An Elementary Method for Fast Modular Exponentiation With Factored Modulus

Anay Aggarwal[1] Manu Isaacs[1]

[1]Jesuit High School, Portland, OR, USA

ISEF, May 2024

Introduction - What is modular exponentiation?

- Whenever one sends a text or an email, it is encrypted by the system used. This is so that hackers with malicious intent cannot retrieve it. When a friend wants to view the message, the system decrypts it so that only they can view it. This system has a special name: a *cryptosystem*. Cryptosystems are crucial for the security of all information systems.

- Many cryptosystems use *modular exponentiation* for the encryption and decryption processes. This is the problem of computing $a^n \pmod{m}$ for positive integers n, m and objects a such that this is defined. Generally, a is a positive integer. This problem is beneficial for cryptosystems because the inverse problem, solving for n in the equation $a^n \equiv d \pmod{m}$, called the *Discrete Logarithm Problem*, is computationally difficult. In other words, it is hard for one to hack the system.

- The current state-of-the-art algorithms do not asymptotically improve the standard repeated squaring method, which exponentiates $a^n \pmod{m}$ in $O(\log m)$ steps, where steps are multiplications modulo m.

Introduction - What do *we* bring to the table?

- We present an extremely applicable fast algorithm for modular exponentiation when the factorization of the modulus is known. In other words, we speed up the encryption and decryption of data.
- A particular cryptosystem for which we potentially do *particularly* fast encryption is Takagi's cryptosystem, a variant of the classic RSA cryptosystem that uses moduli for which our algorithm performs well.
- We additionally present extensions of our algorithm to matrices, linear sequences, and general ring extensions. Our ring extension results also allow us to perform modular exponentiation in $\mathbb{Z}[i]$, the ring of Gaussian Integers.
- Due to new research on using matrices and linear sequences for error-correcting codes, we potentially provide for fast encoding – a key aspect of secure data transmission.

Main Result

Our main algorithm produces the following theorem.

Theorem (Aggarwal-Isaacs, 2024)

Let m be a positive integer with known factorization $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ and let a, n be positive integers such that $\gcd(a, m) = 1$. For each $1 \leq i \leq k$, choose an integer parameter $t_i \in [0, e_i]$. With $\mathcal{T} = \{t_1, t_2, \cdots, t_k\}$, we may compute $a^n \pmod m$ in

$$O\left(\max(e_i/t_i) + \sum_{i=1}^{k} t_i \log p_i\right)$$

steps.

We have a memory-optimized variant that achieves a similar complexity, with an additional term of $O\left(\max(e_i/t_i) \log(\max(e_i/t_i))\right)$. A nice property of our algorithm is that if someone were to come up with a faster algorithm $\mathcal{A}$, our algorithm would be able to metamorphose to use $\mathcal{A}$. In turn, it would be at least as fast as $\mathcal{A}$.

Generalizations

We have similar results for matrices and linear sequences, which make asymptotic improvements to state-of-the-art methods (such as Fiduccia's algorithm).

Theorem (Aggarwal-Isaacs, 2024)

Let $d \in \mathbb{N}$. Let $A \in GL_d(\mathbb{Z}/m\mathbb{Z})$ and $n \in \mathbb{N}$. Then we may compute A^n in

$$O\left(\max(e_i/t_i) + \sum_{j=1}^{k}(t_j - 1)d^2 \log p_j + \sum_{j=1}^{k}\sum_{i=0}^{d-1} \log(p_j^d - p_j^i)\right)$$

matrix multiplications. This quantity is also $O(\max(e_i/t_i) + d^2 \sum t_i \log p_i)$.

Corollary

Let $(u_n)_{n=0}^{\infty}$ be a sequence of elements of $\mathbb{Z}/m\mathbb{Z}$ related with a linear recurrence relation of degree d. Suppose that $\gcd(c_0, m) = 1$, where c_0 is the coefficient of the smallest-indexed u term in the recurrence. Given sufficient initial terms, we may compute any element u_N in the same complexity as the above theorem.

Theoretical Results (General Moduli)

We begin by analyzing the general case, when m is an "average" positive integer. Results of Niven imply that it is optimal to set all $t_i = O(1)$ in this setting. The complexity is then $O(\log \mathrm{rad} m)$, where rad is the largest squarefree divisor of m. The following theorem of analytic number theory shows that we are asymptotically faster than the state-of-the-art in the general case by certain average-order metrics.

Theorem (Aggarwal-Isaacs, 2024)

For any $x \in \mathbb{N}$ and $A > 0$, we have that

$$\sum_{m \leq x} \frac{m}{\mathrm{rad} m} \neq O(x(\log x)^A).$$

This theorem is proven by considering the Dirichlet Series of the function being summed, then comparing to the Dirichlet Series of powers of the Riemann Zeta Function. Using Abel's technique of partial summation and some integral tricks, the result follows.

Theoretical Results (Particular Moduli)

While our algorithm does well in the general case, we shift our focus to particular moduli for which our algorithm runs significantly faster than repeated squaring.

Our algorithm does particularly well when the exponents in the prime factorization of m are large. The following result exemplifies this.

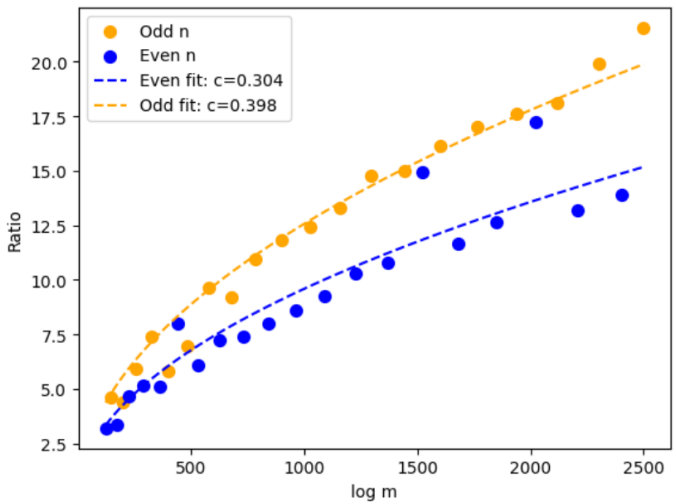
Theorem (Aggarwal-Isaacs, 2024)

Let m be a positive integer with known factorization $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ such that all primes p_i have the same bit length as an integer P, and all e_i are such that $e_i \sim k \log P$. Let a, n be positive integers such that $\gcd(a, m) = 1$. We may then compute $a^n \pmod{m}$ in $O(\sqrt{\log m})$ steps. If the standard algorithm takes T time, our algorithm takes on the order of $T^{5/6}$ time.

This is incredible: in particular cases, we take (on the order of) the *square root* of the number of steps that the current fastest algorithm does!

Empirical Analysis

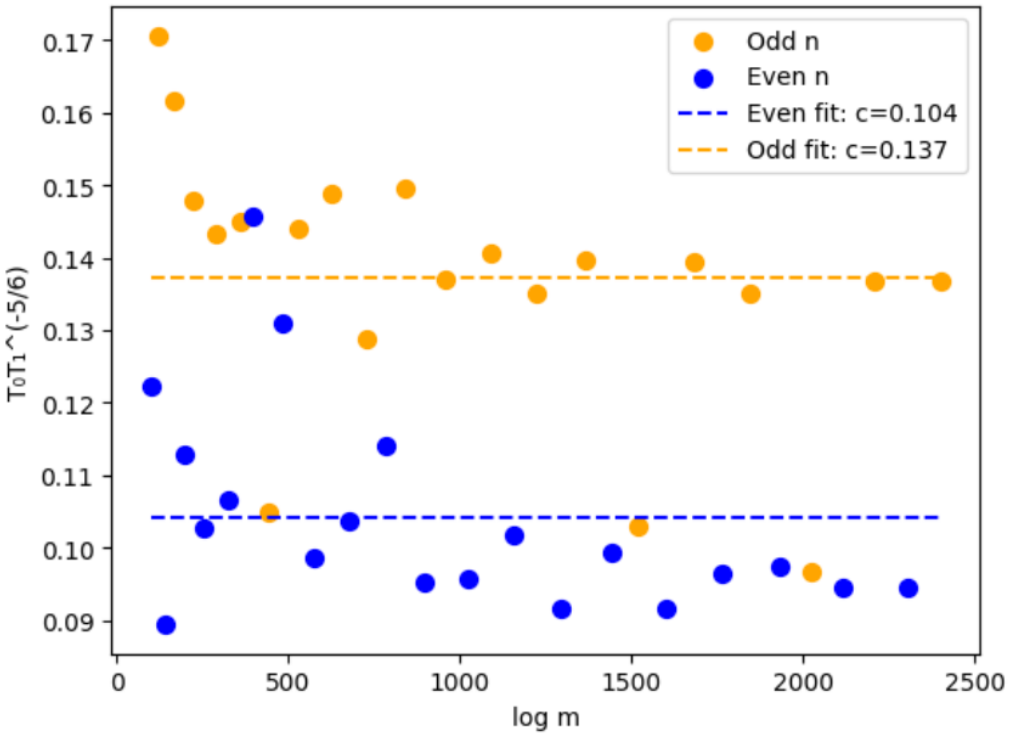
We may verify our results empirically for prime powers by comparing to Python's pow function. For $10 \leq n \leq 50$, we let $m = \mathcal{P}(n)^n$, where $\mathcal{P}(n)$ denotes the smallest prime greater than 10^n (this is arbitrary, m is expected to be "random" to some extent). Letting the runtime of our algorithm be T_1 and the runtime of pow be T_0, we graph the ratio T_0/T_1 against $\log m$.



As predicted, we see a curve in the $y = \sqrt{x}$ family. Note that a different curve fits the even and odd n. This phenomenon is fascinating, and we are still unable to prove it mathematically! Note that the fully-optimized pow function still pales in comparison to our algorithm which doesn't yet use modular multiplication optimizations.

Empirical Analysis

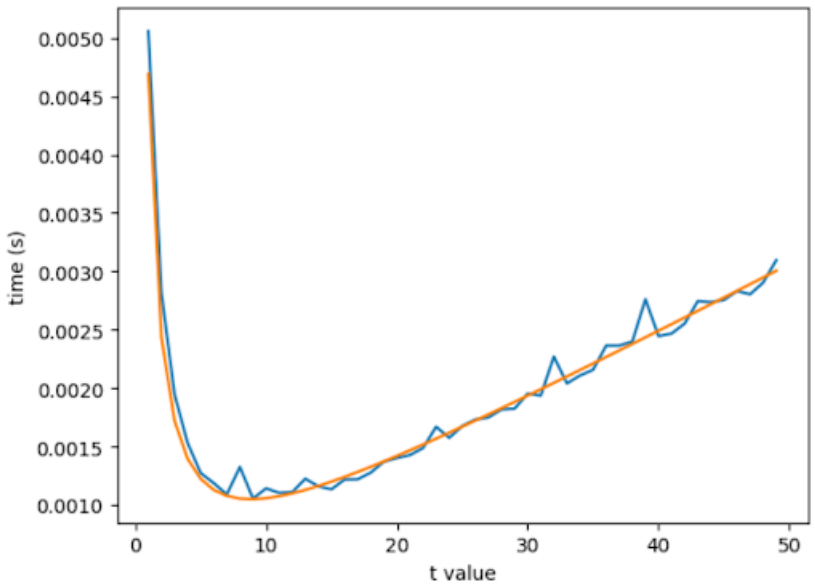
We may also verify our results about time empirically. With the same setup, we graph $T_1 T_0^{-5/6}$ versus $\log m$. As expected, we see this quantity approach a constant value as m grows large.



Again, we see a different curve fit for even and odd n. We hope to explain this phenomenon mathematically.

Empirical Results

While it is optimal to have the input parameters t_i on the order of 1, there is high difference between small values. The following graph shows this in the case of only one input parameter (when m is a prime power).



We vary the input t parameter, plotting against runtime. As the figure shows, there is a large difference between $t = 1$ and $t = 7$, say. This introduces the problem of optimizing the t_i. We have attempted to attack this with various machine learning methods (neural networks, gradient descent methods, etc.).

Conclusions & Future Work

In conclusion, we have a fast algorithm for modular exponentiation with factored modulus. It proves to be fast empirically and theoretically. It extends to more general algebraic structures, and has many applications in the fields of cryptography and secure data transmission.

In the future, we plan to do the following:

- Implement optimizations such as Montgomery Multiplication.
- Mathematically explain phenomena in our empirical analysis, like the even and odd split.
- Empirically analyze our algorithm for general m (rather than just prime powers).
- Make further progress on the front of the problem of optimizing input parameters.

Selected References

[1] Alin Bostan and Ryuhei Mori. *A Simple and Fast Algorithm for Computing the N-th Term of a Linearly Recurrent Sequence*. 2020. arXiv: 2008.08822 [cs.SC].

[2] Charles M. Fiduccia. "An Efficient Formula for Linear Recurrences". In: *SIAM Journal on Computing* 14.1 (1985), pp. 106–112. DOI: 10.1137/0214007. eprint: https://doi.org/10.1137/0214007. URL: https://doi.org/10.1137/0214007.

[3] Ivan Niven. "Averages of Exponents in Factoring Integers". In: *Proceedings of the American Mathematical Society* 22.2 (1969), pp. 356–360. ISSN: 00029939, 10886826. URL: http://www.jstor.org/stable/2037055.

[4] Tsuyoshi Takagi. "Fast RSA-type cryptosystem modulo $p^k q$". In: *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings 18*. Springer. 1998, pp. 318–326.