IT313: Software Engineering
Lab Session – Mutation Testing
Harshal Patel - 202201070

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.
Code:

```
Vector doGraham(Vector p) {
        int i,j,min,M;

        Point t;
        min = 0;

        // search for minimum:
        for(i=1; i < p.size(); ++i) {
            if( ((Point) p.get(i)).y <
                        ((Point) p.get(min)).y )
            {
                min = i;
            }
        }

        // continue along the values with same y component
        for(i=0; i < p.size(); ++i) {
            if(( ((Point) p.get(i)).y ==
                        ((Point) p.get(min)).y ) &&
                 (((Point) p.get(i)).x >
                        ((Point) p.get(min)).x ))
            {
                 min = i;
            }
        }
```

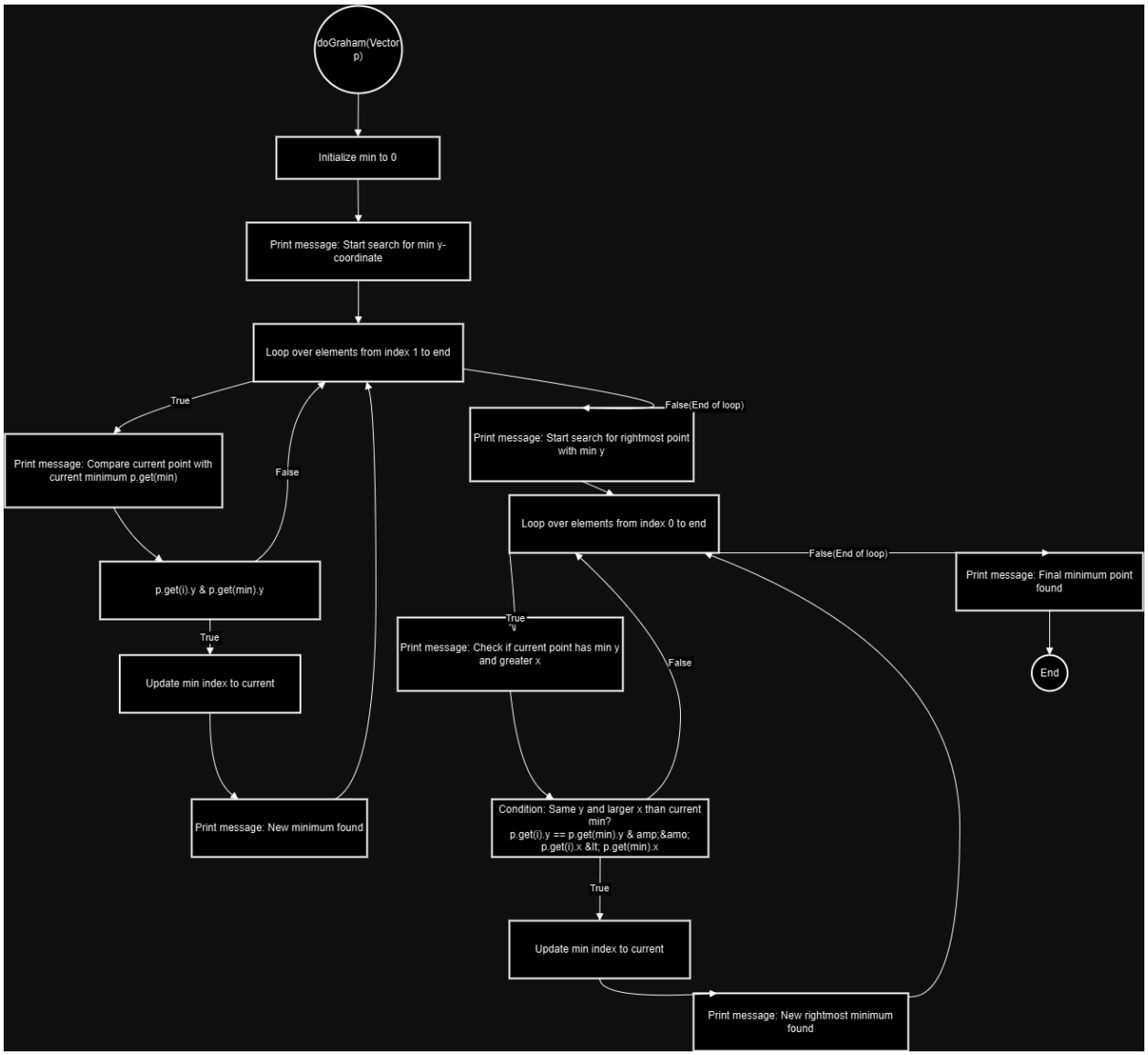
1. Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG).
You are free to write the code in any programming language.

Executable Code:

```
import java.util.Vector;

class Point {
    int x, y;

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

public class ConvexHull {

    public int doGraham(Vector<Point> p) {
        int min = 0;
        for (int i = 1; i < p.size(); i++) {
            if (p.get(i).y < p.get(min).y) {
                min = i;
            }
        }
        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;
            }
        }

        return min;
    }

    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(0, 2));
        points.add(new Point(3, 3));

        ConvexHull convexHull = new ConvexHull();
        int minIndex = convexHull.doGraham(points);
```

```
        System.out.println("Index of minimum point: " + minIndex);
    }
}
```

Control Flow Diagram:



2. Construct test sets for your flow graph that are adequate for the following criteria:
a. Statement Coverage.
b. Branch Coverage.
c. Basic Condition Coverage.

Test Set for Statement Coverage

1. **Test Case 1**:
    - **Input**: `p = [(3, 4), (1, 5), (2, 3)]`
    - **Explanation**: This set contains points with distinct y-coordinates

2. Test Case 2:

    - **Input**: `p = [(5, 5), (5, 5), (6, 6)]`
    - **Explanation**: Points have the same y-coordinates, which will help the condition to find the rightmost minimum point

2. Branch Coverage

Covering all the if conditions in the loops.

Test Set for Branch Coverage

1. **Test Case 1**:
    - **Input**: `p = [(3, 7), (2, 5), (4, 6)]`
    - **Explanation**: This will take the true branch in the first loop when y is smaller than `min` and will ensure all statements are covered for the first loop.
2. **Test Case 2**:
    - **Input**: `p = [(4, 5), (4, 5), (7, 5)]`
    - **Explanation**: This set has points with the same y-coordinates, ensuring that the branch chasks for equal y values and larger x values when the second loop executes.
3. **Test Case 3**:
    - **Input**: `p = [(7, 9), (6, 8), (5, 10)]`
    - **Explanation**: Ensures the flow takes the false branch when checking for new minimum y-coordinates and the leftmost check.

3. Basic Condition Coverage

Basic condition coverage ensures each individual condition in compound boolean expressions is tested.

Test Set for Basic Condition Coverage

1. **Test Case 1**:
    - **Input**: `p = [(6, 2), (5, 3), (8, 4)]`
    - **Explanation**: Tests both conditions in the y-coordinate comparisons within the first loop, covering true/false for `p.get(i).y < p.get(min).y`.
2. **Test Case 2**:
    - **Input**: `p = [(4, 3), (4, 3), (3, 5)]`
    - **Explanation**: Covers testing of x coordinate while y are same.

3. **Test Case 3**:
    - **Input**: `p = [(7, 4), (6, 4), (5, 6)]`
    - **Explanation**: Ensures both conditions in the second loop are covered

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 : Mutations :

1) In the first loop `if (p.get(i).y < p.get(min).y))`:

   Mutation 1: Changed the comparison from < to >.

   Expected result: Changes the behavior of selecting points, potentially favoring those with higher y-coordinates.

2) In the second loop : `if (p.get(i).y == p.get(min).y && p.get(i).x > p.get(min).x))`:

   Mutation 2: Change the condition to use ! = instead of ==.
   if (p.get(i).y != p.get(min).y && p.get(i).x > p.get(min).x) {
   Expected result: It will not select points with the same minimum y-coordinate.

3) **Mutation 3:** Removed the check for the x-coordinate entirely.

   if (p.get(i).y == p.get(min).y) {

   Expected result: This change will select the last occurrence of points with the same y-coordinate, disregarding their x-coordinate position.

   Mutation Outputs : Original Output of code :

   Mutation output of the first one : import java.util.Vector;

```
class Point {
    int x, y;

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

public class ConvexHull {

    public int doGraham(Vector<Point> p) {
        int min = 0;
        for (int i = 1; i < p.size(); i++) {
            if (p.get(i).y > p.get(min).y) { // Mutation: Changed < to >
                min = i;
            }
        }
        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;
            }
        }
```

```
        return min;
    }

    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(0, 2));
        points.add(new Point(3, 3));

        ConvexHull convexHull = new ConvexHull();
        int minIndex = convexHull.doGraham(points);

        System.out.println("Index of minimum point: " + minIndex);
    }
}
```

Mutation 2 : import java.util.Vector;

```
class Point {
    int x, y;

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

public class ConvexHull {

    public int doGraham(Vector<Point> p) {
        int min = 0;
        for (int i = 1; i < p.size(); i++) {
            if (p.get(i).y < p.get(min).y) {
                min = i;
            }
        }
        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) { // Mutation: Changed == to !=
                min = i;
            }
        }
```

```
        return min;
    }

    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(0, 2));
        points.add(new Point(3, 3));

        ConvexHull convexHull = new ConvexHull();
        int minIndex = convexHull.doGraham(points);

        System.out.println("Index of minimum point: " + minIndex);
    }
}
```

Output :

```
java -cp /tmp/xg2yRYYIH8/ConvexHull
Index of minimum point: 3

=== Code Execution Successful ===
```

Mutation 3: import java.util.Vector;

```
class Point {
    int x, y;

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

public class ConvexHull {

    public int doGraham(Vector<Point> p) {
        int min = 0;
        for (int i = 1; i < p.size(); i++) {
            if (p.get(i).y < p.get(min).y) {
                min = i;
            }
        }
        for (int i = 0; i < p.size(); i++) {
            if (p.get(i).y == p.get(min).y) { // Mutation: Removed check for x coordinate
                min = i;
            }
        }
```

```
        return min;
    }

    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(0, 2));
        points.add(new Point(3, 3));

        ConvexHull convexHull = new ConvexHull();
        int minIndex = convexHull.doGraham(points);

        System.out.println("Index of minimum point: " + minIndex);
    }
}
```

Output :

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

one or two times.

Ans : **Path Coverage Test Set**

1. **Test Case 1**: For 0 iterations.
    - **Input:** `p = [(1, 2)]`
    - **Explanation:** This input has only one point, so the first loop will not execute at all. Output should return 0.
2. **Test Case 2**: For one iteration.

    Input: `p = [(2, 3), (2, 3), (2, 3)]`

    - **Explanation:** All points have the same y-coordinate, so the first loop will iterate once (min 0), and the second loop will not execute as there are no further points that satisfy the condition. Output should return the value 0.

      Test Case 3: Has one iteration of the first loop and one iteration of the second loop.

    - **Input:** `p = [(1, 2), (2, 3), (0, 3)]`
    - **Explanation:** The first loop will iterate twice, updating `min` to 0, and will then stay at 0 for `i=2`. The second loop will also iterate once, comparing `x` values and not changing `min`. Expected output is 0 as min will not change.
3. **Test Case 4**: Has two iterations of the first loop and one iteration of the second loop.
    - **Input:** `p = [(1, 4), (2, 2), (0, 4)]`
    - **Explanation:** The first loop will set `min` to 1 when `i=1` since `y` of the second point is smaller.Second loop will iterate again to check for 3rd point, but `min` will not change as its `x` value is less than `x` of the second point. It should return 1.

.