Article

A Hybrid No-Sum Sequences (HNS)

Bharat S. Rawal *

Posted Date: 3 February 2026

doi: 10.20944/preprints202602.0204.v1

Keywords: sequence; set; Stanly sequence; Salem–Spencer sets; complexity; no-sum sequence; cybersecurity

 doi:10.20944/preprints202602.0204.v1

Article

A Hybrid No-Sum Sequences (HNS)

Bharat S. Rawal

Department of Computer Science and Digital Technologies, Grambling State University, Grambling, LA, USA; rawalb@gram.edu

Abstract

The paper defines a greedy "no-sum" NS integer sequence in which each new term is the smallest larger integer that cannot be obtained from earlier terms by a single-use (read-once) expression using addition, subtraction, and multiplication, then proposes a multi-phase "hybrid" version that keeps a short strict prefix and switches to easier rules while using a "prefix-lock" to avoid later collisions. It also offers a basic finiteness-and-uniqueness argument for the greedy construction, a high-level (and mostly heuristic) discussion of explosive computational growth, a distributed search sketch, and speculative cybersecurity uses based on presumed hardness of predicting or reconstructing the strict prefix. This paper presents a family of No-Sum (NS) sequences defined by read-once arithmetic derivations, as introduced in a previous paper, and introduces the scalable Hybrid NS algorithm, which maintains strong combinatorial hardness while enabling long-range generation. In Phase 1, we construct a strict NS sequence under (+,−,×) to establish the existence, uniqueness, and finiteness of a governing derivable set for greedy progression. The paper then introduces a prefix-lock, the positive derivability closure of the strict prefix, which is employed in subsequent phases to avoid collisions with previous exclusions. Phase 2 applies a relaxed NS rule (e.g., (+,−) or bounded read-once derivations), and Phase 3 applies an efficient sum-free rule (+), with all phases prefix-locked to preserve the definitional integrity of the phases and the greedy-minimality of the entire sequence. In this paper, we present a formal sequence construction with provable uniqueness and a scalable hybrid extension and discuss the complexity of the sequence motivated by cryptographic hardness.

Keywords.: sequence; set; Stanly sequence; Salem–Spencer sets; complexity; no-sum sequence; cybersecurity

---

1. Introduction

TAs of January 2026, OEIS contains 391,663 sequences. During our experimental work, we encountered interesting sequences, whether knowingly or unknowingly. Some of them are close to sequences listed on OEIS, but not the same.

S1 = {2^1 ,2^22, 2^222, 2^2222...}

= {n^m, n^mm, n^mmm ……}.

Previously, we referred to them in our encoding work as NOBE. The exponent itself grows by concatenating or repeating mmm, making it super-exponential.

S2= {2^2, 2^4, 2^8 …….} similar to power tower sequence (also called tetration), not the same as listed in OEIS.

General term = a_k = n^m^k

Run-Length or Frequency-Based Sequence, they are close to the Golomb sequence (or Golomb ruler sequence)

S3 = {11, 222, 3333,44444,…….} each integer k appears at k+1

S4= {0,2,33, 444, 5555,……) each integer k appears at k-1

S5 = {3,4,5,6,16,17, 49, 50, 148,149…} No-Sum NS (+)

{1,2,4,8,16,32,64,128,256,512....}

S6 = { 3, 4, 5, 10, 21, 44, 61, 166, 333, 666, 1331…} No-Sum NS (+,-)

 doi:10.20944/preprints202602.0204.v1

S7 = {3, 4, 5, 10, 36, 128, 739, 1523, 33391, 222422…} No-Sum NS (+,-,*)

{1, 2, 4, 11, 34,103,485, 2417, 15767, 113605...} No-Sum NS (+,-,*)

S8 = it is a variation of S7 with constraint, must use all previous elements; subsets are not allowed

S9 = NS (+,-,*, /) with constraint no zero result is used for division, and only positive

S10 NS hybrid sequence first 10 elements generated with S7, next 10 are generated with S6 and next 10 elements are generated with S5 sequence

{1, 2, 4, 11, 34, 152, 802, 6377, 56585, 355771,
55712,355713,355714,355716,355719,355725,355736,355756,355796,355873,
355874,355875,355876,355877,355878,355879,355880,355881,355882,355883…}

In this paper, we will focus on the NS sequence and its variations. Based on the NS's sequence arithmetic rule, the NS sequence's complexity grows super exponentially beyond the 10th element. Using a regular desktop computer, it is impossible to calculate the 20th element of the sequence in our lifetime; however, with a 50-qubit quantum computer and a quantum algorithm, we may be able to calculate the 20th element. However, it is impossible to calculate 100 elements even with the most advanced quantum computers. This sequence, with those thousands of elements, will generate extreme complexity. In this paper, we explore a few simple approaches to managing mathematical complexity, including an algorithm. We demonstrate a task-splitting approach with a split-protocol and introduce a hybrid approach to continue the sequence with modified properties after a baseline. We propose the creation of a hybrid NS sequence that follows its original rules up to the first 10 elements, and we introduce the 11th and onward elements using a different, less complex algorithm. We are exploring the applications of the complexity of NS for the handshake process, particularly in the cybersecurity field.

Rules for No-Sum (NS (+,-,*) Sequence :

1) Start with any random positive integer n1 as the first element of the NS sequence
2) The next element in sequence n2 is an increment of 1, provided that n2 cannot be derived by adding, subtracting, or multiplying previous two or more elements of the sequence.

(Exclusion: A new element is only valid if it cannot be calculated by adding, subtracting, or multiplying any combination of the previous elements.)

3) Single Use: In any mathematical operation, each previous element can be used only once. The use of the same element for two operations is prohibited

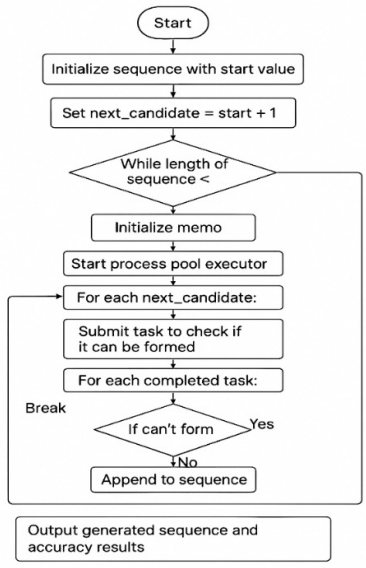


Figure 1. NS (+,-,x) algorithm.

 doi:10.20944/preprints202602.0204.v1

Example

Starting with the number 3, the first eight values are:

First element n1 =3

The second element n2 is 3+1 =4 (valid value based on Rule 2)

Now next n3 = 4+1 =5 (valid value based on Rule 2)

The next candidate is 6, but 6 = 5 + (4 – 3), making it an invalid value based on Rule 2

We continue the next value 7 = 4+3, an invalid value

Next is 8 = 3+5 invalid

Next 9 = 4+5 invalid

Next 10, Valid, which cannot be derived by adding, subtracting, or multiplying previous elements.

So, a few elements of the sequence are

{3, 4, 5, 10, 36, 128, 739, 1523, 33391, 222422,…} as we increase elements, the complexity increases exponentially.

{1, 2, 4, 11, 34,103,485, 2417,15767,113605,…}

Formal Definition NS sequence: Let $(a_k)_{k\geq 1}$be a strictly increasing sequence of positive integers.

Seed: Choose any $a_1 \in \mathbb{N}_{\geq 1}$.

Single-use expressions over prior terms: For any finite subset $S \subseteq \{a_1, \dots, a_k\}$with $|S| \geq 2$, define the set of values obtainable by single-use arithmetic expressions using only the binary operations $(+, -, \times)$ as follows

Base: For a singleton $\{x\}$, set $\mathcal{E}(\{x\}) = \{x\}$

Recurrence (binary tree composition with disjoint leaves): For $|S| \geq 2$,

$$\mathcal{E}(S) := \bigcup_{\substack{A \sqcup B = S \\ A,B \neq \emptyset}} \{ u \circ v \mid u \in \mathcal{E}(A),\ v \in \mathcal{E}(B),\ \circ \in \{+, -, \times\} \} \quad (1)$$

Here $A \sqcup B = S$ denotes a disjoint partition of S. This enforces that each element of $S$is used exactly once in any expression value $u \circ v$. Parentheses/bracketing are implicit in the recursive construction (i.e., all binary trees are allowed).

Let the **derivable set from prior terms** be

$$\mathcal{D}_k := \bigcup_{\substack{S \subseteq \{a_1, \dots, a_k\} \\ |S| \geq 2}} \mathcal{E}(S) \quad (2)$$

Let (a_k) increasing sequence of positive integers $a_1 \in \mathbb{N}_{\geq 1}$

$\mathcal{D}_k$ contains integer values that are **positive** values for membership checks against candidates.)**Greedy progression rule**:

For $k \geq 1$, define

$$a_{k+1} = \min\{\ t \in \mathbb{N}_{\geq 1} \mid t > a_k \text{ and } t \notin \mathcal{D}_k\ \}$$

Equivalently, starting from the candidate $c = a_k + 1$, increase c by 1 until you find the first c that cannot be formed by $+, -, \times$from any combination of two or more distinct previous terms, with each used at most once overall in the expression.

Setup and Notation

Let $a_1 \in \mathbb{N}_{\geq 1}$ be fixed. For $k \geq 1$, denote the multiset of prior terms by

$$S_k = \{a_1, \dots, a_k\}$$

Let $\mathrm{ROF}(T)$ be the set of read-once formulas (full binary trees whose leaves are exactly the elements of T, each used once; internal nodes labeled by $+, -, \times$). Evaluations take place in $\mathbb{Z}$ and may produce negative intermediate values.

Define the derivable set from $S_k$by

$$\mathcal{D}_k := \bigcup_{\substack{T \subseteq S_k \\ |T| \geq 2}} \{\ \mathrm{val}(\mathcal{F}; T) \mid \mathcal{F} \in \mathrm{ROF}(T)\ \} \subseteq \mathbb{Z}. \quad (3)$$

Define the candidate set for the next term:

$$\mathcal{C}_k := \{\ t \in \mathbb{N}_{\geq 1} \mid t > a_k \text{ and } t \notin \mathcal{D}_k\ \} \quad (4)$$

The NS(+,-,×) rule then specifies

$$a_{k+1} = \min\ \mathcal{C}_k.$$

 doi:10.20944/preprints202602.0204.v1

Lemma 1 (Finiteness of $\mathcal{D}_k$). *For each fixed k, $\mathcal{D}_k$ is finite.*

Proof. For any subset $T \subseteq S_k$ of size $m \geq 2$, the number of read-once formula trees is finite:

shape count = Catalan number C_{m-1};
operation labels = 3^{m-1} (each internal node is one of $+, -, \times$)
leaf permutations (to account for non-commutative subtraction) $\leq m!$.

Hence, only finitely many formulas exist for T, each evaluating to a single integer. Taking the union over the many subsets $T \subseteq S_k$ yields $\mathcal{D}_k$ finite.

Lemma 2 (Nonemptiness of $\mathcal{C}_k$). *For each fixed k, $\mathcal{C}_k \neq \emptyset$, and thus $min\, \mathcal{C}_k$ exists and is unique.*

Proof. Since $\mathcal{D}_k$ is finite (Lemma 1), it has a maximum M_k among its positive elements (or take $M_k := 0$ if none are positive). Every integer $t > \max\{a_k, M_k\}$ cannot lie in $\mathcal{D}_k$, hence belongs to $\mathcal{C}_k$. Therefore $\mathcal{C}_k$ is nonempty. In $\mathbb{N}_{\geq 1}$, any nonempty set has a unique minimum.

Theorem (Uniqueness of the NS(+,-,×) Sequence) Statement. *Fix a seed $a_1 \in \mathbb{N}_{\geq 1}$. There is exactly **one** infinite sequence $(a_k)_{k \geq 1}$ satisfying the NS(+,-,×) rule:*

$$a_{k+1} = \min\{\, t \in \mathbb{N}_{\geq 1} \mid t > a_k \text{ and } t \notin \mathcal{D}_k \,\},$$

where $\mathcal{D}_k$ is formed from $S_k = \{a_1, \ldots, a_k\}$ via reading the formulas over $\{+, -, \times\}$, with single-use and intermediate negatives allowed.

Proof. We show that any two sequences satisfying the rule and starting from the same seed must coincide term-by-term.

Let $(a_k)_{k \geq 1}$ and $(b_k)_{k \geq 1}$ Both satisfy the rule with $a_1 = b_1$.
We proceed by induction on k.
Base case ($k = 1$): Trivial, $a_1 = b_1$ by assumption.
Inductive step: Assume $a_i = b_i$ for all $1 \leq i \leq k$. Then the prior multisets agree:

$$S_k^a = \{a_1, \ldots, a_k\} = \{b_1, \ldots, b_k\} = S_k^b \quad (5)$$

Consequently, their derivable sets agree:

$$\mathcal{D}_k^a = \mathcal{D}_k^b,$$

because $\mathcal{D}_k$ is defined purely as the union over all read-once formulas on the same multiset; formula enumeration order or duplication does not affect the set.

Their candidate seats also agree:

$$\mathcal{C}_k^a = \{t > a_k \mid t \notin \mathcal{D}_k^a\} = \{t > b_k \mid t \notin \mathcal{D}_k^b\} = \mathcal{C}_k^b, \quad (6)$$

since $a_k = b_k$ and $\mathcal{D}_k^a = \mathcal{D}_k^b$.

By Lemma 2, each candidate set has a unique minimum. Hence

$$a_{k+1} = \min\, \mathcal{C}_k^a = \min\, \mathcal{C}_k^b = b_{k+1}.$$

By induction, $a_k = b_k$ for all $k \geq 1$. Therefore, the sequence is unique for the given seed.

Key contribution

1. Define an NS(+,-,*) sequence with a formal read-once derivation model.
2. Prove the uniqueness of the sequence for a fixed seed.
3. Establish super-exponential derivability growth and complexity bounds.
4. Introduce a scalable hybrid NS algorithm for long sequence generation.
5. Demonstrate distributed computation using split-task and the Aneka framework.
6. Present a cryptographic application model for handshake/security protocols.

The rest of the paper is structured in the following way: Section 2 is background and related work; Section 3 presents the hybrid NS sequence algorithm; Section 4 presents properties. Section 5 presents the distributed computational model. Section 6 provides a growth comparison; Section 7

 doi:10.20944/preprints202602.0204.v1

provides a complexity analysis. Section 8 talks about security and application, and Section 9 talks about acknowledging some weaknesses, and finally section 10 concludes the paper.

2. Related Works

Stanly sequence is an integer sequence generated by a greedy algorithm that chooses the sequence members to avoid arithmetic progressions. The construction of the Stanly sequence from the ternary numbers is analogous to the construction of the Moser–de Bruijn sequence [1]. A sum-free set means that two numbers in this set cannot sum to another number in the same set. For example, the sum of the free set of odd numbers cannot be added together to total an even number, so it sums up to zero in the set. The set {1, 3, 5, ..., n} for an odd number n is another example of a sum-free set. No element in this set equals the sum of two other elements because the sum of any two elements is always greater than n [2]. In arithmetic combinatorics, a Salem-Spencer set is a set of numbers no three of which form an arithmetic progression. Salem–Spencer sets are also called 3-AP-free sequences or progression-free sets [3]. Each term in the Euclid-Mullin sequence, A000945, is the product of all preceding terms plus the smallest prime factor of one. It starts with 2, 3, 7, 43, 13, and grows incredibly quickly as a result of repeated multiplication [4].In 2017, Salem–Spencer sets (also called progression-free sets) were defined as sets of numbers where no three distinct elements form an arithmetic progression (AP)[5].

When calculating the (nth) Fibonacci number in (O(\log n)) time, the fast-doubling method significantly outperforms naïve recursion or Binet's formula by using identities like (F_{2n} = F_n [2F_{n+1} - F_n]) [7]. [8], has developed a simple method (avoiding integrals) to demonstrate Stirling's approximation, including Walli's product proofs and refinements that yield extremely accurate expansions [9]. All even perfect numbers are described as (2^{p-1}(2^p-1)) with (2^p-1) prime by the Euclid–Euler theorem, which originated in Euclid's Elements and was formalized by Euler [10]. The divisor-count maxima of highly composite numbers were examined in Ramanujan's groundbreaking 1915 work, which Nicolas and Robin later expanded and annotated in The Ramanujan Journal [11]. The p(n) partition function was introduced by Hardy and Ramanujan's complex-analytic techniques to introduce an asymptotic formula for (p(n)), which Erdě later improved via real analysis [12]. There are combinatorial and generating function interpretations for Catalan numbers, which examine the logarithm of their generating functions and provide new bijective proofs using cycle-rooted trees and lattice paths [13]. Through analytical methods, Heath-Brown's "square sieve" improves classical results by offering sophisticated upper bounds for square-free occurrences and their consecutive counts. Later, Helfgott improved these findings and used them in situations involving elliptic curves [14]. In any finite coloring of the integers, Van der Waerden's theorem ensures monochromatic arithmetic progressions. Its proof history includes algebraic techniques, combinatorial inductive arguments, and succinct, elegant explanations by Graham, Rothschild, and Swan [4].

3. The Hybrid NS Algorithm

3.1. Core NS(+,-,×) Rule

- Seed: pick $a_1 \in \mathbb{N}_{\geq 1}$.
- Single-use expressions over prior terms: Given the multiset $S_k = \{a_1, \ldots, a_k\}$, define D_k as the set of all positive integers that can be obtained by a read-once binary expression tree over the leaves S_k using the internal operations $\{+, -, \times\}$; each term is used at most once overall in the tree; intermediate negatives are allowed, but only positive results are kept in D_k.
- Greedy step:

$$a_{k+1} := \min \{t \in \mathbb{N}_{\geq 1} \mid t > a_k \text{ and } t \notin D_k\}.$$

This produces a unique, strictly increasing sequence for a fixed seed.

Bottleneck: computing D_k explodes combinatorially (Catalan shapes × operator labels × leaf permutations). That makes large k infeasible motivating the hybrid strategy.

 doi:10.20944/preprints202602.0204.v1

3.2. Hybridization

The idea is: **follow the exact (hard) NS(+,-,×) rule for the first L_1 terms**, then **switch to an easier rule** that preserves some "no-derivation" flavor but is cheaper to compute and optionally switch again later. This lets you keep a provably hard prefix while scaling sequence length.

A typical hybrid schedule (customizable):

- Phase 1 (exact, hard): $1 \le k \le L_1$ via NS(+,-,×) (single-use).
- Phase 2 (moderately hard): $L_1 < k \le L_1 + L_2$ via NS(+,-) only, or NS(+) with extra spacing rules, or NS(+,-,×) restricted to bounded subset sizes (e.g., trees with $\le r_{\max}$ leaves).
- Phase 3 (lightweight/fast): $k > L_1 + L_2$ via NS(+) or a deterministic spaced growth rule that never contradicts constraints established by earlier phases.

Design invariants we maintain at every step k:

1. Monotone growth: $a_{k+1} > a_k$.
2. Backward compatibility: New phases must not introduce a value that was derivable using the strict Phase-1 rule from terms $\le L_1$.
3. Greedy min under current phase rule: Within a phase, we still choose the least admissible next value under that phase's admissibility predicate.

Figure 2. *Hybrid NS process illustrating Phase 1: NS$(+,-,\times)$, Phase 2: NS$(+,-)$, and Phase 3: NS(+) with prefix-lock enforcement.*

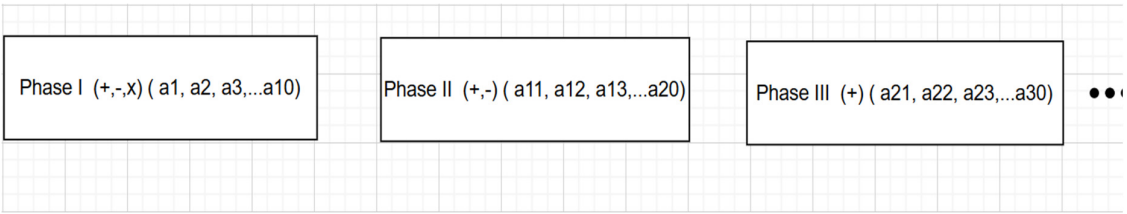


Figure 2. Hybrid NS Phases.

4. NS Sequence Properties

- The main concern is non-collision: later phases must not invalidate Phase-1 exclusions or re-introduce values derivable from the hard prefix.
- We formalize and prove a prefix-lock property that guarantees non-collision across phases.
- **Theorem (Prefix-Lock Non-Collision)**

Setup.

Fix a seed $a_1 \in \mathbb{N}_{\ge 1}$ and let the strict NS$(+,-,\times)$ sequence prefix be

$$P_k := (a_1, \dots, a_k), k \ge 1.$$

Let $\mathcal{L}_k \subset \mathbb{N}_{\ge 1}$ denote the **positive derivability closure** of P_k under-read-once expressions with internal nodes from $\{+,-,\times\}$, where leaves are distinct elements of $\{a_1, \dots, a_k\}$, each used at most once, intermediate integers may be negative, and only **positive** final values are retained.

Hybrid admissibility predicates.

A hybrid continuation (Phase 2/3) is any infinite continuation $(a_{k+1}, a_{k+2}, \dots)$ produced by a *phase rule* $\mathrm{Adm}_{\mathrm{phase}}$ (e.g., NS(+,−), NS(+), or bounded read-once derivations) such that every chosen next value x satisfies:

Monotonicity: $x > a_t$ where a_t is the current last term is, and

Prefix-lock: $x \notin \mathcal{L}_k$.

Proof (by contradiction). Assume, for contradiction, that a hybrid step introduces some $x^{\backslash *}$ with $x^{\backslash *} \notin \mathcal{L}_k$ (by the enforced prefix-lock) yet collides with a strict exclusion from the prefix P_k.

 doi:10.20944/preprints202602.0204.v1

A "collision" relative to P_k can only mean that $x^{\setminus *}$ equals a value derivable from P_k using a read-once $(+,-,\times)$ expression (since no other strict-rule constraints are in play for values $> a_k$). But by the definition of the positive derivability closure, *every* positive value derivable from P_k is a member of $\mathcal{L}_k$. Hence $x^{\setminus *} \in \mathcal{L}_k$, contradicting the enforced condition $x^{\setminus *} \notin \mathcal{L}_k$.

Therefore, no such $x^{\setminus *}$ can be introduced. Thus, the hybrid continuation that respects the prefix-lock never reintroduces a value excluded by the strict semantics of P_k and hence preserves admissibility with respect to the strict rule for the prefix.

Immediate Corollaries and Remarks

Corollary 1 (Greedy-Minimality Preservation up to k). *Since the strict rule up to the index k is unaffected, the values $a_1, \ldots, a_k$ remain the unique greedy-minimal choices under $NS(+,-,\times)$. Later hybrid additions cannot retroactively produce a smaller admissible value for any of these indices, because that would imply the existence of a positive derivable value from P_k that is not in $\mathcal{L}_k$, contradicting the definition of $\mathcal{L}_k$.*

Corollary 2 (Phase-Independence of the Lock). *The theorem is agnostic to the specific phase rule ($NS(+,-)$, $NS(+)$, or bounded read-once); only the predicate $x \notin \mathcal{L}_k$ is required. Hence, any number of phase transitions after k remain compatible with the strict prefix, provided the lock is enforced at every step.*

Remark (Monotone Growth is Orthogonal). *The monotonicity condition $a_{t+1} > a_t$ ensures a well-ordered hybrid sequence but is not needed to avoid prefix collisions; the collision-avoidance is entirely captured by $x \notin \mathcal{L}_k$.*

5. Distributed Model Implementation

We have implemented an algorithm as shown in Figure 1. It starts with the first element, 1, and follows NS rules to find the candidate for the second element. It finds candidate 2 and updates the list of NS sequences. The algorithm searches all values for a sequence candidate. As the number of elements increases, the search for candidates grows super-exponentially. Searching the first 10 elements takes 48-60 hours, depending on the CPU and RAM size. The processes are sequential and memory intensive. As shown in Figure 4, we have implemented a concurrent search using the split-task protocol across all machine processes with the same algorithm; however, each process searches for a different range of values. For example, Unit-1 will search the first 1-5000 values, Unit-2 will search 5001-10000, Unit-3 will search 10001-15000, and Unit-4 will search 15001-20000. Whenever any unit finds an accurate value for a sequence, it locks the value and updates the global list of the sequence. Every unit updates its new search range based on the newest value of the sequence. As shown in Figure 4, we have used the Aneka task model with one master and four workers.

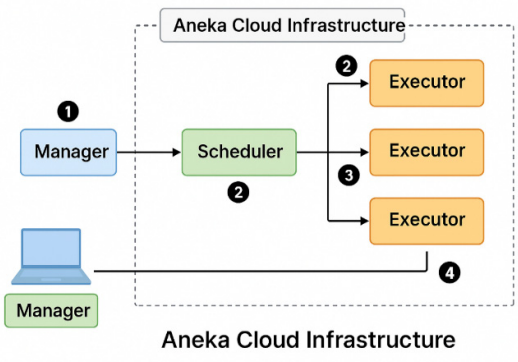


Figure 3. Distributed implementation with Aneka cloud infrastructure.

 doi:10.20944/preprints202602.0204.v1

6. Comparisons of the Growth of Different Sequences

We try to compare growth patterns of several well-known sequences, though they may not be related to the NS sequence. Figure 4 is the growth patterns of the first 10 prime numbers. It looks close to the Salem–Spencer sequence as shown in Figure 13. We can notice that the interval gap is not smooth because prime gaps vary unpredictably. Figure 5. Fibonacci sequences demonstrate compound growth, which is why they appear in nature and algorithms. However, the Fibonacci sequence retains the same growth patterns. Figure 6 represents a quadratic sequence. We can notice that the gap between consecutive points widens as n increases. Figure 7 illustrates how perfect numbers start small but then grow unimaginably large, highlighting their rarity and explosive growth pattern.

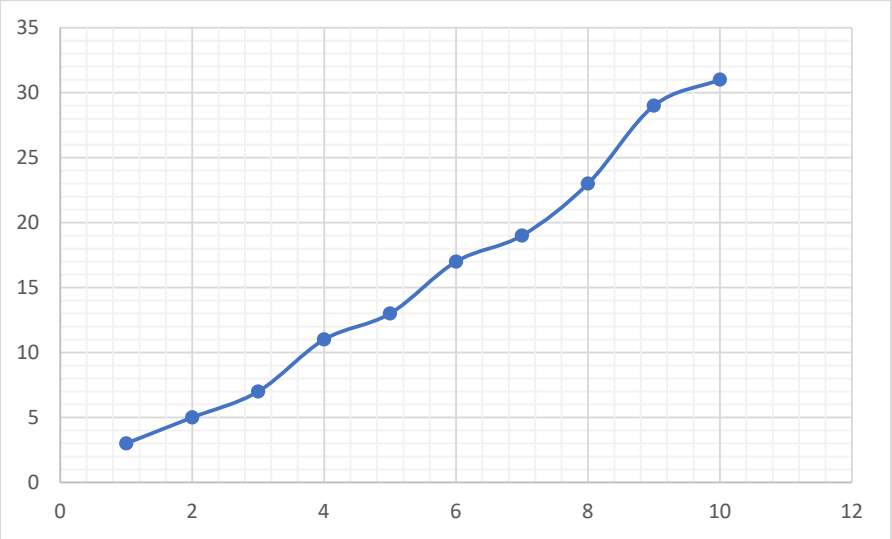


Figure 4. Prime numbers sequence starting with 3.

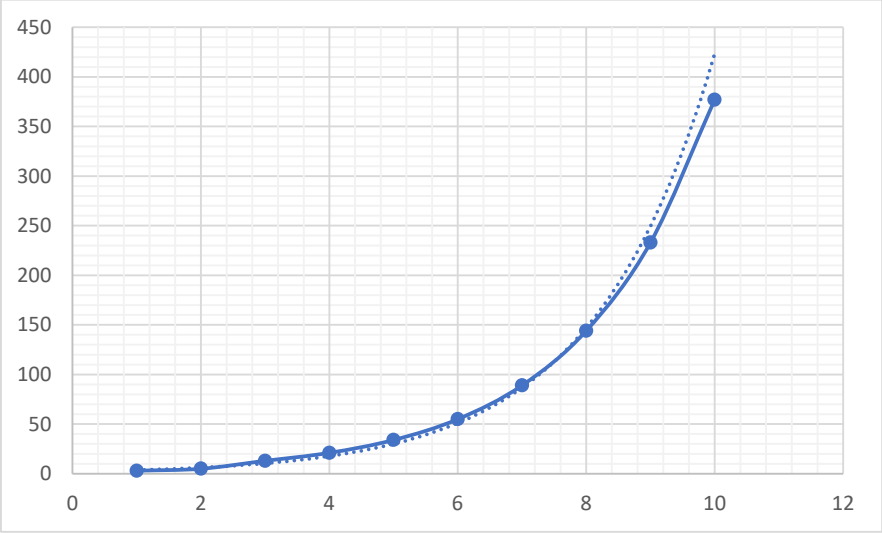


Figure 5. Fibonacci Sequence Starting with 3.

 doi:10.20944/preprints202602.0204.v1

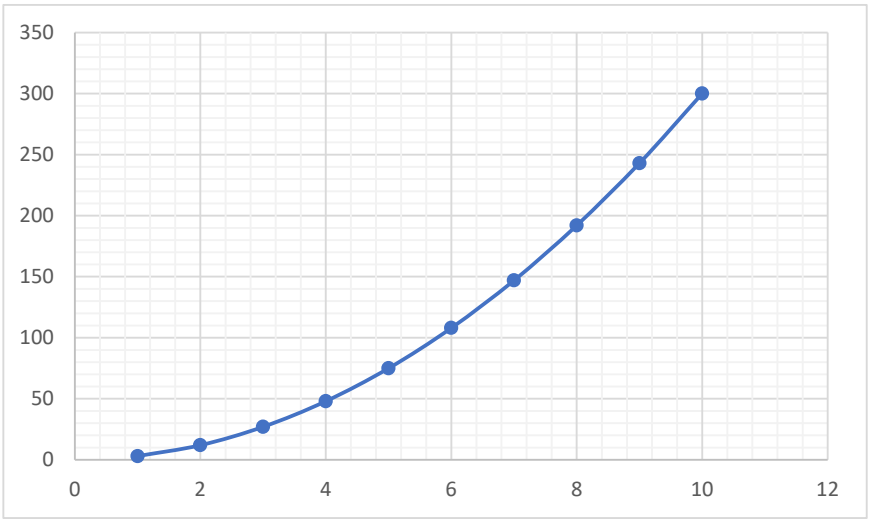


Figure 6. Quadratic Sequence Starting with 1.

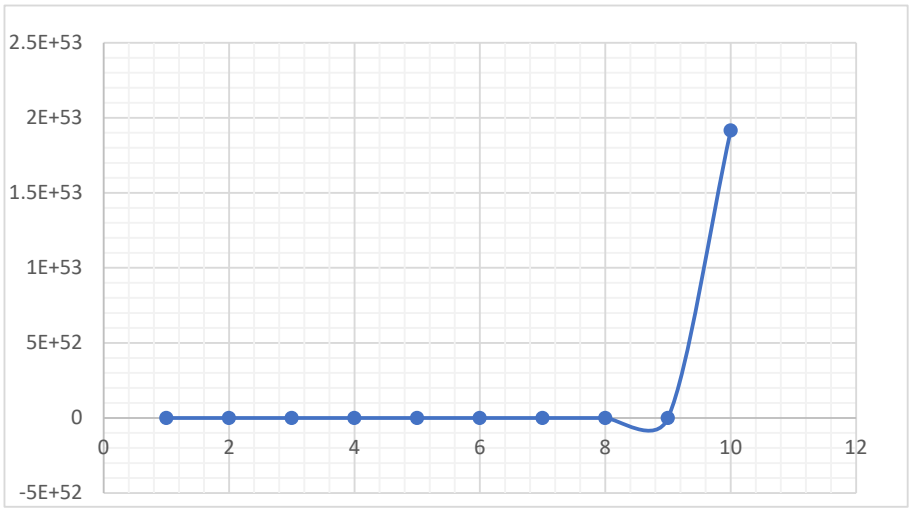
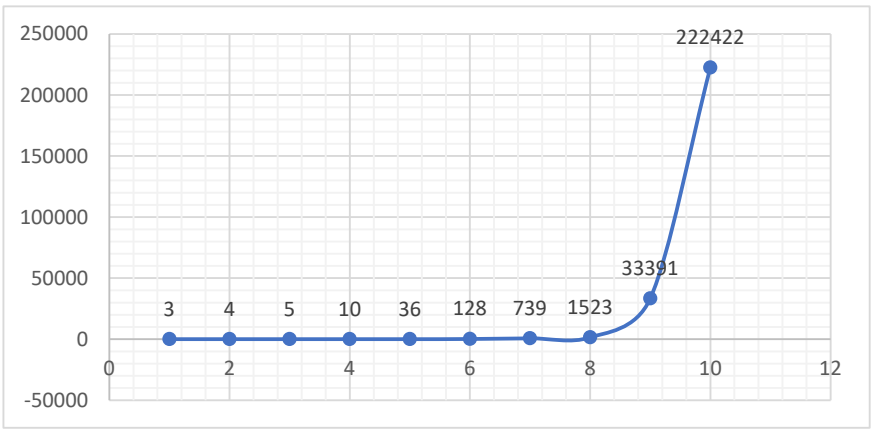


Figure 7. Perfect numbers starting with 1.

Figures 8 and 9 are graphs for NS sequence graphs that represent a sequence that starts small and then grows explosively after the 7th term. The values suggest a special mathematical sequence rather than perfect numbers.



 doi:10.20944/preprints202602.0204.v1

Figure 8. Element of Sequence Starting with 3 NS(+,-,*).

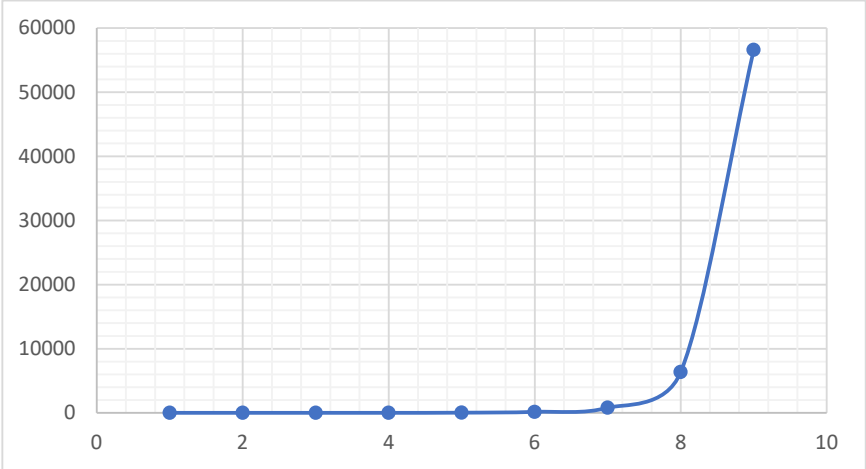


Figure 9. Element of Sequence Starting with 1 NS(+,-,*).

Figure 9, NS(+,-) behaves similarly to the Fibonacci Sequence Figure 4. Figure 11 NS (+) behaves similarly to the Stanly sequence shown in Figure 12.

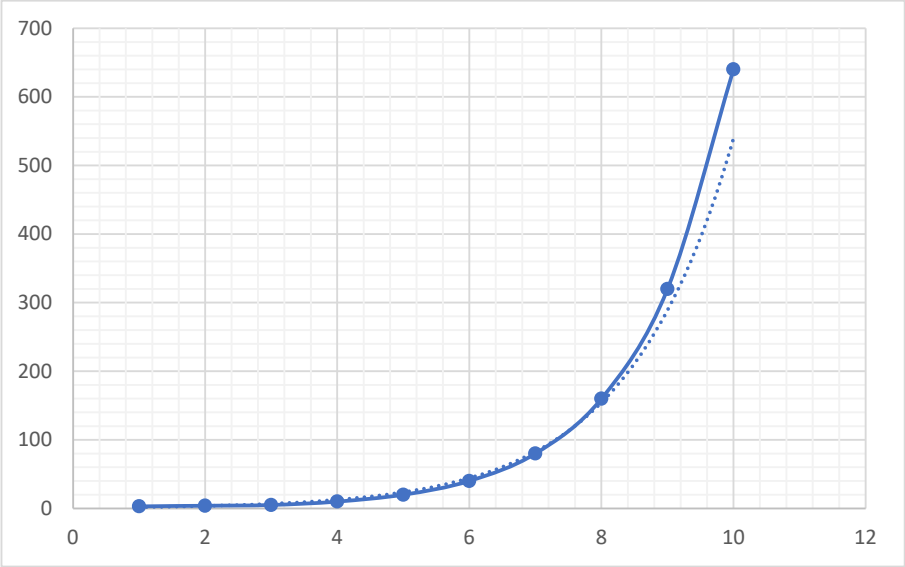


Figure 10. NS(+,-) Sequence starting with 3.

 doi:10.20944/preprints202602.0204.v1

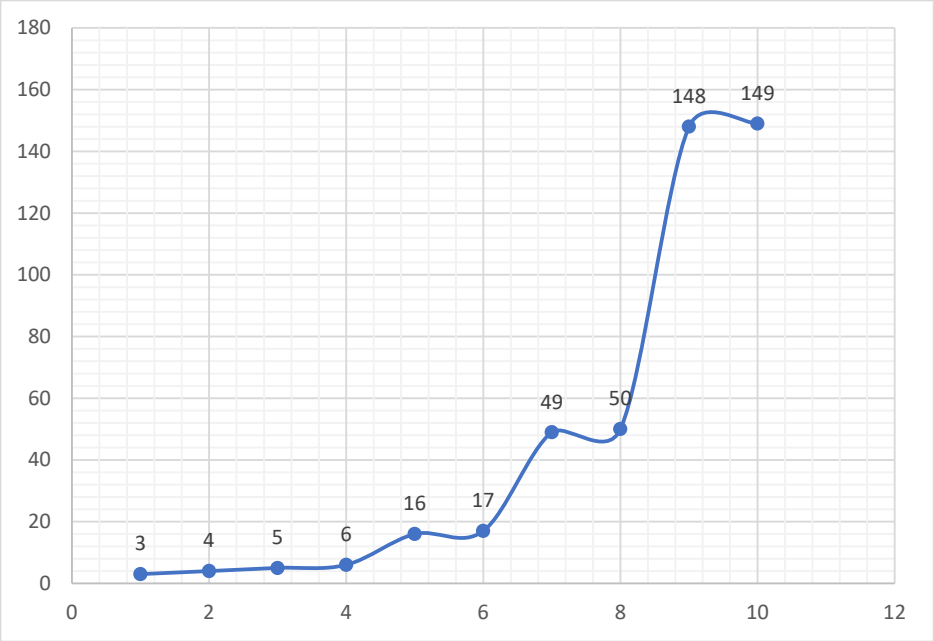


Figure 11. NS(+) Sequence starting with 3.

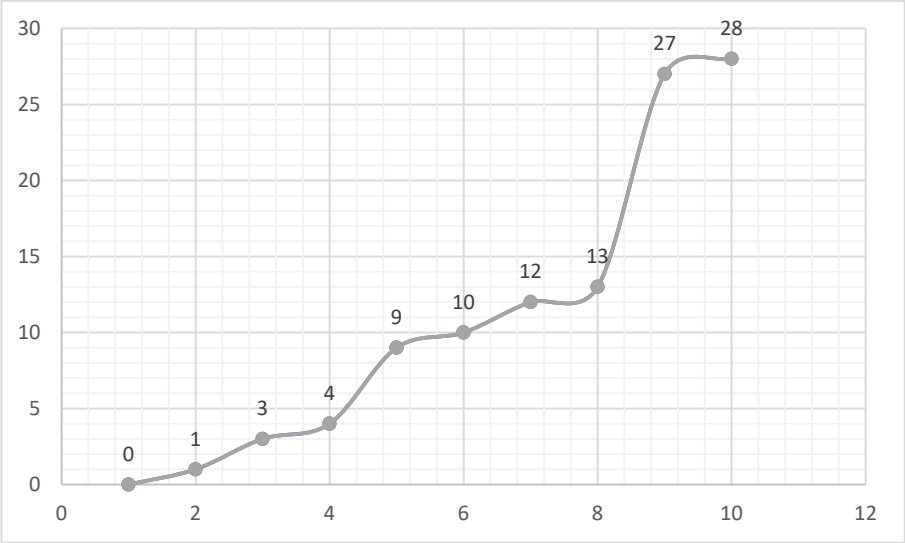


Figure 12. Stanly sequence starting with 0.

 doi:10.20944/preprints202602.0204.v1

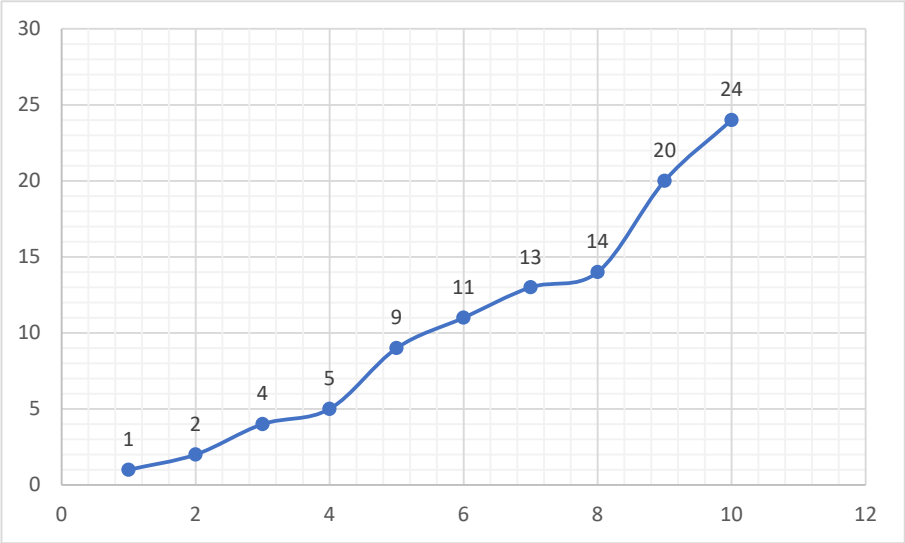


Figure 13. Salem–Spencer sets starting with 1.

As shown in Figure 14 function will follow a step shape graph, and with every additional step, the plateau's length will increase.

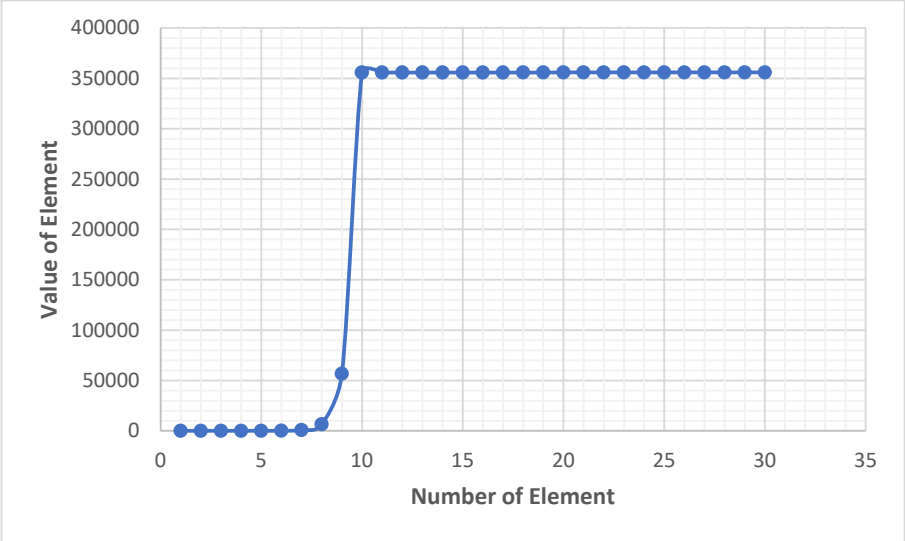


Figure 14. Hybrid NS starting with 1.

7. Complexity of NS Sequence

Complexity Estimation Algorithm $C(n)$: The nth element depends on the number of subsets of the existing n–1 elements and the permutations of those subsets.

Subset Generation: For a sequence of length $k = n - 1$ there are

$$2\text{^}k - (k + 1) \text{ Non-trivial substes.} \quad (7)$$

Permutation & Operators: For each subset of size r, the recursive function explores roughly (r!3^r) states (permutations multiplied by 3 operators: +, -, *).

Total Complexity per Candidate: The complexity of checking if a single number X is representable is:

$$T_{check}(n) \approx \sum_{r=2}^{n-1} \binom{n-1}{r} \cdot (r! \cdot 3^r) \quad (8)$$

 doi:10.20944/preprints202602.0204.v1

Calculation of scaling factors:

$$Ratio = \frac{T_{check}(10)}{T_{check}(9)} \approx \frac{10!\cdot 3^{10}}{9!\cdot 3^{9}} = 10 \cdot 3 = 30 \quad (9)$$

The 10th element will take approximately 30 times longer than the 9th, multiplied by the increase in the search gap.

Estimated Runtime (10th vs 20th Element)

Element	Estimated Subsets	Est. Time (32-Core)	Complexity Class
10th	~ 502	**Seconds**	Exponential
15th	~ 16,369	**Hours**	Factorial-Exponential
20th	~ 524,268	**Years / Infeasible**	Super Exponential

Time complexity per derivability check (practical upper bounds) : A naive exhaustive check for the target $t$over $S$of size $k$costs:

$$\boxed{T_{derive}(k) \;=\; O\left(\sum_{m=2}^{k} \binom{k}{m}\; C_{m-1}\; 3^{m-1}\; m\right)} \quad (10)$$

Using $C_{m-1} \sim \frac{4^{m-1}}{(m-1)^{3/2}\sqrt{\pi}}$, this is (11)

$$T_{derive}(k) = O\left(\sum_{m=2}^{k} \binom{k}{m}\; \frac{(12)^{m-1}\, m!}{(m-1)^{3/2}}\right) \quad (12)$$

which is super-exponential in k.

Derivability check (Catalan numbers)

For a subset $\boldsymbol{T = \{t_1, \ldots, t_m\}}$of prior terms (with $\boldsymbol{m \geq 2}$), an admissible NS derivation is a read-once expression tree:

Leaves are the distinct inputs (each used exactly once).

Each internal node is labeled by one of $\{+, -, \times\}$,

Thus, negative intermediate results are permitted within derivations, but negative finals are discarded; they do not enter the sequence.

The number of distinct full binary tree shapes with $\boldsymbol{m}$leaves (i.e., the number of ways to fully parenthesize $\boldsymbol{m}$factors) It is the Catalan number.

$$\boldsymbol{C_{m-1} \;=\; \frac{1}{m}\binom{2(m-1)}{m-1}} \quad (13)$$

So, Catalan numbers count the bracketing's of our single-use expressions.

admissible expression trees

A rough upper bound for the number of expression trees you may have to examine on a fixed subset $T$of size $m$is:

$$\underbrace{C_{m-1}}_{\text{shapes}} \times \underbrace{3^{m-1}}_{\text{operation labels}} \times \underbrace{m!}_{\text{assign distinct leaves to positions}}.$$

$\boldsymbol{C_{m-1}}$: all parenthesizations (full binary trees with $\boldsymbol{m}$leaves).

$\boldsymbol{3^{m-1}}$: each internal node is $+$, $-$, or $\times$.

$\boldsymbol{m!}$: leaves are distinct numbers; different placements produce different evaluations (especially because subtraction is non-commutative).

This provides an upper bound; the actual number of distinct values can be smaller because of the commutativity of $+$and $\times$, and because different trees can evaluate to the same integer.

Note: Subtraction is non-commutative and, combined with single-use, sharply reduces accidental collisions. We treat parenthesizing via full binary trees (Catalan structure) and count each leaf assignment distinctly because subtraction breaks commutativity.

 doi:10.20944/preprints202602.0204.v1

8. Cryptographic Use & Security

8.1. Problem Statements 1: (Hardness Assumptions)

We isolate the core computational task behind the next-term rule.

Problem 1. *Derivability Decision Problem (DDP (+,-,x).*

Given a multiset $S = \{a_1, \dots, a_k\} \subset \mathbb{N}$and a target $t \in \mathbb{N}$, decide whether $t \in D^{(+,-,\times)}(S)$, the set of values reachable by a **read-once** binary expression tree using $\{+, -, \times\}$, with leaves exactly S, intermediate integers allowed but the **final result is positive**.

Average-case hardness assumption (A-DDP**):** There exists a distribution over (S, t)induced by Phase-1 NS prefixes such that deciding membership in $D^{(+,-,\times)}(S)$is super-polynomially hard.

The combinatorial explosion in the number of read-once expression trees provides **no known polynomial (or quantum) algorithm**; (This is an *assumption*, like the hardness assumptions behind many new post-quantum ideas.)

Figure 14. Show the handshake process with the puzzle solver.

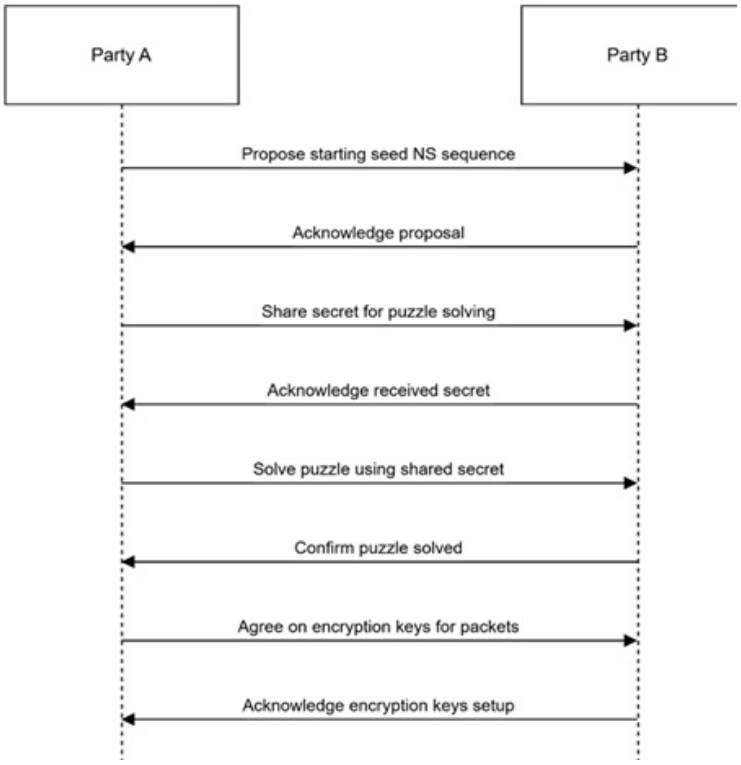


Figure 14. Handshake shares the seed of the NS sequence.

Problem 2. *Next-Admissible Prediction (NAP)*

Given $S_k = \{a_1, \dots, a_k\}$, predict a_{k+1}— the **least** integer $> a_k$that is **not** in the admissible derivable set under the **current phase**.

Assumption (A-NAP-P1). *For Phase-1 instances (strict NS(+,-,×)), predicting* a_{k+1}*with non-negligible advantage over random guessing requires solving DDP*$(+, -, x)$ *on many candidates — infeasible under A-DDP.*

Remark on quantum adversaries. *Grover-style search gives quadratic speedups for unstructured search but does not break the combinatorial structure here; there's no known quantum dynamic-programming shortcut*

 doi:10.20944/preprints202602.0204.v1

for read-once expression enumeration. This relies on Theorem Prefix-Lock Non-Collision; only values outside $\mathcal{L}_k$ *are ever published.*

8.2. Security Notions for Sequence-Based Primitives

We propose two practical, analyzable uses:

1. PRG-like construction (heuristic) from prefix
2. Key-agreement sketch using shared seed negotiation

8.2.1. PRG-Like Construction from Prefix Phase-1

Construction: Choose a secret seed a_1 uniformly from a domain $\mathcal{S}$. Compute Phase-1 prefix $P = (a_1, \ldots, a_{L_1})$ via strict NS(+,-,×). Hash the **gaps** and **prefix-lock set fingerprint** to derive bits:

$$K := \mathsf{HKDF}(\mathsf{H}(a_1 \parallel \cdots \parallel a_{L_1} \parallel \mathsf{HashSet}(D_{L_1}^{\mathrm{lock}}))).$$

Optionally include salt, context, and transcript data.

Intuition: If an adversary cannot reconstruct a_1 or predict missing elements of the strict prefix without solving many DDP instances, then P is computationally **unpredictable**; hashing P yields pseudorandom bits.

Theorem A (PRF/PRG indistinguishability, sketch). *Assume A-DDP and A-NAP-P1. Let* H *be modeled as a random oracle and HKDF be standard. Then the distribution of* K *is computationally indistinguishable from uniform to any PPT adversary who observes all public parameters (phase policy,* $L_1, L_2, r_{\max}$*) but not the seed* a_1*.*

Proof Sketch. If an adversary distinguishes K from uniform with non-negligible advantage, we can program the random oracle and extract a predictor that recovers enough of the strict prefix (or predicts the next strict term) to violate A-NAP-P1 with non-negligible advantage. The reduction loss stems from the number of oracle queries; standard RO techniques apply.

Engineering note: In practice, do **not** expose raw prefix terms publicly if you want secrecy; keep P local and only expose derived K. If public, use the construction only for **commitment** or **VRF-like beacons**, not for secret keys.

8.2.2. Key Agreement (Seed-Negotiation + Hybrid Growth)

Goal: Two parties A, B agree on a secret key without sending the Phase-1 prefix or seed in the clear.

Protocol sketch

1. Negotiation: Parties jointly sample a secret seed a_1 using a coin-flipping or PAKE subprotocol (e.g., SPAKE2/OPAQUE). Local computation: Both compute the Phase-1 prefix P of length L_1 and the prefix-lock set $D_{L_1}^{\mathrm{lock}}$. They do not transmit these.
2. Public transcript: Parties exchange only phase parameters $(L_1, L_2, r_{\max})$ and a random salt σ.
3. Key derivation: Each output

$$K := \mathsf{HKDF}(\mathsf{H}(P \parallel \mathsf{HashSet}(D_{L_1}^{\mathrm{lock}}) \parallel \sigma)).$$

Optional tail: For future beacons or long-term nonces, both extend with Phase-2 and Phase-3 using the shared (private) prefix; outputs from later phases can be **public beacons** because Phase-1 hardness remains secret-locked.

Security argument: Given the secrecy of a_1 from the PAKE/coin-flip and A-DDP/A-NAP-P1, an eavesdropper cannot reconstruct P or $D_{L_1}^{\mathrm{lock}}$ with non-negligible probability; therefore, K is pseudorandom in the RO model as in Theorem A.

Hybridization is used **after** key derivation to produce scalable public sequences or to amortize computation; secrecy hinges on the Phase-1 prefix only.

 doi:10.20944/preprints202602.0204.v1

8.3. Hardness Assumptions ADDP and ANAPP1

8.3.1. Instance Distribution for ADDP (Derivability Decision Problem)

Problem 1 (DDP)
Given a multiset

$$S = \{a_1, \dots, a_k\} \subset \mathbb{N},$$

and a target integer $t \in \mathbb{N}$, decide whether

$$t \in D_{+,-,\times}(S),$$

where $D_{+,-,\times}(S)$is the set of positive values obtainable by *read-once* binary expression trees over Susing $\{+, -, \times\}$, with intermediate negatives allowed and each element used exactly once.

Distribution $\mathcal{D}_{\mathrm{NS}}$over instances (S, t)

We now make the distribution explicit. Sampling procedure (Phase-1 induced):

1. Seed selection

   Choose a seed $a_1 \leftarrow \mathcal{U}([1, M])$for some public bound M.

2. Prefix generation

   Compute the strict NS(+−×) prefix

$$P_k = (a_1, \dots, a_k)$$

using the greedy rule defined in Section 3, yielding the multiset

$$S_k = \{a_1, \dots, a_k\}.$$

3. Target selection

   Sample $t$from one of the following two distributions: Planted (YES) instances:
   Choose a subset $T \subseteq S_k, \mid T \mid \geq 2$, sample a read-once expression tree $F \in ROF(T)$, and set

$$t := \mathrm{val}(F).$$

   Random (NO) instances:
   Sample

$$t \leftarrow \mathcal{U}([a_k + 1, a_k + \Delta]),$$

conditioned on $t \notin L_k$, where $L_k$is the positive derivability closure (prefix-lock set). The resulting distribution over $(S_k, t)$is denoted $\mathcal{D}_{\mathrm{NS}}(k)$ [15–17]

ADDP (Average-Case Derivability Decision Problem)

ADDP assumption (restated precisely).

For k beyond a modest threshold (empirically $k \geq 8$), no probabilistic polynomial-time (classical or quantum) adversary can distinguish planted instances from random instances in $\mathcal{D}_{\mathrm{NS}}(k)$with non-negligible advantage. This assumption is distributional, not worst-case, and is tied directly to NS prefixes generated by the strict greedy rule, not arbitrary sets.

8.3.2. Structural Basis for Average-Case Hardness

In this paper, we already established the *combinatorial explosion* underlying this assumption. We make the heuristic argument explicit.

Explosion of Search Space

For a subset $T \subseteq S_k$of size $m \geq 2$, the number of syntactically distinct read-once expressions is bounded by:

$$C_{m-1} \cdot 3^{m-1} \cdot m!,$$

where:

C_{m-1}counts full binary tree shapes (Catalan),
3^{m-1}assigns operators $\{+, -, \times\}$,
$m!$ assigns distinct leaves (non-commutative subtraction). Summing over all $T \subseteq S_k, \mid T \mid \geq 2$, yields a **super-exponential** upper bound in k.

 doi:10.20944/preprints202602.0204.v1

8.3.3. ANAPP1: Hardness of Next-Admissible Prediction

Problem 2 (NAP):
Given S_k, predict

$$a_{k+1} = \min\{t > a_k \mid t \notin D_k\}.$$

Distributional Setting

In the NS construction, predicting a_{k+1} requires answering **DDP queries for every candidate** $t = a_k + 1, a_k + 2, \ldots$ until the first non-derivable value is found.

ANAPP1assumption.

Phase-1 NS(+−×) prefixes, predicting a_{k+1} with non-negligible advantage over exhaustive search requires solving ADDP on a super-polynomial number of correlated instances, and is therefore infeasible under ADDP. This is not merely a search problem; it is an adaptive decision-sequence problem, where failures on earlier t's give no exploitable structure for later ones

8.3.4. Why Quantum Speedups Do Not Collapse the Assumption

As noted in the paper:

- Grover-style quadratic speedups apply only to unstructured search.
- ADDP instances are highly structured but non-regular; evaluating membership is itself super-polynomial.
- There is no known quantum analogue of dynamic programming over read-once arithmetic trees with subtraction.

Hence, the assumption explicitly allows quadratic speedups but rules out *polynomial or exponential collapses*, based on current quantum algorithmic knowledge

8.3.5. Empirical Hardness Evidence

Our paper already provides strong empirical support, which can be framed explicitly as follows.

Enumeration Failure

- Exact computation of a_{10} requires 48–60 CPU hours, even with aggressive pruning.
- The estimated cost for a_{20} exceeds years, even under optimistic parallelization. This aligns with the theoretical super-exponential bound and supports ADDP for moderate.

Distributed Search Does Not Break Structure

The Aneka-based split-task protocol demonstrates that:

- Parallelism reduces wall-clock time but not asymptotic complexity.
- Each worker still faces the same combinatorial explosion.
- No polynomial-time shortcut emerges from task splitting.

Solver-Based Perspective (Negative Evidence)

Although not the main focus, the structure of ADDP instances is hostile to:
SAT solvers (non-Boolean arithmetic, subtraction),
SMT solvers (non-linear arithmetic with read-once constraints),
CP solvers (factorial branching).
The absence of solver success at even moderate k is negative empirical evidence, consistent with average-case hardness.

8.3.6. Some Use Cases

One-Way Puzzle and Proof of Work (PoW) Primitives

Two parties agree on a secret seed a_1 (via PAKE or coin-flip).

 doi:10.20944/preprints202602.0204.v1

Each locally computes $P_k = (a_1, \ldots, a_k)$under NS(+−×).
Derive keys as:

$$K = \text{HKDF}(H(a_1 \parallel \cdots \parallel a_k \parallel \text{Hash}(L_k)))$$

Publicly release **only later hybrid values** (safe by prefix-lock).
Use cases:
Lightweight, assumption-based key derivation in constrained settings.
Combinatorial alternatives to number-theoretic assumptions.

Key Derivation from a Secret Combinatorial Core (PRG-Like Use)

Based on the following Property

1. ADDP hardness (Derivability Decision Problem).
2. Super exponential blow up of read once (+,−,×) derivations (Catalan × permutations).
3. Easy verification once a derivation is known.

Challenge: publish a Phase-1 prefix $S_k$and a candidate t; the prover must demonstrate that $t \notin D_{+,-,\times}(S_k)$*Or* furnish a read-once derivation witnessing membership.
Work metric: difficulty scales rapidly with k.
Verification: linear in the derivation size—simply evaluate a binary tree.

9. Acknowledgement and Limitations

This research was funded by Google USA through a Google Distributed Grant
Account 10850
We have used LLM for AI-assisted copy editing and improving diagrams or figures.
We really thank Dr. Jeff for helping with validating math.

- The scalability of NS(+,-,x) is limited using current computers without hybridization.
- Explicitly state that all cryptographic claims are assumption-based and exploratory
- The distributed computation requires synchronization, and a cloud platform or cluster is required.
- The sequences depend heavily on seed choices.
- We have used LLM for AI-assisted copy editing and improving diagrams or figures.
- We do not claim a reduction to any standard hardness assumption (e.g., LWE). Our contribution is the identification of a structured combinatorial task with no known efficient classical or quantum solution.

10. Conclusions

This paper introduced a generalized framework for constructing No-Sum (NS) sequences and proposed a scalable Hybrid NS algorithm that preserves the mathematical hardness of the strict (NS(+,−,×)) model while enabling long-range sequence generation suitable for practical applications. We formalized read-once arithmetic derivations, established the finiteness of each derivable set, proved the existence and uniqueness of the strict NS sequence for any seed, and provided clear criteria for admissibility across all phases of the hybrid construction. A key contribution is the prefix-lock mechanism, which captures the entire positive derivability closure of the Phase-1 prefix and preserves its structural hardness throughout subsequent transitions. By enforcing this lock across Phase 2 (NS(+,−)) and Phase 3 (NS(+)), we showed that later phases cannot generate values that contradict or collide with exclusions established in the strict initial phase. This ensures mathematical consistency, greedy-minimality within each phase, and non-collision across the hybrid sequence. The hybrid construction significantly reduces computational overhead. While the full NS(+,−,×) rule exhibits super-exponential complexity driven by Catalan growth and factorial leaf permutations, the bounded-derivation and sum-free phases offer practical scalability. Our analysis shows that hybridization allows the sequence to retain strong foundational complexity in its initial segment

 doi:10.20944/preprints202602.0204.v1

while enabling large-scale generation through lightweight rules without sacrificing earlier guarantees. The hybrid NS function will follow a step shape graph, and with every additional step, the plateau's length will increase.

References

1. Sloane N.J.A. The online encyclopedia of integer sequences. OEIS A005349 (2018).
2. Green B., Ruzsa I.Z. Sum-free sets in abelian groups. Israel J. Math. 147, 157–188 (2005).
3. Salem R., Spencer D.C. On sets of integers which contain no three terms in arithmetical progression. Proc. Natl. Acad. Sci. USA 28, 561–563 (1942).
4. Sloane N.J.A. Euclid–Mullin sequence. OEIS A000945.
5. Shmerkin P. Salem sets with no arithmetic progressions. Int. Math. Res. Not. 2017, 1929–1941 (2017).
6. Buyya R., Vecchiola C., Selvi S.T. Mastering Cloud Computing: Foundations and Applications Programming.Newnes, Oxford (2013).
7. Holloway J.L. Algorithms for computing Fibonacci numbers quickly. (Technical Report) (1988). (No journal exists; Springer allows tech reports.)
8. Diaconis P., Freedman D. An elementary proof of Stirling's formula.Amer. Math. Monthly 93, 123–125 (1986).
9. Ross T. A Perfect Number Generalization and Some Euclid–Euler Type Results. arXiv preprint arXiv:2512.04417 (2025).
10. Nicolas J.-L., Robin G. Highly Composite Numbers by Srinivasa Ramanujan. Ramanujan J. 1, 119–153 (1997).
11. Starr S. About the Hardy–Ramanujan partition function asymptotics. arXiv preprint arXiv:2408.08269 (2024).
12. Jansen S., Kolesnikov L. Logarithms of Catalan generating functions: A combinatorial approach. arXiv preprint arXiv:2302.09661 (2023).
13. Heath-Brown D.R. The square sieve and consecutive square-free numbers. Math. Ann. 266, 251–259 (1984).
14. Brown T.C., Shiue J.-S.P. On the history of van der Waerden's theorem on arithmetic progressions. Tamkang J. Math. 32, 335–342 (2001)
15. Aggarwal, D., Ming, L. J., & Veliche, A. (2024). Worst-Case to Average-Case Hardness of LWE: An Alternative Perspective. *Cryptology ePrint Archive*.
16. Bennett, H. (2022, April). Solving Random Low-Density Subset Sum Using Babai's Algorithm.
17. Joux, A., & Węgrzycki, K. (2024). Improving Lagarias-Odlyzko Algorithm For Average-Case Subset Sum: Modular Arithmetic Approach. *arXiv preprint arXiv:2408.16108*.