Problem 5: Dry Run & Analyze - Time and Space Complexity

---

1. Function: `printTriangle(int n)`

```
void printTriangle(int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j <= i; j++)
            System.out.print("*");
}
```

Dry Run for n = 4:

- i = 0 → j = 0 → print 1 star
- i = 1 → j = 0,1 → print 2 stars
- i = 2 → j = 0,1,2 → print 3 stars
- i = 3 → j = 0,1,2,3 → print 4 stars

Total stars printed = 1 + 2 + 3 + 4 = 10

Time Complexity: O(n^2)
Space Complexity: O(1)

---

2. Function: `printPattern(int n)`

```
void printPattern(int n) {
    for (int i = 1; i <= n; i *= 2)
        for (int j = 0; j < n; j++)
            System.out.println(i + "," + j);
}
```

Dry Run for n = 8:

- i = 1 → j = 0 to 7 → 8 iterations
- i = 2 → 8 iterations
- i = 4 → 8 iterations
- i = 8 → 8 iterations

Total Iterations = 4 (outer) × 8 (inner) = 32

Time Complexity: O(n log n)
Space Complexity: O(1)

---

3. Function: `recHalf(int n)`

```
void recHalf(int n) {
    if (n <= 0) return;
    System.out.print(n + " ");
    recHalf(n / 2);
}
```

Dry Run for n = 20:

- recHalf(20) → print 20
- recHalf(10) → print 10
- recHalf(5) → print 5
- recHalf(2) → print 2
- recHalf(1) → print 1
- recHalf(0) → return

Printed values: 20 10 5 2 1
Total recursive calls = 6

Time Complexity: O(log n)
Space Complexity: O(log n)

---

4. Function: `fun(int n)`

```
void fun(int n) {
    if (n == 0) return;
    fun(n - 1);
    fun(n - 1);
}
```

Dry Run for n = 3:

- T(3) = 2×T(2) = 2×(2×T(1)) = 2×2×(2×T(0)) = 2^3 = 8 total calls

Total calls = 8

Time Complexity: O(2^n)
Space Complexity: O(n)

---

5. Function: `tripleNested(int n)`

```
void tripleNested(int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                System.out.println(i + j + k);
}
```

Dry Run for n = 3:

- i: 0 to 2 → 3 iterations
- j: 0 to 2 → 3 iterations
- k: 0 to 2 → 3 iterations

Total Iterations = 3 × 3 × 3 = 27

Time Complexity: O(n^3)
Space Complexity: O(1)