Spot-Checkers*

Funda Ergün† Sampath Kannan† S Ravi Kumar‡
Ronitt Rubinfeld§ Mahesh Viswanathan†

November 11, 1999

Abstract

On Labor Day weekend, the highway patrol sets up spot-checks at random points on the freeways with the intention of deterring a large fraction of motorists from driving incorrectly. We explore a very similar idea in the context of program checking to ascertain with minimal overhead that a program output is *reasonably* correct. Our model of *spot-checking* requires that the spot-checker must run asymptotically much faster than the combined length of the input and output. We then show that the spot-checking model can be applied to problems in a wide range of areas, including problems regarding graphs, sets, and algebra. In particular, we present spot-checkers for sorting, convex hull, element distinctness, set containment, set equality, total orders, and correctness of group and field operations. All of our spot-checkers are very simple to state and rely on testing that the input and/or output have certain simple properties that depend on very few bits. Our results also give property tests as defined by [RS96, Rub94, GGR98].

1 Introduction

Ensuring the correctness of computer programs is an important yet difficult task. For testing methods that work by querying the programs, there is a tradeoff between the time spent for testing and the kind of guarantee obtained from the process. Program result checking [BK95] and self-testing/correcting programs [BLR93, Lip91] make runtime checks to certify that the program is giving the right answer. Though efficient, these methods often add small multiplicative factors to the runtime of the programs. Efforts to minimize the overhead due to program checking have been somewhat successful [BW94a, Rub94, BGR96] for linear functions.

Can the overhead be minimized further by settling for a weaker, yet nontrivial, guarantee on the correctness of the program's output? For example, it could be very useful to know that the program's output is reasonably correct (say, close in Hamming distance to the correct output). Alternatively, for programs that verify whether an input has a particular property, it may be useful to know whether the input is at least close to some input which has the property.

In this paper, we introduce the model of *spot-checking*, which performs only a small amount (sublinear) of additional work in order to check the program's answer. In this context, three seemingly different

*This work was supported by ONR N00014-97-1-0505, MURI. The second author is also supported by NSF Grant CCR96-19910. The third author is also supported by DARPA/AF F30602-95-1-0047. The fourth author is also supported by the NSF Career grant CCR-9624552 and Alfred P. Sloan Research Award. The fifth author is also supported by ARO DAAH04-95-1-0092.

†Email: {fergun@saul, kannan@central, maheshv@gradient}.cis.upenn.edu. Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104.

‡Email: ravi@almaden.ibm.com. IBM Almaden Research Center, San Jose, CA 95120.

§Email: ronitt@cs.cornell.edu. Department of Computer Science, Cornell University, Ithaca, NY 14853.

prototypical scenarios arise. However, each is captured by our model. In the following, let f be a function purportedly computed by program P that is being spot-checked, and x be an input to f.

- *Functions with small output.* If the output size of the program is smaller than the input size, say $|f(x)| = o(|x|)$ (as is the case for example for decision problems), the spot-checker may read the whole output and only a small part of the input.
- *Functions with large output.* If the output size of the program is much bigger than the input size, say $|x| = o(|f(x)|)$ (for example, on input a domain D, outputting the table of a binary operation over $D \times D$), the spot-checker may read the whole input but only a small part of the output.
- *Functions for which the input and output are comparable.* If the output size and the input size are about the same order of magnitude, say $|x| = \Theta(|f(x)|)$ (for example, sorting), the spot-checker may only read part of the input and part of the output.

One naive way to define a weaker checker is to ask that whenever the program outputs an incorrect answer, the checker should detect the error with some probability. This definition is disconcerting because it does not preclude the case when the output of the program is very wrong, yet is passed by the checker most of the time. In contrast, our spot-checkers satisfy a very strong condition: if the output of the program is far from being correct, our spot-checkers output FAIL with high probability. More formally:

Definition 1 *Let $\Delta(\cdot,\cdot)$ be a distance function. We say that $\mathcal{C}$ is an* ϵ-spot-checker *for f with distance function Δ if*

1. *Given any input x and program P (purporting to compute f), and ϵ, $\mathcal{C}$ outputs with probability at least 3/4 (over the internal coin tosses of $\mathcal{C}$)* PASS *if $\Delta(\langle x, P(x)\rangle, \langle x, f(x)\rangle) = 0$ and* FAIL *if for all inputs y, $\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) > \epsilon$.*
2. *The runtime of $\mathcal{C}$ is $o(|x| + |f(x)|)$*

The spot-checker can be repeated $O(\lg 1/\delta)$ times to get confidence $1 - \delta$. Thus, the dependence on δ need never be more than $O(\lg 1/\delta)$. The choice of the distance function Δ is problem specific, and determines the ability to spot-check. For example, for programs with small output, one might choose a distance function for which the distance is infinite whenever $P(x) \neq f(y)$, whereas for programs with large output it may be natural to choose a distance function for which the distance is infinite whenever $x \neq y$. The condition on the runtime of the spot-checker enforces the "little-oh" property of [BK95], i.e., as long as f depends on all bits of the input, the condition on the runtime of the spot-checker forces the spot-checker to run faster than any correct algorithm for f, which in turn forces the spot-checker to be different than any algorithm for f.

OUR RESULTS. We show that the spot-checking model can be applied to problems in a wide range of areas, including problems regarding graphs, computational geometry, sets, and algebra. We present spot-checkers for sorting, convex hull, element distinctness, set containment, set equality, total orders, and group and field operations. All of our spot-checker algorithms are very simple to state and rely on testing that the input and/or output have certain simple properties that depend on very few bits; the non-triviality lies in the choice of the distribution underlying the test. Some of our spot-checkers run much faster than $o(|x| + |f(x)|)$. All of our spot-checkers have the additional property that if the output is incorrect even on one bit, the spot-checker will detect this with a small probability. In order to construct these spot-checkers, we develop several new tools, which we hope will prove useful for constructing spot-checkers for a number of other problems.

Our sorting spot-checker runs in $O(\lg n)$ time to check the correctness of the output produced by a sorting algorithm on an input consisting of n numbers: in particular, it checks that the edit distance of

the output from the correct sorted list is small (at most ϵn^2). Very recently, the work of [EKR99] has used the techniques developed here for spot-checking sorting in order to construct efficient probabilistically checkable proofs for a number of optimization problems.

The convex hull spot-checker, given a sequence of k points with the claim that they form the convex hull of the input set of n points, checks in $O(\lg k)$ time whether this sequence is *close* (in edit distance) to the actual convex hull of the input set. We also show that there is an $O(1)$ spot-checker to check a program that determines whether a given relation is close to a total order.

One of the techniques that we developed for testing group operations allows us to efficiently test that an operation is associative. Recently in a surprising and elegant result, [RaS96] show how to test that operation $\circ$ over domain D is associative in $O(|D|^2)$ steps, rather than the straightforward $O(|D|^3)$. They also show that $\Omega(|D|^2)$ steps are necessary, even for cancellative operations. In contrast, we show how to test that $\circ$ is *close* (equal on most inputs) to some cancellative associative operation $\circ'$ over domain D in $\tilde{O}(|D|)$ steps[1]. We also show how to modify the test to accommodate operations that are not known to be cancellative, in which case the running time increases to $\tilde{O}(|D|^{3/2})$. Though our test yields a weaker conclusion, we also give a self-corrector for the operation $\circ'$, i.e., a method of computing $\circ'$ correctly for *all* inputs in constant time. Another motivation for studying this problem is its application to program checking, self-testing, and self-correcting [BK95, BLR93, Lip91]. Using techniques from [Rub94], our method yields a reasonably efficient self-tester and self-corrector (over small domains) for all functions that are solutions to the associative functional equation

$$F[F[x, y], z] = F[x, F[y, z]]$$

[Acz66].

We next investigate operations that are both associative and commutative. We show that one can test whether an operation is close to an associative, commutative, and cancellative group operation $\circ'$ in $\tilde{O}(|D|)$ time. This is slightly more efficient than our associativity tester. In contrast, we show that quadratic time is necessary and sufficient to test that a given operation is cancellative, associative, and commutative. As for the associative case, we then give a sub-quadratic algorithm for the case when $\circ$ is not known to be cancellative. Again, we show how to compute $\circ'$ in constant time, given access to $\circ$. We show that our simple test can be used to quickly check the validity of tables of abelian groups and fields. Our results can be summarized in the table below.

Input promise	Output guarantee	Running Time	Reference
None	Associative, exact	$\Theta(\|D\|^2)$	[RaS96]
Cancellative	Associative, close	$\tilde{O}(\|D\|)$	this paper
None	Associative, cancellative, close	$\tilde{O}(\|D\|^{3/2})$	this paper
Cancellative	Associative, commutative, close	$\tilde{O}(\|D\|)$	this paper
None	Associative, cancellative, commutative, close	$\tilde{O}(\|D\|^{3/2})$	this paper
None	Associative, commutative, exact	$\Omega(\|D\|^2)$	this paper

The solutions of the functional equation

$$F[F[x, y], z] = F[x, F[z, y]]$$

are the set of associative and commutative operations[Acz66]. Our results can be used in testing programs purporting to compute functions which are solutions to such a functional equation.

RELATIONSHIP TO PROPERTY TESTING. It is often useful to distinguish whether a given object has a certain property or is very far from having that property. For example, one might want to test if a function

[1]The notation $\tilde{O}(n)$ suppresses polylogarithmic factors of n.

is linear in such a way that linear functions pass the test while functions that are not close to any linear function fail. A second example is one might want to determine whether a graph is bipartite or not close to any bipartite graph (where closeness is defined in terms of the number of locations in the adjacency matrix that differ). Models of property testing were defined by [RS96] and [GGR98] (see also [Rub94]) in order to formalize this notion.

For the purposes of this exposition, we give a simplified definition of property testing that captures the common features of the definitions given by [RS96, Rub94, GGR98]. Given a domain H and a distribution $\mathcal{D}$ over H, a function g is ϵ-*close* to a function h over $\mathcal{D}$ if $\Pr_{x\in\mathcal{D}}[g(x) \neq h(x)] \leq \epsilon$. $\mathcal{A}$ is a *property tester* for a class of functions $\mathcal{F}$ ($\mathcal{F}$ is the set of functions which have the property) if for any given ϵ and function g to which $\mathcal{A}$ has oracle access, with high probability (over the coin tosses of $\mathcal{A}$) $\mathcal{A}$ outputs PASS if $g \in \mathcal{F}$ and FAIL if there is no $h \in \mathcal{F}$ such that f and h are ϵ-close[2]. Note that this model applies to graph properties by considering g and h to be descriptions of the adjacency matrix of the graph, i.e., they are functions from pairs of vertices $\langle u, v\rangle$ to $\{0, 1\}$ such that $g(u, v) = 1$ exactly when there is an edge between u and v [GGR98]. In any case, the notion of closeness can be captured by a "Hamming-like" distance function as in the definition of property testers. In the case that $\mathcal{D}$ is a uniform distribution, the distance function would correspond to the fraction of the domain on which g and h differ.

Property testing has had several applications. Many program result checkers [BK95] have used forms of property testing to ensure that the program's output satisfies certain properties characterizing the function that the program is supposed to compute (cf., [BLR93, EKS99, KS96, AHK95, ABC$^+$93]). Linear and low-degree polynomial property testers have been used to construct probabilistically checkable proof systems (PCPS) (cf., [BLR93, BFL91, FGL+96, BFLS90, RS96, AS98, ALM+98]). As we mentioned earlier, techniques developed in this paper for testing whether a sequence has (the property of containing) a long increasing subsequence were used to construct efficient PCPS for a number of optimization problems [EKR99]. Property testers for Max-CUT have been used to construct constant time approximation schemes for Max-CUT in dense graphs [GGR98].

Our focus on the checking of program results motivates a definition of spot-checkers that is natural for testing input/output relations for a wide range of problems. All previous property testers used a "Hamming-like" distance function. Our general definition of a distance function allows us to construct spot-checkers for set and list problems such as sorting and element distinctness, where the Hamming distance is not useful.

All property testers in [GGR98] can be turned into spot-checkers for the function f such that $f(x) = 1$ exactly when x has the property. Define a distance function Δ which forces $P(x) = f(x) = 1$ (by taking the value ∞ if otherwise) and such that $\Delta(\langle x, 1\rangle, \langle y, 1\rangle)$ is equal to the fraction of entries where x and y differ. Then the property tester gives a spot-checker with distance function Δ: both pass exactly when x is close to a y which has the property.

Conversely spot-checkers can also be viewed as property testers with more general distance functions: Given a distance function Δ, say that $\langle x, x'\rangle$ is ϵ-*close* to $\langle y, f(y)\rangle$ if $\Delta(\langle x, x'\rangle, \langle y, f(y)\rangle) \leq \epsilon$. Alternatively, define the property $\mathcal{F} = \{\langle x, f(x)\rangle \mid \text{inputs } x\}$ characterizing the correct input-output pairs of the function f. Then spot-checkers with distance function Δ also test if the input-output pair $\langle x, P(x)\rangle$ is close to a member of $\mathcal{F}$.

One must, however, be careful in choosing the distance function. For instance, consider a program which decides whether an input graph is bipartite or not. Every graph is close to a graph that is not bipartite (just add a triangle), so property testing for nonbipartiteness is trivial. Thus, unless the distance function satisfies a property such as $\Delta(\langle x, y\rangle, \langle x, y'\rangle)$ is greater than ϵ when $y \neq y'$, the spot-checker will have an

[2]The definition of property testing given by [GGR98] is more general. For example, it allows one to separately consider two different models of the tester's access to f. The first case is when the tester may make queries to f on any input. The second case is when the tester cannot make queries to f but is given a random sequence of $\langle x, g(x)\rangle$ pairs where x is chosen according to $\mathcal{D}$. In our setting, the former is the natural model.

uninteresting behavior.

2 Set and List Problems

2.1 Sorting

Given an input to and output from a sorting program, we show how to determine whether the output of the program is close in edit-distance to the correct sorting of the input, where the edit-distance $\rho(u, v)$ is the number of insertions and deletions required to change string u into v. The distance function that we use in defining our spot-checker is as follows: for all x, y lists of elements, $\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle)$ is infinite if either $x \neq y$ or $|P(x)| \neq |f(y)|$; otherwise it is $\rho(P(x), f(y))/|P(x)|$. Since sorting has the property that for all x, $|x| = |f(x)|$, we assume that the program P satisfies $\forall x, \ \ |x| = |P(x)|$. It is straightforward to extend our techniques to obtain similar results when this is not the case.

We assume that the elements are drawn from an ordered set and this ordering relation can evaluated in constant time. We also assume that all the elements in our list are distinct. (This assumption is not necessary for testing for the existence of a long increasing subsequence.)

In Section 2.1.3, we show that the running time of our sorting spot-checker is tight.

2.1.1 The Test

Our spot-checker first checks if there is a long increasing subsequence in $P(x)$ (Theorem 2). It then checks that the sets $P(x)$ and x have a large overlap (Lemma 8). If $P(x)$ and x have an overlap of size at least $(1-\epsilon)n$, where $n = |x|$, and $P(x)$ has an increasing subsequence of length at least $(1-\epsilon)n$, then $\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) \leq 2\epsilon$. Hence, this spot-checker is a 2ϵ-spot-checker.

The spot-checker is given an input array A of length n whose elements are accessible in constant time. The algorithm presented in the figure checks if A has a long increasing subsequence by picking random pairs of indices $i < j$ and checking that $A[i] < A[j]$. An obvious way of picking i and j is to pick i uniformly and then pick j to be $i+1$. Another way is to pick i and j uniformly, making sure that $i < j$. However, one can find sequences that pass these tests, even though they do not contain long increasing subsequences. The choice of distribution on the pairs i, j is crucial to the correctness of the checker.

```
Procedure Sort-Check(A, ε)
repeat O(1/ε) times
   choose i ∈_R [1, n]
   for k ← 0 ... ⌈lg i⌉ do
      repeat O(1) times
         choose j ∈_R [1, 2^k]
         if (A[i − j] > A[i]) then return FAIL
   for k ← 0 ... ⌈lg(n − i)⌉ do
      repeat O(1) times
         choose j ∈_R [1, 2^k]
         if (A[i] > A[i + j]) then return FAIL
return PASS
```

Theorem 2 *Procedure* `Sort-Check`(A, ϵ) *runs in* $O((1/\epsilon)\lg n)$ *time, and satisfies:*

- *If A is sorted,* $\texttt{Sort-Check}(A, \epsilon) = \textsf{PASS}$.
- *If A does not have an increasing subsequence of length at least $(1-\epsilon)n$, then with probability at least $3/4$,* $\texttt{Sort-Check}(A, \epsilon) = \textsf{FAIL}$.

To prove this theorem we need some basic definitions and lemmas.

Definition 3 *The graph* induced *by an array A, of integers having n elements, is the directed graph G_A, where $V(G_A) = \{v_1, \ldots, v_n\}$ and $E(G_A) = \{\langle v_i, v_j\rangle \mid i < j$ and $A[i] < A[j]\}$.*

We now make some trivial observations about such graphs.

Observation 4 *The graph G_A induced by an array $A = \{v_1, v_2, \ldots, v_n\}$ is transitive, i.e., if $\langle u, v\rangle \in E(G_A)$ and $\langle v, w\rangle \in E(G_A)$ then $\langle u, w\rangle \in E(G_A)$.*

We shall use the following notation to define neighborhoods of a vertex in some interval.

NOTATION. For $t, t', i \in \mathbb{Z}$, let $\Gamma^{+}_{(t,t')}(i)$ denote the set of vertices v_j such that $t < j < t'$ that have an incoming edge from v_i. Similarly, let $\Gamma^{-}_{(t,t')}(i)$ denote the set of vertices v_j such that $t < j < t'$ that have an outgoing edge to v_i.

It is useful to define the notion of a *heavy* vertex in such a graph to be one whose in-degree and out-degree, in every 2^k interval around it, is a significant fraction of the maximum possible in-degree and out-degree, in that interval.

Definition 5 *A vertex v_i in the graph G_A is said to be* heavy *if for all k, $0 \le k \le \lg i$, $\mid \Gamma^{-}_{(i-2^k,i)}(i) \mid \ge \eta 2^k$ and for all k, $0 \le k \le \lg(n-i)$, $\mid \Gamma^{+}_{(i,i+2^k)}(i) \mid \ge \eta 2^k$, where $\eta = 3/4$.*

Theorem 6 *A graph G_A induced by an array A, that has $(1-c)n$ heavy vertices, has a path of length at least $(1-c)n$.*

The theorem follows as a trivial consequence of the following:

Lemma 7 *If v_i and v_j $(i < j)$ are heavy vertices in the graph G_A, then $\langle v_i, v_j\rangle \in E(G_A)$.*

Proof. Since G_A is transitive, in order to prove the above lemma, all we need to show is that between any two heavy vertices, there is a vertex v_k such that $\langle v_i, v_k\rangle \in E(G_A)$ and $\langle v_k, v_j\rangle \in E(G_A)$.

Let m be such that $2^m \le (j-i)$, but $2^{(m+1)} \ge (j-i)$. Let $l = (j-i) - 2^m$. Let I be the closed interval $[j-2^m, i+2^m]$ with $|I| = (i+2^m) - (j-2^m) + 1 = 2^m - l + 1$. Since v_i is a heavy vertex, the number of vertices in I that have an edge from v_i is at least $\eta 2^m - ((j-2^m) - i) = \eta 2^m - l$. Similarly, the number of vertices in I, that are adjacent to v_j is at least $\eta 2^m - (j - (i+2^m)) = \eta 2^m - l$.

Now, we use the pigeonhole principle to show that there is a vertex in I that has an incoming edge from i and an outgoing edge to j. By transitivity that there must be an edge from i to j. This is true if $(\eta 2^m - l) + (\eta 2^m - l) \ge |I| = 2^m - l + 1$. Since $\eta = 3/4$, this condition holds if $l \le 2^{m-1}$.

Now consider the case when $l > 2^{m-1}$. In this case we can consider the intervals of size 2^{m+1} to the right of i and to the left of j and apply the same argument based on the pigeonhole principle to complete the proof. □

Proof. [of Theorem 2] Clearly if the checker returns FAIL, then the array is not sorted.

We will now show that if the induced graph G_A does not have at least $(1-c)n$ heavy vertices then the checker returns FAIL with probability $1-\delta$. Assume that G_A has greater that cn light vertices. The checker can fail to detect this if either of the following two cases occurs: (i) the checker only picks heavy vertices,

or (ii) the checker fails to detect that a picked vertex is light. A simple application of Chernoff bound shows that the probability of (i) is at most $\delta/2$.

By the definition of a light vertex, say v_i, there is a k such that $|\Gamma^+_{(i,i+2^k)}(i)|$ (or $|\Gamma^-_{(i,i-2^k)}(i)|$) is less than $(3/4)2^k$. The checker looks at every neighborhood; the probability that the checker fails to detect a missing edge when it looks at the k neighborhood (all v_j such that $i \leq j \leq i \pm 2^k$) can be shown to be at most $\delta/2$ by an application of Chernoff's bound. Thus the probability of (ii) is at most $\delta/2$. □

In order to complete the spot-checker for sorting, we give a method of determining whether two lists A and B (of size n) have a large intersection, where A is presumed to be sorted.

Lemma 8 *Given lists A, B of size n, where A is presumed to be sorted and distinct. There is a procedure that runs in $O(\lg n)$ time such that if A is sorted and $|A \cap B| = n$, it outputs* PASS *with high probability, and if $|A \cap B| < \epsilon n$ for a suitable constant ϵ, it outputs* FAIL *with high probability.*

REMARK. The algorithm may also fail if it detects that A is not sorted or is not able to find an element of B in A.

Proof. [of Lemma 8] Suppose A is sorted. Then, one can randomly pick $b \in B$ and check if $b \in A$ using binary search. If binary search fails to find b (either because $b \notin A$ or A is wrongly sorted), the test outputs FAIL. Each test takes $O(\lg n)$ time, and constant number of tests are sufficient to make the conclusion. □

2.1.2 An Alternate Test

We give an alternate test that is slightly simpler. We begin by assuming that the elements in A are distinct.

```
Procedure Sort-Check-II (A, ε)
repeat O(1/ε) times
   choose i ∈_R [1, n]
   perform binary search as if to determine whether x_i is in A
   if not found return FAIL
return PASS
```

We prove the following theorem:

Theorem 9 *Procedure* `Sort-Check-II`(A, ϵ) *runs in $O((1/\epsilon)\lg n)$ time, and satisfies the same conditions as Theorem 2*

Proof. Say that $i \in [n]$ is *good* if the binary search for x_i is successful. Clearly, if at least ϵ fraction of i's is not good, the test fails with high probability. Now, we show that the set of good i's form an increasing subsequence: Given $i < j$, both good, at some point the binary search for x_i must diverge from the binary search for x_j. At this point, it must be because x_i is less than the pivot element and x_j is greater than it, so $x_i < x_j$. □

It is easy to modify the above spot-checker to the case when the elements are not distinct by treating element x_i as $\langle x_i, i\rangle$.

2.1.3 A Lower Bound for Spot-Checking Sorting

We have shown in the two preceding sections that $O(\lg n)$ time is sufficient for our checkers to spot-check sorting on a list of size n. We now show that for comparison-based spot-checkers $\lg n$ is also a lower bound. We do this by showing that any comparison-based spot-checker for sorting running in $o(\lg n)$ time will either fail a completely sorted sequence or pass a sequence that contains no increasing subsequence of length $\Omega(n)$, thus violating the requirements in its definition. In other words, for any comparison-based spot-checker $P(A, \epsilon)$ with distance parameter ϵ which runs in $o(\lg n)$ time, there exists a sequence A^* of length n such that either i) A^* is completely sorted and the checker fails A^* with high probability, or ii) $\Delta(\langle A^*, P(A^*)\rangle, \langle B, f(B)\rangle) > \epsilon$ for all sequences B of length n and P will pass the sequence with high probability[3].

We describe sets of input sequences that present a problem for such spot-checkers. We will call these sequences *3-layer-saw-tooth inputs.*

We define k-layer-saw-tooth inputs (k-lst's) inductively. For the base case we define $\text{lst}_1(x_1)$ to be the set of increasing sequences in $\mathbb{Z}^{x_1}$ (sequences of length x_1 of integers). Then 2-lsts are comprised of a sequence of 1-lsts, such that every element of the ith 1-lst is smaller than every element of the $i+1$st 1-lst. More generally, k-lsts take k integer arguments, $(x_1, x_2, \ldots, x_k)$ and are denoted by $\text{lst}_k(x_1, x_2, \ldots, x_k)$. $\text{lst}_k(x_1, x_2, \ldots, x_k)$ represents the set of sequences in $\mathbb{Z}^{x_1 x_2 \cdots x_k}$ which are comprised of x_k blocks of sequences from $\text{lst}_{k-1}(x_1, x_2, \ldots, x_{k-1})$. Moreover, if k is odd, then the largest integer in the i-th block is less than the smallest integer in the $(i+1)$-st block for $1 \leq i < x_k$. If k is even, then the smallest integer in the i-th block is greater than the largest integer in the $(i+1)$-st block for $1 \leq i < x_k$.

An example $\text{lst}_3(3, 3, 2)$ is:

$$\overbrace{\underbrace{7\,8\,9}_{\in \text{lst}_1(3)}\;\; 4\,5\,6\;1\,2\,3}^{\in \text{lst}_2(3,3)}\;16\;17\;18\;13\;14\;15\;10\;11\;12$$

In Figure 1 we present a 3-layer saw-tooth as a graph. Note that the longest increasing subsequence in $\text{lst}_3(i, j, k)$ is of length ik and can be constructed by choosing one $\text{lst}_1(i)$ from each $\text{lst}_2(i, j)$.

We now show that $o(\lg n)$ comparisons are not enough to spot-check sorting using any comparison-based checker (including that presented in the previous section).

Lemma 10 *A checker of the kind described above must either* FAIL *a completely sorted sequence or* PASS *a sequence that contains no increasing sequence of length* $\Omega(n)$.

Proof. Suppose, for contradiction, that there is a checker that runs in $f(n) = \Theta(\lg n/\alpha(n))$ time where $\alpha(n)$ is an unbounded, increasing function of n. Assume the checker generates $O(f(n))$ index pairs $(a_1, b_1), \ldots (a_k, b_k)$, where the $a_l < b_l$ for $1 \leq l \leq k$ and returns PASS if and only if, for all l, the value at position a_l is less than the value at position b_l (otherwise, one can construct a completely sorted sequence which the checker fails).

We maintain an array consisting of $\lg n$ buckets. For each (a_l, a_l) pair generated by the checker, we put this pair in the bucket whose index is $\lfloor \lg(b_l - a_l) \rfloor$. It follows that there is a sequence of $c\alpha(n)$ buckets (for some $c < 1$) such that the probability (over all possible runs of the checker) that one of the pairs falls in one of these buckets is at most c. Let these $c\alpha(n)$ buckets range from p to q. In other words, $q = p + c\alpha(n)$ and there are very few pairs (a, b) such that $b - a$ is between 2^p and 2^q.

Our analysis uses the structure of 3-lst inputs, specifically that if the checker compares pairs in different lst_2 blocks or the same lst_1 block, it will not detect an error. However, it will detect an error if it compares pairs in different lst_1 blocks but the same lst_2 blocks.

[3] for simplicity, assume that the paramater ϵ is hardwired into P and f, in effect making them P_ϵ and f_ϵ.

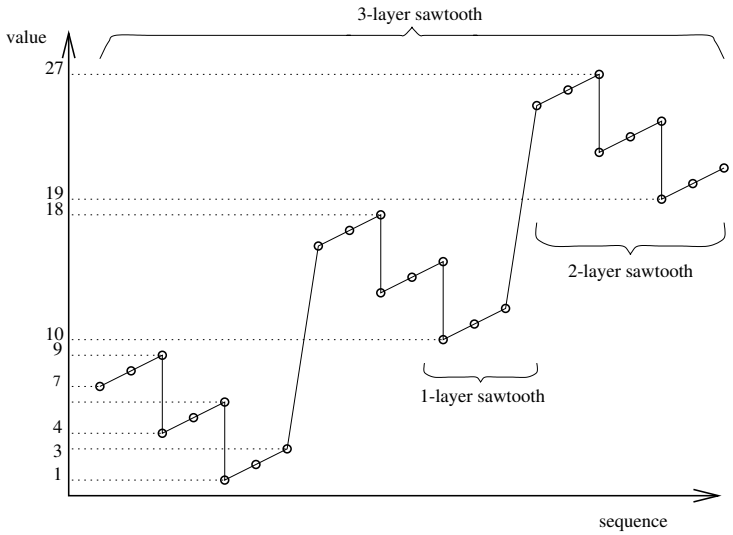

Figure 1: 3-layer saw-tooth: an $\mathrm{lst}_3(3,3,3)$ sequence.

Assume that the checker generates (a,b) pairs such that a is chosen uniformly. Consider an input from $\mathrm{lst}_3(i,j,k)$ with $i = 2^{p+d}$ and $j = 2^{c\alpha(n)-d}$ for some constant d, and $k = n/(ij)$. If the checker generates an (a,b) pair such that $b-a > ij$, then a and b are in different lst_2 blocks. Hence, the checker will not detect that the input is not sorted. If the checker generates an (a,b) pair such that $b - a < i/2^d$, and if a is in the first $(1 - 1/2^d)$ fraction of the lst_1-block, then a and b will be in the same lst_1 block. In this case, checker will not detect an error. If a is in the last $1/2^d$ fraction of the lst_1-block, then the checker may or may not detect that the input is not sorted depending on whether b is in the same lst_1-block or not. However, the latter happens with probability at most $1/2^d$. Finally, if the checker compares elements coming from different lst_1 blocks but within the same lst_2 block, it will detect that the input is not sorted. However, the choice of i, j, k is such that this probability is at most c. Thus, even though this input has no increasing sequence of length more than $n/(2^{\Omega(\alpha(n))})$, the probability that the checker will return FAIL is less than a constant.

If a is not chosen uniformly, one can consider not only the lst_3's described, but also concatenations of an increasing sequence of length uniformly chosen from $\{1..i\}$ to an lst_3 structure. There will still be no increasing sequence of length more than $n/(2^{\Omega(\alpha(n))})$, and a will land in the first $1 - 1/2^d$ fraction of the lst_1-block with probability at least $1 - 1/2^d$. □

2.2 Convex Hull

We assume that program P, given a set of n points on the Euclidean plane, returns a sequence $\langle x_0, x_1, \ldots, x_k \rangle$ of $k+1$ pointers ($k < n$) to the points in the input. The claim of P is that there exists a convex polygon whose vertices are $\langle x_0, x_1, \ldots, x_k \rangle$, if read in counterclockwise order (convexity), and all of the n input points lie on or within this polygon (hullness).

Checking convex hulls has been investigated before in the context of the Leda software package by Mehlhorn *et. al.* [MNS$^+$98]. Their checkers work for convex polyhedra of any dimension greater than two. Since they are checkers in the traditional sense, they aim at finding any discrepancy from the correct answer and therefore have higher running times (which mostly depend on the dimension and therefore not necessarily comparable to ours, but they are at least linear in the size n of the input set). In addition, they conclude that while convexity is efficiently checkable, checking whether all the points lie in the convex

polygon (the hullness property) is hard. This is due to the necessity of checking every point against many facets.

Let f be the function that gives the correct convex hull of a set of points. The spot-checker for convex hull uses the following distance function: Let X, Y be sets of points on the plane. Define $\Delta(\langle X, P(X)\rangle, \langle Y, f(Y)\rangle)$ to be ∞ if $Y \neq P(X)$ (i.e., f(Y) is the convex hull of the set of points returned by the program) and $\max\{d_{\text{con}}, d_{\text{hull}}\}$ otherwise, where d_{con} is the minimum fraction of points in X whose removal makes $P(X)$ the convex curve $f(Y)$, and d_{hull} is the fraction of points in X that are outside $f(Y)$. We prove the following theorem

Theorem 11 *Given n points in the plane, there is an ϵ-spot-checker that runs in $O(\lg n)$ time for spot-checking convex hull.*

We will develop the spot-checker in two phases; one will check that the output is close to convex, and the next will make sure that it is close to a hull.

2.2.1 Spot-Checking Convexity

We show how to check in $O(\lg k)$ time whether a sequence of $k+1$ nodes can be turned into a convex polygon by deleting at most ϵk of the nodes. Let CH be a sequence of edges where edge $e_i = (x_i, x_{i+1 \bmod k+1})$. We may also construct new edges, e.g., $e = (x_k, x_l)$ between pairs of output nodes.

All edges[4] make an angle in the interval $[0, 2\pi)$ with the x-axis. Without loss of generality, the axes are so that $\angle e_0 = 0$.

We now define a relation on the edges of a polygon which is closely related to its convexity. It will be used to replace the usual "$<$" of sorting.

Definition 12 *For $0 \leq i, j, \leq k$, $e_i \, R \, e_j$ iff* (i) *$i < j$ and* (ii) *either $x_{i+1} = x_j$ and $0 < \angle e_j - \angle e_i < \pi$ or $0 < \angle e_j - \angle(x_{i+1}, x_j) < \pi$ and $0 < \angle(x_{i+1}, x_j) - \angle e_i < \pi$. In addition, $e_k \, R \, e_0$ if $\angle e_k > \pi$.*

The realtion R is not transitive. However, observe that if $e_i \, R \, e_j$ and $x_{i+1} \neq x_j$ then $e_i \, R \, (x_{i+1}, x_j)$ and $(x_{i+1}, x_j) \, R \, e_j$.

A quick observation shows that the sequence of edges of a convex polygon forms an increasing sequence with respect to R. We now proceed to show that a sequence of edges on the plane which is increasing with respect to R corresponds to a convex polygon.

Lemma 13 *Let $S = \langle f_0, \ldots, f_l \rangle$ be a sequence of edges such that the head of edge f_l is connected to the tail of f_0 and $f_i \, R \, f_{i+1 \bmod l+1}$ for all $0 \leq i \leq l$. Construct polygon C by connecting, for all edges f_i in S, the head of f_i to the tail of f_{i+1} if they are not already connected. Then,* (i) *C is not self-intersecting,* (ii) *C is convex.*

Proof. (i) Consider C as a sequence of edges starting with e_0 and ending with e_k. Due to the definition of R, for any edge e and e' that immediately follows e in C, $e \, R \, e'$, therefore the angles of the edges in C are increasing. Assume now that C has multiple (say two) loops. Add node x to C where it intersects itself. This results in the division of the edges that intersect into two separate edges each (Figure 2). Of the two loops joined at x, remove the one that does not contain e_0[5] to obtain C'. C' is a closed curve where the angles of the edges are in increasing order. Now look at edges d and d' incident on x (assume d precedes d' in C'). We could have three situations: (a) $\angle d' < \angle d$, (b) $\angle d' - \angle d > \pi$, or (c) $0 < \angle d' - \angle d < \pi$.

[4] Since they are directed, it might be helpful to think of them as vectors.

[5] There is an implicit assumption here that x does not lie on e_0, but the argument works for any labeling of edges and shifting of the coordinate axes accordingly.

(a) is not possible since the angles are in increasing order. Assume (b) is true. Since the angles in C' form an increasing sequence, there is an interval $(\angle d, \angle d')$ of $\geq \pi$ radians such that no edge of C' has an angle within this interval. This implies the existence of a direction such that any progress made in this direction by an edge is never compensated for, contradicting the closedness of C'. If (c) holds, with a similar argument to (b), the closedness of the second loop (that we deleted) is violated.

(ii) C is a simple polygon where the angles of the edges are in increasing order. As a result of this, the increase of angle from one edge to the next is always under π. (see (i) for how the closedness of C is violated if it is π or more.) This means that all the interior angles of the polygon are less than π, thus it must be convex.

□

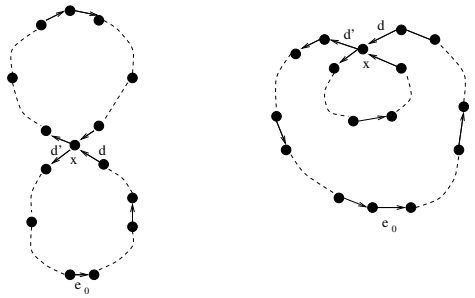

Figure 2: Looping closed curves.

THE CONVEXITY TEST.

We give a procedure to spot-check if CH is convex. We assume that CH is accessed as a list of edges (which are pairs of points) that represent the (purportedly) convex polygon.

```
Procedure Convex-Check(CH, c, δ)
run Sort-Check (CH, c/2, δ), replacing < with R
if e_k or e_0 is not heavy return FAIL
if ∠e_k ≤ π return FAIL
return PASS
```

Clearly, if CH is convex, it will pass this test. We now show that if CH passes this test then it is possible to join a large fraction of its nodes (respecting the order that they occur in CH) to obtain a closed curve that respects R for every pair of adjacent edges.

Theorem 14 *If* CH *passes the above test then it can be made convex by removing at most* ϵk *nodes.*

Proof. Note that to be able to use the argument in the sorting spot-checker proof, we need to have a transitive relation. We first show that the relation R is transitive when angles are restricted.

Lemma 15 *Given edges* e_i, e_j, e_k *such that* $\angle e_k - \angle e_i < \pi$*, if* $e_i \, R \, e_j$ *and* $e_j \, R \, e_k$ *then* $e_i \, R \, e_k$*.*

Let e_{mid} be the last heavy edge in CH (with respect to R) with angle less than π, and let $e_{\text{mid}'}$ be the first heavy edge that comes after e_{mid}. Then, if the test passes, there exist two disjoint increasing subsequences

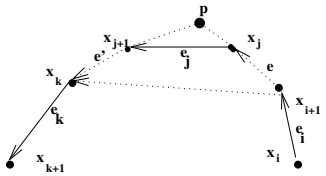

Figure 3: Transitivity under restricted conditions.

of CH with respect to R, of total length at least $(1-\epsilon)k$, the first one beginning with e_0 and ending with e_{mid} and the second one beginning with $e_{\text{mid}'}$ and ending with e_k. Closing the gaps in these sequences yields two piecewise linear curves which we will call *chain-1* and *chain-2* respectively. These chains form a closed curve if joined at their endpoints. The joining might involve adding an edge from e_{mid} to $e_{\text{mid}'}$ (Figure 4). If, at the joining points, $e_{\text{mid}}\, R\, e_{\text{mid}'}$ and $e_k\, R\, e_0$, then the closed curve must be convex (since the chains satisfy R within themselves). We know that $e_k\, R\, e_0$, since this is explicitly checked by the checker. We

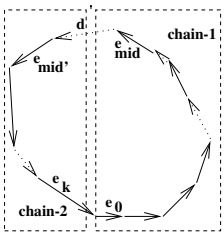

Figure 4: The two chains.

now show that the other joining point does not pose a problem either.

Lemma 16 *If the convexity spot-checker returns* PASS*, then* $e_{\text{mid}}\, R\, e_{\text{mid}'}$.

Thus, the two chains join together to form a convex polygon. Also note that for every node that is removed from the node sequence, at most two edges are left out from CH.

Putting those results together, the theorem follows. □

We now give the proofs of the two lemmas.

Proof. (Lemma 15) We show only the case where $x_{i+1} \neq x_j$ and $x_{j+1} \neq x_k$; the other cases are similar and simpler. Let $e = (x_{i+1}, x_j)$ and $e' = (x_{j+1}, x_k)$. We have $\angle e_i < \angle e < \angle e_j < \angle e' < \angle e_k$. Then, $\angle e' - \angle e < \pi$; thus, there exists a point p where extensions of e and e' intersect (Figure 3). x_{i+1}, p and x_k form a triangle, as a result of which $\angle e < \angle(x_{i+1}, x_k) < \angle e'$, and therefore, $e_i\, R\, e_k$. □

Proof. (Lemma 16) Assume that $e_{\text{mid}}\ \not R\, e_{\text{mid}'}$. e_{mid} and $e_{\text{mid}'}$ cannot be adjacent, since then e_{mid} would not be heavy. Then as in the sorting spot-checker proof, there must exist a (non-heavy) edge $e_r \in \text{CH}$, $\text{mid} < r < \text{mid}'$, such that and $e_{\text{mid}}\, R\, e_r\, R\, e_{\text{mid}'}$. This implies that $\angle e_{\text{mid}'} - \angle e_{\text{mid}} > \pi$, for otherwise, by the limited transitivity of R, $e_{\text{mid}}\, R\, e_{\text{mid}'}$ would hold.

Now construct $d = (x_{\text{mid}+1}, x_{\text{mid}'})$. Since $e_{\text{mid}}\ \not R\, e_{\text{mid}'}$ either $\angle d - \angle e_{\text{mid}} > \pi$, or $\angle e_{\text{mid}'} - \angle d > \pi$. Without loss of generality, assume the former. With d, the two chains join to form a closed curve C (recall that they are already joined at e_k and e_0). Since R holds for every pair of consecutive edges in C except between e_{mid} and $e_{\text{mid}'}$, the angles are increasing and no edge of C (including those from CH and

those added later) has an angle in the interval $(\angle e_{\mathrm{mid}}, min(\angle d, \angle e_{\mathrm{mid}'}))$, which is at least π radians. This contradicts the closedness of the curve. Thus, it must be that $e_{\mathrm{mid}}\, R\, e_{\mathrm{mid}'}$. □

2.2.2 Spot-Checking Hullness

To check whether the convex body obtained in the previous section covers all but an ϵ fraction of the nodes, we do the following. We sample $O(1/\epsilon)$ nodes and check in $O(\lg n)$ time whether each lies within the convex polygon obtained in the previous section. A simple application of Chernoff bounds shows that this test works.

To check whether a given sample node lies within the convex body, we use the fact that for any node v inside a convex hull, and for any node y on the hull, there exist two points y' and y'' such that y and y'' have adjacent locations in the sequence of points which make up the hall, and v lies inside the triangle $\langle yy'y''\rangle$.

To find whether a sample point v is inside the hull, the checker picks an arbitrary point y on the polygon and checks whether the edges incident on it are heavy with respect to R. It then tries to locate the candidate adjacent nodes y' and y'' on the convex polygon by binary search, such that $\angle(y', y) \leq \angle(v, y) \leq \angle(y'', y)$.

Note however that we have only CH to use in our search[6], while our actual search domain should be the convex polygon obtained from CH in the previous section. The angles in CH are not necessarily entirely sorted, therefore binary search might return a false positive or a false negative. False negatives do not cause a problem since they are caused by out of sequence elements in the list, which constitute a valid reason for rejection. The only way that a false positive can be obtained is if the search returns an edge (y', y'') in CH which is not in the convex polygon obtained from CH (Figure 5). This problem can be eliminated by requiring that the checker ensure that (y', y'') is a heavy edge in $O(\lg k)$ time. Then the checker checks

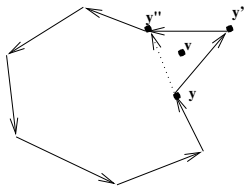

Figure 5: Potential problem caused by vertex out of sequence in CH.

constant time whether v is inside the triangle $\langle yy'y''\rangle$; if it is, it returns FAIL, otherwise it returns PASS.

The spot-checker spends $O(\lg k)$ time for each sample node. Since only a constant number of samples are used, total amount of work done is $O(\lg k)$.

2.3 Element Distinctness

Given membership access to a multiset A of size n, we would like to determine if A is distinct. However, suppose it is enough to ensure that A is mostly distinct, i.e., has at least $(1-\epsilon)n$ elements for a given ϵ. We show that this can be done in $O(\sqrt{\epsilon n})$ time. We assume that we can sample uniformly from A in constant time and testing equality of elements takes constant time. The test we propose is the following:

[6]To be precise, we use the sequence of nodes that we used to construct CH in the beginning, but the two sequences contain exactly the same information.

```
Procedure Element-Distinctness-Check (A, ε)
choose random √(εn) elements X from A
if X has any repeated elements return FAIL
return PASS
```

Note that by hashing it is possible to determine whether X has any repeated elements in $O(\sqrt{\epsilon n})$ time.

Our distance function captures the number of elements of the input set that need to be changed in order to make the output correct. Given multisets A, B, let $\rho(A, B)$ be the minimum number of elements that need to be inserted to or deleted from A in order to obtain B. If the program says "not distinct", then since X is trivially close to a nondistinct set, the distance can be set appropriately. Let $f(X) = 1$ if all the elements of X are distinct and 0 otherwise. Let P be a program that claims to compute f. One way to define the distance function is: $\Delta(\langle X, P(X)\rangle, \langle Y, f(Y)\rangle)$ is infinite if $P(X) \neq f(Y)$, and $\rho(X, Y)/|X|$ otherwise. We prove the following theorem:

Theorem 17 *For a constant $\epsilon > 0$, procedure* `Element-Distinctness-Check` (A, ϵ) *is an ϵ-spot-checker that runs in $\tilde{O}(\sqrt{\epsilon n})$ time, where the size of the multiset A is n.*

Proof. Let m be the number of elements to be sampled. Consider a set with k distinct elements and the proposed test (assume $k \mid n$). If x_i denotes the probability of picking the i-th element, noting that $\sum_{i=1}^{k} x_i^2$ is minimized when $x_1 = \cdots = x_k = n/k$, the worst-case is to assume that each element occurs n/k times.

If m elements are sampled uniformly with replacement, then the probability that all are distinct is upper bounded by the standard birthday analysis:

$$\prod_{i=0}^{m-1} \left(1 - \frac{i}{k}\right) \leq \prod_{i=0}^{m-1} \exp\left(\frac{-i}{k}\right) = \exp\left(\frac{-1}{k} \sum_{i=0}^{m-1} i\right) = \exp\left(\frac{-m(m-1)}{2k}\right)$$

We want this to be less than some constant. Simple manipulations yield condition $m \geq \Omega(\sqrt{k})$. Thus, if we need $k = \epsilon n$, we need $m \geq \Omega(\sqrt{\epsilon n})$. We can sort this sample in order to tell whether all elements are distinct, which adds an extra $\lg m$ factor. □

2.4 Set Equality

Given sets A, B of size n, we would like to determine whether $A = B$. However, suppose it is enough to distinguish the case when $A = B$ from the case when $|A \cap B|$ is relatively small (such as $|A \cap B| < \epsilon|A|$ for some $\epsilon > 0$.

Let $\rho(A, B)$ be the minimum number of elements that need to be inserted to or deleted from A in order to obtain B. Let $f(A, B) = 1$ if $A = B$ and 0 otherwise and let P be the program that claims to compute f. Then for sets X_1, X_2, Y_1, Y_2, we define $\Delta(\langle (X_1, X_2), P(X_1, X_2)\rangle, \langle (Y_1, Y_2), f(Y_1, Y_2)\rangle)$ to be infinite if either $X_1 \neq Y_1$ or $P(X_1, X_2) \neq f(Y_1, Y_2)$, and to be $\rho(X_2, Y_2)/|X_2|$ otherwise.

The following is a spot-checker for set equality. We assume that access to any element in A or B requires constant time.

```
Procedure Set-Equality-Check (A, B, ε)
set k = √(3/(ε(1 − ε)²))
choose a subset X of A by picking each element of A
independently with probability k/√n
choose a subset Y of B by picking each element of B
independently with probability k/√n
if |X ∩ Y| < k²√ε return FAIL
return PASS
```

The following lemma shows the validity of this spot-checker.

Lemma 18 *Given two sets of size n and constant $\epsilon < 1/2$,* `Set-Equality-Check` *is an ϵ-spot-checker for set equality that runs in $O(\sqrt{n/\epsilon})$ time.*

Proof. Let A, B be the given sets of size n. For a constant k to be determined, the checker simply chooses subsets of expected size $k\sqrt{n}$ at random from each list and spot-checks that the intersection of the samples has cardinality "close" to k^2 where "close" will be defined in the sequel. Notice that by hashing the two samples this checker can be made to run in $O(\sqrt{n})$ time with high probability.

To analyze the checker, consider first the case where $|A \cap B| = n$ (i.e., B is a permutation of A). For each element x_i let the random variable X_i be the indicator of the event that x_i occurs in both samples. $\Pr[X_i = 1] = (k/\sqrt{n})^2 = k^2/n$. Thus, $E[X_i] = k^2/n$. Letting X be the sum of the X_i, $E[X] = k^2$. Since the X_i are independent random variables, we can use Chernoff bounds to establish $\Pr[k^2 - X \geq \mu k^2] \leq \exp(-k^2\mu^2/2)$.

Now if $|A \cap B| < \epsilon n$, we are summing over ϵn X_i's instead of n. Thus the expected value of X is $k^2\epsilon$. Once again Chernoff bounds imply that $\Pr[X - k^2\epsilon \geq \mu k^2\epsilon] \leq \exp(-k^2\epsilon\mu^2/3)$.

We now need to choose k and the threshold at which the checker outputs PASS. For any desired constant $\epsilon < 1$ set the threshold to be $k^2\sqrt{\epsilon}$. Corresponding to this threshold, set $\mu = (1 - \sqrt{\epsilon})$ in both inequalities above. Finally, k should be chosen so as to make the probability of wrong classification a small constant. This is achieved by choosing k such that $k^2\epsilon\mu^2/3$ is bigger than $\lg 4/3$ in order to achieve an an error of at most $1/4$. □

3 Total Orders

In this section we show how to test whether a given relation "$\prec$" on the set $\{a_i \mid 1 \leq i \leq n\}$ is close to a total order. We represent the relation as a directed graph $H_\prec$ with vertex set $[n] = \{1, \ldots, n\}$, where $a_i \prec a_j$ iff (i, j) is an edge in $H_\prec$, and for every pair of nodes i and j, either (i, j) or (j, i) is an edge in $H_\prec$. We assume that given i and j we can query whether $i \prec j$ or $j \prec i$ in unit time. Note that $\prec$ is a total order iff $H_\prec$ is acyclic.

Given an input x (a relation assumed to be represented as a directed graph), let $f(x)$ return TOTAL ORDER if x represents a total order (x is a directed acyclic graph) and NOT TOTAL ORDER if x is not, and let P be a program purporting to compute f. The distance function is defined as follows: $\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle)$ is infinite whenever $P(x) \neq f(y)$ and is equal to the fraction of edges that need to be reversed to change x into y otherwise. Thus, the total order y with minimum Δ from x is the total order closest to x in terms of the number of edges that the two respective graphs share.

Though the problem of testing that a given graph is close to an acyclic graph seems similar to testing that a list has a long increasing subsequence, we show that it can be accomplished in constant time!

For any permutation π of $[n]$ and and $1 \leq i, j \leq n$, let $D_\pi(H_\prec)$ denote the number of edges $(\pi(i), \pi(j))$ of $H_\prec$ such that $\pi(i) > \pi(j)$. In other words, D counts the number of edges that go *backward* with respect to the order induced by π. We quantify how far $H_\prec$ is from being acyclic (or, equivalently, how far $\prec$ is from being a total order) by the function $D^*(H_\prec) = \min_\pi D_\pi(H_\prec)$. We also let π^* denote an ordering which achieves D^*. Without loss of generality we assume that the vertices are numbered in the order defined by π^*. We say that an edge (i, j) is *bad* if $i > j$. Otherwise we will say that the edge is *good*. Note that due to the numbering of the vertices, the goodness or badness of an edge is defined with respect to π^*.

The following fact about $H_\prec$ shows that $H_\prec$ cannot have too many bad edges with respect to π^*.

Observation 19 *For each i and for each $k > i$, at least half the edges between i and vertices in the interval $[i + 1, k]$ must be good edges. Similarly, for each i and for each $k < i$, at least half the edges between the interval $[k, i - 1]$ and i must be good edges.*

The above observation follows from the optimality of π^*. Otherwise moving i to the position right before k would yield an order with fewer bad edges. This is because in the interval between i and k the number of bad edges which would become good would exceed the number of good edges which would become bad. Outside the interval, the good and bad edges would stay the same. This fact also implies that at most half the edges in $H_\prec$ can be bad with respect to the optimal order.

The following corollary links bad edges to cycles of length 3.

Corollary 20 *If for $i < j$, the edge between i and j is a bad edge (i.e., from j to i), then there is a $k \in [i + 1, j - 1]$ such that the edges between i and k and between k and j are good edges. Hence the* triangle (i, j, k) witnesses *the fact that $H_\prec$ contains a cycle (of length 3).*

Strictly more than half of the edges between i and the vertices in the interval $[i+1, j-1]$ as well as between j and $[i + 1, j - 1]$ are good, because at least half of the edges between i (resp. j) and the interval $[i + 1, j]$ (resp. $[i, j - 1]$) are good, and (j, i) is bad. Thus, there exists a point k where both (i, k) and (k, j) are good edges. The corollary follows as a result of this, and yields an $O(n)$ spot-checker: The mapping from bad edges to witness triangles described above is injective. Thus, the checker picks $O(n)$ sets of three vertices at random and outputs PASS if and only if none of the triangles forms a cycle. We now show how to obtain a constant time spot-checker.

Let B_i denote the set of vertices in $[i + 1, n]$ that have bad edges to i. Let $G_i = [i + 1, n] \backslash B_i$. By Observation 19, $|G_i| \geq |B_i|$.

We are now ready to state our main theorem for spot-checking total orders. First we describe the spot-checker:

```
Procedure Total-Order-Check (H_≺):
choose O(1) random vertices X from H_≺
if the graph induced by H_≺ on X is not acyclic
   return FAIL
return PASS
```

Theorem 21 `Total-Order-Check` *is an ϵ-spot-checker for the total order problem and runs in constant time.*

Proof. Let $H_\prec$ be such that $P(H_\prec) =$ TOTAL ORDER. If $H_\prec$ is acyclic, the spot-checker outputs PASS. Conversely, suppose the fraction of bad edges is at least c. There is a constant $c' = c/(2-c)$, and a set S, with $|S| \geq (c/2)n$, such that for all $i \in S$, $|B_i| \geq c'n$. This is because if the number of i such that $|B_i| \geq c'n$ is less than $(c/2)n$ then the maximum number of bad edges in the graph is less than $M = \{(c/2)n - 1\}(n) + \{n - (c/2)n + 1\}(c'n - 1)$. Now for $c' = c/(2-c)$, $M \leq cn$ which is a contradiction as $H_\prec$ has at least cn bad edges.

Call (i, x, y) a *witness-triple*, where $i \in S, x \in B_i, y \in G_i$ and $(y, x) \in H_\prec$. Since we have $(x, i), (i, y) \in H_\prec$, locating a witness-triple is tantamount to causing the spot-checker to output FAIL.

For $i \in S$, we have $|B_i| \geq c'n$. We now consider the interaction between B_i and G_i for an $i \in_R S$. The outline of the argument is: first, if most edges between G_i and B_i in $H_\prec$ go from G_i to B_i, then the spot-checker detects witness-triples with constant probability. If this does not occur, then, most edges *must* go from B_i to G_i. We then argue that this scenario violates the optimality assumption of the order. Hence, the former case should indeed occur and thus witness-triples are detected with constant probability.

Suppose at least k_0 fraction of edges between G_i and B_i are pointed from G_i to B_i. (We will fix k_0 later.) The spot checker looks at a constant-sized sample of the vertices. Since $|S| \geq (c/2)n$, the probability that the spot-checker hits S is at least $c/2$. Since $|G_i| \geq |B_i| \geq c'n$, for each $i \in S$, the sample will also contain an $x \in B_i$ and a $y \in G_i$ with probability at least c'^2. Now, since k_0 fraction of edges go from G_i to B_i, and x and y are uniformly distributed in B_i and G_i respectively, with probability $k_0 c c'^2/2$, (i, x, y) is a witness-triple. (To boost the probability that the checker will pick a witness-triple by a factor of d, one has to increase the number of vertices proportional to $\lg d$.)

Assume now that less than k_0 fraction of edges between G_i and B_i are pointing from G_i to B_i. Let $k_4 = |G_i|/|B_i|$. Thus, $1 \leq k_4 \leq 2(1/c - 1)$. Fix k_2 to be $1/(48k_4)$, and pick k_1 such that $12/(1-k_2) < k_1 < (1-k_2)/(2k_2k_4)$. Finally let k_0 be such that $k_0 \leq c'^2 k_2/k_1$.

Call $x \in B_i$ *typical* if at most $|B_i|/k_1$ edges from G_i are directed to x. Observe that at least $(1-k_2)$ fraction of the vertices of B_i are typical, for otherwise, the number of edges from G_i to B_i is at least $(k_2|B_i|) \cdot (|B_i|/k_1) \geq (c'^2 k_2/k_1)n^2 \geq k_0 n^2$, which is a contradiction since it violates the assumption about the fraction of the edges between G_i and B_i that point from G_i to B_i.

In the list of vertices that succeed i in the optimal ordering, consider the vertex j such that there are $3|B_i|/k_1$ vertices from G_i between i and j. Let $1 - k_3$ be the fraction of typical vertices between i and j. The two cases are:

[$k_3 > 3/4$:] In this case, we claim that by moving all the vertices in B_i (without disrupting the ordering among them) ahead of all the vertices in G_i, we can cut down the number of bad edges, thus contradicting the optimality of the ordering. We now analyze the number of bad edges eliminated and added by this operation. This operation must add new bad edges from the following possibilities: (i) all the edges between G_i and $k_2|B_i|$ non-typical vertices could become bad; by counting, we have at most $k_2|B_i||G_i|$ of them, and (ii) for the $(1-k_2)|B_i|$ typical vertices, the edges that were originally pointed from G_i could turn bad; by counting, we have at most $(1-k_2)|B_i||B_i|/k_1$ of them. This operation may eliminate bad edges as per the following: for at least $k_3(1-k_2)|B_i|$ typical vertices, at least $2|B_i|/k_1$ of the edges that were originally bad (i.e., pointing from these typical vertices back to vertices in G_i that preceded them) turn good; by counting, the number of bad edges eliminated is at least $(2|B_i|/k_1) \cdot k_3(1-k_2)|B_i|$. By our choice of k_1, $k_1 \leq (1-k_2)/(2k_2k_4)$, and by our assumption that $k_3 > 3/4$ the new ordering has fewer bad edges.

[$k_3 \leq 3/4$:] In this case, we show that one can relocate i just after j to reduce the number of bad edges, contradicting the optimality of the ordering. The number of new bad edges added by this relocation is at most $3|B_i|/k_1$ while the number of bad edges eliminated is $\geq (1-k_3)(1-k_2)|B_i| \geq (1-k_2)|B_i|/4$. Since $k_1 \geq 12/(1-k_2)$, and $k_3 \leq 3/4$, the net change in the number of bad edges is negative. □

4 Algebraic Structures

In this section, we describe methods for testing whether a given operation is close to a group (Section 4.1) or field (Section 4.2) operation. We begin by assuming that the operation is cancellative and in Section 4.3, we describe how to extend both testers to the noncancellative case.

PRELIMINARIES. Suppose we are given a program P purporting to compute a group or field operation f as follows. On input a finite set G, program P and function f output the tables for binary operations $\circ$ and $\diamond$ respectively on G. Let $x \circ y$ (resp. $x \diamond y$) denote the (x, y) entry from the table produced by P (resp. by f) on G. We assume that an entry in the table representing $\circ$ can be accessed in constant time. We assume that equality tests on two elements in G can be done in constant time and also that a random element can be chosen in constant time.

We say that $\circ$ is cancellative if for all a, b, c, $(a \circ c = b \circ c) \Rightarrow a = b$ and $(a \circ b = a \circ c) \Rightarrow b = c$. We use the following distance function: $\Delta(\langle G, \circ\rangle, \langle H, \diamond\rangle)$ is infinite if $G \neq H$ and is $\Pr_{a,b\in G}[a \circ b \neq a \diamond b]$ otherwise.

We denote an element α which is chosen with distribution D from G or has distribution D in G by $\alpha \in_D G$. The notation $\Pr_\alpha[\cdot]$ is synonymous with $\Pr_{\alpha\in_R G}[\cdot]$.

The L_1-*distance* between two discrete distributions D, D' on G is defined to be $\sum_{x\in G} |D(x) - D'(x)|$ where $D(x)$ (resp. $D'(x)$) denotes the probability of generating x according to D (resp. D'). A distribution is ϵ-*uniform* if its L_1-distance to the uniform distribution is $\leq \epsilon$.

Let $T_\circ$ be the $|G| \times |G|$ cancellative Cayley table (i.e., the operation table) corresponding to $\circ$. In this case, each row and column of $T_\circ$ is a permutation of elements in G. Using these, we can make the following simple observation.

Observation 22 *If $\circ$ is cancellative, then for any $b \in G$, if $\alpha \in_R G \Rightarrow \alpha \circ b \in_R G$.*

Note that if $\circ$ is cancellative then for any a, if $\alpha_1 \in_R G$ and $\alpha_1 \circ \alpha_2 = a$, then $\alpha_2 \in_R G$, though α_2 is not independent from α_1. For a cancellative $\circ$, let $\mathrm{LI}(\alpha, a)$ denote the unique α' such that $\alpha' \circ \alpha = a$ and let $\mathrm{RI}(\alpha, a)$ denote the unique α' such that $\alpha \circ \alpha' = a$. We now define what it means for two operations to be close to each other.

Definition 23 *Let $\circ$ and $\circ'$ be binary operations over domain G. $\circ$ is* ϵ-close *to $\circ'$ if $\Pr_{\alpha,\beta\in G}[\alpha \circ \beta = \alpha \circ' \beta] \geq 1 - \epsilon$.*

We extend this notion to define an almost (abelian) group.

Definition 24 *Let $\circ$ be a closed binary operation on G. $(G, \circ)$ is an* ϵ-(abelian) group *if there exists a binary operation $\circ'$ that is ϵ-close to $\circ$ such that $(G, \circ')$ is an (abelian) group.*

This notion can be extended to fields as well.

Definition 25 *Let $\circ, \diamond$ be closed binary operations on G. $(G, \circ, \diamond)$ is an (ϵ_1, ϵ_2)*-field *if there exist binary operations $\circ'$ (resp. $\diamond'$) that is ϵ_1-close to $\circ$ (resp. ϵ_2-close to $\diamond$) such that $(G, \circ', \diamond')$ is a field.*

REMARK ON CONFIDENCE. Our tests rely on random sampling to determine whether a bad event happens with probability more than ϵ. It requires $O(\frac{1}{\epsilon} \ln \frac{1}{\rho})$ trials to ascertain this with a confidence of ρ.

4.1 Groups

We assume the spot-checker is given a table for P (i.e., $\circ$); the values of f (i.e., $\diamond$) on a small number of selected inputs, specifically, the values of $g \diamond a, \quad \forall g \in S_G, a \in G$, where S_G is a set of generators of G with respect to $\diamond$ (we note below that this representation has size $\tilde{O}(|G|)$); and parameter ϵ. We present a method for spot-checking very efficiently whether $\circ$ is ϵ-close to a specific $\diamond$ such that $\diamond$ is a group operation. Though the output of P is of size $O(|G|^2)$, for any given distance ϵ our checker runs in $\tilde{O}(|G|/\epsilon)$ time. In this section, we assume that $\circ$ is known to be cancellative. Cancellativity is a necessary but not sufficient condition for an operation to be a group. We make this assumption in order to simplify the tests and the proofs. In Section 4.3 we sketch briefly how to handle the case when $\circ$ is not known to be cancellative.

4.1.1 The Test

In order to test that $\circ$ is close to $\diamond$, we check the following: (i) $\circ$ is close to some cancellative associative operation $\circ'$, (ii) $\circ'$ has an identity element, (iii) each element in G has an inverse under $\circ'$, and (iv) $\circ'$ is close to $\diamond$. We will show a way of computing $\circ'$ in constant time by making calls to $\circ$ for testing properties (ii) through (iv).

If P passes tests (i) through (iii), then one can show the existence of a group operation $\circ'$ that differs from $\circ$ on at most 4ϵ fraction of $G \times G$. In the final stage we test (iv), whether $\circ$ is computing the *specific* group operation $\diamond$.

Observation 26 *G has a set S_G of generators of size* $\lg |G|$.

The most interesting and challenging part of checking whether a given operation is close to a group is to design a method of checking that the operation is close to associative. The first $o(|G|^3)$ algorithm for checking if $\circ$ is associative is given in [RaS96]. In particular, their randomized algorithm runs in $O(|G|^2)$ steps for cancellative operations. They also give a lower bound which shows that any randomized algorithm required $\Omega(|G|^2)$ steps to verify associativity, even in the cancellative case. Despite this lower bound, we show that one can check if $\circ$ is "close" to an associative function table — i.e., if there is an associative operation which agrees with $\circ$ on a large fraction of $G \times G$ — in only $\tilde{O}(|G|)$ steps.

ASSOCIATIVITY. For (i), we describe our check that the table for $\circ$ is associative. To do this, the checker repeats each of following checks several times (the number to be determined shortly) and fails the program if any one fails. All of the elements come from G.

(1) Pick random β, γ; check for all a that $a \circ (\beta \circ \gamma) = (a \circ \beta) \circ \gamma$.

(2) Pick random α, β; check for all c that $\alpha \circ (\beta \circ c) = (\alpha \circ \beta) \circ c$.

(3) Pick random α, γ; check for all b that $\alpha \circ (b \circ \gamma) = (\alpha \circ b) \circ \gamma$.

If $\circ$ passes this test, then with high probability it must have the following properties:

(T1) $\Pr_{\beta,\gamma}[\forall a,\ a \circ (\beta \circ \gamma) = (a \circ \beta) \circ \gamma] \geq 1 - \epsilon$,

(T2) $\Pr_{\alpha,\beta}[\forall c,\ \alpha \circ (\beta \circ c) = (\alpha \circ \beta) \circ c] \geq 1 - \epsilon$, and

(T3) $\Pr_{\alpha,\gamma}[\forall b,\ \alpha \circ (b \circ \gamma) = (\alpha \circ b) \circ \gamma] \geq 1 - \epsilon$.

Since our definition of a result-checker includes a confidence parameter, and since we have $O(|G|)$ probabilistic tests for each (1), (2), and (3), the overall confidence δ has to be apportioned. It is easy to see that it is sufficient to repeat each test $O((1/\epsilon) \cdot \lg(|G|/\delta)) = (1/\epsilon) \cdot (\lg |G| - \lg \delta)$ times.

The following theorem states that the above properties are sufficient to conclude that $\circ$ is close to being a group operation. We postpone the proof to the next section.

Theorem 27 *Let $\epsilon < 1/15$. If $\circ$ is a cancellative operation on G and satisfies* (T1) *through* (T3) *above, then there is a cancellative associative operation $\circ'$ on G satisfying*

1. $\forall b \in G, \Pr_\alpha[\alpha \circ' b = \alpha \circ b] \geq 1 - 4\epsilon$
2. $\forall a \in G, \Pr_\beta[a \circ' \beta = a \circ \beta] \geq 1 - 4\epsilon$.

In fact, we will see how to construct $\circ'$ such that it is computable (in $O(\lg 1/\delta)$ time) with a probability of $1 - \delta$ being correct.

For $a, b \in G$, let

$$a \circ' b = \operatorname*{maj}_{\beta\circ\gamma=b} \{(a \circ \beta) \circ \gamma\}.$$

The intuition behind taking a majority vote is that if $\circ$ were associative, we would have $(a \circ \beta) \circ \gamma = a \circ (\beta \circ \gamma) = a \circ b$. By defining $\circ'$ to be a majority over all $\beta \circ \gamma = b$, we will show that $\circ'$ is a corrected version of $\circ$.

To compute $\circ'$ efficiently, we use the standard *self-corrector* algorithm (cf. [BLR93, Lip91]). On inputs $b \in G$ and security parameter δ, pick $\beta \in_R G$, and then set $\gamma = RI(\beta, b)$. Similar to Observation 22, we have that $\gamma \in_R G$. Set $s = (a \circ \beta) \circ \gamma$. If there really is a majority answer for $a \circ' b$, then this will output the majority answer with probability $1/2$. We will show that the majority answer will be output with probability $1 - 3\epsilon$ (Lemma 30). The self-corrector repeats this computation $O(\lg 1/\delta)$ times, checks that s is always set to the same value, and if so, outputs s and otherwise outputs FAIL (since $\circ$ is clearly not a group). By Lemma 30, $s = a \circ' b$ with probability at least $1 - \delta$.

Computing $RI(\cdot, b)$ takes time $O(|G|)$. Another way to implement this is to have several random β and γ such that $\beta \circ \gamma = b$ at hand. In order to make available a sufficient number of such pairs ($O(\lg 1/\delta)$, where δ is an upper bound on the probability of outputting a wrong answer), the checker can generate several α_1, α_2 pairs, storing the pair in the bucket labeled $\alpha_1 \circ \alpha_2$. By a coupon-collector argument, the samples collected will, with high probability, provide a sufficiently large sample for each $b \in G$ so that $\circ'$ can be computed from them correctly with high probability. Note that the overhead for each computation of $\circ'$ need only be $O(\lg 1/\delta)$. From now on, we can assume $\circ'$ is available. However, if the self-corrector has to be called k times, then it should be given a security parameter of δ/k so that using the union bound it can be assumed that *all* the calls are correct with probability at least $1 - \delta$. In our tests, $k = O(|G|)$, so the running time per call to the self-corrector is $O(\lg |G| + \lg 1/\delta)$. We use the $\tilde{O}(\cdot)$ notation to absorb the dependence on $\lg |G|$. Also, as mentioned earlier, we suppress the dependence on δ.

IDENTITY AND INVERSE. The following procedure shows how to test whether $\circ'$ has an identity. For any element a, by cancellativity, there is a b such that $a \circ' b = a$ which can be found in $O(|G|)$ time by trying all possible b's. Then, b should be the identity e, if G were to be a group. That e is an identity can be verified in $O(|G|)$ time by checking $e \circ b = e = b \circ e$ for all b. Note that the cancellativity of $\circ'$ implies that e is unique: if e' were also an identity, $a \circ' e = a = a \circ' e' \Rightarrow e' = e$.

Now, since $\circ'$ is cancellative, for every $a \in G$, there is a $b \in G$ such that $a \circ' b = e$. In other words, each $a \in G$ has an inverse and (iii) follows without any additional tests.

EQUALITY. Finally, we have to check if $\circ'$ is the same as $\diamond$, the specific group operation (*equality testing*, [BLR93, RS96]). To do this in $|G| \lg |G|$ steps, check $\forall b \in G, g \in S_G$ if $g \circ' b = g \diamond b$, where the latter is given. To see that this uniquely identifies the group, we induct on $|b|$, the length of the string when b is expressed in terms of $\diamond$ and elements from S_G. Suppose for $k > 1$, $a = g_1 \diamond \cdots \diamond g_k$. Then, by induction $a \circ' b = (a' \diamond g) \circ' b = (a' \circ' g) \circ' b = a' \circ' (g \circ' b) = a' \circ' (g \diamond b) = a' \diamond (g \diamond b) = (a' \diamond g) \diamond b = a \diamond b$, where $a' = g_1 \diamond \cdots \diamond g_{k-1}, g = g_k$, the claim follows.

The required number of repetitions for identity, inverse, and equality tests can be derived using a similar argument to that involving the associativity test, as a result of which the following theorem ensues.

Theorem 28 *For $\epsilon < 4/15$ and for a cancellative $\circ$, there is an ϵ-spot-checker that runs in $\tilde{O}(|G|/\epsilon)$ time for spot-checking if $(G, \circ)$ is a group.*

4.1.2 Associativity

This section is dedicated to proving Theorem 27.

Proof. [of Theorem 27] The following series of lemmas establish the theorem. Lemma 30 shows that $\circ'$ is well-defined and Lemma 31 shows $\circ'$ is cancellative. Then, Lemma 32 shows that $\circ'$ agrees with $\circ$ on a large fraction of $G \times G$. Lemma 33 proves an intermediate step that is used in Lemma 34, which finally eliminates all probabilistic quantifiers. □

The following lemma is an easy consequence of (T3):

Lemma 29 *Given b, if $\beta_2, \gamma_1 \in_R G$ and $\beta_1 = \mathrm{LI}(\beta_2, b), \gamma_2 = \mathrm{RI}(\gamma_1, b)$, and $\delta = \mathrm{LI}(\beta_2, \gamma_2)$, then $\Pr[(\gamma_1 \circ \delta) = \beta_1] \geq 1 - \epsilon$.*

Proof. Note that β_1, γ_2, and δ exist and are uniformly distributed by the cancellativity of $\circ$. Then, $\beta_1 \circ \beta_2 = b = \gamma_1 \circ \gamma_2 = \gamma_1 \circ (\delta \circ \beta_2) = (\gamma_1 \circ \delta) \circ \beta_2$, with the last step true with probability $1 - \epsilon$ by (T3). Since $\circ$ is cancellative, we have $\beta_1 = \gamma_1 \circ \delta$. □

First, we show $\circ'$ is well-defined for $\epsilon < 1/6$. For a given $b \in G, \beta_1, \gamma_1 \in_R G$, let $\beta_2, \gamma_2, \delta$ be such that $\beta_1 \circ \beta_2 = \gamma_1 \circ \gamma_2$ and $\delta \circ \beta_2 = \gamma_2$. Note that $\gamma_1, \delta, \beta_2$ are pairwise independent random variables. Using Lemma 29 and (T1), we have for given $a, b \in G$, $\Pr[(a \circ \beta_1) \circ \beta_2 = (a \circ (\gamma_1 \circ \delta)) \circ \beta_2 = ((a \circ \gamma_1) \circ \delta) \circ \beta_2 = (a \circ \gamma_1) \circ (\delta \circ \beta_2) = (a \circ \gamma_1) \circ \gamma_2] \geq 1 - 3\epsilon$. Since $\circ'$ is defined to be the majority over $\beta \circ \gamma = b$ of $\{(a \circ \beta) \circ \gamma\}$, and since the collision probability lower bounds the probability of the most likely element, we obtain the following lemma.

Lemma 30 *For all $a, b \in G$, $\Pr_{\beta_1}[a \circ' b = (a \circ \beta_1) \circ \beta_2,$ where $\beta_1 \circ \beta_2 = b] \geq 1 - 3\epsilon$.*

The following lemma shows that $\circ'$ is cancellative. This will be useful for the rest of the discussion.

Lemma 31 *If $a \circ' b = a \circ' c$, then $b = c$. If $a \circ' c = b \circ' c$, then $a = b$.*

Proof. Let $\beta \in_R G$. Let $\alpha_1, \alpha_2 \in G$ be such that $\alpha_1 \circ \beta = b$, $\alpha_2 \circ \beta = c$ and $(a \circ \alpha_1) \circ \beta = a \circ' b = a \circ' c = (a \circ \alpha_2) \circ \beta$ holds. Note that such $\alpha_1, \alpha_2, \beta$ exist by Lemma 30. Now, by the cancellativity of $\circ$, we have first $a \circ \alpha_1 = a \circ \alpha_2$ and next $\alpha_1 = \alpha_2$, thus finally $b = c$.

Let $\beta_1 \in_R G$. Let $\beta_2 \in G$ be such that $\beta_1 \circ \beta_2 = c$ and $(a \circ \beta_1) \circ \beta_2 = a \circ' c = b \circ' c = (b \circ \beta_1) \circ \beta_2$ holds. Note that such β_1, β_2 exist by Lemma 30. Now, by the cancellativity of $\circ$, we have $a \circ \beta_1 = b \circ \beta_1$ and hence $a = b$. □

The following lemma proves part of Theorem 27 — that $\circ'$ agrees with $\circ$.

Lemma 32 $\forall b, \Pr_\alpha[\alpha \circ' b = \alpha \circ b] \geq 1 - 4\epsilon$. $\forall a, \Pr_\beta[a \circ' \beta = a \circ \beta] \geq 1 - 4\epsilon$.

Proof. Let $\beta_1 \in_R G$ and β_2 be such that $\beta_1 \circ \beta_2 = b$. We have $\alpha, \beta_1 \in_R G$. $\Pr_{\beta_1}[\alpha \circ' b = (\alpha \circ \beta_1) \circ \beta_2 = \alpha \circ (\beta_1 \circ \beta_2) = \alpha \circ b] \geq 1 - 4\epsilon$, where the first equality follows from Lemma 30 and the second equality follows using (T2).

Similarly, $\Pr_{\beta_1}[a \circ' \beta = (a \circ \beta_1) \circ \beta_2 = a \circ (\beta_1 \circ \beta_2) = a \circ \beta] \geq 1 - 4\epsilon$, where the first equality follows from Lemma 30 and the second equality follows using (T1). □

The following is a useful step in proving the other part of Theorem 27 — that $\circ'$ is associative.

Lemma 33 *$\forall b, c$, $\Pr_{\beta_1,\gamma_1}[b \circ' c = (\beta_1 \circ (\beta_2 \circ \gamma_1)) \circ \gamma_2] \geq 1 - 4\epsilon$, where $\beta_2 = \mathrm{RI}(\beta_1, b)$ and $\gamma_2 = \mathrm{RI}(\gamma_1, c)$.*

Proof. Using Lemma 30 and (T1), we have $\Pr_{\beta_1,\gamma_1}[b \circ' c = (b \circ \gamma_1) \circ \gamma_2 = ((\beta_1 \circ \beta_2) \circ \gamma_1) \circ \gamma_2 = (\beta_1 \circ (\beta_2 \circ \gamma_1)) \circ \gamma_2] \geq 1 - 4\epsilon$. □

Finally, the following lemma shows $\circ'$ is associative, completing the proof of Theorem 27.

Lemma 34 *If $\epsilon < 1/15$, for all $a, b, c \in G$, $a \circ' (b \circ' c) = (a \circ' b) \circ' c$.*

Proof. Let $\beta_1, \gamma_1 \in_R G$ and $\beta_2, \gamma_2 \in G$ be such that $\beta_1 \circ \beta_2 = b, \gamma_1 \circ \gamma_2 = c$. Then, it follows that $\beta_1, \beta_2 \circ \gamma_1 \in_R G$ and $\beta_2, \gamma_1 \in_R G$. Using Lemma 33, (T1), and Lemma 30, we have $\Pr_{\beta_1,\gamma_1}[a \circ' (b \circ' c) = a \circ' ((\beta_1 \circ (\beta_2 \circ \gamma_1)) \circ \gamma_2) = (a \circ (\beta_1 \circ (\beta_2 \circ \gamma_1))) \circ \gamma_2 = ((a \circ \beta_1) \circ (\beta_2 \circ \gamma_1)) \circ \gamma_2 = (((a \circ \beta_1) \circ \beta_2) \circ \gamma_1) \circ \gamma_2 = ((a \circ \beta_1) \circ \beta_2) \circ' c = (a \circ' b) \circ' c] \geq 1 - 15\epsilon > 0$. The lemma follows since the probabilistic assertion is independent of α_2. □

Our result can be used to show that a class of functional equations is useful for testing program correctness over small domains. The class of functional equations that our results apply to are those satisfying the the associativity equation $F[F[x, y], z] = F[x, F[y, z]]$, which characterize functions of the form $F[x, y] = f(f^{-1}(x) + f^{-1}(y))$ where f is a continuous and strictly monotone function [Acz66].

4.2 Fields

We show that testing whether a cancellative $\circ$ is ϵ-close to a cancellative, associative, and commutative $\circ'$ over a domain of size $|G|$ can be done in randomized $\tilde{O}(|G|)$ time (Section 4.2.1). As in Section 4.1, we assume that $\circ$ is cancellative. Later, in Section 4.3, we show how to extend these techniques to the non-cancellative case. In Sections 4.2.2 4.2.3, we use the results of this section to test if $(G, \circ)$ is an ϵ-abelian group and if $(G, \circ, \diamond)$ is an (ϵ, ϵ)-field respectively. In Section 4.4 we show that there is an $\Omega(|G|^2)$ lower bound to check if $\circ$ is exactly (0-close to) associative and commutative.

Since the reader is by now familiar with the general outline of our arguments, we will follow a different order of presentation from the previous section.

4.2.1 Testing Associativity and Commutativity

Given a group, one may use the results of [LZ78] to test that it is abelian in constant time. We give a method in which one can test associativity and commutativity simultaneously.

We use the following equation (which we call the *AC-property*) to test:

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

We prove the following theorem which shows that if a cancellative $\circ$ satisfies some conditions that can be tested in $\tilde{O}(|G|)$ time, then it is close to a cancellative, associative, and commutative $\circ'$. Furthermore, as in the previous section, the theorem will also imply the existence of a self-corrector for $\circ'$.

Theorem 35 *Let $\epsilon < 1/21$. If $\circ$ is cancellative and satisfies*

(1) $\Pr_{\alpha}[\forall b,\ \alpha \circ b = b \circ \alpha] \geq 1 - \epsilon$,

(2) $\Pr_{\beta,\gamma}[\forall a,\ (a \circ \beta) \circ \gamma = a \circ (\gamma \circ \beta)] \geq 1 - \epsilon$,

(3) $\Pr_{\alpha,\gamma}[\forall b,\ (\alpha \circ b) \circ \gamma = \alpha \circ (\gamma \circ b)] \geq 1 - \epsilon$, *and*

(4) $\Pr_{\alpha,\beta}[\forall c,\ (\alpha \circ \beta) \circ c = \alpha \circ (c \circ \beta)] \geq 1 - \epsilon$,

then there is an $\circ'$ such that

(1) *$\circ'$ is cancellative,*

(2) $\forall a, b, c,\ a \circ' (b \circ' c) = (a \circ' b) \circ' c$,

(3) $\forall a, b,\ a \circ' b = b \circ' a$,

(4) *$\circ'$ is 5ϵ-close to $\circ$, and*

(5) *$\circ'$ is computable in constant time, given oracle access to $\circ$.*

Proof outline: Let maj denote the majority function which returns the element that occurs the most number of times in a (multi)set. Define the following binary operation $\circ'$ as follows: for $a, b \in G$, define

$$a \circ' b \stackrel{\Delta}{=} \underset{\beta\circ\gamma=b}{\mathrm{maj}} \{(a \circ \gamma) \circ \beta\}.$$

The intuition is if $\circ$ were to satisfy the AC-property, we would have $(a \circ \gamma) \circ \beta = a \circ (\beta \circ \gamma) = a \circ b$. In fact, we will see that $\circ'$ is crucial to circumvent the lower bound shown in Section 4.4.

We first show that, in some sense, $\circ'$ is well-defined (Lemma 37). We use this to show that $\circ'$ is cancellative (Lemma 38) and $\circ'$ is 5ϵ-close to $\circ$ (Lemma 39). Then, we show (Theorem 42) that if $\epsilon < 1/21$, then $\circ'$ satisfies the AC-property on all elements of G. Finally, we show (Theorem 43) that if $\epsilon < 1/13$ then $\circ'$ is commutative. Putting these together, Corollary 44 completes the proof of this theorem. □

The following lemma is an easy consequence of the hypotheses:

Lemma 36 *$\forall b$, $\Pr_{\beta_2,\gamma_2}[\beta_1 \circ \delta = \gamma_1$, where $\beta_1 = \mathrm{LI}(\beta_2, b), \gamma_1 = \mathrm{LI}(\gamma_2, b)$, and $\delta = \mathrm{LI}(\gamma_2, \beta_2] \geq 1 - 2\epsilon$.*

Proof. Note that $\gamma_2, \delta, \beta_1$ are pairwise independent random variables. Now, $\Pr_{\beta_2,\gamma_2}[\beta_1 \circ \beta_2 = \beta_1 \circ (\delta \circ \gamma_2) = \beta_1 \circ (\gamma_2 \circ \delta) = (\beta_1 \circ \delta) \circ \gamma_2] \geq 1 - 2\epsilon$. Since $\delta, \gamma_2 \in_R G$, the second equality holds with probability at least $1 - \epsilon$ by Hypothesis (1); and the third equality holds with probability at least $1 - \epsilon$ by Hypothesis (2). But, $\gamma_1 \circ \gamma_2 = b = \beta_1 \circ \beta_2$. Since $\circ$ is cancellative, we therefore have $\gamma_1 = \beta_1 \circ \delta$ with probability at least $1 - 2\epsilon$. □

First, we show $\circ'$ is well-defined. For a given $b \in G$, let $\beta_1, \gamma_1 \in_R G$ and fix $\beta_2, \gamma_2, \delta$ such that $\beta_1 \circ \beta_2 = b = \gamma_1 \circ \gamma_2$ and $\delta \circ \gamma_2 = \beta_2$. Since $\gamma_2, \delta, \beta_1$ are pairwise independent random variables, we can obtain the following probabilistic statement for given $a, b \in G$: $\Pr_{\beta_1,\gamma_1}[(a \circ \beta_2) \circ \beta_1 = (a \circ (\delta \circ \gamma_2)) \circ \beta_1 = ((a \circ \gamma_2) \circ \delta) \circ \beta_1 = (a \circ \gamma_2) \circ (\beta_1 \circ \delta) = (a \circ \gamma_2) \circ \gamma_1] \geq 1 - 4\epsilon$. Since $\delta, \gamma_2 \in_R G$, the second equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); since $\delta, \beta_1 \in_R G$, the third equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); and the fourth equality holds with probability at least $1 - 2\epsilon$ by Lemma 36. Since $\circ'$ is defined to be the majority over β, γ such that $\beta \circ \gamma = b$ of $\{(a \circ \gamma) \circ \beta\}$, by the collision argument used in the proof of Lemma 30, we obtain the following lemma.

Lemma 37 *For all $a, b \in G$, $\Pr_{\beta_1}[a \circ' b = (a \circ \beta_2) \circ \beta_1,$ where $\beta_1 \circ \beta_2 = b] \geq 1 - 4\epsilon$.*

We first show that $\circ'$ is cancellative.

Lemma 38 (Cancellativity) *Let $\epsilon < 1/8$. If $a \circ' b = a \circ' c$, then $b = c$. If $a \circ' c = b \circ' c$, then $a = b$.*

Proof. For $\beta \in_R G$, let α_1, α_2 be such that $\beta \circ \alpha_1 = b$ and $\beta \circ \alpha_2 = c$. Then, $\Pr_\beta[(a \circ \alpha_1) \circ \beta = a \circ' b = a \circ' c = (a \circ \alpha_2) \circ \beta] \geq 1 - 8\epsilon$, by Lemma 37. Now, repeatedly using the cancellativity of $\circ$, we have first $a \circ \alpha_1 = a \circ \alpha_2$ and next $\alpha_1 = \alpha_2$, thus finally $b = c$.

Let $\beta_1 \in_R G$ and fix β_2 such that $\beta_1 \circ \beta_2 = c$. Then, $\Pr_{\beta_1}[(a \circ \beta_2) \circ \beta_1 = a \circ' c = b \circ' c = (b \circ \beta_2) \circ \beta_1] \geq 1 - 8\epsilon$, by Lemma 37. Now, by the cancellativity of $\circ$, we have $a \circ \beta_2 = b \circ \beta_2$ and hence $a = b$.

If $\epsilon < 1/8$, the above probabilistic statements are independent of their respective random variables and hence always hold. □

The following lemma proves part (4) of Theorem 35.

Lemma 39 (Closeness) $\forall b, \Pr_\alpha[\alpha \circ' b = \alpha \circ b] \geq 1 - 5\epsilon$. $\forall a, \Pr_\beta[a \circ' \beta = a \circ \beta] \geq 1 - 5\epsilon$.

Proof. Let $\beta_1 \in_R G$ be such that $\beta_1 \circ \beta_2 = b$. Now, α, β_1 are independent random variables. Therefore, $\Pr_{\alpha,\beta_1}[\alpha \circ' b = (\alpha \circ \beta_2) \circ \beta_1 = \alpha \circ (\beta_1 \circ \beta_2) = \alpha \circ b] \geq 1 - 5\epsilon$, where the first equality follows with probability at least $1 - 4\epsilon$ from Lemma 37 and the second equality follows with probability at least $1 - \epsilon$ using Hypothesis (3).

Similarly, let $\beta_1 \in_R G$ be such that $\beta_1 \circ \beta_2 = \beta$. Since $\beta \in_R G$, we have that β_1, β_2, β are pairwise independent random variables. Therefore, $\Pr_{\beta,\beta_1}[a \circ' \beta = (a \circ \beta_2) \circ \beta_1 = a \circ (\beta_1 \circ \beta_2) = a \circ \beta] \geq 1 - 5\epsilon$, where the first equality follows with probability at least $1 - 4\epsilon$ from Lemma 37 and the second equality follows with probability at least $1 - \epsilon$ using Hypothesis (2). □

We show that $\circ'$ satisfies the AC-property. The following is a simple corollary of Hypothesis (1).

Corollary 40 $\Pr_{\alpha_1}[\forall a,\ \alpha_2 \circ \alpha_1 = a$, *where* $\alpha_2 = \mathrm{RI}(\alpha_1, a)] \geq 1 - \epsilon$.

Proof. For $\alpha_1 \circ \alpha_2 = a$, since $\alpha_1 \in_R G$, by Hypothesis (1), we have $a = \alpha_1 \circ \alpha_2 = \alpha_2 \circ \alpha_1$ with probability at least $1 - \epsilon$. □

The next lemma is a useful intermediate step in proving that $\circ'$ satisfies the AC-property.

Lemma 41 $\forall b, c, \Pr_{\beta_1,\gamma_1}[(\gamma_1 \circ (\gamma_2 \circ \beta_1)) \circ \beta_2 = (c \circ' b)$, *where* $\beta_2 = \mathrm{RI}(\beta_1, b)$ *and* $\gamma_2 = \mathrm{RI}(\gamma_1, c)] \geq 1 - 7\epsilon$.

Proof. Consider the following probabilistic statement: $\Pr_{\beta_1,\gamma_1}[(\gamma_1 \circ (\gamma_2 \circ \beta_1)) \circ \beta_2 = (\gamma_1 \circ (\beta_1 \circ \gamma_2)) \circ \beta_2 = ((\gamma_1 \circ \gamma_2) \circ \beta_1) \circ \beta_2 = (c \circ \beta_1) \circ \beta_2 = c \circ' (\beta_2 \circ \beta_1) = (c \circ' b)] \geq 1 - 7\epsilon$. Since $\beta_1, \gamma_2 \in_R G$, the first equality holds with probability at least $1 - \epsilon$ by Hypothesis (1); since $\beta_1, \gamma_2 \in_R G$, the second equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); and since $\beta_1 \in_R G$, the fourth equality holds with probability at least $1 - 4\epsilon$ by Lemma 37; and since $\beta_1 \in_R G$, the fifth equality holds with probability at least $1 - \epsilon$ by Corollary 40. □

Finally, the following theorem shows that $\circ'$ satisfies the AC-property.

Theorem 42 (AC-property) *If* $\epsilon < 1/21$, *for all* $a, b, c \in G$, $(a \circ' b) \circ' c = a \circ' (c \circ' b)$.

Proof. Let $\beta_1, \gamma_1 \in_R G$ be such that $\beta_1 \circ \beta_2 = b$ and $\gamma_1 \circ \gamma_2 = c$. Now, consider the following probabilistic statement: $\Pr_{\beta_1,\gamma_1}[(a \circ' b) \circ' c = ((a \circ \beta_2) \circ \beta_1) \circ' c = (((a \circ \beta_2) \circ \beta_1) \circ \gamma_2) \circ \gamma_1 = ((a \circ \beta_2) \circ (\gamma_2 \circ \beta_1)) \circ \gamma_1 = (a \circ \beta_2) \circ (\gamma_1 \circ (\gamma_2 \circ \beta_1)) = a \circ' ((\gamma_1 \circ (\gamma_2 \circ \beta_1)) \circ \beta_2) = a \circ' (c \circ' b) \geq 1 - 21\epsilon > 0$. Since $\beta_1 \in_R G$, the first equality holds with probability at least $1 - 4\epsilon$ by Lemma 37; since $\gamma_1 \in_R G$, the second equality holds with probability at least $1 - 4\epsilon$ by Lemma 37; since $\beta_1, \gamma_2 \in_R G$, the third equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); since $a \circ \beta_2, \gamma_1 \in_R G$, the fourth equality holds with probability at least $1 - \epsilon$ by Hypothesis (3); since $\beta_2 \in_R G$, the fifth equality holds with probability at least $1 - 4\epsilon$ by Lemma 37; and the last equality holds with probability at least $1 - 7\epsilon$ by Lemma 41. Since the probabilistic statement is independent of β_1, γ_1 and holds with non-zero probability, it must hold with probability 1. □

The following theorem shows that $\circ'$ is also commutative.

Theorem 43 (Commutativity) *If $\epsilon < 1/13$, for all $a, b \in G$, $a \circ' b = b \circ' a$.*

Proof. First, let $\alpha_1 \circ \alpha_2 = a, \beta_1 \circ \beta_2 = b$ for $\alpha_1, \alpha_2, \beta_1, \beta_2 \in_R G$. Consider the following probabilistic statement: $\Pr_{\alpha_1,\beta_1}[a \circ' b = (a \circ \beta_2) \circ \beta_1 = ((\alpha_1 \circ \alpha_2) \circ \beta_2) \circ \beta_1 = (\alpha_1 \circ (\beta_2 \circ \alpha_2)) \circ \beta_1 = \alpha_1 \circ (\beta_1 \circ (\beta_2 \circ \alpha_2)) = (\beta_1 \circ (\beta_2 \circ \alpha_2)) \circ \alpha_1 = (\beta_1 \circ (\alpha_2 \circ \beta_2)) \circ \alpha_1 = ((\beta_1 \circ \beta_2) \circ \alpha_2) \circ \alpha_1 = (b \circ \alpha_2) \circ \alpha_1 = b \circ' a] \geq 1 - 13\epsilon > 0$. Since $\beta_1 \in_R G$, the first equality holds with probability at least $1 - 4\epsilon$ by Lemma 37; since $\beta_2, \alpha_2 \in_R G$, the third equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); since $\beta_2 \circ \alpha_2, \beta_1 \in_R G$, the fourth equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); since $\alpha_1 \in_R G$, the fifth equality holds with probability at least $1 - \epsilon$ by Hypothesis (1); since $\alpha_2 \in_R G$, the sixth equality holds with probability at least $1 - \epsilon$ by Hypothesis (1); since $\alpha_2, \beta_2 \in_R G$, the seventh equality holds with probability at least $1 - \epsilon$ by Hypothesis (2); and since $\alpha_1 \in_R G$, the last equality holds with probability at least $1 - 4\epsilon$ by Lemma 37. Since the probabilistic statement is independent of α_1, β_1 and holds with non-zero probability, it must hold with probability 1. □

Using Theorem 42 and Theorem 43, we get part (3) of the Theorem 35:

Corollary 44 (Associativity) *If $\epsilon < 1/21$, for all $a, b, c \in G$, $(a \circ' b) \circ' c = a \circ' (b \circ' c)$.*

4.2.2 Testing Abelian Group Operations

To check if $(G, \circ)$ is an ϵ-abelian group, we check if a cancellative $\circ$ is ϵ-close to an $\circ'$ such that $\circ'$ has the following properties: (i) $\circ'$ is associative, (ii) $\circ'$ is commutative, (iii) $\circ'$ has an identity element, and (iv) each element in G has an inverse under $\circ'$.

For (i) and (ii), we appeal to Theorem 35 which shows that if $\circ$ satisfies certain conditions (which can be verified by random sampling), then is it 5ϵ-close to an $\circ'$ that is both commutative and associative. Also, the theorem shows that for any $a, b \in G$, $a \circ' b$ can be computed correctly in $O(1)$ time (with high probability).

Properties (iii) and (iv) follow as in the previous section on testing groups (Section 4.1.1).

Theorem 45 *For $\epsilon < 5/21$ and for a cancellative $\circ$, there is an ϵ-spot-checker that runs in $\tilde{O}(|G|/\epsilon)$ time for spot-checking if $(G, \circ)$ is an abelian group.*

4.2.3 Testing Field Operations

In this section, we show how to test in $\tilde{O}(|G|)$ randomized time if $\circ$ (resp. $\diamond$) is ϵ-close to $\circ'$ (resp. $\diamond'$) such that $(G, \circ', \diamond')$ constitutes a field. As before, we assume both $\circ$ and $\diamond$ are cancellative.

Theorem 46 *For $\epsilon < 5/44$ and for a cancellative $\circ, \diamond$, $\tilde{O}(|G|/\epsilon)$ time for spot-checking if $(G, \circ, \diamond)$ is a field.*

Proof. We outline the steps involved below:

(i) Using Theorem 45, we can test if $(G, \circ)$ is a 5ϵ-abelian group.

(ii) We would like to know if $\diamond$ satisfies distributive laws, i.e.,

$$a \diamond (b \circ' c) = (a \diamond b) \circ' (a \diamond c) \qquad \text{and} \qquad (a \circ' b) \diamond c = (a \diamond c) \circ' (b \diamond c).$$

We use the following theorem which can be inferred from [BLR93]:

Theorem 47 (BLR93) *Let $\epsilon < 1/44$. If $\diamond$ satisfies*

(5) $\Pr_{a,b,c,d}[(a \circ' b) \diamond (c \circ' d) = (a \diamond c) \circ' (a \diamond d) \circ' (b \diamond c) \circ' (b \diamond d)] > 1 - \epsilon$,

for an abelian group operation $\circ'$, then the operator defined by

$$a \diamond' b = \operatorname*{maj}_{\alpha_1 \circ' \alpha_2 = a, \beta_1 \circ' \beta_2 = b} \{(\alpha_1 \diamond \beta_1) \circ' (\alpha_1 \diamond \beta_2) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2)\}$$

is bilinear, 2ϵ-close to $\diamond$, and satisfies $\forall a, b$,

$$\Pr_{\alpha_1, \beta_1}[a \diamond' b = (\alpha_1 \diamond \beta_1) \circ' (\alpha_1 \diamond \beta_2) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2), \text{ where } \alpha_1 \circ' \alpha_2 = a, \beta_1 \circ' \beta_2 = b\,] \geq 1 - 8\epsilon.$$

Using this theorem, we can perform $O(1)$ tests to ensure that $\diamond$ is 2ϵ-close to a bilinear $\diamond'$.

(iii) Now, we set out to establish $\diamond'$ is also cancellative. We need an additional $O(|G|)$ tests on $\diamond$ that essentially checks if $\diamond$ distributes over $\circ'$. More precisely, we prove the following lemma:

Lemma 48 *Let $\epsilon < 1/18$. If $\diamond$ satisfies the following additional hypotheses*

(6) $\Pr_{\alpha,\beta_1}[\forall b,\ \alpha \diamond b = (\alpha \diamond \beta_1) \circ' (\alpha \diamond \beta_2),$ *where* $\beta_1 \circ' \beta_2 = b] \geq 1 - \epsilon$, *and*

(7) $\Pr_{\alpha_1,\beta}[\forall a,\ a \diamond \beta = (\alpha_1 \diamond \beta) \circ' (\alpha_2 \diamond \beta),$ *where* $\alpha_1 \circ' \alpha_2 = a] \geq 1 - \epsilon$.

then $\diamond'$ is cancellative.

Proof. First, we show for any b, if $a \diamond' b = a' \diamond' b$ then $a = a'$. Let $\epsilon < 1/18$. First, for $\alpha_1, \alpha_2, \beta_1, \beta_2 \in_R$ such that $\alpha_1 \circ' \alpha_2 = a$, $\beta_1 \circ' \beta_2 = b$, we have that $\Pr_{\alpha_1,\beta_1}[a \diamond' b = (\alpha_1 \diamond \beta_1) \circ' (\alpha_1 \diamond \beta_2) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2)$ $= (\alpha_1 \diamond b) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2)] \geq 1 - 9\epsilon$, where the first equality holds with probability at least $1 - 8\epsilon$ by Theorem 47 and the second equality holds with probability at least $1 - \epsilon$ by Hypothesis (6).

Similarly, for $\alpha_1' \circ' \alpha_2 = a'$, $\beta_1 \circ' \beta_2 = b$, $\Pr_{\alpha_1',\beta_1}[a' \diamond' b = (\alpha_1' \diamond b) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2)] \geq 1 - 9\epsilon$. Now, since $a \diamond' b = a' \diamond' b$, we have with probability at least $1 - 18\epsilon > 0$,

$$(\alpha_1 \diamond b) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2) = (\alpha_1' \diamond b) \circ' (\alpha_2 \diamond \beta_1) \circ' (\alpha_2 \diamond \beta_2),$$

which by the cancellativity of $\circ'$ is equivalent to

$$\alpha_1 \diamond b = \alpha_1' \diamond b.$$

Since $\diamond$ is also cancellative, we get $\alpha_1 = \alpha_1'$ from which $a = a'$.

Similarly, using Hypothesis (7), we can show $\diamond'$ is right cancellative as well. □

(iv) Finally, using Theorem 45, we test if $(G, \diamond')$ is a 5ϵ-abelian group.

Thus, we obtain that $(G, \circ, \diamond')$ is a $(5\epsilon, 5\epsilon)$-field of order $|G|$ where $\diamond'$ is 2ϵ-close to $\diamond$. □

4.3 Discarding the Cancellativity Assumption

In this section we give the additional tests required for testing associativity and associativity-commutativity when $\circ$ is not known to be cancellative.

Theorem 49 *There exists an $\epsilon_0 > 0$ such that for any $\epsilon < \epsilon_0$, and for any G and any $\circ, \diamond$, there are ϵ-spot-checkers that run in $\tilde{O}(|G|^{3/2})$ randomized time for spot-checking (i) if $(G, \circ)$ is a group, (ii) if $(G, \circ)$ is an abelian group, and (iii) if $(G, \circ, \diamond)$ is a field.*

The general intuition is that even if the table for $\circ$ is not cancellative, we can detect the situation where it does not contain a reasonably "even" distribution of the elements of G. To ensure that, we require the conditions below, for a small enough ϵ'. Note that since ϵ' contributes additional error, we need to modify the parameters used in the previous tests to allow smaller error.

We check the following conditions via random sampling:

(T4) $\forall a, |\{a \circ b \mid b \in G\}| \geq (1-\epsilon')|G|$.

(T5) $\Pr_\beta[|\{a \circ \beta \mid a \in G\}| = |G|] \geq 1-\epsilon'$.

(T6) $\Pr_\alpha[|\{\alpha \circ b \mid b \in G\}| = |G|] \geq 1-\epsilon'$.

Checking the first condition involves using the element distinctness algorithm of Section 2.3, which increases the running time to $\tilde{O}(|G|^{3/2})$.

Using the above conditions and the tests implied by them, and modifying the proofs to accommodate the bias in the distribution of elements due to the small probability of non-cancellative behavior, our spot-checker can be made to work even in the non-cancellative case. The main modification to the proofs involves the quantification of the following:(i) the error when given an arbitrary b, and a uniformly distributed β_1, we cannot find a β_2 such that $\beta_1 \circ \beta_2 = b$; (ii) the distributions of β and $a \circ \beta$ for fixed a and uniformly distributed β; (iii) whatever cancellativity we can infer from the additional conditions; and (iv) the error in probabilistic statements when the random variables are from distributions that are close to uniform.

We use the following observation and lemma:

Observation 50 *For an event $\mathcal{E}(x)$ and for an ϵ-uniform distribution D, $|\Pr_{x \in_D G}[\mathcal{E}(x)] - \Pr_x[\mathcal{E}(x)]| \leq \epsilon$.*

Lemma 51 *If $\circ$ satisfies conditions* (T4), (T5)*, and* (T6)*, then*

(1a) *$\forall a, \Pr_{\alpha_1}[\exists \alpha_2$ such that $\alpha_1 \circ \alpha_2 = a] \geq 1-\epsilon'$ and the distribution of α_2 is $2\epsilon'$-uniform.*

(1b) *$\forall a, \Pr_{\alpha_2}[\exists \alpha_1$ such that $\alpha_1 \circ \alpha_2 = a] \geq 1-\epsilon'$ and the distribution of α_1 is $2\epsilon'$-uniform.*

(2) *For all a, if α is from a distribution that is $2\epsilon'$-uniform, then the distribution of $a \circ \alpha$ is $4\epsilon'$-uniform.*

(3) *$\forall a, a'$, $\Pr_\beta[(a \circ \beta = a' \circ \beta) \Rightarrow (a = a')] \geq 1-\epsilon'$.*

Proof. For (1a), condition (T6) ensures that for $1-\epsilon'$ fraction of α_1's, there exist α_2's such that $\alpha_1 \circ \alpha_2 = a$. Since by condition (T4) the set of such α_2's is $\geq (1-\epsilon')|G|$, the distribution of α_2 is $2\epsilon'$-uniform. (1b) follows similarly.

For (2), note that condition (T4) ensures that for any $a \in G$ and $\alpha \in_R G$, the distribution of $a \circ \alpha$ is $2\epsilon'$-uniform. If α is from a $2\epsilon'$-uniform distribution, then the distribution of $a \circ \alpha$ is $4\epsilon'$-uniform.

(3) is obvious from condition (T5), where for a random β, it is checked if $\{a \circ \beta \mid a \in G\} = G$. □

We then define

$$a \circ' b = \operatorname*{maj}_{\beta \text{ such that } \exists \gamma \text{ for which } \beta \circ \gamma = b} \{(a \circ \beta) \circ \gamma\}.$$

for the associativity test, and we define

$$a \circ' b = \operatorname*{maj}_{\beta \text{ such that } \exists \gamma \text{ for which } \beta \circ \gamma = b} \{(a \circ \gamma) \circ \beta\}.$$

for the associativity-commutativity test.

The rest of the proofs of Theorems 27 and 35 can be mimicked using Observation 50 and Lemma 51, with some loss in efficiency that results in stricter requirements on ϵ. This does not affect the overall asymptotic efficiency of the test.

For purposes of illustration, we outline the details for mimicking Lemma 36.

Lemma 52 *Given b, if $\beta_2, \gamma_1 \in_R G$ and $\beta_1, \gamma_2, \delta$ are such that $\beta_1 \circ \beta_2 = b = \gamma_1 \circ \gamma_2$, and $\delta \circ \beta_2 = \gamma_2$, then* $\Pr[(\gamma_1 \circ \delta) = \beta_1] \geq 1 - \epsilon - 4\epsilon'$.

Proof. Note that by Lemma 51, β_1 and γ_2 each exist with probability at least $1 - \epsilon'$ and each are $2\epsilon'$-uniform. Also, δ exists with probability at least $1 - \epsilon'$ and is $2\epsilon'$-uniform. Then, $\beta_1 \circ \beta_2 = b = \gamma_1 \circ \gamma_2 = \gamma_1 \circ (\delta \circ \beta_2) = (\gamma_1 \circ \delta) \circ \beta_2$, with the last step true with probability $1 - \epsilon$ by (T3) in the associativity test (since γ_1 and β_2 are independent and random). Since β_2 is uniform, by Lemma 51 it can be cancelled with probability at least $1 - \epsilon'$, in which case we have $\beta_1 = \gamma_1 \circ \delta$. □

For the proof of Theorem 49(iii), we wish to use Lemma 48. However, the cancellativity of $\diamond$ is used in the last few lines of the proof. Note that, condition (T4) and an added condition (T4)'

(T4)' $\forall b, |\{a \circ b \mid a \in G\}| \geq (1 - \epsilon')|G|$.

are sufficient for proving Lemma 48 using a slightly smaller ϵ, but without assuming the cancellativity of $\diamond$..

4.4 Lower bounds on determining exact associativity and commutativity

4.4.1 Determing exact commutativity

By examining the $O(|G|^2)$ entries of $T_\circ$, a deterministic $O(|G|^2)$ procedure, by checking for symmetry, can determine if $\circ$ is commutative. If $\circ$ is not required to be cancellative, the lower bound of $\Omega(|G|^2)$ is immediate because an unexamined pair $a \circ b$ and $b \circ a$ could be made non-commutative.

If $\circ$ is deemed to be cancellative, the above argument fails since this simple-minded operation could render the Cayley table non-cancellative. First, we argue that any deterministic algorithm requires $\Omega(|G|^2)$ time. Suppose only $o(|G|^2)$ locations in the table are looked at. Then, let c be a constant such that $c||G|, c \geq 3$ (for sufficiently large $|G|$). We can view $T_\circ$ as being an $(|G|/c) \times (|G|/c)$ matrix of $c \times c$ "blocks". (These blocks can be indexed $(i, j), 1 \leq i, j \leq (|G|/c)$.) Then, note that there is a block (i, j) in the table that is not looked at by the algorithm (for otherwise, $\geq (|G|/c)^2 = \Omega(|G|^2)$ entries are looked at, which is a contradiction). The idea is to recursively construct an $|G| \times |G|$ latin square $T_\circ$ by first constructing a symmetric $|G|/c \times |G|/c$ latin square and then replacing each entry except (i, j) in $T_\circ$ by a $c \times c$ symmetric latin square. The entry (i, j) is replaced by an asymmetric $c \times c$ latin square.

More formally, let T^* be a symmetric $(|G|/c) \times (|G|/c)$ latin square. Let $k' = T^*(i, j)$, i.e., let k' be the (i, j)-th entry of T^*. Let T_k (resp., T'_k) be a $c \times c$ symmetric (resp., asymmetric) latin square where the entries are translated $t \mapsto t + ck$ (such latin squares exist for $c \geq 3$, for instance, non-abelian groups). This translation map is used to make the entries in $T_\circ$ distinct. Construct $T_\circ$ by first replacing the (i, j)-th entry by the block $T'_{k'}$ and then replacing all other entries with value k by the block T_k. It is straightforward from our construction to see that $\circ$ is cancellative but not symmetric.

We can extend this lower bound to randomized algorithms as in [RaS96] using Yao's minimax principle [Yao77] (which is the application of von Neumann's minimax theorem to show the equivalence of randomized and distributional complexities). Thus, the following theorem is immediate:

Theorem 53 *The deterministic and randomized complexity of deciding whether $\circ$ is commutative on G is* $\Theta(|G|^2)$.

4.4.2 Determing exact associativity and commutativity

Suppose a deterministic algorithm performs only $o(|G|^2)$ operations and determines that $T_\circ$ is both commutative and associative. We show that there is a $T_\circ$ on which it errs. Let $T_\circ$ correspond to $\mathbb{Z}_2^{|G|}$, an abelian group. We can view the elements in $\mathbb{Z}_2^{|G|}$ (listed in the canonical binary ordering) in consecutive blocks of two.

Consider two such blocks of elements $A = \{\alpha 0, \alpha 1\}, B = \{\beta 0, \beta 1\}$ for $\alpha, \beta \in \mathbb{Z}_2^{|G|-1}$. Let $A \circ B$ denote the 2×2 block of products $a \circ b$ where $a \in A, b \in B$. From the structure of $\mathbb{Z}_2^{|G|}$, we can see that both $A \circ B$ and $B \circ A$ in $T_\circ$ are isomorphic to a symmetric 2×2 latin square. If this pair of blocks is not looked at by an algorithm, then one can always change one of these latin squares to a different one (there are two distinct latin squares of size 2), thus still preserving cancellativity and thereby get a non-abelian $T_\circ$ that is unsuspectingly passed.

Since there are $|G|^2/4$ such disjoint 2×2 blocks in $T_\circ$, any algorithm for determing associativity and cancellativity must look at $\Omega(|G|^2/4)$ entries in $T_\circ$.

To argue against randomized algorithms, we use Yao's minimax principle as before. In conjunction with [RaS96], the following theorem then follows:

Theorem 54 *The deterministic and randomized complexity of simultaneously deciding whether $\circ$ is both associative and commutative on G is $\Theta(|G|^2)$.*

References

[Acz66] J. Aczel. *Lectures on Functional Equations and their Applications*. Academic Press, 1966.

[AHK95] L. M. Adleman, M-D. Huang, and K. Kompella. Efficient checkers for number-theoretic computations. *Information and Computation*, 121(1):93–102, 1995

[ABC+93] S. Ar, M. Blum, B. Codenotti, and P. Gemmell. Checking approximate computations over the reals. *Proc. 25th Symposium on Theory of Computing*, pp. 786–795, 1993.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems, *J. of the ACM*, 45(3):501–555, 1998.

[AS98] S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. *J. of the ACM*, 45(1):70–122, 1998.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols, *Computational Complexity*, pp. 3–40, 1991.

[BFLS90] L. Babai, L. Fortnow, C. Lund, and M. Szegedy. Checking computations in polylogarithmic time. *Proc. 31st Foundations of Computer Science*, pp. 16–25, 1990.

[BGR96] M. Bellare, J. Garay, T. Rabin. Batch verification with applications to cryptography and checking. *Proc. Latin American Theoretical Informatics 98*, Springer LNCS 1830:267–288, 1998. By the same authors Fast Batch Verification for modular exponentiation and digital signatures. Proceedings of Eurocrypt 98, Springer-Verlag LNCS, Editor K. Nyberg, 1998, to appear.

[BK95] M. Blum and S. Kannan. Designing programs that check their work. *J. of the ACM*, 42(1):269–291, 1995.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. of Computing and System Sciences*, 47(3):549–595, 1993.

[BW94a] M. Blum and H. Wasserman. Program result-checking: A theory of testing meets a test of theory. *Proc. 35th Foundations of Computer Science*, pp. 382–392, 1994.

[BW94b] M. Blum and H. Wasserman. Reflections on the Pentium division bug. *Proc. 8th Intl. Software Quality Week*, 1994.

[CR92] E. Castillo and M.R. Ruiz-Cobo. *Functional Equations and Modeling in Science and Engineering*. Marcel Dekker Inc., 1992.

[EKR96] F. Ergün, S. Ravi Kumar, and R. Rubinfeld. Approximate checking of polynomials and functional equations. *Proc. 37th Foundations of Computer Science*, pp. 592–601, 1996.

[EKR99] F. Ergün, S. Ravi Kumar, and R. Rubinfeld. Fast approximate PCPs. *Proc. 31st Symposium on Theory of Computing*, 1999. To appear.

[EKS99] F. Ergün, S. Ravi Kumar, and D. Sivakumar. Self-testing without the generator bottleneck. *SIAM J. on Computing*, to appear.

[FGL+96] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques, *J. of the ACM*, 43(2):268–292, 1996.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. of the ACM*, 45(4):653–750, 1998.

[GR97] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Proc. 29th Symposium on Theory of Computing*, pp. 406–415, 1997.

[KS96] S. Ravi Kumar and D. Sivakumar. Efficient self-testing/self-correction of linear recurrences. *Proc. 37th Foundations of Computer Science*, pp. 602–611, 1996.

[Lip91] R. Lipton. New directions in testing. *Proc. DIMACS Workshop on Distr. Comp. and Cryptography*, pp. 191–202, 1991.

[LZ78] R. Lipton and Y. Zalcstein. Probabilistic algorithms for group-theoretic problems. Manuscript. Abstract appeared in *ACM SIGSAM Bulletin*, 12:8–9, 1978.

[MNS+98] K. Mehlhorn, S. Naher, T. Schilz, S. Schirra, M. Seel, C. Uhrig. Checking geometric programs or verification of geometric structures. *Proc. 12th Annual Symposium on Computational Geometry*, pp. 159–165, 1996.

[RaS96] S. Rajagopalan and L. Schulman. Verifying identities. *Proc. 37th Foundations of Computer Science*, pp. 612–616, 1996.

[Rub94] R. Rubinfeld. Robust functional equations with applications to self-testing/correcting. *Proc. 35th Foundations of Computer Science*, pp. 288–299, 1994.

[RS96] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials and their applications to program testing. *SIAM J. on Computing*, 25(2):252–271, 1996.

[Yao77] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. *Proc. 18th Foundations of Computer Science*, pp. 222–227, 1977.