Understanding the RANDOM(a,b) Algorithm and Its Runtime

The Problem

We have an algorithm called RANDOM(a,b) that's supposed to return a random integer between a and b (inclusive). However, this algorithm can only make calls to RANDOM(0,1), which returns either 0 or 1 with equal probability (50% each).

The question is: **What's the expected running time of this algorithm?**

How the Algorithm Works

Let's trace through the algorithm step by step:

Step 1: Calculate Array Size

```
n = ⌈log₂(b - a + 1)⌉
```

- We need enough bits to represent all numbers from a to b
- There are $(b - a + 1)$ possible numbers
- We need $\lceil \log_2(b - a + 1) \rceil$ bits to represent them all

Step 2: Generate Random Bit Arrays

The algorithm then enters a loop that:

1. Creates an array A of length n
2. Fills each position with a random bit (0 or 1) using RANDOM(0,1)
3. Checks if this bit pattern represents a valid number in our range [a, b]
4. If yes, returns that number; if no, tries again

Example

Say we want RANDOM(3,7):

- We have 5 possible numbers: 3, 4, 5, 6, 7
- We need $n = \lceil \log_2(5) \rceil = 3$ bits

- We generate 3-bit patterns until we get one representing 3, 4, 5, 6, or 7

Why This Creates a Geometric Distribution

Each iteration of the while loop has:

- **Success probability (p)**: The probability that our random n-bit pattern represents a valid number in [a, b]
- **Failure probability (1-p)**: The probability we need to try again

This is exactly a **geometric distribution** - we keep trying until we succeed.

Calculating the Success Probability

With n bits, we can represent 2^n different numbers (0 to 2^n-1).
But we only want (b - a + 1) of these numbers.

Therefore: $\mathbf{p = (b - a + 1) / 2^n}$

Expected Running Time Analysis

Expected Number of Iterations

For a geometric distribution: **E[iterations] = 1/p**

So: $\mathbf{E[iterations] = 2^n / (b - a + 1)}$

Work Per Iteration

Each iteration does n calls to RANDOM(0,1), so each iteration costs **n** time units.

Total Expected Time

$\mathbf{E[total\ time] = E[iterations] \times n = (2^n \times n) / (b - a + 1)}$

Substituting n

Since $n = \lceil \log_2(b - a + 1) \rceil$, we have $2^n \approx (b - a + 1)$

Therefore: $\mathbf{E[total\ time] = n = \lceil \log_2(b - a + 1) \rceil = O(\log(b - a))}$

Key Insight

The expected running time is **logarithmic** in the size of the range. This makes sense because:

- We need $\log_2$(range size) bits to represent the range
- On average, about half our random bit patterns will be valid
- So we expect roughly 2 iterations, each taking $\log_2$(range size) time
- Total: O(log(range size))

This is actually quite efficient for generating random numbers in arbitrary ranges!