Daily Coding Problem #240

Problem

This problem was asked by Spotify.

There are `N` couples sitting in a row of length `2 * N`. They are currently ordered randomly, but would like to rearrange themselves so that each couple's partners can sit side by side.

What is the minimum number of swaps necessary for this to happen?

Solution

Suppose the couples are labeled `(0, 1), (2, 3), (4, 5), ...`, and they are sitting on consecutive couches in the following arrangement: `[(0, 4), (1, 3), (2, 5), ...]`.

Then for the first pair, there are two ways to make a swap such that a couple will end up together on the first couch. That is, we can either swap person `4` with person `1`, or swap person `0` with person `5`.

For either option, we have possibly two options for creating a couple on the second couch, and then possibly two options for creating a couple on the third couch, and so on. (We say possibly, because it could be the case that the couch already has a couple sitting on it). So we could recursively go through each of these options, tracking the total number of swaps, and at the end return the minimum.

```
def swap(row, i, positions):
    """
    Find the position  where the partner of the ith person (either the first or second) is
    currently sitting. Then swap the other person in the pair with this partner.
    """
```

```
    if row[i] % 2 == 0:
        partner = positions[row[i] // 2 * 2 + 1]
    else:
        partner = positions[row[i] // 2 * 2]

    row[1 - i], row[partner] = row[partner], row[1 - i]

    return row

def min_swaps_helper(row, positions):
    """
    If there are no couples to pair up, return 0. Otherwise, find both options for placing
    a couple on the first couch, shift everyone down a couch, and proceed recursively.
    At each stage, return one plus the better of the two options.
    """
    if len(row) == 0:
        return 0

    elif row[0] // 2 == row[1] // 2:
        return min_swaps_helper(row[2:], positions)

    else:
        first = swap(row, 0, positions)[2:]
        second = swap(row, 1, positions)[2:]
        positions = [i - 2 for i in positions]

    return 1 + min(min_swaps_helper(first, positions), min_swaps_helper(second, positions))

def min_swaps(row):
    n = len(row)

    positions = [0 for i in range(n)]
    for i in range(n):
        positions[row[i]] = i

    return min_swaps_helper(row, positions)
```

This would take $O(2^N)$, since we branch off into two paths at every couch. We also use $O(2^N)$ space, since each time we branch we create two new rows.

However, sometimes simpler is better, and in this case, a greedy solution is actually optimal. Note that if we look at the people sitting on the first couch, one of the following must be true:

- We can make one swap to set both couples right, as in the following: `(0, 3), (1, 2), ..., (6, 8)`.
- Only one couple can be fixed with one swap, as in `(0, 3), (1, 5), ... (2, 8)`.

In either case, we cannot do better than choosing the first person from the pair, finding the position of their partner, and swapping the second person with their partner. Our greedy algorithm, then, will simply go through each pair in the list and perform this swap if the pair does not already consist of a proper couple.

```
def min_swaps(row):
    n = len(row)

    positions = [0 for i in range(n)]
    for i in range(n):
        positions[row[i]] = i

    swaps = 0

    for i in range(0, n, 2):
        if row[i] // 2 == row[i + 1] // 2:
            continue

        if row[i] % 2 == 0:
            partner = positions[row[i] // 2 * 2 + 1]
        else:
            partner = positions[row[i] // 2 * 2]

        row[i + 1], row[partner] = row[partner], row[i + 1]
        swaps += 1

    return swaps
```

This solution is `O(N)` in both time and space.