

① **Initial state:** A sequence of n numbers a_1, a_2, \dots, a_n is given. You are allowed to perform the following operation: choose two indices i and j such that $i < j$, swap a_i and a_j , and then move the sequence $a_1, a_2, \dots, a_{j-1}, a_{j+1}, \dots, a_n$ to the back of the sequence. This operation is called a "swap and move".
② **Goal state:** The sequence a_1, a_2, \dots, a_n is sorted in non-decreasing order.
③ **Optimal Solution:** Find the minimum number of swap and moves required to sort the sequence.

④ **Input:** A sequence of n integers a_1, a_2, \dots, a_n .
⑤ **Output:** The minimum number of swap and moves required to sort the sequence.

⑥ **Example:** Given the sequence $[3, 1, 2]$, the minimum number of swap and moves required to sort it is 2. One possible sequence of operations is: swap a_1 and a_2 (resulting in $[1, 3, 2]$), then move the sequence $[1, 3]$ to the back (resulting in $[2]$).

⑦ **Complexity:** The problem can be solved using dynamic programming or divide-and-conquer algorithms, with a time complexity of $O(n^2)$ or $O(n \log n)$ respectively.

⑧ **Implementation:** A simple implementation of a divide-and-conquer algorithm for this problem is as follows:

```
function minSwapsAndMoves(a)
    if n == 1 then return 0
    else if n == 2 then return 1 if a[1] > a[2] else 0
    else
        m = n // 2
        l = a[:m]
        r = a[m:]
        ll = minSwapsAndMoves(l)
        lr = minSwapsAndMoves(r)
        lrll = minSwapsAndMoves(l + r)
        return min(ll + lr, lrll) + 1
```