202201502
Tushar S. Mori
IT-313

Lab - 9 Mutation Testing

Question 1:
Given logic converted to java code:

```
class Point {
    int x;
    int y;

    public Point() {
        this.x = 0;
        this.y = 0;
    }

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

public class Main {
    public static Point doGraham(Point[] p) {
        int min = 0;

        for (int i = 0; i < p.length; i++) {
            if (p[i].y < p[min].y) {
                min = i;
            }
        }

        for (int i = 0; i < p.length; i++) {
            if (p[i].y == p[min].y && p[i].x > p[min].x) {
                min = i;
            }
```

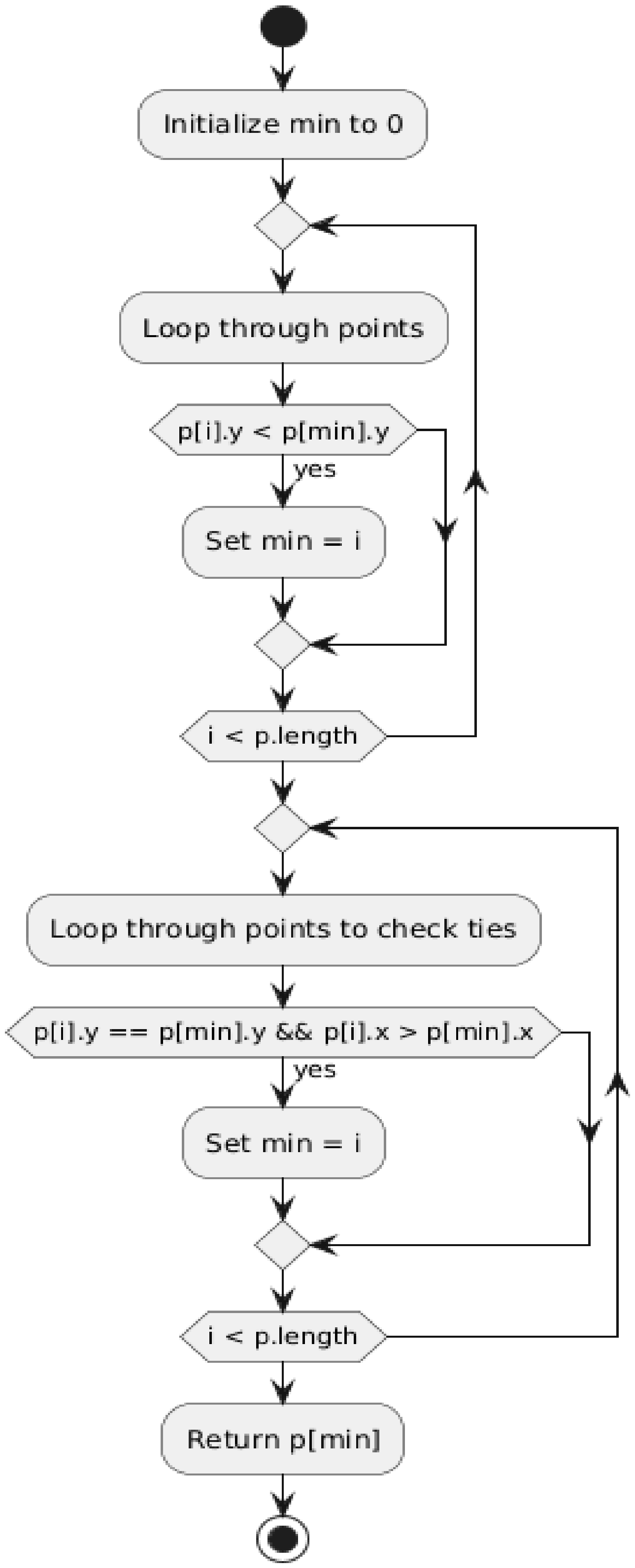
```java
        }

        return p[min];
    }

    public static void main(String[] args) {
        Point[] p = {
            new Point(3, 1),
            new Point(1, 1),
            new Point(1, 3),
            new Point(2, 4),
            new Point(1, 2),
            new Point(5, 1)
        };

        Point acquiredPoint = doGraham(p);
        System.out.println("Acquired Point: x=" + acquiredPoint.x + "
y=" + acquiredPoint.y);
    }
}
```

Here is the control flow diagram created by me for the same:



Yes the control flow diagram is similar to the ones generated using tools

Question 2:

a. Statement Coverage

Objective: Ensure every statement is executed at least once.

Test Set

1. Input: p = [(1, 2), (3, 1), (5, 4)]
    - This test will execute all statements. It sets min to the point (3,1) after the first loop and checks for ties in the second loop without updating min.

b. Branch Coverage

Objective: Ensure that each possible branch (true/false) of each decision point is taken at least once.

Test Set

1. Input 1: p = [(1, 2), (3, 1), (5, 4)]
    - Covers:
        - p[i].y < p[min].y as true when i=1.
        - p[i].y < p[min].y as false for other points.
        - p[i].y == p[min].y && p[i].x > p[min].x as false for all points in the second loop.
2. Input 2: p = [(3, 1), (1, 1), (5, 4)]
    - Covers:
        - p[i].y == p[min].y && p[i].x > p[min].x as true, updating min to the point with the highest x-coordinate among ties (in this case, (3,1)).

c. Basic Condition Coverage

Objective: Ensure that each basic condition (each part of a compound condition) is true and false at least once.

Test Set

1. Input 1: p = [(1, 3), (3, 1), (2, 2)]
    - Ensures:
        - p[i].y < p[min].y is both true (when i=1) and false.
        - p[i].y == p[min].y is false in the second loop.
2. Input 2: p = [(3, 1), (5, 1), (2, 4)]
    - Ensures:
        - p[i].y == p[min].y is true for the tie condition.

        - p[i].x > p[min].x is true in the second loop, selecting (5,1) over (3,1).

Question 3:

1. Deletion Mutation

- **Mutation Description**: Remove the initialization line min = 0; at the beginning of the method.
- **Expected Behavior**: Without initializing min, the doGraham function may produce an undefined or random starting index. This could lead to incorrect selection during comparisons in both loops, as the code relies on a clear starting point for min.
- **Impact on Test Cases**: This mutation might go undetected if the uninitialized min variable defaults to 0 or another harmless value. However, the test cases with different points would ideally fail as they cannot rely on the initial minimum point being accurately established. Including a test where the starting point varies would help detect this mutation.

2. Change Mutation

Mutation Description: Modify the condition in the first if statement from < to <=:
if (p[i].y <= p[min].y)

- **Expected Behavior**: This change will make the code consider points with the same y-coordinate as the minimum, potentially resulting in an inaccurate selection when y-values are tied but should not affect min.
- **Impact on Test Cases**: This mutation can be detected by test cases where multiple points have identical y-coordinates but different x-coordinates. For instance, a test case with points like (3,1), (1,1), and (5,1) could identify the error as it would lead to an incorrect choice when doGraham is supposed to prioritize the point with a strictly lower y-coordinate.

3. Insertion Mutation

- **Mutation Description**: Insert an additional line min = i; at the end of the second loop.

Example insertion:
```
for (int i = 0; i < p.length; i++) {
    if (p[i].y == p[min].y && p[i].x > p[min].x) {
        min = i;
    }
}
min = i; // New insertion line
```

  -

- **Expected Behavior**: By inserting min = i after the loop, the final index in p could override any previous minimum selection, causing min to reflect the last point in p. This might result in the doGraham function returning an incorrect point, especially if the minimum y-coordinate is not at the end of the array.
- **Impact on Test Cases**: This mutation might pass undetected if the final point in p is already the intended minimum. However, introducing a test case where the last point has a non-minimal y or x value would detect this fault, as the expected outcome would not align with the erroneous result produced by this mutation.

Question 4:

Test Case 1: Zero Iterations

- Input: p = [] (an empty array)
- Description: An empty input ensures that neither loop executes, as there are no elements to iterate over.
- Expected Output: The function should handle this case gracefully (e.g., return null or throw an exception indicating no points are present).

Test Case 2: One Iteration in First Loop Only

- Input: p = [(3, 4)] (a single point)
- Description: With only one point, the first loop runs exactly once and sets min to 0. The second loop does not run as there are no other points to check for ties.
- Expected Output: The function should return the only point, (3, 4).

Test Case 3: One Iteration in Both Loops

- Input: p = [(1, 2), (3, 2)] (two points with the same y-coordinate but different x-coordinates)
- Description: The first loop will run twice to identify the minimum y-coordinate, and the second loop will run once to check for the tie on the y-coordinate and pick the point with the larger x-coordinate.
- Expected Output: The function should return (3, 2), as it has the largest x among points with the minimum y.

Test Case 4: Two Iterations in First Loop Only

- Input: p = [(3, 1), (5, 4), (2, 1)] (multiple points, two with the same minimum y-coordinate)
- Description: The first loop will run three times to identify the minimum y-coordinate point (at index 0 or 2), while the second loop does not need to run as there is no tie to resolve.
- Expected Output: The function should return (2, 1) since it has the highest x-coordinate among points with the minimum y.

Test Case 5: Two Iterations in Both Loops

- Input: p = [(1, 1), (4, 1), (3, 2)] (multiple points with at least two points sharing the minimum y-coordinate)
- Description: The first loop runs three times to find the minimum y-coordinate point. The second loop runs twice to check for ties, selecting the point with the highest x-coordinate among those with the minimum y.
- Expected Output: The function should return (4, 1) since it has the highest x-coordinate among points with the minimum y.