IT314 Software Engineering
Lab 9

- Maitrey Pandya - 202201335

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.

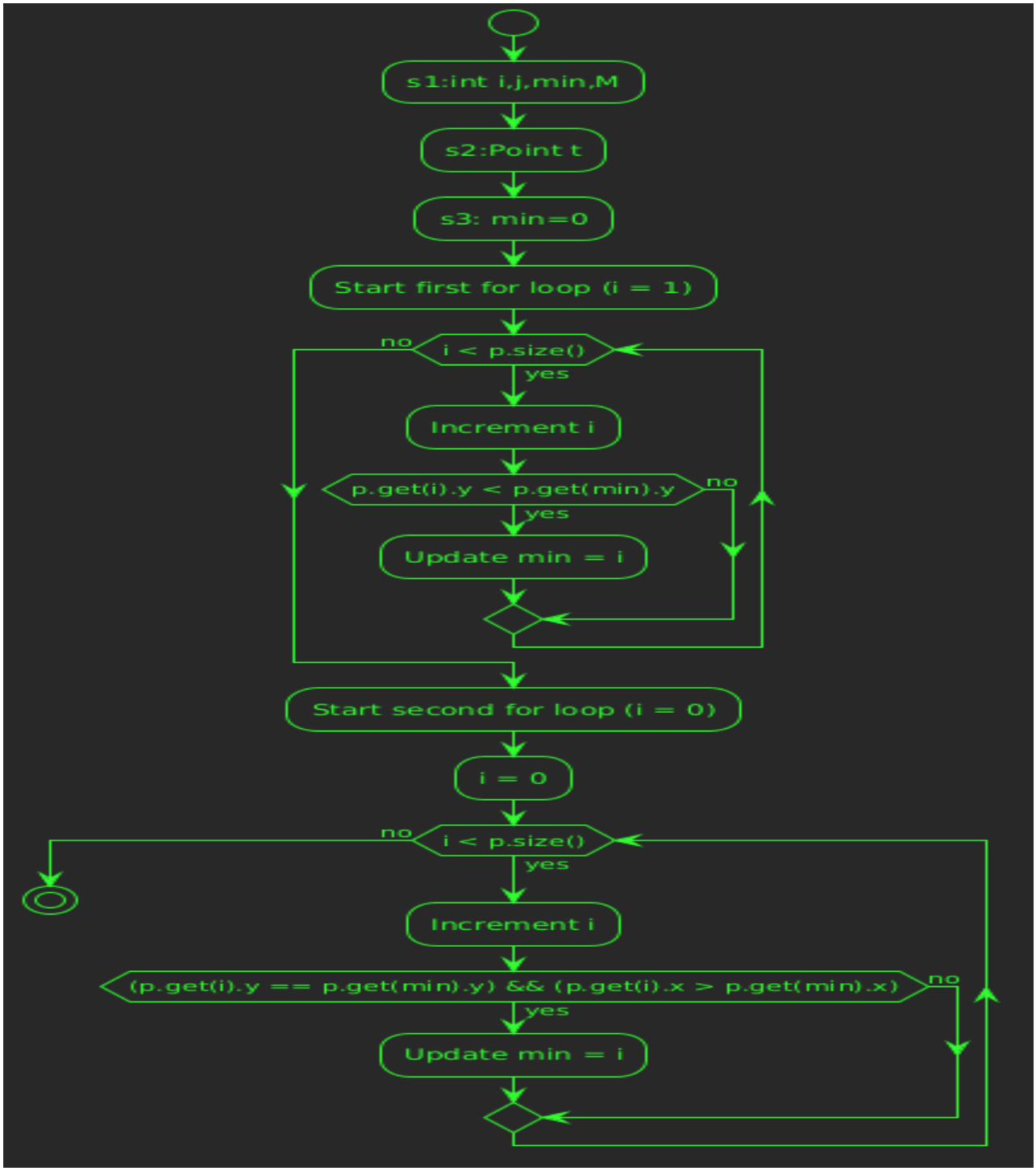
Given Code:

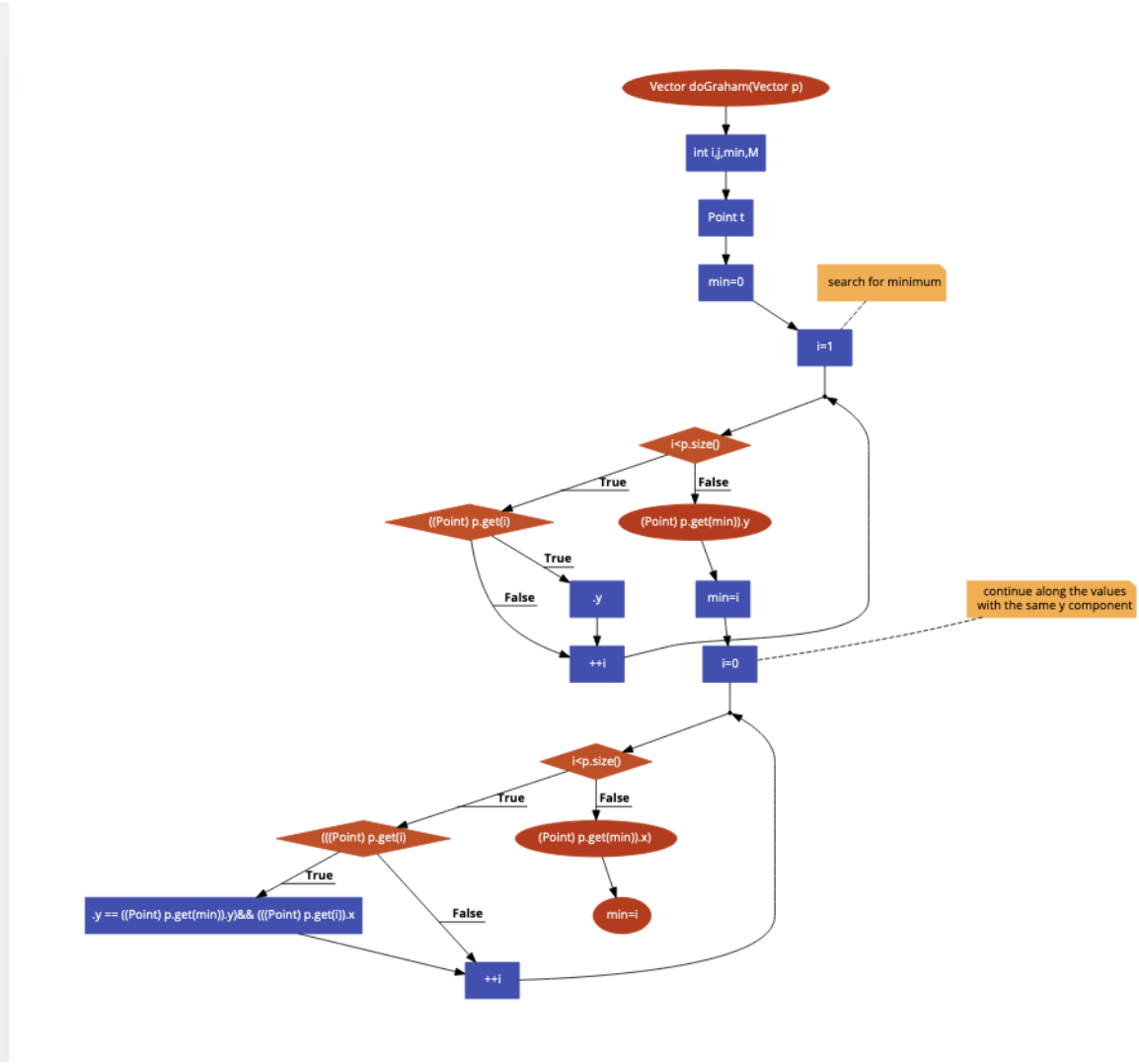
```
Vector doGraham(Vector p){
        int i,j,min,M;
        Point t;
        min=0;
        //search for minimum
        for(i=1;i<p.size();++i){
            if ( ((Point) p.get(i)).y < ((Point) p.get(min)).y){
                min=i;
            }
        }
        //continue along the values with the 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.

A1. Control Flow Graph :


s1:int i,j,min,M
s2:Point t
s3: min=0
Start first for loop (i = 1)
no
i < p.size()
yes
Increment i
p.get(i).y < p.get(min).y
no
yes
Update min = i
Start second for loop (i = 0)
i = 0
no
i < p.size()
yes
Increment i
(p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x)
no
yes
Update min = i



Control Flow Graph generated by Code2Flow (online platform)

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).
Yes

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.

Here, I have written a code in python implementing the same functionality as above java code as testing and mutation is easier in python.

Python Code:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Vector:
    def __init__(self):
        self.points = []

    def add(self, point):
        self.points.append(point)

    def size(self):
        return len(self.points)

    def get(self, index):
        return self.points[index]

def doGraham(p):
    min_index = 0
    # Search for minimum based on the y coordinate
    for i in range(1, p.size()):
        if p.get(i).y < p.get(min_index).y:
            min_index = i

    # Continue along the values with the same y component and check for maximum x
    for i in range(p.size()):
        if p.get(i).y == p.get(min_index).y and p.get(i).x > p.get(min_index).x:
            min_index = i

    return min_index

# Example usage
vector = Vector()
vector.add(Point(1, 5))
vector.add(Point(2, 3))
vector.add(Point(3, 3))
vector.add(Point(4, 4))

min_index = doGraham(vector)
print("Index of the point with minimum y (and max x if y is equal):", min_index)
```

Test Case	Points	Expected	Explanation
Test Case 1	[]	None	Empty list
Test Case 2	[(1, 2), (2, 3), (3, 4)]	(1, 2)	No tie in y
Test Case 3	[(1, 3), (2, 3), (3, 3)]	(3, 3)	Tie in y

Test Case 4	[(2, 2), (1, 1), (2, 1)]	(2, 1)	Tie in points

Python Code For Testing:

```python
import pytest
from SELab9 import doGraham, Point

@pytest.mark.parametrize("points, expected", [
    # Test case 1: Empty list
    ([], None),  # This covers the case where len(points) == 0

    # Test case 2: No tie in y, the point with the smallest y should be selected
    ([Point(1, 2), Point(2, 3), Point(3, 4)], (1, 2)),  # This ensures the first loop works for no ties

    # Test case 3: Same y values, select the point with the largest x if tied
    ([Point(1, 3), Point(2, 3), Point(3, 3)], (3, 3)),  # This ensures the second loop works for tie-breaking

    # Test case 4: Two points have tie
    ([Point(2, 2), Point(1, 1),Point(2,1)], (2, 1)),  #tie in 2nd and 3rd value
])

def test_doGraham(points, expected):
    result = doGraham(points)
    if expected is None:
        assert result is None, f"Failed for points: {[(p.x, p.y) for p in points]}"
    else:
        assert (result.x, result.y) == expected, f"Failed for points: {[(p.x, p.y) for p in points]}"
```

Running the test cases using pytest library

```
============================= test session starts ==============================
platform darwin -- Python 3.11.0, pytest-8.3.3, pluggy-1.5.0
rootdir: /Users/maitrey/Downloads/Lab6_Group37/202201335_Lab6
plugins: cov-6.0.0
collected 5 items

SELab9_unittest.py .....                                                 [100%]

============================== 5 passed in 0.01s ===============================
```

Testing for coverage using pytest-cov,

```
============================= test session starts ==============================
platform darwin -- Python 3.11.0, pytest-8.3.3, pluggy-1.5.0
rootdir: /Users/maitrey/Downloads/Lab6_Group37/202201335_Lab6
plugins: cov-6.0.0
collected 4 items

SELab9_unittest.py ....                                                  [100%]

---------- coverage: platform darwin, python 3.11.0-final-0 ----------
Name        Stmts   Miss  Cover   Missing
-----------------------------------------
SELab9.py      17      0   100%
-----------------------------------------
TOTAL          17      0   100%


============================== 4 passed in 0.02s ===============================
```

Now for analysis from CFG,

Coverage Type	Test Case	Points	Expected	Explanation

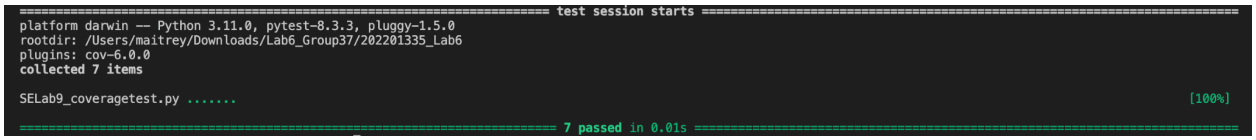
Branch Coverage	Test Case 1	[(1, 2), (2, 3), (3, 4)]	(1, 2)	No tie
Branch Coverage	Test Case 2	[(1, 3), (2, 3), (3, 3)]	(3, 3)	Tie, larger x
Statement Coverage	Test Case 1	[]	None	Empty list
Statement Coverage	Test Case 2	[(1, 2), (2, 3), (3, 4)]	(1, 2)	No tie
Statement Coverage	Test Case 3	[(1, 3), (2, 3), (3, 3)]	(3, 3)	Tie, larger x
Basic Condition Coverage	Test Case 1	[(2, 2), (1, 1), (2, 1)]	(2, 1)	Tie, x comparison
Basic Condition Coverage	Test Case 2	[(5, 3), (2, 1), (1, 5)]	(2, 1)	Smallest y

Python Code For Testing:

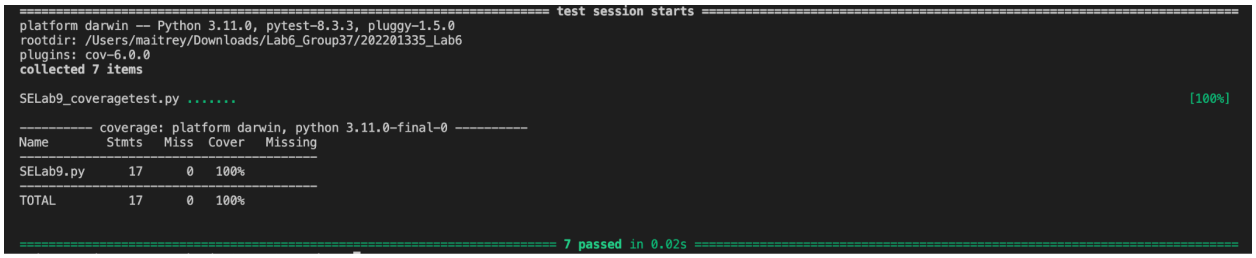
```python
import pytest
from SELab9 import doGraham, Point

# Branch Coverage Test Cases
@pytest.mark.parametrize("points, expected", [
    ([Point(1, 2), Point(2, 3), Point(3, 4)], (1, 2)),  # No tie
    ([Point(1, 3), Point(2, 3), Point(3, 3)], (3, 3)),  # Tie, larger x
])
def test_branch_coverage(points, expected):
    result = doGraham(points)
    if expected is None:
        assert result is None
    else:
        assert (result.x, result.y) == expected

# Statement Coverage Test Cases
@pytest.mark.parametrize("points, expected", [
    ([], None),  # Empty list
    ([Point(1, 2), Point(2, 3), Point(3, 4)], (1, 2)),  # No tie
    ([Point(1, 3), Point(2, 3), Point(3, 3)], (3, 3)),  # Tie, larger x
])
def test_statement_coverage(points, expected):
    result = doGraham(points)
    if expected is None:
        assert result is None
    else:
        assert (result.x, result.y) == expected

# Basic Condition Coverage Test Cases
@pytest.mark.parametrize("points, expected", [
    ([Point(2, 2), Point(1, 1), Point(2, 1)], (2, 1)),  # Tie, x comparison
    ([Point(5, 3), Point(2, 1), Point(1, 5)], (2, 1)),  # Smallest y
])
```

```
def test_basic_condition_coverage(points, expected):
    result = doGraham(points)
    if expected is None:
        assert result is None
    else:
        assert (result.x, result.y) == expected
```

On running testcases in pytest,



On checking code coverage,



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 #	Points List	Expected Result	Loop Iteration Explanation
1	[]	None	No loops are entered (empty list).
2	[(1, 2)]	(1, 2)	The first loop runs once and exits since there's only one point.
3	[(1, 2), (2, 3)]	(1, 2)	The first loop runs twice, the second loop runs once (no tie).
4	[(1, 2), (2, 3), (3, 3)]	(3, 3)	First loop runs 3 times, second loop runs once (tie condition).
5	[(1, 2), (2, 3), (3, 2), (2, 1)]	(2, 1)	First loop runs 4 times, second loop runs twice (x comparison).

Python Code For Testing :

```
import pytest
from SELab9 import doGraham, Point

@pytest.mark.parametrize("points, expected", [
    # Test Case 1: Empty list (no loops)
    ([], None),
```

```python
    # Test Case 2: One point in the list (loop runs once)
    ([Point(1, 2)], (1, 2)),

    # Test Case 3: Two points, no tie in y (loop runs once)
    ([Point(1, 2), Point(2, 3)], (1, 2)),

    # Test Case 4: Three points, tie in y, larger x wins (loop runs twice)
    ([Point(1, 3), Point(2, 3), Point(3, 3)], (3, 3)),

    # Test Case 5: Four points, tie in y and comparison in x values (loop runs twice)
    ([Point(1, 2), Point(2, 3), Point(3, 2), Point(2, 1)], (2, 1)),
])
def test_path_coverage(points, expected):
    result = doGraham(points)
    if expected is None:
        assert result is None
    else:
        assert (result.x, result.y) == expected
```

Output For Test Coverage:

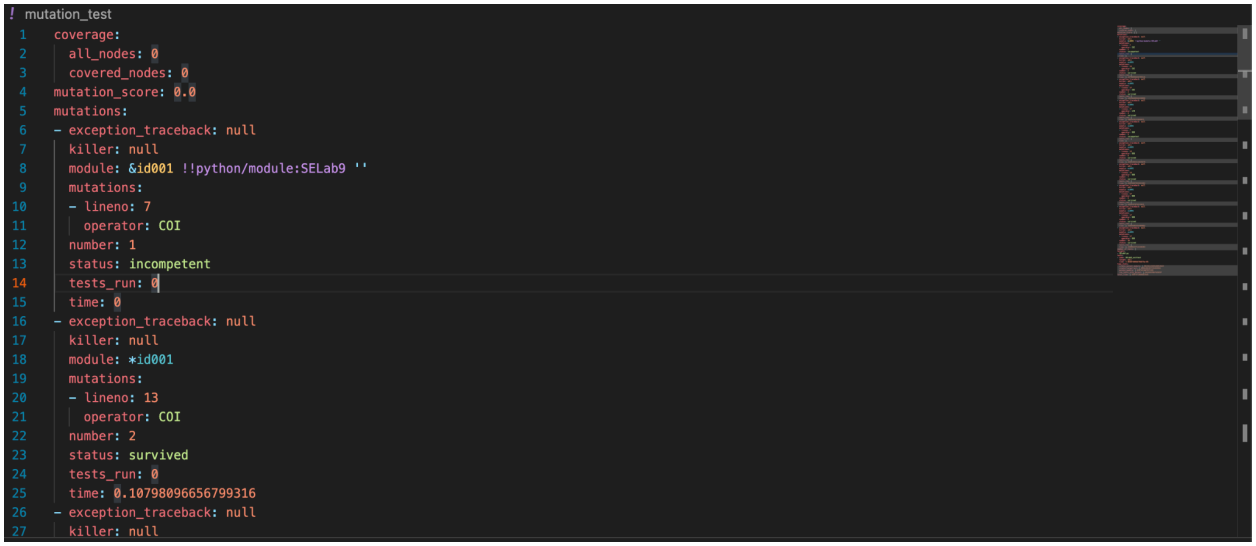
```
============================= test session starts ==============================
platform darwin -- Python 3.11.0, pytest-8.3.3, pluggy-1.5.0
rootdir: /Users/maitrey/Downloads/Lab6_Group37/202201335_Lab6
plugins: cov-6.0.0
collected 5 items

SELab9_LoopCover.py .....                                                [100%]

---------- coverage: platform darwin, python 3.11.0-final-0 ----------
Name          Stmts   Miss  Cover   Missing
-------------------------------------------
SELab9.py        17      0   100%
-------------------------------------------
TOTAL            17      0   100%


============================== 5 passed in 0.02s ===============================
```

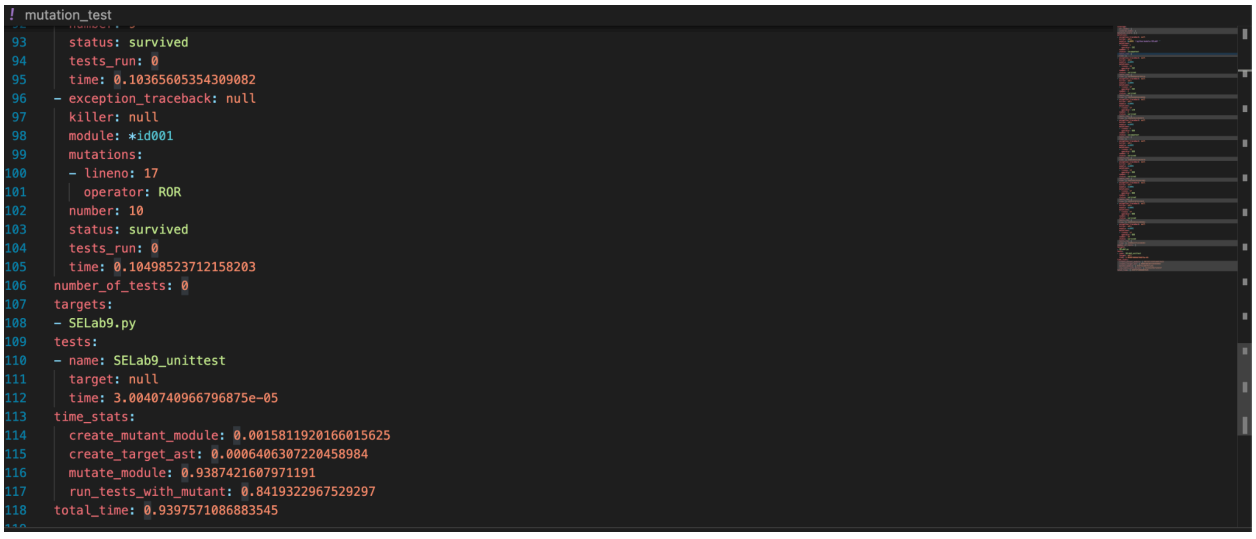
3. For the test set you have just checked can you find a mutation of the code (i.e. the deletion, change or insertion of some code) that will result in failure but is not detected by your test set. You have to use the mutation testing tool.

Result Of Mutation Testing

```
  16:     for i in range(0, len(points)):
- 17:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 17:         if (points[i].y == points[min_index].y and points[i].x >= points[min_index].x):
  18:             min_index = i
  19:
  20:     return points[min_index]
  21:
--------------------------------------------------------------------------------
3 3
[0.10499 s] survived
[*] Mutation score [0.93976 s]: 0.0%
   - all: 10
   - killed: 0 (0.0%)
   - survived: 8 (80.0%)
   - incompetent: 2 (20.0%)
   - timeout: 0 (0.0%)
maitrey@Maitreys-MacBook-Air 202201335_Lab6 %
```

Mutation Report:





8 out of 10 mutations survived while 2 were declared incompetent