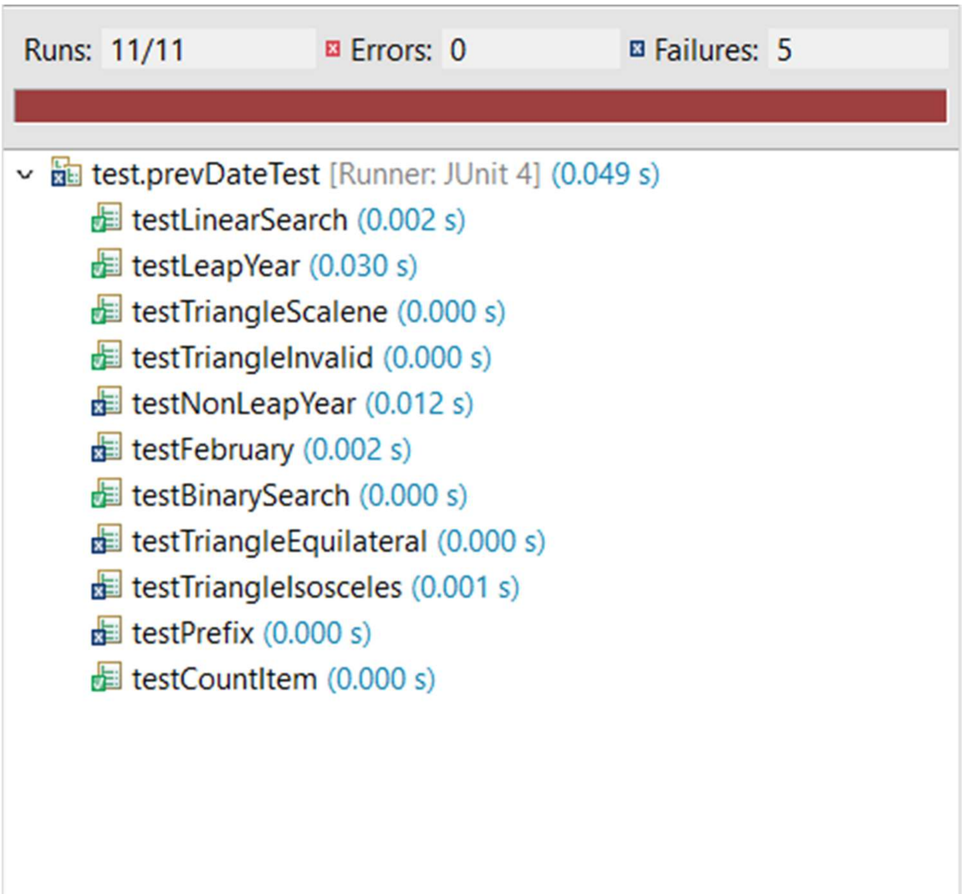


Equivalence Partitioning Test Cases:

Tester Action and Input Data	Expected Outcome
Valid input: day=1, month=1, year=1900	Invalid date
Valid input: day=31, month=12, year=2015	Previous date
Invalid input: day=0, month=6, year=2004	An error message
Invalid input: day=32, month=6, year=2003	An error message
Invalid input: day=30, month=2, year=2002	An error message

Boundary Value Analysis Test Cases:

Tester Action and Input Data	Expected Outcome
Valid input: day=1, month=1, year=1900	Invalid date
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=2001	An error message
Invalid input: day=29, month=2, year=2002	An error message
Valid input: day=1, month=6, year=2005	Previous date
Valid input: day=29, month=5, year=2000	Previous date
Valid input: day=15, month=6, year=2000	Previous date
Invalid input: day=31, month=4, year=2000	An error message

Problem 1 :

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

public class LinearSearchTest {

 @Test
 public void testExistingValue() {
  int[] arr = {1, 2, 3, 4, 5};
  int index = linearSearch(3, arr);
  assertEquals(3, index);
 }

 @Test
 public void testFirstElement() {
  int[] arr = {1, 2, 3, 4, 5};
  int index = linearSearch(1, arr);
  assertEquals(0, index);
 }

 @Test
 public void testNonExistingValue() {
  int[] arr = {1, 2, 3, 4, 5};
  int index = linearSearch(7, arr);
  assertEquals(-1, index);
 }
```

```
  @Test
  public void testLastElement() {
    int[] arr = {1, 2, 3, 4, 5};
    int index = linearSearch(5, arr);
    assertEquals(4, index);
  }

  @Test
  public void testNullArray() {
    int[] arr = null;
    int index = linearSearch(1, arr);
    assertEquals(-1, index);
  }

  @Test
  public void testEmptyArray() {
    int[] arr = {};
    int index = linearSearch(1, arr);
    assertEquals(-1, index);
  }
}
```

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an non-empty array a[]	-1
Test with v as a non-existent value and a empty array a[]	-1
Test with v as an existent value and an non-empty array a[]	the index of v in a[]
Test with v as an existent value and a empty array a[] where v exists	-1
Test with v as an existent value and a non-empty array a[] where v does not exist	-1

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an non-empty array a[]	-1
Test with v as a non-existent value and a empty array a[]	-1
Test with v as an existent value and an array a[] of length 0	-1
Test with v as an existent value and an array a[] of length 1, where v exists	0
Test with v as an existent value and an array a[] of length 1, where v does not exist	-1
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the beginning of the array	0
Test with v as an existent value and an array a[] of length greater than 1, where v exists in the middle of the array	the index where v is found
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the end of the array	the last index where v is found

Problem 2 :

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and an empty non-array a[]	0
Test with v as a non-existent value and a empty array a[]	0
Test with v as an existent value and an empty array a[]	0
Test with v as an existent value and a non-empty array a[] where v exists only once	1
Test with v as an existent value and a non-empty array a[] where v exists multiple times	number of occurrences of v in a[]

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
Test with v as a non-existent value and a non-empty array a[]	0
Test with v as a non-existent value and an empty array a[]	0
Test with v as an existent value and an array a[] of length 0	0
Test with v as an existent value and an array a[] of length 1, where v does not exist	0
Test with v as an existent value and an array a[] of length 1, where v exists	1
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the beginning of the array	number of occurrences of v in a[]
Test with v as an existent value and an array a[] of length greater than 1, where v exists at the end of the array	number of occurrences of v in a[]
Test with v as an existent value and an array a[] of length greater than 1, where v exists in the middle of the array	number of occurrences of v in a[]

Problem 3 :

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
v=6, a=[2, 4, 6, 8, 10]	2
v=2, a=[2, 4, 6, 8, 10]	0
v=10, a=[2, 4, 6, 8, 10]	4
v=5, a=[2, 4, 6, 8, 10]	-1
v=9, a=[2, 4, 6, 8, 10]	-1

Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
v=1, a=[1]	0
v=9, a=[9]	0
v=5, a=[]	-1
v=4, a=[4, 7, 9]	0 (smallest element in the array)
v=5, a=[1, 2, 5]	2 (largest element in the array)

Problem 4 :

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

Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
Valid input: a=3, b=3, c=3	EQUILATERAL
Valid input: a=3, b=3, c=5	ISOSCELES
Valid input: a=3, b=4, c=5	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=6	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

Problem 5 :

Equivalence Partitioning:

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

Boundary Value Analysis:

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

Problem 6 :

(a) Equivalence Classes:

Tester Action and Input Data	Expected Outcome
a = -1, b = 5, c = 3	Invalid input
a = 1, b = 1, c = 1	Equilateral triangle
a = 5, b = 5, c = 5	Isosceles triangle
a = 3, b = 4, c = 5	Scalene right angled triangle
a = 3, b = 5, c = 4	Scalene right angled triangle
a = 5, b = 3, c = 4	Scalene right angled triangle
a = 3, b = 4, c = 6	Not a triangle

b) Test cases:

Invalid inputs: a = 0, b = 0, c = 0, a + b = c, b + c = a, c + a = b
Invalid inputs: a = -1, b = 1, c = 1, a + b = c
Equilateral triangles: a = b = c = 1, a = b = c = 100
Isosceles triangles: a = b = 10, c = 5; a = c = 10, b = 3; b = c = 10, a = 6
Scalene triangles: a = 4, b = 5, c = 6; a = 10, b = 11, c = 13
Right angled triangle: a = 3, b = 4, c = 5; a = 5, b = 12, c = 13
Non-triangle: a = 1, b = 2, c = 3
Non-positive input: a = -1, b = -2, c = -3

c) Boundary condition A + B > C:
a = Integer.MAX_VALUE, b = Integer.MAX_VALUE, c = 1 a =
Double.MAX_VALUE, b = Double.MAX_VALUE, c = Double.MAX_VALUE

d) Boundary condition A = C:
a = Integer.MAX_VALUE, b = 2, c = Integer.MAX_VALUE a =
Double.MAX_VALUE, b = 2.5, c = Double.MAX_VALUE

e) Boundary condition A = B = C:
a = Integer.MAX_VALUE, b = Integer.MAX_VALUE, c = Integer.MAX_VALUE a =
Double.MAX_VALUE, b = Double.MAX_VALUE, c = Double.MAX_VALUE

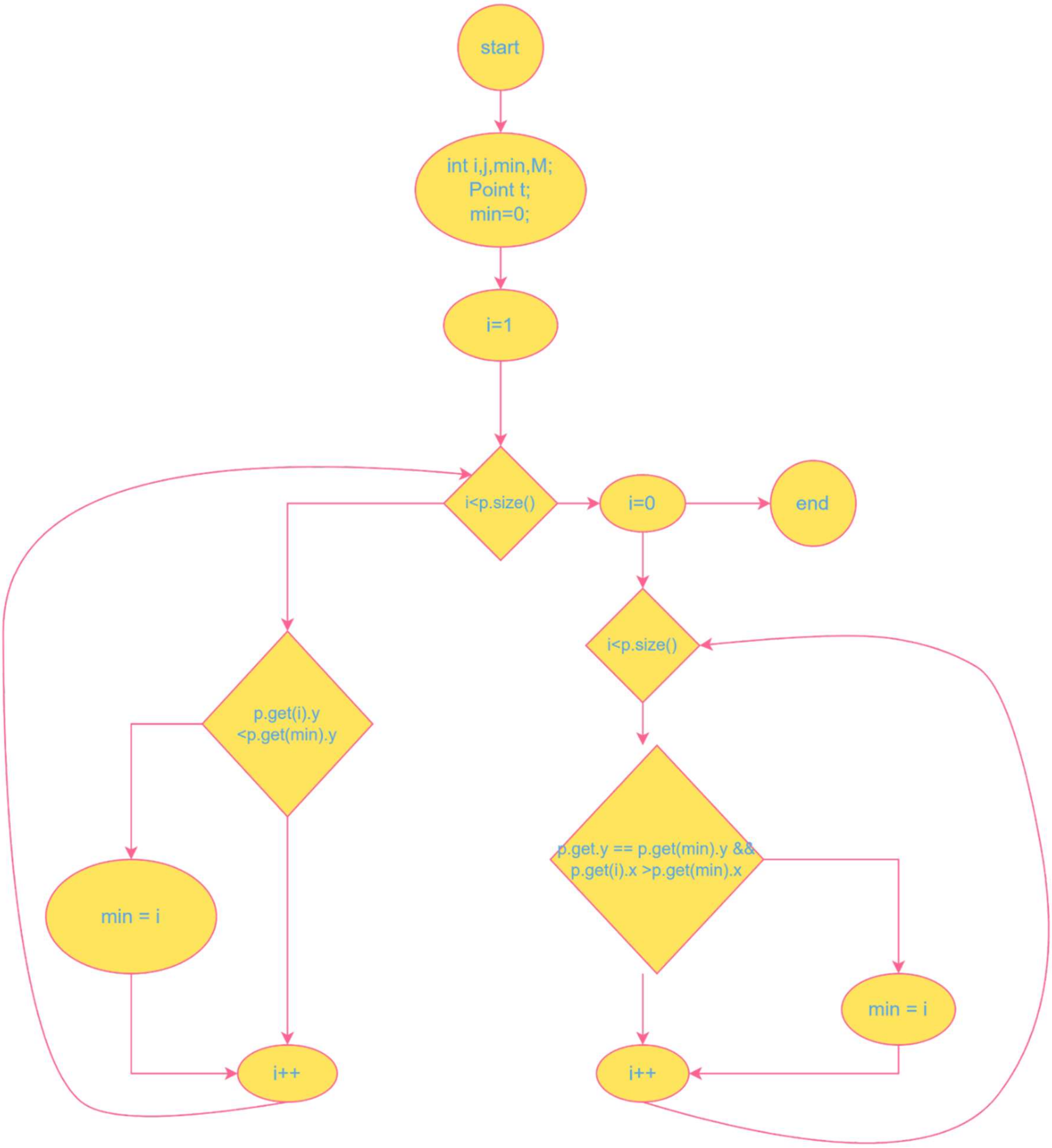
f) Boundary condition A^2 + B^2 = C^2:
a = Integer.MAX_VALUE, b = Integer.MAX_VALUE, c = Integer.MAX_VALUE a =
Double.MAX_VALUE, b = Double.MAX_VALUE, c =
Math.sqrt(Math.pow(Double.MAX_VALUE, 2) + Math.pow(Double.MAX_VALUE, 2))

g) Non-triangle:
a = 1, b = 2, c = 4 a = 2, b = 4, c = 8

h) Non-positive input:
a = -1, b = -2, c = -3 a = 0, b = 1, c = 2

Section B

1. Control Flow Graph (CFG):



2. Test sets for each coverage criterion:

a. Statement Coverage:

- Test 1: p = {new Point(0, 0), new Point(1, 1)}
- Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}

b. Branch Coverage:

- Test 1: p = {new Point(0, 0), new Point(1, 1)}
- Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}
- Test 3: p = {new Point(0, 0), new Point(1, 0), new Point(1, 1)}

c. Basic Condition Coverage:

- Test 1: p = {new Point(0, 0), new Point(1, 1)}
- Test 2: p = {new Point(0, 0), new Point(1, 0), new Point(2, 0)}
- Test 3: p = {new Point(0, 0), new Point(1, 0), new Point(1, 1)}
- Test 4: p = {new Point(0, 0), new Point(1, 0), new Point(0, 1)}
- Test 5: p = {new Point(0, 0), new Point(0, 1), new Point(1, 1)}