Solutions for Exercise Sheet 1

Lecture: Advanced Programming and Algorithms – Part I

Author: Anja Rey

Problem 1: Find the Error

a) Compute remainder of integer division

Error: Loop subtracts b until a <= 0, returning negative or zero instead of remainder.

Fix: Stop when a < b.

Corrected Python Code:

```
def get_remainder(a, b):
    while a >= b:
        a -= b
    return a
```

Optimized: Use modulo operator:

```
def get_remainder(a, b):
    return a % b
```

b) Check divisibility by 3

Error: Line 3 always sets result to False, overriding True.

Fix: Use else clause.

Corrected Python Code:

```
def is_divisible_by_three(n):
    return n % 3 == 0
```

c) Compute power m^n

Original code works for positive n but can be improved.

Optimized Python Code:

```
def power(m, n):
    return m ** n
```

d) Sum of squares

Error: range(n) goes from 0 to n-1, but sum should be from 1 to n.

Corrected Python Code:

```
def sum_squares(n):
    total = 0
    for index in range(1, n + 1):
        total += index ** 2
    return total
```

Optimized using formula:

```
def sum_squares(n):
    return n * (n + 1) * (2 * n + 1) // 6
```

Problem 2: Efficiency and Refactoring

a) Running Times:

- get_remainder: O(a/b)
- is_divisible_by_three: O(1)
- power: O(n)
- sum_squares: O(n)

b) Improvements:

- get_remainder: Use modulo operator (O(1))
- is_divisible_by_three: Already optimal
- power: Use operator (O(log n) internally)
- sum_squares: Use formula (O(1))

Problem 4: Example Algorithm

a) Inspecting the provided pseudocode

Given pseudocode:

```
is_divisible_by_three(n):
  if n ≡ 0 mod 3 then
    result ← True
  result ← False
  return result
```

Issues with the pseudocode:

- Missing else branch / incorrect control flow: The False assignment overwrites True.
- Unnecessary variable: Can return boolean directly.
- Ambiguous input domain: Does not specify integer requirement.

Improved pseudocode:

```
is_divisible_by_three(n):
  if n mod 3 == 0 then
    return True
  else
    return False
```

Even cleaner:

```
is_divisible_by_three(n):
  require n ∈ ℤ
  return (n mod 3 == 0)
```

b) What does it do? What happens in the background? What problem does it solve?

The algorithm checks if an integer n is divisible by 3. It solves the decision problem: Given n, decide if $\exists k \in \mathbb{Z}$ such that $n = 3k$.

Background: It uses n mod 3 to compute the remainder. If remainder is 0, n is a multiple of 3.

Complexity: $O(1)$ in word-RAM model; $O(\log n)$ in bit complexity. Memory: $O(1)$.

Alternative approaches: Digit sum method (based on $10 \equiv 1 \bmod 3$) or DFA with 3 states for streaming digits.

c) How can the algorithm and the code be improved? Useful quality criteria

Improvements:

- Use direct return of boolean expression.
- Validate input type and domain.
- Handle edge cases: negatives, zero, non-integers.
- Consider streaming approach for very large numbers.

Quality criteria:

1. Correctness: Meets specification.
2. Clarity: Simple and readable.
3. Robustness: Handles invalid inputs gracefully.
4. Efficiency: O(1) for fixed-width integers.
5. Maintainability: Clear structure and documentation.
6. Portability: Avoid language-specific quirks.
7. Testability: Easy to test with unit and property-based tests.
8. Security: Avoid overflow and unsafe parsing.
9. Documentation: State preconditions and examples.