Sample Questions

1. Consider the file RoasterDAO.java within the attached folder. Apply the input space partitioning technique to all the methods within that file. You need to use two different test criteria: base choice coverage and pairwise coverage. You are required to:
    a. List all of the variables
    b. Apply base choice coverage once. You need to show your complete tests design and test cases as well
    c. Apply pair-wise coverage once. You need to show your complete tests design and test cases as well.
2. Consider the file SomeClass.java.

```
// Introduction to Software Testing
// Authors: Paul Ammann & Jeff Offutt

********************************************************** */
public class SomeClass
{

  private static boolean someMethod (int i, int j)
  {
    if (j%i == 0)
      return true;
    else
      return false;
  }
  private static void SomeClass (int n)
  {
    int curNum;          // Value currently considered for primeness
    int numsFound;       // Number of nums found so far.
    boolean isPrime;     // Is curNum prime?
    int [] nums = new int [100]; // The list of prime numbers.

    // Initialize 2 into the list of nums.
    nums [0] = 2;
    numsFound = 1;
    curNum  = 2;
    while (numsFound < n)
    {
      curNum++;  // next val to consider ...
      isPrime = true;
      for (int i = 0; i <= numsFound-1; i++)
      {
        if (someMethod (nums[i], curNum))
        {  // Found a divisor, curNum is not prime.
          isPrime = false;
          break;
        }
      }
      if (isPrime)
      {   // save it!
        nums[numsFound] = curNum;
```

```
            numsFound++;
          }
        } // End while

        // Print all the nums out.
        for (int i = 0; i <= numsFound-1; i++)
        {
          System.out.println ("Prime: " + nums[i]);
        }
      } // end SomeClass

      public static void main (String []arg)
      { // Driver method for SomeClass
        int integer = 0;
        if (argv.length != 1)
        {
          System.out.println ("Usage: java SomeClass v1 ");
          return;
        }

        try
        {
          integer = Integer.parseInt (arg[0]);
        }
        catch (NumberFormatException e)
        {
          System.out.println ("Entry must be a integer, using 1.");
          integer = 1;
        }

        SomeClass (integer);
      }
    }
```

For the above SomeClass.java source code, you are required to show your detailed step-by-step solution for each of the following:

a. Draw the control flow graph for the someMethod() method.
b. Show the test requirements for node coverage
c. Show the test requirements for edge coverage
d. Show the test requirements for prime path coverage.
e. Show test paths that achieve node coverage but not edge coverage on the graph.
f. Show test paths that achieve edge coverage but not prime path coverage on the graph

3. Consider the following graph:

Graph IV.

$N = \{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$
$N_0 = \{1\}$
$N_f = \{6\}$
$E = \{(1,2),\ (2,3),\ (2,6),\ (3,4),\ (3,5),\ (4,5),\ (5,2)\}$
$def(1) = def(5) = use(5) = use(6) = \{x\}$
// Assume the use of x in 5 precedes the def
Test Paths:
$t1 = [1,\ 2,\ 6]$
$t2 = [1,\ 2,\ 3,\ 4,\ 5,\ 2,\ 3,\ 5,\ 2,\ 6]$
$t3 = [1,\ 2,\ 3,\ 5,\ 2,\ 3,\ 4,\ 5,\ 2,\ 6]$

a. Draw the control flow graph.
b. List all of the du-paths with respect to *x*. (Note: Include all dupaths, even those that are subpaths of some other du-path).
c. Determine which du-paths each test path tours. Write them in a table with test paths in the first column and the du-paths they cover in the second column. For this part of the exercise, you should consider both direct touring and sidetrips.
d. List a minimal test set that satisfies *all defs* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.
e. List a minimal test set that satisfies *all uses* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.