Gas Cost and Algorithmic Complexity in EVM-Based Smart Contracts

Understanding Complexity and Computability in Blockchain Environments

Lucas Kim

Problem Statement

Key Question

How does algorithmic complexity translate to financial cost in blockchain?

Traditional Computing

Complexity affects performance (time/space).

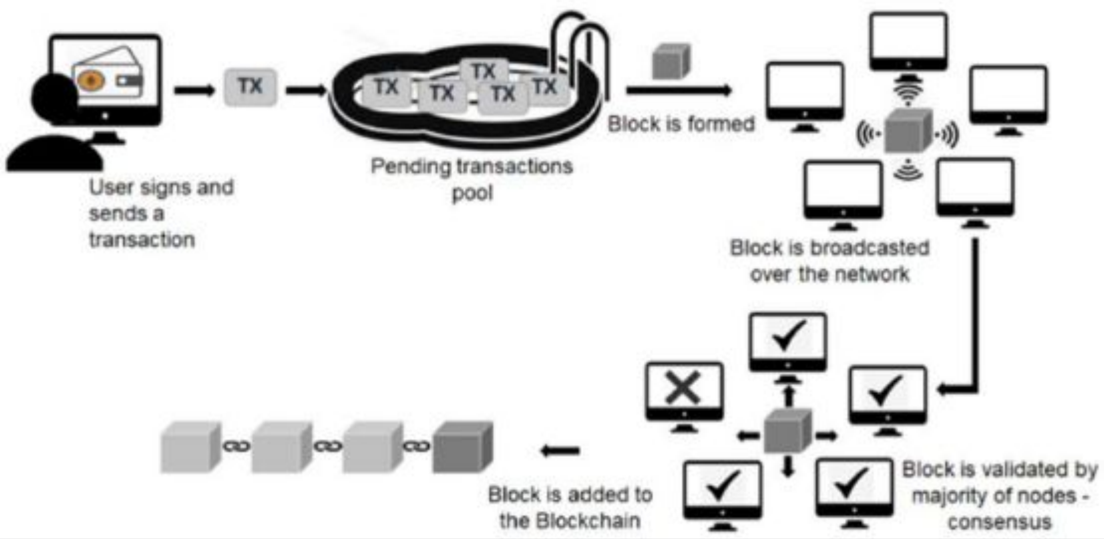
The developer or company bears the cost.

Blockchain Computing

Complexity directly affects financial cost (gas fees).

The user bears the cost.

Unique Paradigm:
Big-O → Economic Cost



Research Objectives

- **Question 1:** How does algorithmic complexity (Big-O) translate into tangible financial cost (Gas) in Ethereum?
- **Question 2:** What specific EVM operations (e.g., storage, memory, loops) are the primary drivers of gas cost spikes?
- **Question 3:** What practical, measurable optimization strategies can developers implement to reduce gas consumption?

Focus Areas

Complexity Analysis, Computability Constraints, Economic Efficiency

Methodology Overview

Experimental Contract Patterns

1. **Loop Complexity:** O(n) vs O(1) operations
2. **Storage vs Memory:** Persistent vs temporary operations
3. **Function Modularity:** Code duplication vs reusable functions
4. **Token Airdrop:** Naive vs optimized batch processing

Measurement Framework

- **Framework:** Hardhat local network
- **Language:** Solidity 0.8.20
- **EVM Version:** Paris

Exp 1: Loop Complexity (O(n) vs O(1))

O(n) Linear Search (Array)

Gas cost scales linearly with array size.

(e.g., ~50,000+ gas for large arrays)

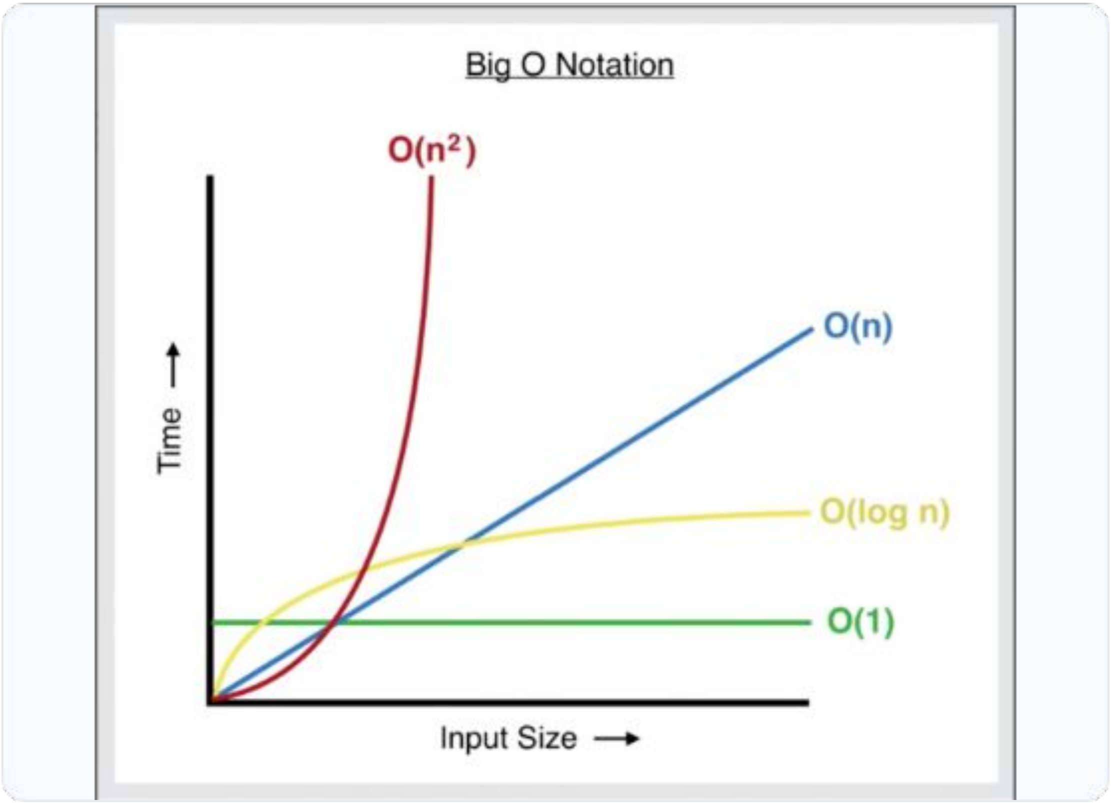
O(1) Constant Lookup (Mapping)

Gas cost is constant (~2,000 gas), independent of data size.

Precomputation Pattern

Trade one-time O(n) cost (303,191 gas init) for repeated O(1) access (~2,000 gas).

Key Finding: Use mappings for constant-time access to avoid unbounded loops.



Exp 2: Storage vs Memory Operations

~6,000 X

More Expensive (Storage vs. Memory)

Storage (Persistent)

- First write: ~20,000 gas
- Subsequent writes: ~5,000 gas
- Reads: ~2,100 gas

Memory (Temporary)

- Writes & Reads: ~3 gas per word

Strategy: Batch operations in memory, then write to storage *once*.

Exp 3: Function Modularity

Code Duplication Approach

Repeated logic in multiple functions.

- Higher deployment cost (larger contract size).
- Avg. Function Gas: **43,716**

Modular Design Approach

Reusable internal functions.

- Lower deployment cost.
- Avg. Function Gas: **43,762**

Key Finding: No Runtime Overhead

The Solidity compiler **inlines** internal functions.
The small gas difference is negligible noise.
Implication: Modular design improves maintainability without a gas penalty.

Exp 4: Token Airdrop Case Study (10 Recipients)

Gas Savings: 168,907 (56.26% Reduction)

Techniques: `unchecked` arithmetic (~20 gas per iteration),
batch event emission, and chunked processing.
This cost difference **compounds** as the number of recipients increases.

Complexity to Gas Cost Mapping

Operation	**Complexity**	**Approx. Gas Cost**	**Notes**
Storage Write (New)	O(1)	~20,000	Extremely expensive.
Storage Write (Update)	O(1)	~5,000	Cheaper, but still very high.
Storage Read / Mapping	O(1)	~2,100	Constant time access.
Memory Write/Read	O(1)	~3	Virtually free by comparison.
Array Iteration (Loop)	O(n)	~10-50 per iter + ops	Cost scales with data size.
Nested Loop	$O(n^2)$	(Cost A × Cost B)	Risk of exceeding block gas limit.

Computability Constraints

Block Gas Limit

A hard cap on computation per block (~30 million gas). Inefficient O(n) or O(n²) algorithms can easily exceed this limit, making them unusable.

Immutability Constraint

Contracts cannot be patched. Optimization and complexity analysis are critical *before* deployment. You cannot fix an inefficient loop once it's live.

Network Resource Constraints

Limited block space and network throughput. Inefficient contracts are "bad neighbors" that contribute to overall network congestion and high fees for everyone.

Key Findings: Six Design Principles

1. **Prefer O(1) over O(n):** Use mappings (O(1)) instead of array iteration (O(n)) for lookups.
2. **Minimize Storage:** Batch operations in memory first, then write to storage once.
3. **Precompute:** Trade a one-time O(n) cost (on-chain or off-chain) for repeated O(1) access.
4. **Use `unchecked` Arithmetic:** Safely save gas in loops (post-Solidity 0.8) where overflow is impossible.
5. **Batch Operations:** Reduce fixed transaction overhead by processing multiple items in one call.
6. **Modularize Code:** Improves readability and maintainability at no extra runtime gas cost.

Conclusion

Key Insight & Contribution

This research establishes a clear, measurable framework linking algorithmic complexity directly to financial cost in blockchain.

These cost-reduction strategies have immediate, practical applications for developers in DeFi and beyond.

Future Research

- Extended complexity analysis (O(log n), O(n log n))
- Development of automated gas cost prediction models
- Creation of static analysis tools to flag gas-inefficient patterns

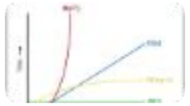
Takeaway: In blockchain,
optimizing for complexity is optimizing for economics.

Thank you

Image Sources

https://www.mdpi.com/mathematics/mathematics-11-02212/article_deploy/html/images/mathematics-11-02212-g001.png

Source: www.mdpi.com



https://miro.medium.com/1*yiyfZodqXNwMouC0-B0Wlg.png

Source: levelup.gitconnected.com

https://img.freepik.com/premium-vector/abstract-futuristic-technology-network-background-with-glowing-blue-lines-nodes-dark-background-connecting-geometric-shapes-data-points-innovation-concepts_1222051-791.jpg

Source: www.freepik.com

https://img.freepik.com/premium-photo/abstract-plexus-blue-geometrical-shapes-connection-web-concept-digital-communication-technology-network-background-with-moving-lines-dots_34629-1247.jpg

Source: www.freepik.com