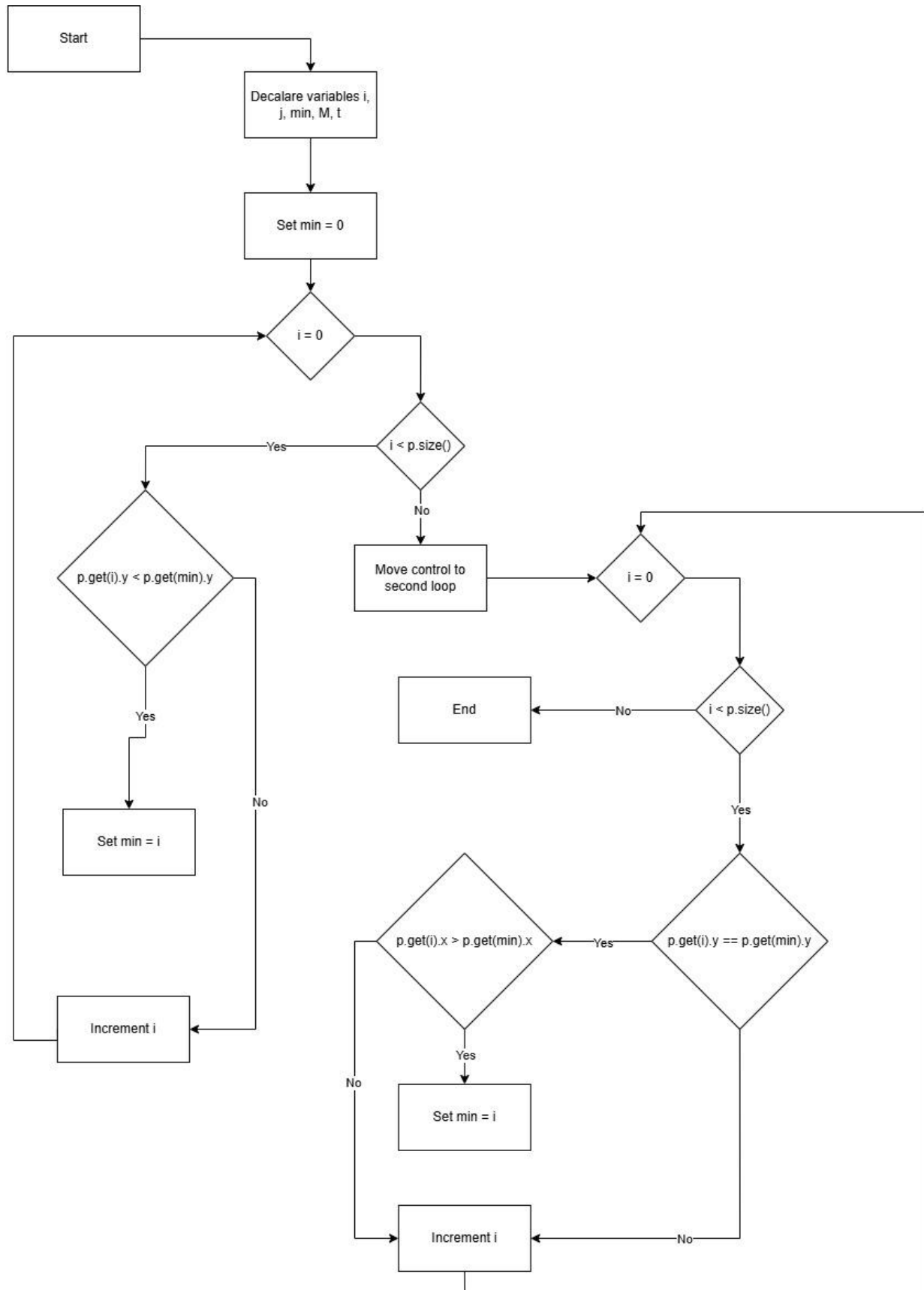# IT 314 – Software Engineering

## Lab 9

### Akshat Jindal | 202201299

**Q.1.** The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code, so the focus is on creating test sets that satisfy some particular coverage criteria.

**1.** Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG). You are free to write the code in any programming language.

```
Start

Decalare variables i,
j, min, M, t

Set min = 0

i = 0

i < p.size()
  Yes
    p.get(i).y < p.get(min).y
      Yes
        Set min = i
      No
        Increment i
  No
    Move control to
    second loop

    i = 0

    i < p.size()
      No
        End
      Yes
        p.get(i).y == p.get(min).y
          Yes
            p.get(i).x > p.get(min).x
              Yes
                Set min = i
              No
                Increment i
          No
            Increment i
```

**2.** Construct test sets for your flow graph that are adequate for the following criteria:

**a. Statement Coverage.**

To achieve statement coverage, every line of code must be executed at least once. The test cases should ensure that:

- The check for an empty vector (if (p.size() == 0)) is executed.
- The for loop is entered and iterated.

**Test Cases:**

1. **Test Case 1:** p is empty (e.g., p = new Vector<Point>();)
   - Expected outcome: The method returns immediately without doing anything.
2. **Test Case 2:** p contains multiple points with varying y-values (e.g., p = new Vector<>(Arrays.asList(new Point(1, 2), new Point(2, 3), new Point(3, 1)));)
   - Expected outcome: The point with the smallest y-coordinate should be moved to the first position in the vector.

**b. Branch Coverage.**

For branch coverage, each branch of each decision (e.g., if statements) needs to be tested. This involves:

- Testing the if (p.size() == 0) condition with an empty vector and a non-empty vector.
- Testing both conditions within the if statement inside the loop: (p.get(i).y < p.get(minY).y) and (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x).

**Test Cases:**

1. **Test Case 1:** Empty vector (p = new Vector<Point>();)
   - ○ Expected outcome: Method returns immediately.
2. **Test Case 2:** Non-empty vector where all points have different y-values (e.g., p = new Vector<>(Arrays.asList(new Point(1, 2), new Point(3, 3), new Point(2, 1)));)
   - ○ Expected outcome: The point with the smallest y-value is moved to the first position.
3. **Test Case 3:** Non-empty vector where multiple points have the same y-value (e.g., p = new Vector<>(Arrays.asList(new Point(1, 2), new Point(2, 2), new Point(0, 2)));)
   - ○ Expected outcome: Among points with the same y-value, the one with the smallest x-value is moved to the first position.

## c. Basic Condition Coverage.

For basic condition coverage, each individual condition within a compound condition should be tested for both true and false. This means we should test cases that cover:

- (p.get(i).y < p.get(minY).y) being true and false.
- (p.get(i).y == p.get(minY).y) being true, and (p.get(i).x < p.get(minY).x) being true and false.

**Test Cases:**

1. **Test Case 1:** Non-empty vector where p.get(i).y < p.get(minY).y is true (e.g., p = new Vector<>(Arrays.asList(new Point(3, 4), new Point(1, 2)));)
   - ○ Expected outcome: The point with the smallest y-value is moved to the first position.
2. **Test Case 2:** Non-empty vector where p.get(i).y == p.get(minY).y is true, and p.get(i).x < p.get(minY).x is also true (e.g., p = new Vector<>(Arrays.asList(new Point(3, 2), new Point(1, 2)));)
   - ○ Expected outcome: The point with the smallest x-value among those with the same y-value is moved to the first position.
3. **Test Case 3:** Non-empty vector where p.get(i).y == p.get(minY).y is true, and p.get(i).x < p.get(minY).x is false (e.g., p = new Vector<>(Arrays.asList(new Point(1, 2), new Point(3, 2)));)

○ Expected outcome: The first point remains in position as it has the smallest x-value.

**3.** For the test set you have just checked can you find a mutation of the code (i.e. the deletion, change or insertion of some code) that will result in failure but is not detected by your test set. You have to use the mutation testing tool

Ans:

**Mutation 1: Change Comparison Operator in p.get(i).y < p.get(minY).y**

**Mutation**: Change p.get(i).y < p.get(minY).y to p.get(i).y <= p.get(minY).y.

**Explanation**: This change would mean that the algorithm would potentially select the last instance of the smallest y instead of the first, which could impact the correctness of the hull.

**Analysis**: Our current tests with points having unique y values and points with the same y but different x values might not catch this mutation directly because:

- We have no test where two points have both equal x and y values.
- We assume a strict less-than comparison.

**New Test Case**:

- **Input**: p = new Vector<>(Arrays.asList(new Point(1, 2), new Point(1, 2), new Point(0, 2)));
- **Expected Outcome**: The first occurrence of (1, 2) should stay at the front.
- **Reasoning**: This will ensure the code behaves correctly under strict y ordering.

**Mutation 2: Removing the if (p.size() == 0) Check**

**Mutation**: Delete the check if (p.size() == 0), allowing the function to proceed even if the vector is empty.

**Explanation**: Without this check, calling the method with an empty vector would result in an IndexOutOfBoundsException when attempting to access elements in the empty list.

**Analysis**: Our current test set would catch this mutation, as **Test Case 1** is an empty vector test, which would fail with this mutation.

## Mutation 3: Removing or Swapping the First Element Swap Operation

**Mutation**: Remove or incorrectly swap p.set(0, p.get(minY)); and p.set(minY, temp);.

**Explanation**: This mutation would result in an incorrect ordering of the points, as the point with the smallest y (or smallest x if y values are equal) might not be placed in the first position.

**Analysis**: Our current test set should catch this mutation, as **Test Case 2** requires the smallest y point to move to the first position, and it would fail if this didn't occur correctly.

## Mutation 4: Changing Condition to Only Consider x Without y

**Mutation**: Change the compound condition to consider only x values without regard to y, e.g., change if (p.get(i).y < p.get(minY).y || (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x)) to if (p.get(i).x < p.get(minY).x).

**Explanation**: This mutation would ignore the y ordering completely, leading to incorrect results if points differ in y but not in x.

**Analysis**: Our current tests might miss this mutation because we lack tests where:

- The smallest x isn't necessarily the smallest y.
- We have cases where x changes without y changes.

**New Test Case**:

- **Input**: p = new Vector<>(Arrays.asList(new Point(3, 2), new Point(1, 3), new Point(2, 1)));
- **Expected Outcome**: The point (2, 1) should be placed first, as it has the smallest y, rather than the point with the smallest x alone.

**4.** Create a test set that satisfies the path coverage criterion where every loop is explored at least zero, one or two times.

<span style="color:red">Ans:</span>

To achieve **path coverage** for the doGraham method, we need to cover paths that:

1. Skip the loop entirely.
2. Enter the loop once.
3. Enter the loop multiple times (specifically twice, as per the criterion).

**Test Case 1: Empty Vector (Loop is Skipped)**

**Input**: An empty vector, p = new Vector<Point>();

- **Expected Outcome**: The method returns immediately without entering the loop.
- **Reasoning**: This case covers the path where p.size() == 0, so the for loop is skipped entirely.

**Test Case 2: Vector with One Point (Loop Executes Zero Times)**

**Input**: A vector with one point, e.g., p = new Vector<>(Arrays.asList(new Point(1, 1)));

- **Expected Outcome**: The method should handle this without errors, and no swapping should occur.
- **Reasoning**: Since there is only one point, the for loop initializes but does not execute any iterations, as i = 1 exceeds p.size() - 1.

**Test Case 3: Vector with Two Points (Loop Executes Once)**

**Input**: A vector with two points, e.g., p = new Vector<>(Arrays.asList(new Point(2, 3), new Point(1, 2)));

- **Expected Outcome**: The point with the lowest y value (or lowest x if y values are the same) is moved to the first position.

- **Reasoning**: This case covers the path where the for loop executes exactly once.

# Lab Execution

**Q1. After generating the control flow graph, check whether your CFG match with the CFG generated by Control Flow Graph Factory Tool and Eclipse flow graph generator.**

Ans:

 Control Flow Graph Factory :- **YES**

Eclipse flow graph generator :- **YES**

**Q2. Devise the minimum number of test cases required to cover the code using the aforementioned criteria.**

Ans:

**Statement Coverage :** 2 test cases

**Branch Coverage:** 3 test cases

**Basic Condition Coverage:** 3 test cases

**Path Coverage:** 3 test cases

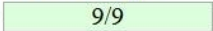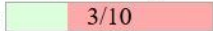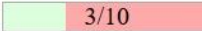**Total: 2 (Statement) + 3 (Branch) + 3 (Basic Condition) + 3 (Path) = 11 test cases**

# Pit Test Coverage Report

## Package Summary

### default

| Number of Classes | Line Coverage | Mutation Coverage | Test Strength |
|---|---|---|---|
| 1 | 100% 9/9 | 30% 3/10 | 30% 3/10 |

### Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|---|---|---|---|
| ConvexHull.java | 100% 9/9 | 30% 3/10 | 30% 3/10 |

Report generated by PIT 1.15.8

# ConvexHull.java

```java
1    import java.util.Vector;
2
3    class ConvexHull {
4        Vector<Point> doGraham(Vector<Point> p) {
5            int i, j, min, M;
6            Point t;
7            min = 0;
8
9            // search for minimum:
10 2         for (i = 1; i < p.size(); ++i) {
11 2             if (p.get(i).y < p.get(min).y) {
12                     min = i;
13                 }
14             }
15
16             // continue along the values with the same y component
17 2         for (i = 0; i < p.size(); ++i) {
18 3             if ((p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x)) {
19                     min = i;
20                 }
21             }
22
23 1         return p; // Placeholder return; actual return might vary
24         }
25    }
```

## Mutations

| | |
|---|---|
| 10 | 1. changed conditional boundary → KILLED |
|    | 2. negated conditional → SURVIVED |
| 11 | 1. negated conditional → SURVIVED |
|    | 2. changed conditional boundary → SURVIVED |
| 17 | 1. changed conditional boundary → KILLED |
|    | 2. negated conditional → SURVIVED |
| 18 | 1. negated conditional → SURVIVED |
|    | 2. changed conditional boundary → SURVIVED |
|    | 3. negated conditional → SURVIVED |
| 23 | 1. replaced return value with null for ConvexHull::doGraham → KILLED |

## Active mutators

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

## Tests examined

- ConvexHullTest.[engine:junit-jupiter]/[class:ConvexHullTest]/[method:testDoGraham_ConditionCoverage_YEqualAndXLargerThanMinX()] (0 ms)
- ConvexHullTest.[engine:junit-jupiter]/[class:ConvexHullTest]/[method:testDoGraham_StatementCoverage()] (2 ms)
- ConvexHullTest.[engine:junit-jupiter]/[class:ConvexHullTest]/[method:testDoGraham_BranchCoverage_SameYValues()] (2 ms)
- ConvexHullTest.[engine:junit-jupiter]/[class:ConvexHullTest]/[method:testDoGraham_ConditionCoverage_YSmallerThanMinY()] (2 ms)
- ConvexHullTest.[engine:junit-jupiter]/[class:ConvexHullTest]/[method:testDoGraham_BranchCoverage_DifferentYValues()] (75 ms)