Algorithmic Testing Project Report

Mutation Testing with Jest and Stryker

Abheet Sethi
Roll No: MT2024004

Ishtiyak Ahmad Khan
Roll No: MT2024062

Link: https://github.com/Ishtiyak3000/Algorithmic_testing

International Institute of Information Technology, Bangalore

Executive Summary

This project demonstrates a comprehensive testing framework for classical algorithms implemented in JavaScript. The primary objective is to showcase the effectiveness of mutation testing in evaluating test suite quality. We have implemented various algorithms across different categories including graph algorithms, string matching, searching, and sorting algorithms. Each algorithm is thoroughly tested using Jest for unit testing and Stryker for mutation testing to ensure code reliability and test effectiveness.

Key Achievement: Successfully implemented a robust testing framework that combines unit testing with mutation testing to achieve high code quality and test coverage. The project demonstrates best practices in software testing and quality assurance.

Project Objectives

- **Algorithm Implementation:** Develop clean, efficient implementations of classical algorithms in JavaScript
- **Comprehensive Testing:** Write complete test coverage using Jest framework for all implemented algorithms
- **Integration Testing:** Validate complex workflows and algorithm interactions through integration tests
- **Mutation Testing:** Use Stryker to measure test effectiveness and identify weaknesses in the test suite
- **Code Quality:** Maintain high code quality standards with modular, extendable, and well-documented code
- **Best Practices:** Demonstrate industry-standard testing methodologies and practices

Tools & Technologies

Node.js

JavaScript runtime environment for executing server-side code and running tests

Jest

Comprehensive testing framework for unit and integration tests with built-in assertions and mocking

Stryker Mutator

Mutation testing tool that evaluates test suite quality by introducing code mutations

GitHub

Version control system for code management, collaboration, and project tracking

Why These Tools?

Jest: Chosen for its zero-configuration setup, fast execution, built-in code coverage, and excellent developer experience. It provides snapshot testing, parallel test execution, and comprehensive mocking capabilities.

Stryker: Selected as the mutation testing framework because it supports multiple test runners, provides detailed HTML reports, and offers fine-grained control over mutation strategies. It helps identify weak spots in test suites that traditional coverage metrics miss.

Algorithms Implemented

Graph Algorithms

- Breadth-First Search
- Depth-First Search
- Floyd Warshall Algorithm
- Kruskal's Algorithm
- Prim's Algorithm
- Dijkstra's Algorithm

Graph algorithms are fundamental for solving problems related to networks, paths, and connectivity. Our implementations include traversal algorithms (BFS, DFS), shortest path algorithms (Floyd-Warshall, Dijkstra), and minimum spanning tree algorithms (Kruskal, Prim). Each algorithm handles edge cases such as disconnected graphs, negative cycles, and weighted/unweighted edges.

String Algorithms

- Rabin-Karp Algorithm
- Knuth-Morris-Pratt
- Boyer-Moore Algorithm
- Naive String Matching

String matching algorithms are crucial for text processing, pattern recognition, and search operations. We've implemented both naive and optimized approaches. The Rabin-Karp uses hashing, KMP uses prefix functions, and Boyer-Moore uses bad character and good suffix heuristics for efficient pattern matching.

Searching Algorithms

- Linear Search
- Binary Search
- Ternary Search

Searching algorithms form the basis of data retrieval operations. Linear search works on unsorted data, while binary and ternary search require sorted arrays. Binary search divides the search space in half, while ternary search divides it into three parts, useful for finding maxima/minima in unimodal functions.

Sorting Algorithms

- Merge Sort
- Quick Sort
- Selection Sort
- Bubble Sort

Sorting algorithms are essential for organizing data efficiently. We've implemented both simple (Bubble Sort, Selection Sort) and efficient (Merge Sort, Quick Sort) algorithms. Merge Sort guarantees O(n log n) time complexity, while Quick Sort offers average O(n log n) with good cache performance. Simple sorts are included for educational purposes and small dataset scenarios.

Project Structure

The project is organized into algorithm implementations, unit tests, integration tests, and mutation testing reports. This modular structure helps maintain clean separation of concerns.

algorithms

This folder contains all algorithm implementations, grouped by category:

- graph/ → Contains graph algorithm implementations (graph.js)

- strings/ → Contains string manipulation and matching algorithms (string.js)

- searching/ → Contains search algorithms (search.js)

- sorting/ → Contains sorting algorithms (sort.js)

tests

Contains all test cases written using Jest. Tests are divided into categories matching the algorithms:

- graph/ → Unit tests for graph algorithms (graph.test.js)

- strings/ → Unit tests for string algorithms (string.test.js)

- searching/ → Unit tests for searching algorithms (search.test.js)

- sorting/ → Unit tests for sorting algorithms (sort.test.js)

Integration Tests

Located under tests/integration/, these ensure that combinations of modules work correctly:

- graphint.test.js → Integration tests for graph-related operations

- stringint.test.js → Integration tests for complex string operations

-sortint.test.js → Integration test for sorting algorithms

-searchint.test.js→Integration test for search algorithms

Mutation Testing – Stryker

Mutation testing evaluates the quality of test cases by injecting small changes (mutations) into the code.

Output is stored in:

reports/mutation/mutation.html → Stryker mutation report

Configuration Files

The root directory includes important configuration files:

- jest.config.js → Jest setup for running unit & integration tests
- stryker.conf.js → Stryker setup for mutation testing
- package.json → Project dependencies & scripts
- README.md → High-level project documentation

The project follows a clean, modular structure with clear separation of concerns. Algorithms are organized by category, and each category has corresponding test files. Integration tests validate complex scenarios and algorithm interactions.

Testing Methodology

Unit Testing with Jest

Unit testing forms the foundation of our testing strategy. Each algorithm is tested in isolation with multiple test cases covering:

- **Normal Cases:** Standard inputs with expected outputs
- **Edge Cases:** Empty inputs, single elements, boundary values
- **Error Cases:** Invalid inputs, null values, type mismatches
- **Performance Cases:** Large datasets to verify algorithmic complexity
- **Special Cases:** Algorithm-specific scenarios (e.g., negative cycles in graphs, duplicate patterns in strings)

Test Structure

Below is the breakdown of the structure:

- describe(): Groups a set of related test cases.

- test(): Defines an individual test.

- Arrange: Setup input data and expected output.

- Act: Execute the algorithm or function being tested.

- Assert: Validate the output using expect().

Template Explanation (Step-by-Step)

```
describe('Algorithm Name', () => {
  test('should handle normal case', () => {
    // Arrange → Prepare test input
    const input = ...;
    const expected = ...;

    // Act → Call the function
    const result = algorithm(input);

    // Assert → Verify correctness
    expect(result).toEqual(expected);
  });

  test('should handle edge case', () => {
    // Similar structure for edge scenarios
    ...
  });

  test('should handle error case', () => {
    // Used to test failures or exceptions
    ...
  });
});
```

Integration Testing

Integration tests validate how algorithms work together and handle complex workflows. Examples include:

- Using BFS/DFS results as input to shortest path algorithms
- Combining multiple string matching algorithms for pattern analysis
- Testing algorithm chains with realistic data scenarios
- Validating data flow between different algorithm modules

Mutation Testing with Stryker

Mutation testing evaluates the quality of our test suite by introducing small changes (mutations) to the code and checking if tests catch these changes. This helps identify:

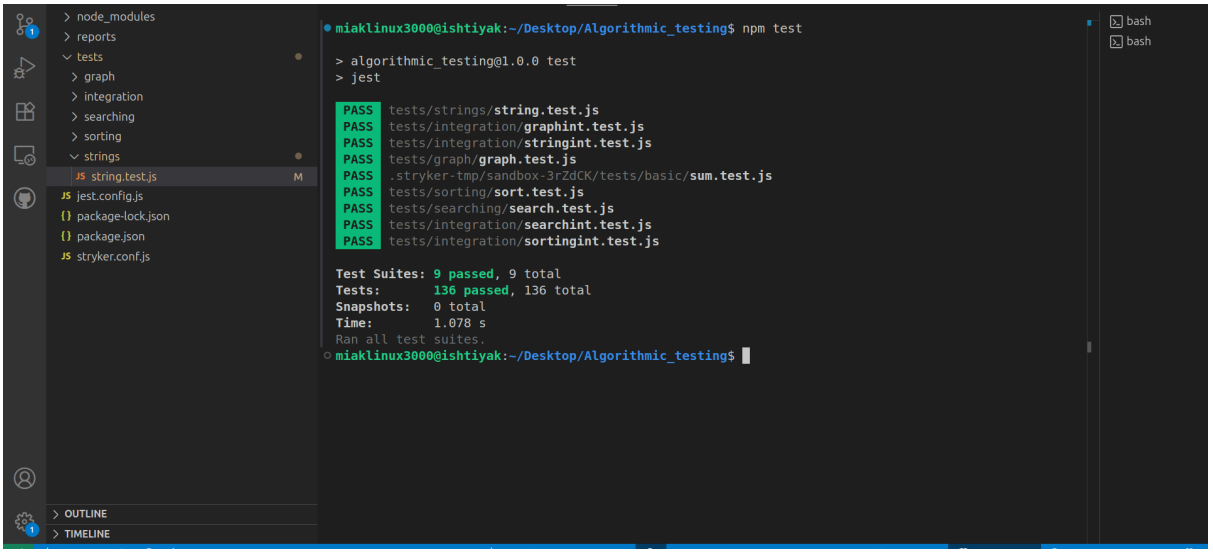
- **Weak Tests:** Tests that pass even when code is broken
- **Missing Assertions:** Tests that don't verify important behaviors
- **Dead Code:** Code paths that aren't properly tested
- **Redundant Tests:** Tests that don't add value to the suite

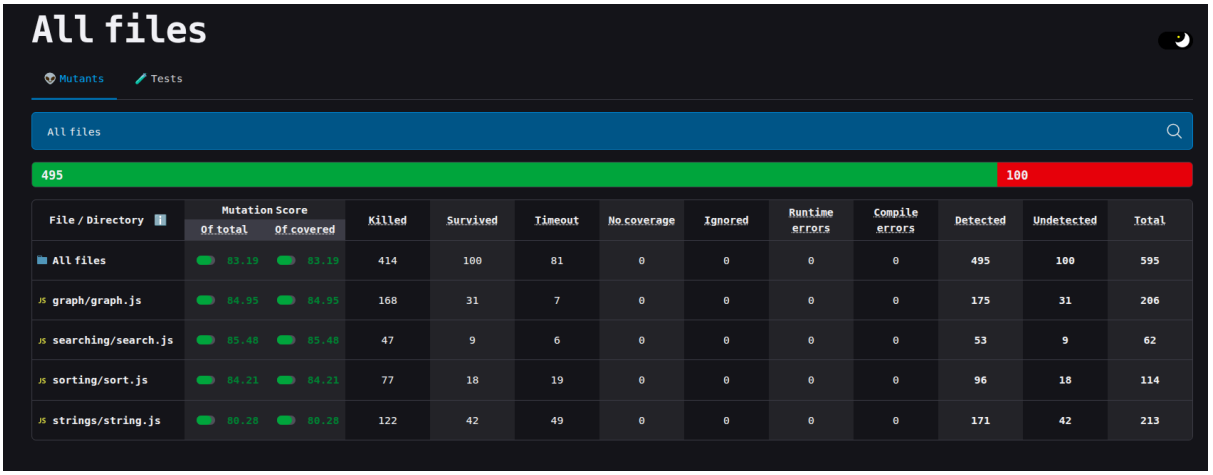
Mutation Types:

- **Arithmetic Operators:** + → -, * → /, etc.
- **Conditional Boundaries:** > → >=, < → <=, etc.
- **Logical Operators:** && → ||, ! → identity
- **Return Values:** return x → return undefined
- **Array Methods:** push → pop, shift → unshift

Test Results

MUTATION SCORE:83.19





File / Directory	Mutation Score Of total	Mutation Score Of covered	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	83.19	83.19	414	100	81	0	0	0	0	495	100	595
graph/graph.js	84.95	84.95	168	31	7	0	0	0	0	175	31	206
searching/search.js	85.48	85.48	47	9	6	0	0	0	0	53	9	62
sorting/sort.js	84.21	84.21	77	18	19	0	0	0	0	96	18	114
strings/string.js	80.28	80.28	122	42	49	0	0	0	0	171	42	213

Detailed Analysis

Test Suite Effectiveness

The mutation score provides a more accurate measure of test quality than traditional code coverage. While code coverage tells us which lines were executed, mutation testing tells us whether our tests actually verify the correctness of those lines.

Mutation Score Interpretation:

- **90-100%:** Excellent - Very strong test suite with comprehensive assertions
- **80-89%:** Good - Solid test coverage with minor gaps
- **70-79%:** Fair - Adequate testing but room for improvement
- **Below 70%:** Needs Improvement - Significant testing gaps exist

Implementation Details

Configuration Files

Jest Configuration (jest.config.js)

```
module.exports = { testEnvironment: "node", testMatch: ["**/tests/**/*.test.js"], testTimeout: 5000, collectCoverage: true, coverageDirectory: "coverage", coverageReporters: ["text", "lcov", "html"] };
```

Jest is configured to run in Node.js environment, automatically discover test files in the tests directory, and generate comprehensive coverage reports in multiple formats.

Stryker Configuration (stryker.conf.js)

```
module.exports = { mutate: ["algorithms/**/*.js"], mutator: "javascript", packageManager: "npm", reporters: ["clear-text", "html"], testRunner: "jest", jest: { projectType: "custom", config: require("./jest.config.js"), }, coverageAnalysis: "off", };
```

Stryker is configured to mutate all algorithm files, use Jest as the test runner, and generate both console and HTML reports for easy analysis of mutation testing results.

Running the Tests

Step 1: Install Dependencies

```
npm install
```

Installs all required packages including Jest and Stryker with their dependencies.

Step 2: Run Unit Tests

```
npm test
```

Executes all Jest unit and integration tests, displaying results in the console with coverage metrics.

Step 3: Run Mutation Tests

```
npm run mutation
```

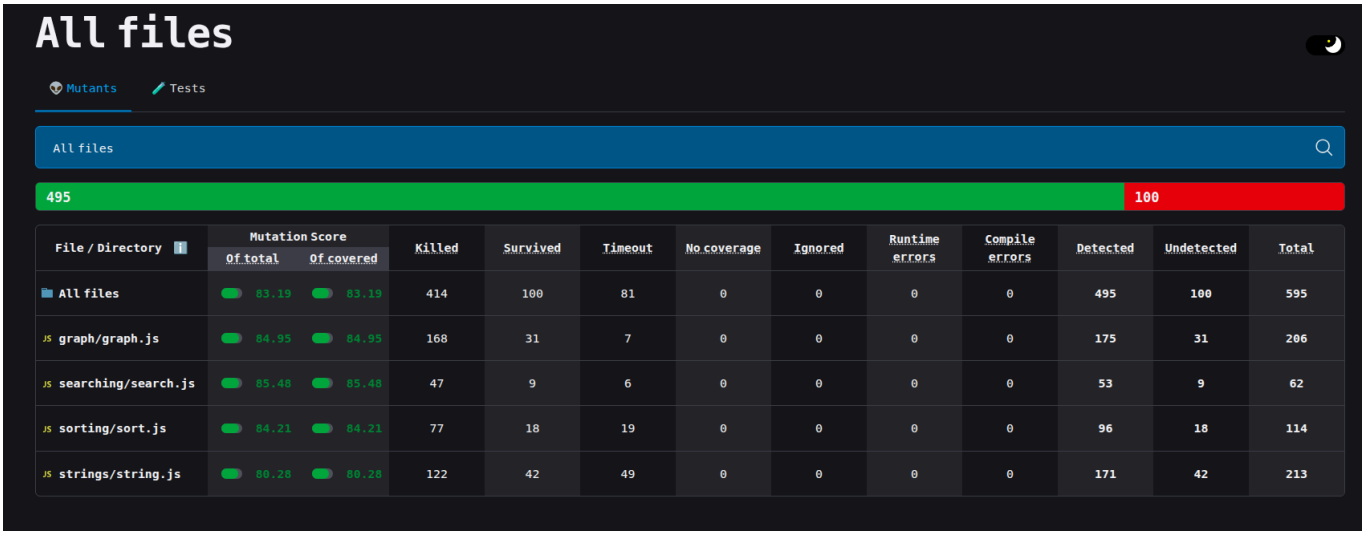
Runs Stryker mutation testing, which creates mutants of the code and verifies if tests catch the mutations. Generates an HTML report in the reports/mutation directory.

Step 4: View Reports

After running mutation tests, open

```
reports/mutation/mutation.html
```

in a browser to view detailed mutation testing results with interactive visualizations.



References

1. **Jest Documentation**
   Official Jest testing framework documentation
   https://jestjs.io/docs/getting-started
2. **Stryker Mutator Documentation**
   Official Stryker mutation testing framework documentation
   https://stryker-mutator.io/docs/
3. **Introduction to Algorithms (CLRS)**
   Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,

Clifford Stein
MIT Press, 3rd Edition

4. **JavaScript Algorithms and Data Structures**
   GitHub Repository by trekhleb
   https://github.com/trekhleb/javascript-algorithms

5. **Node.js Documentation**
   Official Node.js runtime documentation
   https://nodejs.org/docs/