1. Data Used

Inputs:

Base(a) : Random integers 2, 15, 256

Exponent(b): Small, medium and large values 28, 216, 264

Modulus(n): Large primes or semi-random integers

Ensure edge cases

Precomputed values:

will choose a range, a1 mod n, a2 mod n, a4 mod n, a8 mod n

1. Experiments Done

Baseline Approach:

Implement modular exponentiation using standard Square-and-Multiply (S&M) without optimizations.

Algorithm:

- Convert the exponent to binary.

- Start with result = 1.

- For each bit in the exponent:

- Square the result.

- Multiply by the base if the bit is 1.

- Take modulo n after each step.

Optimized Approach:

Write another function that:  
-Precomputes a2^I mod n for i = 0,1,2,…

-uses these precomputed values instead of recalculating them during the process.

Experiment Parameters:

Run both methods over the same sets of input data.

Vary:

Base, exponent, modulus size.

Table size (precomputing to analyze memory-speed trade-offs)

Performance Metrics:

Time taken for each computation (measured in nanoseconds or microseconds).

Memory usage comparison.

Algorithm complexity analysis for varying input sizes.

3. Validating Results

Correctness:

Verify outputs of both methods against known correct values (e.g., using Python's pow(base, exp, mod) for validation).

Use randomized tests and edge cases (e.g., minimal and maximal values).

Consistency:

Ensure repeatability of experiments over different runs and systems to rule out anomalies.

4. Comparing Results

Execution Time:

Use statistical metrics like average time, median time, and standard deviation to compare the two approaches.

Include performance graphs (e.g., log-log plots of input size vs. time).

Efficiency Gain:

Compute relative speedup as:

Speedup=Time without table / Time with table

Scalability:

Examine how each approach performs as inputs scale (e.g., modulus size increases).

Memory Overhead:

Quantify additional memory requirements for the precomputed tables.

5. Model to Describe Experiments

Mathematical Description

Baseline (Standard S&M):

Given ab mod  n, iterate b in binary and multiply relevant terms.

Given ab mod n, iterate b in binary and multiply relevant terms.

Optimized S&M:

Precompute a2imod  n for i=0,1,…,k, and use lookup tables for faster computation.

Precompute a2imodn for i=0,1,…,k, and use lookup tables for faster computation.

Experimental Model

Input: {a,b,n,k}, where k is the number of precomputed values.

Procedure:

Precompute table for kk values.

Perform modular exponentiation using:

Direct computation (Baseline).

Precomputed lookup (Optimized).

Output: Computation time, memory usage, and correctness validation.

Flowchart

Generate random test cases.

Run both methods:

Record times and outputs.

Compare outputs for correctness.

Log performance metrics.

6. Reporting Results

Tables:

Present data as a table with inputs, execution times, and memory usage for both methods.

Graphs:

Plot execution time vs. input size.

Visualize the speedup factor as a function of the precomputed table size.

Discussion:

Analyze where the optimized approach excels or fails (e.g., for small exponents, precomputations might not be worth it).