Verifying Groups in Linear Time

Vanja Stojanović*
vs66277@student.uni-lj.si
University of Ljubljana
Faculty of Mathematics and Physics
Ljubljana, Slovenia

Bor Pangeršič
University of Ljubljana
Faculty of Mathematics and Physics
Ljubljana, Slovenia

Pia Sotlar
University of Ljubljana
Faculty of Computer Science and Informatics
Ljubljana, Slovenia

Klemen Kavčič
University of Ljubljana
Faculty of Computer Science and Informatics
Ljubljana, Slovenia

ABSTRACT

Your abstract goes here.

KEYWORDS

Groups

ACM Reference Format:
Vanja Stojanović, Bor Pangeršič, Pia Sotlar, and Klemen Kavčič. 2025. Verifying Groups in Linear Time. In *Proceedings of Proceedings of the Algorithms Conference '25 (Algorithms Conference '25).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Motivation

Verifying whether a given multiplication table defines a **group** is a fundamental problem in computational algebra with applications in cryptography, error-correcting codes, and symbolic computation. The key challenge lies in efficiently checking the **associativity axiom**, which naively requires testing all possible triplets (a, b, c) to ensure:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c).$$

A brute-force approach would require $O(n^3)$ time, which is impractical for large n. While prior work has improved this to $O(n^2 \log n)$ deterministically or $O(n^2)$ probabilistically, the question remains: **Can we do better?**

Why Linear Time is Non-Obvious. A natural hope might be that associativity could be verified in **subcubic (or even linear) time** by exploiting algebraic structure. However, as demonstrated by the following theorem:

Theorem 1.1 (Local Non-Associativity). *On every set with at least 4 elements, there exists a binary operation that is associative everywhere except on one triplet.*

*Corresponding author.

Algorithms Conference '25, May 20-June 3, 2025, Ljubljana, Slovenia

ACM ISBN 978-1-4503-XXXX-X/25/05
https://doi.org/10.1145/XXXXXXX.XXXXXXX

Proof. Let S be a set with $|S| \geq 4$. Choose distinct elements $a, u, v, w \in S$. Define a binary operation $*$ on S as follows:

$$x * y = \begin{cases} u & \text{if } x = a \text{ and } y = a, \\ v & \text{if } x = a \text{ and } y = u, \\ w & \text{otherwise.} \end{cases}$$

We claim $*$ is associative except for the triplet (a, a, a).
Non-associativity at (a, a, a)**:**

$$(a * a) * a = u * a = w, \quad \text{but} \quad a * (a * a) = a * u = v.$$

Since $w \neq v$, $(a * a) * a \neq a * (a * a)$.
Associativity for all other triplets:
First, let's analyze LHS = $(x * y) * z$. Let $p_1 = x * y$.

- If $(x, y) = (a, a)$, then $p_1 = u$. LHS = $u * z$. Since $u \neq a$, the pair (u, z) is neither (a, a) nor (a, u). So, $u * z = w$ by definition (3).
- If $(x, y) = (a, u)$, then $p_1 = v$. LHS = $v * z$. Since $v \neq a$, the pair (v, z) is neither (a, a) nor (a, u). So, $v * z = w$ by definition (3).
- If $(x, y) \neq (a, a)$ and $(x, y) \neq (a, u)$, then $p_1 = w$. LHS = $w * z$. Since $w \neq a$, the pair (w, z) is neither (a, a) nor (a, u). So, $w * z = w$ by definition (3).

In all possible cases for (x, y), LHS = $(x * y) * z = w$.
Next, let's analyze RHS = $x * (y * z)$. Let $p_2 = y * z$.

- If $(y, z) = (a, a)$, then $p_2 = u$. RHS = $x * u$.
  - If $x = a$, this corresponds to the triplet (a, a, a). As shown before, RHS = $a * u = v$.
  - If $x \neq a$, the pair (x, u) is not (a, a) (since $x \neq a$) and not (a, u) (since $x \neq a$). So, $x * u = w$ by definition (3).
- If $(y, z) = (a, u)$, then $p_2 = v$. RHS = $x * v$. Since $v \neq a$ and $v \neq u$ (as a, u, v, w are distinct), the pair (x, v) cannot be (a, a) (as $v \neq a$) and cannot be (a, u) (as $v \neq u$). So, $x * v = w$ by definition (3).
- If $(y, z) \neq (a, a)$ and $(y, z) \neq (a, u)$, then $p_2 = w$. RHS = $x * w$. Since $w \neq a$ and $w \neq u$ (as a, u, v, w are distinct), the pair (x, w) cannot be (a, a) (as $w \neq a$) and cannot be (a, u) (as $w \neq u$). So, $x * w = w$ by definition (3).

□

This construction shows that:

- **A single inconsistency suffices to break associativity.**

- **No local or sparse testing suffices**—even if almost all triplets are associative, the operation may still fail to be a group.

1.2 Problem Statement

Given an $n \times n$ multiplication table, decide if it represents a valid group.

1.3 Prior Work

- Brute-force: $O(n^3)$ associativity checks.
- Light's $O(n^2 \log n)$ deterministic method.
- Rajagopalan & Schulman's randomized $O(n^2 \log(1/\delta))$.

1.4 Contributions

- First deterministic $O(n^2)$ algorithm.
- Use of **basis sets** and **4-associativity**.
- Reduction in the search for large subgroups by group decomposition.

2 TECHNICAL OVERVIEW

2.1 Testing Group Axioms Efficiently

- Identity: Check $\exists e \in G$ s.t. $e \cdot g = g \cdot e = g$ ($O(n)$).
- Inverses: Find g^{-1} via exponentiation ($O(n \log n)$).

2.2 4-Associativity and Basis Sets

2.2.1 Basis Definition. A set $S \subseteq G$ where $S \cdot S = G$ and $|S| = O(\sqrt{n})$.

2.2.2 Key Lemma. If S satisfies 4-associativity, then G is associative.
Proof Sketch: Expand arbitrary triples into basis elements using consistency of quadruples.

2.3 Reduction to Finding Large Subgroups

The core challenge in this subsection is constructing a *basis* $S \subseteq G$ with $|S| = O(\sqrt{n})$ and $S^2 = G$, which enables efficient associativity testing via 4-associativity. The authors show that this reduces to finding *large subgroups* $H \leq G$ where $|H| \geq \sqrt{|G|}$.

Definition 2.1 (Large Subgroup). A proper subgroup $H \leq G$ is **large** if $\sqrt{|G|} \leq |H| < |G|$.

Definition 2.2 (Group Decomposition). For a group G of size n and parameter ℓ, an **ℓ-decomposition** is a pair (A, B) where:

- $A \cdot B = G$
- $|A| \leq 2\ell$
- $|B| \leq n/\ell$

The reduction proceeds via three main phases:

Phase 1: Recursive Decomposition Given a large subgroup H, we recursively decompose G:

- **Base Case**: If G is cyclic of prime order, use arithmetic progressions.
- **Recursive Case**:
  (1) If $|H| \geq n/(2\ell)$, set A as a left transversal of H, $B = H$.
  (2) Else, recursively decompose H and lift to G via transversals.

Algorithm 1 GroupDecomposition

if $|G|$ is prime **then**
  Return cyclic decomposition
else
  $H \leftarrow \text{LargeSubgroup}(G)$
  if $|H| \geq n/(2\ell)$ **then**
    $A \leftarrow \text{LeftTransversal}(G, H)$
    $B \leftarrow H$
  else
    $(A', B') \leftarrow \text{GroupDecomposition}(H, \ell)$
    $A \leftarrow A'$
    $B \leftarrow B' \cdot \text{RightTransversal}(G, H)$
  end if
end if
return (A, B)

Phase 2: Transversal Computation

Lemma 2.3 (Left Transversal). *A left transversal T of $H \leq G$ can be found in $O(n)$ time by:*

(1) *Greedily selecting representatives from distinct cosets*
(2) *Removing entire cosets after selection*

Phase 3: Basis Construction

Corollary 2.4. *Setting $\ell = \sqrt{n/2}$ yields a basis $S = A \cup B$ with $|S| \leq 2\sqrt{2n} < 3\sqrt{n}$.*

2.4 Reduction to Simple Groups

The algorithm reduces the large subgroup search problem to the case of *simple groups* by:

- Finding normal subgroups efficiently
- Recursively processing quotient groups
- Handling simple groups as base cases

Definition 2.5 (Simple Group). A group G is **simple** if its only normal subgroups are $\{e\}$ and G itself.

2.4.1 Identifying normal subgroups. The algorithm identifies normal subgroups through:
Step 1: Small Generating Set

- Compute a generating set S with $|S| \leq \log |G|$ in $\widetilde{O}(n)$ time
- Uses greedy expansion of non-generator elements

Algorithm 2 Generators

$H \leftarrow \{e\}$
$S \leftarrow \emptyset$
for each $a \in A$ **do**
  if $a \notin H$ **then**
    $S \leftarrow S \cup \{a\}$
    $H \leftarrow \langle S \rangle$
  end if
end for
return S

Step 2: Conjugacy Class Graph

- Build graph where vertices are conjugacy classes
- Edge $C_i \to C_j$ if elements from C_i multiply into C_j
- Compute in $\widetilde{O}(n)$ time using generators

Step 3: Closed Union Detection

- Normal subgroups correspond to closed unions of classes
- Dynamically maintain reachability in the graph

2.4.2 *Recursive Reduction.*

LEMMA 2.6 (6.2). *Given an algorithm for simple groups, LargeSubgroup runs in $O(n^{3/2+\epsilon})$ time for arbitrary groups.*

Algorithm 3 LargeSubgroup

$N \leftarrow$ NormalSubgroup(G)
if $N = G$ **then**
  return LargeSubgroupOfSimpleGroup(G)
else if $|N| \geq \sqrt{n}$ **then**
  return N
else if G/N has prime order **then**
  Find element g of prime order p
  return $\langle g \rangle$
else
  $M \leftarrow$ LargeSubgroup(G/N)
  return $M \cdot N$
end if

2.4.3 *Complexity Analysis.*

- Normal subgroup finding: $\widetilde{O}(n)$ via dynamic graph maintenance
- Recursion depth: $O(\log n)$ (each step reduces problem size)
- Bottleneck: Simple group handling ($O(n^{3/2+\epsilon})$)

THEOREM 2.7. *The reduction to simple groups preserves the overall $O(n^{3/2+\epsilon})$ time complexity when combined with:*

- *$\widetilde{O}(n)$ normal subgroup detection*
- *$O(n^{3/2+\epsilon})$ simple group processing*

2.5 Finding Large Subgroups of Simple Groups

The algorithm handles simple groups through a classification-based approach:

Definition 2.8 (Simple Group Types). Finite simple groups fall into three categories:

- Cyclic groups of prime order
- Alternating groups A_n ($n \geq 5$)
- Groups of Lie type and sporadic groups

2.5.1 *Classification Strategy.*

- **Step 1: Group Identification** - Enumerate possible isomorphisms:

Algorithm 4 EnumerateGroups

for each simple group family f **do**
  Solve $|f(m, q)| = n$ for parameters (m, q)
  if solution exists **then**
    Add (f, m, q) to candidate list
  end if
end for
return candidates

- **Step 2: Type-Specific Handling** - Different approaches per category

2.5.2 *Algorithmic Approaches.* **Case 1: Alternating Groups (A_n)**

- Find the natural subgroup A_{n-1}
- Implementation steps:
  (1) Identify all 3-cycles (elements of order 3)
  (2) Find a set conjugate to $\{(1, 2, k) | 3 \leq k \leq n\}$
  (3) Remove one generator to get A_{n-1}
- Runtime: $\widetilde{O}(n)$

Case 2: Groups of Lie Type

- Key tool: Borel subgroups

  Definition 2.9 (Borel Subgroup). A maximal connected solvable subgroup.

- Implementation:

Algorithm 5 BorelSubgroup

Find Sylow p-subgroup P (char. p of field)
Compute normalizer $B = N_G(P)$
return B

- Construct parabolic subgroup:
  - Find element a such that $P = \langle B, a \rangle$ is large
  - Verify $|P| \geq \sqrt{|G|}$
- Runtime: $O(n^{3/2+\epsilon})$

Case 3: Sporadic Groups

- Finite list of 26 exceptions + Tits group
- Precomputed large subgroups from group theory literature
- Constant-time lookup

2.5.3 *Theoretical Guarantees.*

THEOREM 2.10 (EXISTENCE). *Every non-prime simple group contains a large subgroup.*

PROOF SKETCH.
- Alternating: $|A_{n-1}| \geq \sqrt{|A_n|}$
- Lie type: Parabolic subgroups satisfy size requirement
- Sporadic: Verified case-by-case

□

2.5.4 *Complexity Analysis.*

- Alternating groups: $\widetilde{O}(n)$ (dominated by 3-cycle detection)
- Lie type: $O(n^{3/2+\epsilon})$ (Borel subgroup construction)
- Sporadic: $O(1)$ (table lookup)
- Classification: $O(\log^4 n)$ (parameter solving)

LEMMA 2.11. *The simple group handling preserves the overall $O(n^{3/2+\epsilon})$ time complexity.*

3 CRITICAL ANALYSIS

3.1 Strengths

- Optimal complexity ($\Omega(n^2)$ lower bound).
- Novel techniques: 4-associativity, group decomposition.

3.2 Limitations

3.3 Open Problems

4 FORMAL PROOFS

4.1 4-Associativity Lemma

4.2 Existence of Large Subgroups

Proof that non-prime finite groups have subgroups of size $\geq \sqrt{n}$.

5 CONCLUSION

Summary of results, implications for computational group theory, and future directions.