Name: Kashish Shroff
ID: 202001425
Lab 7: Software Testing

Section A

Equivalence Partitioning Test Cases:

Input data	Expected output
Valid input: day=1, month=1, year=1900	Invalid
Valid input: day=31, month=12, year=2015	Previous date
Invalid input: day=0, month=6, year=2000	An error message
Invalid input: day=32, month=6, year=2000	An error message
Invalid input: day=29, month=2, year=2001	An error message

Boundary Value Analysis Test Cases:

Input data	Expected output
Valid input: day=1, month=1, year=1900	Invalid
Valid input: day=31, month=12, year=2015	Previous date
Invalid input: day=0, month=6, year=2000	An error message
Invalid input: day=32, month=6, year=2000	An error message
Invalid input: day=29, month=2, year=2001	An error message
Valid input: day=1, month=6, year=2000	Previous date
Valid input: day=31, month=5, year=2000	Previous date

Valid input: day=15, month=6, year=2000	An error message
Invalid input: day=31, month=4, year=2000	An error message

Program 1:

```
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
‘v’ is not present in array ‘a’	-1
‘v’ is present in array ‘a’	Index of ‘v’

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array ‘a’	-1
‘v’ is present at 2nd index in array ‘a’	-2
‘v’ is not present in array ‘a’	-1

Test Cases:
1. v = 2, a = {4,3,2}, expected output: 2
2. v = 3, a = {4,2,1}, expected output: -1
3. v = 4, a = {}, expected output: -1
4. v = 2, a = {1,2,3,2,4}, expected output: 1

Junit Testing :

```
package tests;

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

    // Program 2
    public int countItem(int v, int a[])
    {
        int count = 0;
        for (int i = 0; i < a.length; i++)
        {
            if (a[i] == v)
                count++;
        }
        return count;
    }

    // Program 3
    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;
```

Program 2 :

```
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 Partitioning:

Tester Action and Input Data	Expected Outcome
‘v’ is not present in array ‘a’	0
‘v’ is present in array ‘a’	Number of ‘v’ times appears in array ‘a’

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome

Empty array ‘a’	-1
‘v’ is present once in array ‘a’	1
‘v’ is present multiple times in array ‘a’	Number of times ‘v’ is present
‘v’ is not present in array ‘a’	0

Test Cases:
1. v = 2, a = {4,2,3,2,1}, expected output: 2
2. v = 3, a = {4,2,3}, expected output: 1
3. v = 20, a = {1,2,3}, expected output: 0
4. v = 1, a = {}, expected output: 0

```java
package tests;

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

    // Program 2
    public int countItem(int v, int a[])
    {
        int count = 0;
        for (int i = 0; i < a.length; i++)
        {
            if (a[i] == v)
                count++;
        }
        return count;
    }

    // Program 3
    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);
    }
```

Program 3 :

```
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);
}
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
‘v’ is not present in array ‘a’	-1
‘v’ is present in array ‘a’	Index of ‘v’ in array ‘a’

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Empty array ‘a’	-1
‘v’ is present at 1st index in array ‘a’	0

‘v’ is not present in array ‘a’	-1

Test Cases:
1. v = 2, a = {0,1,2,3,4}, expected output: 2
2. v = -4, a = {1,2,3,4,5}, expected output: -1
3. v = 5, a = {2,3,4,5,5,6}, expected output: 3 or 4

Program 4 :

Equivalence Partitioning:

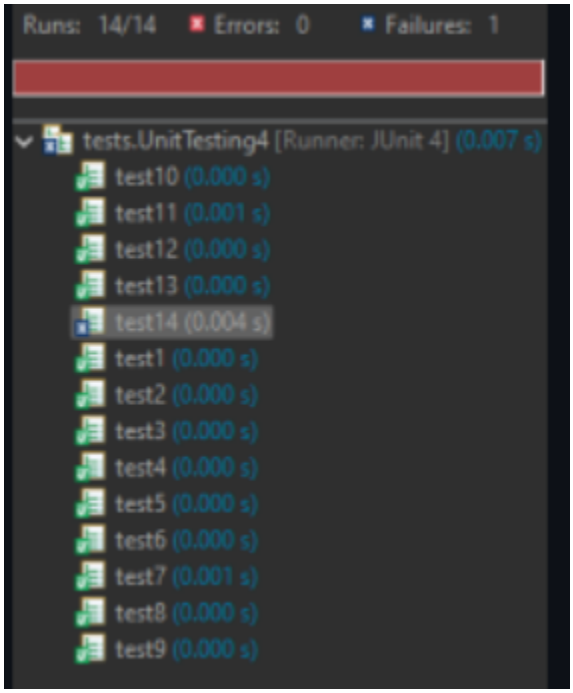
Tester Action and Input Data	Expected Outcome
Valid input: a=3, b=3, c=3	EQUILATERAL
Valid input: a=4, b=4, c=5	ISOSCELES
Valid input: a=5, b=4, c=3	SCALENE
Invalid input: a=0, b=0, c=0	INVALID
Invalid input: a=-1, b=2, c=3	INVALID
Valid input: a=1, b=1, c=1	EQUILATERAL
Valid input: a=2, b=2, c=1	ISOSCELES
Valid input: a=3, b=4, c=5	SCALENE
Invalid input: a=0, b=1, c=1	INVALID
Invalid input: a=1, b=0, c=1	INVALID
Invalid input: a=1, b=1, c=0	INVALID

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Invalid inputs: a = 0, b = 0, c = 0	INVALID
Invalid inputs: a + b = c or b + c = a or c + a = b (a=3, b=4, c=8)	INVALID
Equilateral triangles: a = b = c = 1	EQUILATERAL
Equilateral triangles: a = b = c = 100	EQUILATERAL
Isosceles triangles: a = b ≠ c = 10	ISOSCELES
Isosceles triangles: a ≠ b = c = 10	ISOSCELES
Isosceles triangles: a = c ≠ b = 10	ISOSCELES
Scalene triangles: a = b + c - 1	SCALENE
Scalene triangles: b = a + c - 1	SCALENE
Scalene triangles: c = a + b - 1	SCALENE

Maximum values: a, b, c = Integer.MAX_VALUE	INVALID
Minimum values: a, b, c = Integer.MIN_VALUE	INVALID

Output:



Program 5 :

Equivalence Partitioning:

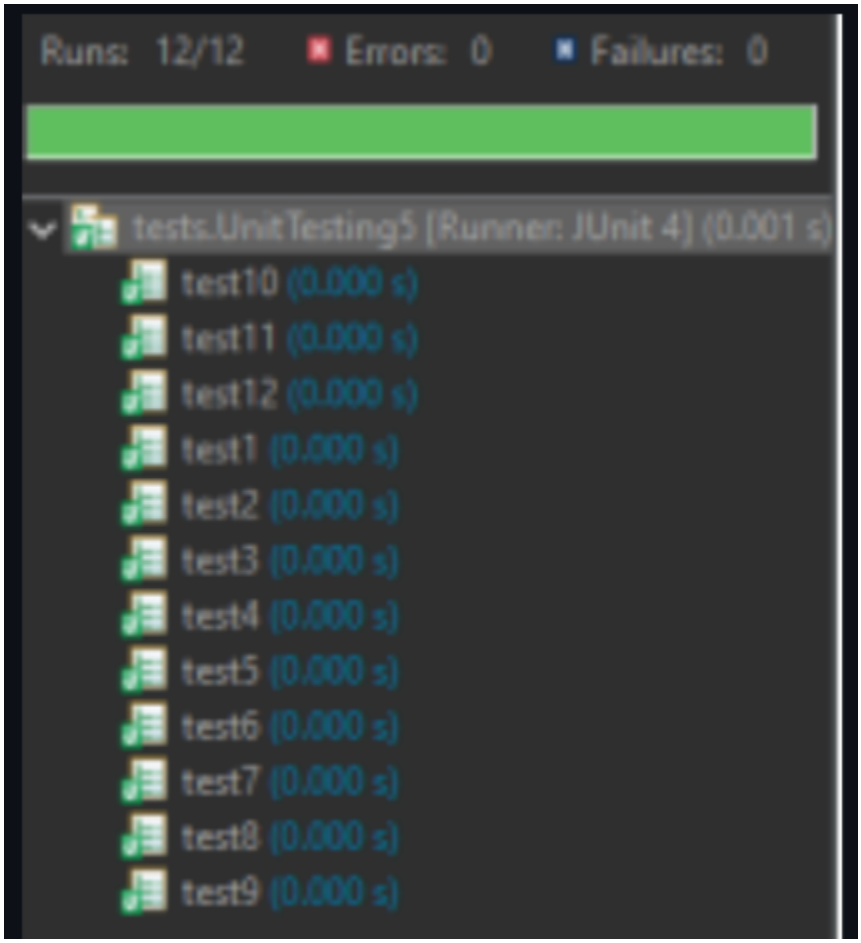
Tester Action and Input Data	Expected Outcome
Valid Inputs: s1 = "hello", s2 = "hello world"	true
Valid Inputs: s1 = "a", s2 = "abc"	true
Invalid Inputs: s1 = "", s2 = "hello world"	false

Invalid Inputs: s1 = "world", s2 = "hello world"	false

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
s1 = "", s2 = "abc"	False
s1 = "ab", s2 = "abc"	True
s1 = "abc", s2 = "ab"	False
s1 = "a", s2 = "ab"	True
s1 = "aaaaaaaaaaaaaaaaaaaaaa", s2 = "aaaaaaaaaaaaaaaaaaaaab"	True
s1 = "abc", s2 = "abc"	True
s1 = "a", s2 = "b"	False
s1 = "a", s2 = "a"	True
s1 = "a", s2 = "b"	False
s1 = "a", s2 = " "	False

Output :



Program 6:

a) Equivalence classes for the system
EC1: All sides are positive, real numbers.
EC2: One or more sides are negative or zero.
EC3: The sum of the lengths of any two sides is less than or equal to the length of the remaining side (impossible lengths).
EC4: The sum of the lengths of any two sides is greater than the length of the remaining side (possible lengths).

b) Test cases to cover equivalence classes
TC1 (EC1): A=3, B=4, C=5 (right-angled triangle)
TC2 (EC1): A=5, B=5, C=5 (equilateral triangle)
TC3 (EC1): A=5, B=6, C=7 (scalene triangle)
TC4 (EC1): A=5, B=5, C=7 (isosceles triangle)
TC5 (EC2): A=-2, B=4, C=5 (invalid input)

TC6 (EC2): A=0, B=4, C=5 (invalid input)

c) Test cases for boundary condition A+B>C
TC7 (EC4): A=4, B=3, C=6 (sum of A and B > C)

d) Test case for boundary condition A=C
TC8 (EC4): A=5, B=6, C=5 (A equals to C)

e)Test case for the boundary condition A=B=C
TC9 (EC4): A=5, B=5, C=5 (all sides are equal)

f) Test case for the boundary condition A^2 + B^2 = C^2

TC10 (EC4): A=3, B=4, C=5 (right-angled triangle)

g) Test cases for the boundary condition of non-triangle case:
TC11 (EC3): A=2, B=2, C=4 (sum of A and B is equal to C)

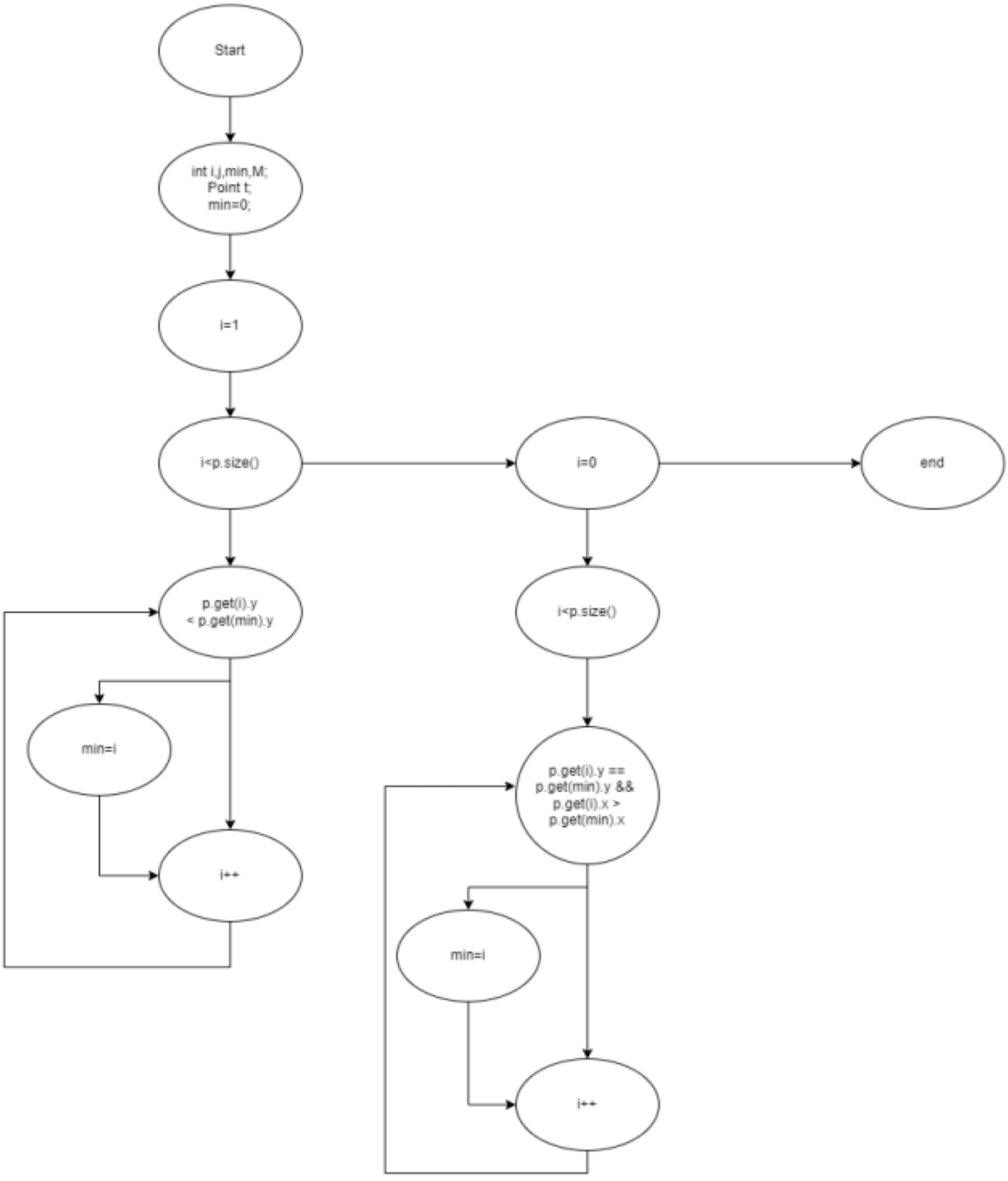
h) For non-positive input, identify test points.

TP1 (EC2): A=0, B=4, C=5 (invalid input)
TP2 (EC2): A=-2, B=4, C=5 (invalid input)

The Test cases TC1 to TC10 covers all identified equivalence classes.

SECTION B :

Control flow graph



Branch coverage test sets: Each branch of the code is run at least once.

Test 1: p = empty vector

Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component
Test 4: p = vector with two points with different y components
Test 5: p = vector with three or more points with different y components, and none of them have the same x component
Test 6: p = vector with three or more points with the same y component, and some of them have the same x component
Test 7: p = vector with three or more points with the same y component, and all of them have the same x component

Statement coverage test cases: The code runs each statement at least once.

Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component
Test 4: p = vector with two points with different y components
Test 5: p = vector with three or more points with different y components
Test 6: p = vector with three or more points with the same y component

Basic condition coverage test sets: There is at least one execution of each boolean expression.

Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component, and the first point has a smaller x component
Test 4: p = vector with two points with the same y component, and the second point has a smaller x component
Test 5: p = vector with two points with different y components
Test 6: p = vector with three or more points with different y components, and none of them have the same x component
Test 7: p = vector with three or more points with the same y component, and some of them have the same x component
Test 8: p = vector with three or more points with the same y component, and all of them have the same x component.

Examples of such test cases Test cases :
1) p=[(x=2,y=2),(x=2,y=3),(x=1,y=3),(x=1,y=4)]
2) p=[(x=2,y=3),(x=3,y=4),(x=1,y=2),(x=5,y=6)]
3) p=[(x=1,y=5),(x=2,y=7),(x=3,y=5),(x=4,y=5),(x=5,y=6)]
4) p=[(x=1,y=2)] 5) p=[]

All of the tests mentioned above are covered by these 5 test cases.