IT-314
Software Engineering

LAB -7

Name: Patel Vedant
ID: 202001172

Section A

Solution:
Equivalent classes:

ID	class	Validity
E1	1<=date<=31	Valid
E2	date<1	invalid
E3	date >31	Invalid
E4	1<=month <=12	Valid
E5	month <1	Invalid
E6	month >12	Invalid
E7	1900<=year<=2015	Valid
E8	year <1900	invalid
E9	year>2015	invalid

Equivalence class test cases:

Equivalent class ID	day	month	year	output
E1	11	11	2011	10/11/2021
E2	0	11	2011	Invalid date
E3	34	5	2011	Invalid date
E4	2	2	2005	1/2/2005
E5	2	-1	1977	Invalid
E6	20	15	1943	Invalid
E7	1	3	2003	28/2/2003
E8	11	12	1899	Invalid
E9	4	3	2016	Invalid

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.

1. Enlist which set of test cases have been identified using Equivalence Partitioningand Boundary Value Analysis separately.
2. Modify your programs such that it runs on eclipse IDE, and then execute your test suites on the program. While executing your input data in a program, check whetherthe identified expected outcome (mentioned by you) is correct or not.

Programs:

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);
}
```

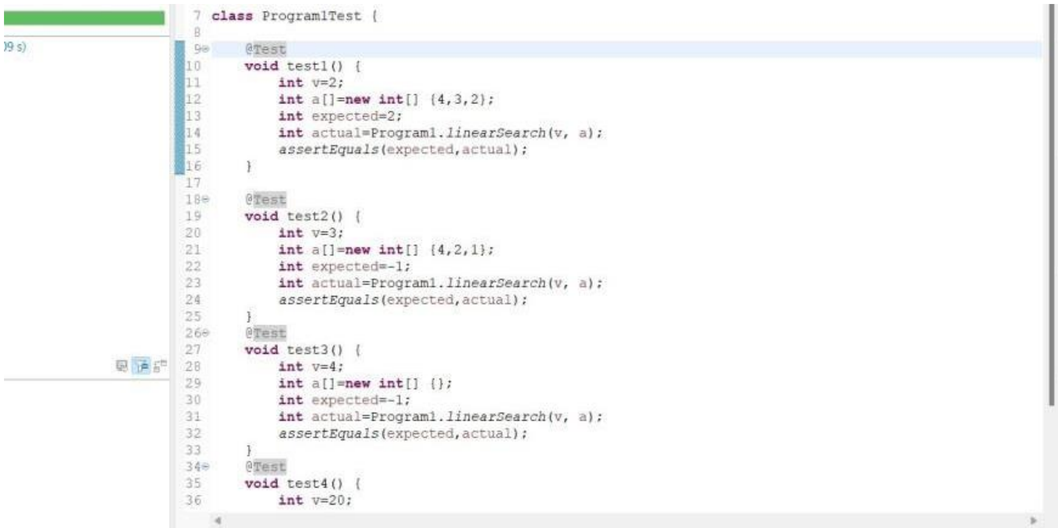
Equivalence Partitioning

Tester Action and Input Data	Expected Outcome
[2, 4, 6, 8, 10],v = 2	0
[1, 3, 5, 7, 9],v = 2	-1
[2, 4, 6, 8, 10],v = 11	-1
[-100, 100]	0
[1,2,3,4,5,6],v = 6	5
[],v = 10	-1
NULL,v = 111	-1

Boundary value analysis	
Tester Action and Input Data	**Expected Outcome**
NULL	-1
[],v = 4	-1

[5],v = 5	0
[5],v = 60	-1
[3,5],v = 3	0
[3,5],v = 5	1
[3,5],v = 14	-1
[4,3,2],v = 2	2
[1,2,3,4,5,6,7,8,9,1,0,11,111],v = 1	0
[1,2,3,4,5,6,7,8,9,1,0,11,111],v = 1111	-1

Junit Testing:



P2. The function count Item 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);
}
```

Equivalence Class partitioning for counting occurance

Tester Action and Input Data	Expected Outcome
[265, 41, 60, 80, 100],v = 100	1
[2655, 451, 6560, 1050, 1050],v = 1050	2
[[265545, 451, 65460, 1050, 105024]],v = 1	0
[[10,10,10,10]],v = 11	0
[],v = 100	0
NULL,v = 51	0
[0],v = 0	1
[-89,-89],v = -89	2

Boundary value analysis

Tester Action and Input Data	Expected Outcome
[1, 2, 3],v = 20	0
[4, 2, 3,2, 1],v = 2	2
15, 10, 15, 15],v = 15	3
[],v = 100	0
NULL,v = 51	0
[-100,100,100,100],v = 10000	0

Tester Action and Input Data	Expected Outcome
[-89,89],v = -89	1
[-890,890],v = 890	1

Junit Testing:

```
Program2.java    Program2Test.java
        int v=2;
        int a[]=new int[] {4,2,3,2,1};
        int expected=2;
        int actual=Program2.countItem(v, a);
        assertEquals(expected,actual);
    }
    @Test
    void test2() {
        int v=3;
        int a[]=new int[] {4,2,3};
        int expected=1;
        int actual=Program2.countItem(v, a);
        assertEquals(expected,actual);
    }
    @Test
    void test3() {
        int v=20;
        int a[]=new int[] {1,2,3};
        int expected=0;
        int actual=Program2.countItem(v, a);
        assertEquals(expected,actual);
    }
    @Test
    void test4() {
        int v=1;
        int a[]=new int[] {};
        int expected=0;
        int actual=Program2.countItem(v, a);
        assertEquals(expected,actual);
    }


}
```

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 is 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;
```

Equivalent testcases for binary search:

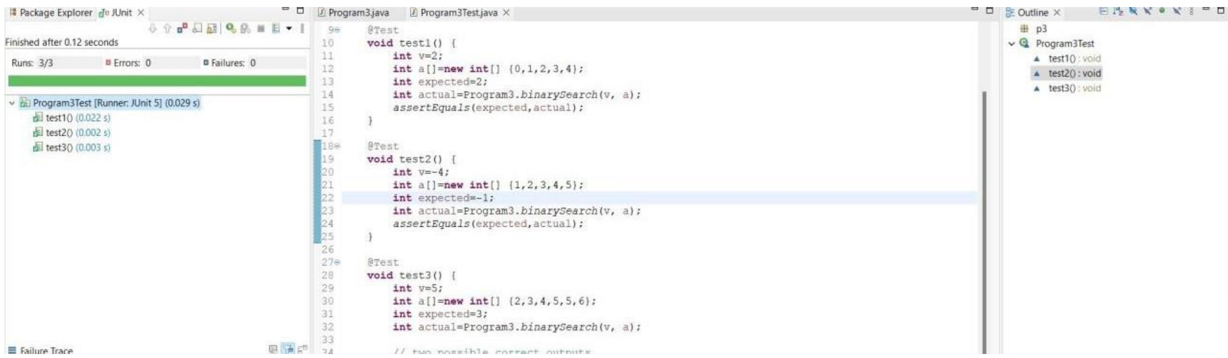
Tester Action and Input Data	**Expected Outcome**
[1, 21, 30, 40, 50],v = 21	1
[10, 20, 30, 40, 50, 60],v = 30	2
[10,100,1000,10000],v = 100000	-1
[,11,22,33,44],v = 444	-1
[11,20,200,300],v=11	0
[-100,-90,-80,100,1000],v = 10000	4

Tester Action and Input Data	Expected Outcome
[],v = 12	-1
NULL,v = 168	-1
[1,2],v = 3	-1
[1,3],v=3	1

Boundary value analysis

Tester Action and Input Data	Expected Outcome
[0,1, 2, 3, 4],v = 2	2
[2, 3, 4, 5, 5, 6],v = 5	3
[1,22,33,44,55],v = 66	-1
[2, 4, 6, 8, 10],v = 51	-1
[-100, 0, 1000],v = -100	0
[-100, 0, 1000],v = 1000	2
[],v=0	-1
NULL,v = 4	-1

Junit Testing:



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);
}
```

Equivalent class test cases

Tester Action and Input Data	Expected Outcome
a=2,b=2,c=2	EQUILATERAL
a=1,b=1,c=1	EQUILATERAL
a=0,b=0,c=0	INVALID
a=-1,b=-1,c=-1	INVALID
a=10,b=10,c=0	INVALID
a=17,b=17,c=5	ISOCELES
a=15,b=2,c=15	ISOCELES
a=6,b=11,c=5	SCALENE
a=16,b=21,c=25	SCALENE
a=-1,b=21,c=25	INVALID
a=2,b=3,c=4	SCALENE

Boundary value analysis

Tester Action and Input Data	Expected Outcome
a=2,b=2,c=2	EQUILATERAL
a=3,b=4,c=5	SCALENE
a=0,b=0,c=0	INVALID
a=INT_MAX,b = INT_MAX,c = INT_MAX	EQUILATERAL
a=INT_MIN,b=INT_MIN,c=INT_MIN	INVALID
a=5,b=5,c=10	ISOCELES
a=15,b=12,c=15	ISOCELES
a = INT_MAX,b = 1,c = INT_MAX	ISOCELES
a=1,b=2,c=3	INVALID
a = INT_MAX,b = 1,c = INT_MAX - 1	SCALENE

Junit Testing:

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;
}
```

Equivalent test cases

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1 = "",s2 = ""	true

Tester Action and Input Data	Expected Outcome
s1 = "po",s2 = "poojan"	true
s1 = "poo",s2 = "po"	false
s1 = "abc",s2 = ""	false
s1 = "",s2 = "abc"	true
s1 = "o",s2 = "ott"	true
s1 = "abc",s2 = "def"	false
s1 = "deg",s2 = "def"	false

Boundary value analysis

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1= "",s2 = ""	true
s1= "one",s2 = "two"	false
s1= "soft",s2 = "software"	true
s1= "abcd",s2 = ""	false
s1= "",s2 = "abcd"	true
s1 = "aef",s2 = "def"	false
s1 = "def",s2 = "deg"	false

Tester Action and Input Data	Expected Outcome
s1 = "a",s2 = "att"	true

Junit Testing:

```java
@Test
void test1() {
    String s1="soft",s2="software";
    boolean output=Program5.prefix(s1, s2);
    boolean expected=true;
    assertEquals(expected,output);
}
@Test
void test2() {
    String s1="abd",s2="abc";
    boolean output=Program5.prefix(s1, s2);
    boolean expected=false;
    assertEquals(expected,output);
}
@Test
void test3() {
    String s1="health",s2="health";
    boolean output=Program5.prefix(s1, s2);
    boolean expected=true;
    assertEquals(expected,output);
}
@Test
void test4() {
    String s1="one",s2="two";
    boolean output=Program5.prefix(s1, s2);
    boolean expected=false;
    assertEquals(expected,output);
}
```

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

The following are the equivalence classes for different types of triangles

Invalid case:

E1 : a+b<=c

E2 : a+c<=b

E3: b+c <=a

Equilateral case:

E4 : a=b,b=c,c=a

Isosceles case:

E5 : a=b , a!=c

E6: a= c, a!=b

E7: b=c, b!=a

Scalene case:

E8 : a!=b ,b!=c, c!=a

Right-angled triangle case:

E9 : a^2 + b^2 = c^2

E10: b^2+c^2 =a^2

E11: a^2 +c^2=b^2

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.(Hint: you must needto be ensure that the identified set of test cases cover all identified equivalence classes)

Test case	Output	Equivalent class covered
a=1.5, b=2.6 , c=4.1	Not a triangle	E1
a = -1.6, b=5, c=6	Not a triangle	E2
a=7.1, b=6.1, c=1	Not a triangle	E3
a=5.5, b= 5.5, c=5.5	Equilateral	E4
a=4.5, b=4.5, c=5	isosceles	E5

a=6, b=4, c=6	isosceles	E6
a=8, b=5, c=5	isosceles	E7
a=6,b=7,c=8	scalene	E8
a=3,b=4,c=5	Right-angled triangle	E9
a=0.13,b=0.12,c=0.05	Right-angled triangle	E10
a=7,b=25,c=23	Right-angled triangle	E11

All of the equivalent classes are covered with the above test cases

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

Test cases to verify the boundary condition:

1) a=5 b=5 c=9 (a+b=c)
2) a=5.5 b=5.5 c=10.9 (a+b just greater than c)
3) a=5.5 b=5 c=9.6 (a+b just less than c)

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.

Test cases to verify the boundary condition:

1) a=5 b=5 c=5 (a=c)
2) a=5.5 b=5.5 c=5.6 (a just less than c)
3) a=5.5 b=5 c=5.4 (a just greater than c)

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

Test cases to verify the boundary condition:

1) a=5 b=5 c=5 (a=b=c)
2) a=10 b=10 c=9 (a= b but a≠c) 3)
a=10 b=11 c=10 (a=c but a≠b)

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

Test cases to verify the boundary condition:

1) a=3, b=4, c=5 ($a^2+b^2=c^2$)
2) a=0.12, b=0.5 , c=0.14 (a^2+b^2 just less than c^2)
3) a=7 , b=23 , c=24 (a^2+b^2 just greater than c^2)

g) For the non-triangle case, identify test cases to explore the boundary.Test cases to verify the boundary condition:

1) a=1,b=2,c=3
2) a=5,b=5,c=10
3) a=0,b=0,c=0

h) For non-positive input, identify test points.Test points for non-positive input:

1) a=-4.0 b=3.2 c=4.5
2) a=5,b=-4.2,c=-3.2
3) a=4, b=5, c=-10

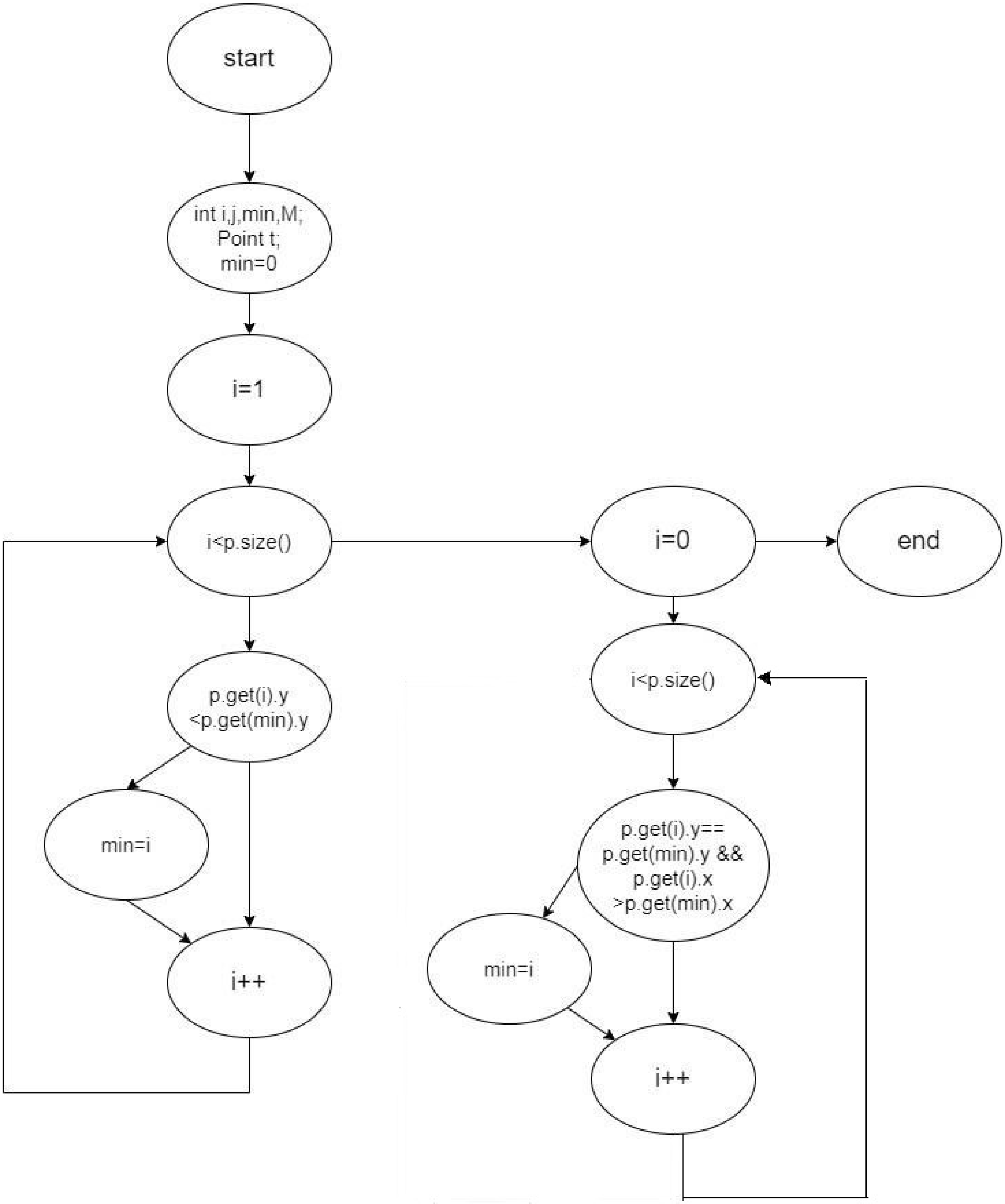
Section B

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 themethod. 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 thefocus is on creating test sets that satisfy some particular coverage criterion.

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

1. Convert the Java code comprising the beginning of the do Graham method into acontrol flow graph(CFG).

Control Flow graph (CFG):



start
int i,j,min,M;
Point t;
min=0
i=1
i<p.size()
p.get(i).y
<p.get(min).y
min=i
i++
i=0
end
i<p.size()
p.get(i).y==
p.get(min).y &&
p.get(i).x
>p.get(min).x
min=i
i++

2. **Construct test sets for your flow graph that are adequate for the** followingcriteria:

a. Statement Coverage.

b. Branch Coverage.

c. Basic Condition Coverage.

Let the re-written code with line numbers for statements be as follows:

```
1.    int i,j,min,M;
2.    Point t;
3.    min=0;
4.    for(i=1;i<p.size();++i)
      {
5.        if(((Point)P.get(i)).y<((Point)P.get(min)).y)
6.            min=i;
      }

7.    for(i=0;i<p.size();++i)
      {
8.        if(((Point)P.get(i)).y==((Point)P.get(min)).y
          &&((Point)P.get(i)).x>((Point)P.get(min)).x)
9.            min=i;
      }
```

The following are the test cases and their corresponding coverages of statements

Test cases:

1)p=[(x=1,y=3),(x=1,y=4),(x=2,y=1),(x=2,y=3)]

Statements covered = { 1,2,3,4,5,7,8}

Branches covered = {5,8}

Basic conditions covered = {5-false, 8-false}

2) p=[]

Statements covered = { 1,2,3}

Branches covered = {}

Basic conditions covered = {}

3) p=[(x=1,y=2)]

Statements covered = { 1,2,3,7,8}

Branches covered = {8}

Basic conditions covered = {}

4) p=[(x=2,y=3),(x=3,y=4),(x=1,y=2),(x=5,y=6)]

Statements covered = { 1,2,3,4,5,6,7}

Branches covered = {5,8}

Basic conditions covered = {5-false,true, 8-false}

5) p=[(x=1,y=5),(x=2,y=7),(x=3,y=5),(x=4,y=5),(x=5,y=6)]

Statements covered = { 1,2,3,4,5,6,7,8,9}

Branches covered = {5,8}

Basic conditions covered = {5-false,true, 8-false,true}