# LeetCode 801

https://leetcode.com/problems/minimum-swaps-to-make-sequences-increasing/description/

Yifeng Zeng

# Description

801. Minimum Swaps To Make Sequences Increasing

We have two integer sequences A and B of the same non-zero length.

We are allowed to swap elements A[i] and B[i]. Note that both elements are in the same index position in their respective sequences.

At the end of some number of swaps, A and B are both strictly increasing. (A sequence is strictly increasing if and only if A[0] < A[1] < A[2] < … < A[A.length - 1].)

Given A and B, return the minimum number of swaps to make both sequences strictly increasing. It is guaranteed that the given input always makes it possible.

```
Example:
Input: A = [1,3,5,4], B = [1,2,3,7]
Output: 1
Explanation:
Swap A[3] and B[3].  Then the sequences are:
A = [1, 3, 5, 7] and B = [1, 2, 3, 4]
which are both strictly increasing.
```

Note:

- A, B are arrays with the same length, and that length will be in the range [1, 1000].
- A[i], B[i] are integer values in the range [0, 2000].

# Idea Report

For a minimum/maximum problem we can try DP. At each index i, we only have two options: swap or not swap. So we can construct a 2-D array to save the minimum swaps needed at index i. We can have a int[] swap, for which swap[i] means the minimum swaps needed at index i and we swap the number at i. We have another int[] noswap, for which noswap[i] means the minimum swaps needed at index i and we do not swap the number at i. The base case is swap[0] = 1, noswap[0] = 0, the meaning is self explained. For the DP function, because "It is guaranteed that the given input always makes it possible", there are two different situations:

1. Current number A[i] and B[i] are at correct position and no need to swap (A[i] > A[i-1] && B[i] > B[i-1]). So swap[i] = swap[i-1] + 1, we add 1 swap because at last index, we already swapped the number, so we need to swap again at this index, so add 1. And noswap[i] = noswap[i-1], because we did not swap at last index, so we don't have to swap at current index.
2. Current number A[i] and B[i] are not at correct position and we do need a swap. So swap[i] = noswap[i-1] + 1, we add 1 swap because at last index we didn't swap, then we need to swap at current index. And noswap[i] = swap[i-1] because we already swapped at last index, so we don't have to swap at current index. Of course we need to get the minimum number of each swap[i] and noswap[i].

Code:

```
class Solution {
    public int minSwap(int[] A, int[] B) {
        int len = A.length;
        int[] swap = new int[len];
        int[] noswap = new int[len];
        swap[0] = 1;
        for (int i = 1; i < len; i++) {
            swap[i] = len;
            noswap[i] = len;
            if (A[i - 1] < A[i] && B[i - 1] < B[i]) {
                swap[i] = swap[i - 1] + 1;
                noswap[i] = noswap[i - 1];
            }
            if (A[i - 1] < B[i] && B[i - 1] < A[i]) {
                swap[i] = Math.min(noswap[i - 1] + 1, swap[i]);
                noswap[i] = Math.min(noswap[i], swap[i - 1]);
            }
        }

        return Math.min(swap[len - 1], noswap[len - 1]);
    }
}
```

# Summary

- Discuss the state prepresentation and state tranfer function.