity: this is interesting part, we need to compute two types of complexities: classical one and how many flips we make. Note, that we make `O(n)` flips, because at each step we need to make no more than `2`. Now, note, that on each step we make no more than `O(n)` operations, because we need to inverse some part of array. So, overall complexity will be `O(n^2)`. Space complexity is `O(n)` to keep our answer.

```python
class Solution:
    def pancakeSort(self, A):
        result, n = [], len(A)
        for i in range(n,0,-1):
            pl = A.index(i)
            if pl == i-1: continue
            if pl != 0:
                result.append(pl+1)
                A[:pl+1] = A[:pl+1][::-1]
            result.append(i)
            A[:i] = A[:i][::-1]

        return result
```