IT314
Lab Session: Software Engineering

Name: Vyom Atrikumar Patel
Student ID: 202001216
Date: 13/04/2023

SECTION A

1. **Previous Date:** Write a set of test cases (i.e., test suite) – specific set of data – to properly test the programs. Your test suite should include both correct and incorrect inputs.

➢ **Set of Test Cases:**

Sr. No.	Day	Month	Year	Expected output
1.	1	5	2015	30/4/2015
2.	20	6	2010	19/6/2010
3.	31	4	1962	INVALID
4.	18	12	2000	17/12/2000
5.	1	1	2001	31/12/2000
6.	29	2	1997	INVALID
7.	20	0	2005	INVALID
8.	15	13	1978	INVALID
9.	29	2	2012	28/2/2012
10.	1	3	2012	29/2/2012
11.	0	5	2012	INVALID
12.	12	3	2022	INVALID

➢ **Equivalence class partition:**

1. **Day**

Class Partition ID	Day Range	Expected output
1	1 <= Day <= 28	VALID
2	Day < 1	INVALID
3	Day > 31	INVALID
4	Day = 30	VALID EXCEPT 2nd month
5	Day = 29	VALID FOR LEAP YEAR
6	31	VALID EXCEPT 2nd month

2. **Month**

Class Partition ID	Month Range	Expected output
1	1 <= Month <= 12	VALID
2	Month < 1	INVALID
3	Month > 12	INVALID

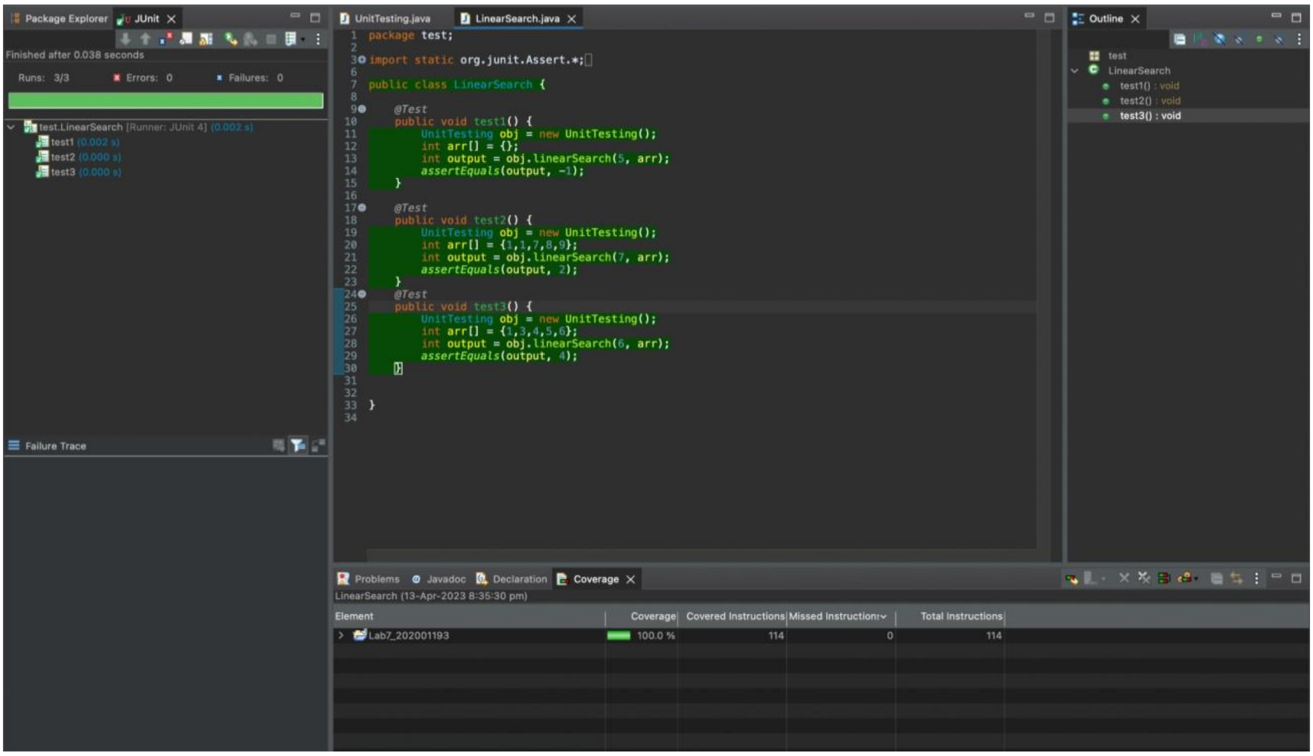
3. **Year**

Class Partition ID	Day Range	Expected output
1	1900 <= Year <= 2015	VALID
2	Year < 1900	INVALID
3	Year > 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

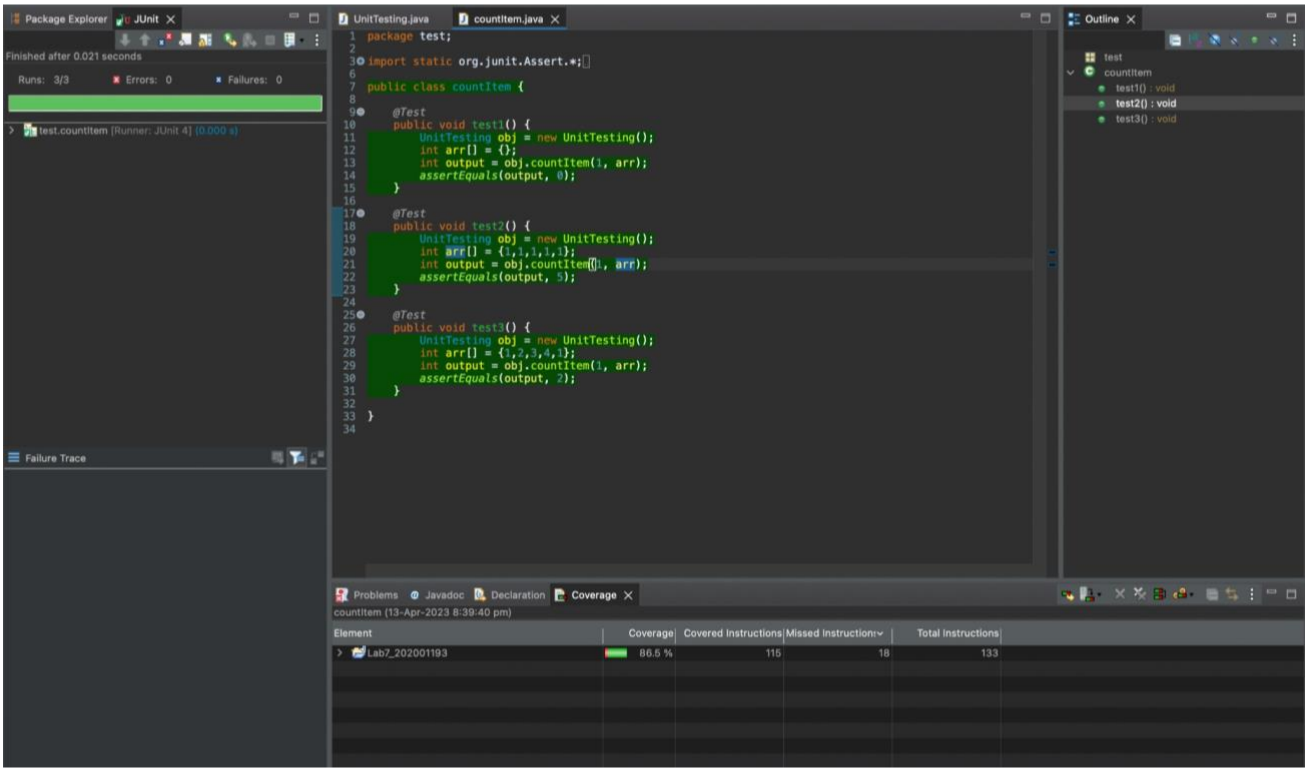


P2. The function `countItem` returns the number of times a value `v` appears in an array of integers a.

```
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



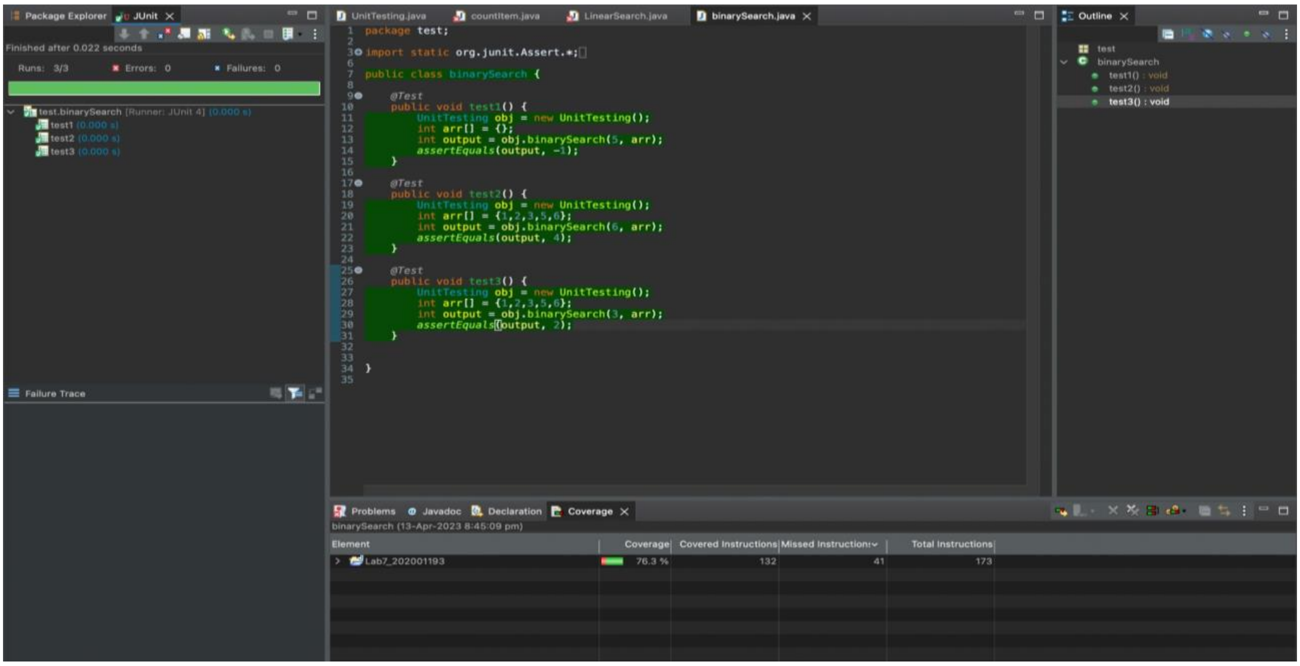
P3. The function `binarySearch` searches for a value `v` in an ordered array of integers a. If `v` appears in the array a, then the function returns an index `i`, such that `a[i] == v`; otherwise, -1 is returned.

Assumption: the elements in the array a are sorted in non-decreasing order.

```
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

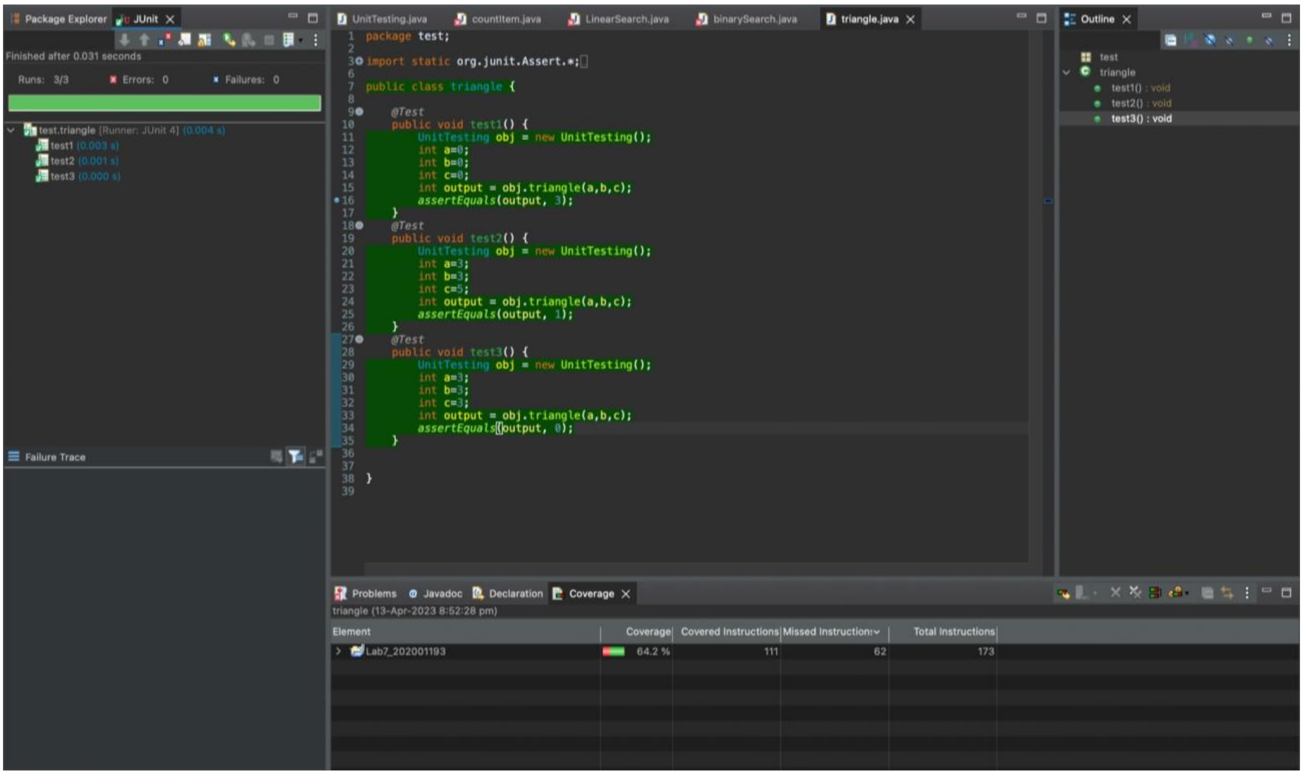


P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The function `triangle` takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two 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);
    }
```

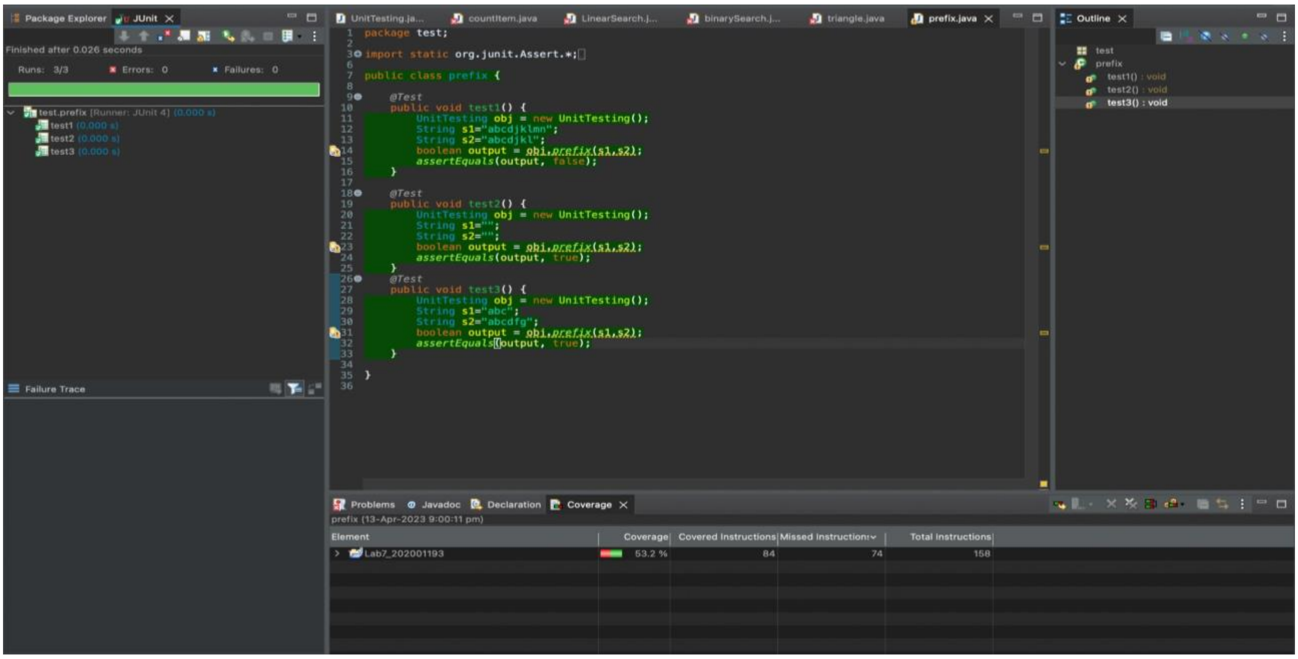
Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a=b<c where a>,b>0,c>0	ISOSCELES
a=b=c where a>0,b>0,c>0	EQUILATERAL
a<b+c, b<a+c, c<a+b where a>0,b>0,c>0	SCALENE
a=0 ,b=0,c=0	INVALID
a>=b+c, b>=a+c, c>=a+b where a>0,b>0,c>0	INVALID
a=0, b=c where b>0 ,c>0	INVALID
Boundary Value 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=214748364 7	EQUILATERAL
a=2147483647,b=2147483647,c=214748364 5	ISOSCELES
a=1,b=1,c=0	INVALID
a=1,b=1,c=2147483647	INVALID



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=”abcdfg”	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=”abcdjklmn” ,s2=”abcdjkl”	False



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:

A. 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)

B. 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

C. 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.

D. 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.

E. 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.

F. 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 $A^2 + B^2 = C^2$

G. 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.

H. For non-positive input, identify test points.

Test case:

- a=-1,b=0,c=5
- 1=0,b=7,c=3

This Above Test case is Satisfies non-positive case.

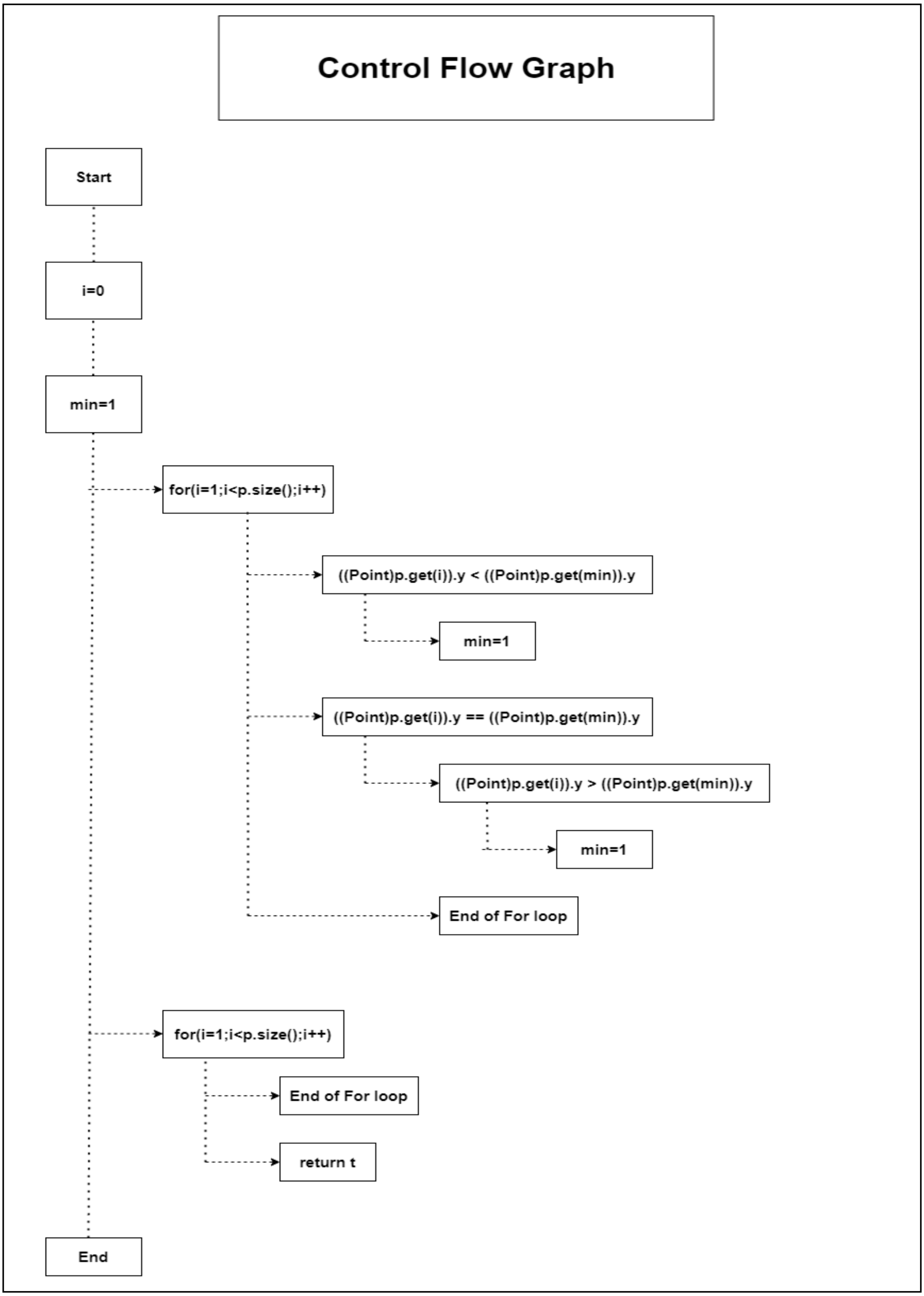
SECTION B

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 possible 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 possible 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.