Name : Jainam Bhavsar
Student ID : 202001233
Course : IT314 Software Engineering
Date : 13/04/23
Lab : 7

Section A

1. **Consider a program for determining the previous date. Its input is triple of day, month and year with the following ranges 1 <= month <= 12, 1 <= day <= 31, 1900 <= year <= 2015.The possible output dates would be previous date or invalid date. Design the equivalence class test cases?**
    a. **Equivalence Class 1:** Valid dates (1, 1, 1900) to (31, 12, 2015) such that day, month and year are in the given ranges. ex: (15, 10, 2022), (1, 1, 2015), (31, 3, 2000) etc.
    b. **Equivalence Class 2:** Invalid dates (1, 1, 1900) to (31, 12, 2015) such that day, month and year are in the given ranges but the input is invalid. ex: (29, 2, 2001), (31, 4, 2010), (30, 2, 2000) etc.
    c. **Equivalence Class 3:** Invalid range (1, 1, 1900) to (31, 12, 2015) such that day, month and year are not in the given ranges. ex: (32, 2, 2022), (13,14,2003), (11,11,2020)
    d. **Equivalence Class 4:** Invalid input (1, 1, 1900) to (31, 12, 2015) such that day, month and year are in the given ranges but the input is invalid. ex: (2.2, 2, 2022), (a, 4, 2010), (30, 2, 2000) etc.

Tester Action and Input Data Expected Outcome

Valid dates:	**Input**	**Output**
Test Case 1	(11,10,2010)	(10,10,2010)
Test Case 2	(1,1,1900)	(31,12,1899)
Invalid dates:		

Test Case 1	(10,10,2010)	(9,10,2010)
Test Case 2	(1,1,1900)	(31,12,1899)
Out of range:		
Test Case 1	(32,2,2022)	Invalid date
Test Case 2	(13,14,2003)	Invalid date
Test Case 3	(11,11,2020)	Invalid date
Boundary Value Analysis:		
Test Case 1	(1,1,1900) Valid First possible date	(31,12,1899) not in range
Test Case 2	(31,12,2015) Valid Last possible date	(30,12,2015)
Test Case 3	(31,1,1899) one day before first possible date	Invalid input
Test Case 4	(1,1,2016) one day after last possible date	Invalid input
Test Case 5	(29,2,2000) Valid leap year date	(28,2,2000)
Test Case 6	(29,2,1900) Invalid leap year date	Invalid input
Test Case 7	(1,3,2000) valid date after leap year date	(29,2,2000)
Test Case 8	(1,3,2019)	(28,2,2019)

	valid date after non leap year date	
Test Case 9	(1,3,2000) valid first day of month	(31,12,1999)
Test Case 10	(1,1,2000) valid first day of year	(31,12,1999)

- Based on these boundary test cases, we can design the following test cases:

Equivalence Class Testing

Tester Action and Input Data	Expected Outcome
Valid partition	
(1,1,1900)	(31,12,1899)
(31,12,2015)	(30,12,2015)
Invalid partition	
(32,2,2022)	Invalid date
(13,14,2003)	Invalid date
(11,11,2020)	Invalid date
(29,2,2001)	Invalid date
(31,4,2010)	Invalid date
(2.2, 2, 2022)	Invalid date
(a, 4, 2010)	Invalid date
(30, 2, 2000)	Invalid date
(31,1,1899)	Invalid date

Boundary Value Analysis

Tester Action and Input Data	Expected Outcome
(1,1,1900)	(31,12,1899)
(31,12,2015)	(30,12,2015)
(31,1,1899)	Invalid input
(1,1,2016)	Invalid input
(29,2,2000)	(28,2,2000)
(29,2,1900)	Invalid input
(1,3,2000)	(29,2,2000)
(1,3,2019)	(28,2,2019)
(1,3,2000)	(31,12,1999)
(1,1,2000)	(31,12,1999)

---

2. Write a set of test cases (i.e., test suite) – specific set of data – to properly test the UnitTesting. Your test suite should include both correct and incorrect inputs.

A. Enlist which set of test cases have been identified using Equivalence Partitioning and Boundary Value Analysis separately.

P1:

Equivalence Partitioning	Expected Outcome
a = [1, 2, 3, 4], v = 2	1
a = [5, 6, 7, 8], v = 10	-1
a = [1, 1, 2, 3], v = 1	0
a = null, v = 5	Error Message

Boundary Analysis	Expected Outcome
Minimum array length: a = [], v = 7	-1
Maximum array length: a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], v = 3	2
Minimum value of v: a = [5, 6, 7], v = 5	0
Maximum value of v: a = [1, 2, 3], v = 3	2

P2:

Equivalence Partitioning	Expected Outcome
Invalid input: v is not an integer	Error Message
Empty array: a = []	0
Single item array: a = [v], v = a[0]	1
Multiple item array with v appearing:	
v appears once	1
v appears multiple times	Count > 1
Multiple item array with v not appearing	0

Boundary Analysis	Expected Outcome
Minimum input values: v = a[0] = 1	Count > 0
Maximum input values: v = a[9999] = 10000	Count > 0
One occurrence of v: a = [1, 2, 3, ..., 9999, v-1, 10000]	1
All occurrences of v: a = [v, v, v, ..., v, v]	10000
No occurrences of v: a = [1, 2, 3, ...,	0

9999]	

P3:

Equivalence Partitioning	Expected Outcome
v = 5, a = [1, 3, 5, 7, 9]	2
v = 1, a = [1, 3, 5, 7, 9]	0
v = 9, a = [1, 3, 5, 7, 9]	4
v = 2, a = [1, 3, 5, 7, 9]	-1
v = 2, a = [1, 3, 5, 7, 9]	-1
v = 6, a = []	-1

Boundary Value Analysis	Expected Outcome
v = 5, a = [5, 6, 7]	0
v = 6, a = [5, 6, 7]	1
v = 7, a = [5, 6, 7]	2
v = 5, a = [1, 5, 6, 7, 9]	1
v = 6, a = [1, 5, 6, 7, 9]	2
v = 7, a = [1, 5, 6, 7, 9]	3
v = 9, a = [1, 5, 6, 7, 9]	4
v = 1, a = [1]	0
v = 5, a = [5]	0
v = 5, a = []	-1
v = 2, a = [1, 3, 5, 7, 9]	-1
v = 6, a = [1, 3, 5, 7, 9]	-1
v = 10, a = [1, 3, 5, 7, 9]	-1
v = 1, a = [2, 3, 4, 5, 6]	-1

v = 4, a = [2, 3, 4, 5, 6]	-1
v = 7, a = [2, 3, 4, 5, 6]	-1

P4:

Equivalence Partitioning	Expected Outcome
a=b=c, where a, b, c are positive integers	Equilateral
a=b<c, where a, b, and c are positive integers	Isosceles
a=b=c=0	Invalid
a<b+c, b<a+c, c<a+b, where a, b, c are positive integers	Scalene
a=b>0, c=0	Invalid
a>b+c	Invalid

Boundary Value Analysis	Expected Outcome
a=1, b=1, c=1	Equilateral
a=1, b=2, c=2	Isosceles
a=0, b=0, c=0	Invalid
a=2147483647, b=2147483647, c=2147483647	Equilateral
a=2147483646, b=2147483647, c=2147483647	Isosceles
a=1, b=1, c=2^{31}-1	Scalene
a=0, b=1, c=1	Invalid

P5:

Equivalence Partitioning	Expected Outcome
s1 is empty, s2 is non-empty string	True
s1 is non-empty string, s2 is empty	Flase
s1 is a prefix of s2	True
s1 is not a prefix of s2	False
s1 has same characters as s2, but not a prefix	False

Boundary Value Analysis	Expected Outcome
s1 = "a", s2 = "ab"	True
s1 = "ab", s2 = "a"	Flase
s1 = "a", s2 = "a"	True
s1 = "a", s2 = "A"	Flase
s1 = "abcdefghijklmnopqrstuvwxyz", s2 = "abcdefghijklmnopqrstuvwxyz"	True
s1 = "abcdefghijklmnopqrstuvwxyz", s2 = "abcdefghijklmno"	True
s1 = "", s2 = ""	True

B. 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 whether the identified expected outcome (mentioned by you) is correct or not.

Modified Java Codes for the given programs:

```
package tests;
public class UnitTesting {
	public int linearSearch(int v, int a[]) // p1
	{
	int i = 0;
	while (i < a.length)
	{
	if (a[i] == v)
	return(i);
	i++;
	}
	return (-1);
	}
	public int countItem(int v, int a[]) //p2
	{
	int count = 0;
	for (int i = 0; i < a.length; i++)
	{
	if (a[i] == v)
	count++;
	}
	return (count);
	}
	public int binarySearch(int v, int a[]) //p3
	{
	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);
        }
        final int EQUILATERAL = 0;
        final int ISOSCELES = 1;
        final int SCALENE = 2;
        final int INVALID = 3;
        public int triangle(int a, int b, int c) //p4
        {
        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);
        }
        public boolean prefix(String s1, String s2) //p5
        {
        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;
        }
}
```

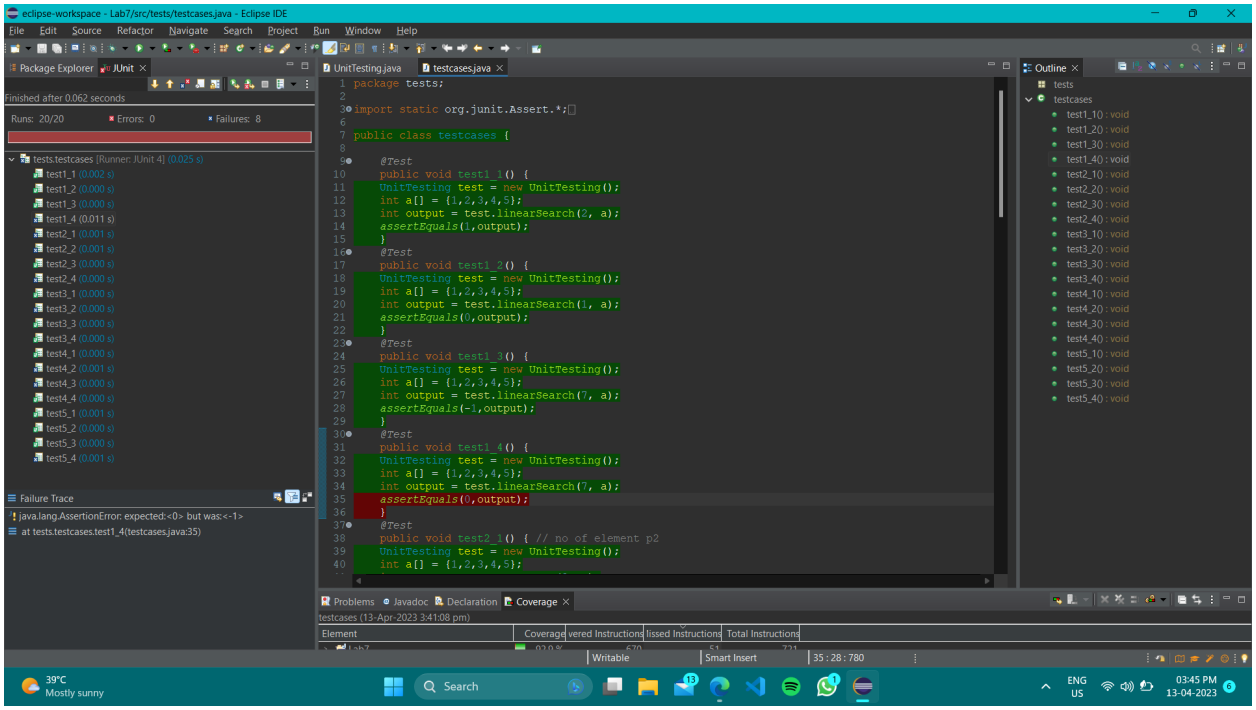
Testing Cases with Coverage:

```
package tests;

import static org.junit.Assert.*;

import org.junit.Test;

public class testcases {
        @Test
        public void test1_1() {
        UnitTesting test = new UnitTesting();
        int a[] = {1,2,3,4,5};
        int output = test.linearSearch(2, a);
        assertEquals(1,output);
        }
```

```
    @Test
    public void test1_2() {
    UnitTesting test = new UnitTesting();
    int a[] = {1,2,3,4,5};
    int output = test.linearSearch(1, a);
    assertEquals(0,output);
    }

    @Test
    public void test1_3() {
    UnitTesting test = new UnitTesting();
    int a[] = {1,2,3,4,5};
    int output = test.linearSearch(7, a);
    assertEquals(-1,output);
    }

    @Test
    public void test1_4() {
    UnitTesting test = new UnitTesting();
    int a[] = {1,2,3,4,5};
    int output = test.linearSearch(7, a);
    assertEquals(0,output);
    }

    @Test
    public void test2_1() { // no of element p2
    UnitTesting test = new UnitTesting();
    int a[] = {1,2,3,4,5};
    int output = test.countItem(2, a);
    assertEquals(2,output);
    }

    @Test
    public void test2_2() { //no of element p2
    UnitTesting test = new UnitTesting();
    int a[] = {1,2,3,4,5};
    int output = test.countItem(4, a);
    assertEquals(2,output);
    }

    @Test
    public void test2_3() { //no of element p2
    UnitTesting test = new UnitTesting();
    int a[] = {1,2,3,4,5};
    int output = test.countItem(6, a);
    assertEquals(0,output);
    }
```

```
	@Test
	public void test2_4() { //no of element p2
	UnitTesting test = new UnitTesting();
	int a[] = {1,2,3,4,5};
	int output = test.countItem(6, a);
	assertEquals(-1,output);
	}

	@Test
	public void test3_1() { //binary search p3
	UnitTesting test = new UnitTesting();
	int a[] = {1,2,3,4,5};
	int output = test.binarySearch(2, a);
	assertEquals(1,output);
	}

	@Test
	public void test3_2() { //binary search p3
	UnitTesting test = new UnitTesting();
	int a[] = {1,2,3,4,5};
	int output = test.binarySearch(3, a);
	assertEquals(3,output);
	}
	@Test
	public void test3_3() { //binary search p3
	UnitTesting test = new UnitTesting();
	int a[] = {1,2,3,4,5};
	int output = test.binarySearch(8, a);
	assertEquals(-1,output);
	}

	@Test
	public void test3_4() { //binary search p3
	UnitTesting test = new UnitTesting();
	int a[] = {1,2,3,4,5};
	int output = test.binarySearch(8, a);
	assertEquals(-1,output);
	}

	@Test
	public void test4_1() {
	UnitTesting test = new UnitTesting();
	int output = test.triangle(8,8,8);
	assertEquals(0,output);
	}

	@Test
	public void test4_2() {
	UnitTesting test = new UnitTesting();
```

```
        int output = test.triangle(8,8,10);
        assertEquals(2,output);
        }

        @Test
        public void test4_3() {
        UnitTesting test = new UnitTesting();
        int output = test.triangle(0,0,0);
        assertEquals(1,output);
        }

        @Test
        public void test4_4() {
        UnitTesting test = new UnitTesting();
        int output = test.triangle(0,0,0);
        assertEquals(3,output);
        }

        @Test
        public void test5_1() {
        UnitTesting test = new UnitTesting();
        boolean output = test.prefix("","nonEmpty");
        assertEquals(true,output);
        }

        @Test
        public void test5_2() { // example of s1 is prefix of s2
        UnitTesting test = new UnitTesting();
        boolean output = test.prefix("hello","hello world");
        assertEquals(true,output);
        }

        @Test
        public void test5_3() { // example of s1 is not prefix of s2
        UnitTesting test = new UnitTesting();
        boolean output = test.prefix("hello","world hello");
        assertEquals(false,output);
        }

        @Test
        public void test5_4() { // example of s1 is not prefix of s2
        UnitTesting test = new UnitTesting();
        boolean output = test.prefix("hello","world hello");
        assertEquals(true,output);
        }
}
```



Program No.	Valid Test Cases	Invalid Test Cases
1	1, 2, 3	4
2	3	1, 2, 4
3	1, 3, 4	2
4	1, 4	2, 3
5	1, 2, 3	4

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.

- **Equivalence Class 1:** Invalid inputs (negative or zero values)
- **Equivalence Class 2:** Non-triangle (sum of the two shorter sides is not greater than the longest side)
- **Equivalence Class 3:** Scalene triangle (no sides are equal)
- **Equivalence Class 4:** Isosceles triangle (two sides are equal)
- **Equivalence Class 5:** Equilateral triangle (all sides are equal)
- **Equivalence Class 6:** Right-angled triangle (satisfies the Pythagorean theorem)

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

- For Equivalence Class 1: -1, 0, 2 (invalid input)
- For Equivalence Class 2: 1, 3, 6 (Non-triangle)
- For Equivalence Class 3: 4, 5, 6 (Scalene triangle)
- For Equivalence Class 4: 6, 6, 10 (Isosceles triangle)
- For Equivalence Class 5: 4, 4, 4 (Equilateral triangle)
- For Equivalence Class 6: 3, 4, 5 (Right-angled triangle)

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

Test case 1: A=5, B=6, C=10	(c=a+b-1)	(valid scalene triangle)
Test case 2: A=5, B=6, C=11	(c=a+b)	(invalid input)
Test case 3: A=5, B=6, C=12	(c=a+b+1)	(invalid input)

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

Test case 1: A=5, B=6, C=5	(c=a)	(valid isosceles triangle)
Test case 2: A=5, B=11, C=5	(c=a)	(invalid input as b>a+c)
Test case 3: A=5, B=5, C=5	(c=a)	(equilateral triangle)

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

Test case 1: A=5, B=5, C=5 (a=b=c) (valid equilateral triangle)

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

Test case 1: A=3, B=4, C=5 (valid right-angle triangle)
Test case 2: A=IntMax, B=IntMax, C=IntMax (overflow error)

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

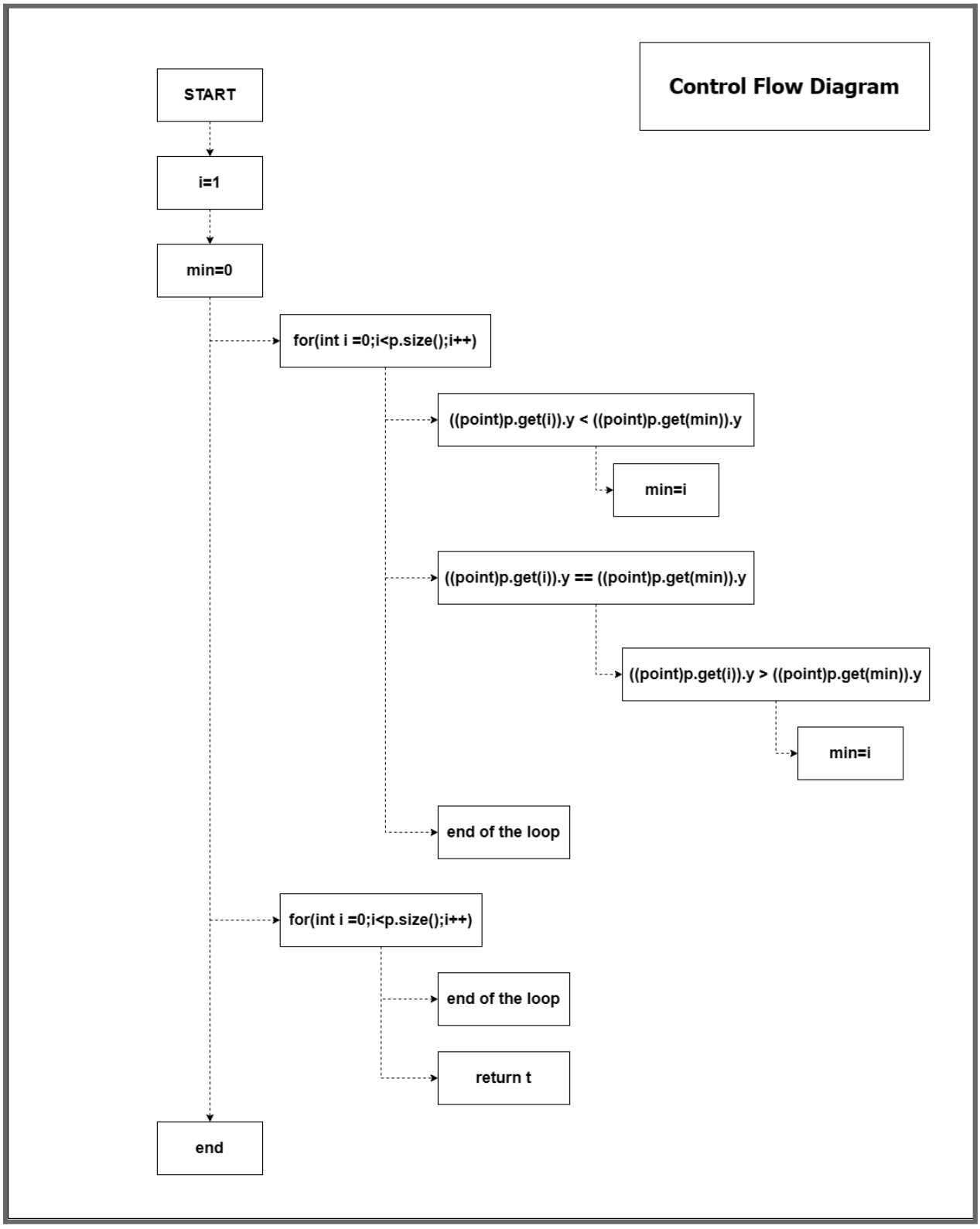
Test case 1: A=5, B=6, C=11 (invalid input as a+b=c)
Test case 2: A=5, B=6, C=12 (invalid input as a+b+1=c)

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

Test case 1: A=0, B=6, C=11 (invalid input as a=0)
Test case 2: A=-5, B=6, C=11 (invalid input as a<0)

Section B

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.

- To achieve statement coverage, we need to make sure that every statement in the code is executed at least once.

Test case set:

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

b. Branch Coverage.

- To achieve branch coverage, we need to make sure that every possible branch in the code is taken at least once.

Test case set:

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

7	vector with three or more points with the same y component, and all of them have the same x component

c. Basic Condition Coverage.

- 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 case set:

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

---

***