Question: Could you design a new approach to improve the average-case?

Yes. By checking the most common scenario first (i.e., reordering the `if` conditions so that the "most likely" condition is evaluated first), the average number of comparisons/branches decreases.

If we know that most inputs are greater than 1 (e.g., uniform distribution on {0,1,2,3,4,5}, so 4 out of 6 inputs are indeed >1), we can rearrange the condition checks so that the frequent case is tested first.

```
int Classify(int a) {
  // Check a>1 first:
  if (a > 1)
    return 3;
  else if (a == 0)
    return 1;
  else
    return 2; // covers the case a == 1
}
```

Operation counting:

- **Case a > 1**:
    1. Compare `(a > 1)` → true
    2. Return
    3. That's only 2 operations total.
- **Case a == 0**:
    1. Compare `(a > 1)` → false
    2. Branch
    3. Compare `(a == 0)` → true
    4. Return
    5. About 4 operations.
- **Case a == 1**:
    1. Compare `(a > 1)` → false
    2. Branch

3. Compare `(a == 0)` → false
4. Return (the "else" case)
5. Also 4 operations.

New Average (Uniform over {0,1,2,3,4,5})

- a = 2, 3, 4, 5: 4 inputs, each costs 2 operations
- a = 0, 1: 2 inputs, each costs 4 operations

New Average = (4 × 2 + 2 × 4) / 6 = 16/6 ≈ 2.67.

This is significantly better than the original one 28/6, thereby improving the average cost.

Question: What the average case would be if the input is through 1 to 3?

For the original one:

```
int Classify(int a) {
    if (a == 0) return 1;
    else if (a == 1) return 2;
    else return 3;
}
```

For a = 1: The code does two comparisons, two branches, and one return = 5 operations.

For a = 2 or a = 3: Same path after failing both comparisons, so also 5 operations each.

Average = (5+ 5 + 5) / 3 = 5

For the new one:

```
int Classify(int a) {
  if (a > 1)
    return 3;
  else if (a == 0)
    return 1;
  else
    return 2;
}
```

For a = 1: The code does two comparisons, two branches, and one return = 5 operations.

For a = 2 or a = 3: The code does one comparison, one branch, and one return = 3 operations.

Average = (5+ 3 + 3) / 3 ≈ 3.66

Question: f(x), g(x), h(x) are 3 functions. Try to proof if f(x) = O(g(x)) and g(x) = O(h(x)), then f(x) = O(h(x)).

Let $f(x)$, $g(x)$, and $h(x)$ be there functions (or integer sequences).

Suppose: $f(x) = O(g(x))$. Formally, there exist constants $C_1 > 0$ and x_1 such that

$$|f(x)| \leq C_1 \cdot |g(x)|, \quad \forall x \geq x_1.$$

$g(x) = O(h(x))$. Formally, there exist constants $C_2 > 0$ and x_2 such that

$$|g(x)| \leq C_2 \cdot |h(x)|, \quad \forall x \geq x_2.$$

Prove that $f(x) = O(h(x))$.

Proof

From $f(x) = O(g(x))$, we have $|f(x)| \leq C_1 |g(x)|$.

From $g(x) = O(h(x))$, we have $|g(x)| \leq C_2 |h(x)|$.

Combining the above inequalities:

$$|f(x)| \leq C_1 |g(x)| \leq C_1 \big(C_2 |h(x)|\big) = (C_1 C_2) |h(x)|.$$

Let $C = C_1 C_2$. Also let $x_0 = \max(x_1, x_2)$. For all $x \geq x_0$, we get

$$|f(x)| \leq C |h(x)|.$$

Hence, by the definition of Big-O notation $f(x) = O(h(x))$.

Question: ∀a, b that 1 < a, b ∈ N, proof $\log_a(n) = O(\log_b(n))$

Using the change-of-base formula for logarithms, we have

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)}.$$

Since a > 1 and b > 1, $\log_b(a)$ is a positive constant. Let

$$C = \frac{1}{\log_b(a)} > 0.$$

Thus, for sufficiently large n,

$$\log_a(n) = C \cdot \log_b(n) \le C\,|\log_b(n)|\,.$$

By the definition of Big-O notation,

$$\log_a(n) = O(\log_b(n))\,.$$

Conclusion: Different logarithm bases differ by only a constant factor, so $\log_a(n)$ and $\log_b(n)$ are equivalent in asymptotic complexity.