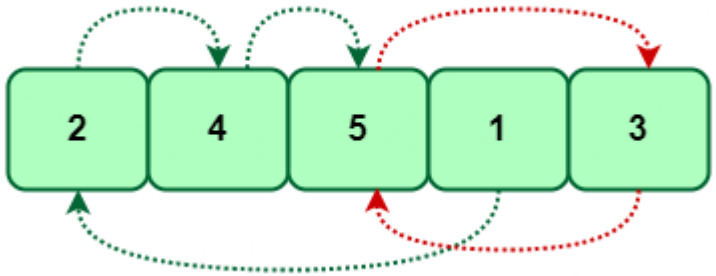
Dash
All
Articles
Videos
Problems
Quiz
Prev
Next

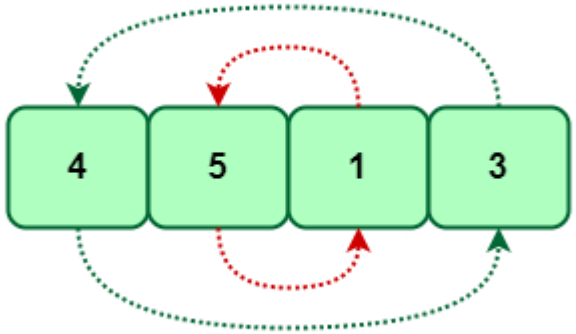
Cycle Sort

Cycle sort is an in-place sorting Algorithm, unstable sorting algorithm, and a comparison sort that is theoretically optimal in terms of the total number of writes to the original array.

- It is optimal in terms of the number of memory writes. It minimizes the number of memory writes to sort (Each value is either written zero times if it's already in its correct position or written one time to its correct position.)
- It is based on the idea that the array to be sorted can be divided into cycles. Cycles can be visualized as a graph. We 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.
  Cycle in arr[] = {2, 4, 5, 1, 3}



Cycle in arr[] = {4, 3, 2, 1}



Implementation

C++ Java

```
// Java program to implement cycle sort

import java.util.*;
import java.lang.*;

class GFG {
    // Function sort the array using Cycle sort
    public static void cycleSort(int arr[], int n)
    {
        // count number of memory writes
        int writes = 0;
```

```
// traverse array elements and put it to on
// the right place
for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
    // initialize item as starting point
    int item = arr[cycle_start];

    // Find position where we put the item. We basically
    // count all smaller elements on right side of item.
    int pos = cycle_start;
    for (int i = cycle_start + 1; i < n; i++)
        if (arr[i] < item)
            pos++;

    // If item is already in correct position
    if (pos == cycle_start)
        continue;

    // ignore all duplicate elements
    while (item == arr[pos])
        pos += 1;

    // put the item to it's right position
    if (pos != cycle_start) {
        int temp = item;
        item = arr[pos];
        arr[pos] = temp;
        writes++;
    }
```

```
            // Rotate rest of the cycle
            while (pos != cycle_start) {
                pos = cycle_start;

                // Find position where we put the element
                for (int i = cycle_start + 1; i < n; i++)
                    if (arr[i] < item)
                        pos += 1;

                // ignore all duplicate elements
                while (item == arr[pos])
                    pos += 1;

                // put the item to it's right position
                if (item != arr[pos]) {
                    int temp = item;
                    item = arr[pos];
                    arr[pos] = temp;
                }
            }
        }

        // Driver program to test above function
        public static void main(String[] args)
        {
            int arr[] = { 1, 8, 3, 9, 10, 10, 2, 4 };
            int n = arr.length;
```

```
        cycleSort(arr, n);

        System.out.println("After sort : ");
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
    }
}

// Code Contributed by Mohit Gupta_OMG <(0_o)>
```

Dash
All
Articles
Videos
Problems
Quiz

Time Complexity Analysis:

- **Worst Case:** $O(n^2)$
- **Average Case:** $O(n^2)$
- **Best Case:** $O(n^2)$

Auxiliary Space: O(1)

- The space complexity is constant cause this algorithm is in place so it does not use any extra memory to sort.