Assignment 6 - Testing

Exercise 1

This is a pen and paper exercise. You are given buggy implementations of three methods:

- The first method `indexOf(int[] a, int n)` returns the index of the first occurrence of the integer `n` in the array `a`.
- The second method `average(int[] a)` computes and returns the average of all integers in the array `a`.
- The third method `averageSubarray(int[] a, int l, int r)` computes the average of the values at the positions ranging from `l` (inclusive) to `r` (exclusive) of the array `a`.

```
public static int indexOf(int[] a, int n) {
   if (a == null) {
      throw new IllegalArgumentException("Array is null");
   }
   int index = -1;
   for (int i = 0; i <= a.length; i++) {
      if (a[i] == n) {
         index = i;
         break;
      }
   }
   return index;
}
```

1. Draw the control flow graph of the method `indexOf`.
2. Write a test case that reveals the bug from the method `indexOf`.
3. Write a test suite that achieves 100% statement coverage for `indexOf` and does not find the bug.
4. Is there a test suite that achieves 100% branch coverage for `indexOf` and still misses the bug?

```
public static int average(int[] a) {
   if (a == null) {
      throw new IllegalArgumentException("Array is null");
   }
   if (a.length == 0) {
      throw new IllegalArgumentException("Array is empty");
   }
   int sum = 0;
   for (int i = 1; i < a.length; i++) {
      sum += a[i];
   }
   return sum / a.length;
}
```

5. Write a test case that reveals the bug from the method `average`.
6. Write a test suite that achieves 100% branch coverage for `average` and still misses the bug.

```
public static int averageSubarray(int[] a, int l, int r) {
   if (a != null && 0 <= l && r <= a.length && l <= r) {
      int sum = 0;
      int count = 0;
      while (l < r) {
          sum = sum + a[l];
          count = count + 1;
          l = l + 1;
      }
      return sum / count;
    }
   else {
     return 0;
   }
}
```

7. Mark all coverage measures that guarantee to find the bug. That is, mark each of the following coverage measures if and only if all sets of test inputs that maximize the coverage measure reveal the bug.

   ☐ Statement coverage
   ☐ Branch coverage
   ☐ Loop coverage

8. Imagine we want to apply random testing. Assume that the method is tested with a non-null array of length 100 where the values `a[i]` as well as the parameters `l` and `r` are chosen independently and uniformly

at random from the set $\{0, 1 \ldots, 100\}$. For which inputs do we find the bug? What is the probability of finding the bug?

Exercise 2: Functional vs. Structural Testing

In this exercise, we test an algorithm that solves the *knapsack problem*: Given a set of items, each with a weight and a value, determine the subset of items to pack into your knapsack so that the total weight does not exceed its capacity and the total value is maximized.

A buggy implementation of an algorithm that solves the knapsack problem is given in `Knapsack.java`.

1. Derive a test suite that exercises the functional behaviour of the algorithm. Fix any bugs you find.
2. Measure the branch coverage achieved by the initial test suite provided in `KnapsackTest.java`. Add tests until you get 100% branch coverage. Fix any bugs you find. To measure the branch coverage you can use IDEA Coverage Runner for IntelliJ[1] or EclEmma for Eclipse[2].
3. Discuss the advantages and disadvantages of structural and functional testing.

[1]See `https://confluence.jetbrains.com/display/IDEADEV/IDEA+Coverage+Runner` for instructions.

[2]See `https://www.eclemma.org/` for instructions.