More...

Hashing

Count distinct elements in every window of size k

Largest subarray with equal number of 0s and 1s

More...

Minimum number of swaps required to sort an array

Given an array of **n** distinct elements, find the minimum number of swaps required to sort the array.

4.2

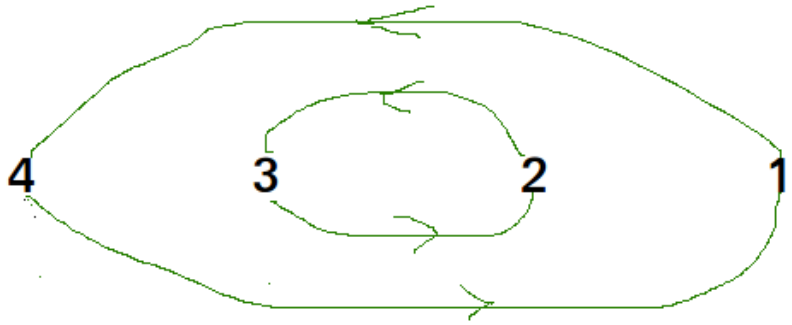
Examples:

```
Input : {4, 3, 2, 1}
Output : 2
Explanation : Swap index 0 with 3 and 1 with 2 to
              form the sorted array {1, 2, 3, 4}.

Input : {1, 5, 4, 3, 2}
Output : 2
```

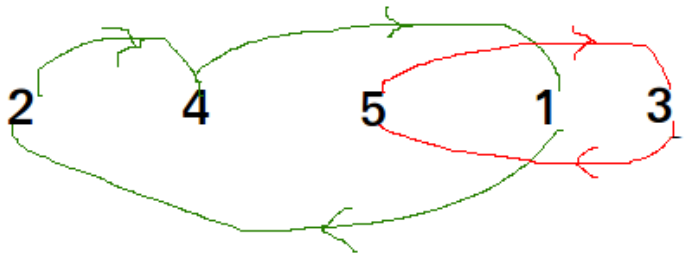
Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

This can be easily done by visualizing the problem as a graph. We will have **n** nodes and an edge directed from node **i** to node **j** if the element at i'th index must be present at j'th index in the sorted array.



Graph for {4, 3, 2, 1}

The graph will now contain many non-intersecting cycles. Now a cycle with 2 nodes will only require 1 swap to reach the correct ordering, similarly a cycle with 3 nodes will only require 2 swap to do so.



Graph for {4, 5, 2, 1, 5}

Hence,

- ans = $\Sigma_{i=1}^{k}$(cycle_size – 1)

where **k** is the number of cycles

Below is the C++ implementation of the idea.

C++

```
// C++ program to find  minimum number of swaps
// required to sort an array
#include<bits/stdc++.h>
using namespace std;

// Function returns the minimum number of swaps
// required to sort the array
int minSwaps(int arr[], int n)
{
    // Create an array of pairs where first
    // element is array element and second element
    // is position of first element
    pair<int, int> arrPos[n];
    for (int i = 0; i < n; i++)
```

```
    {
        arrPos[i].first = arr[i];
        arrPos[i].second = i;
    }

    // Sort the array by array element values to
    // get right position of every element as second
    // element of pair.
    sort(arrPos, arrPos + n);

    // To keep track of visited elements. Initialize
    // all elements as not visited or false.
    vector<bool> vis(n, false);

    // Initialize result
    int ans = 0;

    // Traverse array elements
    for (int i = 0; i < n; i++)
    {
        // already swapped and corrected or
        // already present at correct pos
        if (vis[i] || arrPos[i].second == i)
            continue;

        // find out the number of  node in
        // this cycle and add in ans
        int cycle_size = 0;
        int j = i;
        while (!vis[j])
        {
            vis[j] = 1;

            // move to next node
            j = arrPos[j].second;
            cycle_size++;
        }

        // Update answer by adding current cycle.
        ans += (cycle_size - 1);
    }

    // Return result
    return ans;
}

// Driver program to test the above function
int main()
{
    int arr[] = {1, 5, 4, 3, 2};
    int n = (sizeof(arr) / sizeof(int));
    cout << minSwaps(arr, n);
    return 0;
}
```

Run on IDE

Java

```
// Java program to find  minimum number of swaps
// required to sort an array
import javafx.util.Pair;
```