# Testing Algorithm Correctness

## Introduction

This manual provides a step-by-step guide on how to test the correctness of an algorithm using quick sort as an example. The process involves generating test data, implementing the algorithm, creating a validation program, and executing the testing procedure.

## Step 1: Generating Test Data

```
// GenerateTestData.java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class GenerateTestData {
    public static void main(String[] args) {
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(args[0]));
            int n = 10; // Number of elements in the array
            writer.write(n + "\n");

            Random random = new Random();
            for (int i = 0; i < n; i++) {
                writer.write(random.nextInt(100) + " ");
            }

            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

## Step 2: Implementing the Quick Sort Algorithm

```
// YourQuickSort.java
import java.util.Arrays;
import java.util.Scanner;

public class YourQuickSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // Read the number of elements
        int[] nums = new int[n];
        for (int i = 0; i < n; i++) {
```

```
            nums[i] = scanner.nextInt(); // read each element
        }

        // Implement your quick sort algorithm here
        quickSort(nums, 0, n - 1);

        // Output the sorted array to standard output
        System.out.println(Arrays.toString(nums));

    }

    // Quick sort and partition methods here
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

}
```

## Step 3: Creating the Validation Program

```
// StandardQuickSort.java
import java.util.Arrays;
import java.util.Scanner;

public class StandardQuickSort {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // Read the number of elements
        int[] nums = new int[n];
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt(); // read each element
        }
        // Use Arrays.sort() as the standard quick sort implementation
        Arrays.sort(nums);

        // Output the sorted array to standard output
        System.out.println(Arrays.toString(nums));

    }
}
```

### Step 4: Testing Procedure

```
javac GenerateTestData.java
javac YourQuickSort.java
javac StandardQuickSort.java
java GenerateTestData data.in
java YourQuickSort < data.in > output_custom.out
java StandardQuickSort < data.in > output_standard.out

if diff output_custom.out output_standard.out -b; then
    echo "Accepted"
else
    echo "Wrong Answer"
fi
```

The above script generates test data and sorts it using two different sorting algorithms. Then, it checks if the two output files are the same. You can save this script as a file (e.g., check_test_case.sh) and run it in the terminal by typing `bash check_test_case.sh` or `/bin/bash check_test_case.sh` . This example script only checks one test case. You can try to modify the script to check multiple test cases.

### Conclusion

By following these steps and comparing the outputs of the custom quick sort algorithm with a standard implementation, you can effectively verify the correctness of the algorithm. Adjust the implementations as needed for different algorithms and continue to refine the testing process for reliable results.