[Reminder: Two mandatory components of ***each test case***: ***input value(s)*** and ***expected output*** — point(s) will always be **deducted** if any of these is missing]

1. (9 points) Consider the following faulty program. The test input that results in a failure is given.

```
/**
 * Find last index of element
 *
 * @param x array to search
 * @param y value to look for
 * @return last index of y in x; -1 if absent
 * @throws NullPointerException if x is null
 */
 line  1  public static int findLast(int[] x, int y)
 line  2  {
 line  3     if (x == null)
 line  4        throw new NullPointerException();
 line  5     for (int i=x.length-1; i>0; i--)
 line  6     {
 line  7        if (x[i] == y)
 line  8           return i;
 line  9     }
 line 10     return -1;
 line 11   }
 // test: x = [2, 3, 5]; y = 2;
// expected = 0
```

Answer the following questions.

Hint: What is the fault?

1. (1 point) If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.

   - **Input:** x = [2, 3, 5]; y = 3
   - **Expected Output:** 1

   The value 3 is at index 1, which is within the loop's range, so the test case does not involve index 0 and does not reveal the fault.

2. (2 points) If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.

   - **Input:** x = [2, 3, 5]; y = 2
   - **Expected Output:** 0

     The value 2 is at index 0, but the loop misses index 0 and returns -1, incorrectly indicating 2 is not found.

3. (2 points) If possible, give a test case that results in an error, but **not** a failure. If not, briefly explain why not.

   Not applicable here, as the method does not cause errors like exceptions or abnormal terminations; it only produces incorrect results.

4. (2 points) If possible, give a test case that results in a failure. If not, briefly explain why not. Note: for credit, your test case must **not** be the same as the given test case.

   - **Input:** x = [2, 3, 2, 5, 2]; y = 2
   - **Expected Output:** 4

   The last 2 is at index 4, but the method returns -1 due to missing index 0, failing to find the correct last index.

5. (2 points) For the **given** test case

```
// test input: x = [2, 3, 5]; y = 2;
// expected = 0
```

   Describe the **first** error state. Be sure to describe the **complete** state. Note: For grading purposes, use the format introduced in class; refer to the slides and the Fault, Error, Failure activity.

   - **Input:** x = [2, 3, 5]; y = 2
   - **Expected Output:** 0
   - **Actual Output:** -1
   - **Fault:** Loop condition i > 0 skips index 0.
   - **Error:** Index 0 is not checked.
   - **Failure:** Method returns -1 instead of 0.

(9 points) Consider the following faulty program. The test input that results in a failure is given.

```
/**
 * Compute an average
 *
 * @param arr array of numbers
 * @return average of numbers in arr
 * @throws NullPointerException if arr is null
 */
public static double computeAverage(double[] arr)
{
   if (arr == null)
      throw new NullPointerException();

   double sum = 0.0;
   for (int i=0; i < arr.length-1; i++)
      sum += arr[i];

   double average = sum / arr.length;
   return average;
}
// Given test case
//   test input: arr = [90.5, -65.0, 72.25]
//   expected = 32.58
```

Answer the following questions.

1. (1 point) What is the fault? Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code — what the correct version should be.

   Fault: The loop condition i < arr.length - 1 causes the loop to iterate only up to arr.length - 2, excluding the last element of the array from the sum calculation.

   Correct Code: The loop condition should be i < arr.length so that it includes the last element in the sum calculation.

2. (2 points) If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.

   Input: arr = [1.0, 2.0, 3.0]

   Expected Output: 2.0

   The sum of the elements 1.0 + 2.0 = 3.0 (excluding the last element 3.0) results in 3.0. The average is 3.0 / 3 = 1.0. This is not an error state because the code executes without exceptions but calculates an incorrect average due to the fault.

3. (2 points) If possible, give a test case that results in an error, but **not** a failure. If not, briefly explain why not.

   The given code does not lead to errors like exceptions or crashes under normal conditions, only incorrect results. Therefore, there is no test case where an error (like an exception) occurs but not a failure.

4. (2 points) If possible, give a test case that results in a failure. If not, briefly explain why not. Note: for credit, your test case must **not** be the same as the given test case.

   Input: arr = [90.5, -65.0, 72.25]

   Expected Output: 32.58

   The loop excludes the last element 72.25 from the sum calculation. The sum calculated will be 90.5 - 65.0 = 25.5. The average will be 25.5 / 3 = 8.5 instead of the correct average 32.58. This demonstrates a failure because the output is incorrect compared to the expected result.

5. (2 points) For the **given** test case

```
// test input: arr = [90.5, -65.0, 72.25]
// expected = 32.58
```

- **Input:** arr = [90.5, -65.0, 72.25]
- **Expected Output:** 32.58
- **Actual Output:** 8.5

Fault: The loop condition i < arr.length - 1 causes the loop to exclude the last element from the summation.

Error: The last element (72.25) is not included in the sum, so the sum used to compute the average is incorrect.

Failure: The method returns 8.5 instead of the correct average 32.58, leading to an incorrect result.

Complete Error State:

1. **Fault:** Loop condition i < arr.length - 1 excludes the last element from the sum.
2. **Error:** The sum is incorrectly calculated without the last element, resulting in an incorrect total.
3. **Failure:** The method returns 8.5 instead of the correct 32.58.

My experience with software development is limited. As a test designer, I feel most at home. Based on a combination of analytical skills and knowledge of user expectations and system needs, this conclusion can be drawn. A test design involves creating extensive test cases and scenarios that cover many aspects of the program, ensuring that all potential problems are addressed before development. To do this, you must understand both how the software works as well as what the end-user needs. Plus, you must be able to predict how the various components will interact.

Over the years, I've regularly developed comprehensive, effective tests that consider edge cases, user stories, and requirements in both academic and personal projects. Experience in creating comprehensive test plans and scenarios in line with the software's objectives would be highly valuable. Despite the importance of automation, execution, and assessment, each requires a well-thought-out testing strategy. Due to this, having superior test design skills may enhance the effectiveness and efficiency of tests.