Cycle Sort

Cycle sort is an in-place, unstable sorting algorithm that is particularly useful when sorting arrays containing elements with a small range of values.

Algorithm

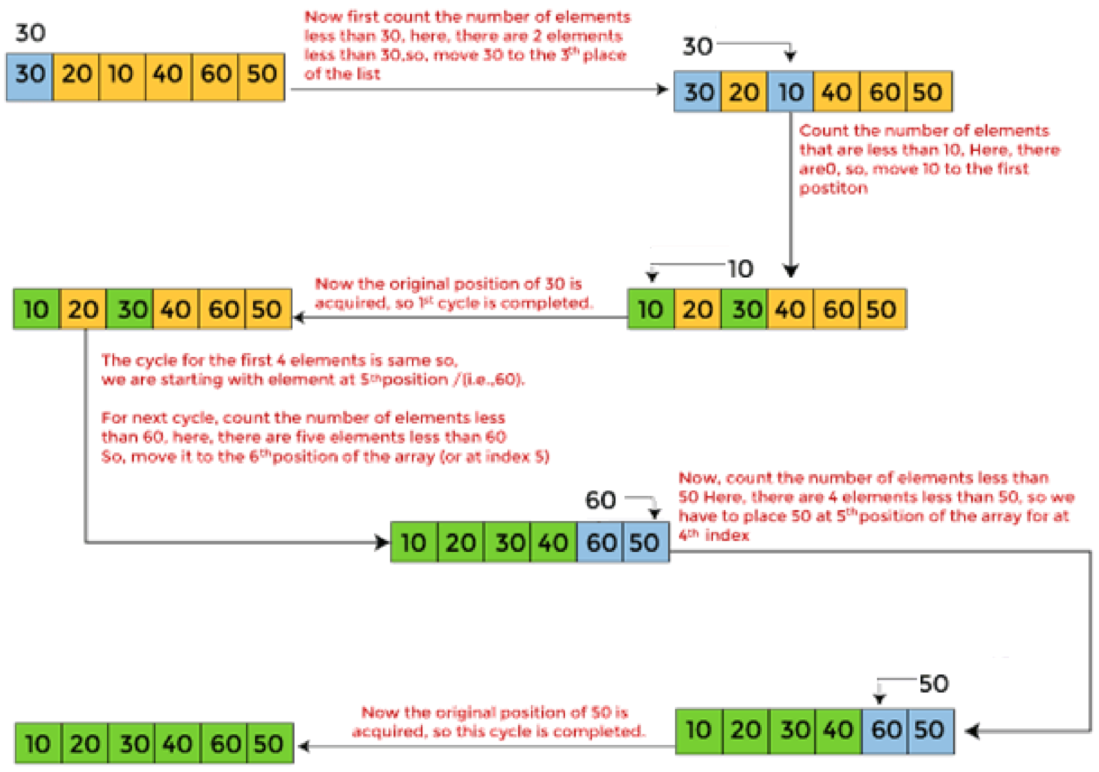
Suppose there is an array **arr** with **n** distinct elements. Given an element **A**, we can find its index by counting the number of elements smaller than **A**.

1. The element is at its correct position, simply leave it as it is.
2. Else, we have to find the correct position of **A** by counting the number of elements smaller than it. Another element **B** is replaced to be moved to its correct position. This process continues until we get an element at the original position of **A**.

Working of Cycle sort Algorithm

Now, let's see the working of Cycle sort Algorithm. To understand the working of cycle sort algorithm, let's take an unsorted array.

Let the elements of array are - **{30, 20, 10, 40, 60, 50}**



30
30 20 10 40 60 50
Now first count the number of elements less than 30, here, there are 2 elements less than 30,so, move 30 to the 3rd place of the list
30
30 20 10 40 60 50
Count the number of elements that are less than 10, Here, there are0, so, move 10 to the first postiton
10
10 20 30 40 60 50
Now the original position of 30 is acquired, so 1st cycle is completed.
10 20 30 40 60 50
The cycle for the first 4 elements is same so, we are starting with element at 5th position /(i.e.,60).
For next cycle, count the number of elements less than 60, here, there are five elements less than 60 So, move it to the 6th position of the array (or at index 5)
60
10 20 30 40 60 50
Now, count the number of elements less than 50 Here, there are 4 elements less than 50, so we have to place 50 at 5th position of the array for at 4th index
50
10 20 30 40 60 50
Now the original position of 50 is acquired, so this cycle is completed.
10 20 30 40 60 50

Now, the given array is completely sorted.

Cycle sort complexity

Now, let's see the time complexity of cycle sort in the best case, average case, and worst case. We will also see the space complexity of cycle sort.

1. Time Complexity

Case	Time Complexity
Best Case	$O(n^2)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$

The time complexity of cycle sort in all three cases in **$O(n^2)$**. Even in the best case, it takes $O(n^2)$ time to sort the array elements. In Cycle sort, there is always the traversing of the entire subarray from the current position in order to count the number of elements less than the current element.

So, in cycle sort, it doesn't matter whether the given array is already sorted or not. It has no consequence on the running time of the algorithm. So, the cycle sort must run in the quadratic time.

2. Space Complexity

Space Complexity	O(1)
Stable	No

- The space complexity of cycle sort is O(1).

Cycle Sort Algorithm

```java
package Sorting;

import java.util.Arrays;
public class CyclicSort {
    public static void main(String[] args) {
        int[] arr = {3,5,2,1,4};
        sort(arr);
    }

    public static void sort(int[] arr) {
        int i = 0;
        while (i < arr.length) {
            int correct = arr[i] - 1;

            if (arr[i] != arr[correct]) {
                swap(arr, i, correct);
            }
            else {
                i++;
            }
        }
        System.out.println(Arrays.toString(arr));
    }

    public static void swap (int[] arr, int start, int end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
    }
}
```

Find Missing Number in an Array

```java
package CyclicSortQuestion;


//Q: find Missing Number in an Array
public class MissingNumber {
    public static void main(String[] args) {
        int[] nums = {3,0,1};
        System.out.println(missingNumber(nums));
    }

    public static int missingNumber(int[] nums) {
        int i = 0;

        while (i < nums.length) {
            int correct = nums[i];
            if (nums[i] < nums.length && nums[i] != nums[correct]) {
                swap(nums, i, correct);
            }
            else {
                i++;
            }
        }
        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index) {
                return index;
            }
        }
        return nums[i];
    }

    public static void swap(int[] nums, int start, int end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
    }
}
```

Find all missing Number in an array

```java
package CyclicSortQuestion;
import java.util.ArrayList;
import java.util.List;

//Q: find all number disappered in an array
public class FindAllMissingNumber {
    public static void main(String[] args) {
        int[] nums = {1,1};
        System.out.println(findDisappearedNumbers(nums));

    }
    public static List<Integer> findDisappearedNumbers(int[] nums) {
        ArrayList<Integer> list = new ArrayList<>();
        int i = 0;
        while (i < nums.length) {
            int correct = nums[i]-1;
            if ( nums[i] != nums[correct]) {
                swap(nums, i, correct);
            }
            else {
                i++;
            }
        }
        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index+1) {
                list.add(index+1);
            }
        }
        return list;
    }

    public static void swap (int[] nums, int start, int end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
    }
}
```

Find the Duplicate Number

```java
package CyclicSortQuestion;
//Q: Find the Duplicate Number
public class FindTheDuplicateNumber {

    public static void main(String[] args) {
        int[] nums = {3,1,3,4,2};
        System.out.println(findDuplicate(nums));
    }
    public static int findDuplicate(int[] nums) {
        int i = 0;
        while (i < nums.length) {
            int correct = nums[i];
            if (nums[i] != nums[correct]) {
                swap(nums, i, correct);
            }
            else {
                i++;
            }
        }

        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index+1){
                return nums[index];
            }
        }
        return -1;
    }

    public static void swap (int[] nums, int start, int end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
    }
}
```

Find All Duplicate In an Array

```java
package CyclicSortQuestion;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
//Q: Find All Duplicates in an Array
public class FindAllDuplicatesInAnArray {
    public static void main(String[] args) {
        int[] nums = {4,3,2,7,8,2,3,1};
        System.out.println(findDuplicates(nums));
    }

    public static List<Integer> findDuplicates(int[] nums) {
        ArrayList<Integer> list = new ArrayList<>();
        int i = 0;
        while (i < nums.length) {
            int correct = nums[i]-1;
            if (nums[i] != nums[correct]) {
                swap(nums, i, correct);
            }
            else {
                i++;
            }
        }

        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index+1){
                list.add(nums[index]);
                Collections.sort(list);
            }
        }
        return list;

    }

    public static void swap (int[] nums, int start, int end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
    }
}
```

Find First Positive Number

```java
package CyclicSortQuestion;

//Q: first positive number
import java.util.Arrays;
public class FirstMissingPositive {
    public static void main(String[] args) {
        int[] nums = {2,3,6,4,5};
        System.out.println(firstMissingPositive(nums));

    }

    public static int firstMissingPositive(int[] nums) {
        int i = 0;
        while (i < nums.length) {
            int correct = nums[i]-1;

            if (nums[i] > 0 && nums[i] <= nums.length && nums[i] !=
nums[correct]) {
                swap(nums, i, correct);
            }
            else {
                i++;
            }
        }

        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index+1) {
                return index +1;
            }
        }
        return nums.length+1;
    }

    public static void swap(int[] nums, int start, int end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
    }
}
```

Set MisMatch

Input: nums = [1,2,2,4]

Output: [2,3]

```
package CyclicSortQuestion;

//Q: Set MisMatch
import java.util.Arrays;
public class Set_Mismatch {
    public static void main(String[] args) {
        int[] nums = {1,2,2,4};
        System.out.println(Arrays.toString(findErrorNums(nums)));
    }
    public static int[] findErrorNums(int[] nums) {
        int i = 0;
        while (i < nums.length) {
            int correct = nums[i] -1;
            if (nums[i] != nums[correct]) {
                swap(nums, i, correct);
            }
            else{
                i++;
            }
        }

        for (int index = 0; index < nums.length; index++) {
            if (nums[index] != index+1) {
                return new int[] {nums[index], index+1};
            }
        }
        return new int[] {-1,-1};
    }

    public static void swap(int[] nums, int start, int end) {
        int temp = nums[start];
        nums[start] = nums[end];
        nums[end] = temp;
    }
}
```