Name: Anurag
ID:202001187

Section A:

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?

Test Cases:

day	month	year	Expected Outcome
10	4	2023	09-04-2023
1	3	2002	28-04-2002
1	1	2001	31-12-2000
29	2	2001	Invalid
15	8	2202	14-8-2202
30	4	2016	Invalid

Day:

class	validity
dd<1	Invalid
1<=dd<=31	Valid
dd > 31	Invalid

Month:

class	validity
mm<1	Invalid
mm = 1,3,5,7,8,10,12(months with 31 days)	Valid
mm = 4,6,9,11(months with 30 days)	Valid
mm = 2(month with either 28 or 29 days)	Valid
mm > 12	Invalid

Year:

class	validity
yy<1900	Invalid
leap year 1900<=yy<=2015	Valid
non leap year 1900<=yy<=2015	Valid
yy > 2015	Invalid

Programs 1:

Tester Action and Input Data	**Expected Outcome**
[2, 4, 6, 8, 10],v = 2	0
[1, 3, 5, 7, 9],v = 2	-1
[2, 4, 6, 8, 10],v = 11	-1

[-100, 100]	0
[1,2,3,4,5,6],v = 6	5
[],v = 10	-1
NULL,v = 111	-1

Boundary value analysis:

Tester Action and Input Data	Expected Outcome
NULL	-1
[],v = 5	-1
[5],v = 5	0
[5],v = 60	-1
[3,5],v = 3	0
[3,5],v = 5	1
[3,5],v = 14	-1
[1,3,5],v = 1	0
[1,3,5],v = 5	2
[1,3,5],v = 10	-1
[1,2,3,4,5,6,7,8,9,1,0,11,111],v = 1	0

[1,2,3,4,5,6,7,8,9,1,0,11,111],v = 111	12
[1,2,3,4,5,6,7,8,9,1,0,11,111],v = 1111	-1

Program 2:

Equivalence Class partitioning for counting occurance:1

Tester Action and Input Data	Expected Outcome
[265, 41, 60, 80, 100],v = 100	1
[2655, 451, 6560, 1050, 1050],v = 1050	2
[[265545, 451, 65460, 1050, 105024]],v = 1	0
[[10,10,10,10]],v = 11	0
[],v = 100	0
NULL,v = 51	0
[0],v = 0	1
[-89,-89],v = -89	2

Boundary value analysis:

Tester Action and Input Data	Expected Outcome
[1, 2, 3, 4],v = 2	2
15, 10, 15, 15],v = 15	3
[],v = 100	0
NULL,v = 51	0
[-100,100,100,100],v = 10000	0
[-89,89],v = -89	1
[-890,890],v = 890	1

Program 3:

Equivalent test cases for binary search:

Tester Action and Input Data	**Expected Outcome**
[1, 21, 30, 40, 50],v = 21	1
[10, 20, 30, 40, 50, 60],v = 30	2
[10,100,1000,10000],v = 100000	-1
[,11,22,33,44],v = 444	-1
[11,20,200,300],v=11	0
[-100,-90,-80,100,1000],v = 10000	4
[],v = 12	-1
NULL,v = 168	-1

[1,2],v = 3	-1
[1,3],v=3	1

Boundary value analysis:

Tester Action and Input Data	Expected Outcome
[1, 2, 3, 4, 5],v = 2	1
[1, 2, 2, 351, 551],v = 2	2
[1,22,33,44,55],v = 66	-1
[2, 4, 6, 8, 10],v = 51	-1
[-100, 0, 1000],v = -100	0
[-100, 0, 1000],v = 1000	2
[],v=0	-1
NULL,v = 4	-1

Program 4:

Equivalent class test cases of checking the type of triangle

Tester Action and Input Data	Expected Outcome
a=2,b=2,c=2	EQUILATERAL
a=1,b=1,c=1	EQUILATERAL

a=0,b=0,c=0	INVALID
a=-1,b=-1,c=-1	INVALID
a=10,b=10,c=0	INVALID
a=17,b=17,c=5	ISOCELES
a=15,b=2,c=15	ISOCELES
a=6,b=11,c=5	SCALENE
a=16,b=21,c=25	SCALENE
a=-1,b=21,c=25	INVALID
a=2,b=3,c=4	SCALENE

Boundary value analysis:

Tester Action and Input Data	Expected Outcome
a=2,b=2,c=2	EQUILATERAL
a=0,b=0,c=0	INVALID
a=INT_MAX,b = INT_MAX,c = INT_MAX	EQUILATERAL
a=INT_MIN,b=INT_MIN,c=INT_MIN	INVALID
a=1,b=1,c=2	ISOSCELES
a=15,b=12,c=15	ISOSCELES
a = INT_MAX,b = 1,c = INT_MAX	ISOSCELES
a=1,b=2,c=3	SCALENE
a = INT_MAX,b = 1,c = INT_MAX -	SCALENE

1	

Program 5

Equivalent test cases for prefix searching are as follows:

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1 = "",s2 = ""	true
s1 = "po",s2 = "poojan"	true
s1 = "poo",s2 = "po"	false
s1 = "abc",s2 = ""	false
s1 = "",s2 = "abc"	true
s1 = "o",s2 = "ott"	true
s1 = "abc",s2 = "def"	false
s1 = "deg",s2 = "def"	false

Boundary value analysis:

Tester Action and Input Data	Expected Outcome
s1= "abcd",s2 = "abcd"	true
s1= "",s2 = ""	true
s1= "abcd",s2 = ""	false
s1= "",s2 = "abcd"	true
s1 = "aef",s2 = "def"	false

s1 = "def",s2 = "deg"	false
s1 = "a",s2 = "att"	true
s1 = "poojan",s2 = "patel"	false

Program 6:

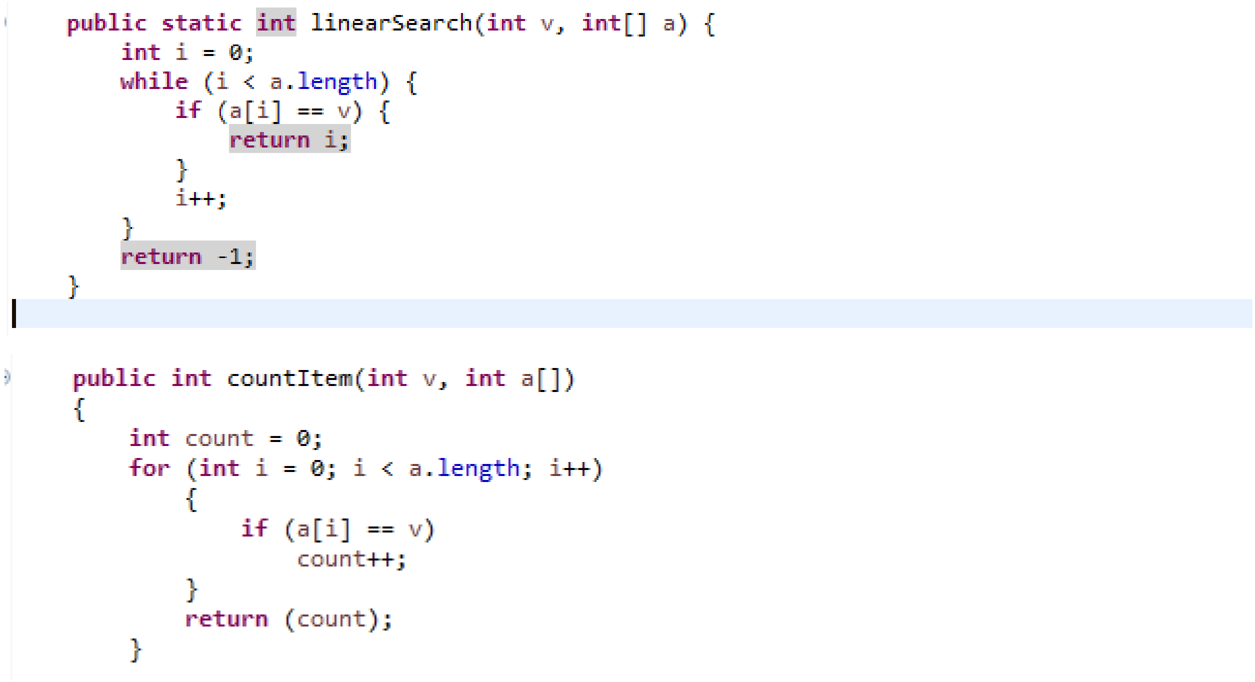
(a) All equivalent classes

Class ID	Class
E1	All sides are positive
E2	two of its sides are zero
E3	One of its sides are negative
E4	Sum of two sides is less than third side
E5	Any of the side/sides is negative

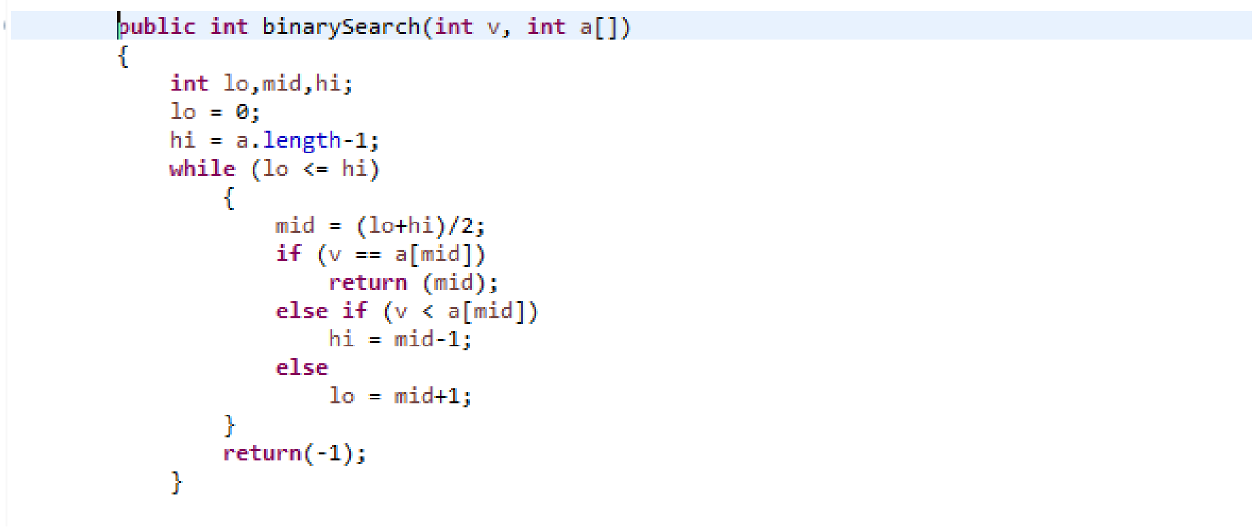
(b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.

Test Case ID	Class ID	Test Case
T1	E1	A = 1,B = 1,C = 1
T2	E1	A = 3, B = 4, C= 5
T3	E2	A = 0,B = 0,C = 1
T4	E3	A = 0,B = 1,C = 2
T5	E4	A = 1, B = 3, C = 8
T6	E5	A = -1,C = 1,D = 5

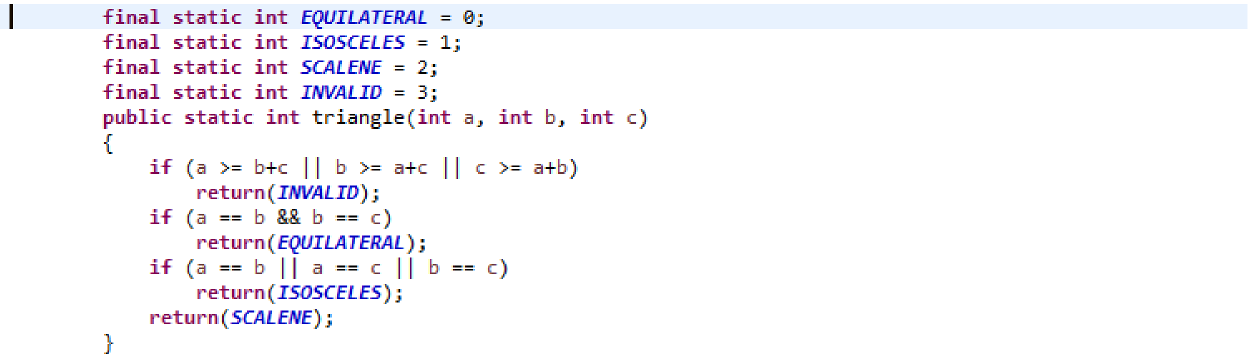
(c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.
A = 1, B = 3, C = 2 (d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.
A = 3,B = 2, C = 3 (e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.
A = 30,B = 30,C = 30 (f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.
A = 6,B = 8,C = 10 (g) For the non-triangle case, identify test cases to explore the boundary. A = 20, B = 10,C = 5 (h) For non-positive input, identify test points. A = 0, B = 10, C = 0
A = 0,B = 0,C = 0
A = 0, B = -1, C = 10



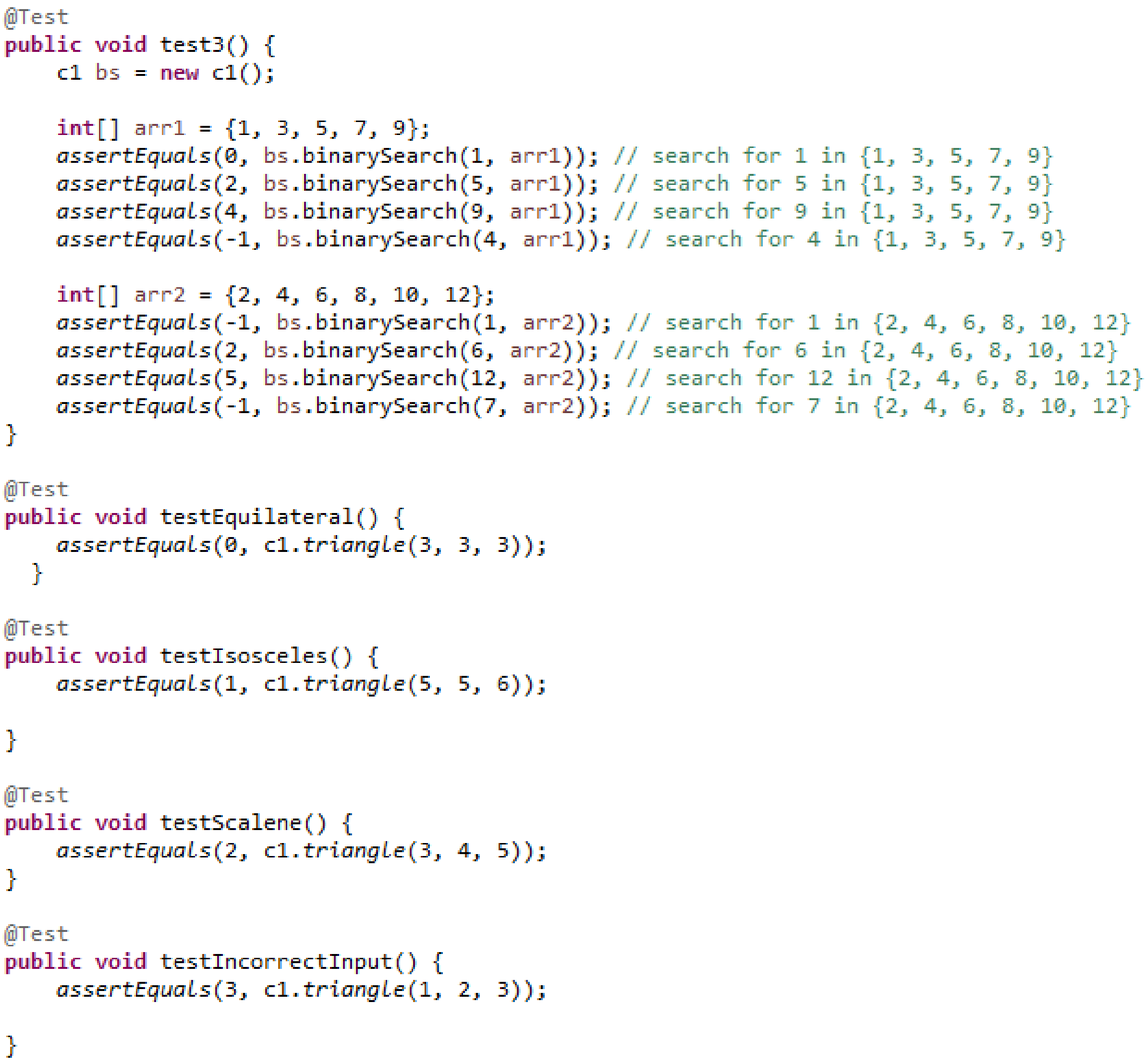
```
public static int linearSearch(int v, int[] a) {
    int i = 0;
    while (i < a.length) {
        if (a[i] == v) {
            return i;
        }
        i++;
    }
    return -1;
}

 public int countItem(int v, int a[])
 {
     int count = 0;
     for (int i = 0; i < a.length; i++)
         {
             if (a[i] == v)
                 count++;
         }
         return (count);
     }
```



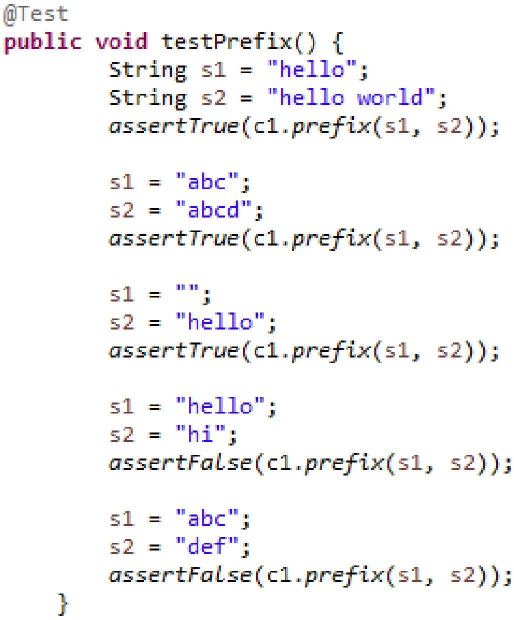
```java
public int binarySearch(int v, int a[])
{
    int lo,mid,hi;
    lo = 0;
    hi = a.length-1;
    while (lo <= hi)
        {
            mid = (lo+hi)/2;
            if (v == a[mid])
                return (mid);
            else if (v < a[mid])
                hi = mid-1;
            else
 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



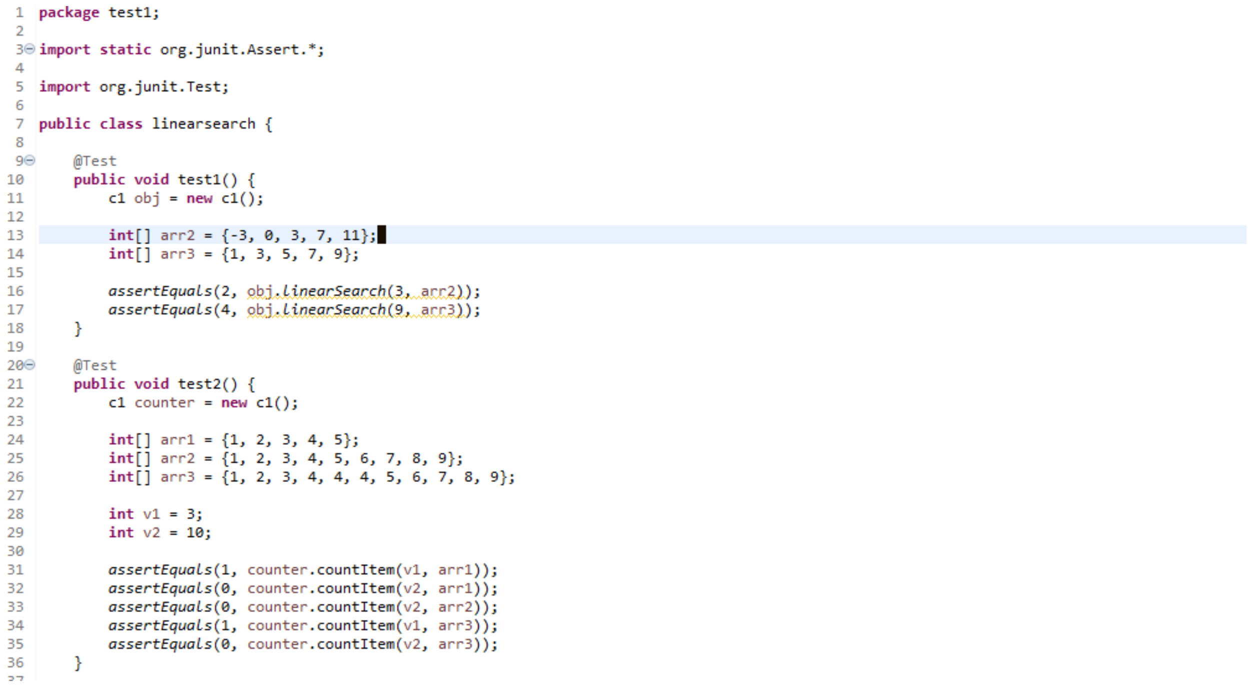
```java
final static int EQUILATERAL = 0;
final static int ISOSCELES = 1;
final static int SCALENE = 2;
final static int INVALID = 3;
public static int triangle(int a, int b, int c)
{
    if (a >= b+c || b >= a+c || c >= a+b)
        return(INVALID);
    if (a == b && b == c)
        return(EQUILATERAL);
    if (a == b || a == c || b == c)
        return(ISOSCELES);
    return(SCALENE);
}
```



```
@Test
public void test3() {
    c1 bs = new c1();

    int[] arr1 = {1, 3, 5, 7, 9};
    assertEquals(0, bs.binarySearch(1, arr1)); // search for 1 in {1, 3, 5, 7, 9}
    assertEquals(2, bs.binarySearch(5, arr1)); // search for 5 in {1, 3, 5, 7, 9}
    assertEquals(4, bs.binarySearch(9, arr1)); // search for 9 in {1, 3, 5, 7, 9}
    assertEquals(-1, bs.binarySearch(4, arr1)); // search for 4 in {1, 3, 5, 7, 9}

    int[] arr2 = {2, 4, 6, 8, 10, 12};
    assertEquals(-1, bs.binarySearch(1, arr2)); // search for 1 in {2, 4, 6, 8, 10, 12}
    assertEquals(2, bs.binarySearch(6, arr2)); // search for 6 in {2, 4, 6, 8, 10, 12}
    assertEquals(5, bs.binarySearch(12, arr2)); // search for 12 in {2, 4, 6, 8, 10, 12}
    assertEquals(-1, bs.binarySearch(7, arr2)); // search for 7 in {2, 4, 6, 8, 10, 12}
}

@Test
public void testEquilateral() {
    assertEquals(0, c1.triangle(3, 3, 3));
  }

@Test
public void testIsosceles() {
    assertEquals(1, c1.triangle(5, 5, 6));

}

@Test
public void testScalene() {
    assertEquals(2, c1.triangle(3, 4, 5));
}

@Test
public void testIncorrectInput() {
    assertEquals(3, c1.triangle(1, 2, 3));

}
```



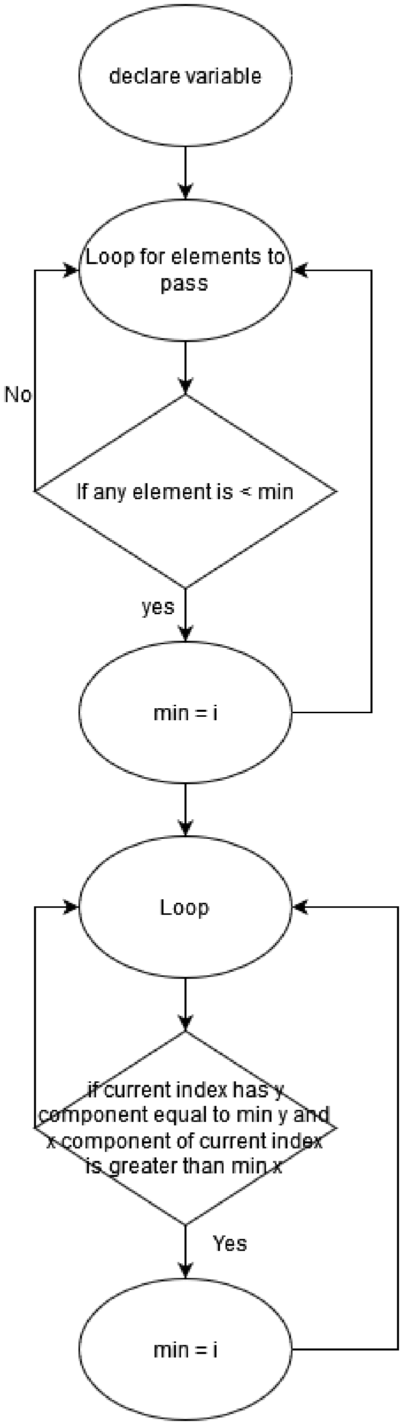
```java
@Test
public void testPrefix() {
        String s1 = "hello";
        String s2 = "hello world";
        assertTrue(c1.prefix(s1, s2));

        s1 = "abc";
        s2 = "abcd";
        assertTrue(c1.prefix(s1, s2));

        s1 = "";
        s2 = "hello";
        assertTrue(c1.prefix(s1, s2));

        s1 = "hello";
        s2 = "hi";
        assertFalse(c1.prefix(s1, s2));

        s1 = "abc";
        s2 = "def";
        assertFalse(c1.prefix(s1, s2));
    }
```



```java
package test1;

import static org.junit.Assert.*;

import org.junit.Test;

public class linearsearch {

    @Test
    public void test1() {
        c1 obj = new c1();

        int[] arr2 = {-3, 0, 3, 7, 11};
        int[] arr3 = {1, 3, 5, 7, 9};

        assertEquals(2, obj.linearSearch(3, arr2));
        assertEquals(4, obj.linearSearch(9, arr3));
    }

    @Test
    public void test2() {
        c1 counter = new c1();

        int[] arr1 = {1, 2, 3, 4, 5};
        int[] arr2 = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        int[] arr3 = {1, 2, 3, 4, 4, 4, 5, 6, 7, 8, 9};

        int v1 = 3;
        int v2 = 10;

        assertEquals(1, counter.countItem(v1, arr1));
        assertEquals(0, counter.countItem(v2, arr1));
        assertEquals(0, counter.countItem(v2, arr2));
        assertEquals(1, counter.countItem(v1, arr3));
        assertEquals(0, counter.countItem(v2, arr3));
    }
```

Section B

(1) Control flow graph


declare variable
Loop for elements to pass
No
If any element is < min
yes
min = i
Loop
if current index has y component equal to min y and x component of current index is greater than min x
Yes
min = i

(2) Test Cases

(a) Statement coverage test set: ** In this all the statements in code should be covered

Test Number	Test Case
1	p is empty array
2	p has one point object
3	p has two points object with different y component
4	p has two points object with different x component
5	p has three or more point object with different y component

(b) Branch Coverage test set: ** In this all branch are taken atleast once

Test Number	Test Case
1	p is empty array
2	p has one point object
3	p has two points object with different y component
4	p has two points object with different x component
5	p has three or more point object with different y component
6	p has three or more point object with same y component
7	p has three or more point object with all same x component

8	p has three or more point object with all different x component
9	p has three or more point object with some same and some different x component

(c) Basic condition coverage test set: **Each boolean expression has been evaluated to both true and false

Test Number	Test Case
1	p is empty array
2	p has one point object
3	p has two points object with different y component
4	p has two points object with different x component
5	p has three or more point object with different y component
6	p has three or more point object with same y component
7	p has three or more point object with all same x component
8	p has three or more point object with all different x component
9	p has three or more point object with some same and some different x component
10	p has three or more point object with some same and some different y component

11	p has three or more point object with all different y component
12	p has three or more point object with all same y component