IT314-Lab 7
ID-202001060

Section -A

P1. The function linearSearch searches for a value v in an array of integers a. If v appears in the array
a, then the function returns the first index i, such that a[i] == v; otherwise, -1 is returned.

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

Code::

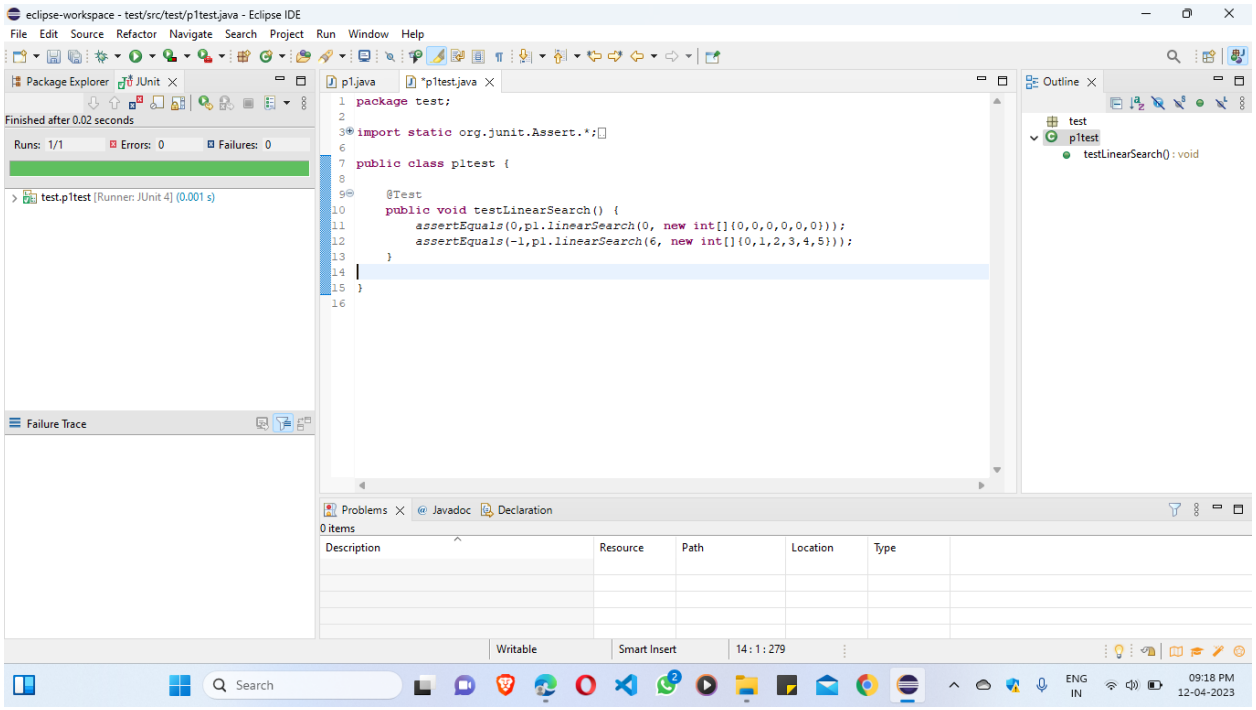
```
public int linearSearch(int v, int a[])
{
int i = 0;
while (i < a.length)
{
if (a[i] == v)
return(i);
i++;
}
return (-1);
}
```

Equivalence class number=2

Test case

a[]=0,0,0,0,0,0 v=0 y(output)=0

a[]=0,1,2,3,4,5 v=6 y= -1



P2. The function countItem returns the number of times a value v appears in an array of integers a.

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

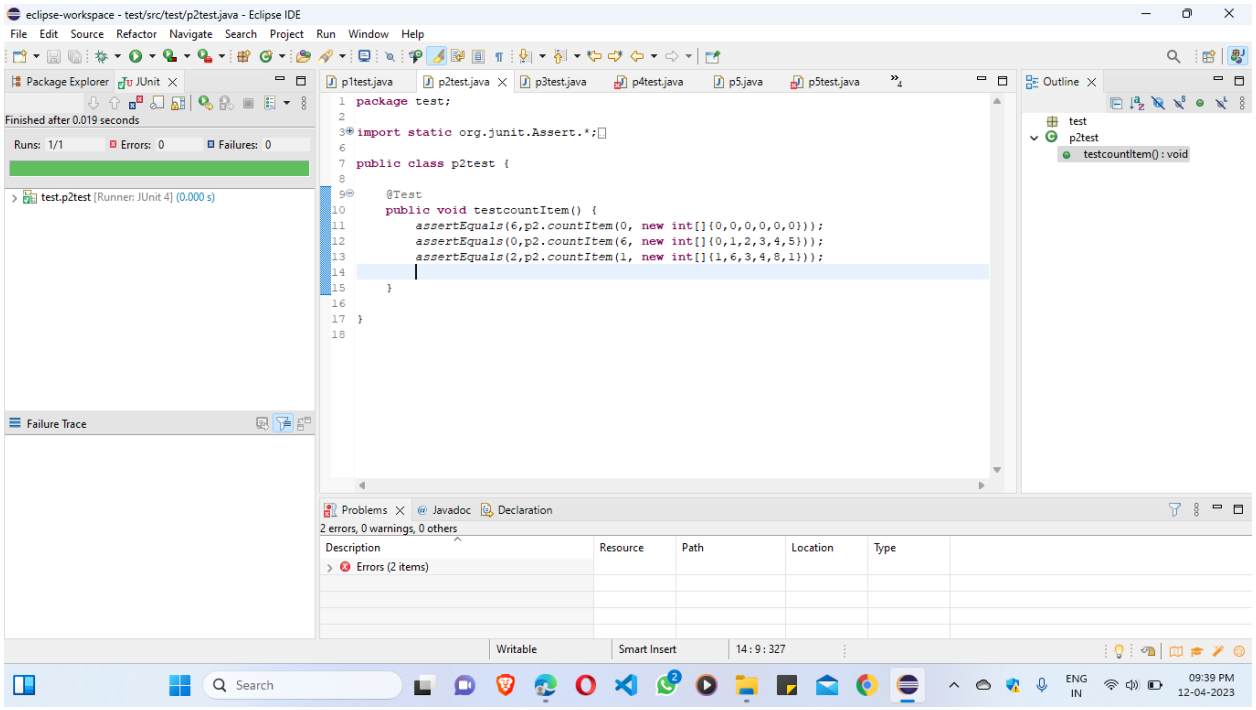
Code::

```
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;
}
```

Equivalence class number=3

Test case

a[]=0,0,0,0,0,0	v=0	y(output)=6
a[]=0,1,2,3,4,5	v=6	y=0
a[]=1,6,3,4,8,1	v=1	y=2
a[]=NULL	v=98	y=-1



P3. The function binarySearch searches for a value v in an ordered array of integers a. If v appears in
the array a, then the function returns an index i, such that a[i] == v; otherwise, -1 is returned.
Assumption: the elements in the array a are sorted in non-decreasing order.

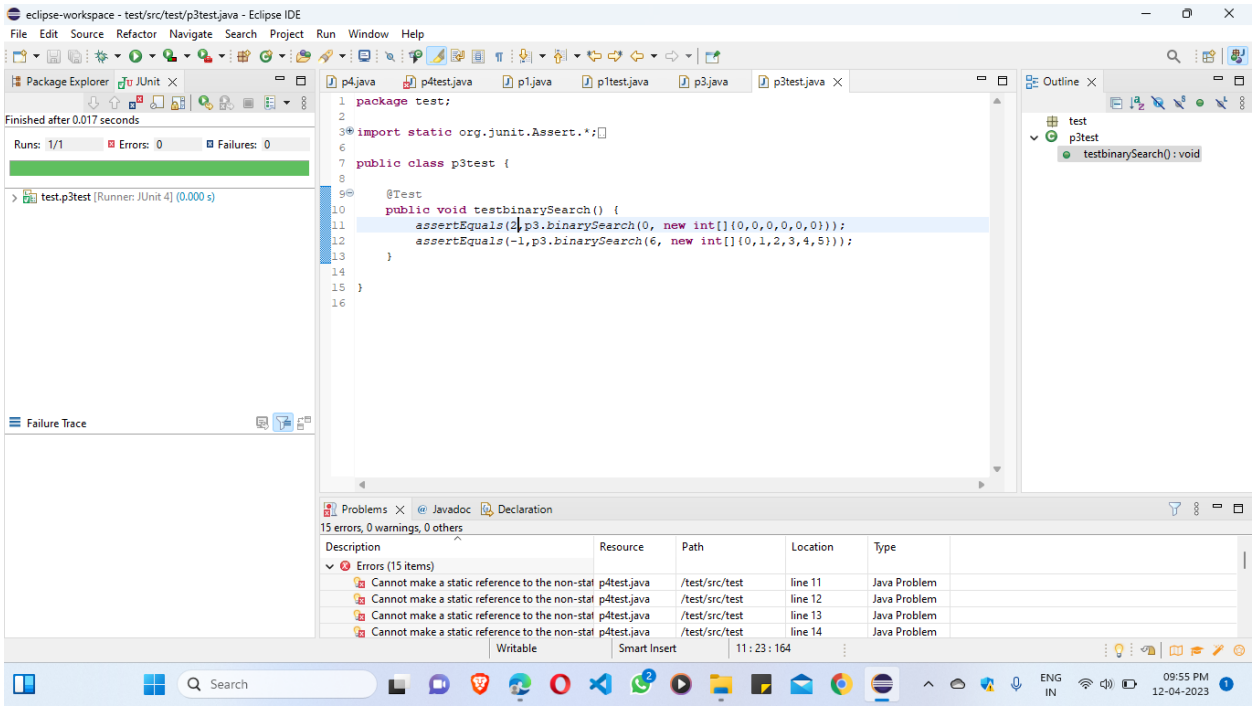
```
int binarySearch(int v, int a[])
{
int lo,mid,hi;
lo = 0;
hi = a.length-1;
while (lo <= hi)
{
mid = (lo+hi)/2;
if (v == a[mid])
return (mid);
else if (v < a[mid])
hi = mid-1;
else
lo = mid+1;

}
return(-1);
}
```

Equivalence class number=2

Test case

a[]=0,0,0,0,0,0	v=0	y(output)=0
a[]=0,1,2,3,4,5	v=6	y=-1



P4. The following problem has been adapted from The Art of Software Testing, by G. Myers (1979). The
function triangle takes three integer parameters that are interpreted as the lengths of the sides of a
triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal),
scalene (no lengths equal), or invalid (impossible lengths).

```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
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);

Equivalence class number=6

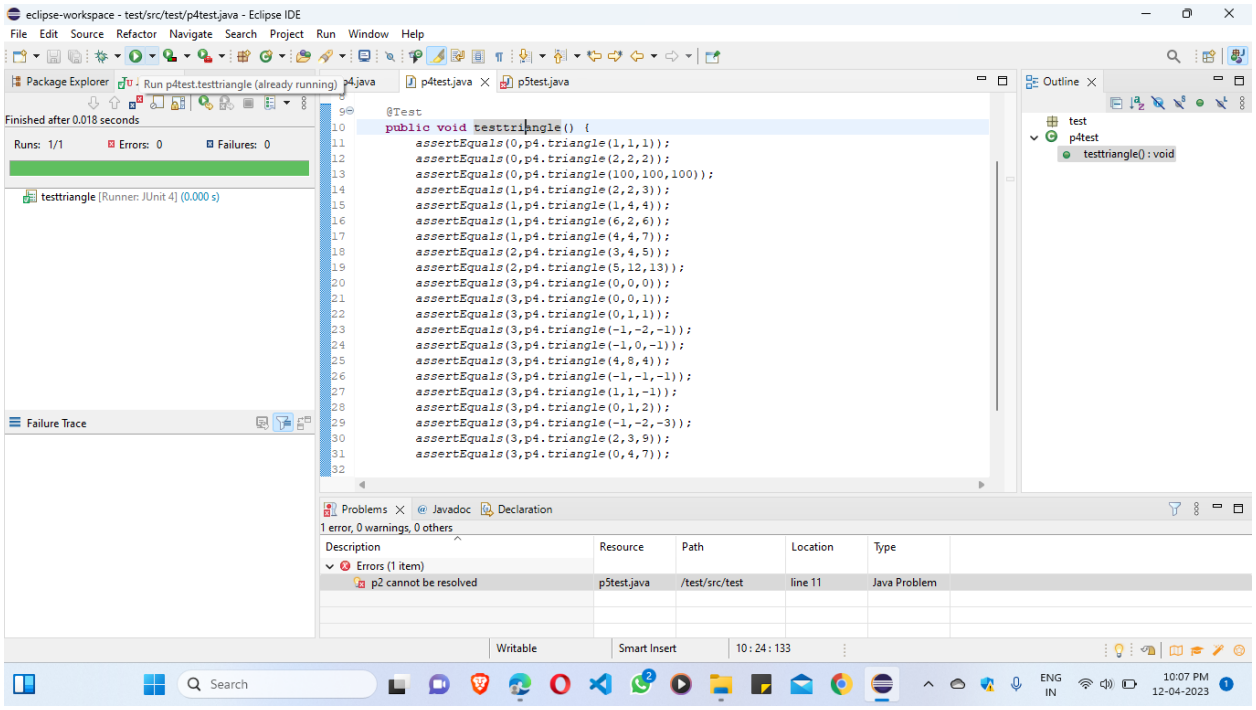
Test case

a. Equilateral
   - 1,1,1 0
   - 2,2,2 0
   - 100,100,100 0

b. Isosceles
   - 2,2,3 1
   - 1,4,4 1
   - 6,2,6 1
   - 4,4,7 1

c. Scalene
   - 3,4,5 2
   - 5,12,13 2

d. Invalid
   - 0,0,0 3
   - 0,0,1 3
   - 0,1,1 3
   - -1,-2,-1 3
   - -1,0,-1 3
   - 4,8,4 3
   - -1,-1,-1 3
   - 1,1,-1 3
   - 0,1,2 3
   - -1,0,1 3
   - -1,-2,-3 3
   - 2,3,9 3
   - 0,4,7 3



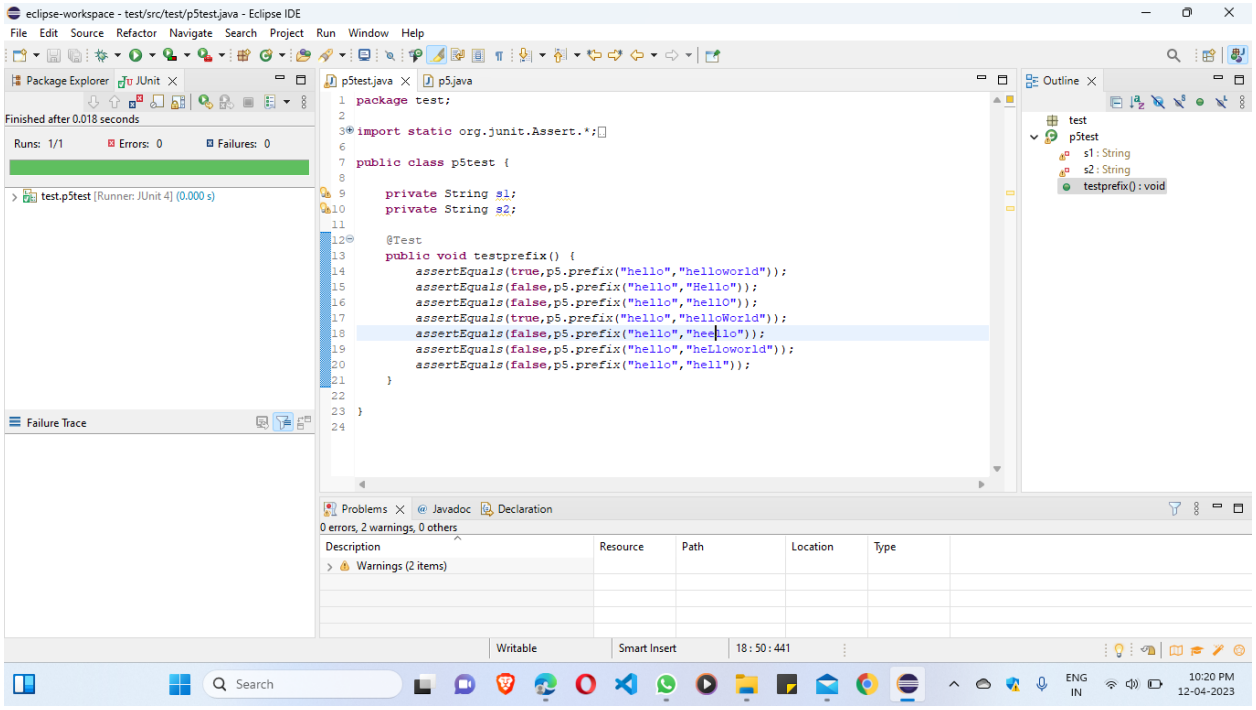
P5. The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix
of string s2 (you may assume that neither s1 nor s2 is null).

```
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;
}
```

Equivalence class number=7

Test case

s1=hello	s2=helloworld	y=true
s1=hello	s2=Hello	y=false
s1=hello	s2=hellO	y=false
s1=hello	s2=helloWorld	y=true
s1=hello	s2=heello	y=false
s1=hello	s2=heLloworld	y=false
s1=hello	s2=hell	y=false



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

- Equilateral
- Isosceles
- Scalene

- One or more zero
- Sum of any 2 side more than third
- One ore more negative value

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

- 1-3
- 4-7
- 8-9
- 10-22

(Hint: you must need to be ensure that the identified set of test cases cover all identified equivalence
classes)

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

- 13-16, 18, 20-23

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

- 4-8

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

- 4-7

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.

- 8-11

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

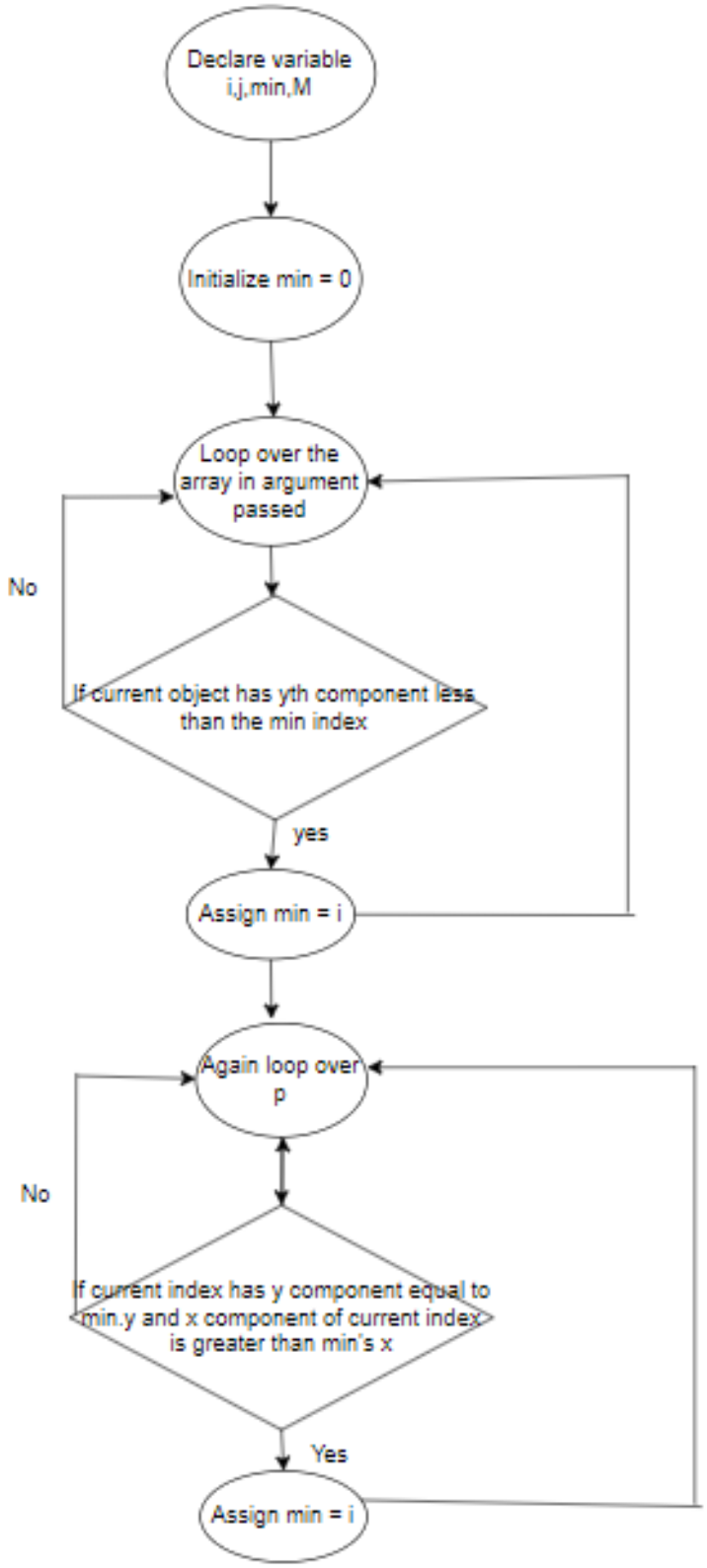
- 10-22

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

- 13,14,16,17,19,20

SECTION-B

1) Control flow graph



2) Test Cases

a)Statement coverage

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 coverge

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 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