Common faults in algorithms

- Incorrect logical conditions
- Calculation performed in wrong part of construct
- Non-terminating loop or recursion
- Incorrect preconditions for an algorithm
- Not handling null conditions
- Off-by-one errors
- Operator precedence errors
- Use of inappropriate standard algorithms
Numerical faults in algorithms

- Not using enough bits or digits
- Not using enough places before or after the decimal point
- Assuming a floating point value will be exactly equal to some other value
- Ordering operations poorly so errors build up

Other faults in algorithms

- Poor performance with minimal configurations
- Defects handling peak loads or missing resources
- Hardware and software configuration incompatibility
- Defects in crash recovery
- Deadlock and livelock
- Critical races
- Inappropriate resource management
Black-box Testing

• Specification drives test inputs and expected outputs

• No code, design or internal documentation are available

Equivalence classes

• Groups of possible inputs that should be treated similarly
  – Numbers: <0, 0, >0
  – Numbers: <0, 0..1, >1
  – Months: [-∞..0], [1..12], [13..∞]
  – Years: <0, [0..99], >99
  – Years: <0, [0..9999], >9999
Equivalence partition testing

- Test at least one value of every equivalence class for each individual input.
- Test all combinations where one input is likely to affect the interpretation of another input.
- Test random combinations of equivalence classes.

Boundary value testing

- Expand equivalence classes to test values at extremes of each equivalence class.
- Number ranges:
  - minimum, slightly above minimum, nominal or median value, slightly below maximum, and maximum values
  - values slightly and significantly outside the range
public static long gCD(long a, long b)

import ...;
class GCDTest {
    @Test
    void testGCD() {
        assertEquals(gCD(6, 15), 3);
        assertEquals(gCD(15, 6), 3);
        assertEquals(gCD(30, 30), 30);
        assertEquals(gCD(14, 15), 1);
        assertEquals(gCD(15, 14), 1);
    }
}

public static long gCD(long a, long b)

import ...;
class GCDTest {
    @Test
    void testGCD() {
        // either 0
        assertEquals(gCD(0, 1), 0);
        assertEquals(gCD(1, 0), 0);
        // common divisor
        assertEquals(gCD(6, 15), 3);
        assertEquals(gCD(15, 6), 3);
        assertEquals(gCD(30, 30), 30);
        // no common divisor
        assertEquals(gCD(14, 15), 1);
        assertEquals(gCD(15, 14), 1);
        // negative numbers
        assertEquals(gCD(-15, 6), 3);
        assertEquals(gCD(15, -6), 3);
        assertEquals(gCD(-15, -6), 3);
    }
}
public final class DMS {
    private DMS() {} // prevent class instantiation

    public static double toDegrees( String dms ) {
        ...  
    }
}

class DMSTest{
    @Test
    void zerosNspaces() {
        assertEquals(0.0, DMS.toDegrees(" 0° 0' 0" N");
        assertEquals(0.0, DMS.toDegrees("0°0.0000'S"));
        assertEquals(0.0, DMS.toDegrees("0°0.0 ' 0.0000 " W"));
        assertEqls(0.0, DMS.toDegrees("0.0000°E"));
    }
    ...