Software Engineering
Lab 9
Lab Session – Mutation Testing

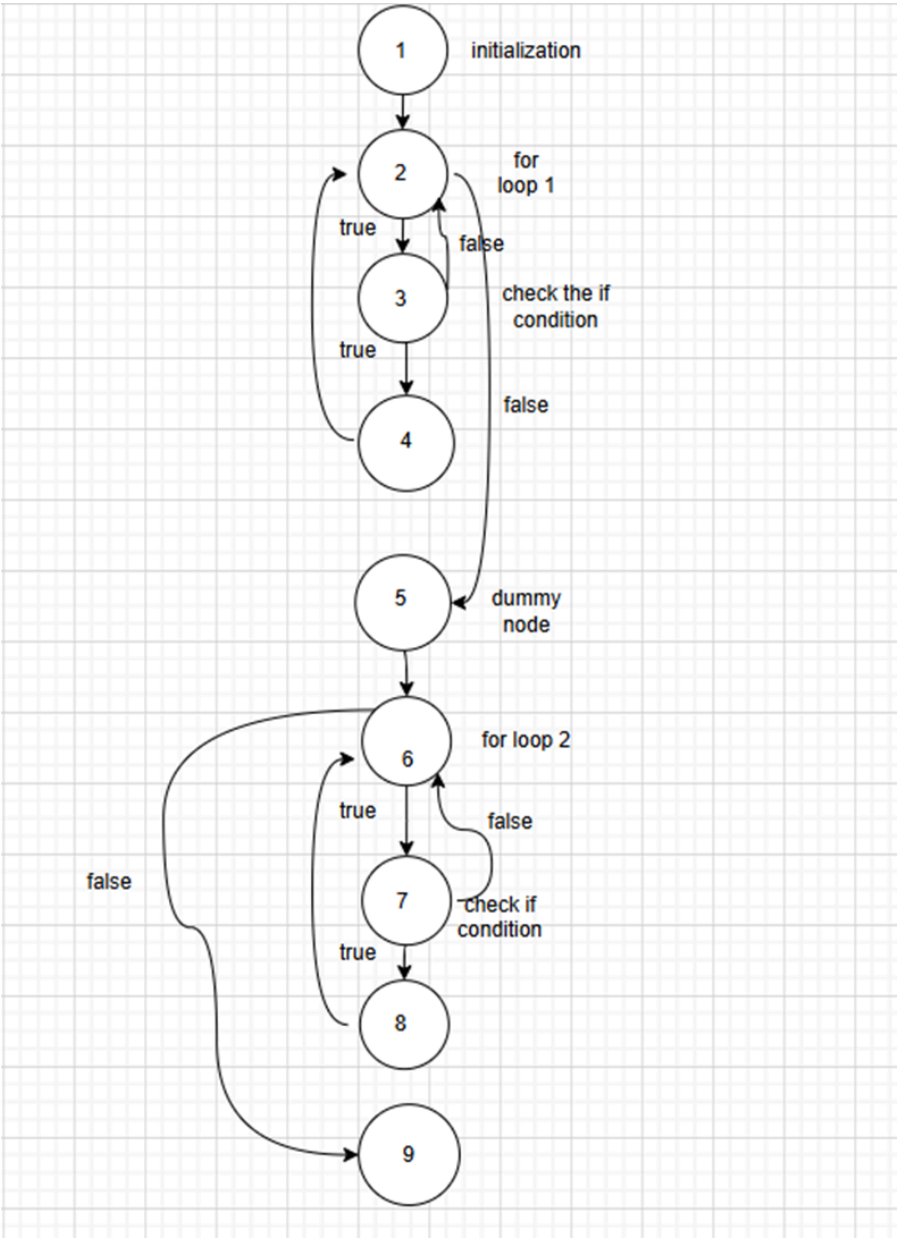
Name: Bansi Patel
ID : 202201190

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.

```
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;
            }
        }
```

For the given code fragment, you should carry out the following activities.
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.
Ans:

Control flow graph :



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.
Ans:

Test Cases for Statement Coverage

We want to ensure every statement in the function is executed at least once.

1. Test Case 1: `p = [{x:1, y:2}, {x:2, y:3}]`
    - Explanation: This test case ensures the first `for` loop is executed since `p` has more than one point.
2. Test Case 2: `p = [{x:1, y:2}, {x:2, y:2}, {x:3, y:2}]`
    - Explanation: This test case ensures the second `for` loop is executed, as there are points with the same `y`-coordinate but different `x`-coordinates.

Test Cases for Branch Coverage

We need to create tests that cover both outcomes of each condition.

1. Test Case 1: `p = [{x:0, y:0}, {x:1, y:-1}, {x:2, y:1}]`
    - Explanation: Ensures `((Point) p.get(i)).y < ((Point) p.get(min)).y` is true.
2. Test Case 2: `p = [{x:0, y:0}, {x:1, y:2}, {x:2, y:3}]`
    - Explanation: Ensures `((Point) p.get(i)).y < ((Point) p.get(min)).y` is false.
3. Test Case 3: `p = [{x:1, y:2}, {x:2, y:2}, {x:3, y:2}]`

- Explanation: Ensures `((Point) p.get(i)).y == ((Point) p.get(min)).y` is true, and `((Point) p.get(i)).x > ((Point) p.get(min)).x` is also true.

4. Test Case 4: `p = [{x:1, y:2}, {x:2, y:3}, {x:3, y:4}]`
    - Explanation: Ensures `((Point) p.get(i)).y == ((Point) p.get(min)).y` is false

<u>Test Cases for Basic Condition Coverage</u>

We must ensure that every condition is tested for both true and false outcomes.

1. Test Case 1: `p = [{x:0, y:0}, {x:-1, y:-1}, {x:1, y:1}]`
    - Condition `((Point) p.get(i)).y < ((Point) p.get(min)).y)` is both true and false in different iterations.
2. Test Case 2: `p = [{x:0, y:0}, {x:1, y:0}, {x:2, y:0}]`
    - Condition `((Point) p.get(i)).y == ((Point) p.get(min)).y)` is true, and `((Point) p.get(i)).x > ((Point) p.get(min)).x)` is also tested as true and false.
3. Test Case 3: `p = [{x:0, y:0}, {x:-1, y:0}, {x:1, y:0}]`
    - Condition `((Point) p.get(i)).x > ((Point) p.get(min)).x)` is both true and false.

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.

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

one or two times.

Ans:

```
import unittest



class Point:

    def _init_(self, x, y):

        self.x = x

        self.y = y



def do_graham(points):

    minindex = 0



    for i in range(1, len(points)):

        if points[i].y < points[minindex].y:

            minindex = i



    for i in range(len(points)):

        if points[i].y == points[minindex].y and points[i].x >
points[minindex].x:

            minindex = i



    return minindex
```

```
class TestGrahamFunction(unittest.TestCase):
```

```
    def test_single_point(self):
        points = [Point(1, 2)]
        result = do_graham(points)
        self.assertEqual(result, 0)

    def test_multiple_points(self):
        points = [Point(2, 3), Point(1, 1), Point(3, 1), Point(0, 2)]
        result = do_graham(points)
        self.assertEqual(result, 2)

    def test_same_y_coordinate(self):
        points = [Point(1, 1), Point(2, 1), Point(3, 1)]
        result = do_graham(points)
        self.assertEqual(result, 2)

    def test_negative_coordinates(self):
        points = [Point(-1, -2), Point(-3, -1), Point(-2, -3)]
        result = do_graham(points)
        self.assertEqual(result, 2)

    def test_mixed_coordinates(self):
        points = [Point(1, 2), Point(-1, -1), Point(0, 0), Point(2, -2)]
        result = do_graham(points)
        self.assertEqual(result, 3)

# Run the tests explicitly
if _name_ == '_main_':
    suite =
unittest.TestLoader().loadTestsFromTestCase(TestGrahamFunction)
    runner = unittest.TextTestRunner()
    runner.run(suite)
```

```
----------------------------------------------------------------------
Ran 5 tests in 0.012s

OK
```

Mutant 1: changes

```
for i in range(len(points)):

        if points[i].y == points[minindex].y and points[i].x <
points[minindex].x:

            minindex = i
```

```
.F.F.
======================================================================
FAIL: test_multiple_points (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-15-ed3669ad05c4>", line 31, in test_multiple_points
    self.assertEqual(result, 2)
AssertionError: 1 != 2

======================================================================
FAIL: test_same_y_coordinate (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-15-ed3669ad05c4>", line 36, in test_same_y_coordinate
    self.assertEqual(result, 2)
AssertionError: 0 != 2

----------------------------------------------------------------------
Ran 5 tests in 0.011s

FAILED (failures=2)
```

Mutant 2: changes

```
for i in range(1, len(points)):

        if points[i].y > points[minindex].y:

            minindex = i
```

```
FFF..
======================================================================
FAIL: test_mixed_coordinates (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-16-785722b6b6bd>", line 46, in test_mixed_coordinates
    self.assertEqual(result, 3)
AssertionError: 0 != 3

======================================================================
FAIL: test_multiple_points (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-16-785722b6b6bd>", line 31, in test_multiple_points
    self.assertEqual(result, 2)
AssertionError: 0 != 2

======================================================================
FAIL: test_negative_coordinates (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-16-785722b6b6bd>", line 41, in test_negative_coordinates
    self.assertEqual(result, 2)
AssertionError: 1 != 2

----------------------------------------------------------------------
Ran 5 tests in 0.010s

FAILED (failures=3)
```

Mutant 3: changes

```
for i in range(len(points)):

        if points[i].y != points[minindex].y and points[i].x >
points[minindex].x:

            minindex = i
```

```
..FF.
======================================================================
FAIL: test_negative_coordinates (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-17-e01ec9d50bc3>", line 41, in test_negative_coordinates
    self.assertEqual(result, 2)
AssertionError: 0 != 2

======================================================================
FAIL: test_same_y_coordinate (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-17-e01ec9d50bc3>", line 36, in test_same_y_coordinate
    self.assertEqual(result, 2)
AssertionError: 0 != 2

----------------------------------------------------------------------
Ran 5 tests in 0.013s

FAILED (failures=2)
```

Mutant 4: changes

```
minindex=-1

    return minindex
```

```
FFFFF
======================================================================
FAIL: test_mixed_coordinates (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-18-f6f89158d475>", line 47, in test_mixed_coordinates
    self.assertEqual(result, 3)
AssertionError: -1 != 3

======================================================================
FAIL: test_multiple_points (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-18-f6f89158d475>", line 32, in test_multiple_points
    self.assertEqual(result, 2)
AssertionError: -1 != 2

======================================================================
FAIL: test_negative_coordinates (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-18-f6f89158d475>", line 42, in test_negative_coordinates
    self.assertEqual(result, 2)
AssertionError: -1 != 2

======================================================================
FAIL: test_same_y_coordinate (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-18-f6f89158d475>", line 37, in test_same_y_coordinate
    self.assertEqual(result, 2)
AssertionError: -1 != 2

======================================================================
FAIL: test_single_point (__main__.TestGrahamFunction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-18-f6f89158d475>", line 27, in test_single_point
    self.assertEqual(result, 0)
AssertionError: -1 != 0

----------------------------------------------------------------------
Ran 5 tests in 0.018s

FAILED (failures=5)
```