Electronic Colloquium on Computational Complexity, Report No. 195 (2023)

Verifying Groups in Linear Time

Shai Evra[*] Shay Gadot[‡] Ohad Klein[‡] Ilan Komargodski[‡]

Abstract

We consider the following problem: Given an $n \times n$ multiplication table, decide whether it is a Cayley multiplication table of a group. Among deterministic algorithms for this problem, the best known algorithm is implied by F. W. Light's associativity test (1949) and has running time of $O(n^2 \log n)$. Allowing randomization, the best known algorithm has running time of $O(n^2 \log(1/\delta))$, where $\delta > 0$ is the error probability of the algorithm (Rajagopalan and Schulman, FOCS 1996, SICOMP 2000).

In this work, we improve upon both of the above known algorithms. Specifically, we present a deterministic algorithm for the above problem whose running time is $O(n^2)$. This performance is optimal up to constants. A central tool we develop is an efficient algorithm for finding a subset A of a group G satisfying $A^2 = G$ while $|A| = O(\sqrt{|G|})$.

[*]Einstein Institute of Mathematics, The Hebrew University of Jerusalem, Jerusalem, 9190401, Israel. Email: `shai.evra@mail.huji.ac.il`. Shai Evra is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship and is supported by the Israel Science Foundation (ISF) grant 1577/23.

[‡]School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, 9190401, Israel. Emails: `shay.gadot.42@gmail.com, ohadkel@gmail.com, ilank@cs.huji.ac.il`. Supported in part by an Alon Young Faculty Fellowship, by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643). Ilan Komargodski is the Incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University.

ISSN 1433-8092

Contents

1 Introduction

A Cayley table describes the structure of a finite group by arranging the products of all pairs of group elements in a square table reminiscent of a multiplication table. In this paper we are interested in the problem of testing whether a given $n \times n$ table describes a Cayley table of a group. That is, we are given a non-empty set S and a binary operation $\cdot : S \times S \to S$. The goal is to decide whether $\cdot$ corresponds to a group, i.e., it is associative, an identity element exists, and every element in S has an inverse. Throughout the paper, we let $|S| = n$.

Since the input to the problem consists of an $n \times n$ multiplication table of group elements, it is obvious that every algorithm for the above problem must run in time $\Omega(n^2)$. What about upper bounds?

Testing for the existence of an identity element and verifying that each element has an inverse can both be trivially done in time $O(n^2)$ by brute force. The non-trivial task is testing associativity, i.e., that for every $a, b, c \in S$ it holds that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. A brute-force algorithm would require $\Theta(n^3)$ time[1]. An observation attributed to F. W. Light in 1949 (see [1]) is that it suffices to test associativity only for all triples a, b, c such that $a, c \in S$ but $b \in R$, where R is a set of generators of S. Furthermore, it is possible to deterministically compute a set of generators of size $\lfloor \log n \rfloor$ in $O(n^2)$ time [2]. Thus, we can already improve upon the brute force approach and get a deterministic algorithm running in time $O(n^2 \log n)$.

A classical work of Rajagopalan and Schulman [2] studied the general problem of testing associativity of a given relation. They showed that an arbitrary relation $\cdot$ as above can be tested for associativity using a randomized algorithm that has complexity $O(n^2 \log(1/\delta))$, where $\delta > 0$ is the error probability of the algorithm. This implies a randomized algorithm for group testing with complexity $O(n^2 \log(1/\delta))$ and δ probability of error. This complexity is better than the one implied by Light's observation for rather large values of δ; for instance, if we settle for small constant non-zero probability of error, the complexity of the algorithm is optimal up to constants. However, as mentioned, this algorithm uses randomness and introduces some probability of error.

The above state of affairs leaves the following fundamental problem open:

What is the asymptotic complexity of group testing? In particular, is there a deterministic algorithm running in time $O(n^2)$?

We fully resolve the above question by presenting a deterministic $O(n^2)$ time algorithm for group testing. This complexity is optimal up to constants.

Theorem 1.1. *There exists a deterministic algorithm that receives an $n \times n$ table of elements from $\{1, \ldots, n\}$ as input, and decides whether it is the Cayley table of a group in time $O(n^2)$.*

[1]We assume multiplication in the group can be computed in $O(1)$, and we work in the word-RAM model, where each group elements fits a memory word.

Paper organization. An overview of our algorithmic ideas is given in Section 2. In Section 3 we give necessary terminology. In Section 4 we describe the main framework of our algorithm, and reduce associativity testing to finding a basis of a group. In Section 5 we reduce this task to finding large subgroups for an input group. In Section 6 we reduce this problem to the case of simple groups. Finally, in Section 7 we solve the problem for simple groups, along with relevant introduction to simple groups of Lie type.

2 Technical overview

Our algorithm is derived through a series of reductions, beginning with the original goal of determining whether a given $n \times n$ Cayley table represents a group. Our reduction comprises three primary steps, which are detailed subsequently. Following these reductions, we arrive at the task of identifying sufficiently large subgroups within a given *simple* group. The procedure for achieving this is elaborated upon later. The primary reduction steps are illustrated in Figure 1.

Testing for identity and inverses. Given an $n \times n$ Cayley table, we test by brute force whether there is an identity element and if every element has an inverse. These tasks take $O(n^2)$ time (or $O(n\log(n)$, See the proof of Theorem 4.3). The main challenge is testing for associativity.

Associativity testing by 4-associativity over a basis (Section 4). We simplify the problem of verifying associativity in $(G,\cdot)$ by reducing it to testing of a property we term 4-associativity over a basis of G. A *basis* for $(G,\cdot)$ is a set $S \subseteq G$ such that $|S| = O(\sqrt{n})$ and $S \cdot S = G$ (i.e., every element of G can be written as a product of two elements from S). We define *4-associativity* of S by requiring that every $a, b, c, d \in S$ satisfy

$$((ab)c)d = (ab)(cd) = (a(bc))d = a((bc)d) = a(b(cd)).$$

Our main observation is summarized as follows:

If 4-associativity holds for some basis S for G, then G is associative.

The proof of this observation follows by directly expanding the associativity relation of G, replacing every variable with its representation in $S \cdot S$, and then performing a series of manipulations to establish equality. Despite its simplicity, we believe that this step is interesting on its own, and could be useful elsewhere. In terms of running time, note that testing 4-associativity takes $O(|S|^4) = O(n^2)$ time, since S is a basis of size $O(\sqrt{n})$. Thus, our next task is to find a basis for G, which we solve in $o(n^2)$ time.

Finding a basis by finding a large subgroup (Section 5). A natural question is whether a basis is always guaranteed to exist. Kozma and Lev [3], as well as Finkelstein, Kleitman, and Leighton [4], demonstrated that this is indeed the case. Both proofs rely on the existence of a *large subgroup* for any group of non-prime order. A proper subgroup $H \leqslant G$ is said to be *large* if $|H| \geq \sqrt{|G|}$. The existence of a large subgroup was proved by Lev [5] as well as in [4].

Our reduction between the problems follows an approach similar to [3] and [4], with the exception that our result is algorithmic. We establish that if a large subgroup can be identified in time T, then a basis can be computed in asymptotically the same time complexity.

For this, we *decompose* G as $G = A \cdot B$ with $A, B \subseteq G$ satisfying $|A| \cdot |B| \leq 2|G|$, where A can be of virtually any required size.

To find such a decomposition, we compute a large subgroup $H \subseteq G$. If H has roughly the same size as that of the required B, then we are done by setting $B = H$ and choosing A which contains exactly one element from each left coset of H (this set is called a *left transversal*). If this is not the case, then since H is large, it must be larger than either the required size of A or that of B. Suppose the latter case. Then, we recursively write $H = A' \cdot B'$ with B' having roughly the required size of B. Then, we set $B = B'$ and $A = L \cdot A'$ where L is a left transversal of H in G. This reduction takes $\widetilde{O}(n)$ time.

Reduction to finding a large subgroup only at simple groups (Section 6). The existence of normal subgroups can significantly simplify the computation of large subgroups. If H is a normal subgroup of G, then any large subgroup of G/H can be lifted to a large subgroup of G. Therefore, if a proper normal subgroup can be efficiently identified, the computation of large subgroups is only required if the group is simple, i.e., if no proper normal subgroup exists.

In order to find a normal subgroup, we note that it must be a union of conjugacy classes of G that remains closed under multiplication. To determine such a set of conjugacy classes, we construct a graph whose vertices represent conjugacy classes in G, and a directed edge $C \to C'$ indicates that the minimal normal subgroup containing C must also contain C'. We dynamically add an edge $C \to C'$ if elements a and b found in vertices reachable from C (including C) have a product $a \cdot b$ belonging to C'. Once we exhaust adding edges, the existence of a proper normal subgroup is equivalent to that the graph is not strongly connected.

Naively, it is not clear how to implement this algorithm efficiently. To achieve $\widetilde{O}(n)$ running time for this reduction we use two optimizations:

- For an edge $C \to C'$ not closing a cycle, we focus solely on normal subgroups containing C' instead of considering any normal subgroup containing C, thereby eliminating redundant computations.
- Rather than performing a direct $O(|A|^2)$ check to determine whether a set $A \subseteq G$ is closed to multiplication, we first identify a compact set of generators S for A and then verify $A \cdot S \subseteq A$ in $\widetilde{O}(|A|)$ time.

Finding large subgroups of simple groups (Section 7). At this point, all that is left to complete the group testing algorithm is to provide an $o(n^2)$ algorithm that finds a large subgroup of any simple group. The fact that such a subgroup exists is well known [4, 5], but is rather existential and not algorithmic. Specifically, these proofs employ a case-by-case analysis based on the classification theorem for finite simple groups, demonstrating the existence of a large subgroup for each such a group. Since the constructions are not sufficiently explicit, we make two steps toward an algorithmic result.

- Give a shortlist of *names* of finite simple groups, at least one of which is isomorphic to the input group.
- Enumerate over these names, and based on each case, find a large subgroup.

For the first task, we find all finite simple groups of the same order as that of the input group. We use the fact that the classification of finite simple groups partitions these groups into a finite number of families, where for each family, there is a simple formula for the size the groups it contains. Basically, we solve the relevant equations that arise.

For the second step, we split into cases. If the group is sporadic, we find a large subgroup by a sheer brute-force (enumerating all subsets, searching for a large subgroup). If the group is an alternating group A_m, we present a method to find the large subgroup A_{m-1} using non-explicit oracle queries to group multiplication. Finally, if the input group is a simple group of Lie type, we use the constructions of [4, 5], together with algorithmic optimizations to fit in the required $o(n^2)$ time complexity.

Remark 2.1 (Model of computation)**.** All algorithms in this paper are assumed to run in the standard RAM model of computation. What happens when we consider a weaker model of computation? In fact, except for the 4-associativity test, our algorithm runs in $O(n^{3/2+\epsilon})$ time. Hence, the running time of our algorithm will be determined by the running time of the 4-associativity test, as long as the simulation of RAM is possible with a reasonable overhead (so that the emulation of a $O(n^{3/2+\epsilon})$ RAM algorithm does not exceed $O(n^2)$).

Due to this $O(n^{1/2-\epsilon})$ slack, all algorithms that we present assume that a black-box group is given as input, where group operations can be performed in $O(1)$ time.

3 Preliminaries

For an integer $n > 0$, $[n]$ stands for the set $\{1, 2, \ldots, n\}$. When an algorithm is said to run in $O(n^{\alpha+\epsilon})$ time, without specifying ϵ, it means that for all $\epsilon' > 0$ there exists a constant $c(\epsilon') < \infty$, so that the running time of the algorithm is $\leq c(\epsilon')n^{\alpha+\epsilon'}$. Time complexity of $\widetilde{O}(f(n))$ incdicates a running time of $O(f(n)\log(f(n))^C)$ for some universal constant $C < \infty$. In this paper, n is typically the size of the group under consideration.

Setting (groups). We regard groups via their Cayley tables, which comprises an $n \times n$ matrix of group elements. We assume for simplicity that the elements of the group are indexed by $1 \ldots n$, and the table contains only elements from $[n]$ (if the matrix contains other elements, then it does not correspond to a group). The size of the group n is also called the order of the group. We denote the identity element of a group G by e_G. When there is only one group in the context, we may abbreviate this notation to e.

Definition 3.1 (Multiplication Table)**.** *Given a set G of size n along with a binary operation $\cdot \colon G \times G \to G$, and an arbitrary permutation $\sigma \colon G \to [n]$, a multiplication table of G is defined as the function $T \colon [n] \times [n] \to [n]$ satisfying*

$$T(i, j) = \sigma(\sigma^{-1}(i) \cdot \sigma^{-1}(j)).$$

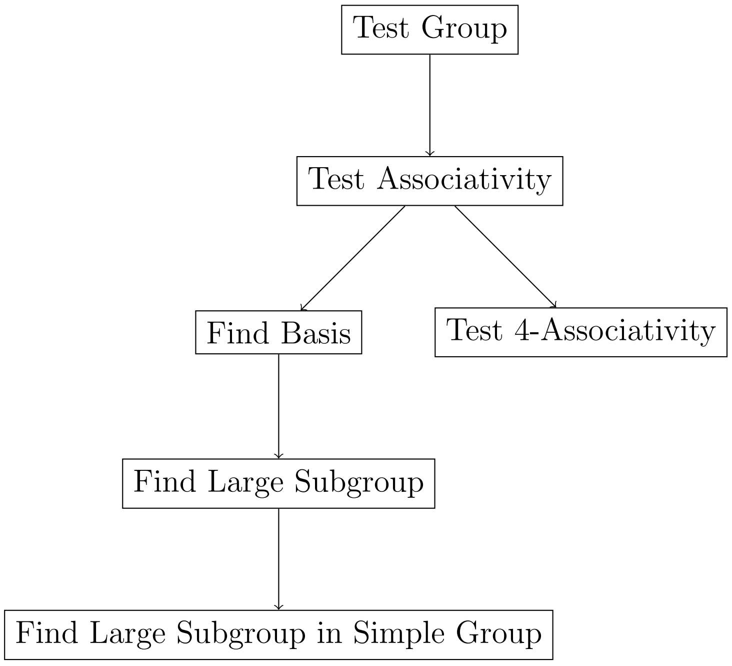

Figure 1: Main steps in our reduction from testing whether a given Cayley table corresponds to a group to finding a large subgroup in a given simple group.

Input format. Most of the algorithms described in the paper receive a set of group elements along with oracle access that enables the multiplication of two group elements in $O(1)$ time.

4 Associativity testing via 4-associativity

In this section we present the main framework of our algorithm. Essentially, given a set G pretending to be a group, we find a *basis* $S \subseteq G$ such that $S \cdot S = G$ and $|S| = O(\sqrt{|G|})$. We then verify that G is indeed associative by merely checking that S is 4-associative, in $O(|S|^4) = O(|G|^2)$ time. The following definitions are due.

Definition 4.1 (Basis). *Let G be a set with a binary operation $\cdot \colon G \times G \to G$. A subset S of G is said to be a **basis** of G if $S \cdot S := \{s_1 \cdot s_2 \mid s_1, s_2 \in S\} = G$ while $|S| \leq 3\sqrt{|G|}$.*

Definition 4.2 (4-associativity). *An operation $\cdot \colon G \times G \to G$ is said to be 4-**associative** over a set S with $S \subseteq G$, if for all $a, b, c, d \in S$ we have*

$$((ab)c)d = (ab)(cd) = (a(bc))d = a((bc)d) = a(b(cd)).$$

The main theorem of this section is stated next. It is a reduction from verifying that G is a group to finding a basis of the (proposed) group.

Theorem 4.3. *Let $\mathcal{A}$ be an algorithm that receives a group G of size n as input, and returns a basis $S \subseteq G$ such that $S \cdot S = G$ and $|S| = O(\sqrt{n})$ in time complexity of $O(n^2)$.*

Then, there exists an algorithm $\mathcal{B}$ that receives a set G along with a binary operation $\cdot \colon G \times G \to G$, and returns whether G is isomorphic to a group in $O(n^2)$ time.

While verifying the existence of an identity element and inverses for all elements is straightforward in time linear in the input size, the main challenge is checking associativity. The following lemma formalizes the correctness of the above-mentioned reduction from testing associativity in G to testing the 4-associativity of a basis S.

Lemma 4.4. *Let $\cdot : G \times G \to G$ be a binary operation on a set G. Further suppose that $S \subseteq G$ satisfies $S \cdot S = G$. If $\cdot$ is 4-associative over S, then $\cdot$ is associative over G.*

Proof. We verify the associativity of $\cdot$ over G. Let $\alpha, \beta, \gamma \in G$. By assumption, there exist elements $a, b, c, d, e, f \in S$ such that

$$\begin{aligned} \alpha &= ab, \\ \beta &= cd, \\ \gamma &= ef. \end{aligned}$$

In order to show G is associative, we must prove

$$((ab)(cd))(ef) = (\alpha \cdot \beta) \cdot \gamma = \alpha \cdot (\beta \cdot \gamma) = (ab)((cd)(ef)). \tag{1}$$

Since $G = S \cdot S$, we can write

$$b(cd) = uv, \quad a(uv) = wx, \quad (cd)e = st, \quad (st)f = pq, \quad (b(cd))e = yz,$$

with $p, q, s, t, u, v, w, x, y, z \in S$. Using this, and 4-associativity of $\cdot$ on S we get that

$$\begin{aligned} ((ab)(cd))(ef) &= (a(b(cd)))(ef) = (wx)(ef) = ((wx)e)f \\ &= ((a(uv))e)f = (a((uv)e))f = (a((b(cd))e))f. \end{aligned} \tag{2}$$

Completely symmetrically, if we reflect all parentheses, we get

$$\begin{aligned} (ab)((cd)(ef)) &= (ab)(((cd)e)f) = (ab)(pq) = a(b(pq)) \\ &= a(b((st)f)) = a((b(st))f) = a((b((cd)e))f). \end{aligned} \tag{3}$$

Finally, we show that the right hand sides of (2) and (3) are equal, to deduce (1):

$$(a((b(cd))e))f = (a(yz))f = a((yz)f) = a(((b(cd))e)f) = a((b((cd)e))f).$$

This completes the proof. ☐

Proof of Theorem 4.3. **The algorithm.** CheckGroup proceeds by computing a basis S for G using $\mathcal{A}$. Since $\mathcal{A}$ is designed for groups, and G might not be a group, applying $\mathcal{A}$ to G might run for too long, in which case we determine that G is not a group. We set a time limit so that if G is a group then the execution of $\mathcal{A}$ will terminate successfully. Then, we test that S is indeed a basis for the set G by definition. Once again, if G is a group then it will pass this test.

Subsequently, we check that the multiplication is 4-associative over S. By Lemma 4.4, G passes this test for the verified basis S if and only if G is associative.

Algorithm 1: CheckGroup

```
Input: A multiplication table $T\colon [n] \times [n] \to [n]$, an algorithm $\mathcal{A}$.
Output: A Boolean indicating whether $G$ is a group or not.
1  Emulate the run of $S \leftarrow \mathcal{A}(G)$, where $G := [n]$ is regarded as a group, with
     multiplications defined using the multiplication table.
2  if running time of $\mathcal{A}(G)$ exceeds the expected $\mathcal{T}(A(G))$ time then
3      return false                                   ▷ The execution of $\mathcal{A}$ failed
4  end
5  if $|S| > 3\sqrt{|G|}$ or $S^2 \neq G$ then
6      return false                                   ▷ The execution of $\mathcal{A}$ failed
7  end
8  for $(a, b, c, d) \in S^4$ do
9      if not $(((ab)c)d = (ab)(cd) = (a(bc))d = a((bc)d) = a(b(cd)))$ then
10         return false                               ▷ Not associative
11     end
12 end
13 if $\forall e \in G\colon \exists g \in G\colon \{e \cdot g, g \cdot e\} \neq \{g\}$ then
14     return false                                   ▷ No identity element
15 end
16 if $\exists a \in G\colon \forall b \in G\colon (a \cdot b) \cdot a \neq a$ then
17     return false                                   ▷ No inverse
18 end
19 return true
```

Once we confirm G is associative, we perform brute-force tests for having an identity element and inverses for all elements. This determines whether G is a group.

Time complexity. In lines 1-4, we let the emulation of $\mathcal{A}$ to run for $O(n^2)$ time. We reach line 5 only if $\mathcal{A}$ succeeded, meaning $|S| = O(\sqrt{n})$, and hence the time of computing S^2 is $O(n)$. In lines 8-12, we enumerate all sets of four elements of S in $O((n^{1/2})^4) = O(n^2)$ time. In lines 13-18, as mentioned before, we test for the existence of an identity and inverse elements, both of which are done by brute-force in $O(n^2)$ time. Overall, the time complexity is $O(n^2)$, as desired.

We note that identity, if exists, can be found (and then tested) in $O(n)$ time by searching for any element $e \in G$ satifying $e \cdot e = e$. Furthermore, the inverse of g, if exists, can be found (and tested) in $O(\log(n))$ by computing g^{n-1} using exponentiation by squaring (relying on associativity). $\square$

5 Reduction to finding large subgroups

We show that the problem of finding a basis for a group G can be efficiently reduced to the problem of finding a *large* subgroup of G. In other words, we give an algorithm that uses an efficient oracle for finding large subgroups to find a basis for any group G. The notion of *large* subgroups is defined as follows.

Definition 5.1 (Large subgroup)**.** *A subgroup H of G is said to be* ***large*** *if $\sqrt{|G|} \leq |H| < |G|$.*

The existence of a basis has been proved in [3] by Kozma and Lev, using the existence of a large subgroup, which was proved in [4] and [5] for groups of non-prime size. We present an algorithmic version of [3] to find a basis for a group. The algorithm uses the following notion of decomposition of a group.

Definition 5.2 (Group Decomposition)**.** *Let G be a group of size n and let ℓ be a real number. An ℓ-decomposition of G is a pair of subsets $A, B \subseteq G$ with $A \cdot B = G$ and $|A| \leq 2\ell$ and $|B| \leq n/\ell$.*

The main result of this section is stated next.

Lemma 5.3. *If there exists an algorithm LargeSubgroup that finds a large subgroup of any group of non-prime order n in $O(n^{3/2+\epsilon})$ time, then there exists an algorithm GroupDecomposition that finds an ℓ-decomposition ($1 \leq \ell \leq n$) of any group G of size n, and whose running time is $O(n^{3/2+\epsilon})$.*

An immediate corollary of Lemma 5.3 is an efficient algorithm for computing a basis for a group. Given a group G of size n, applying algorithm GroupDecomposition of Lemma 5.3 to G, with $\ell = \sqrt{n/2}$ we obtain $A, B \subseteq G$ with $A \cdot B = G$ while $|A| \leq \sqrt{2n}$ and $|B| \leq \sqrt{2n}$. Then, $S = A \cup B$ is a basis for G because $S \cdot S = G$ and $|S| \leq 2\sqrt{2n} < 3\sqrt{n}$.

Corollary 5.4. *If there exists an algorithm LargeSubgroup that finds a large subgroup of any group of non-prime order n in $O(n^{3/2+\epsilon})$ time, then there exists an algorithm $\mathcal{C}$ that finds a basis for an input group, whose running time is $O(n^{3/2+\epsilon})$.*

The rest of this section is devoted to the proof of Lemma 5.3.

5.1 Computing transversals

The algorithm we present for Lemma 5.3 requires computing the cosets of G with respect to a subgroup $H \leqslant G$. Recall that the left cosets of a group are the sets of the form gH, where g is any element of the group. The number of left cosets is called the *index* of H in G, and is denoted $[G : H] = |G|/|H|$. A set containing exactly one element from each left coset is called a *left transversal*.

Definition 5.5 (Left Transversal). *Let H be a subgroup of a group G. A left transversal of H in G is a subset $T \subseteq G$ such that $G = T \cdot H := \bigcup_{t \in T} tH$ and $|T| = [G : H]$.*

We note that the analogous definition of a *right transversal* T' means that $G = H \cdot T'$ while $|T'| = [G : H]$. Next, we present an algorithm for finding left transversals. An analogous algorithm can be used to find right transversals.

Lemma 5.6. *There exists an algorithm* LeftTransversal *which receives a group G of size n, along with a subgroup $H \leqslant G$, and returns a left transversal of H in G in $O(n)$ time.*

Proof. **The algorithm.** The LeftTransversal algorithm operates in a greedy manner. Initially, it assigns the entire set of group elements to a set A. Subsequently, it selects an arbitrary element $g \in A$ and removes all elements belonging to the left coset of g from A. This process is iteratively applied, as long as A is nonempty.

Algorithm 2: LeftTransversal

```
Input: A group $G$ of size $n$ and a subgroup $H \leqslant G$
Output: A left transversal of $H$ in $G$
1 $R \leftarrow \emptyset$
2 $A \leftarrow G$
3 while $A \neq \emptyset$ do
4     Pick $g \in A$ arbitrarily
5     $R \leftarrow R \cup \{g\}$
6     $A \leftarrow A \setminus gH$
7 end
8 return $R$
```

Time complexity. We implement A as an array of bits indicating the membership of any $g \in G$ to A as in Algorithm 2. This renders each loop iteration of LeftTransversal to run in $O(|H|)$ time. Since the number of iterations is the number of cosets $[G : H] = |G|/|H|$, the total running time is $O(n)$. This is in addition to the selection of elements $g \in A$, which is done in linear time by scanning the array once and is shared across all iterations. ☐

5.2 Computing group decompositions

In this section we prove Lemma 5.3. Recall that here we assume the existence of an algorithm $\mathcal{A}$ that finds a large subgroup of any group (of non-prime order) in $O(n^{3/2+\epsilon})$ time. We derive an algorithm for finding an ℓ-decomposition A, B of any group G in the same asymptotic time. The construction of A, B depends on ℓ, $n = |G|$ and the size of the subgroup H found by running $\mathcal{A}(G)$.

Proof of Lemma 5.3. **The algorithm.** GroupDecomposition operates recursively. There are two base cases:

- If G has prime order n, then G is effectively the cyclic group $\mathbb{Z}_n$. In this case we denote $q = \lfloor n/\ell \rfloor$, and use that any number $0, \dots, (n-1)$ may be written as $a \cdot q + b$ for $b \in \{0 \dots (q-1)\}$ and $a \in \{0 \dots \lfloor n/q \rfloor\}$. Correspondingly, we use the decomposition $A = \{q \cdot a \,|\, a \leq \lfloor n/q \rfloor\}$ and $B = \{b \,|\, b < q\}$. We check that A, B is an ℓ-decomposition. First, $|B| = q \leq n/\ell$. Moreover, if $q = 1$ then $|A| = n$, and by definition of $q = \lfloor n/\ell \rfloor$ this means $|A| < 2\ell$. If $q > 1$ then $|A| = \lfloor n/q \rfloor + 1$. Since $n - q\ell < \ell$ then $n/q < \ell + \ell/q$ and $|A| \leq \lfloor \ell + \ell/q \rfloor + 1$. If $\ell \geq 2$ then we get $|A| \leq \ell + \ell/q + 1 \leq \ell + \ell/2 + 1 \leq 2\ell$. If $\ell < 2$ then $q \geq n/2$. Since n is prime then in fact $q > n/2$ (if $n = 2$, recall $q > 1$) and $|A| = \lfloor n/q \rfloor + 1 = 2 \leq 2\ell$.

- If G has a large subgroup H with $n/(2\ell) \leq |H| \leq n/\ell$. In this case, we output $A = \mathsf{LeftTransversal}(G, H)$ and $B = H$, which is valid since $|A| \leq 2\ell$.

If these conditions are not met, we find a large subgroup $H \leqslant G$, and split into two cases:

- If $|H| > n/\ell$, then we look for a decomposition A', B' of H with $|B'| \leq n/\ell$ and correspondingly $|A'| \leq 2|H|/(n/\ell) = 2\ell'$ with $\ell' = \ell|H|/n$. Note $1 \leq \ell' \leq |H|$ which is required in order to guarantee an ℓ'-decomposition of H, as stated in Lemma 5.3. We compute $T = \mathsf{LeftTransversal}(G, H)$ and output the decomposition $A = T \cdot A'$ and $B = B'$. This is a decomposition since

$$G = T \cdot H = T \cdot (A' \cdot B') = (T \cdot A') \cdot B'.$$

  We check (A, B) is an ℓ-decomposition: $|B| = |B'| \leq |H|/\ell' = n/\ell$ and $|A| = |T| \cdot |A'| \leq [G : H] \cdot 2\ell' = 2\ell$.

- If $|H| < n/(2\ell)$, we then we look for an ℓ-decomposition A', B' of H with $|A'| \leq 2\ell$ and $|B'| \leq |H|/\ell$. We claim $\ell \leq |H|$ in order to justify the existence of an ℓ-decomposition of H. To see this, recall H is a large subgroup and hence $\sqrt{n} \leq |H| < n/(2\ell)$, that is $2\ell \leq \sqrt{n}$.

  Finally, $A = A'$ and $B = B' \cdot \mathsf{RightTransversal}(G, H)$ is an ℓ-decomposition of G similarly to the previous case.

Time complexity. The GroupDecomposition algorithm exhibits a recursive structure. However, a recursive call is employed only once, and applied to a subgroup H of size $\leq n/2$, ensuring that the recursive calls do not significantly impact the overall running time. All operations within the algorithm exhibit a linear time complexity, except for the call to the $\mathsf{LargeSubgroup}(G)$ subroutine, which dominates the time complexity and and dictates the time complexity of $O(n^{3/2+\epsilon})$. □

Algorithm 3: GroupDecomposition

```
Input: A group $G$ of size $n$, and a real number $\ell$ satisfying $1 \leq \ell \leq n$.
Output: Subsets $A, B \subseteq G$ satisfying $G = A \cdot B$ and $|A| \leq 2\ell$, $|B| \leq 2n/\ell$.
Setting: $\mathcal{A}$ is an algorithm that finds a large subgroup in any group.
if $n$ is prime then
    $g \leftarrow$ Any non-identity element of $G$
    $q \leftarrow \lfloor n/\ell \rfloor$
    $A \leftarrow \{0, 1, 2, \ldots, \lfloor n/q \rfloor\} \cdot (q \cdot g)$
    $B \leftarrow \{0, 1, 2, \ldots, q-1\} \cdot g$
    return $A, B$
end
$H \leftarrow$ LargeSubgroup($G$)
if $|H| > n/\ell$ then
    $A', B' \leftarrow$GroupDecomposition($H, \frac{\ell|H|}{n}$)
    $A \leftarrow$ LeftTransversal($G, H$) $\cdot A'$
    $B \leftarrow B'$
else if $|H| \geq n/(2\ell)$ then
    $A \leftarrow$ LeftTransversal($G, H$)
    $B \leftarrow H$
else
    $A', B' \leftarrow$GroupDecomposition($H, \ell$)
    $A \leftarrow A'$
    $B \leftarrow B' \cdot$ RightTransversal($G, H$)
end
return $A, B$
```

6 Reduction to simple groups

In the following, we efficiently reduce the problem of finding a large subgroup for an **arbitrary** finite group to the problem of finding such a subgroup for a finite **simple** group. We recall the definitions of a simple group and a normal subgroup first, and then state our theorem.

Definition 6.1 (Simple group and normal subgroups)**.** *A (nontrivial) group is said to be* simple *if its only normal subgroups are the trivial group and the group itself. A subgroup N of the group G is said to be* normal *iff $gng^{-1} \in N$ for any $g \in G$ and $n \in N$.*

Lemma 6.2. *Let LargeSubgroupOfSimpleGroup be an algorithm that finds a large subgroup for any simple group of non-prime size n in time $O(n^{3/2+\epsilon})$. Then there exists an algorithm LargeSubgroup that finds a large subgroup for an* ***arbitrary*** *group G of non-prime order $n > 1$ in time $O(n^{3/2+\epsilon})$.*

The algorithm in Lemma 6.2 requires a method to efficiently test whether an input group is simple or not, and to find a nontrivial normal subgroup if the group is non-simple. Next, we devise such a procedure, and give the proof of Lemma 6.2 in Section 6.2.

6.1 Finding normal subgroups of non-simple groups

The algorithm that we present finds a minimal nontrivial normal subgroup of G. It is composed of three steps:

- Find a small set of generators of G. This is important in order to reduce the running time of the following steps from $O(n^2)$ to $\widetilde{O}(n)$.
- Find all conjugacy classes of G.
- Find a normal subgroup as a collection of classes whose union is closed under multiplication.

6.1.1 Step 1: Find a small set of generators

We use the standard notion of *generators* and *Cayley graph*.

Definition 6.3 (Generators)**.** *Let G be a group and let $S \subseteq G$. The group generated by S is the smallest subgroup of G that contains S. It is denoted $\langle S \rangle$. If $G = \langle S \rangle$ we say that S is a set of generators for G.*

Definition 6.4 (Cayley graph)**.** *The Cayley graph of a group G with respect to a set $S \subseteq G$ has vertices which are all the elements of G, and the edges are comprised of $g \leftrightarrow gs$ for every $g \in G$ and $s \in S$.*

The following lemma is standard, and asserts that we can efficiently find a small set of generators for a group.

Lemma 6.5. *There exists an algorithm* Generators *that receives a set A of a group G, and outputs a subset $S \subseteq A$ that satisfies $\langle S \rangle = X$ and $|S| \leq \log_2 |X|$, where $X := \langle A \rangle$.* Generators *runs in $\widetilde{O}(|X|)$ time.*

Proof. **The algorithm.** In Generators we iterate over elements $a \in A$. At any point, we maintain a set S that generates all elements of A that we have seen so far, and the group $H = \langle S \rangle$. If $a \in H$, then we do not need to update S and H. If, however, $a \notin H$, then we add a to S, and compute $H = \langle S \rangle$ all over again. When we finish to enumerate A, S generates $X = \langle A \rangle$. Note that the size of H at least doubles every time it is updated, as the old H is a proper subgroup of the new H. Hence $2^{|S|} \leq |H|$, and in particular $|S| \leq \log_2 |X|$

Algorithm 4: Generators

```
Input: A subset $A$ of a group $G$
Output: A set $S$ satisfying $\langle S \rangle = \langle A \rangle$ and $|S| \leq \log_2 |\langle A \rangle|$.
1 $H \leftarrow \{e_S\}$
2 $S \leftarrow \emptyset$
3 for $a \in A$ do
4     if $a \notin H$ then
5         $S \leftarrow S \cup \{a\}$
6         $H \leftarrow \langle S \rangle$
7     end
8 end
9 return $S$
```

Time complexity. Enumerating A takes at most $O(|X|)$ time. The dominant cost of the algorithm is computing the set H. Computing H at line 6 takes time $O(|S||H|)$ using any exploration algorithm (BFS or DFS) on the Cayley graph of G with respect to the generators S. Since the size of H at least doubles at every update of S, the total running time is dominated by the last update, which runs in $O(|S||X|)$ time. Since $|S| \leq \log_2 |X|$ we have $O(|S||X|) = \widetilde{O}(|X|)$ □

6.1.2 Step 2: Find conjugacy classes

Definition 6.6. *The* conjugacy class *of an element $g \in G$ is denoted*

$$\mathrm{Cl}(g) := \left\{ h \cdot g \cdot h^{-1} \,\middle|\, h \in G \right\}.$$

Lemma 6.7. *There exists an algorithm* ConjugacyClasses *that receives a group G of size n as input and returns the set of all conjugacy classes of G in $\widetilde{O}(n)$ time.*

Proof. **The algorithm.** In ConjugacyClasses we define a conjugacy graph, where each group element $g \in G$ is connected to all elements of the form sgs^{-1}. However, we only consider $s \in S$ where S is a set that generates G. Since each element $h \in G$ can be written as a product of elements of S, we deduce that there exists a path in the graph

Algorithm 5: ConjugacyClasses

```
    Input: A group $G$ of size $n$
    Output: Set $R$ of all conjugacy classes of $G$.
1 $S \leftarrow$ Generators($G$)
2 $E \leftarrow \{g \leftrightarrow sgs^{-1} \colon s \in S, g \in G\}$
3 return ConnectedComponents(E)
```

between g and hgh^{-1}. Hence, we compute the set of connected components of this graph, which coincides with the set of conjugacy classes of G.

Time complexity. By Lemma 6.5, we can find the set S of generators in $\widetilde{O}(n)$ time. Furthermore, we compute the connected components in the conjugacy graph that we define in linear time in the number of edges, which is $O(n \log_2(n))$ since $|S| \leq \log_2(n)$,. □

6.1.3 Step 3: Find a union of conjugacy classes, closed under multiplication

We now present the algorithm for computing a minimal nontrivial normal subgroup.

Lemma 6.8. *There exists an algorithm* NormalSubgroup *that receives a group G of size n as input, and returns a normal subgroup of G which is minimal with respect to inclusion (excluding $\{e_G\}$).* NormalSubgroup *runs in $\widetilde{O}(n)$ and can be used to test whether G is simple or not.*

We introduce a concept that we use in the description of the algorithm of Lemma 6.8.

Definition 6.9. *Given a subset C of a group G, we denote by $\mathrm{MN}(C)$ the Minimal Normal subgroup of G that contains C, where 'minimal' is with respect to inclusion.*

In order to prove Lemma 6.8, we observe that every normal subgroup is a union of conjugacy classes that is closed under multiplication, and vice versa. Therefore, we search for a minimal closed union of conjugacy classes. To do this, we maintain a dynamic directed graph whose vertices are conjugacy classes of G, and whose edges $C \to C'$ indicate that $C' \subseteq \mathrm{MN}(C)$.

We add an edge from C to C' if we find elements $a, b \in C$ with $a \cdot b \in C'$. More generally, if a, b lie in conjugacy classes that are *reachable* from C, and $a \cdot b \in C'$, then we add an edge $C \to C'$.

Naively, we can entirely expand this graph until no more edges can be added, and for every conjugacy class C, determine $\mathrm{MN}(C)$ to be the union of classes reachable from C.

However, since we are only interested in one minimal normal subgroup, we can use the following important optimization. Once there exists an edge from C to $C' \neq \{e\}$, we can disregard $\mathrm{MN}(C)$, and only consider $\mathrm{MN}(C')$, since $\mathrm{MN}(C') \subseteq \mathrm{MN}(C)$. This optimization is eligible as long as there are no cycles in the graph. We generalize this optimization in the presence of cycles by defining for every conjugacy class C a set S_C, which is the union of classes which can reach C in the graph. If there is an edge $C \to C'$ with $C' \subseteq S_C$, then we do not disregard C, since C' was already disregarded, and in fact $\mathrm{MN}(C) = \mathrm{MN}(C')$.

An algorithmic building block we use in NormalSubgroup is the following.

Claim 6.10. *There exists an algorithm that receives two subsets A, B of a group G, along with a set S of generators for G, that runs in $\widetilde{O}(|B||S|)$ time, which either*

- *Determines that* $\mathrm{MN}(A) \subseteq B$ *and returns* $\mathrm{MN}(A)$.
- *Finds an element* $h \in \mathrm{MN}(A) \setminus B$.

We are now ready to present the NormalSubgroup algorithm.

Proof of Lemma 6.8. **The algorithm.** NormalSubgroup virtually maintains the graph mentioned above, where the vertices are conjugacy classes of G, and where edges $C \to C'$ indicate that $\mathrm{MN}(C') \subseteq \mathrm{MN}(C)$. At any point of time, a few properties are met:

- The graph is a forest.
- Each connected component has a sink vertex T, reachable from any other vertex in the component.
- A set V holds the set of sinks.
- For any sink T, S_T is the union of conjugacy classes in the connected component of T (plus $\{e\}$).

At each iteration of the algorithm, a sink C with minimal size of S_C is chosen. If $\mathrm{MN}(C) \subseteq S_C$, then $\mathrm{MN}(C)$ is a minimal normal subgroup (with respect to inclusion) and we output $\mathrm{MN}(C)$. Otherwise, the algorithm from Claim 6.10 finds an element c in $\mathrm{MN}(C)$ which is not in S_C. This element c belongs to a conjugacy class at a different connected component (recall S_C holds all elements in the component of C). We find C', the sink of this connected component containing c and add an edge $C \to C'$. We remove C from V, and add the elements of S_C to $S_{C'}$. The above properties continue to hold.

Time complexity. Using Claim 6.10, the computation of either $\mathrm{MN}(C_i)$ or the element c of Algorithm 6 is done in $\widetilde{O}(|S_{C_i}| \log(n))$ time. It is possible by using a precomputed set of generators for G of size $\leq \log_2(n)$, using Lemma 6.5. C' may be found in $O(1)$ by keeping an additional table which maps any element of S_T to (a pointer to) T. The sets $\{S_T\}$ for all $T \in V$ can be stored as balanced search trees. Each iteration of the loop in lines 7-13 can hence be implemented in $\widetilde{O}(|S_{C_i}|)$ time. The total time required for all iterations is hence of the order of $\sum_i |S_{C_i}|$, up to poly logarithmic factors. We use double counting:

$$\sum_i |S_{C_i}| = \sum_{g \in G} \#\{i \colon g \in S_{C_i}\}. \tag{4}$$

To estimate the quantity on the right hand side, we note that for any element $g \in G$, if we consider the sets S_{C_i} containing g in order, then the ratio between sizes of consctuive such sets is at least 2. This is because of our choice of C_i to be the class having smallest S_{C_i}. Indeed, when the elements of S_{C_i} are added to $S_{C'}$, the size of $S_{C'}$ is at least double the size of S_{C_i}. Therefore, if $g \in C_i$, then the next time g may appear as an element of S_{C_j} (with j minimal such with $j > i$), is when $C_j = C'$, at which point S_{C_j} has at least double the size of S_{C_i}.

Algorithm 6: NormalSubgroup

```
Input: A group $G$ of size $n$
Output: A normal subgroup $H \trianglelefteq G$ which is minimal with respect to inclusion
1  $V \leftarrow \mathsf{ConjugacyClasses}(G) \setminus \{\{e\}\}$
2  for $N \in V$ do
3      $S_N \leftarrow N \cup \{e\}$
4  end
5  $i \leftarrow 0$
6  go to line 13
7  while $\mathrm{MN}(C_i) \not\subseteq S_{C_i}$ do
8      $V \leftarrow V \setminus \{C_i\}$
9      $c \leftarrow$ Any element of $\mathrm{MN}(C_i) \setminus S_{C_i}$
10     $C' \leftarrow$ A class $N \in V$ satisfying $c \in S_N$
11     $S_{C'} \leftarrow S_{C'} \cup S_{C_i}$
12     $i \leftarrow i + 1$
13     $C_i \leftarrow \arg\min_{N \in V}(|S_N|)$
14 end
15 return $\mathrm{MN}(C_i)$
```

In particular, each $g \in G$ belongs to S_{C_i} for at most $\log_2(n)$ values of i. By (4) the total running time is at most

$$\widetilde{O}(|G| \log_2(n)) = \widetilde{O}(n).$$

□

Proof of Claim 6.10. The algorithm we use is a mix of both Generators and Conjugacy-Classes algorithms. We repeatedly expand A in two ways:

- $A \leftarrow \bigcup_{g \in G} gAg^{-1}$.
- $A \leftarrow \langle A \rangle$.

These steps are repeated until either A has converged or we encounter an element outside of B (and we stop computation as early as we see such an element). Since the size of A at least doubles after applying these two steps, the runtime is dominated by the last expansion step. We show that these steps can be performed in $\widetilde{O}(k|S|)$ time, where k the size of the new A we compute. However, since all elements we generate are confined to B, except for at most one element, then the running time is $\widetilde{O}(|B||S|)$.

To perform the first expansions step in near-linear time, we explore the conjugacy graph used in ConjugacyClasses Algorithm 5, exploiting a precomputed set of generators S of G. We implement the second step as first computing $S' = \mathsf{Generators}(A)$, and then exploring the Cayley graph of G with respect to generators S' (starting at e_G), similarly to line 6 of Algorithm 4. □

6.2 Constructing large subgroups for non-simple groups

We now proceed with the proof of Lemma 6.2.

Proof of Lemma 6.2. **The algorithm.** LargeSubgroup starts by computing a minimal nontrivial normal subgroup N of G using NormalSubgroup. If no such subgroup exists (i.e. $N = G$), then G is simple, and we determine a large subgroup of G using LargeSubgroupOfSimpleGroup. Otherwise, if $|N| \geq \sqrt{n}$, then N is a large subgroup and we are done. Otherwise, if G/N is a group of prime order p, then we have $p = |G|/|N| > n/\sqrt{n} = \sqrt{n}$. In this case we output a subgroup generated by an element of order p. This is a valid large subgroup since $\sqrt{n} < p = |G|/|N| < n$. We find such an element by using that any element g of $G \setminus N$ has order which is divisible by p, and hence $g^{\mathrm{order}(g)/p}$ has order exactly p.

Otherwise, G/N is a group of non-prime order > 1, and we can apply LargeSubgroup recursively to G/N. Let $M = \mathsf{LargeSubgroup}(G/N)$, then we output $H = M \cdot N$ which is a proper subgroup of G using the correspondence theorem for groups. H is indeed large, because

$$|H| = |M| \cdot |N| \geq \sqrt{|G|/|N|} \cdot |N| = \sqrt{|G||N|} \geq \sqrt{n}.$$

Algorithm 7: LargeSubgroup

```
Input: A group $G$ of non-prime order $n$
Output: $H \leqslant G$ with $\sqrt{n} \leq |H| < n$
1  $N \leftarrow$ NormalSubgroup($G$)
2  if $|N| = n$ then
3      return LargeSubgroupOfSimpleGroup(G)                ▷ $G$ is simple
4  else if $|N| \geq \sqrt{n}$ then
5      return $N$                                          ▷ $N$ is a large subgroup
6  else if $|G|/|N|$ is prime then
7      $p \leftarrow G/|N|$
8      $g \leftarrow$ Any element of $G \setminus N$
9      $h \leftarrow g^{\mathrm{order}(g)/p}$
10     return $\{e, h, h^2, \ldots, h^{p-1}\}$             ▷ A $p$-subgroup is large
11 else
12     $M \leftarrow$ LargeSubgroup($G/N$)                 ▷ The quotient group has a large subgroup
13     return $M \cdot N$                                  ▷ Lift the subgroup to $G$
14 end
```

Time complexity. The algorithm may call itself recursively with G replaced by G/N. The format in which G/N is given is by a left transversal of N in G. Moreover, we pass an array which maps each $g \in G$ to its candidate in gN that appears in the transversal. This way, multiplication of two elements in the transversal is done in $O(1)$ using usual multiplication, and then a single access to the table that takes us back to the transversal. Since the recursion has depth of at most $\log_2(n)$, using these tables incurs an additive $O(n \log(n))$ time overhead, and a constant factor slowdown. All other computations in

LargeSubgroup are $\widetilde{O}(n)$, except for a single call to LargeSubgroupOfSimpleGroup(G) at some point of the recursion, which may take up to $O(n^{3/2+\epsilon})$ time, by assumption. □

7 Finding large subgroups of simple groups

In this section, we prove the following lemma, which completes the proof of Theorem 1.1.

Lemma 7.1. *There exists an algorithm LargeSubgroupOfSimpleGroup that receives a finite simple group G of non-prime order n, and returns a large subgroup of G in $O(n^{3/2+\epsilon})$ time.*

Both proofs in the literature of the fact that there always exists a large subgroup [4, 5] to any finite simple group of non-prime size go by case analysis. This case analysis relies on the classification theorem of finite simple groups, which is also essential to our algorithmic result. The classification theorem states that every finite simple group belongs to one of a finite number of explicit families of groups, which are summarized in Table 2. For each family of groups, a relatively explicit construction of a large subgroup has been found.

Next, we outline our approach to the proof of Lemma 7.1.

In order to find a large subgroup for the simple group G, we enumerate *names* of finite simple groups of the same order as G. The number of such candidate groups from Table 2 is $O(\log |G|)$. For each such group, we search for a large subgroup, assuming that G is isomorphic to that group. Since one of these assumptions is correct (in case the input is valid), one of these efforts will produce a large subgroup for G.

The following definition is due.

Notation. The *name* of a finite simple group is a tuple (f, m, q) where f is a family (index to a row in Table 2) and m, q are the (possibly redundant) parameters that indicate the specific group within the family. The group with this name is denoted $f(m, q)$. When either m or q are not parameters to f as listed in Table 2, we keep the convention of setting these to *null*.

7.1 Enumerating simple groups

Our first step toward finding large subgroups, is to give a short list of named groups, so that one of them is isomorphic to our input group. We generate the groups of size $|G|$.

For example, when presented with the Cayley table of the group $A_6(8)$, we aim to give a few names of simple groups, that includes the tuple (f, m, q) for $f = A_m(q)$ and $m = 6$, $q = 8$.

Claim 7.2. *The procedure EnumerateGroups applied with the input argument n outputs the names of all finite simple groups of size n in time $O(\log^4(n))$. The number of groups it outputs is at most $O(\log(n))$.*

We remark that there has been extensive research on the problem of recognizing a group using a few queries to its multiplication table. Although some of these results or

Family Name	**Notation**	**Parameters**	**Size**	$\|W\|$
Cyclic groups	$\mathbb{Z}_q$	q prime	q	
Alternating groups	A_m	$m > 4$	$m!/2$	
Chevalley Linear groups	$A_m(q)$	$m > 1 \vee q \geq 5$	$\frac{q^{m(m+1)/2}}{\gcd(m+1,q-1)} \prod_{i=1}^{m}(q^{i+1}-1)$	$(m+1)!$
Chevalley Orthogonal groups	$B_m(q)$	$m > 1$ $(m,q) \neq (2,2)$	$\frac{q^{m^2}}{\gcd(2,q-1)} \prod_{i=1}^{m}(q^{2i}-1)$	$2^m m!$
Chevalley Symplectic groups	$C_m(q)$	$m > 2$, q	$\frac{q^{m^2}}{\gcd(2,q-1)} \prod_{i=1}^{m}(q^{2i}-1)$	$2^m m!$
Chevalley Orthogonal groups	$D_m(q)$	$m > 3$, q	$\frac{q^{m(m-1)}(q^m-1)}{\gcd(4,q^m-1)} \prod_{i=1}^{m-1}(q^{2i}-1)$	$2^{m-1}m!$
Exceptional Chevalley group	$E_6(q)$	q	$\frac{q^{36}}{\gcd(3,q-1)} \prod_{i\in\{2,5,6,8,9,12\}}(q^i-1)$	$2^7 \cdot 3^4 \cdot 5$
Exceptional Chevalley group	$E_7(q)$	q	$\frac{q^{63}}{\gcd(2,q-1)} \prod_{i\in\{2,6,8,10,12,14,18\}}(q^i-1)$	$2^{10} \cdot 3^4 \cdot 35$
Exceptional Chevalley group	$E_8(q)$	q	$q^{120} \prod_{i\in\{2,8,12,14,18,20,24,30\}}(q^i-1)$	$2^{14} \cdot 3^5 \cdot 175$
Exceptional Chevalley group	$F_4(q)$	q	$q^{24} \prod_{i\in\{2,6,8,12\}}(q^i-1)$	$2^7 \cdot 3^2$
Exceptional Chevalley group	$G_2(q)$	$q \neq 2$	$q^6(q^2-1)(q^6-1)$	12
Steinberg Unitary groups	${}^2A_m(q^2)$	$m > 1$ $(m,q) \neq (2,2)$	$\frac{q^{m(m+1)/2}}{\gcd(m+1,q+1)} \prod_{i=1}^{m}(q^{i+1}+(-1)^i)$	$2^{\lceil m/2\rceil}\lceil m/2\rceil!$
Steinberg Orthogonal groups	${}^2D_m(q^2)$	$m > 3$, q	$\frac{q^{m(m-1)}}{\gcd(4,q^m+1)}(q^m+1) \prod_{i=1}^{m-1}(q^{2i}-1)$	$2^{m-1}(m-1)!$
Exceptional Steinberg group	${}^2E_6(q^2)$	q	$\frac{q^{36}}{\gcd(3,q+1)} \prod_{i\in\{2,5,6,8,9,12\}}(q^i-(-1)^i)$	$2^7 \cdot 3^2$
Exceptional Steinberg group	${}^3D_4(q^3)$	q	$q^{12}(q^8+q^4+1)(q^6-1)(q^2-1)$	12
Suzuki groups	${}^2B_2(q)$	$q = 2^{2k+1}$, $k \geq 1$	$q^2(q^2+1)(q-1)$	2
Ree groups	${}^2F_4(q)$	$q = 2^{2k+1}$, $k \geq 1$	$q^{12}(q^6+1)(q^4-1)(q^3+1)(q-1)$	16
Ree groups	${}^2G_2(q)$	$q = 3^{2k+1}$, $k \geq 1$	$q^3(q^3+1)(q-1)$	2
Sporadic groups + Tits group	27 groups	$m \in \{1 \ldots 27\}$	27 different sizes	

Figure 2: An exhaustive list of all simple finite groups [6].
The parameter m ranges over all positive integers and q ranges over all prime powers.
Families marked in blue are the twisted Chevalley groups.
Families marked in red are the exceptional (untwisted) Chevalley groups.
All groups with a nonempty $|W|$ column are of Lie type.

methods could be used to produce a list of size 1 in Claim 7.2 (perhaps with a worse run time), we leave it out of scope of this paper.

Proof. **The Algorithm.** EnumerateGroups works by checking first if n is a size of a sporadic group, and accordingly reports this as an option. For any other family f of groups, we note that if m is a parameter of f then $|f(m,q)| > 2^m$ for all eligible m, q. For this reason we limit $m \leq \log_2(n)$. Moreover, fixing m, there is always a small number of gcd values g that might appear in the size column in Table 2 at the row corresponding to f. For example, if f is the family of Chevalley Linear groups $A_m(q)$, then g is a divisor of $m+1$. Likewise, if f if the family of Chevalley Orthogonal groups $D_m(q)$, then g is a divisor of 4. In any case, g is always a divisor of $12(m+1)$. Syntactically fixing the value of this gcd (if existent) to g, we note that the size of $f(m,q)$, which we denote by $\mathsf{size}(q)$ is a function strictly increasing in q. Hence, it is possible to perform a binary search to find the maximal $q \in \mathbb{N}$ for which $\mathsf{size}(q) \leq n$ (note the requested q always satisfies $q \leq n$). Finally, we check whether this maximal value q_0 satisfies $|f(m,q_0)| = n$ (this time, the size is computed with the correct value of the gcd). If it does, we output the tuple (f, m, q_0) accordingly. Note the computation of size can be done in $O(m)$ time, and it can be tweaked to work with $n^{O(1)}$ numbers, because at the moment some intermediate computation inside size yields a value greater than n^2, we know $\mathsf{size}(q)$ is larger than n, and we should (binary) search for a smaller q.

Time complexity. The most time-consuming part is the computation of size inside the binary search. This computation (which takes at most $O(m) \leq O(\log(n))$ time) is done for each $m \in M$ (at most $\log_2(n)$ options), for any g (at most $12\log_2(n)$ options) in the binary search (at most $2\log_2(q_0)$ iterations; note $q_0 \leq n$). □

7.2 Finding large subgroups of named simple groups

In order to prove Lemma 7.1, we need to present a method to determine a large subgroup for G, given its name $f(m,q)$. The characterization of this subgroup must rely only on black-box properties of the group $f(m,q)$, and not on some specific ways to present this group. We summarize several such characterizations that we use.

Sporadic groups For the 27 sporadic groups, we do not exhibit an explicit construction of a large subgroup, however their existence is known [4]. In these cases, we tweak our algorithm to brute-force over all possible options. Since there is only a finite number of these groups, the time complexity of our algorithm is not affected.

Alternating Groups An alternating group A_m with $m \geq 5$, has the large subgroup of even permutation on $m-1$ elements, that is, A_{m-1}. It is large due to a simple computation

$$|A_{m-1}| = \frac{(m-1)!}{2} \geq \sqrt{m!/2} = \sqrt{|A_m|}.$$

However, from an algorithmic perspective, given $G = A_m$ as a black-box group, it is not clear how to extract A_{m-1}. The steps we use in order to do so are as follows.

1. Find all 3-cycles $\pi = (a,b,c)$ in G. That is, permutations satisfying $\pi(a) = b$, $\pi(b) = c$ and $\pi(c) = a$ with $a,b,c \in [m]$ and $\pi(x) = x$ for other x's.

Algorithm 8: EnumerateGroups

```
Input: A size $n$
Output: A list of name tuples $(f, m, q)$ of all simple groups with $|f(m,q)| = n$
if $n$ is a size of a sporadic group then
    Names $\leftarrow$ {(Sporadic groups, Corresponding index of sporadic group, null)}
else
    Names $\leftarrow \emptyset$
end
for $f \in$ Families of non-sporadic finite simple groups do
    if $f$ depends on parameter $m$ by third column of Table 2 then
        $M \leftarrow \{1, 2, \ldots, \lfloor \log_2(n) \rfloor\}$                ▷ List of relevant $m$'s
    else
        $M \leftarrow \{null\}$
    end
    for $m \in M$ do
        if $f$ depends on parameter $q$ by third column of Table 2 then
            for Any $g \mid 12(m+1)$ do
                $\mathsf{size}(q) :=$ Size of $f(m,q)$ as in Table 2, but with gcd set to $g$
                $q_0 \leftarrow \mathsf{BinarySearch}(q \in [1,n], \mathsf{size}(q) \leq n)$
                if $f(m, q_0)$ is a group (by third column of Table 2) and
                 $|f(m, q_0)| = n$ then
                    Names $\leftarrow$ Names $\cup \{(f, m, q)\}$
                end
            end
        else if $|f(m, null)| = n$ then
            Names $\leftarrow$ Names $\cup \{(f, m, null)\}$
        end
    end
end
return Names
```

2. Find a subset of these 3-cycles which is conjugate to the set $\{(1,2,k)\colon k = 3 \dots m\}$ in $G = A_m$.

3. Identify A_{m-1} as the group generated by any subset of size $m-3$ of the above set.

Step 1. To find all 3-cycles in G, we search in linear time for all elements of order 3. In this set of elements, 3-cycles form a conjugacy class. Except for when $m = 6$, this conjugacy class is the only one of size $2 \cdot \binom{m}{3}$, and it may be found using Algorithm 5. Indeed, conjugacy classes of elements of order 3 are characterized by the number r of disjoint 3-cycles in each permutation ($3r \leq m$). The number of elements in each conjugacy class is

$$\frac{2^r m!}{3!^r (m-3r)! r!}.$$

Except for when $m = 6$, substituting $r > 1$ yields larger conjugacy classes than $r = 1$. Hence we can find the set of 3-cycles in the group $G = A_m$ that is given as a black box, in near-linear time. The case $m = 6$ can be treated separately, e.g. by brute-force, similarly to the sporadic groups.

Step 2. Given two 3-cycles x, y, there are three options for the interaction between x and y. If the cycles are disjoint, then $x \cdot y$ has order 3. If x, y share a single element of $\{1, \dots, m\}$, then $x \cdot y$ has order 5. If x, y share two elements of $[m]$, then either $x \cdot y$ is of order 2 or is of order 3, depending on the orientation of the intersection. We demonstrate this using two examples.

$$x \cdot y \text{ of order 2:} \quad x = (1,2,3), \quad y = (1,2,4).$$
$$x \cdot y \text{ of order 3:} \quad x = (1,3,2), \quad y = (1,2,4).$$

In particular, consider two 3-cycles a, b with $a \cdot b$ of order 2. Then, there are exactly $m-4$ other 3-cycles c with both $a \cdot c$ and $b \cdot c$ of order 2. We find this set of permutations, together with a, b in $m^{O(1)}$ time in the set of 3-cycles that we found in previous step. This set is conjugate to $\{(1,2,k)\colon k = 3 \dots m\}$.

Step 3. The set $\{(1,2,k)\colon k = 3 \dots (m-1)\}$ is known to generate A_{m-1}. Similarly, if we drop any 3-cycle from the set of permutations that we found in the previous step, we end up with a set of generators for a copy of A_{m-1}, which is a large subgroup of A_m.

Groups of Lie Type The finite simple groups which are not cyclic, sporadic, or belong to the alternating family, are called groups of Lie type.

For this type of groups, our algorithm LargeSubgroupOfSimpleGroup follows the (common) construction by [4] and [5]. These constructions rely on the notion of a Borel subgroup of a group of Lie type, whose existence is well known. Specifically, the large subgroups provided both in [4] and [5] are of the following form.

Theorem 7.3. *Let G be a finite simple group of Lie type with a Borel subgroup B. Then, there exists an element $a \in G$ so that the parabolic subgroup*

$$P = \langle B, a \rangle$$

is a large subgroup of G.

Theorem 7.3 is the result of Case 3 in the proof of [5]. There, the parabolic subgroup is denoted P_J, and is defined as $P_J = BN_JB$ (for a certain set N_J). It is moreover shown that P_J is a large subgroup if G is not one of the 27 sporadic groups (including the Tits group). By [7, Proposition 8.3.1] it follows that P_J has the form $\langle B, a\rangle$ for some $a \in G$.

Accordingly, our algorithm includes the following ingredients to handle this case of groups of Lie type.

Lemma 7.4. *There exists an algorithm* EnumerateParabolic *that receives a group G of order n, and a subgroup B of G, which enumerates all subgroups of the form*

$$P_a = \langle B, a\rangle, \quad a \in G \tag{5}$$

in $O\left(n^2 \cdot \frac{\log|B|+1}{|B|}\right)$ time.

In order to control the running time of Lemma 7.4 in the case where B is a Borel subgroup, we show that Borel subgroups are almost large.

Lemma 7.5. *Let $\epsilon > 0$. Except for only a finite number of finite simple groups G of Lie type, every Borel subgroup B of $|G|$ satisfies*

$$|B| \geq |G|^{1/2-\epsilon}.$$

Moreover, we show that it is possible to find a Borel subgroup efficiently.

Lemma 7.6. *There exists an algorithm* BorelSubgroup *that receives a group G of Lie type together with a name (f, m, q), and outputs a Borel subgroup of G if $G \cong f(m, q)$. The running time of the algorithm is $\widetilde{O}(n)$.*

Proof of Lemma 7.1. **The Algorithm.** LargeSubgroupOfSimpleGroup receives a simple group G as input. It uses EnumerateGroups in order to enumerate over a shortlist of names (f, m, q) of finite simple groups, one of which is isomorphic to G.

For each group name $f(m, q)$, we attempt to construct a large subgroup of G under the assumption that G is isomorphic to $f(m, q)$. If we succeed we return our results; otherwise, we proceed to the next candidate (f, m, q).

If $f(m, q)$ is a sporadic group, we try to find a large subgroup by brute-force (or by any clever method using the known structure of these groups). If $f(m, q)$ is an alternating group, then we attempt to find a large subgroup using the method described at the start of Section 7.2.

Otherwise, $f(m, q)$ is of Lie type. In this case we find a Borel subgroup for $f(m, q)$ and enumerate all parabolic subgroups containing B. By Theorem 7.3, one of these parabolic groups will be a large subgroup of $f(m, q)$.

Time complexity. For f a sporadic family we invest at most $O(1)$ runtime. For f the alternating group, we attempt to locate A_{m-1} in $\widetilde{O}(n)$ time using the method described at the beginning of Section 7.2. When f is of Lie type, the Borel subgroup is found in $\widetilde{O}(n)$ time by Lemma 7.6, and the parabolic subgroups are enumerated in $O(n^{3/2+\epsilon})$ time due to the bound of Lemma 7.5 on B that we enforce, and Lemma 7.4. □

Algorithm 9: LargeSubgroupOfSimpleGroup

```
Input: A simple group $G$ of non-prime size $n$
Output: A large subgroup of $G$
for $(f, m, q) \in$ EnumerateGroups$(n)$ do
    if $f$ is "sporadic groups" then
        if (Can find a large subgroup in $G$ by brute-force) then
            return The large subgroup
        end
    else if $f$ is "alternating groups" then
        if (Can find a large subgroup in $G$ by treating $G$ as $A_m$) then
            return The large subgroup
        end
    else
        $B \leftarrow$ BorelSubgroup$(G, f, m, q)$
        if ($B$ is as large as projected by Lemma 7.5) then
            for $P \in$ EnumerateParabolic$(f(m, q), B)$ do
                if $|P| \geq \sqrt{n}$ and $|P| < n$ then
                    return $P$
                end
            end
        end
    end
end
                                        ▷ Unreachable, unless input is invalid
```

We present the short proof that Borel subgroups are almost large.

Proof of Lemma 7.5. The existence of Bruhat decompositions [7, Proposition 8.2.3] is well-known for finite simple groups of Lie type. This decomposition implies the existence of a set $W' \subseteq G$ with

$$G = \bigcup_{w \in W'} BwB. \tag{6}$$

In (6), W' has a 1-to-1 correspondence with the Weyl group W of G. We claim that $|W| \leq |G|^{2\epsilon}$ except for a finite number of simple groups. This can be verified using Table 2, where the values for $|W|$ are listed and are taken from [5, Table II]. Hence, with only a finite number of exceptions, (6) implies

$$|G| \leq |B|^2 \cdot |W| \leq |B|^2 \cdot |G|^{2\epsilon} \implies |G|^{1/2-\epsilon} \leq |B|.$$

□

7.2.1 Enumerating parabolic subgroups

Proof of Lemma 7.4. **The Algorithm.** EnumerateParabolic enumerates all groups $P_a = \langle B, a \rangle$ by considering only elements a that belong to some left transversal A of B in G. This is valid since $P_a = P_{ab}$ for any $b \in B$. To compute P_a, the algorithm considers the Cayley graph of G with respect to $S \cup \{a\}$ where S is a set of generators for B. The connected component of the identity element e_G equals $P_a = \langle B, a \rangle$.

Algorithm 10: EnumerateParabolic

```
Input: A finite group $G$ of size $n$ and a subgroup $B \leqslant G$
Output: A list of all subgroups $P_a$ from Equation (5)
1  $A \leftarrow$ LeftTransversal$(G, B)$
2  $S \leftarrow$ Generators$(B)$
3  $E \leftarrow \{g \leftrightarrow gs \colon g \in G, s \in S\}$
4  $P \leftarrow \{\}$
5  for $a \in A$ do
6      $E' \leftarrow E \cup \{g \leftrightarrow ga \colon g \in G\}$
7      $P_a \leftarrow$ ConnectedComponent$(E', e_G)$
8      $P \leftarrow P \cup \{P_a\}$
9  end
10 return $P$
```

Time complexity. LeftTransversal(G, B) runs in $O(n)$ time. Generators(B) performs only $\widetilde{O}(|B|)$ multiplications. Initializing E requires $n \log |B|$ multiplications. Since A is of size $n/|B|$, computing the edges $g \leftrightarrow ga$ for all $a \in A$ requires $n^2/|B|$ operations in the group. Finally, finding a connected component is done in linear time in the number of edges E' (at worst), which amounts to $n(\log |B| + 1)$, and is done $|A| = n/|B|$ times. □

7.2.2 Computing a Borel subgroup

In order to construct a Borel subgroup, we introduce several basic definitions.

Definition 7.7. *Let H be a subgroup of G. The normalizer of H (in G) is defined as*

$$N_G(H) := \{g \in G \colon gHg^{-1} = H\}$$

Definition 7.8. *Let G be a group whose order divides a prime p. A p-Sylow subgroup H of G is a p-subgroup (subgroup whose order is a power of p) which is maximal with respect to inclusion.*

The following characterization of Borel subgroups is presented in [8].

Fact 7.9 ([9, Corollary 24.11])**.** *Let G be a finite group of Lie type, defined over the field $\mathbb{F}_q$. Let p be the prime number dividing q. Then the normalizer of any p-Sylow group of G is a Borel subgroup of G.*

We need a few more group-theoretic facts.

Fact 7.10 (Cauchy's theorem)**.** *Let G be a group of order divisible by a prime p. Then, there exists a non-identity element $g \in G$ satisfying $g^p = e$.*

Fact 7.11 (Sylow theorems [10])**.** *Let G be a group with order divisible by a prime p.*

1. *There exists a subgroup of G of order p^k with p^k the largest p-power dividing $|G|$.*

2. *All p-Sylow subgroups are conjugate to each other.*

Fact 7.12 ([11])**.** *Let H be a proper subgroup of a p-group S. Then the normalizer of H in S properly contains H.*

Proof of Lemma 7.6. By Fact 7.9, the normalizer of a p-Sylow subgroup is a Borel subgroup. Thus, the construction of a Borel subgroup immediately follows from two procedures, that we describe next.

- PSylow: An algorithm to find a p-Sylow subgroup of G.

- Normalizer: An algorithm to find the normalizer of a subgroup $H \leqslant G$.

Both algorithms have $\widetilde{O}(n)$ running time.

The Normalizer Algorithm. We start by computing a small set S of generators for H. We observe that if $g \in G$ has $gSg^{-1} \subseteq H$, then $gHg^{-1} \subseteq H$ and hence $gHg^{-1} = H$. The opposite is also true – if $g \in N_G(H)$ then $gSg^{-1} \subseteq H$. Hence we compute in $\widetilde{O}(n)$ time the set of $g \in G$ satisfying $gSg^{-1} \subseteq H$.

The PSylow Algorithm. We iteratively find p-subgroups of G with orders p^i, for $i = 0, 1, 2, 3, \cdots$. Given a non-maximal p-subgroup H of G, we compute the normalizer of H in G. We search for an element $g \in N_G(H)$ which is outside of H so that g^p is inside H (using exponentiation by squaring). Once such a g in found, $H = \langle g, H\rangle$ is of order $p|H|$, and we proceed likewise by trying to enlarge H. The algorithm runs in $\widetilde{O}(n)$,

Algorithm 11: Normalizer

```
Input: A subgroup $H$ of a group $G$
Output: The normalizer $N_G(H)$ of $H$ in $G$
$S \leftarrow$ Generators$(H)$
$N \leftarrow G$
for $s \in S$ do
    $N \leftarrow N \cap \{g \in G : gsg^{-1} \in H\}$
end
return $N$
```

and we only need to prove that such a g with $g^p \in H$ is found, and that $\langle g, H\rangle$ has order $p|H|$.

First, if $g \in N_G(H) \setminus H$ satisfies $g^p \in H$, then we notice $(g^iH)(g^jH) = g^{i+j}H$ and so $\langle g, H\rangle = \bigcup_{i=1}^{p}(g^iH)$ is of order $p|H|$.

Second, we show such a g exists. Recall our assumption that $p|H|$ divides $|G|$. It implies that H is a proper subgroup of a p-Sylow subgroup S of G. By Fact 7.12 we have that $H \subsetneq N_S(H)$. Therefore, H is a proper *normal* subgroup of $N_S(H)$. Since $N_S(H)$ is a p-subgroup of G by itself, $N_S(H)/H$ has order divisible by p. Hence, by Cauchy's theorem (Fact 7.10), there exists an element gH of $N_S(H)/H$ that is of order p. This corresponds to our sought-of element $g \in N_S(H) \subseteq N_G(H)$ that satisfies $g^p \in H$.

Algorithm 12: PSylow

```
Input: A group $G$ and a prime number $p$ dividing its order
Output: A $p$-Sylow subgroup of $G$
$H \leftarrow \{e\}$
while $p|H|$ divides $|G|$ do
    for $g \in$ Normalizer$(G, H)$ do
        if $g \notin H$ and $g^p \in H$ then
            $H \leftarrow \langle g, H\rangle$                ▷ $\langle g, H\rangle$ has size $p|H|$
            break
        end
    end
end
return $H$
```

□

Acknowledgements

We thank Jack Schmidt for pointing us to Fact 7.9, which is essential to our proof. We are grateful to Yotam Shomroni, Lior Schain and Arieh Lev for insightful discussions.

References

[1] Alfred H Clifford and Gordon B Preston. The algebraic theory of semigroups, vol. 1. *AMS surveys*, 7:1967, 1961.

[2] Sridhar Rajagopalan and Leonard J. Schulman. Verification of identities. *SIAM J. Comput.*, 29(4):1155–1163, 2000.

[3] Gadi Kozma and Arieh Lev. Bases and decomposition numbers of finite groups. *Arch. Math*, 58:417–424, 1992.

[4] Larry Finkelstein, Daniel Kleitman, and Tom Leighton. Applying the classification theorem for finite simple groups to minimize pin count in uniform permutation architectures. In *VLSI Algorithms and Architectures*, pages 247–256, 1988.

[5] Arieh Lev. On large subgroups of finite groups*. *Journal of Algebra*, (152):434–438, Nov 1990.

[6] Wikipedia. List of finite simple groups — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=List%20of%20finite%20simple%20groups&oldid=1143483120, 2023. [Online; accessed 29-October-2023].

[7] Roger W. Carter. *Simple Groups of Lie Type.* John Wiley & Sons, 1989.

[8] Jack Schmidt (https://math.stackexchange.com/users/583/jack schmidt). Sylow p-subgroups of finite simple groups of lie type. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/834460 (version: 2014-06-15).

[9] G. Malle and D. Testerman. *Linear Algebraic Groups and Finite Groups of Lie Type (Cambridge Studies in Advanced Mathematics).* Cambridge University Press, 2011.

[10] Wikipedia. Sylow theorems — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Sylow_theorems&oldid=1168817793#Statement, 2023. [Online; accessed 29-October-2023].

[11] Wikipedia. p-group — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=P-group&oldid=1181826218#Non-trivial_center, 2023. [Online; accessed 29-October-2023].