IT-314
Lab-7 Lab Session: Software Engineering

Name : Kushal Soni
St. Id: 202001058

Section A:

Q1: 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?

Equivalence Partitioning Test Cases:

- Test Case 1: Input day, month, and year within the valid range. For example: day = 15, month = 6, year = 2000. Expected Outcome: Valid previous date.
- Test Case 2: Input day, month, and year with the day value exceeding the valid range. For example: day = 32, month = 7, year = 1995. Expected Outcome: Invalid date error message.
- Test Case 3: Input day, month, and year with the month value exceeding the valid range. For example: day = 28, month = 13, year = 1985. Expected Outcome: Invalid date error message.
- Test Case 4: Input day, month, and year with the year value exceeding the valid range. For example: day = 10, month = 3, year = 2016. Expected Outcome: Invalid date error message.

Boundary Value Analysis Test Cases:

- Test Case 1: Input day, month, and year with the minimum valid values. For example: day = 1, month = 1, year = 1900. Expected Outcome: Invalid date error message.
- Test Case 2: Input day, month, and year with the maximum valid values. For example: day = 31, month = 12, year = 2015. Expected Outcome: Valid previous date.
- Test Case 3: Input day with the minimum valid value, month with the maximum valid value, and year with the maximum valid value. For example: day = 1, month = 12, year = 2015. Expected Outcome: Valid previous date.
- Test Case 4: Input day with the maximum valid value, month with the minimum valid value, and year with the minimum valid value. For example: day = 31, month = 1, year = 1900. Expected Outcome: Valid previous date.
- Test Case 5: Input day with the maximum valid value, month with the maximum valid value, and year with the minimum valid value. For example: day = 31, month = 12, year = 1900. Expected Outcome: Invalid date error message.

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);
}
```

Test cases:

1.Equivalence Partitioning:

→ Tester Action and Input Data Expected Outcome

a = {2, 3, 5, 7, 11}, v = 5 Result: 2

Explanation: The value v is found in the array a at index 2.

→ Tester Action and Input Data Expected Outcome

a = {2, 3, 5, 7, 11}, v = 8 Result: -1

Explanation: The value v is not found in the array a, so -1 is returned.

2.Boundary Value Analysis:

→ Tester Action and Input Data Expected Outcome

a = {4}, v = 4 Result: 0

Explanation: The array a has only one element which matches the value v, so the expected outcome is 0 (index of v in a).

→ Tester Action and Input Data Expected Outcome

a = {4}, v = 2 Result: -1

Explanation: The array a has only one element which does not match the value v, so -1 is returned.

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);
}
```

Test cases:

1.Equivalence Partitioning:

→ Tester Action and Input Data — Expected Outcome
v = null, a = [1, 2, 3] — An Error message
Explanation: null not present in the array.

→ Tester Action and Input Data — Expected Outcome
v = 1, a = null — An Error message
Explanation: array is null.

→ Tester Action and Input Data — Expected Outcome
v = 1, a = [1, 1, 1] — 3
Explanation: ‘1’ occurs three times in a[1,1,1].

→ Tester Action and Input Data — Expected Outcome
v = 0, a = [1, 2, 3] — 0

→ Tester Action and Input Data — Expected Outcome
v = 4, a = [1, 2, 3] — 0

2.Boundary Value Analysis:

→ Tester Action and Input Data — Expected Outcome
v = -1, a = [1, 2, 3] — 0

→ Tester Action and Input Data — Expected Outcome
v = 1, a = [] — 0

→ Tester Action and Input Data	Expected Outcome
v = 0, a = [0, 0, 0]	3

→ Tester Action and Input Data	Expected Outcome
v = 5, a = [5, 5, 5]	3

→ Tester Action and Input Data	Expected Outcome
v = 2, a = [1, 2, 2, 2, 3]	3

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 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);
}
```

Test cases:

1.Equivalence Partitioning:

→ Tester Action and Input Data   Expected Outcome
v = 5, a = [1, 3, 5, 7, 9]   Returns 2

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

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

2.Boundary Value Analysis:

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

→ Tester Action and Input Data   Expected Outcome
v = 9, a = [1, 3, 5, 7, 9]   Returns 4

→ Tester Action and Input Data   Expected Outcome
v = 3, a = [1, 3, 5, 7, 9]   Returns 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);
}
```

Test cases:

1.Equivalence Partitioning:

Tester Action and Input Data	Expected Outcome
a = 3, b = 3, c = 3	Returns 0 (EQUILATERAL)
a = 3, b = 4, c = 5	Returns 2 (SCALENE)
a = 0, b = 1, c = 2	Returns 3 (INVALID)
a = 3, b = 4, c = 7	Returns 3 (INVALID)

2.Boundary Value Analysis:

Tester Action and Input Data	Expected Outcome
a = 1, b = 1, c = 1	Returns 0 (EQUILATERAL)
a = 0, b = 1, c = 1	Returns 3 (INVALID)
a = 1, b = 0, c = 1	Returns 3 (INVALID)

→ Tester Action and Input Data   Expected Outcome
a = 1, b = 1, c = 0   Returns 3 (INVALID)

→ Tester Action and Input Data   Expected Outcome
a = 2147483647, b = 2147483647, c = 2147483647   Returns 0 (EQUILATERAL)

→ Tester Action and Input Data   Expected Outcome
a = 2147483647, b = 2147483647, c = 1   Returns 2 (SCALENE)

→ Tester Action and Input Data   Expected Outcome
a = 2147483647, b = 2147483647, c = 2147483645   Returns 2 (SCALENE)

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;
}
```

Test cases:

1.Equivalence Partitioning:

→ Tester Action and Input Data Expected Outcome
s1 = "hello", s2 = "helloworld" Returns true

→ Tester Action and Input Data Expected Outcome
s1 = "prefix", s2 = "sufprefix" Returns false

→ Tester Action and Input Data Expected Outcome
s1 = "abc", s2 = "abcdef" Returns true

→ Tester Action and Input Data Expected Outcome
s1 = "prefix", s2 = "prefix" Returns true

2.Boundary Value Analysis:

→ Tester Action and Input Data Expected Outcome
s1 = "", s2 = "hello" Returns true

→ Tester Action and Input Data Expected Outcome
s1 = "hello", s2 = "" Returns false

→ Tester Action and Input Data Expected Outcome
s1 = "", s2 = "" Returns true

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

1. Scalene triangle: A, B, and C are positive floating values such that A + B > C, A + C > B, and B + C > A.
2. Isosceles triangle: A, B, and C are positive floating values such that A = B, A ≠ C, or B = C, A ≠ B.
3. Equilateral triangle: A, B, and C are positive floating values such that A = B = C.
4. Right-angled triangle: A, B, and C are positive floating values such that A^2 + B^2 = C^2 or B^2 + C^2 = A^2 or A^2 + C^2 = B^2.
5. Non-triangle: A, B, and C are positive floating values such that A + B ≤ C, A + C ≤ B, or B + C ≤ A.
6. Non-positive input: A, B, or C is not a positive floating value.

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

1. Scalene triangle: A = 5, B = 7, C = 9 (A + B > C)
2. Isosceles triangle: A = 4, B = 4, C = 6 (A = B, A ≠ C)
3. Equilateral triangle: A = 3, B = 3, C = 3 (A = B = C)
4. Right-angled triangle: A = 3, B = 4, C = 5 (A^2 + B^2 = C^2)
5. Non-triangle: A = 1, B = 2, C = 6 (A + B ≤ C)
6. Non-positive input: A = -2, B = 4, C = 5 (A is not a positive floating value)

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

A = 0.1, B = 0.2, C = 0.3 (A + B > C)

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

A = 2, B = 3, C = 2 (A = C)

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

A = 5, B = 5, C = 5 (A = B = C)

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

A = 3, B = 4, C = 5 (A^2 + B^2 = C^2)

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

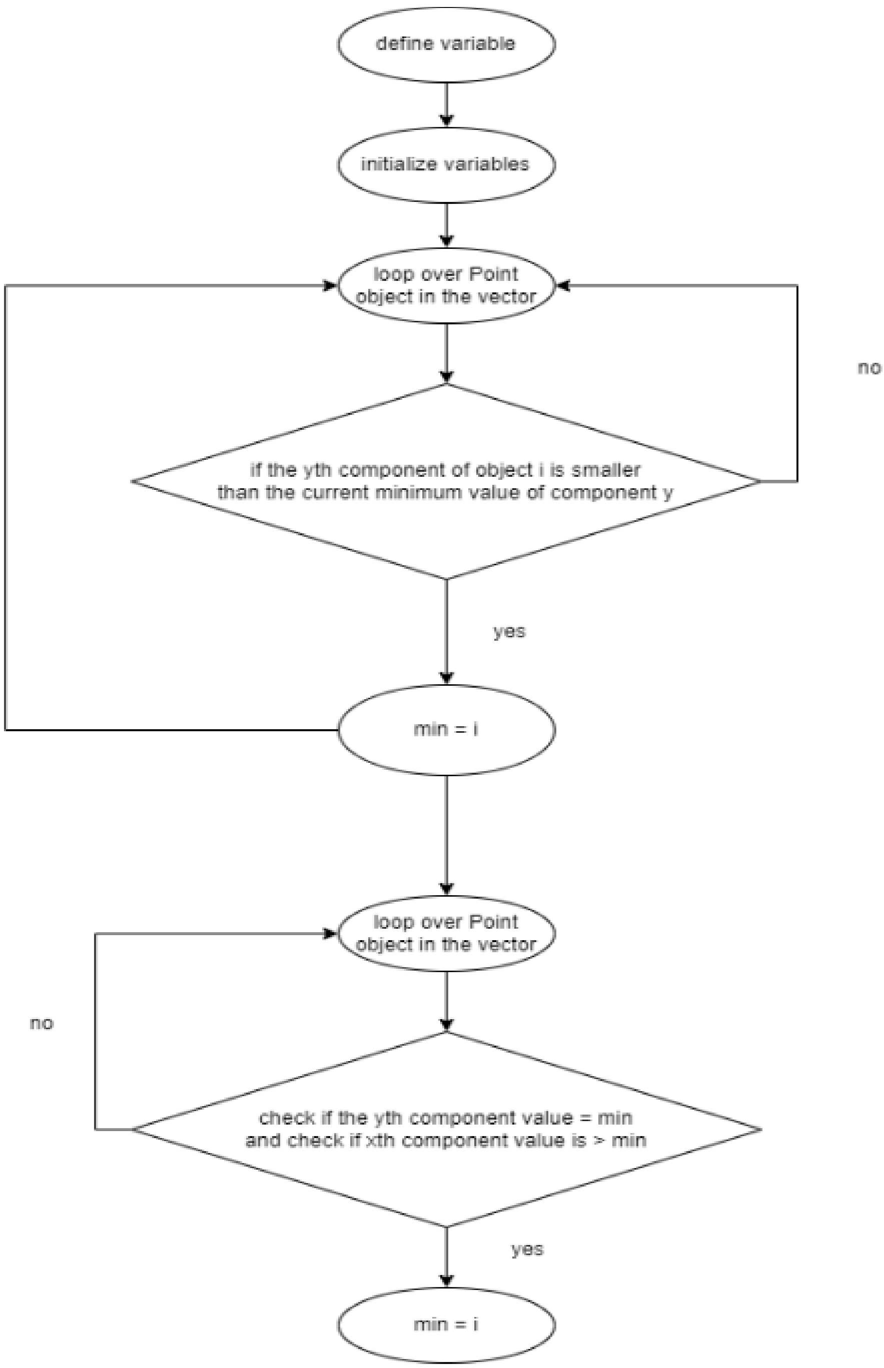
A = 1, B = 2, C = 3 (A + B ≤ C)

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

A = -1, B = 3, C = 4 (A is not a positive floating value)

Section B:

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



b) Construct test sets for your flow graph that are adequate for the following criteria:

1. Statement Coverage
   - Test Case 1: p = [(0,0)]
   - Test Case 2: p = [(0,0), (1,1)]

2. Branch Coverage
   - Test Case 3: p = [(0,0), (1,1)]
   - Test Case 4: p = [(1,1), (0,0)]

3. Basic Condition Coverage
   - Test Case 5: p = [(0,0), (1,1)]
   - Test Case 6: p = [(1,1), (0,0)]
   - Test Case 7: p = [(0,0), (0,1)]
   - Test Case 8: p = [(0,1), (0,0)]