CS 130 – University of California Los Angeles – (Professor Kim)
Class Activity: Test Coverage, Loop Unrolling, Symbolic Execution, Infeasible Path, Test Generation

```
	public static void main (String args[]){
		returnInput(3,true, true, true);
		// (1) what is the statement, branch and path coverage up to
here?
		returnInput(5,false, false, false);
		// (2) what is the statement, branch and path coverage up to
here?
		returnInput(2,false, true, true);
		// (3) what is the statement, branch and path coverage up to
here?

	}

	public static int returnInput(int x, boolean condition1, boolean
condition2,
			boolean condition3) {
		if (condition1) {
			x++;
		}
		if (condition2) {
			x--;
		}
		if (condition3) {
			x = x;
		}
		return x;
	}
```

1. For the above `returnInput` program, create a control flow graph. Each predicate and statement is a node in the control flow graph and each edge represents a control flow from a source node to a sink node.

2. After execution of statements in the `main` program marked as (1), (2), and (3), what is a cumulative statement, branch and path coverage respectively?

```
	public static void main (String args[]){
		int a[] ={3,5,7};
		complexfun(a, 10);
		// (A) what is the statement, branch, and path coverage up to
here?
		int b[] = {5, 6, 9, 11, 15};
		complexfun(b, 4);
		// (B) what is the statement, branch, and path coverage up to
here?
		int c[] = {7, 2, 1, 2, 5, 6};
		complexfun(c, 4);
		// (C) what is the statement, branch, and path coverage up to
here?

		}

	public static int complexfun (int array[], int k) {
		int value = 0;
		for (int i=0; i<2; i++) {
			int a = array[i];
			if (a > k) {
				value = value+a;
			}else {
				value = value-a;
			}
		}
		return value;
	}
```

3. Draw a control flow graph for the above `complexfun` program.

4. What is the maximum number of paths for the above `complexfun` program?

5. What is the statement, branch, and path coverage for the above `complexfun` program at (A), (B), and (C)?

```
public class Paths extends TestCase {

// how many paths does the fun2 have for arbitrary N
// in this exercise, we perform loop unrolling.
// then the number of paths is 2^(# branches) when each branch can be
// executed both true or false.
public static int fun1(int N) {
	int sum = 0;
	for (int i = 1; i <= N; i++) {
		for (int j = 1; j <= Math.pow(3, i); j++) {
			System.out.println("HelloWorld");
			if (new Random().nextInt() % 2 == 0)
				sum++;
		}
	}
	return sum;
}

public static void fun2(int N) {
		int sum = 0;
		Random rand = new Random();
		for ( int i = 1; i <= N; i ++ ){
			for( int j = 1; j <= Math.pow(3,i); j ++){
				if (rand.nextInt() %2 ==0)
					sum++;
			}
			for (int k=1; k<=i; k++) {
				if (Math.pow(2,k) % 2 ==0) sum++;
			}
		}
		System.out.println("N:"+N+ "\tSum:" +sum);
}

public static int infeasiblepath(int x) {
	if (x < 4) {
		return 0;
	}
	int value = 0;
	int y = 3 * x + 1;
	if (x * x > y) {
		value = value + 1;
	} else {
		value = value - 1;
	}
	return value;
```

```
	}

	public void test1() {
		int value = infeasiblepath(3);
		assertEquals(value, 0);
	}

	public void test2() {
		int value = infeasiblepath(4);
		assertEquals(value, 1);
	}

	public void test3() {
		int value = infeasiblepath(5);
		assertEquals(value, 1);
	}
	public static void main (String args[]) {
		fun2(3);
	}
}
```

- How many paths do exist for fun1() for arbitrary N?
- How many paths do exist for fun2() for arbitrary N?
- How many paths do exist for infeasiblepath()?

```
public static int generate_tests_for_this1(int x, int y, int z) {
            if (x < y) {
                  z++;
            } else {
                  z--;
            }
            if (z < 2 * x + 5) {
                  x++;
            } else {
                  y++;
            }
            return y;
      }

public static int generate_tests_for_this2(int x, int y, int z) {
            // a bit time consuming, so we will do this question if we have a
time // otherwise complete it at home.
            for (int i = 0; i < 2; i++) {
                  if (x < y) {
                        z++;
                  } else {
                        z--;
                  }
                  if (z < 2 * x + 5) {
                        x++;
                  } else {
                        y++;
                  }
            }
            return y;
}
```

- Generate test inputs for the above generate_tests_for_this1 so that the tests can achieve 100% path coverage.
- Generate test inputs for the above generate_tests_for_this2 so that the tests can achieve 100% path coverage.