Software Engineering IT314

Lab-7

Name: Yash Chauhan
Student ID: 202001082
Date: 12/04/2023

Section - A

Test cases:-

- **For P1**

→ **Equivalence Partitioning and Boundary Value Analysis**

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
a = [1, 2, 3, 4, 5, 6], v = 4	1
a = [5, 6, 7, 8, 9, 15], v = 1000	-1
a = [1, 1, 7, 10, 2, 3], v = 1	0
a = null, v = 70	An error message
Boundary Value Analysis	
Minimum array length: a = [], v = 10	-1
Maximum array length: a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20], v = 9.	8
The minimum value of v: a = [4, 5, 6, 7], v = 4	0
Maximum value of v: a = [1, 2, 3, 4, 6], v = 6	4

❖ **For P2**

➔ **Equivalence Partitioning and Boundary Value Analysis**

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
Invalid input: v is not an integer	An 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 Value Analysis	
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, ..., 9999]	0

- ❖ **For P3**
  - ➔ **Equivalence Partitioning**

Test Cases for Correct Inputs:

Tester Action and Input Data	Expected Outcome
v = 7, a = [1, 3, 5, 7, 9]	3
v = 0, a = [0, 1, 3, 5, 7, 9]	0
v = 10, a = [1, 3, 5, 7, 9, 10]	5

Test Cases for Incorrect Inputs:

Tester Action and Input Data	Expected Outcome
v = 3, a = [1, 2, 5, 7, 9]	-1
v = 11, a = [1, 3, 5, 7, 9, 10]	-1
v = 3, a = []	-1

Boundary Value Analysis:

Test Cases for Correct Inputs:

Tester Action and Input Data	Expected Outcome
v = 1, a = [1, 2, 3]	0
v = 2, a = [1, 2, 3]	1
v = 3, a = [1, 2, 3]	2
v = 9, a = [1, 5, 6, 7, 9]	4
v = 6, a = [1, 5, 6, 7, 9]	2
v = 5, a = [1, 5, 6, 7, 9]	1
v = 7, a = [1, 5, 6, 7, 9]	3
v = 3, a = [3]	0
v = 7, a = [7]	0
v = 2, a = []	-1

Test Cases for Incorrect Inputs:

Tester Action and Input Data	Expected Outcome
v = 2, a = [1, 3, 5, 7, 9]	-1
v = 10, a = [1, 3, 5, 7, 9]	-1
v = 6, a = [1, 3, 5, 7, 9]	-1
v = 1, a = [2, 3, 4, 5, 6]	-1
v = 7, a = [2, 3, 4, 5, 6]	-1
v = 4, a = [5, 6, 7, 8, 9]	-1

❖ **For P4**

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
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	
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

- **For P5**

Tester Action and Input Data	Expected Outcome
Equivalence Partitioning	
s1 is empty, s2 is non-empty string	true
s1 is a non-empty string, s2 is empty	false
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	
s1 = "hello", s2 = "hello world"	true
s1 = "hello", s2 = "world hello"	false
s1 = "hello", s2 = "hello"	true
s1 = "hello", s2 = "HELLO"	false

s1 = "abcdefghijklmnopqrstuvwxyz", s2 = "abcdefghijklmnopqrstuvwxyz"	**true**
s1 = "abcdefghijklmnopqrstuvwxyz", s2 = "abcdefghijklmno"	**true**
s1 = "", s2 = ""	**true**

- **For P6**

a) Equivalence classes for the system are

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

b) Test cases to cover the identified equivalence classes:

Class 1: -1, 0
Class 2: 1, 2, 5
Class 3: 3, 4, 5
Class 4: 5, 5, 7
Class 5: 6, 6, 6
Class 6: 3, 4, 5
Test case 1 covers class 1, test case 2 covers class 2, test case 3 covers class 3, test case 4 covers class 4, test case 5 covers class 5, and test case 6 covers class 6.

c) Test cases to verify the boundary condition A + B > C for the scalene triangle:

2, 3, 6
3, 4, 8
Both test cases have two sides shorter than the third side and should not form a triangle.

d) Test cases to verify the boundary condition A = C for the isosceles triangle:

2, 3, 3,
5, 6, 5
Both test cases have two equal sides, and should form an isosceles triangle.

e) Test cases to verify the boundary condition A = B = C for the equilateral triangle:

5, 5, 5
9, 9, 9
Both test cases have all sides equal and should form an equilateral triangle.

f) Test cases to verify the boundary condition A^2 + B^2 = C^2 for the right-angled triangle:

3, 4, 5,
5, 12, 13
Both test cases satisfy the Pythagorean theorem and should form a right-angled triangle.

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

2, 2, 4
3, 6, 9
Both test cases have two sides that add to the third side and should not form a triangle.

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

0, 1, 2
-1, -2, -3
Both test cases have at least one non-positive value, an invalid input.

Testing code with converge:

```
package test;
import static org.junit.Assert.*;
import org.junit.Test;
public class testing {
	@Test
	public void test1_1() {
	programs test = new programs();
	int a[] = {1,2,3,4,5};
	int output = test.linearSearch(2, a);
	assertEquals(1,output);
	}
	@Test
	public void test1_2() {
	programs test = new programs();
	int a[] = {1,2,3,4,5};
	int output = test.linearSearch(1, a);
	assertEquals(0,output);
	}
	@Test
	public void test1_3() {
	programs test = new programs();
	int a[] = {1,2,3,4,5};
	int output = test.linearSearch(7, a);
	assertEquals(-1,output);
	}

	@Test
	public void test1_4() {
	programs test = new programs();
	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
	programs test = new programs();
	int a[] = {1,2,2,3,4,5};
	int output = test.countItem(2, a);
```

```java
        assertEquals(2,output);
    }
    @Test
    public void test2_2() { //no of element p2
        programs test = new programs();
        int a[] = {1,2,2,3,4,5};
        int output = test.countItem(6, a);
        assertEquals(0,output);
    }

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

    @Test
    public void test3_1() { //binary search p3
        programs test = new programs();
        int a[] = {1,2,3,4,5};
        int output = test.binarySearch(3, a);
        assertEquals(2,output);
    }
    @Test
    public void test3_2() { //binary search p3
        programs test = new programs();
        int a[] = {1,2,3,4,5};
        int output = test.binarySearch(8, a);
        assertEquals(-1,output);
    }

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

    @Test
    public void test3_4() { //binary search p3
        programs test = new programs();
        int a[] = {1,2,3,4,5};
        int output = test.binarySearch(10, a);
        assertEquals(9,output);
    }
```

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

@Test
public void test4_2() {
    programs test = new programs();
    int output = test.triangle(8,8,10);
    assertEquals(1,output);
}

@Test
public void test4_3() {
    programs test = new programs();
    int output = test.triangle(0,0,0);
    assertEquals(3,output);
}

@Test
public void test4_4() {
    programs test = new programs();
    int output = test.triangle(0,0,0);
    assertEquals(1,output);
}

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

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

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

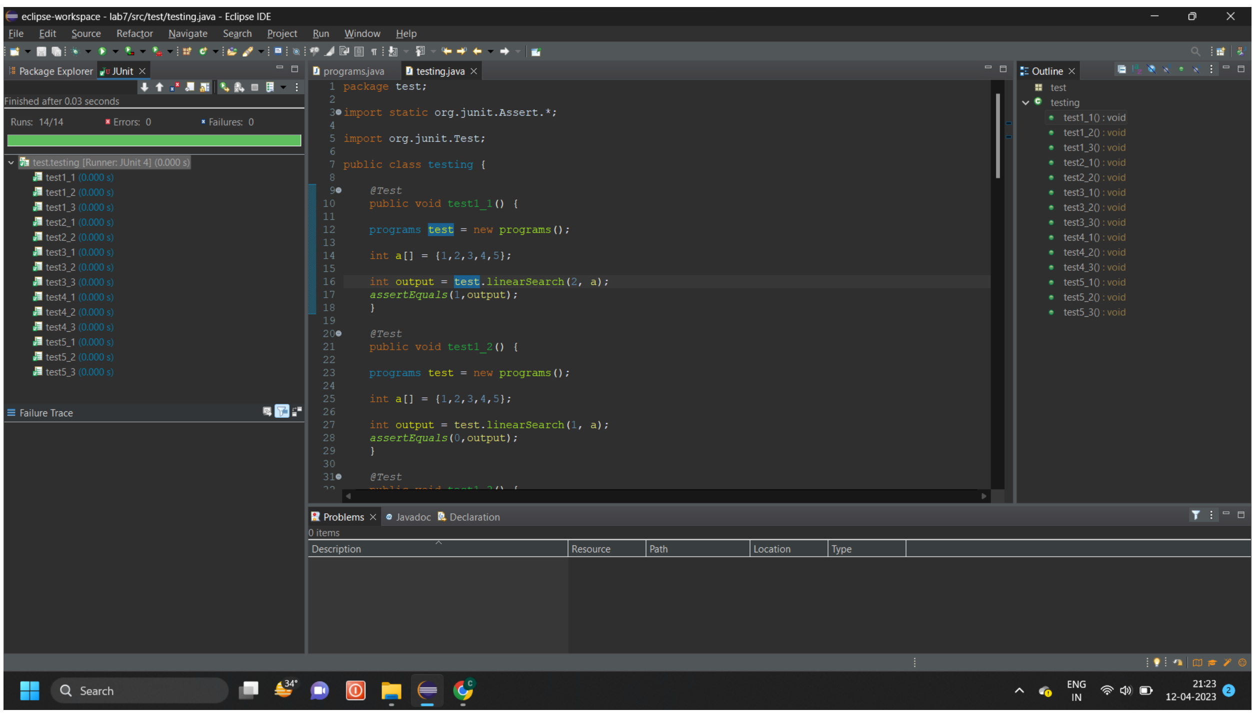
```
    }

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

}
```

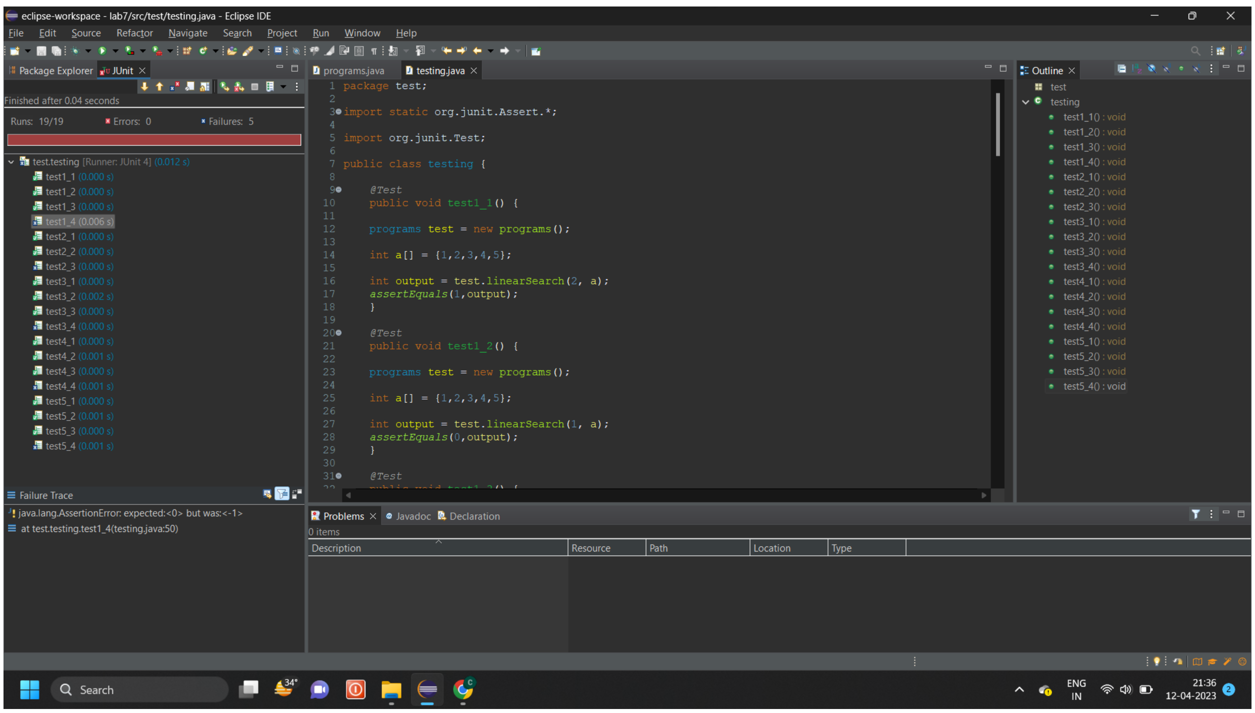
Correct test cases for all the programs:

Correct test cases: 1_1, 1_2, 1_3, 2_1, 2_2, 3_1, 3_2, 3_3, 4_1, 4_2, 4_3, 5_1, 5_2, 5_3



Incorrect test cases for all the programs:

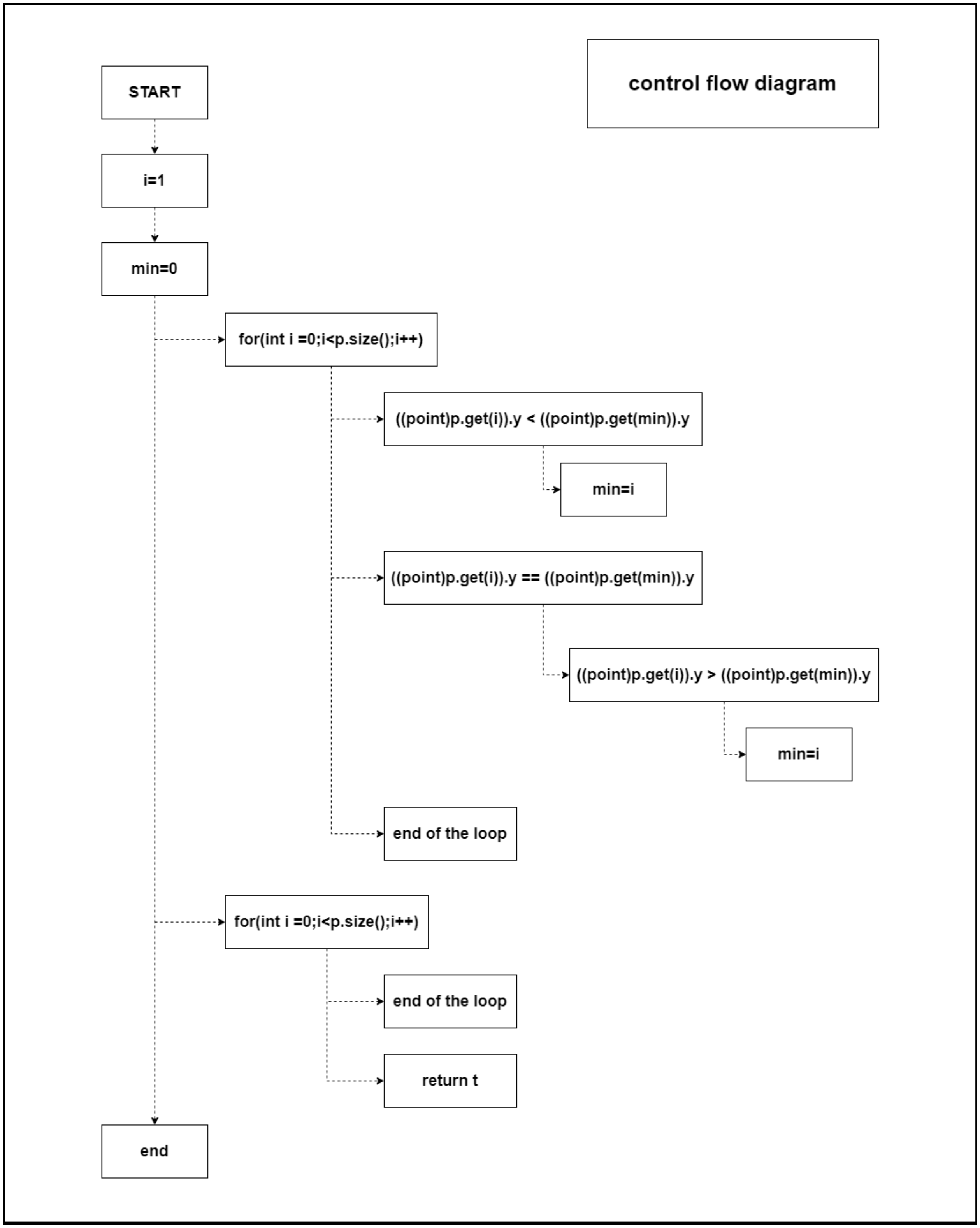
Incorrect Test case: 1_4, 2_3, 3_4, 4_4 & 5_4.



Section: B

1. **Convert the Java code comprising the beginning of the doGraham method into a control flow graph (CFG).**

Control flow diagram:


control flow diagram
START
i=1
min=0
for(int i =0;i<p.size();i++)
((point)p.get(i)).y < ((point)p.get(min)).y
min=i
((point)p.get(i)).y == ((point)p.get(min)).y
((point)p.get(i)).y > ((point)p.get(min)).y
min=i
end of the loop
for(int i =0;i<p.size();i++)
end of the loop
return t
end

2. Test sets for the given criteria:

a. Statement Coverage:

To achieve statement coverage, the following test cases should be sufficient:

- p with a single point.
- p with two or more points, where the first point has the smallest y-coordinate.

b. Branch Coverage:

To achieve branch coverage, the following test cases should be sufficient:

- p with a single point.
- p with two or more points, where the first point has the smallest y-coordinate.
- p with two or more points, where there are two or more points with the same smallest y-coordinate.

c. Basic Condition Coverage:

To achieve basic condition coverage, the following test cases should be sufficient:

- p with a single point.
- p with two or more points, where the first point has the smallest y-coordinate.
- p with two or more points, where there are two or more points with the same smallest y-coordinate.

- p with two or more points, where there are no points with the same smallest y-coordinate.