LAB - 7

Name: Dev Kabra
ID: 202001105

Section - A

Test Case ID	Day	Month	Year	Expected Output
1	1	6	2000	31-5-2000
2	2	6	2015	1-6-2015
3	2	6	2016	Invalid
4	1	1	1900	31-12-1899
5	31	12	1899	Invalid
6	31	12	1900	30-12-1900
7	29	2	2012	28-2-2012
8	1	3	2012	29-2-2012
9	29	2	2011	Invalid
10	30	2	2020	Invalid

Equivalence classes:

For day:

Partition ID	Range	Status
E1	Between 1 and 28	Valid
E2	Less than 1	Invalid
E3	Greater than 31	Invalid
E4	Equals 30	Valid
E5	Equals 29	Valid for leap year
E6	Equals 31	Valid

For month:

Partition ID	Range	Status
E7	Between 1 and 12	Valid
E8	Less than 1	Invalid

E9	**Greater than 12**	**Invalid**

For year:

Partition ID	Range	Status
E10	**Between 1900 and 2015**	**Valid**
E11	**Less than 1**	**Invalid**
E12	**Greater than 2015**	**Invalid**

Equivalence Partitioning : EP

Boundary Value Analysis : BVA

PROGRAM	Type	Tester Action and Input Data	Expected Outcome
P1	EP	A=[1,2,3,4],v= 4	3
	EP	A=[1,2,3],v=4	-1

	BVA	A = [], v = 2	-1
P2	EP	A=[1,2,3,4,4],V=4	2
	EP	A = [1,2,3,3], V=2	1
	EP	A=[1,2,3], V=5	0
	BVA	A=[], v=2	0
P3	EP	A=[1,2,3,4,5], V=3	3
	EP	A=[2,3,4], V=1	-1
	EP	A=[1,2,3], V=5	-1
	EP	A=[1,2,3,5], V=4	-1

	BVA	A=[1,2,3], V=1	0
	BVA	A=[1,2,3], V=3	2
	BVA	A=[1,2,3], V=2	1
P4	EP	a = 5, b = 5, c = 5	0
	EP	a = 6, b = 6, c = 4	1
	EP	a = 3, b = 4, c = 5	2
	EP	a = 0, b = 0, c = 0	3
	EP	a=-1, b=-2, c=-3	3
	BVA	a = 1, b = 1, c = 1	0
	BVA	a = 2, b = 3, c = 4	2
	BVA	a = 1, b = 2, c = 3	3

	BVA	a = 3, b = 4, c = 7	3
	BVA	a = 6, b = 2, c = 5	2
	BVA	a = 8, b = 2, c = 5	3
P5	EP	s1 = "hello", s2 = "hello world"	true
	EP	s1 = "apple", s2 = "apples"	true
	EP	s1 = "", s2 = "hello"	false
	EP	s1 = null, s2 = "hello world"	false
	BVA	s1 = "", s2 = ""	true
	BVA	s1 = "a", s2 = "a"	true
	BVA	s1 = "abcdefghijklmnopqrstuvwxyz", s2 = "abcdefghijklmnopqrstuvwxyz"	true
	BVA	s1 = null, s2 = null	false

	BVA	s1 = "a", s2 = "b"	false

All the functions from P1 to P5:

```
package file1;
public class unittesting {

	public static int linearSearch(int v, int[] a) {
	    int i = 0;
	    while (i < a.length) {
	        if (a[i] == v) {
	            return i;
	        }
	        i++;
	    }
	    return -1;
	}
	public static int countItem(int v, int[] a) {
	    int count = 0;
	    for (int i = 0; i < a.length; i++) {
	        if (a[i] == v) {
	            count++;
	        }
	    }
	    return count;
	}
	public static int binarySearch(int v, int[] a) {
	    int lo = 0;
	    int hi = a.length - 1;
	    while (lo <= hi) {
	        int mid = (lo + hi) / 2;
	        if (v == a[mid]) {
	            return mid;
	        } else if (v < a[mid]) {
	            hi = mid - 1;
	        } else {
	            lo = mid + 1;
	        }
	    }
	    return -1;
	}
```

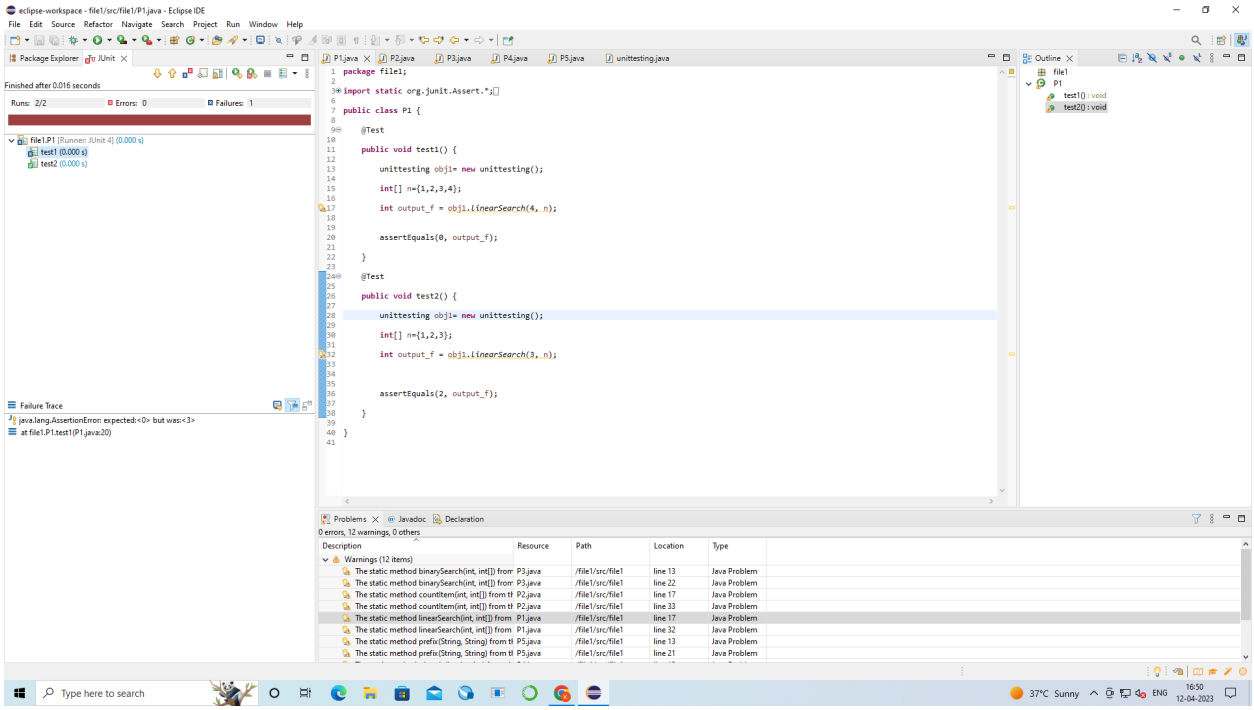
```
public static int triangle(int a,int b,int c) {

        if (a >= b + c || b >= a + c || c >= a + b) {
            return 3;
        }
        if (a == b && b == c) {
            return 0;
        }
        if (a == b || a == c || b == c) {
            return 1;
        }
        return 2;
}
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;
}
```

Test Codes and Outputs:

P1.

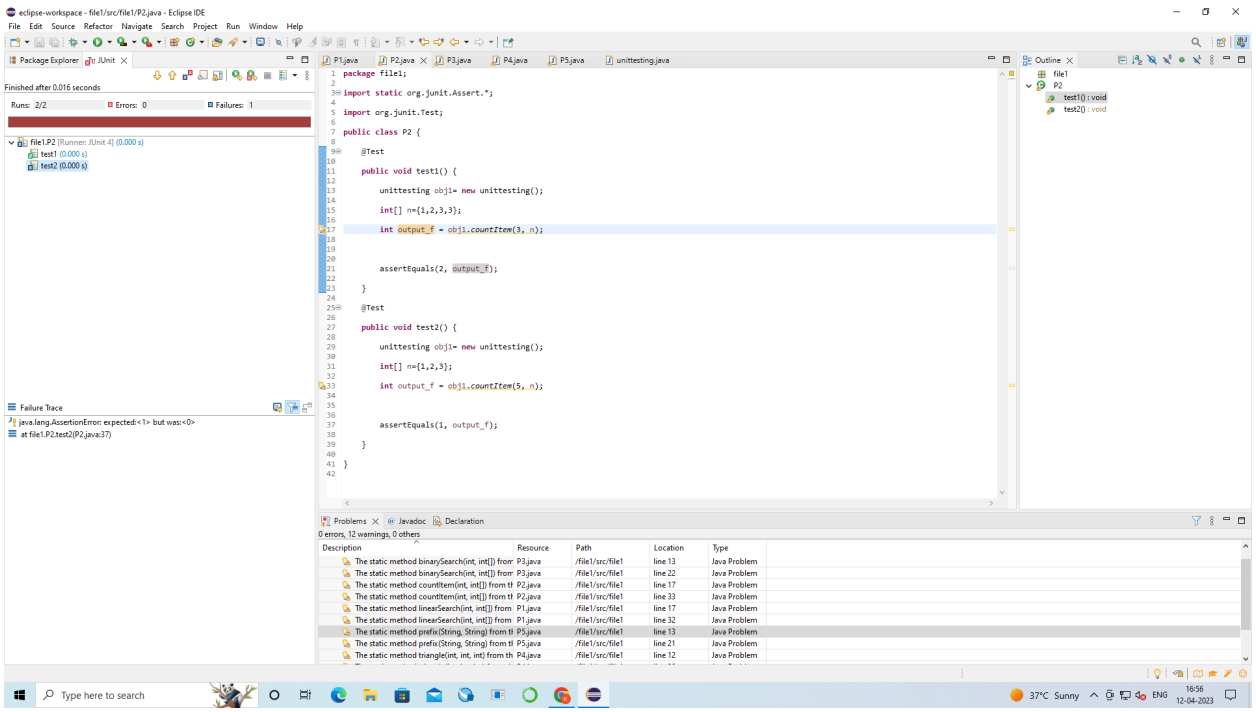
```
package file1;
import static org.junit.Assert.*;
import org.junit.Test;

public class P1 {

        @Test

        public void test1() {

                unittesting obj1= new unittesting();

                int[] n={1,2,3,4};

                int output_f = obj1.linearSearch(4, n);
```

```
        assertEquals(0, output_f);
    }
    @Test
    public void test2() {
        unittesting obj1= new unittesting();
        int[] n={1,2,3};
        int output_f = obj1.linearSearch(3, n);

        assertEquals(2, output_f);
    }
}
```



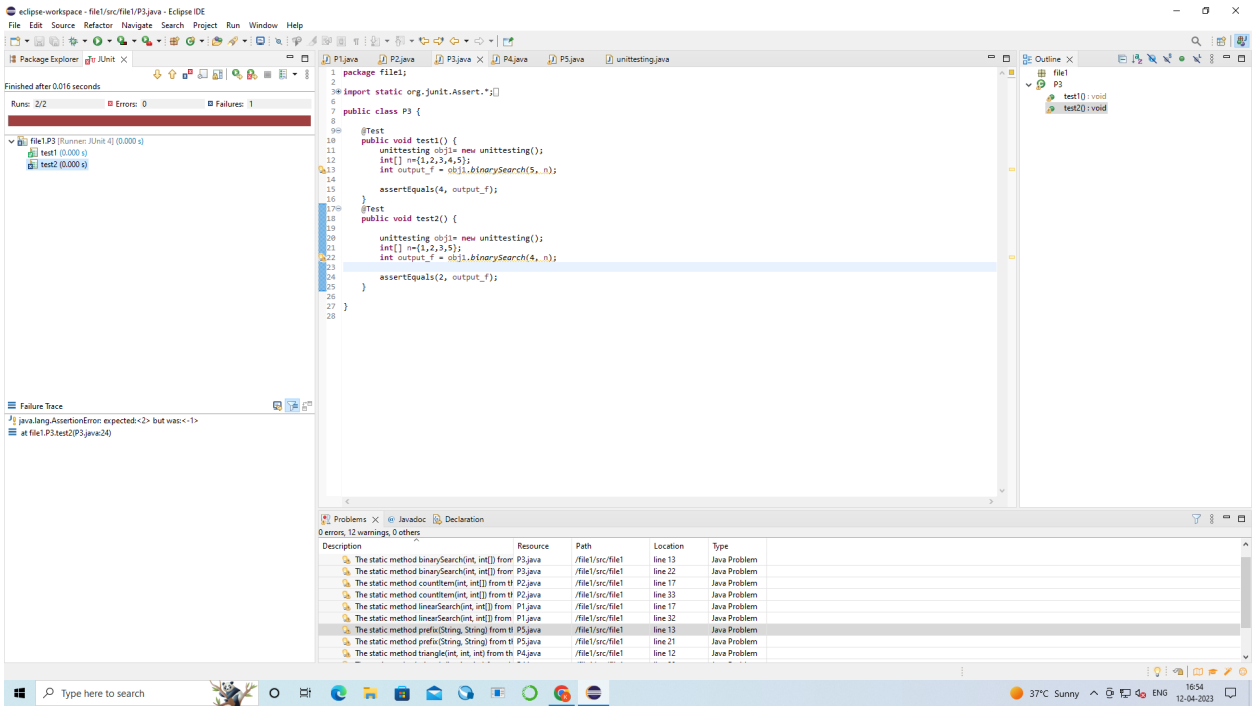
P2.

```
package file1;
import static org.junit.Assert.*;
import org.junit.Test;
public class P2 {
	@Test
	public void test1() {
		unittesting obj1= new unittesting();
		int[] n={1,2,3,3};
		int output_f = obj1.countItem(3, n);

		assertEquals(2, output_f);
	}
	@Test
	public void test2() {
		unittesting obj1= new unittesting();
		int[] n={1,2,3};
		int output_f = obj1.countItem(5, n);

		assertEquals(1, output_f);
	}
}
```



P3.

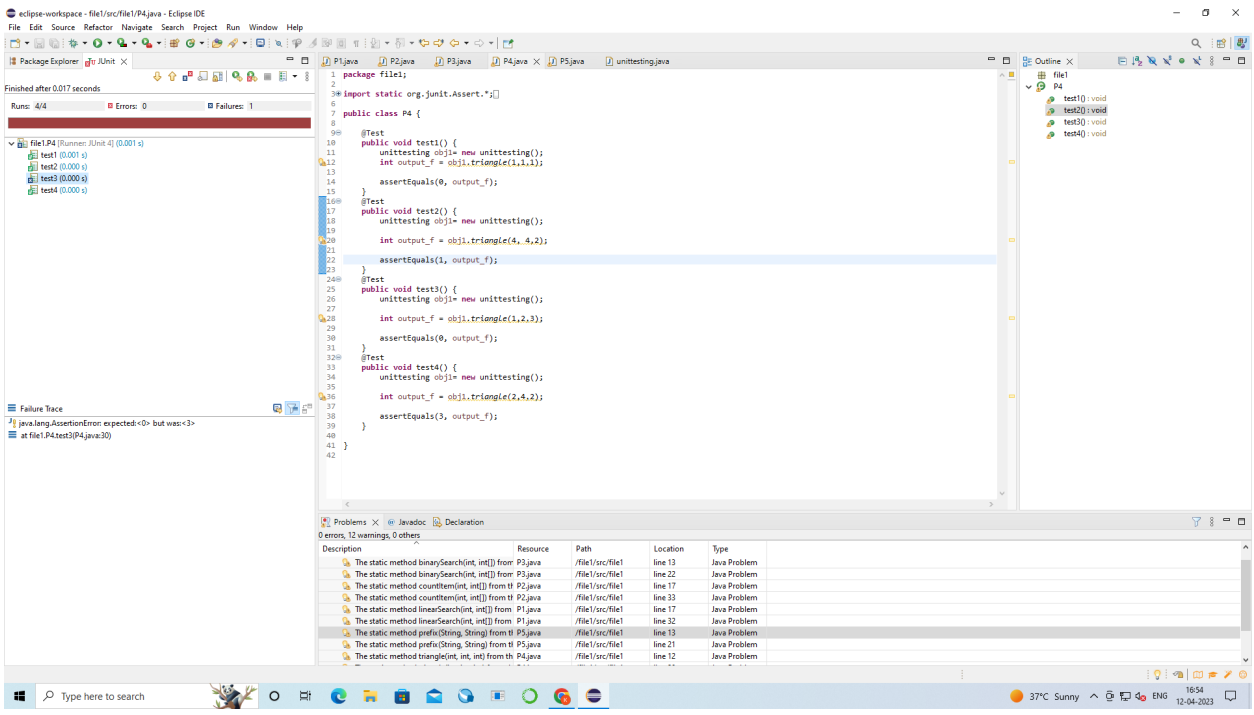
```
package file1;

import static org.junit.Assert.*;

import org.junit.Test;

public class P3 {

	@Test
	public void test1() {
		unittesting obj1= new unittesting();
		int[] n={1,2,3,4,5};
		int output_f = obj1.binarySearch(5, n);

		assertEquals(4, output_f);
	}
	@Test
	public void test2() {

		unittesting obj1= new unittesting();
		int[] n={1,2,3,5};
		int output_f = obj1.binarySearch(4, n);

		assertEquals(2, output_f);
	}

}
```



P4.

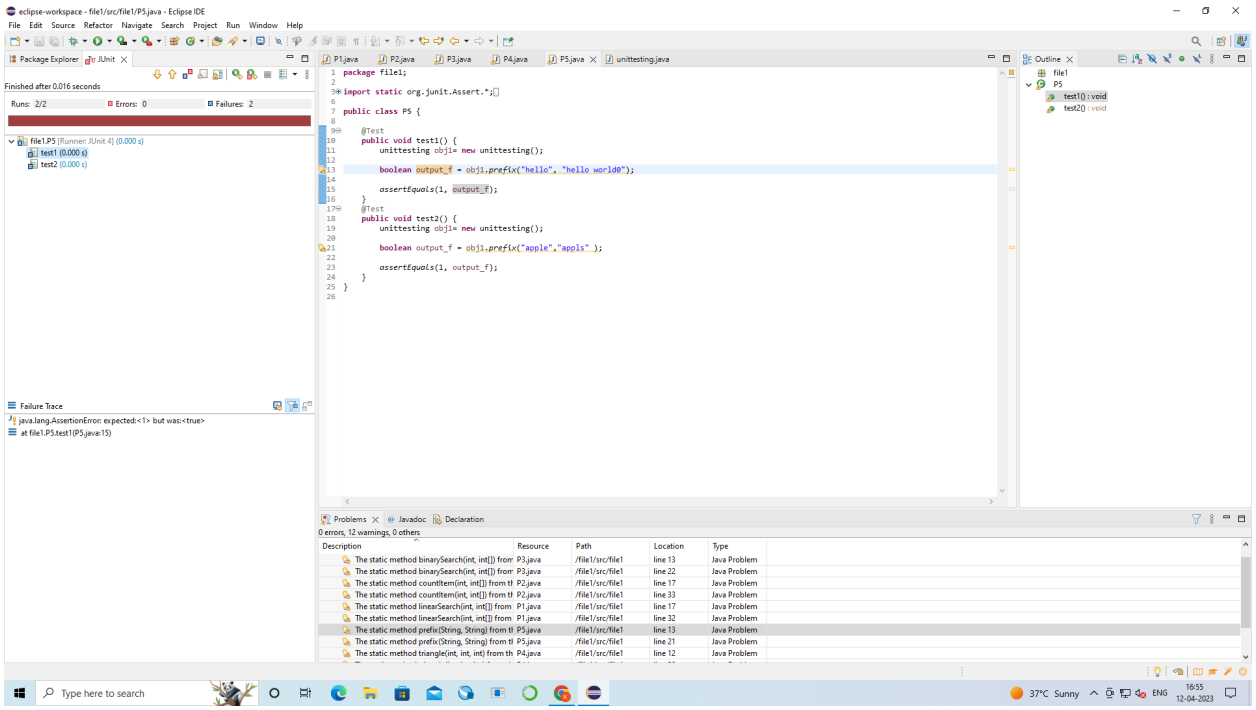
```
package file1;

import static org.junit.Assert.*;

import org.junit.Test;

public class P4 {

	@Test
	public void test1() {
		unittesting obj1= new unittesting();
		int output_f = obj1.triangle(1,1,1);

		assertEquals(0, output_f);
	}
	@Test
	public void test2() {
		unittesting obj1= new unittesting();

		int output_f = obj1.triangle(4, 4,2);
```

```
		assertEquals(1, output_f);
	}
	@Test
	public void test3() {
		unittesting obj1= new unittesting();

		int output_f = obj1.triangle(1,2,3);

		assertEquals(0, output_f);
	}
	@Test
	public void test4() {
		unittesting obj1= new unittesting();

		int output_f = obj1.triangle(2,4,2);

		assertEquals(3, output_f);
	}

}
```



P5.

```
package file1;

import static org.junit.Assert.*;

import org.junit.Test;

public class P5 {

	@Test
	public void test1() {
		unittesting obj1= new unittesting();

		boolean output_f = obj1.prefix("hello", "hello world0");

		assertEquals(1, output_f);
	}
	@Test
	public void test2() {
		unittesting obj1= new unittesting();

		boolean output_f = obj1.prefix("apple","appls" );

		assertEquals(1, output_f);
	}
}
```



P6.

(a) Equivalence Classes:
Isosceles, equilateral, scalene, invalid

(b)
Equivalence Class 1: EQUILATERAL Triangle
Input: a = 5, b = 5, c = 5
Expected Output: EQUILATERAL

Equivalence Class 2: ISOSCELES Triangle
Input: a = 6, b = 6, c = 4
Expected Output: ISOSCELES

Equivalence Class 3: SCALENE Triangle
Input: a = 3, b = 4, c = 5
Expected Output: SCALENE

Equivalence Class 4: INVALID Triangle
Input: a = 0, b = 0, c = 0
Expected Output: INVALID

(c)
Boundary Test Case 1: a + b = c - 1
Input: a = 2, b = 3, c = 4
Expected Output: INVALID

Boundary Test Case 2: a + b = c + 1
Input: a = 3, b = 4, c = 7
Expected Output: INVALID

Boundary Test Case 3: a = b + c - 1
Input: a = 6, b = 2, c = 5
Expected Output: INVALID

Boundary Test Case 4: a = b + c
Input: a = 7, b = 2, c = 5
Expected Output: INVALID

Boundary Test Case 5: a = b + c + 1
Input: a = 8, b = 2, c = 5
Expected Output: INVALID

(d)
Boundary Test Case 1: a = c - 1, b<a+c
Input: a = 2, b = 1, c = 4
Expected Output: scalene

Boundary Test Case 2: a = c + 1, b < a+c
Input: a = 8, b = 4, c = 7
Expected Output: scalene

(e)
Boundary Test Case 1: a = b = c +1
Input: a = 2, b = 2, c = 1
Expected Output: isosceles

Boundary Test Case 2: a = b = c - 1
Input: a = 6, b = 6, c = 7
Expected Output: isosceles

Boundary Test Case 3: a + 1 = b = c - 1
Input: a = 2, b = 3, c = 4
Expected Output: scalene

(f)
Input: a = 3, b = 4, c = 5
Expected Output: scalene (valid right-angle triangle)

Input: a = 5, b = 4, c = 3
Expected Output: scalene (valid right-angle triangle)

Input: a = 6, b = 8, c = 10
Expected Output: scalene (valid right-angle triangle)

Input: a = 1, b = 1, c = 2
Expected Output: Invalid (invalid triangle)

Input: a = 3, b = 3, c = 3
Expected Output: equilateral (equilateral triangle)

Input: a = 4, b = 4, c = 7

Expected Output: isosceles (isosceles triangle)

(g)
Input: a = 1, b = 1, c = 1
Expected Output: EQUILATERAL

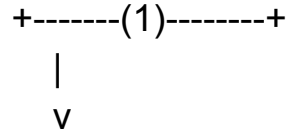
Input: a = 0, b = 0, c = 0
Expected Output: Invalid

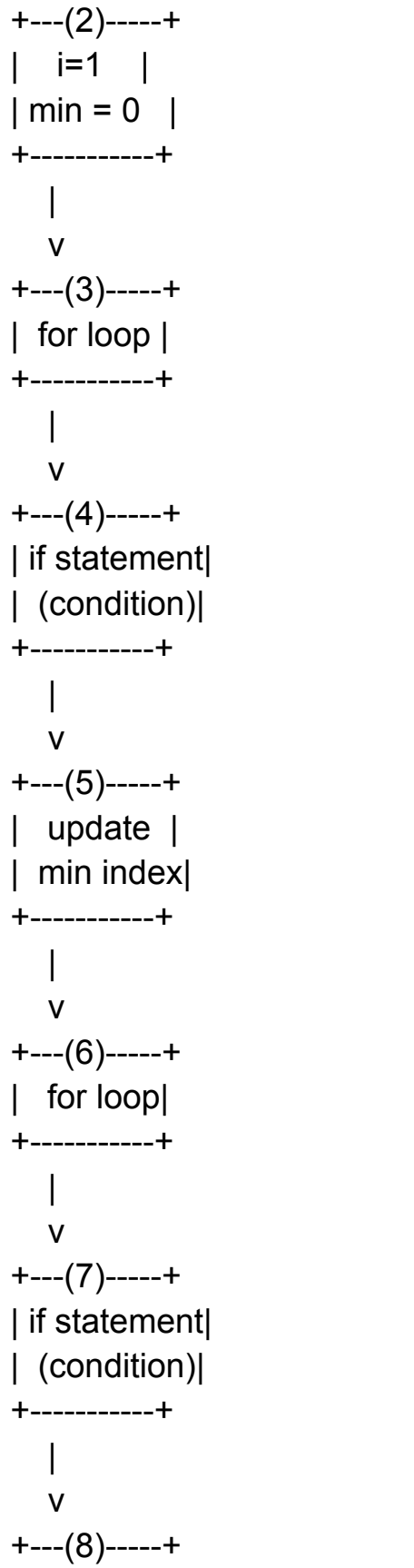
(h)
Input: a = -1, b = -2, c = -3
Expected Output: Invalid

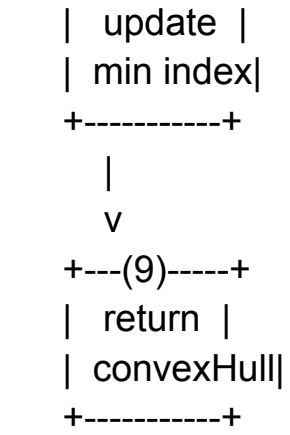
Input: a = -2, b = 4, c = 0
Expected output: Invalid

Section - B

1. **Control flow graph**




(2)
i=1
min = 0
(3)
for loop
(4)
if statement
(condition)
(5)
update
min index
(6)
for loop
(7)
if statement
(condition)
(8)



2.

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)}