CSE 731-Software Testing Project Report

Mutation Testing on Data Structure and Algorithms Using PITest and Strykar

Done By:
Deepanjali Ghosh MT2023119
Ketki Kerkar MT2023056

Project Aim:

The primary goal of this project is to apply Mutation Testing. This process is carried out using open-source tools, ensuring accessibility and practicality while rigorously testing the software for correctness and robustness.

Project Link: Project Repo Link

Code Used:

Algorithms play a critical role in software development, powering a wide range of applications to accomplish specific tasks efficiently. They are at the heart of systems ranging from simple programs to complex operations like data encryption, machine learning, and even space exploration. Given their significance, it is essential to ensure their correctness and reliability. To verify the accuracy and robustness of algorithms, we opted to apply mutation testing.

For this purpose, we utilized a diverse collection of data structure and algorithm (DSA) code that covers topics such as dynamic programming, graph algorithms, searching, sorting, queue, stack, linked list, and string manipulations. These implementations were tested in both Java and JavaScript. By focusing on these foundational algorithms, we aimed to validate their correctness under mutation testing and ensure their outputs remain accurate and reliable in various scenarios.

To be more specific, we have used the algorithms mentioned in this repo(Project Repo Link) for testing. The algorithms are used in both Java and Javascript.

Testing Strategy and Tools Used:

Mutation testing, also known as mutation analysis or program mutation, is a technique used to create new software tests and assess the effectiveness of current ones. It works by making minor changes to a program, resulting in altered versions known as mutants. The goal of testing is to identify and reject these mutants by triggering differences in behavior between the original program and the mutated version. Successfully detecting such differences is referred to as "killing the mutant.

There are 2 ways to "kill" a mutant:

- Strongly Killing a Mutant: A mutant is considered strongly killed if a
 test case produces an output or observable behavior that is different
 from the original program's output. In this scenario, the discrepancy in
 behavior directly demonstrates the presence of a fault in the mutant.
 This is the most robust form of mutant detection, as it ensures the
 mutation affects the program's visible behavior.
- Weakly Killing a Mutant: A mutant is weakly killed when a test case
 exercises the mutation and leads to a state (e.g., variable value) that
 differs from the original program, but this difference does *not* propagate
 to the program's final output or observable behavior. While the mutation
 is triggered, its effects are not visible in the external behavior of the
 program.

We opted for mutation testing as our testing approach, aiming to strongly eliminate the mutants.

- 1. IntelliJ IDEA: It is a widely used development environment tailored for Java and other programming languages. It offers a rich set of features, such as intelligent code suggestions, debugging capabilities, and seamless integration with testing frameworks.
- 2. PIT Mutation testing tool: Pitest is a mutation testing tool designed specifically for Java. It modifies the code in small ways (mutations) to assess whether your tests are robust enough to catch those changes.
- 3. VSCode: VS Code is a lightweight, versatile code editor widely used for JavaScript and other programming languages. It provides a user-friendly interface and is enriched with extensions for efficient development and testing.
- 4. Stryker: Stryker is a versatile mutation testing tool that supports various programming languages, including JavaScript, TypeScript, and C#. It complements Pitest for projects that involve languages other than Java.

MUTATIONS USED

PIT, by default provides a set of mutation operators. These operators are listed below:

BOOLEAN_FALSE_RETURN	INCREMENTS_MUTATOR					
BOOLEAN_TRUE_RETURN	INVERT_NEGS_MUTATOR					
CONDITIONALS_BOUNDARY_MUTATOR	MATH_MUTATOR					
EMPTY_RETURN_VALUES	NEGATE_CONDITIONALS_MUTATOR					
PRIMITIVE_RETURN_VALS_MUTATOR	VOID_METHOD_CALL_MUTATOR					
NULL_RETURN_VALUES						

More information about the mutation operators present in PITest can be found here.

Results Of Mutation Testing On The CodeBase(Java and Pitest):

Pit Test Coverage Report

Project Summary

Number of Classes	Li	ne Coverage	Muta	ation Coverage	Te	est Strength
45	97%	848/873	91%	690/755	92%	690/751

Breakdown by Package

Name	Number of Classes	er of Classes Line Coverage		Mutation Coverage		Test Strength	
org.example.DynamicProgramming	13	98%	237/241	93%	253/271	94%	253/269
$\underline{org.example.GraphAlgorithms}$	9	98%	207/212	91%	99/109	92%	99/108
org.example.LinkedList	1	99%	94/95	96%	52/54	98%	52/53
<u>org.example.Queue</u>	1	100%	27/27	95%	21/22	95%	21/22
org.example.Searching	4	94%	45/48	95%	53/56	95%	53/56
<u>org.example.Sorting</u>	8	95%	132/139	84%	118/140	84%	118/140
<u>org.example.Stack</u>	1	100%	17/17	100%	12/12	100%	12/12
org.example.String	8	95%	89/94	90%	82/91	90%	82/91

Report generated by PIT 1.17.1

Enhanced functionality available at arcmutate.com

1.Dynamic Programming:

Pit Test Coverage Report

Package Summary

 ${\bf org. example. Dynamic Programming}$

Number of Classes	. Li	ne Coverage	Muta	ation Coverage	Te	est Strength
13	98%	237/241	93%	253/271	94%	253/269

Breakdown by Class

Name	Liı	ie Coverage	Muta	tion Coverage	Test Strength		
<u>CoinChange.java</u>	100%	18/18	95%	20/21	95%	20/21	
<u>EditDistance.java</u>	100%	21/21	88%	22/25	88%	22/25	
Knapsack.java	92%	12/13	100%	21/21	100%	21/21	
LongestCommonSubsequence.java	100%	10/10	100%	15/15	100%	15/15	
LongestIncreasingSubsequence.java	100%	44/44	90%	35/39	90%	35/39	
LongestPalindromicSubsequence.java	100%	24/24	100%	14/14	100%	14/14	
MatrixChainMultiplication.java	94%	15/16	94%	15/16	100%	15/15	
PalindromePartitioning.java	100%	23/23	97%	28/29	97%	28/29	
RodCutting.java	100%	9/9	100%	9/9	100%	9/9	
ShortestCommonSupersequence.java	93%	27/29	90%	35/39	92%	35/38	
StockBuyAndSell.java	100%	14/14	84%	16/19	84%	16/19	
<u>TargetSum.java</u>	100%	13/13	93%	14/15	93%	14/15	
<u>UniquePaths.java</u>	100%	7/7	100%	9/9	100%	9/9	

Report generated by \underline{PIT} 1.17.1

2. Graph Algorithms:

Pit Test Coverage Report

Package Summary

org.example.GraphAlgorithms

Number of Classes	s Li	ne Coverage	Muta	ation Coverage	Te	est Strength
9	98%	207/212	91%	99/109	92%	99/108

Breakdown by Class

Lir	ie Coverage	Muta	tion Coverage	Test Strength		
100%	23/23	100%	3/3	100%	3/3	
93%	13/14	92%	11/12	92%	11/12	
95%	18/19	90%	9/10	90%	9/10	
100%	19/19	100%	4/4	100%	4/4	
100%	31/31	75%	6/8	75%	6/8	
94%	15/16	95%	18/19	95%	18/19	
97%	31/32	93%	26/28	96%	26/27	
97%	31/32	94%	16/17	94%	16/17	
100%	26/26	75%	6/8	75%	6/8	
	100% 93% 95% 100% 100% 94% 97%	93% 13/14 95% 18/19 100% 19/19 100% 31/31 94% 15/16 97% 31/32 97% 31/32	100% 23/23 100% 93% 13/14 92% 95% 18/19 90% 100% 19/19 100% 100% 31/31 75% 94% 15/16 95% 97% 31/32 93% 97% 31/32 94%	100% 23/23 100% 3/3 93% 13/14 92% 11/12 95% 18/19 90% 9/10 100% 19/19 100% 4/4 100% 31/31 75% 6/8 94% 15/16 95% 18/19 97% 31/32 93% 26/28 97% 31/32 94% 16/17	100% 23/23 100% 3/3 100% 93% 13/14 92% 11/12 92% 95% 18/19 90% 9/10 90% 100% 19/19 100% 4/4 100% 100% 31/31 75% 6/8 75% 94% 15/16 95% 18/19 95% 97% 31/32 93% 26/28 96% 97% 31/32 94% 16/17 94%	

Report generated by $\underline{PIT}\ 1.17.1$

3.Linked List:

Pit Test Coverage Report

Package Summary

org. example. Linked List

Number of Classes	Number of Classes Lin		Muta	ation Coverage	Test Strength	
1	99%	94/95	96%	52/54	98%	52/53

Breakdown by Class

Name	Line Coverage		Muta	tion Coverage	Test Strength		
<u>LinkedList.java</u>	99%	94/95	96%	52/54	98%	52/53	

Report generated by PIT 1.17.1

4.Queue

Pit Test Coverage Report

Package Summary

org.example.Queue

Number of Classes	s Lin	ie Coverage	Muta	ation Coverage	Te	est Strength
1	100%	27/27	95%	21/22	95%	21/22

Breakdown by Class

Name	Line Coverage		Mutat	ion Coverage	Test Strength		
<u>QueueClass.java</u>	100%	27/27	95%	21/22	95%	21/22	

Report generated by PIT 1.17.1

5.Searching

Pit Test Coverage Report

Package Summary

org.example.Searching

Number of Classes	s Li	ne Coverage	Muta	ation Coverage	Te	est Strength
4	94%	45/48	95%	53/56	95%	53/56

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
<u>BinarySearch.java</u>	100%	12/12	92%	12/13	92%	12/13
ExponentialSearch.java	94%	16/17	96%	22/23	96% [22/23
<u>JumpSearch.java</u>	93%	13/14	93%	14/15	93%	14/15
<u>LinearSearch.java</u>	80%	4/5	100%	5/5	100%	5/5

Report generated by PIT 1.17.1

6.Sorting:

Pit Test Coverage Report

Package Summary

org.example.Sorting

Number of Classes L		Line Coverage	Mut	Mutation Coverage		Test Strength	
8	95%	132/139	84%	118/140	84%	118/140	
Breakdown by Class							
Name	Liı	ie Coverage		tion Coverage		st Strength	
<u>BubbleSort.java</u>	89%	8/9	75%	9/12	75%	9/12	
BucketSort.java	96%	27/28	78%	18/23	78%	18/23	
CountingSort.java	94%	15/16	88%	14/16	88%	14/16	
<u>HeapSort.java</u>	96%	22/23	78%	18/23	78%	18/23	
InsertionSort.java	90%	9/10	90%	9/10	90%	9/10	
MergeSort.java	100%	24/24	97%	31/32	97%	31/32	
<u>QuickSort.java</u>	94%	17/18	81%	13/16	81%	13/16	
SelectionSort.java	91%	10/11	75%	6/8	75%	6/8	

Report generated by PIT 1.17.1

7.Stack:

Pit Test Coverage Report

Package Summary

org.example.Stack

Number of Classes Line Coverage		ge Mutation Cover	rage Test Strength	1
1	100% 17/17	7 100% 12/12	100% 12/12	
Breakdown by	Class			
Name	Line Coverage	Mutation Coverage	Test Strength	
StackClace java 1	00% 17/17	100% 12/12	100% 12/12	

Report generated by <u>PIT</u> 1.17.1

8.String

Pit Test Coverage Report

Package Summary

org.example.String

Number of Classes	s Li	ne Coverage	Muta	ation Coverage	Te	est Strength
8	95%	89/94	90%	82/91	90%	82/91

Breakdown by Class

Name	Liı	ne Coverage	Muta	tion Coverage	Te	st Strength
<u>Anagram.java</u>	100%	6/6	100%	4/4	100%	4/4
FirstOccurance.java	80%	4/5	80%	4/5	80%	4/5
<u>IsDigit.java</u>	100%	7/7	100%	8/8	100%	8/8
<u>IsPalindrome.java</u>	100%	8/8	88%	7/8	88%	7/8
KMP.java	97%	31/32	73%	16/22	73%	16/22
LengthOfString.java	80%	4/5	100%	2/2	100%	2/2
<u>RabinKarp.java</u>	92%	22/24	97%	34/35	97%	34/35
<u>ReverseString.java</u>	100%	7/7	100%	7/7	100%	7/7

Unit Mutation Operators Used:

LongestCommonSubsequence.java

```
package org.example.DynamicProgramming;
        public class LongestCommonSubsequence {
                public int lcs(String text1, String text2) {
  int n = text1.length();
  int m = text2.length();
  int[][] dp = new int[n + 1][m + 1];
5
6
7 2
8
9 2
10 2
11 3
12 3
                          for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= m; j++) {
    if (textl.charAt(i - 1) == text2.charAt(j - 1)) {
        dp[i][j] = 1 + dp[i - 1][j - 1];
    }</pre>
13
                                                 dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
142
                          return dp[n][m];
18 1
        Mutations

    Replaced integer addition with subtraction → KILLED
    Replaced integer addition with subtraction → KILLED

    changed conditional boundary → KILLED
    negated conditional → KILLED

        1. changed conditional boundary → KILLED 2. negated conditional → KILLED

    negated conditional - KILLED
    Replaced integer subtraction with addition - KILLED
    Replaced integer subtraction with addition - KILLED
    negated conditional - KILLED
    negated conditional - KILLED
    Replaced integer subtraction with addition - KILLED
    Replaced integer subtraction with addition - KILLED
    Replaced integer subtraction with subtraction - KILLED

12
       1. Replaced integer subtraction with addition \rightarrow KILLED 2. Replaced integer subtraction with addition \rightarrow KILLED
18 1. replaced int return with 0 for org/example/DynamicProgramming/LongestCommonSubsequence::lcs - KILLED
```

```
Mutations
\underline{16} 1. Replaced integer subtraction with addition \rightarrow KILLED 2. negated conditional \rightarrow KILLED
20 1. Replaced integer addition with subtraction → KILLED
    1. negated conditional → KILLED
    1. replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED
    1. Replaced integer subtraction with addition → KILLED
    1. replaced int return with 0 for org/example/Stack/StackClass::pop - KILLED
    1. negated conditional → KILLED
    1. replaced int return with 0 for org/example/Stack/StackClass::top - KILLED

    replaced int return with 0 for org/example/Stack/StackClass::top → KILLED

    negated conditional → KILLED
    replaced boolean return with true for org/example/Stack/StackClass::isEmpty → KILLED
```

Active mutators

- CONDITIONALS BOUNDARY
 EMPTY RETURNS
 FALSE RETURNS
 INCREMENTS
 INCREMENTS

- INVERT NEGS
 MATH
 NEGATE CONDITIONALS
 NULL RETURNS
 PRIMITIVE RETURNS
 TRUE RETURNS

- VOID METHOD CALLS

StackClass.java

```
package org.example.Stack;
3
    public class StackClass {
4
        private int maxSize;
5
        private int[] stackArray;
        private int top;
6
8
        public StackClass(int size) {
9
            maxSize = size;
10
             stackArray = new int[maxSize];
11
             top = -1;
12
13
14
         // Method to push an element onto the stack
        public void push(int value) {
15
16 <u>2</u>
             if (top == maxSize - 1) {
18
                return;
19
             stackArray[++top] = value;
20 1
21
22
23
         // Method to pop an element from the stack
24
        public int pop() {
   if (top == -1)
2.5
26 <u>1</u>
28 <u>1</u>
29
             int poppedElement = stackArray[top--];
30 <u>1</u>
31
32 <u>1</u>
             return poppedElement;
33
34
35
         // Method to peek the top element of the stack
        public int top() {
36
             if (top == -1)
37 <u>1</u>
38
39 <u>1</u>
40
             return stackArray[top];
41 <u>1</u>
42
43
```

Mutations

```
1. Replaced integer subtraction with addition \rightarrow KILLED 2. negated conditional \rightarrow KILLED
1. Replaced integer addition with subtraction → KILLED
1. negated conditional → KILLED

    replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED

    Replaced integer subtraction with addition → KILLED

    replaced int return with 0 for org/example/Stack/StackClass::pop → KILLED

    negated conditional → KILLED

    replaced int return with 0 for org/example/Stack/StackClass::top → KILLED

1. replaced int return with 0 for org/example/Stack/StackClass::top → KILLED
1. negated conditional \rightarrow KILLED 2. replaced boolean return with true for org/example/Stack/StackClass::isEmpty \rightarrow KILLED
```

Active mutators

- CONDITIONALS BOUNDARY
 EMPTY RETURNS
 FALSE RETURNS
 INCREMENTS

- INVERT_NEGS MATH
- MATH
 NEGATE CONDITIONALS
 NULL_RETURNS
 PRIMITIVE RETURNS
 TRUE_RETURNS
 VOID_METHOD_CALLS

Integration Mutation Operators Used:

```
public class MergeSort {
4
         public static void merge(int[] arr, int left, int mid, int right) {
              int n1 = mid - left + 1;
int n2 = right - mid;
7 3
8 <u>1</u>
9
              int[] leftArray = new int[n1];
10
              int[] rightArray = new int[n2];
11
12
              System.arraycopy(arr, left, leftArray, 0, n1);
System.arraycopy(arr, mid + 1, rightArray, 0, n2);
14 2
15 <u>4</u>
16
17
18 <u>2</u>
              int i = 0, j = 0;
              int k = left;
while (i < n1 && j < n2) {</pre>
19
20 8
                   if (leftArray[i] <= rightArray[j]) {</pre>
21 <u>4</u>
22 <u>4</u>
                        arr[k++] = leftArray[i++];
23
                   } else {
24 <u>4</u>
                     arr[k++] = rightArray[j++];
25
26
28 <u>4</u>
              while (i < n1) {
29 <u>4</u>
                  arr[k++] = leftArray[i++];
30
31
32
33 <u>4</u>
               while (j < n2) {
34 <u>4</u>
                   arr[k++] = rightArray[j++];
35
36
37
38
39
          public static void mergeSort(int[] arr, int left, int right) {
              if (left < right) {
40 <u>4</u>
41 3
                   int mid = (left + right) / 2;
42
43
                   mergeSort(arr, left, mid);
441
                   mergeSort(arr, mid + 1, right);
45 <u>3</u>
46
47
48 <u>1</u>
                  merge(arr, left, mid, right);
50
```

Active mutators

```
CONDITIONALS BOUNDARY
CONSTRUCTOR CALLS
EMPTY RETURNS
EXPERIMENTAL ARGUMENT PROPAGATION
EXPERIMENTAL BIG DECIMAL
EXPERIMENTAL BIG INTEGER
EXPERIMENTAL MEMBER VARIABLE
EXPERIMENTAL NAKED RECEIVER
EXPERIMENTAL SWITCH
EXPERIMENTAL SWITCH
INCREMENTAL SWITCH
INCREMENTS
INCREMENTS
INCREMENTS
INVERT NEGS
MATH

MEGATE CONDITIONALS
```

- INVERT_NEGS

 MATH

 NEGATE CONDITIONALS

 NON VOID METHOD CALLS

 NULL RETURNS

 PRIMITIVE RETURNS

 PRIMITIVE RETURNS

 REMOVE CONDITIONALS EQUAL ELSE

 REMOVE CONDITIONALS ORDER ELSE

 REMOVE CONDITIONALS ORDER ELSE

 REMOVE CONDITIONALS ORDER ELSE

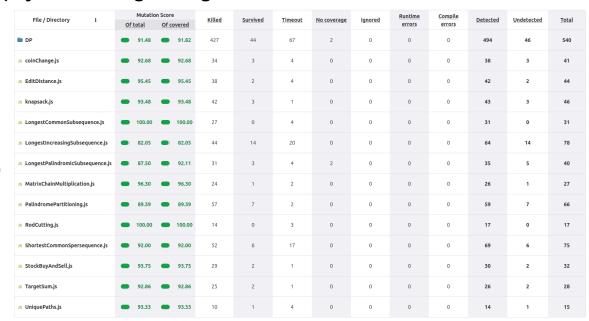
 TRUE RETURNS

 VOID METHOD CALLS

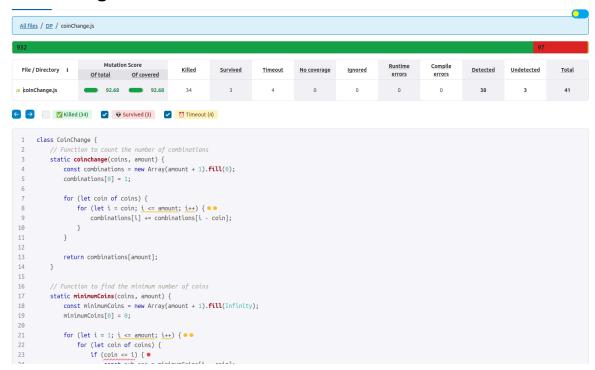
Results Of Mutation Testing On The CodeBase(Javascript and Stryker):



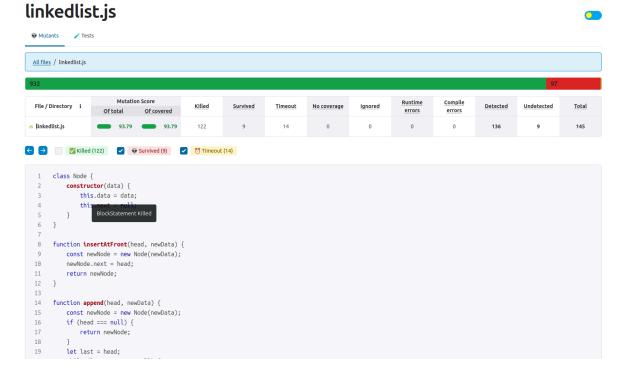
1)Dynamic Programing



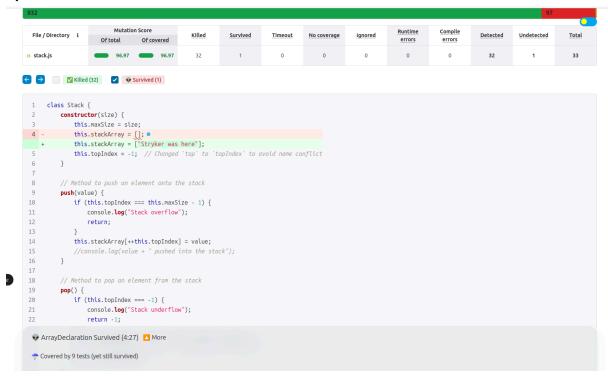
Coin Change:



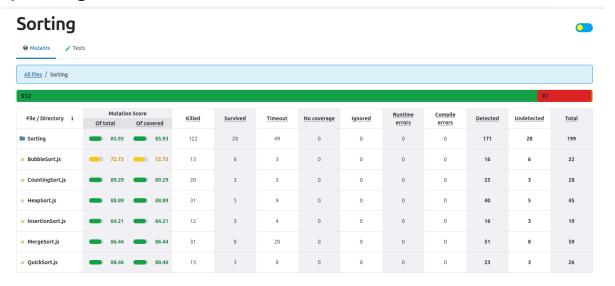
2)Linked List



3)Stack



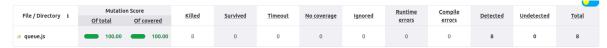
4)Sorting



5)Searching



6)Queue



```
← → ✓ Killed (8)
       class Queue {
            constructor() {
                this.items = {}
this.frontIndex = 0
                 this.backIndex = 0
            enqueue(item) {
                 this.items[this.backIndex] = item
                 this.backIndex++
return item + ' inserted'
 10
11
12
13
            dequeue() {
                const item = this.items[this.frontIndex]
 14
15
16
17
18
19
20
21
22
23
24
25
                 delete this.items[this.frontIndex]
                 this.frontIndex++
                 return item
            peek() {
                 return this.items[this.frontIndex]
            get printQueue() {
                 return this.items;
       module.exports = Queue;
```