Software Testing, Lab 6, April 4, 2023.

**Tasks:**

1. Download the PITest example to learn how to use PITest for mutation testing based on Ant projects.
   https://github.com/hcoles/pitest-ant-example
2. Write code to implement the following functions:
   - BubbleSort.java is an implementation of bubble sort algorithm
   - BackPack.java is a solution of 01 backpack problem.
3. Try to generate Mutants of 2 programs with PITest.
4. Write testing cases for 2 functions with Junit according to your previous study (MC/DC, boundary value, equivalence partitioning, etc.), guarantee the sufficiency and diversity of your test set. Each function should have 15 test cases.
5. Use Cobertura to produce coverage.
6. Then run mutants on the test sets with PITest. In order for you to learn how to modify the build.xml file, please make the final file structure like follow:

```
├── lib
├── pitResults
| ├── 202304030933
| └── export
├── src
| ├── main
| | └── java
| └── test
| └── java
└── target
├── classes
├── test-classes
└── test-result
```

   - lib : the jars that need to be used
   - pitResults/ : the web page generated after the mutation test
   - pitResults/export : the mutants created after the mutation test
   - src/main/java : source code
   - src/test/java : test program source code
   - target/classes : class file of source code
   - target/test-classes : class file of test program
   - target/test-result : the execution results of the 30 test cases you designed
7. Analyzing the report provided by PITest
8. Discuss and explain your results

**Requirements for the experiment**:

1. Finish the tasks above individually.
2. Post your experiment report to “智慧树” , the following information should be included in your report:
    - source code 1
    - test program 1
    - The coverage of the test cases you design 1
    - Generate the file structure using tree -d 2
    - The coverage of the test cases generated by PITest 2
    - Analyzing the report 1.5
    - Discuss and explain your results 1.5

**Submission deadline:**

23:59 April 17, 2023.

## 1.source code

```
public class BubbleSort {
    public static int[] sort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
        return array;
    }
}
```

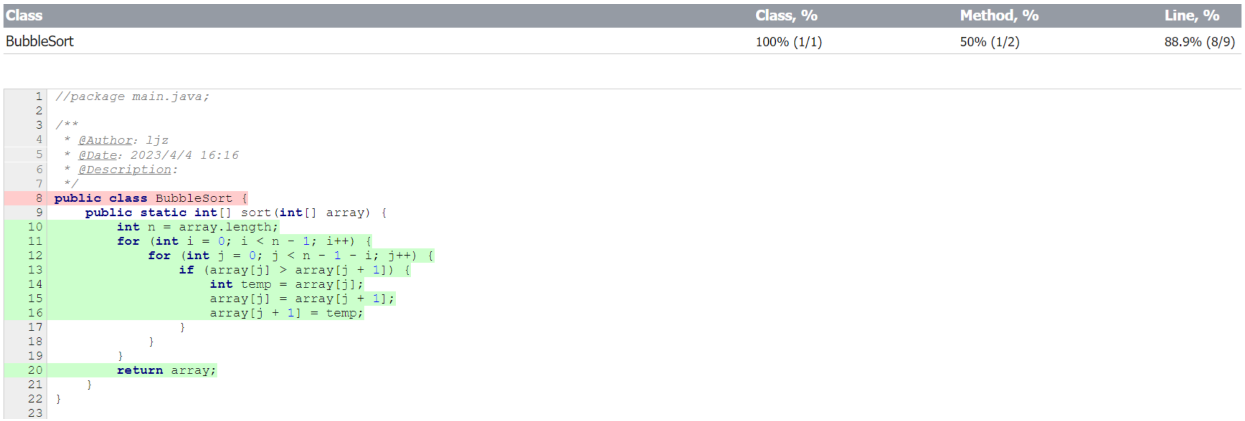
```
public class BackPack {
    public static int maxWeight(int capacity, int[] weights, int[] values) {
        int n = weights.length;
        int[][] dp = new int[n + 1][capacity + 1];

        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= capacity; w++) {
                if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]]
+ values[i - 1]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        return dp[n][capacity];
    }
}
```

## 2.test program

```
public class BubbleSortTest {



    @Test
    public void testArray0() {
        int[] input = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
        int[] expected = {1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray1() {
        int[] input = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int[] expected = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray2() {
        int[] input = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
        int[] expected = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray3() {
        int[] input = {1};
        int[] expected = {1};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray4() {
        int[] input = {};
        int[] expected = {};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray5() {
        int[] input = {4, 4, 4, 4};
        int[] expected = {4, 4, 4, 4};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray6() {
        int[] input = {-2, 0, 3, -1, 5};
        int[] expected = {-2, -1, 0, 3, 5};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray7() {
        int[] input = {Integer.MAX_VALUE, Integer.MIN_VALUE, 0};
        int[] expected = {Integer.MIN_VALUE, 0, Integer.MAX_VALUE};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }
    @Test
    public void testArray8() {
        int[] input = {7, 5, 3, 1, 2, 4, 6, 8};
        int[] expected = {1, 2, 3, 4, 5, 6, 7, 8};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray9() {
        int[] input = {5, 3, 8, 6, 2};
        int[] expected = {2, 3, 5, 6, 8};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray10() {
        int[] input = {2, 3, 1, 6, 7, 5, 4};
        int[] expected = {1, 2, 3, 4, 5, 6, 7};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray11() {
        int[] input = {4, 2, 9, 6, 23, 12, 34, 0, 1};
        int[] expected = {0, 1, 2, 4, 6, 9, 12, 23, 34};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray12() {
        int[] input = {-5, -9, 8, 12, -1, 0, 6};
        int[] expected = {-9, -5, -1, 0, 6, 8, 12};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray13() {
        int[] input = {0, 0, 0, 0, 0, 0};
        int[] expected = {0, 0, 0, 0, 0, 0};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }

    @Test
    public void testArray14() {
        int[] input = {1, 0, -1, 20, 15, 12, 30, 45, 6};
        int[] expected = {-1, 0, 1, 6, 12, 15, 20, 30, 45};
        assertArrayEquals(expected, BubbleSort.sort(input));
    }
}
```

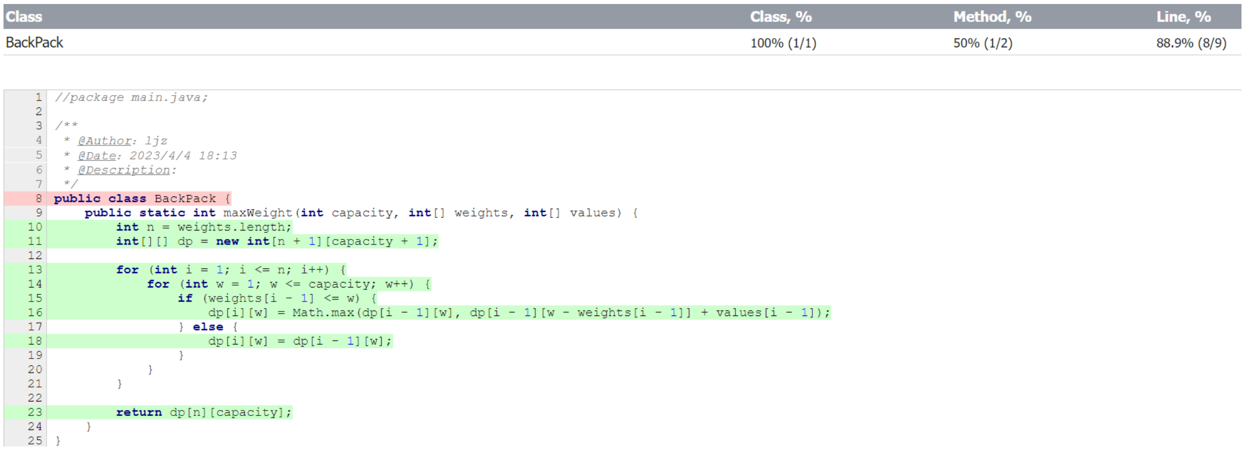
```
public class BackPackTest {

    int[] weights;
    int[] values;

    @Test
    public void testMaxWeight1() {
        int capacity = 10;
        weights = new int[]{1, 4, 3, 5};
        values = new int[]{1500, 3000, 2000, 2000};
        int expected = 6500;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight2() {
        int capacity = 50;
        weights = new int[]{10, 20, 30};
        values = new int[]{60, 100, 120};
        int expected = 220;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight3() {
        int capacity = 0;
        weights = new int[]{};
        values = new int[]{};
        int expected = 0;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight4() {
        int capacity = 12;
        weights = new int[]{3, 6, 9};
        values = new int[]{40, 70, 120};
        int expected = 160;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight5() {
        int capacity = 15;
        weights = new int[]{4, 5, 6};
        values = new int[]{10, 20, 30};
        int expected = 60;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight6() {
        int capacity = 7;
        weights = new int[]{2, 3, 4};
        values = new int[]{30, 40, 50};
        int expected = 90;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight7() {
        int capacity = 60;
        weights = new int[]{10, 15, 25, 30};
        values = new int[]{50, 80, 100, 200};
        int expected = 330;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight8() {
        int capacity = 25;
        weights = new int[]{5, 7, 10, 12};
        values = new int[]{20, 35, 50, 65};
        int expected = 120;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight9() {
        int capacity = 8;
        weights = new int[]{2, 2, 4};
        values = new int[]{40, 50, 100};
        int expected = 190;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight10() {
        int capacity = 20;
        weights = new int[]{5, 10, 15};
        values = new int[]{30, 60, 90};
        int expected = 120;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test


    public void testMaxWeight11() {
        int capacity = 100;
        weights = new int[]{20, 30, 50, 70};
        values = new int[]{60, 90, 140, 210};
        int expected = 300;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight12() {
        int capacity = 30;
        weights = new int[]{5, 10, 15, 20};
        values = new int[]{30, 60, 90, 120};
        int expected = 180;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight13() {
        int capacity = 40;
        weights = new int[]{10, 20, 30, 40};
        values = new int[]{50, 100, 150, 200};
        int expected = 200;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight14() {
        int capacity = 25;
        weights = new int[]{4, 8, 12};
        values = new int[]{20, 45, 70};
        int expected = 135;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }

    @Test
    public void testMaxWeight15() {
        int capacity = 35;
        weights = new int[]{5, 15, 25};
        values = new int[]{30, 75, 120};
        int expected = 150;
        assertEquals(expected, BackPack.maxWeight(capacity, weights, values));
    }
}
```

## 3.The coverage of the test cases



| Class | Class, % | Method, % | Line, % |
|---|---|---|---|
| BubbleSort | 100% (1/1) | 50% (1/2) | 88.9% (8/9) |

```java
//package main.java;

/**
 * @Author: ljz
 * @Date: 2023/4/4 16:16
 * @Description:
 */
public class BubbleSort {
    public static int[] sort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
        return array;
    }
}

```



| Class | Class, % | Method, % | Line, % |
|---|---|---|---|
| BackPack | 100% (1/1) | 50% (1/2) | 88.9% (8/9) |

```java
//package main.java;

/**
 * @Author: ljz
 * @Date: 2023/4/4 18:13
 * @Description:
 */
public class BackPack {
    public static int maxWeight(int capacity, int[] weights, int[] values) {
        int n = weights.length;
        int[][] dp = new int[n + 1][capacity + 1];

        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= capacity; w++) {
                if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        return dp[n][capacity];
    }
}
```

从上图中不难发现，每一行代码均被覆盖。

## 4.Generate the file structure using tree -d

在windows系统中使用tree命令。

```
PS D:\智算专业课程\软件测试技术\lab6\pitest-ant-example-master> tree
卷 Data 的文件夹 PATH 列表
卷序列号为 2EE7-DC3E
D:.
├─.idea
│  └─libraries
├─lib
├─pitResults
│  ├─202304042113
│  │  └─main.java
│  └─export
│      └─main
│          └─java
│              ├─BackPack
│              │  └─mutants
│              │      ├─0
│              │      ├─1
│              │      ├─10
│              │      ├─11
│              │      ├─12
│              │      ├─13
│              │      ├─14
```

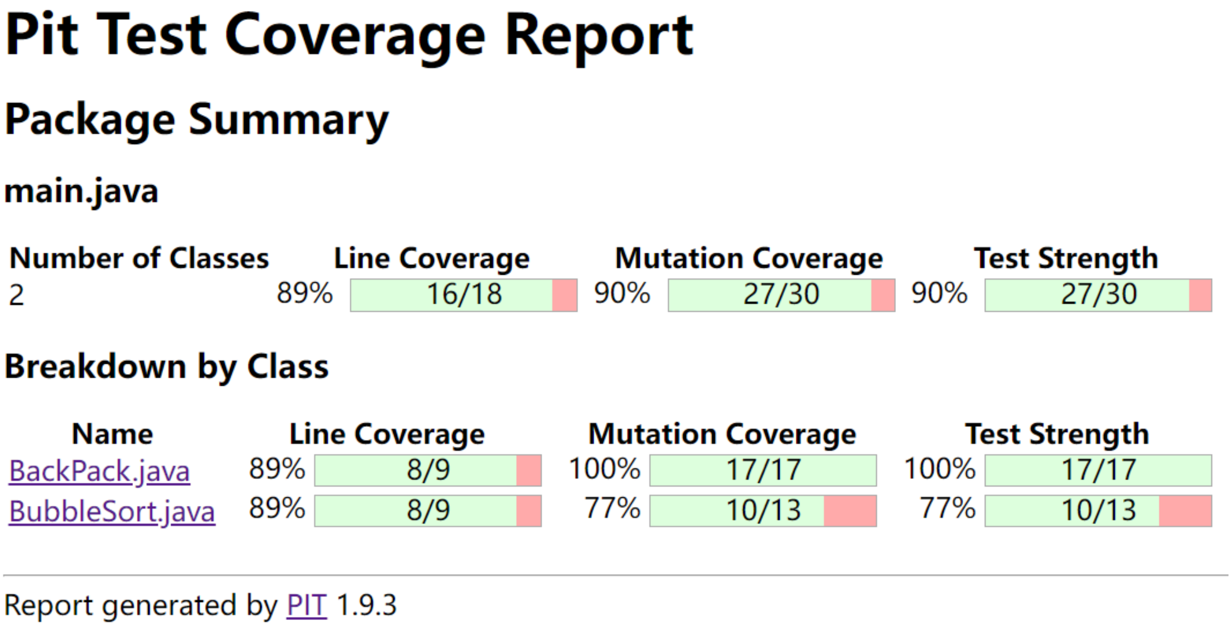
```
|              |      ├─15
|              |      ├─16
|              |      ├─2
|              |      ├─3
|              |      ├─4
|              |      ├─5
|              |      ├─6
|              |      ├─7
|              |      ├─8
|              |      └─9
|              └─BubbleSort
|                  └─mutants
|                      ├─0
|                      ├─1
|                      ├─10
|                      ├─11
|                      ├─12
|                      ├─2
|                      ├─3
|                      ├─4
|                      ├─5
|                      ├─6
|                      ├─7
|                      ├─8
|                      └─9
|
├─src
|  ├─main
|  |  └─java
|  └─test
|      └─java
└─target
    ├─classes
    |  └─main
    |      └─java
    ├─test-classes
    |  ├─main
    |  |  └─java
    |  └─test
    |      └─java
    └─test-result
        └─report
            └─test
                └─java
```

项目的build.xml文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="pit-ant-example">


```

```
    <property name="classOutputDir" value="target"/>
    <!-- classpath for pitest and any plugins -->
    <path id="pitest.path">
        <pathelement location="lib/pitest-1.9.3.jar"/>
        <pathelement location="lib/pitest-entry-1.9.3.jar"/>
        <pathelement location="lib/pitest-ant-1.9.3.jar"/>
    </path>
    <taskdef name="pitest" classname="org.pitest.ant.PitestTask"
classpathref="pitest.path"/>
    <target name="clean">
        <delete dir="${classOutputDir}"/>
        <delete dir="pitResults"/>
    </target>
    <target name="compile" depends="clean">
        <mkdir dir="${classOutputDir}/classes"/>
        <!-- Essential that line numbers and filenames are included in order for PIT
to work -->
        <javac srcdir="src/main/java" includeantruntime="false" debug="true"
debuglevel="source,lines"
               destdir="${classOutputDir}/classes"/>
    </target>
    <!-- classpath for compiling and testing the code. Note it does not include
pitest and it's dependencies -->
    <path id="test.path">
        <pathelement location="${classOutputDir}/classes"/>
        <pathelement location="${classOutputDir}/test-classes"/>
        <pathelement location="lib/hamcrest-all-1.3.jar"/>
        <pathelement location="lib/junit-4.13.2.jar"/>
    </path>

    <target name="test" depends="compile">
        <mkdir dir="${classOutputDir}/test-result"/>
        <mkdir dir="${classOutputDir}/test-classes"/>
        <javac includeantruntime="false" srcdir="src"
destdir="${classOutputDir}/test-classes">
            <classpath refid="test.path"/>
        </javac>
        <junit>
            <classpath refid="test.path"/>
            <batchtest todir="${classOutputDir}/test-result">
                <!-- set test classes -->
                <fileset dir="src">
                    <include name="**/*Test.java"/>
                </fileset>
                <formatter type="xml"/>
            </batchtest>
        </junit>
        <junitreport todir="${classOutputDir}/test-result">
            <fileset dir="${classOutputDir}/test-result">
                <include name="TEST-*.xml"/>
            </fileset>
            <report format="frames" todir="${classOutputDir}/test-result/report"/>
        </junitreport>
    </target>

    <!-- run pitest. note that the filters for tests and classes refer to
package/class names, not source file named -->
    <target name="pit" depends="test">
        <path id="mutation.path">
            <path refid="pitest.path"/>
            <path refid="test.path"/>
        </path>
        <!-- export feature has been activated to write mutants to file -->
        <pitest features="+EXPORT" pitClasspath="pitest.path" threads="2"
classPath="mutation.path"
                targetTests="test.java.*" targetClasses="main.java.*"
reportDir="pitResults" sourceDir="src/main/java"/>
    </target>

</project>
```

## 5.The coverage of the test cases generated by PITest



# Pit Test Coverage Report

## Package Summary

### main.java

| Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| 2 | 89% | 16/18 | 90% | 27/30 | 90% | 27/30 |

## Breakdown by Class

| Name | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| BackPack.java | 89% | 8/9 | 100% | 17/17 | 100% | 17/17 |
| BubbleSort.java | 89% | 8/9 | 77% | 10/13 | 77% | 10/13 |

Report generated by PIT 1.9.3

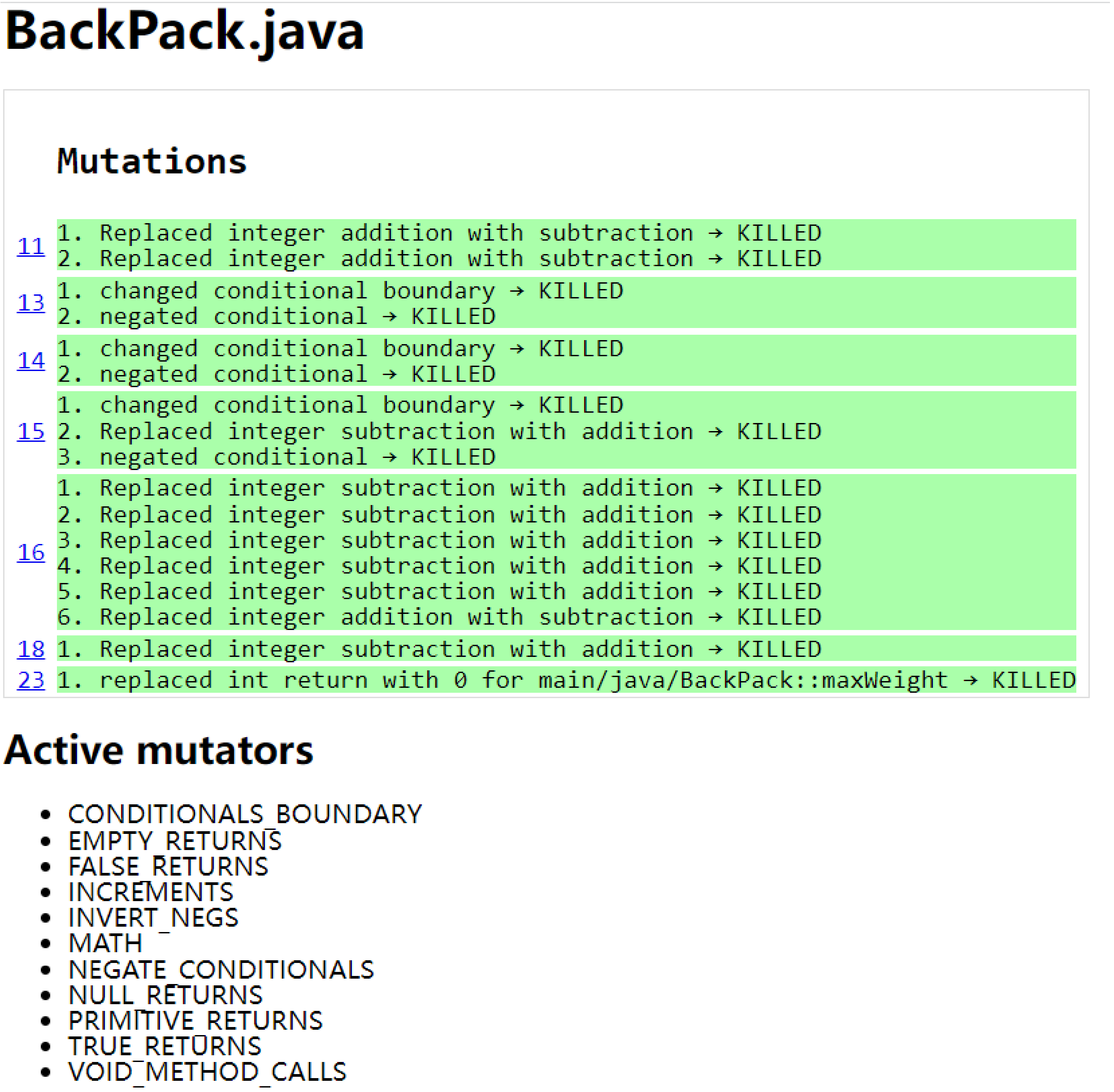
# BubbleSort.java

## Mutations

| | |
|---|---|
| 11 | 1. changed conditional boundary → SURVIVED<br>2. Replaced integer subtraction with addition → SURVIVED<br>3. negated conditional → KILLED |
| 12 | 1. changed conditional boundary → KILLED<br>2. Replaced integer subtraction with addition → KILLED<br>3. Replaced integer subtraction with addition → KILLED<br>4. negated conditional → KILLED |
| 13 | 1. changed conditional boundary → SURVIVED<br>2. Replaced integer addition with subtraction → KILLED<br>3. negated conditional → KILLED |
| 15 | 1. Replaced integer addition with subtraction → KILLED |
| 16 | 1. Replaced integer addition with subtraction → KILLED |
| 20 | 1. replaced return value with null for main/java/BubbleSort::sort → KILLED |

## Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS



# BackPack.java

## Mutations

| | |
|---|---|
| 11 | 1. Replaced integer addition with subtraction → KILLED<br>2. Replaced integer addition with subtraction → KILLED |
| 13 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 14 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 15 | 1. changed conditional boundary → KILLED<br>2. Replaced integer subtraction with addition → KILLED<br>3. negated conditional → KILLED |
| 16 | 1. Replaced integer subtraction with addition → KILLED<br>2. Replaced integer subtraction with addition → KILLED<br>3. Replaced integer subtraction with addition → KILLED<br>4. Replaced integer subtraction with addition → KILLED<br>5. Replaced integer subtraction with addition → KILLED<br>6. Replaced integer addition with subtraction → KILLED |
| 18 | 1. Replaced integer subtraction with addition → KILLED |
| 23 | 1. replaced int return with 0 for main/java/BackPack::maxWeight → KILLED |

## Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

## 6.Analyzing the report

### 6.1对于BubbleSort程序

在整个分析中，总共有13个突变。其中，有3个突变存活，10个突变被杀死。现在我们来逐个分析这些存活突变。

1. 第一个突变是将条件边界更改，从`i < n - 1`变为`i <= n - 1`。这个突变幸存了下来，主要是因为内层循环的限制，i=n-1时，n-i-1=0,内层循环不会执行，也就是说这一突变不会影响程序的正确性。
2. 第二个突变是将整数减法替换为整数加法。`for (int i = 0; i < n - 1; i++)`这个突变也幸存了下来，原因与1类似，同样是因为内层循环条件的限制，此处的突变同样不会影响程序的正确性。
3. 第三个突变同样是将条件边界更改，内层循环的条件从 `j < n - 1 - i` 变为 `j <= n - i - 1`，这意味着对于每次外层循环，内层循环多执行了一次。然而，这个变化对冒泡排序的正确性没有影响。这是因为冒泡排序在每次内层循环中，都会将最大值移动到数组的正确位置。在最后一次内层循环的迭代中，`j` 的值为 `n - i - 2`，而 `j + 1` 的值为 `n - i - 1`。当执行 `j <= n - i - 1` 的额外迭代

时，`j` 的值为 `n - i - 1`，而 `j + 1` 的值为 `n - i`。然而，由于 `i < n - 1`，所以 `n - i` 是一个有效的数组索引。因此，在这个额外的迭代中，代码将比较相邻的元素 `array[n - i - 1]` 和 `array[n - i]`。然而，由于已经执行了 `n - i - 1` 次迭代，因此 `array[n - i - 1]` 和 `array[n - i]` 这两个元素已经是有序的，所以这个额外的迭代不会改变数组的顺序。

**6.2对于BackPack程序**

所有突变体都被标记为 "KILLED"，这意味着对于这些突变体，测试用例都能成功检测到它们。换句话说，测试用例覆盖了这些突变体引入的变化，测试套件的质量相对较高。

## 7.Discuss and explain your results

在本次PITest中，我们针对两个程序（BubbleSort和BackPack）进行了突变测试。分析结果后，我们可以得出以下结论：

针对BubbleSort程序，我们观察到了3个突变存活，但这些突变并未影响程序的正确性。这是因为在这些突变下，代码的逻辑和原始实现在执行时产生的结果是一致的。这意味着虽然测试用例无法覆盖这些突变，但它们对程序的正确性没有影响。然而，我们应该时刻关注这些突变，因为在其他上下文中，类似的突变可能会导致错误。

对于BackPack程序，我们发现所有突变都被成功杀死。这说明我们的测试用例覆盖了所有突变，测试套件的质量较高。这意味着针对这个程序，我们的测试用例能够有效地捕获潜在的错误，从而确保程序的正确性。

总的来说，变异测试是一种非常有价值的技术，它可以帮助我们更好地了解我们的测试用例和测试套件的质量。我们可以根据PITest的结果调整和优化测试用例，从而提高测试的有效性和程序的质量。通过对源代码进行突变，并观察测试用例是否能够捕获这些突变，我们可以发现测试用例的覆盖范围和潜在的不足之处。