Software Enginnering (IT314)

LAB 9

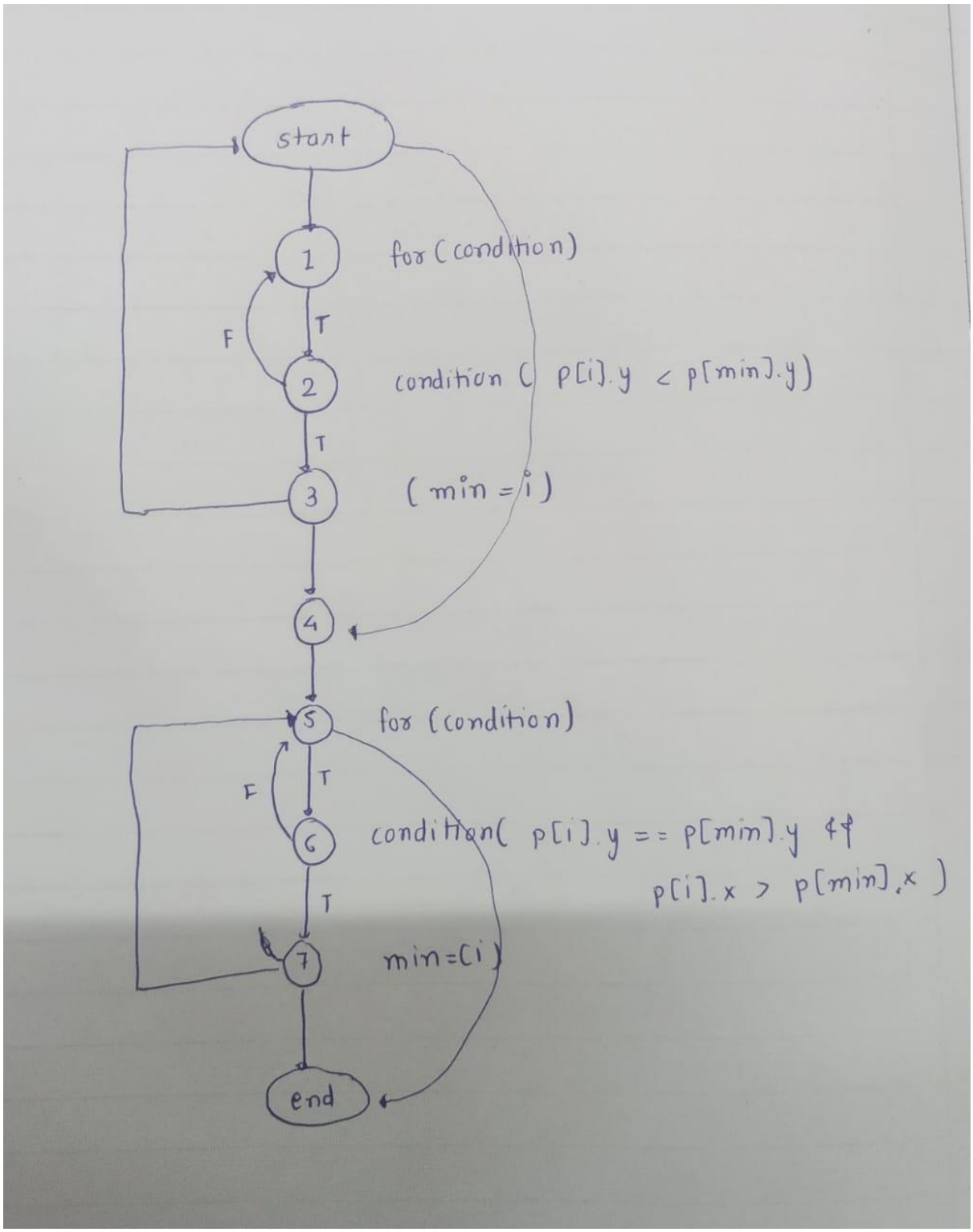
Kishan Pansuriya (202201504)

Code

```
import java.util.Vector;

class Point {
    int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class GrahamScan {

    public Vector<Point> doGraham(Vector<Point> p) {
        int min = 0;

        // Step 1: Find the point with the minimum y-coordinate
        for (int i = 1; i < p.size(); i++) {
            if (p.get(i).y < p.get(min).y) {
                min = i;
            }
        }

        // Step 2: If there are multiple points with the same minimum y-
coordinate,
        // find the one with the largest x-coordinate
        for (int i = 0; i < p.size(); i++) {
            if (p.get(i).y == p.get(min).y && p.get(i).x > p.get(min).x) {
                min = i;
            }
        }

        // The rest of the doGraham algorithm would go here
        // Return or process the points vector as needed

        return p; // This would typically return the convex hull or other
result
    }

    public static void main(String[] args) {
        Vector<Point> points = new Vector<>();
        points.add(new Point(1, 2));
        points.add(new Point(2, 3));
        points.add(new Point(3, 1));
        points.add(new Point(4, 2));
```

```
        GrahamScan grahamScan = new GrahamScan();
        Vector<Point> result = grahamScan.doGraham(points);

        // Print out the result (for demonstration purposes)
        for (Point point : result) {
            System.out.println("Point: (" + point.x + ", " + point.y + ")");
        }
    }
}
```

Control Flow Diagram



Test Sets for Code Coverage

1. Statement Coverage

- **Test Case 1:**
  - **Input:** [(1, 2), (2, 1), (3, 3)]
  - **Expected Outcome:** min updated to (2, 1).

2. Branch Coverage

- **Test Case 1 (min y updated):**
  - **Input:** [(1, 2), (2, 1), (3, 3)]
  - **Outcome:** min points to (2, 1).
- **Test Case 2 (no update):**
  - **Input:** [(1, 2), (2, 2), (3, 3)]
  - **Outcome:** min remains (1, 2).
- **Test Case 3 (tie-breaking on x):**
  - **Input:** [(1, 2), (2, 2), (3, 2)]
  - **Outcome:** min points to (3, 2).

3. Basic Condition Coverage

- **Test Case 1 (Condition 1 true, Condition 2 not checked):**
  - **Input:** [(1, 2), (2, 1), (3, 3)]
- **Test Case 2 (Condition 1 false):**
  - **Input:** [(1, 2), (2, 2), (3, 3)]
- **Test Case 3 (Condition 2 true, Condition 3 true):**
  - **Input:** [(1, 2), (2, 2), (3, 2)]
- **Test Case 4 (Condition 2 true, Condition 3 false):**
  - **Input:** [(3, 2), (1, 2), (2, 2)]

Mutants

1. Relational Operator Replacement

Original Line:

```
if (p.get(i).y < p.get(min).y) {
```

Mutant:

```
if (p.get(i).y <= p.get(min).y) {
    min = i;
}
```

2. Relational Operator Replacement

Original Line:

```
if (p.get(i).y == p.get(min).y && p.get(i).x > p.get(min).x) {
```

Mutant:

```
if (p.get(i).y != p.get(min).y && p.get(i).x > p.get(min).x) {
    min = i;
}
```

3. Conditional Operator Replacement

Original Line:

```
if (p.get(i).y == p.get(min).y && p.get(i).x > p.get(min).x) {
```

Mutant:

```
if (p.get(i).y == p.get(min).y || p.get(i).x > p.get(min).x) {
    min = i;
}
```

4. Arithmetic Operator Replacement

Original Line:

```
for (int i = 1; i < p.size(); i++) {
```

```
}
```

Mutant:

```
for (int i = 1; i < p.size(); i--) {
    if (p.get(i).y < p.get(min).y) {
        min = i;
    }
}
```

5. Statement Deletion

Original Line:

```
min = i;
```

Mutant:

```
// min = i; // Removed
```

6. Logical Negation

Original Line:

```
if (p.get(i).y == p.get(min).y && p.get(i).x > p.get(min).x) {
```

Mutant:

```
if (!(p.get(i).y == p.get(min).y && p.get(i).x > p.get(min).x)) {
    min = i;
}
```

7. Constant Replacement

Original Line:

```
int min = 0;
```

Mutant:

```
int min = 1;
```