Minimum Swaps 2

Isaac DeJager

March 15, 2023

Click here to view the HackerRank page

1 Problem Description

You are given an unordered array of consecutive integers $\in [1, 2, 3, ..., n]$ without any duplicates. You are allowed to swap any two elements. Find the minimum number of swaps required to sort the array in ascending order.

Example
$arr = [7, 1, 3, 2, 4, 5, 6]$
Perform the following steps:

i	arr	swap (indices)
0	$[7, 1, 3, 2, 4, 5, 6]$	swap (0, 3)
1	$[2, 1, 3, 7, 4, 5, 6]$	swap (0, 1)
2	$[1, 2, 3, 7, 4, 5, 6]$	swap (3, 4)
3	$[1, 2, 3, 4, 7, 5, 6]$	swap (4, 5)
4	$[1, 2, 3, 4, 5, 7, 6]$	swap (5, 6)
5	$[1, 2, 3, 4, 5, 6, 7]$	

It took 5 swaps to sort the array.

Function Description
Complete the function minimumSwaps in the editor below.
minimumSwaps has the following parameter(s):

- int arr[n]: an unordered array of integers

Returns

- int: the minimum number of swaps to sort the array

Input Format
The first line contains an integer, n, the size of arr.
The second line contains n space-separated integers $arr[i]$.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq arr[i] \leq n$

Sample Input 0
4
4 3 1 2

Sample Output 0
3

Sample Explanation 0
Given array $arr : [4, 3, 1, 2]$
After swapping $(0, 2)$ we get $arr : [1, 3, 4, 2]$
After swapping $(1, 2)$ we get $arr : [1, 4, 3, 2]$
After swapping $(1, 3)$ we get $arr : [1, 2, 3, 4]$
So, we need a minimum of 3 swaps to sort the array in ascending order.

Sample Input 1
5
2 3 4 1 5

Sample Output 1
3

Sample Explanation 1
Given array $arr : [2, 3, 4, 1, 5]$
After swapping $(2, 3)$ we get $arr : [2, 3, 1, 4, 5]$
After swapping $(0, 1)$ we get $arr : [3, 2, 1, 4, 5]$
After swapping $(0, 2)$ we get $arr : [1, 2, 3, 4, 5]$
So, we need a minimum of 3 swaps to sort the array in ascending order.

Sample Input 2
7
1 3 5 2 4 6 7

Sample Output 2
3

Explanation 2
Given array $arr : [1, 3, 5, 2, 4, 6, 7]$
After swapping $(1, 3)$ we get $arr : [1, 2, 5, 3, 4, 6, 7]$
After swapping $(2, 3)$ we get $arr : [1, 2, 3, 5, 4, 6, 7]$
After swapping $(3, 4)$ we get $arr : [1, 2, 3, 4, 5, 6, 7]$
So, we need a minimum of 3 swaps to sort the array in ascending order.

2 Solution

```
int minimumSwaps(vector<int> arr) {
    int swaps = 0;

    //Array that lists the index of every
    //value in arr
    int index[arr.size()] = { 0 };

    for (int i = 0; i < arr.size(); i++) {
        index[arr[i] - 1] = i;
    }

    for (int i = 0; i < arr.size(); i++) {

        //If the current index of arr does not
        //contain the right value, swap values
        //and change the index array
        if (arr[i] != i + 1) {
            arr[index[i]] = arr[i];
            arr[i] = i + 1;
            index[arr[index[i]] - 1] = index[i];
            swaps++;
        }
    }

    return swaps;
}
```

3 Analysis

This solution hinges on being able to easily access the index of any value in arr. So, we begin by creating an integer array of size n. Each index corresponds to a unique value in arr. Index $0 \rightarrow 1$, index $4 \rightarrow 5$, etc.

So, if the inputted array is:

$arr : [3, 6, 2, 1, 5, 4]$

Then the new array index is:

$index : [3, 2, 0, 5, 4, 1]$.

Creating this array takes $O(n)$ time and $O(n)$ space.

Next, we loop through arr again in order to find the minimum number of swaps necessary. The logic flows like this:

For each index i, check to see if the value at the index is the expected value. E.G. $arr[i] = i+1$. If it is, then that index should not be swapped and we check the next index. If it isn't, then we need to swap it with the value that should be there. For example, if $arr[0] = 5$, then we need to swap the value 5 with the value 1. In order to find the index of value 1, we use our new array $index$. So we perform these steps:

- $arr[index[0]] = arr[0] \rightarrow index[0]$ tells us the index of 1 in arr. So we place 5 at that index
- $arr[0] = 1 \rightarrow$ place the value we just swapped in its correct position
- $index[arr[index[0]] - 1] = index[0] \rightarrow$ we need to update our index array because we just swapped values. Since 1 is now in the correct spot, we don't need to update its position ($index[0]$) because we won't be swapping it in the future. But we do need to change $index[4]$ to equal where 1 used to be because we moved the position of 5. We just assigned $arr[index[0]] = 5$, so we take that 5, subtract 1 to get the corresponding index in $index$, and set that value equal to $index[0]$ because that is where 1 used to be and where 5 is now.

Then we add 1 to our total number of swaps and then continue to the next index.

Because we only ever increment i, a possible concern is what if we swap $arr[i]$ with a value that is smaller. For example, what if $arr[4] = 2$. That means we swap 2 with 5 and keep going. But if 5 was initially placed at a higher index that 4, then we will never return to index 1 so we can put 2 in its correct spot.

The good news is that we do not have to worry about this scenario ever occurring. For every index i, $arr[i] \geq i$ as we loop through. Consider $i = 0$. If $arr[0] = 1$, we know that 1 is in the right spot and we keep moving. If $arr[0]! = 1$, then we find wherever 1 is and swap it. Then we move on to $i = 1$. If $arr[1]! = 2$, we repeat. We don't have to worry about $arr[1] < 2$ because the only value less than 2 is already placed at $arr[0]$. Since we only count upwards, at every index we reach all prior indices have their correct value; hence we will never run into the issue of $arr[i] < i$.

This for loop runs in $O(n)$ since it checks every index once.
So, the total time complexity is:

$$O(n) + O(n) = 2 * O(n) = O(n)$$

And the space complexity is $O(n)$.