Testing Algorithm Correctness

Introduction

This guide demonstrates how to validate algorithm correctness using QuickSort as an example. It covers test data generation, algorithm implementation, validation methods, and cross-platform testing procedures.

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]))) {
            Random random = new Random();

            // Generate random array size between 10 and 10000
            int n = random.nextInt(9991) + 10;  // 10 ~ 10,000
            writer.write(n + "\n");

            // Generate random array elements
            for (int i = 0; i < n; i++) {
                writer.write(random.nextInt(10000) + " ");
            }

        } 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

4.1 Universal Testing Script (Linux/macOS/WSL)

```
#!/bin/bash
javac GenerateTestData.java
javac YourQuickSort.java
javac StandardQuickSort.java

# Single test case
java GenerateTestData data.in
java YourQuickSort < data.in > output_custom.out
java StandardQuickSort < data.in > output_standard.out

if diff -q output_custom.out output_standard.out; then
    echo "Test Passed"
else
    echo "Test Failed"
fi

# Batch testing (5 cases)
for i in {1..5}; do
    java GenerateTestData "data_$i.in"
    java YourQuickSort < "data_$i.in" > "output_custom_$i.out"
    java StandardQuickSort < "data_$i.in" > "output_standard_$i.out"

    if diff -q "output_custom_$i.out" "output_standard_$i.out"; then
        echo "Case $i: Passed"
    else
        echo "Case $i: Failed"
    fi
done
```

4.2 Windows Solutions

PowerShell Script

```
# pws_check.ps1
javac GenerateTestData.java, YourQuickSort.java, StandardQuickSort.java

# Single test
java GenerateTestData data.in
Get-Content data.in | java YourQuickSort | Out-File output_custom.out -Encoding
UTF8
Get-Content data.in | java StandardQuickSort | Out-File output_standard.out -
Encoding UTF8

if (Compare-Object (Get-Content output_custom.out) (Get-Content
output_standard.out)) {
    Write-Host "Test Failed" -ForegroundColor Red
} else {
    Write-Host "Test Passed" -ForegroundColor Green
}
```

```
# Batch testing (5 cases)
1..5 | ForEach-Object {
    java GenerateTestData "data_${_}.in"
    Get-Content "data_${_}.in" | java YourQuickSort | Out-File
"output_custom_${_}.out" -Encoding UTF8
    Get-Content "data_${_}.in" | java StandardQuickSort | Out-File
"output_standard_${_}.out" -Encoding UTF8

    if (Compare-Object (Get-Content "output_custom_${_}.out") (Get-Content
"output_standard_${_}.out")) {
        Write-Host "Case ${_}: Failed" -ForegroundColor Red
    } else {
        Write-Host "Case ${_}: Passed" -ForegroundColor Green
    }
}
```

Platform-Specific Notes

Windows Considerations:

1. **Execution Policies** (PowerShell):

   ```
   Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
   ```

   or

   ```
   powershell -ExecutionPolicy Bypass -File .\pws_check.ps1
   ```

Linux/macOS Considerations:

1. Ensure execute permissions:

   ```
   chmod +x script.sh
   ```

Troubleshooting Guide

Issue	Solution
`javac: command not found`	Add JDK to PATH or reinstall Java
Encoding mismatches	Consistent use of `-Dfile.encoding=UTF8`
File redirection errors	Use full file paths in batch/PowerShell
Path Spaces	Use quotes for paths containing spaces: `cd 'Lab 2025'/src`
Array index issues	Verify test data generation parameters
Java version not match	Verify Java version: `java -version`

Conclusion

This comprehensive guide provides multiple approaches to validate algorithm correctness across different operating systems. Users may choose the method that best fits their environment. For continuous development scenarios, we recommend setting up WSL or Git Bash to maintain consistency with Linux-based workflows. Always verify test data randomness and edge cases to ensure thorough algorithm validation.