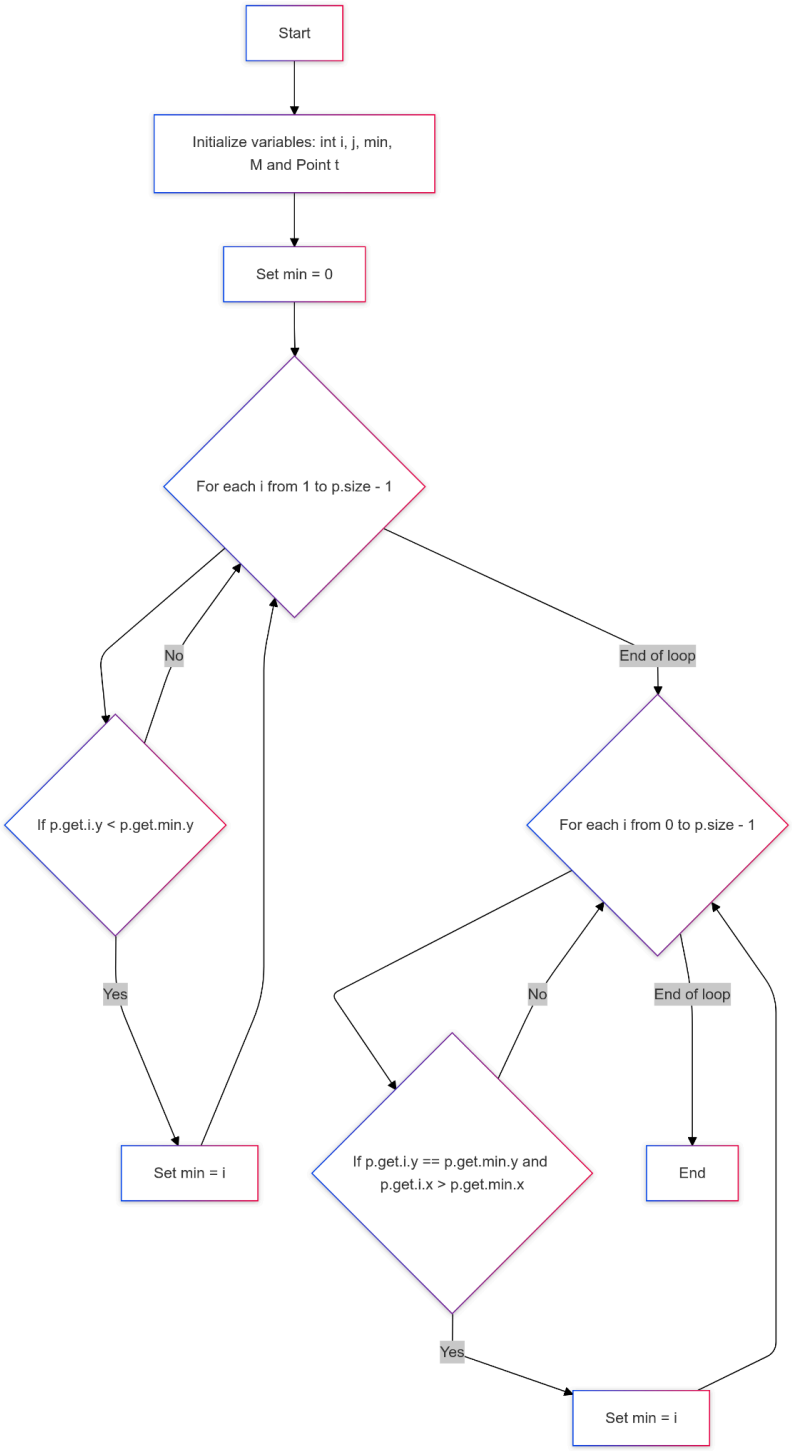
Q1)

```
class Point:
    def init(self, x, y):
        self.x = x
        self.y = y
class ConvexHull:
    def do_graham(self, points):
        min_index = 0
        # Search for minimum y-coordinate (and lowest x-coordinate if y's are the same)
        for i in range(1, len(points)):
            if points[i].y < points[min_index].y:
                min_index = i
        # Continue along the values with the same y component
        for i in range(len(points)):
            if points[i].y == points[min_index].y and points[i].x > points[min_index].x:
                min_index = i
        return min_index  # Returning min index for verification purposes
# Example usage
if name == "main":
    points = [Point(0, 0), Point(1, 1), Point(2, 2), Point(1, 0)]
    convex_hull = ConvexHull()
    min_index = convex_hull.do_graham(points)
    print(f"The index of the minimum point is: {min_index}")
    print(f"The minimum point is: ({points[min_index].x}, {points[min_index].y})")
```

Q2)


Start
Initialize variables: int i, j, min, M and Point t
Set min = 0
For each i from 1 to p.size - 1
No
End of loop
If p.get.i.y < p.get.min.y
For each i from 0 to p.size - 1
Yes
No
End of loop
Set min = i
If p.get.i.y == p.get.min.y and p.get.i.x > p.get.min.x
End
Yes
Set min = i

Q3)

Statement Coverage

Objective: Ensure each line of code is executed at least once.

To achieve statement coverage:

1. We need to run the code through both `for` loops and satisfy all `if` conditions at least once.

Test Case for Statement Coverage

Test Case 1:

- **Input:** `p = [Point(2, 3), Point(4, 1), Point(5, 2)]`
- **Expected Output:** `min = 1`
  This test case will:
  - Execute the first `for` loop and the `if` condition to find the smallest `y`.
  - The second loop will also run, but no tie will occur.

Test Case 2 (for Tie Case):

- **Input:** `p = [Point(2, 3), Point(4, 1), Point(3, 1), Point(5, 2)]`
- **Expected Output:** `min = 2`
  This test case will:
  - Execute both loops and trigger the `if` condition in the second loop to handle a tie on `y` by choosing the point with the larger `x`.

These two test cases cover each line of code, fulfilling **Statement Coverage**.

b. Branch Coverage

Objective: Ensure each branch (true/false for each condition) is covered.

To achieve branch coverage, we need to make sure each possible outcome (true/false) of each conditional expression is tested.

Test Case for Branch Coverage

We can use the same test cases as above, with some additions to ensure all branches are covered.

Test Case 1:

- **Input:** `p = [Point(2, 3), Point(4, 1), Point(5, 2)]`
- **Expected Output:** `min = 1`

This case will:

- Cover the true and false branches of the first loop's `if` statement.

Test Case 2 (for Tie Case):

- **Input:** `p = [Point(2, 3), Point(4, 1), Point(3, 1), Point(5, 2)]`
- **Expected Output:** `min = 2`

This case will:

- Cover both true and false branches in the second loop's `if` statement to resolve the tie by `x`.

Additional Test Case 3 (No Change in `min`):

- **Input:** `p = [Point(2, 3), Point(3, 3), Point(4, 3)]`
- **Expected Output:** `min = 0`

This case will:

- Ensure that the `if` conditions do not trigger any changes in `min`.

These test cases fulfill **Branch Coverage**.

c. Basic Condition Coverage

Objective: Ensure each basic condition within the expressions is evaluated to both true and false.

Each `if` statement has two basic conditions:

1. `(p.get(i).y < p.get(min).y)` in the first loop.

2. `(p.get(i).y == p.get(min).y)` and `(p.get(i).x > p.get(min).x)` in the second loop.

Test Cases for Basic Condition Coverage

Test Case 1 (Condition where y is less than minimum):

- **Input:** `p = [Point(2, 3), Point(4, 1), Point(5, 2)]`
- **Expected Output:** `min = 1`

This case will:

- Test `p.get(i).y < p.get(min).y` to be true.

Test Case 2 (Condition where y is equal and x is greater):

- **Input:** `p = [Point(2, 3), Point(4, 1), Point(3, 1), Point(5, 2)]`
- **Expected Output:** `min = 2`

This case will:

- Test both `p.get(i).y == p.get(min).y` and `p.get(i).x > p.get(min).x` to be true.

Test Case 3 (Condition where both conditions are false):

- **Input:** `p = [Point(2, 3), Point(5, 3)]`
- **Expected Output:** `min = 0`

This case will:

- Test both conditions in the second `if` to be false.

These three test cases provide **Basic Condition Coverage**, ensuring that each individual condition in the expressions has been tested with true and false values.

```
[*] Start mutation process:
  - targets: point
  - tests: test_points
[*] 3 tests passed:
  - test_points [0.24341 s]
[*] Start mutants generation and execution:
  - [#   1] COI point:
--------------------------------------------------------------------------------
   6:
   7: def find_min_point(points):
   8:     min_index = 0
   9:     for i in range(1, len(points)):
- 10:         if points[i].y < points[min_index].y:
+ 10:         if not (points[i].y < points[min_index].y):
  11:             min_index = i
  12:     for i in range(len(points)):
  13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
  14:             min_index = i
--------------------------------------------------------------------------------
[0.15408 s] killed by test_points.py::TestFindMinPointPathCoverage::testMultiplePoints
  - [#   2] COI point:
--------------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
```

```
 12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if not ((points[i].y == points[min_index].y and points[i].x > points[min_index].x)):
 14:             min_index = i
 15:     return points[min_index]
```

[0.14159 s] killed by test_points.py::TestFindMinPointPathCoverage::testMultiplePointSamyY

- [# 3] LCR point:

```
  9:     for i in range(1, len(points)):
 10:         if points[i].y < points[min_index].y:
 11:             min_index = i
 12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y or points[i].x > points[min_index].x):
 14:             min_index = i
 15:     return points[min_index]
```

[0.15599 s] killed by test_points.py::TestFindMinPointPathCoverage::testMultiplePointSamyY

- [# 4] ROR point:

```
  6:
  7: def find_min_point(points):
  8:     min_index = 0
  9:     for i in range(1, len(points)):
```

```
- 10:         if points[i].y < points[min_index].y:
+ 10:         if points[i].y > points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
  13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
  14:             min_index = i
```

[0.14234 s] killed by test_points.py::TestFindMinPointPathCoverage::testMultiplePoints

- [# 5] ROR point:

```
  6:
  7: def find_min_point(points):
  8:     min_index = 0
  9:     for i in range(1, len(points)):
- 10:         if points[i].y < points[min_index].y:
+ 10:         if points[i].y <= points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
  13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
  14:             min_index = i
```

[0.11556 s] survived

- [# 6] ROR point:

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

```
 10:         if points[i].y < points[min_index].y:
 11:             min_index = i
 12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y != points[min_index].y and points[i].x > points[min_index].x):
 14:             min_index = i
 15:     return points[min_index]
```

[0.14255 s] killed by test_points.py::TestFindMinPointPathCoverage::testMultiplePointSamyY

- [# 7] ROR point:

```
  9:     for i in range(1, len(points)):
 10:         if points[i].y < points[min_index].y:
 11:             min_index = i
 12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y and points[i].x < points[min_index].x):
 14:             min_index = i
 15:     return points[min_index]
```

[0.14933 s] killed by test_points.py::TestFindMinPointPathCoverage::testMultiplePointSamyY

- [# 8] ROR point:

```
  9:     for i in range(1, len(points)):
 10:         if points[i].y < points[min_index].y:
```

```
 11:           min_index = i
 12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y and points[i].x >= points[min_index].x):
 14:           min_index = i
 15:     return points[min_index]
```

[0.11332 s] survived

Q4)

```
import unittest
from point import Point, findMinPoint

class TestFindMinPointPathCoverage(unittest.TestCase):

    def TestEmptyList(self):
        points = []
        with self.assertRaises(IndexError):
findMinPoint(points)

    def TestSinglePoint(self):
        points = [Point(2, 2)]
        result = findMinPoint(points)
        self.assertEqual(result, points[0])
    def testTwoUniquePoint(self):
        points = [Point(2, 1), Point(3, 2)]
        result = findMinPoint(points)
        self.assertEqual(result, points[0])
    def TestMultipleuniquePoint(self):
        points = [Point(1, 3), Point(2, 4), Point(3, 5)]
        result = findMinPoint(points)
        self.assertEqual(result, points[0])

    def testMultiplePointSamyY(self):
```

```
        points = [Point(1, 2), Point(3, 2), Point(2, 2)]
        result = findMinPoint(points)
        self.assertEqual(result, points[1])

    def testMultiplePoints(self):
        points = [Point(1, 2), Point(2, 2), Point(3, 1), Point(4, 1)]
        result = findMinPoint(points)
        self.assertEqual(result, points[3])

# Run the tests if this file is executed
if __name__ == "__main__":
    unittest.main()
```

```
Test Result with mut.py
Mutation score [1.52260 s]: 75.0%
   - all: 8
   - killed: 6 (75.0%)
   - survived: 2 (25.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
```