IT-314 : Software Engineering

Lab – 09 :

Software Testing

Lab Session – Mutation Testing

Name: Parth Parekh

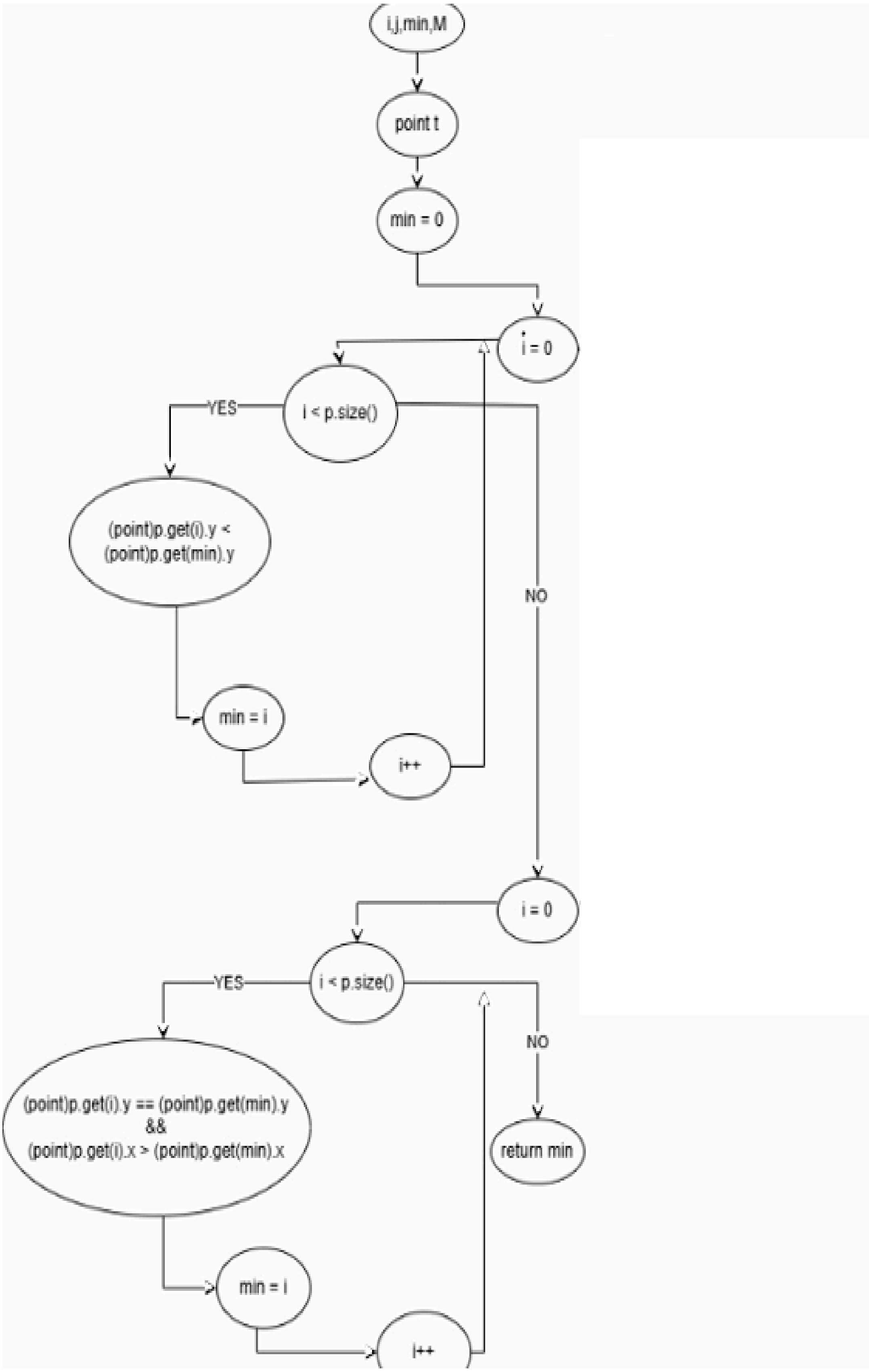
Student ID: 202201200

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.**

```
public class Point {
	double x;
	double y;
	public:
	Point(double x, double y){
	this.x = x;
	this.y= y;
	}
}
public class ConvexHull {
	public void doGraham(Vector<Point> p){
	if (p.size()== 0){
	return;
	}
	int minY= 0;
	for (int i= 1; i < p.size(); i++) {
	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)) {
	minY= i;
	}
	}
	Point temp= p.get(0);
	p.set(0, p.get(minY));
	p.set(minY, temp);
	}
}
```

<u>For the given code fragment, you should carry out the following activities.</u>

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.


i,j,min,M
point t
min = 0
i = 0
i < p.size()
YES
NO
(point)p.get(i).y <
(point)p.get(min).y
min = i
i++
i = 0
i < p.size()
YES
NO
(point)p.get(i).y == (point)p.get(min).y
&&
(point)p.get(i).x > (point)p.get(min).x
return min
min = i
i++

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

a. Statement Coverage.

Test Cases for Statement Coverage

1. Test Case 1: p is an empty vector .
   ○ Expected Behaviour : The method should return immediately without performing any operations.
2. Test Case 2: p contains a single point, e.g., [(0, 0)].
   ○ Expected Behaviour : No swap is needed as there's only one point, but all statements should be executed.
3. Test Case 3: p contains multiple points with distinct y values, e.g., [(1, 3), (2, 2), (3, 4)].
   ○ Expected Behaviour : The method should identify the point (2, 2) as the minimum and swap it with the point at index 0.
4. Test Case 4: p contains points with the same y value, but different x values eg. [(3, 2), (1, 2), (2, 2)].
   ○ Expected Behaviour : Among the points with the same y value, the one with the smallest x value (1, 2) should be identified as the minimum and swapped with the first point.

b. Branch Coverage.

Test Cases for Branch Coverage:

1. Test Case 1: p is an empty vector (tests the if (p.size() == 0) branch).
2. Test Case 3 from Statement Coverage: Multiple points with distinct y values (tests p.get(i).y < p.get(minY).y in the loop).
3. Test Case 4 from Statement Coverage: Points with the same y value but different x values (tests (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x) branch).

c. Basic Condition Coverage.

Test Cases for Basic Condition Coverage

1. Test Case 3: Distinct y values (tests the p.get(i).y < p.get(minY).y condition to be true and false).
2. Test Case 4: Same y values, varying x values (tests both p.get(i).y == p.get(minY).y and p.get(i).x < p.get(minY).x conditions as true and false).

Summary of Test Cases :

Test Case	Input Points (Vector p)	Expected Behaviour
1	[]	Method returns immediately.
2	[(0, 0)]	No swap; single point remains unchanged.
3	[(1, 3), (2, 2), (3, K)]	Point (2, 2) is swapped with (1, 3).
4	[(3, 2), (1, 2), (2, 2)]	Point (1, 2) is swapped with (3, 2).

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.

Mutation 1: Change the Comparison Operator for the y Coordinate

- **Original Code** :

```
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))
```

- **Mutation: Change < to <= for y comparison**

```
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))
```

Impact: This mutation may not always be detected if the minimum y values have different x values because it doesn't affect the outcome unless there are points with the same y value.
Detection: The current test cases might not detect this mutation if there is only one point with the smallest y coordinate, as the outcome would be the same. To detect it, we would need additional test cases with multiple points having the same minimum y value to ensure that only the correct minimum x is chosen.

- **Mutation 2:** Change the x Comparison Operator
- **Original Code :**

```
(p.get(i).y== p.get(minY).y && p.get(i).x < p.get(minY).x)
```

- **Mutation: Change < to > :**

```
(p.get(i).y == p.get(minY).y && p.get(i).x > p.get(minY).x)
```

<u>Impact</u>: This mutation would reverse the logic for choosing the x coordinate when y values are equal. The mutated code would incorrectly choose the point with the larger x instead of the smaller one.
<u>Detection</u>: Our current test case with points having the same y values and different x values (Test Case 4) would likely detect this mutation because it relies on selecting the smallest x value. However, if our test suite lacks cases where y values are the same, this mutation might go undetected.

Mutation 3: Remove the if (p.size() == 0) Check

- **Original Code :**

```
if (p.size() == 0) {

    return;

}
```

- **<u>Mutation</u>**: Delete the line if (p.size() == 0) { return; }.

<u>Impact</u>: Removing this check would cause the method to try accessing elements in an empty vector, resulting in an IndexOutOfBoundsException if the vector is empty.
<u>Detection</u>: Our current test case for an empty vector (p=`[]`) would detect this mutation, as it would throw an exception.

<u>Mutation 4</u>: Modify the Swap Logic

- **Original Code :**

```
Point temp = p.get(0);
```

```
p.set(0, p.get(minY));

p.set(minY, temp);
```

- **Mutation**: Swap p.set(0, p.get(minY)) and p.set(minY, temp) lines

```
Point temp = p.get(0);


p.set(minY, temp);


p.set(0, p.get(minY));
```

Impact: This would break the intended swapping logic, leaving the vector unchanged if minY == 0 or leading to unexpected results otherwise.
Detection: Our current test cases would likely detect this mutation when minY != 0, as the final arrangement of points would be incorrect. However, if the minimum y point is already at index 0, the mutation might go undetected. To ensure detection, we should have cases where minY is not equal to 0.

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

- **Test Case 1:** Loop Explored Zero Times

Input: An empty vector p.
Test Code :

```
Vector<Point> p = new Vector<Point>();

ConvexHull ch = new ConvexHull();

ch.doGraham(p);
```

Expected Result: The method should return immediately without any processing.
Explanation: Since p.size() is 0, the method will hit the if (p.size() == 0) { return; } condition and exit immediately, skipping the rest of the code.

Coverage:
- **Statement Coverage:** Covers the return statement for an empty vector .

● **Branch Coverage:** Covers the if (p.size() == 0) branch where the condition is true.
● **Condition Coverage:** Fully evaluates the condition p.size() == 0 as true.

● **Test Case 2:** Loop Explored Once
Input: A vector with one point, (0, 0).
Test Code :

```
Vector<Point> p = new Vector<Point>();

p.add(new Point(0, 0));

ConvexHull ch = new ConvexHull();

ch.doGraham(p);
```

Expected Result: Since p.size() is 1, the loop should not be entered, and the method should effectively swap the point with itself, leaving the vector unchanged.
Explanation: With only one point, the loop starting from for (int i = 1; i < p.size(); i++) is not entered because i = 1 is equal to p.size(). Thus, the minimum y point is already at the first position, and swapping it with itself does nothing.

Coverage:
● **Statement Coverage:** Covers the initialization of minY and the swap logic.
● **Branch Coverage:** Covers the if (p.size() == 0) branch where the condition is false.
● **Condition Coverage:** Evaluates the loop condition for i = 1 as false, covering the scenario where p.size() == 1.

● **Test Case 3:** Loop Explored Twice
Input: A vector with two points where the first point (1,1) has a higher y-coordinate than the second point (0, 0).

Test Code :

```java
Vector<Point> p = new Vector<Point>();

p.add(new Point(1, 1));

p.add(new Point(0, 0));

ConvexHull ch = new

ConvexHull(); ch.doGraham(p);
```

Expected Result: The method should enter the loop and find that the second point has a lower y-coordinate than the first, updating minY to 1. A swap should occur, placing (0, 0) at the beginning of the vector .
Explanation: The loop iterates once, comparing the two points. Since (0, 0).y is 1).y, minY is updated to 1. The method then swaps the first point with the point at minY.

Coverage:

• **Statement Coverage:** Covers all statements, including the loop, comparison, minY update, and swap logic.
• **Branch Coverage:** Covers both branches of the if condition within the loop, where p.get(i).y < p.get(minY).y is true.
• **Condition Coverage:** Evaluates both conditions within the if statement: p.get(i).y < p.get(minY).y (true) and (p.get(i).y == p.get(minY).y && p.get(i).x < p.get(minY).x) (not evaluated in this case as the first condition is true).

• **Test Case 4:** Loop Explored More Than Twice

Input : This test case verifies that the algorithm correctly identifies the point with the lowest y-coordinate among multiple points in the vector and performs the necessary swap.

Test Code :

```java
Vector<Point> p = new Vector<Point>();

p.add(new Point(2, 2)); // Point A

p.add(new Point(1, 0)); // Point B (minimum y-coordinate)
```

```
p.add(new Point(0, 3)); // Point C

ConvexHull ch = new ConvexHull();

ch.doGraham(p);
```

Expected Result : The method should iterate through all three points in the vector and identify the second point (1, 0) as having the lowest y-coordinate of 0. Consequently, minY will be updated to reflect this value. After performing the necessary swap, the point (1, 0) will be positioned at the front of the vector, leading to the final order of points being (1, 0), (2, 2), (0, 3).

Explanation : Initially, minY is set to the y-coordinate of the first point (2, 2), which is 2. As the loop begins, it compares the first point (2, 2) with the second point (1, 0). Since (1, 0).y (0) is less than (2, 2).y (2), minY is updated to 0, and minIndex is set to 1. In the subsequent iteration, the method compares (1, 0) with the third point (0, 3). Here, (0, 3).y (3) is greater than (1, 0).y (0), so no changes are made to minY or minIndex. After the loop concludes, the algorithm swaps the first point (2, 2) with the point at minIndex, resulting in the updated vector order .

Coverage:

• Statement Coverage: All statements executed, including the loop and swap.
• Branch Coverage: Both branches of the if condition are covered:
○ p.get(i).y < p.get(minY).y (true).
○ p.get(i).y >= p.get(minY).y (evaluated).
• Condition Coverage: Evaluates:
○ p.get(i).y < p.get(minY).y (true).
○ p.get(i).y >= p.get(minY).y (evaluated).

Lab Execution:

1. After generating the control flow graph, check whether your CFG matches with the CFG generated by Control Flow Graph Factory Tool and Eclipse flow graph generator . (In your submission document, mention only “Yes” or “No” for each tool).

Control Flow Graph Factory :- YES
Eclipse flow graph generator :- YES

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

1. Branch Coverage: 4 test cases
2. Basic Condition Coverage: 3 test cases
3. Path Coverage: 2 test cases

Summary of Minimum Test Cases:
- Total: 4 (Branch) + 3 (Basic Condition) + 2 (Path) = 9 test cases