Classes of Exponents Allowing Fewer Steps than the Square-and-Multiply Algorithm

Summer Research Report

Adity Banerjee
Anjan Sadhukhan
Kirit Maitra
Sohan Karmakar
Under the supervision of **Prof. Mridul Nandy**
Applied Statistics Unit, Indian Statistical Institute

Abstract

The square-and-multiply algorithm is a widely used technique for efficient exponentiation, especially in cryptographic applications. While the algorithm offers a general-purpose solution with logarithmic time complexity, certain classes of exponents allow for further optimization, leading to a reduced number of computational steps. This project investigates such special classes of exponents, focusing on their structure and properties that permit fewer operations than the standard square-and-multiply approach. The study aims to characterize these classes, provide theoretical justification for the step reduction, and demonstrate practical examples where significant computational savings are achieved.

Contents

1 Introduction

Efficient computation of large exponents is a fundamental requirement in many areas of computer science, particularly in cryptography, number theory, and computational mathematics. The square-and-multiply algorithm (also known as exponentiation by squaring) is the de facto method for modular and standard exponentiation, offering a significant improvement over naive multiplication.

The algorithm operates by expressing the exponent in binary form and iteratively performing squaring and multiplication based on the binary bits. For a general exponent with k bits, this method typically requires approximately k squaring operations and up to k multiplications.

However, in certain scenarios, the structure of the exponent allows for a more efficient computation path, reducing the number of necessary steps. Exponents with patterns such as runs of 1's, or exponents expressible with minimal additions (addition-chain optimizations) fall into this category.

This project focuses on identifying and analyzing such classes of exponents. By understanding the mathematical structure of these numbers, we can design optimized exponentiation strategies that outperform the standard algorithm in specific cases.

2 Problem Statement

The standard square-and-multiply algorithm provides a general-purpose method for exponentiation that is efficient for arbitrary exponents. However, for specific classes of exponents with particular structural properties, it is possible to reduce the number of required computational steps, leading to faster and more resource-efficient exponentiation.

The primary objective of this project is to:

- Identify classes of exponents for which step reduction is possible compared to the standard square-and-multiply algorithm.
- Theoretically justify why these exponents allow for such optimizations.
- Provide explicit examples and, where applicable, generalize to families of such exponents.
- Compare these optimized strategies with the standard algorithm to illustrate the efficiency gains.

The ultimate goal is to deepen the understanding of how exponent structure influences computational complexity and to highlight practical cases where these optimizations can be leveraged.

3 Exponents of the form $(2^a - 1)(2^b + 1)$

Note that

$$(2^a - 1)(2^b + 1) = (2^a - 1) \times 2^b + (2^a - 1)$$

Here, we are computing this first by the standard algorithm (square and exponentiate), and then by precomputing g^{2^a-1} using square and exponentiate, squaring it b times, and finally multiplying by g^{2^a-1}, which we already computed.

Example 1: $3 \times 2^k + 3$

$$3 \times 2^k + 3 = 2^{k+1} + 2^k + 2 + 1$$

Standard algorithm steps:

$$k + 1 + 4 - 1 = k + 4$$

Optimized approach:

$$g \xrightarrow{\text{square}} g^2 \xrightarrow{\text{multiply by } g} g^3$$

Then, square g^3 k times to get $g^{3\times 2^k}$, then multiply by g^3:

$$\text{Total steps:} \quad 2 \text{ (for } g^3) + k \text{ (squaring)} + 1 \text{ (final multiply)} = k + 3$$

1 step saved, valid for $k \geq 2$

Example 2: $7 \times 2^k + 7$

$$7 \times 2^k + 7 = 2^{k+2} + 2^{k+1} + 2^k + 2^2 + 2 + 1$$

Standard algorithm steps:

$$k + 2 + 6 - 1 = k + 7$$

Optimized approach:

$$g \xrightarrow{\text{square}} g^2 \xrightarrow{\text{multiply by } g} g^3 \xrightarrow{\text{square}} g^6 \xrightarrow{\text{multiply by } g} g^7$$

Then, square g^7 k times to get $g^{7\times 2^k}$, then multiply by g^7:

$$\text{Total steps:} \quad 4 \text{ (for } g^7) + k \text{ (squaring)} + 1 \text{ (final multiply)} = k + 5$$

2 steps saved, valid for $k \geq 3$

Example 3: $15 \times 2^k + 15$

$$15 \times 2^k + 15 = 2^{k+3} + 2^{k+2} + 2^{k+1} + 2^k + 2^3 + 2^2 + 2 + 1$$

Standard algorithm steps:

$$k + 3 + 7 = k + 10$$

Optimized approach:

$$g \xrightarrow{\text{square}} g^2 \xrightarrow{\text{multiply by } g} g^3 \xrightarrow{\text{multiply by } g} g^4 \xrightarrow{\text{multiply by } g^3} g^7 \xrightarrow{\text{square}} g^{14} \xrightarrow{\text{multiply by } g} g^{15}$$

Total steps:

$$6 \text{ (for } g^{15}) + k \text{ (squaring)} + 1 = k + 7$$

3 steps saved, valid for $k \geq 4$

Example 4: $31 \times 2^k + 31$

$$31 \times 2^k + 31 = 2^{k+4} + 2^{k+3} + 2^{k+2} + 2^{k+1} + 2^k + 2^4 + 2^3 + 2^2 + 2 + 1$$

Standard algorithm steps:

$$k + 4 + 9 = k + 13$$

Optimized approach: First compute g^{15} as above:

$$g^2 \to g^3 \to g^4 \to g^7 \to g^{14} \to g^{15}$$

Then:

$$g^{15} \xrightarrow{\text{square}} g^{30} \xrightarrow{\text{multiply by } g} g^{31}$$

Then, square g^{31} k times to get $g^{31 \times 2^k}$, then multiply by g^{31}:

$$\text{Total steps:} \quad 8 \text{ (for } g^{31}) + k \text{ (squaring)} + 1 = k + 9$$

4 steps saved, valid for $k \geq 5$

Note that in all the above examples we have used the standard square and multiplication method to compute the exponent of the form $(2^a - 1)$ for the optimized approach.Now let us look into the general case:

We want to compute g^x where

$$x = (2^a - 1)\left(2^b + 1\right)$$

for $b \geq a \geq 2$.

$$(2^a - 1)(2^b + 1) = (2^{a-1} + 2^{a-2} + \cdots + 2 + 1)(2^b + 1)$$

Expanding this expression:

$$= 2^{b+a-1} + 2^{b+a-2} + \cdots + 2^b + 2^{a-1} + 2^{a-2} + \cdots + 2 + 1$$

Note that for $b \geq a$, the above is the sum of distinct powers of 2.

Standard Algorithm Steps:
To compute g^x, the number of steps required is:

$$(a+b-1)+(a-1)+a = 3a+b-2$$

Optimized Approach:
We can compute $2^a - 1 = 2^{a-1} + 2^{a-2} + \cdots + 2 + 1$, which requires:

$$(a-1)+(a-1) = 2(a-1)$$

steps using square-and-exponentiation method.
Then squaring b times gives 2^b factor.
Thus,

$$g^{(2^a-1)\cdot 2^b+(2^a-1)} = g^{(2^a-1)\cdot 2^b} \cdot g^{(2^a-1)}$$

Therefore, total number of steps required:

$$2(a-1)+b+1 = 2a+b-1$$

Hence, $(a-1)$ steps saved.

Now take $a = 6$ and consider the computation of $g^{2^6-1} = g^{63}$.
By standard algorithm, number of steps required:

$$2(6-1) = 10$$

Optimized Approach:

$$\begin{aligned}
g &\xrightarrow{\text{square}} g^2\\
g^2 &\xrightarrow{\text{multiply by } g} g^3\\
g^3 &\xrightarrow{\text{multiply by } g^2} g^5\\
g^6 &\xrightarrow{\text{square}} g^{10}\\
g^{12} &\xrightarrow{\text{multiply by } g^5} g^{15}\\
g^{15} &\xrightarrow{\text{square}} g^{30}\\
g^{30} &\xrightarrow{\text{square}} g^{60}\\
g^{60} &\xrightarrow{\text{multiply by } g^3} g^{63}
\end{aligned}$$

Here, 8 steps are required. So, 2 steps saved.

Next, compute:

$$2^7 - 1 = 2(2^6 - 1) + 1$$

i.e., we compute g^{2^6-1} in 8 steps, square once, and multiply by g. So:

$$8 + 2 = 10 \quad \text{steps required}$$

By standard algorithm:

$$2(7 - 1) = 12 \quad \text{steps required}$$

Assume for some $k \geq 6$, we save 2 steps in computing g^{2^k-1} compared to the standard algorithm which requires $2(k-1)$ steps.

Thus, in optimized approach, g^{2^k-1} computed in:

$$2k - 2 - 2 = 2k - 4 \quad \text{steps}$$

For $g^{2^{k+1}-1} = g^{2(2^k-1)} \cdot g$, number of steps in optimized approach:

$$2k - 4 + 2 = 2k - 2$$

And in standard algorithm:

$$2(k + 1 - 1) = 2k$$

Thus, by induction, we conclude that $2^a - 1$ can be computed in $2a - 4$ steps for all $a \geq 6$.

Returning to the computation of $g^{(2^a-1)(2^b+1)}$:

Total steps in optimized approach:

$$(2a - 4) + b + 1 = 2a + b - 3$$

Standard algorithm requires:

$$3a + b - 2$$

Therefore, number of steps saved:

$$(3a + b - 2) - (2a + b - 3) = a + 1$$

This saving is valid for $b \geq a \geq 6$ and is greater than previous savings.

4 Exponents of the form $(2^a - 1)(2^b + 1)^2$

Here we are computing g^x where x=$(2^a - 1)(2^b + 1)^2$

$$(2^a - 1)(2^b + 1)^2 = (2^a - 1)\left(1 + 2^{b+1} + 2^{2b}\right)$$

Expanding this expression:

$$= 1 + 2 + \cdots + 2^{a-1} + 2^{b+1} + 2^{b+2} + \cdots + 2^{a+b} + 2^{2b} + 2^{2b+1} + \cdots + 2^{a+2b-1}$$

For the exponents to be distinct, we require:

$$2b \geq a + b + 1 \implies b \geq a + 1, \quad a \geq 6$$

Number of steps in square-and-multiply:

$$2b + a - 1 + a - 1 + a = 4a + 2b - 2$$

We know that g^{2^a-1} can be computed in $(2a - 4)$ steps for $a \geq 6$.

Thus, in our optimized algorithm, the number of steps required:

$$2a - 4 + 2b + 2 = 2a + 2b - 2$$

Therefore, number of steps saved:

$$(4a + 2b - 2) - (2a + 2b - 2) = 2a$$

We can also save some steps for $a \geq 2$ by similar approach.

5 Conclusion

In both these classes of exponents, leveraging algebraic structure and precomputed terms like g^{2^a-1} reduces the total number of steps required for exponentiation. The improvement becomes more significant for larger values of a and b, demonstrating that these forms of exponents allow for provably more efficient computation compared to the standard square-and-multiply method.

6 Optimized Exponentiation for Special Forms: $(2^k - 1), (2^a - 2^r)$ and

Number of Steps in the General Square and Multiply Method:

Let x be a positive integer of the form $2^k - 1$. The binary representation of x be:

$$x = (\underbrace{111...1}_{k})_2$$

Steps Count:

- Squarings : Always k - 1 squaring operations are needed, because for each bit after the first(from most to least significance), we square the result. Hence we need k-1 squaring operations to evaluate g^x.
- Multiplications : One multiplication is done for every bit that is 1 except the leading bit so the number of multiplications is equal to k -1 .

Hence, to calculate g^x we need total 2(k -1) steps using general squaring and multiplication algorithm.

Example:

k = 6, x = 63

$$g \xrightarrow{\text{square}} g^2 \xrightarrow{\text{square}} g^4 \xrightarrow{\text{square}} g^8 \xrightarrow{\text{square}} g^{16} \xrightarrow{\text{square}} g^{32} \xrightarrow{\text{multiply by } g^{16}} g^{48} \xrightarrow{\text{multiply by } g^8} g^{56}$$

$$\xrightarrow{\text{multiply by } g^4} g^{60} \xrightarrow{\text{multiply by } g^2} g^{62} \xrightarrow{\text{multiply by } g} g^{63}$$

Total Steps $= 10$

Efficient Exponentiation Algorithm for g^k, where k is in the form of $(2^m - 1), m \geq 5$

Algorithm Steps

1. **Initialize Base Powers:**
   Compute and store the following:

$$g^1 = g$$
$$g^2 = g^1 \cdot g^1$$
$$g^3 = g^2 \cdot g^1$$
$$g^5 = g^2 \cdot g^3$$
$$g^7 = g^2 \cdot g^5$$

2. **Create Power List:**
   Maintain an ordered list of known powers:

$$\text{Powers} = [1, 2, 3, 5, 7]$$

3. **Iterative Construction:**
   Repeat until k is reached:

   (a) **Squaring Step:**
   Let the current list for evaluating $2^{m-1} - 1$ be $P = [p_1, p_2, \ldots, p_n]$.
   Compute:

$$q = 2 \cdot p_{n-1} \quad \text{(i.e., square the second-last power)}$$

   If $q \leq k$ and $q \notin P$, compute:

$$g^q = (g^{p_{n-1}})^2$$

   and append q to P.

   (b) **Check for Direct Completion:**
   Let $d = k - q$. If $d \in \{1, 3, 5, 7\}$, compute:

$$g^k = g^q \cdot g^d$$

   and terminate.

   (c) **Fallback Completion via 14 and 1:**
   If $d \notin \{1, 3, 5, 7\}$, do the following:

$$g^{q+14} = g^q \cdot g^{14}$$
$$g^k = g^{q+14} \cdot g^1$$

   and terminate.

Binary Justification

This method is motivated by binary patterns. The second-last power in each squaring step typically has a binary form like:

$$\texttt{11..1}, \texttt{11..10}, \texttt{11..100}, \texttt{11..1000}.$$

Squaring gives us a number of a binary form like

$$\texttt{11..10}, \texttt{11..100}, \texttt{11..1000}, \texttt{11..10000}.$$

For the first three cases remaining difference belongs to $\{1,3,5,7\}$. In case 3, adding 14 (1110) followed by 1 forms a three-step binary cycle to reach a binary string of all 1's (from 1110000 to 1111111).

Examples: Computing g^{2^m-1} for $m = 6,7,8$

Example 1: $g^{63} = g^{2^6-1}$

1. $g^2 = g \cdot g$
2. $g^3 = g^2 \cdot g$
3. $g^5 = g^2 \cdot g^3$
4. $g^7 = g^2 \cdot g^5$
5. $g^{14} = (g^7)^2$
6. $g^{28} = (g^{14})^2$
7. $g^{56} = (g^{28})^2$
8. $g^{63} = g^{56} \cdot g^7$

Total squarings: 4 **Total multiplications:** 4

Example 2: $g^{127} = g^{2^7-1}$

1. Build steps 1–7 as in Example 1 to get g^{56}
2. $g^{112} = (g^{56})^2$
3. $g^{126} = g^{112} \cdot g^{14}$
4. $g^{127} = g^{126} \cdot g$

Total squarings: 5 **Total multiplications:** 5

Example 3: $g^{255} = g^{2^8-1}$

1. Build steps 1–9 as in previous examples to get g^{126}
2. $g^{252} = (g^{126})^2$ (square the second-last term)
3. $g^{255} = g^{252} \cdot g^3$

Total squarings: 6 **Total multiplications:** 5

Binary Insight

The powers used in squaring (e.g., g^{28}, g^{56}, g^{126}) typically have binary forms like:

$$11100, 111000, 1111110$$

Efficiency Comparison with the Standard Algorithm

Standard Square-and-Multiply Method

For computing g^{2^m-1}, the standard binary exponentiation algorithm:

- Performs $m-1$ squarings
- Performs $m-1$ multiplications
- **Total steps:** $2m-2$

Our Algorithm's Efficiency

Using our algorithm:

- We square only the **second-last** term each time
- We fill small remaining gaps using powers like g^1, g^3, g^7, g^{14}
- **Total steps:** $< 2m-2$, improving as m increases

Observed Step Savings

Let $k \in \mathbb{N}$. Then the number of steps saved compared to the standard algorithm is:

Form of m	Steps Saved
$m = 4+3k$	$2k$
$m = 5+3k$	$2k+1$
$m = 6+3k$	$2k+2$

Conclusion

This demonstrates that our algorithm is strictly more efficient than the standard method for computing g^{2^m-1}, and the advantage increases almost linearly with m.

Improved Step Count for $g^{2^k-2^r}$

We refine our analysis of computing exponents of the form:

$$g^{2^k-2^r}, \quad \text{where } 0 < r < k-4$$

Claim: Let $k, r \in \mathbb{N}$ such that $k \geq 6$ and $1 < r < k-4$. Then, our proposed algorithm computes $g^{2^k-2^r}$ in strictly fewer steps than the standard square-and-multiply method. Specifically:

- The standard method requires at most $2k - r - 2$ steps.
- Our method requires at most $\frac{4k}{3} - \frac{r}{3}$ steps

Hence, our algorithm is strictly more efficient in this range of r and k.

Proof. We first recall the identity:

$$g^{2^k-2^r} = \left(g^{2^{k-r}-1}\right)^{2^r}$$

So, we compute $g^{2^{k-r}-1}$ first using our custom algorithm.

As analyzed earlier , the number of steps required to compute g^{2^m-1} using our method is:

$$\text{Steps}(g^{2^m-1}) \leq 2(m-1) - \tfrac{2}{3}m + 2$$

Substituting m = k - r, we get:

$$\text{Steps}(g^{2^{k-r}-1}) \leq 2(k-r-1) - \tfrac{2}{3}(k-r) + 2$$

Then, to compute $g^{2^k-2^r} = \left(g^{2^{k-r}-1}\right)^{2^r}$, we need r additional squaring steps. So total:

$$\text{Total Steps} \leq 2(k-r-1) - \tfrac{2}{3}(k-r) + r + 2$$

Now simplify:

$$= 2k - 2r - 2 - \tfrac{2}{3}(k-r) + r + 2 = 2k - r - \tfrac{2}{3}(k-r)$$

So, we can write:

$$\text{Total Steps} \leq 2k - r - \tfrac{2}{3}(k-r)$$

Compare this with the standard method bound:

$$\text{Standard Steps} = 2k - r - 2$$

Thus, our method is better as long as:

$$2k - r - \tfrac{2}{3}(k-r) < 2k - r - 2 \Rightarrow 2 < \tfrac{2}{3}(k-r) \Rightarrow k - r > 3 \Rightarrow k > r + 3$$

Since we assume $k \geq r + 5$, this condition is satisfied.
Therefore, for $0 < r < k - 4$, our method yields strictly fewer steps.

□

Conclusion

This generalization demonstrates that the algorithm is efficient for computing powers of the form $g^{2^k-2^r}$ when $k \geq 6$ and $0 < r < k - 4$, with the total number of steps being fewer than in the standard method. The step savings grow with k, while the additional squarings in the recursive structure introduce only a constant overhead.

Further Generalization of the Algorithm

We now extend our efficient algorithm to exponents of the form

$$g^{2^k-2^r+m}$$

where $1 < r < k - 4$, $k > 6$, and m belongs to a special class of binary numbers.

Algebraic Decomposition: We further simplify the exponent using the identity:

$$2^k - 2^r + m = 2^r \cdot (2^{k-r} - 1) + m$$

Let $s = k - r$. Then:

- We compute g^{2^s-1}efficiently using our recursive chain;
- Raise it to the power 2^r using r squarings;
- Multiply with g^m, where $m \in \mathcal{M}_s$ where $\mathcal{M}_s$ contains all elements(till second last) to evaluate g^{2^s-1} using our algorithm.

Hence, the entire exponentiation reduces to:

$$g^{2^k-2^r+m} = \left(g^{2^s-1}\right)^{2^r} \cdot g^m$$

Step Savings: Now from previous calculation to evaluate $g^{2^k-2^s}$ using our algorithm, the number of saved steps is linear in k, while the added steps (due to squaring or small additions) are constant. Therefore, the overall gain grows as $k \to \infty$.

Binary Structure of m: We generalize the method for values of m whose binary representation follows:

- $m \in \{1, 10_2, 11_2, 101_2, 111_2, 1110_2, 11100_2, \dots\}$

- These correspond to:
- Specifically: Let $s = k - r$. The binary structure of m depends on $s \mod 3$ as follows:
  - If $s \equiv 0 \pmod 3$: then
$$m = \underbrace{11\ldots1}_{(s-3)}\underbrace{000}_{3}$$
  - If $s \equiv 1 \pmod 3$: then
$$m = \underbrace{11\ldots1}_{s-1}0$$
  - If $s \equiv 2 \pmod 3$: then
$$m = \underbrace{11\ldots1}_{s-2}00$$

This structure ensures that we can reuse intermediate powers from previous steps (e.g., g^7, g^{14}, g^{28}) and apply efficient multiplication chains.

Example: For g^{2^k-3}, we write:

$$2^k - 3 = 2 \cdot (2^{k-1} - 2) + 1$$

This is computable efficiently for $k \geq 7$ using our algorithm.

Conclusion: This generalization provides a systematic method to compute a wide class of exponents using fewer steps than standard binary exponentiation. It reveals a deeper binary structure within exponent chains that can be recursively exploited.

7 Future Work

In this project, we have focused on classes of exponents of the forms $(2^a - 1)(2^b + 1)$ and $(2^a - 1)(2^b + 1)^2$. We have provided theoretical justifications for why these forms allow for a reduced number of computational steps compared to the standard square-and-multiply algorithm.

A natural direction for future work is to generalize these results to exponents of the form:

$$(2^a - 1)(2^b + 1)^n$$

where n is an arbitrary natural number. Exploring such exponents may reveal further algebraic structures that can be exploited to optimize exponentiation. In particular, it would be interesting to identify conditions on a, b, and n under which significant step reductions are achievable, and to derive generalized expressions for the number of steps saved in these cases.

Additionally, from our analysis in the end, it is evident that the number of steps saved grows linearly with the parameter k. A potential area for further research is to precisely characterize the

exact increasing function of parameters like k and r that governs this step reduction. Establishing this function analytically would provide a more complete understanding of the computational advantages offered by these special exponent forms.

Finally, extending these ideas to other structured exponents beyond the powers of 2, or investigating their applicability in modular arithmetic settings relevant to cryptography, would broaden the practical significance of this work.

Acknowledgments

We sincerely thank Prof. Mridul Nandi for his guidance and insightful discussions throughout the summer.