
Lab-7
IT314 Software Engineering

Date: 13/04/2023

Name: Dethaliya Yash Himatbhai

Student ID: 202001193

Course: IT314 Software Engineering

- Previous date Problem:

- Test cases:

Test Case number	Day	Month	Year	Expected Output
1	15	6	1900	14-6-1900
2	15	6	1901	14-6-1901
3	15	6	1962	14-6-1962
4	15	6	2014	14-6-2014
5	15	6	2015	14-6-2014
6	1	6	1962	31-5-1962
7	2	6	1962	1-6-1962
8	30	6	1962	29-6-1962
9	31	6	1962	INVALID
10	15	1	1962	14-1-1962
11	15	2	1962	14-2-1962
12	15	11	1962	14-11-1962
13	15	12	1962	14-12-1962

- Equivalence Class Partitions:
- Day:

Class Partition ID	Day Range	Expected Output
1	Day between 1 to 28	VALID
2	Less than 1	INVALID
3	Greater than 31	INVALID
4	30	VALID
5	29	VALID FOR LEAP YEAR

6	31	VALID
---	----	-------

- Month:

Class Partition ID	Range	Expected Output
1	1 to 12	VALID
2	Less than 1	INVALID
3	Greater than 12	INVALID

- Year:

Class Partition ID	Range	Expected Output
1	1900 to 2015	VALID
2	Less than 2019	INVALID
3	Less than 2015	INVALID

P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.

```
int linearSearch(int v, int a[])
{
    int i = 0;
    while (i < a.length)
    {
        if (a[i] == v)
            return(i);
        i++;
    }
    return (-1);
}
```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a=[1,1,7,8,9] ,v=7	2
a=[1,1,7,8,9] ,v=10	-1
a=[], v=5	-1
Boundary Value Analysis	
a[], v=6	-1
a[2] ,v=7	-1
a[2] ,v=2	0
a=[1,3,4,5,6] , v=1	0
a=[1,3,4,5,6] , v=4	2
a=[1,3,4,5,6] , v=6	4
a=[1,3,4,5,6] , v=7	-1

The screenshot shows an IDE interface with the following components:

- Package Explorer:** Shows the project structure with a package named "test".
- JUnit Runner:** Displays the test results: "Runs: 3/3", "Errors: 0", "Failures: 0".
- Code Editor:** Contains two files: `UnitTesting.java` and `LinearSearch.java`. `UnitTesting.java` contains a static import for `org.junit.Assert.*` and three test methods: `test1()`, `test2()`, and `test3()`. `LinearSearch.java` contains a class `LinearSearch` with a method `linearSearch` that takes an integer array and a target value, returning the index of the target or -1 if not found.
- Outline View:** Shows the class hierarchy and methods: `test`, `LinearSearch`, and its methods `test1()`, `test2()`, and `test3()`.
- Coverage Analysis:** A table at the bottom right shows coverage details for the `LinearSearch` class. It has 100.0% coverage with 114 covered instructions, 0 missed instructions, and a total of 114 instructions.

```

package test;

import static org.junit.Assert.*;

public class LinearSearch {
    @Test
    public void test1() {
        UnitTesting obj = new UnitTesting();
        int arr[] = {};
        int output = obj.linearSearch(5, arr);
        assertEquals(output, -1);
    }

    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();
        int arr[] = {1,1,7,8,9};
        int output = obj.linearSearch(7, arr);
        assertEquals(output, 2);
    }

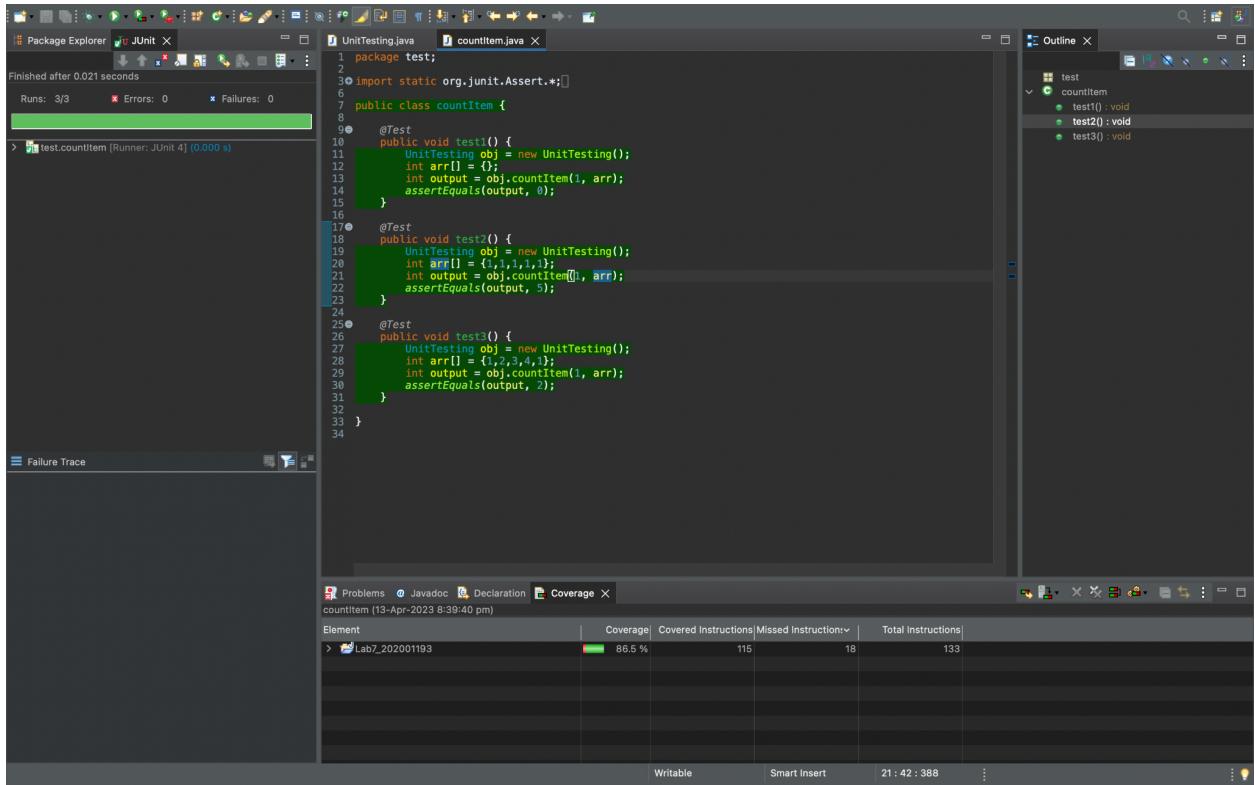
    @Test
    public void test3() {
        UnitTesting obj = new UnitTesting();
        int arr[] = {1,1,7,8,9};
        int output = obj.linearSearch(6, arr);
        assertEquals(output, 0);
    }
}

```

P2. The function countItem returns the number of times a value v appears in an array of integers a.

```
int countItem(int v, int a[])
{
    int count = 0;
    for (int i = 0; i < a.length; i++)
    {
        if (a[i] == v)
            count++;
    }
    return (count);
}
```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a=[], v=1	0
a=[1,2,1,3,4], v=5	0
a=[1,2,1,3,4] ,v=1	2
Boundary Value Analysis	
a[], v=1	0
a[3], v=2	0
a[4], v=4	1
a=[1,2,3,4,1] ,v=2	1
a=[1,2,3,4,1] ,v=5	0
a=[1,2,3,4,1] ,v=1	2
a=[1,1,1,1,1] ,v=1	5



P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in the array a, then the function returns an index i, such that $a[i] == v$; otherwise, -1 is returned.

```
int binarySearch(int v, int a[])
{
    int lo,mid,hi;
    lo = 0;
    hi = a.length-1;
    while (lo <= hi)
    {
        mid = (lo+hi)/2;
        if (v == a[mid])
            return (mid);
        else if (v < a[mid])
            hi = mid-1;
        else
            lo = mid+1;
    }
    return(-1);
}
```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a=[], v=5	-1
a=[1,2,3,5] ,v=4	-1
a=[1,2,3,5,6,7] ,v=5	4
Boundary Value Analysis	
a[], v=5	-1
a[1], v=1	0
a[2],v=3	-1
a[1,2,3,5,6] ,v=6	4
a[1,2,3,5,6] ,v=1	0
a[1,2,3,5,6] ,v=3	2
a[1,2,3,5,6] ,v=0	-1
a[1,2,3,5,6] ,v=8	-1

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows files like UnitTesting.java, countitem.java, LinearSearch.java, and binarySearch.java.
- JUnit View:** Displays test results: Runs: 3/3, Errors: 0, Failures: 0. The test results for binarySearch are shown:
 - test1 (0.000 s)
 - test2 (0.000 s)
 - test3 (0.000 s)
- Code Editor:** Displays the Java code for binarySearch:


```

1 package test;
2
3 import static org.junit.Assert.*;
4
5 public class binarySearch {
6
7     @Test
8     public void test1() {
9         UnitTesting obj = new UnitTesting();
10        int arr[] = {3};
11        int output = obj.binarySearch(3, arr);
12        assertEquals(output, -1);
13    }
14
15    @Test
16    public void test2() {
17        UnitTesting obj = new UnitTesting();
18        int arr[] = {1,2,3,5,6};
19        int output = obj.binarySearch(3, arr);
20        assertEquals(output, 4);
21    }
22
23    @Test
24    public void test3() {
25        UnitTesting obj = new UnitTesting();
26        int arr[] = {1,2,3,5,6};
27        int output = obj.binarySearch(3, arr);
28        assertEquals(output, 2);
29    }
30
31}
32
33}
34
35
      
```
- Outline View:** Shows the class structure with methods test1(), test2(), and test3().
- Coverage View:** Shows code coverage statistics for binarySearch.java:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
> Lab7_20200193	76.3 %	132	41	173

P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function triangle takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

```

final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
    if (a >= b+c || b >= a+c || c >= a+b)
        return (INVALID);
    if (a == b && b == c)
        return (EQUILATERAL);
    if (a == b || a == c || b == c)
        return (ISOSCELES);
    return (SCALENE);

}

```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a=b<c where a>,b>0,c>0	ISOSCELES
a=b=c where a>0,b>0,c>0	EQUILATERAL
a<b+c, b<a+c, c<a+b where a>0,b>0,c>0	SCALENE
a=0 ,b=0,c=0	INVALID
a>=b+c, b>=a+c, c>=a+b where a>0,b>0,c>0	INVALID
a=0, b=c where b>0 ,c>0	INVALID
Boundary Value Analysis	
a=3,b=3,c=3	EQUILATERAL

a=3,b=3,c=7	INVALID
a=3,b=3,c=5	ISOSCELES
a=2147483647,b=2147483647,c=2147483647	EQUILATERAL
a=2147483647,b=2147483647,c=2147483645	ISOSCELES
a=1,b=1,c=0	INVALID
a=1,b=1,c=2147483647	INVALID

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with files like UnitTesting.java, countitem.java, LinearSearch.java, binarySearch.java, and triangle.java.
- JUnit View:** Displays the results of a JUnit 4 run. It shows 3 runs, 0 errors, and 0 failures. The test class is test.triangle, and it contains three test methods: test1, test2, and test3.
- Outline View:** Shows the class hierarchy: test > triangle > test1(), test2(), and test3().
- Code Editor:** Displays the Java code for the triangle class and its test methods. The code uses JUnit annotations (@Test) and assertEqual methods to check triangle classifications based on side lengths a, b, and c.
- Coverage View:** Shows the code coverage report for triangle.java. The coverage is 64.2%, with 111 covered instructions, 62 missed instructions, and a total of 173 instructions.

```

1 package test;
2
3 import static org.junit.Assert.*;
4
5 public class triangle {
6     @Test
7     public void test1() {
8         UnitTesting obj = new UnitTesting();
9         int a=3;
10        int b=3;
11        int c=0;
12        int output = obj.triangle(a,b,c);
13        assertEquals(output, 3);
14    }
15    @Test
16    public void test2() {
17        UnitTesting obj = new UnitTesting();
18        int a=3;
19        int b=3;
20        int c=3;
21        int output = obj.triangle(a,b,c);
22        assertEquals(output, 1);
23    }
24    @Test
25    public void test3() {
26        UnitTesting obj = new UnitTesting();
27        int a=3;
28        int b=3;
29        int c=3;
30        int output = obj.triangle(a,b,c);
31        assertEquals(output, 0);
32    }
33}
34
35
36
37
38}
39

```

P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

```

public static boolean prefix(String s1, String s2)
{
    if (s1.length() > s2.length())
    {
        return false;
    }
    for (int i = 0; i < s1.length(); i++)
    {
        if (s1.charAt(i) != s2.charAt(i))
        {
            return false;
        }
    }
    return true;
}

```

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
s1 is Empty But s2 is not Empty	True
s1 is not Empty but s2 is Empty	False
s1="abc" ,s2="abcdgfg"	True
s1="abc" ,s2="abdefg"	False
s1="ABC" ,s2="abc"	False
s1="abc" ,s2="abc"	True
s1="defg" ,s2="ab"	False
Boundary Value Analysis	
s1="d" ,s2="de"	True
s1="de" , s2="d"	False
s1="p" ,s2="p"	True
s1="p" , s2="P"	False
s1="abcdejhk" ,s2="abcdejhk"	True
s1="" , s2=""	True
s1="abcdijklmn" ,s2="abcdjkl"	False

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows several Java files like UnitTesting.java, countItem.java, LinearSearch.java, binarySearch.java, triangle.java, and prefix.java.
- JUnit View:** Shows a green bar indicating "Finished after 0.026 seconds" with 3/3 runs, 0 errors, and 0 failures.
- Outline View:** Shows the class structure: test > prefix > test10(), test2(), and test3().
- Code Editor:** Displays the Java code for prefix.java and test.prefix.
- Coverage View:** Shows coverage analysis for Lab7_202001193 with 53.2% coverage, 84 covered instructions, 74 missed instructions, and 158 total instructions.

```

package test;
import static org.junit.Assert.*;
public class prefix {
    @Test
    public void test1() {
        UnitTesting obj = new UnitTesting();
        String s1="abcdefghijklm";
        String s2="abcdijkl";
        boolean output = obj.prefix(s1,s2);
        assertEquals(output, false);
    }
    @Test
    public void test2() {
        UnitTesting obj = new UnitTesting();
        String s1="";
        String s2="";
        boolean output = obj.prefix(s1,s2);
        assertEquals(output, true);
    }
    @Test
    public void test3() {
        UnitTesting obj = new UnitTesting();
        String s1="abc";
        String s2="def";
        boolean output = obj.prefix(s1,s2);
        assertEquals(output, true);
    }
}

```

P6: Consider again the triangle classification program (P4) with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right angled. Determine the following for the above program:

1). Identify the equivalence classes for the system.

- **Class 1:** Equilateral Triangle (Three sides are equal and non zero value)
- **Class 2:** Isosceles Triangle (Two sides are equal)
- **Class 3:** Scalene Triangle (all sides are different)
- **Class 4:** Right Angle triangle (satisfies Pythagoras Theorem)
- **Class 5:** Invalid (negative or zero value)
- **Class 6:** Non-Triangle(Sum of two sides is less than third side)

2) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

Test Case 1 For Class 1:

- a=7,b=7,c=7
- a=9,b=9,c=9

This Test Case Satisfied Class 1

Test Case 2 For Class 2:

→ a=3,b=3,c=5

→ a=6,b=6,c=8

This Test Case Satisfied Class 2

Test Case 3 For Class 3:

→ a=4, b=2, c=3

→ a=6,b=8,c=5

This Test Case Satisfied Class 3

Test Case 4 for Class 4:

→ a=3,b=4,c=5

→ a=6,b=8,c=10

This Test Case Satisfied Class 4

Test Case 5 for Class 5:

→ a=0,b=1,c=1;

→ a=5,b=0,c=5

This Test Case Satisfied Class 5

Test Case 6 for Class 6:

→ a=-1,b=2,c=3

→ a=5,b=-2,c=6

This Test case Satisfied Class 6

3). For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.**Test case:**

a=6,b=9,c=11

a=3,b=7,c=5

This above Test case Satisfies A+B > C.

4). For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.**Test case:**

a=5,b=9,c=5

a=4,b=7,c=4

This above Test case Satisfies A=C.

5). For the boundary condition $A = B = C$ case (equilateral triangle), identify test cases to verify the boundary.

Test case:

a=10,b=10,c=10

a=12,b=12,c=12

This above Test case Satisfies A=B=C.

6). For the boundary condition $A^2 + B^2 = C^2$ case (right-angle triangle), identify test cases to verify the boundary.

Test case:

a=8,b=5,c=13

a=3,b=4,c=5

This Above test case Satisfies $A^2 + B^2 = C^2$

7). For the non-triangle case, identify test cases to explore the boundary.

Test case:

a=6,b=2,c=3

a=2,b=4,c=2

This Above Test case is Satisfies non-triangle case.

8). For non-positive input, identify test points.

Test case:

a=-1,b=0,c=5

a=0,b=7,c=-3

This Above Test case is Satisfies non-positive case.

- **Section B:**

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 and so the focus is on creating test sets that satisfy some particular coverage criterion.

```

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

    Point t;
    min = 0;

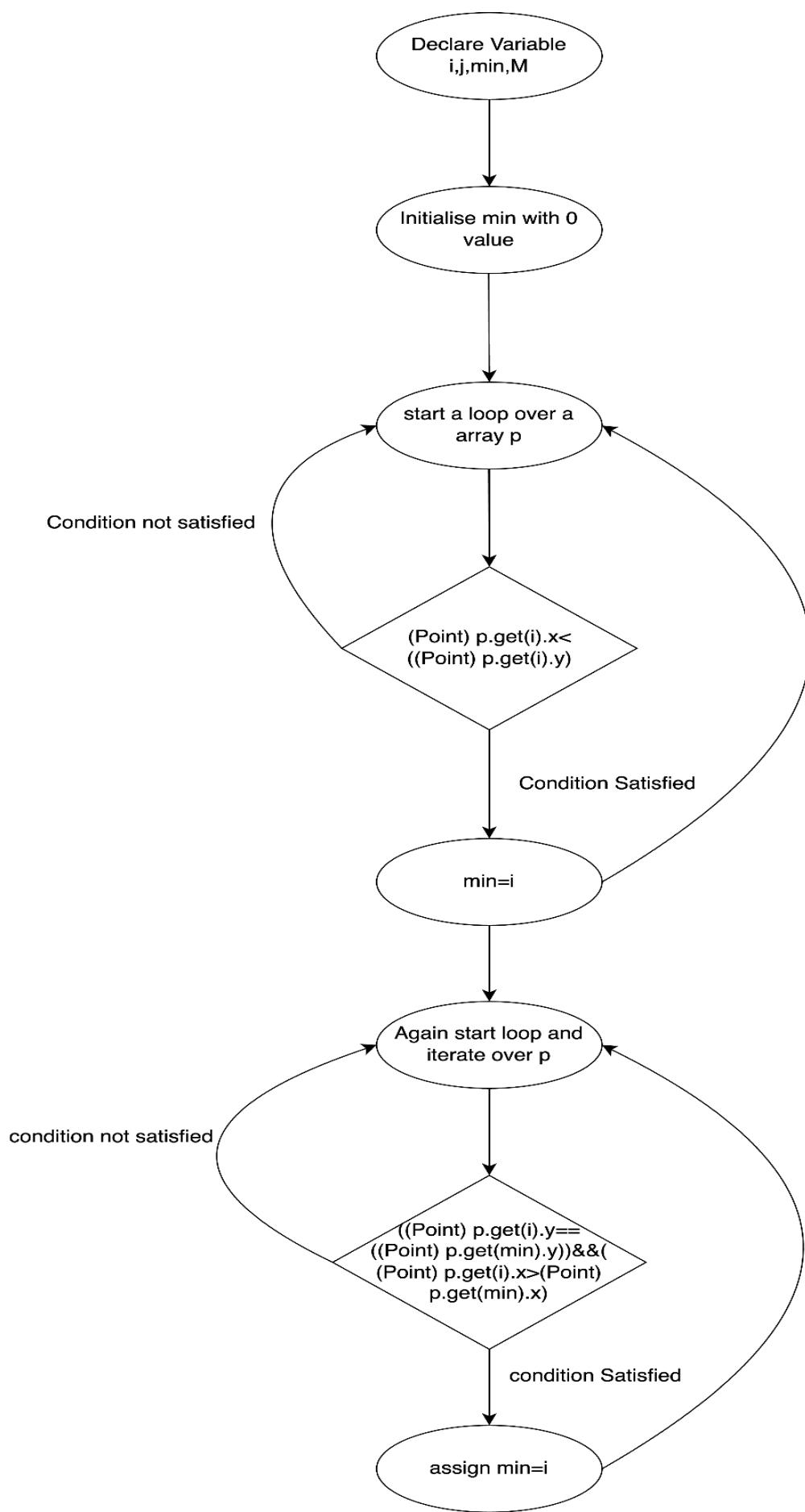
    // search for minimum:
    for(i=1; i < p.size(); ++i) {
        if( ((Point) p.get(i)).y <
            ((Point) p.get(min)).y )
        {
            min = i;
        }
    }

    // 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 Java code comprising the beginning of the doGraham method into a control flow graph (CFG).



2. Construct test sets for your flow graph that are adequate for the following criteria:

a. Statement Coverage:

→ Covers as many as lines of code

Test Case:

$p=\{(0,0),(2,0)\}$

$p=\{(0,6),(0,4),(1,2),(3,2),(5,2)\}$

So here x should decrease so we traverse as much as the loop statement and y should increase so we traverse as much as the next loop statement.

b. Branch Coverage:

→ Covers as many as Branch (if else)

Test Cases:

$p=\{(0,0),(2,0)\}$

$p=\{(0,6),(0,4),(1,2),(3,2),(5,2)\}$

$p=\{(1,5),(1,4),(1,3),(0,2),(0,1)\}$

c. Basic Condition Coverage:

→ Covers as many as Boolean operation

Test Cases:

$p=\{(0,0),(2,0)\}$

$p=\{(0,6),(0,4),(1,2),(3,2),(5,2)\}$

$p=\{(0,7),(0,6),(0,5),(1,3),(2,3),(3,3),(4,3),(5,3)\}$

So here we have to put points so that for minimum value y we get maximum point.