Software Engineering
Lab-7

Name-Amol Patel
ID-202001456

Section A:

Previous Date Problem

Test Cases: For Boundary Value Analysis (DD/MM/YYYY)

Test Case ID	(DD/MM/YYYY)	Expected Output
1	01/03/2004	29/02/2004
2	01/03/2007	28/02/2007
3	29/02/2007	Invalid
4	30/02/2004	Invalid
5	01/01/1900	31/12/1899
6	31/12/2015	30/12/2015
7	31/12/1899	Invalid
8	31/13/1900	Invalid
9	32/12/1900	Invalid
10	18/05/2001	17/05/2001

Equivalence Class Partitions

Day (DD):

Partition ID	Range	Status
E1	1<=DD<=28	Valid
E2	DD<1	Invalid
E3	DD>31	Invalid
E4	DD=30	Valid except month = 2
E5	DD=29	Valid for leap year
E6	DD=31	Valid except month = 2

Month (MM):

Partition ID	Range	Status
E7	1<=MM<=12	Valid
E8	MM<1	Invalid
E9	MM>12	Invalid

Year (YYYY):

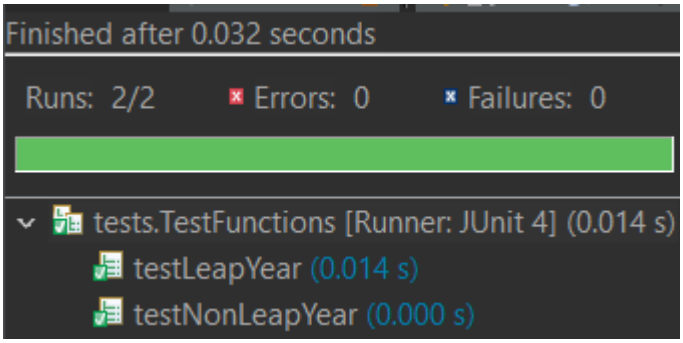
Partition ID	Range	Status

E10	1900<=YYYY<=2015	Valid
E11	YYYY<1900	Invalid
E12	YYYY>2015	Invalid

Junit Testing Code:

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestFunctions {
   @Test
   public void testNonLeapYear() {
      assertEquals("29/02/2004", UnitTest.findPreviousDate("01/03/2004"));
      assertEquals("28/02/2007", UnitTest.findPreviousDate("01/03/2007"));
      assertEquals("INVALID", UnitTest.findPreviousDate("29/02/2007"));
      assertEquals("INVALID", UnitTest.findPreviousDate("30/02/2004"));
   }
   @Test
   public void testNonLeapYear() {
      assertEquals("31/12/1899", UnitTest.findPreviousDate("01/01/1900"));
      assertEquals("30/12/2015", UnitTest.findPreviousDate("31/12/2015"));
      assertEquals("INVALID", UnitTest.findPreviousDate("31/12/1899"));
      assertEquals("INVALID", UnitTest.findPreviousDate("31/13/1900"));
      assertEquals("INVALID", UnitTest.findPreviousDate("32/12/1900"));
      assertEquals("17/05/2001", UnitTest.findPreviousDate("18/05/2001"));
   }
}
```

Junit Testing Output



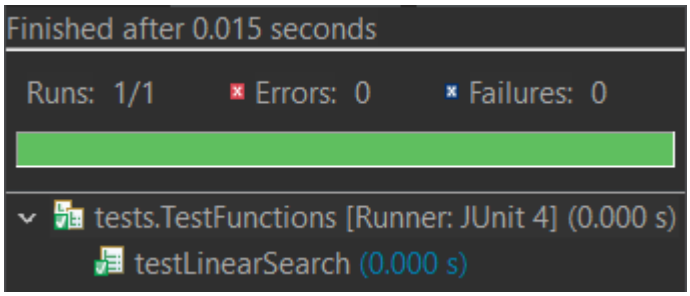
P1: LinearSearch Problem

Tester Action and Input Data	Expected Outcome	Test Case Type
arr=[-3,4,-1,1,1,13,5], v = 1	3	Equivalence Partitioning (first ocurrance)
arr=[-3,4,-1,-1,0,13,5], v = 2	-1	Equivalence Partitioning (not found)
arr=[-3,4,-1,-1,0,13,5], v = -3	0	Boundary Value Analysis
arr=[3,4,1,1,0,13,5], v = 5	6	Boundary Value Analysis
arr=[7], v = 7	0	Boundary Value Analysis
arr=[3,4,1,-1,0,13,5], v = -1	3	Boundary Value Analysis
arr=[], v = 5	-1	Boundary Value Analysis
NULL, v = 5	-1	Boundary Value Analysis

Junit Testing Code

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestFunctions {
   @Test
   public void testLinearSearch() {
      int[] a1 = {-3,4,-1,1,1,13,5};
      int v1 = 1;
      assertEquals(3, UnitTest.linearSearch(v1, a1));

      int[] a2 = {-3,4,-1,-1,0,13,5};
      int v2 = 2;
      assertEquals(-1, UnitTest.linearSearch(v2, a2));

      int[] a3 = {};
      int v3 = 5;
      assertEquals(-1, UnitTest.linearSearch(v3, a3));

      int[] a4 = {7};
      int v4 = 7;
      assertEquals(0, UnitTest.linearSearch(v4, a4));
   }
}
```

Junit Testing Output



P2: CountItem Problem

Tester Action and Input Data		Test Case Type
arr=[1,1,2,4,6,1,4,32,1,5,76,2,1], v = 1	5	Equivalence Partitioning
arr=[1,2,3,4,5,6,7], v = 0	0	Equivalence Partitioning
arr=[1,-1,2,4,-6,6,-4,-32,1,5,-76,2,-1], v = 6	1	Boundary Value Analysis
arr=[6,6,6], v = 6	3	Boundary Value Analysis
arr=[6,6,6], v = -6	0	Boundary Value Analysis
arr=[], v = 5	0	Boundary Value Analysis

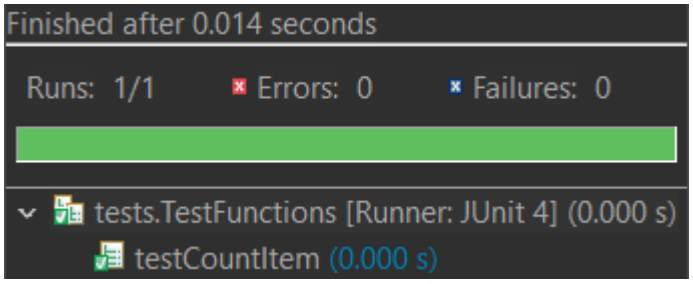
NULL, v = 5	0	Boundary Value Analysis

Junit Testing Code

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestFunctions {

   @Test
   public void testCountItem() {
      int[] a1 = {1,1,2,4,6,1,4,32,1,5,76,2,1};
```

```
        int v1 = 1;
        assertEquals(5, UnitTest.countItem(v1, a1));

        int[] a2 = {1,2,3,4,5,6,7};
        int v2 = 0;
        assertEquals(0, UnitTest.countItem(v2, a2));

        int[] a3 = {6,6,6};
        int v3 = 6;
        assertEquals(3, UnitTest.countItem(v3, a3));

        int[] a4 = {};
        int v4 = 5;
        assertEquals(0, UnitTest.countItem(v4, a4));
    }
}
```

Junit Testing Output



P3: BinarySearch Problem

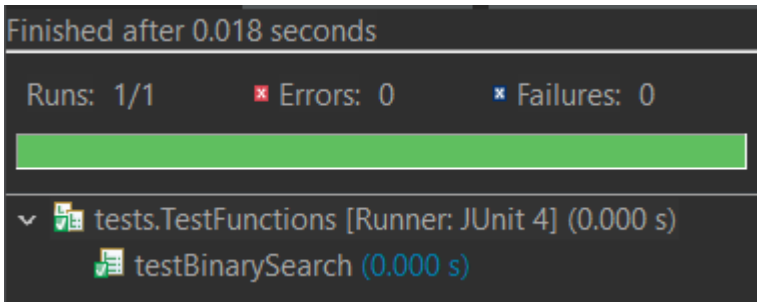
Tester Action and Input Data	Expected Outcone	Test Case Type
arr=[0,1,2,3,4,5,6,7], v = 6	6	Equivalence Partitioning
arr=[-100,-90,-80,10,100],v = -90	1	Equivalence Partitioning
arr=[0,1,2,4], v = 5	-1	Equivalence Partitioning
arr=[0,1,2,3,7], v = 7	4	Boundary Value Analysis
arr=[0,2,4,5,7], v = 0	0	Boundary Value Analysis
arr=[0,2,4,5,7], v = 4	2	Boundary Value Analysis
arr=[4], v = 4	0	Boundary Value Analysis
arr=[0,2], v = 2	1	Boundary Value

		Analysis
arr=[], v = 5	-1	Boundary Value Analysis
NULL, v = 5	-1	Boundary Value Analysis

Junit Testing Code

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestFunctions {

   @Test
   public void testBinarySearch() {
      int[] a1 = {0,1,2,3,4,5,6,7};
      int v1 = 6;
      assertEquals(6, UnitTest.binarySearch(v1, a1));

      int[] a2 = {-100,-90,-80,100,1000};
      int v2 = -90;
      assertEquals(1, UnitTest.binarySearch(v2, a2));

      int[] a3 = {0,1,2,4};
      int v3 = 5;
      assertEquals(-1, UnitTest.binarySearch(v3, a3));

      int[] a4 = {4};
      int v4 = 4;
      assertEquals(0, UnitTest.binarySearch(v4, a4));

      int[] a5 = {};
      int v5 = 5;
      assertEquals(-1, UnitTest.binarySearch(v5, a5));

      int[] a6 = {0,3};
      int v6 = 3;
      assertEquals(1, UnitTest.binarySearch(v6, a6));
   }
}
```

Junit Testing Output



P4: Triangle Problem

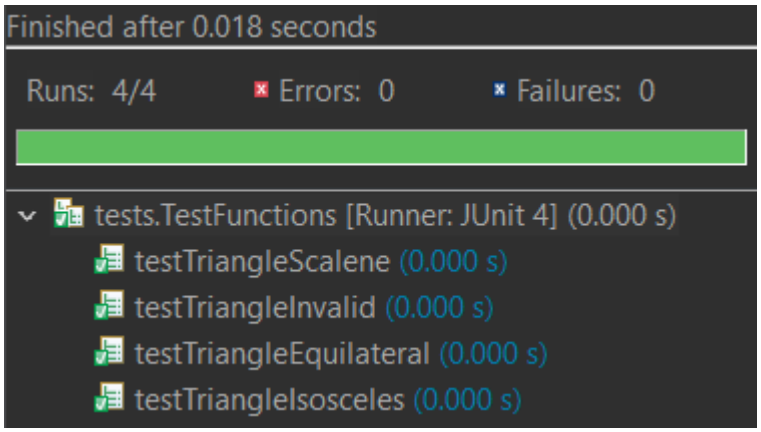
Tester Action and Input Data	Expected Outcome	Test Case Type
a = 0, b = 0, c = 0	INVALID	Boundary Condition
a = 0, b = 5, c = 0	INVALID	Boundary Condition
a = 9, b = 8, c = 0	INVALID	Boundary Condition
a = 9, b = 8, c = 100	INVALID	Equivalance Partitioning
a = -8, b = -8, c = -8	INVALID	Boundary Condition
a = 10, b = 10, c = 10	EQUILATERAL	Equivalence Partitioning
a = 100, b = 100, c = 100	EQUILATERAL	Equivalence Partitioning
a = 10, b = 10, c = 12	ISOSCELES	Equivalence Partitioning

a = 12, b = 10, c = 10	ISOSCELES	Equivalence Partitioning
a = 150, b = 100, c = 150	ISOSCELES	Equivalence Partitioning
a = 5, b = 6, c = 10	SCALENE	Boundary Condition

Junit Testing Code

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class TestFunctions {

   @Test
   public void testTriangleInvalid() {
      assertEquals(3, UnitTest.triangle(0, 0, 0));
      assertEquals(3, UnitTest.triangle(9, 8, 100));
      assertEquals(3, UnitTest.triangle(-8, -8, -8));
   }

   @Test
   public void testTriangleEquilateral() {
      assertEquals(0, UnitTest.triangle(10, 10, 10));
   }

   @Test
   public void testTriangleIsosceles() {
      assertEquals(1, UnitTest.triangle(10, 10, 12));
      assertEquals(1, UnitTest.triangle(12, 10, 10));
      assertEquals(1, UnitTest.triangle(150, 100, 150));
   }

   @Test
   public void testTriangleScalene() {
      assertEquals(2, UnitTest.triangle(5, 6, 10));
   }
}
```

Junit Testing Output



P5: Prefix Problem

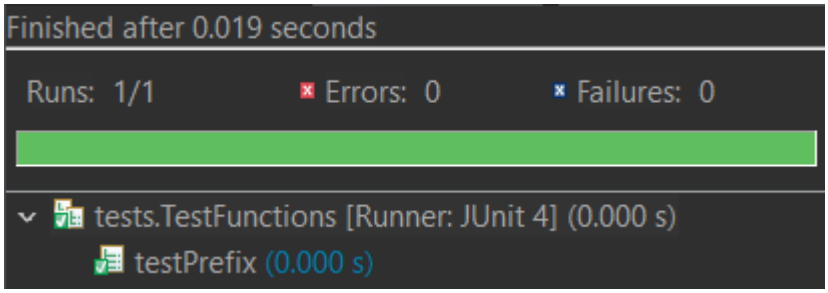
Tester Action and Input Data	Expected Outcome	Test Case Type
s1="", s2=""	true	Boundary Condition
s1="hell", s2="hello"	true	Equivalence Partitioning
s1="hell", s2="hell"	true	Boundary Condition
s1="", s2="hell"	true	Boundary Condition
s1="hello", s2="hell"	false	Equivalence Partitioning
s1="he ll", s2="hell"	false	Boundary Condition
s1=" hell", s2="hello"	false	Boundary Condition

Junit Testing Code

```
package tests;

import static org.junit.Assert.*;
```

```
import org.junit.Test;
public class TestFunctions {
   @Test
   public void testPrefix() {
      assertTrue(UnitTest.prefix("", ""));
      assertTrue(UnitTest.prefix("hell", "hello"));
      assertTrue(UnitTest.prefix("hell", "hell"));
      assertTrue(UnitTest.prefix("", "hell"));
      assertFalse(UnitTest.prefix("hello", "hell"));
      assertFalse(UnitTest.prefix("he ll", "hell"));
   }
}
```

Junit Testing Output



P6: Assumes the problem domain of P4 with A, B, and C as floating values instead of integers

A. Equivalence classes for the system

The possible equivalence classes and their corresponding conditions are as follows:

Equivalence Class	Neccessary Condition
Invalid Triangle	A > B+C or B > A+C or C > A+B
Scalene Triangle	A != B and B != C and C != A
Isoceles Triangle	either A == B or B == C or A ==
C Equilateral Triangle	A == B and B == C
Right-angle Triangle	A^2 + B^2 = C^2 or A^2 = B^2 + C^2 or B^2 + A^2 = C^2

B. Test Cases for Equivalence Classes

Following are list of test cases each belonging to one of the defined Equivalence class

Test Case	Condition	Expected Outcome
1	A = 7, B = 7, C = 7	Equilateral Triangle
2	A = 5, B = 12, C = 13	Right-angle Triangle
3	A = 4, B = 4, C = 3	Isosceles Triangle
4	A = 4, B = 6, C = 7	Scalene Triangle
5	A = 1, B = 2, C = 3	Invalid Triangle

C. Boundary Condition A + B > C (scalene triangle)

Below is the list of possible corner cases looking like scalene triangle but are not

Test Case	Condition	Expected Outcome
1	A = 2, B = 1, C = 5	Invalid Triangle
2	A = 1, B = 2, C = 4	Invalid Triangle
3	A = 2, B = 2, C = 5	Invalid Triangle
4	A = 0.1, B = 0.2, C = 0.3	Invalid Triangle
5	A = 1, B = 2, C = 2.5	Scalene Triangle

D. Boundary Condition A = C (isosceles triangle)

Below is the list of possible corner cases looking like isosceles triangle but are not

Test Case	Condition	Expected Outcome
1	A = -4, B = 3, C = -4	Invalid Triangle
2	A = 1, B = 2, C = 1	Invalid Triangle
3	A = 1, B = 4, C = 1	Invalid Triangle
4	A = 0.1, B = 0.4, C = 0.1	Invalid Triangle
5	A = 0.15, B = 0.25, C = 0.15	Isosceles Triangle

E. Boundary Condition A = B = C (equilateral triangle)

Below is the list of possible corner cases looking like isosceles triangle but are not

Test Case	Condition	Expected Outcome
1	A = -3, B = -3, C = -3	Invalid Triangle
2	A = 0, B = 0, C = 0	Invalid Triangle
3	A = 7, B = 7, C = 7	Equilateral Triangle
4	A = 0.2, B = 0.2, C = 0.2	Equilateral Triangle

F. Boundary Condition

A^2+B^2=C^2 (RIGHT ANGLED TRAINGLE)

Below is the list of possible corner cases looking like right-angle triangle but are not

Test Case	Condition	Expected Outcome
1	A = 5, B = 12, C = 13	Right Angled Triangle
2	A = -4, B = -3, C = 5	Invalid Triangle
3	A = -1, B = -1.414, C = 1.73	Invalid Triangle
3	A = 1, B = 1.414, C = 1.73	Right Angled Triangle

G. Non-triangle Case

Below is the list of possible Invalid Triangle cases

Test Case	Condition	Expected Outcome

1	A = 3, B = 4, C = 9	Invalid Triangle
2	A = -4, B = -2, C = 5	Invalid Triangle
3	A = -1, B = -1, C = -1	Invalid Triangle
4	A = 111, B = 1.414, C = 9.73	Invalid Triangle
5	A = 1, B = 53, C = 9.73	Invalid Triangle
6	A = 1, B = 1.414, C = -9.73	Invalid Triangle
7	A = 0, B = 0, C = 0	Invalid Triangle

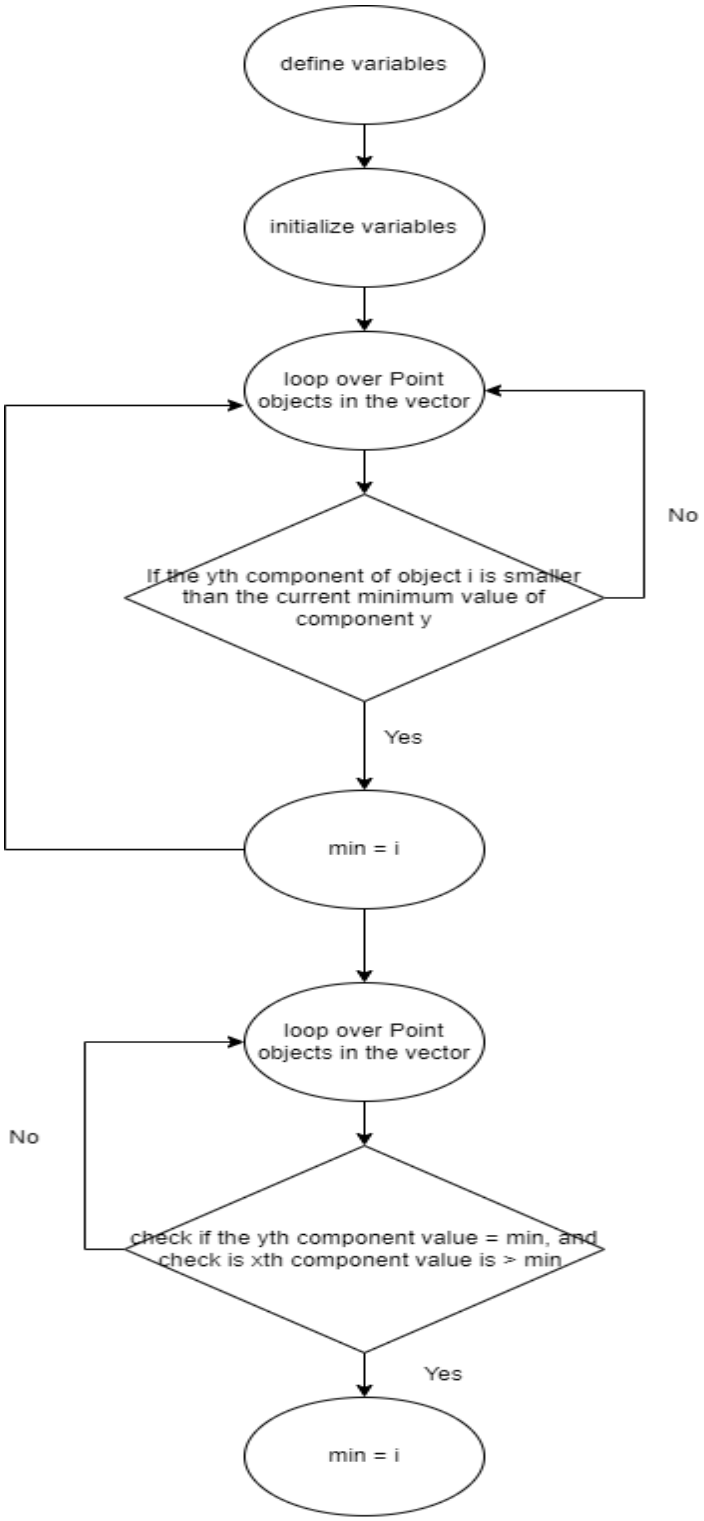
H. Non-positive Input

Below is the list of possible Invalid Triangle cases

Test Case	**Condition**	**Expected Outcome**
1	a=-1, b=2, c=1	Invalid Triangle
2	a=-4, b=-5, c=-7	Invalid Triangle
3	a=1, b=-5, c=7	Invalid Triangle

Section-B:

1. Control flow diagram



2. Test sets

Statement coverage test sets: To achieve statement coverage, we need to make sure that every statement in the code is executed 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

Branch coverage test sets: To achieve branch coverage, we need to make sure that every possible branch in the code is taken 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

Basic condition coverage test sets: To achieve basic condition coverage, we need to make sure that every basic condition in the code (i.e., every Boolean subexpression) is evaluated as both true and false 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, 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.