An Optimal Algorithm for the Detection of Recursive Bilateral Symmetry in Strings

Abstract

This report introduces and formalizes the concept of Recursive Bilateral Symmetry (RBS) in strings, a hierarchical structure analogous to fractal self-similarity. We define RBS as a nested arrangement of palindromic structures, generalizing both standard and gapped palindromes. We present a novel algorithm for detecting all substrings exhibiting RBS within a given binary string S of length N. The proposed algorithm leverages foundational techniques in stringology, including advanced data structures like suffix trees and Longest Common Extension (LCE) queries, combined with a dynamic programming approach. We provide a rigorous analysis of the algorithm's correctness and demonstrate its time complexity. The report situates this new problem and its solution within the context of existing research on palindromic and periodic structures in strings, offering a new dimension for combinatorial pattern matching.

Part I: Problem Formulation and Theoretical Foundations

Section 1: Introduction

1.1 The Ubiquity of Symmetry and Hierarchy

Symmetry and hierarchy are organizing principles that pervade the natural and mathematical sciences. In physics, the Standard Model of particle physics is built

upon the foundation of gauge symmetries, and the spontaneous breaking of these symmetries gives rise to the fundamental forces and mass of elementary particles.[1] In more speculative frameworks like string theory, which models elementary particles as vibrational modes of one-dimensional strings, symmetries play a crucial role in ensuring the theory's consistency and in classifying its different versions.[2] The concept of a hierarchy problem, which questions the vast difference in energy scales between fundamental forces like gravity and the electroweak force, has motivated theories such as supersymmetry, which posits a fundamental symmetry between bosons and fermions.[3]

This theme of hierarchical symmetry breaking is also observed in cosmology. Models involving cosmic strings—topological defects formed in the early universe—often feature intricate sequences of symmetry breaking, such as $SU(2)\times U(1)\rightarrow U(1)\times U(1)\rightarrow U(1)'\rightarrow$Nothing.[4] Such sequences create a hierarchy of structures, with different properties emerging at different scales.

While these concepts from physics provide a rich inspirational backdrop, the focus of this report is to translate the intuitive notion of a hierarchical, self-similar symmetry into a formal, computational problem within the domain of stringology—the study of algorithms on strings.[6] The goal is to abstract the essence of a multi-scale, symmetric structure and develop an efficient algorithm for its detection in the discrete, one-dimensional context of a binary string.

1.2 Problem Statement: Defining Recursive Bilateral Symmetry (RBS)

To develop an algorithm, a precise, formal definition of the target structure is required. The term "recursive bilateral symmetry" suggests a structure that is symmetric about its center and contains smaller instances of a similar structure within it. We build this definition from fundamental concepts in combinatorics on words.

The most basic form of bilateral symmetry in a one-dimensional string is palindromicity. A string w is a palindrome if it reads the same forwards and backwards, i.e., $w=w^R$, where w^R is the reversal of w.[8] This property can be defined recursively: a string is a palindrome if it is the empty string (

λ), a single character, or has the form xwx, where x is a character and w is a palindrome.[10]

A more complex structure is the gapped palindrome, which has the form uvuR, where u is a non-empty string and v is an arbitrary "gap" string.[12] This structure exhibits bilateral symmetry in its "arms" (

u and uR) around a central, potentially non-symmetric component (v). This represents a first-order hierarchy: a symmetric frame around an inner component.

To introduce the "recursive" aspect, we draw an analogy from the study of fractals. Fractals are mathematical sets characterized by self-similarity across different scales; they are generated by the repeated application of a simple rule in an ongoing feedback loop.[14] Formal grammars known as Lindenmayer systems (L-systems) provide a powerful mechanism for generating such fractal-like strings. For example, an L-system rule like

(A→ABA) can generate the Cantor set, a classic fractal, through recursive substitution.[16] The recursive nature of L-systems leads directly to the self-similar structures found in fractals like the Koch snowflake and fractal trees.[16]

Synthesizing these concepts—the symmetry of palindromes, the hierarchy of gapped palindromes, and the self-similarity of fractals—we propose the following formal, inductive definition for Recursive Bilateral Symmetry (RBS):

- A string S has **RBS of order 0** if it is a non-empty standard palindrome.
- A string S has **RBS of order k>0** if it can be decomposed into the form uvuR, where u is a non-empty binary string and the central gap string v has RBS of order k–1.

This definition establishes a clear hierarchy. An RBS-1 string is a gapped palindrome whose gap is a standard palindrome. An RBS-2 string is a gapped palindrome whose gap is an RBS-1 string, and so on. This creates a nested, fractal-like structure of palindromic frames.

1.3 Scope of the Algorithmic Task and Contributions

The central algorithmic problem addressed in this report is as follows:

Given a binary string S of length N, for every substring S[i..j], determine its maximum RBS order.

A substring that does not satisfy the RBS property for any order will be considered to have an order of -1. The main contributions of this report are:

1. A novel, formal definition of Recursive Bilateral Symmetry (RBS) that bridges concepts from stringology and fractal geometry.
2. A survey of the foundational algorithms and data structures from stringology that are prerequisite to solving this problem.
3. The design and presentation of a new, optimized algorithm for detecting all substrings with RBS and determining their maximal order.
4. A rigorous proof of the algorithm's correctness and a detailed analysis of its computational complexity.
5. A discussion of the problem's relationship to existing work and promising directions for future research.

Section 2: Foundational Concepts and Algorithms in Stringology

The development of an efficient algorithm for RBS detection requires a solid foundation in modern string processing techniques. This section reviews the essential prerequisite concepts and algorithms.

2.1 Palindromic Substrings

The detection of palindromes is a classic problem in computer science. A naive approach might check every substring for palindromicity, leading to an O(N3) algorithm. A more refined brute-force method, known as "expand from center," considers each of the 2N−1 possible centers (each character and each space between characters) and expands outwards, checking for symmetry. This improves the complexity to O(N2).[18]

The state-of-the-art for this problem is Manacher's algorithm, which finds all maximal palindromic substrings in optimal O(N) time.[18] The algorithm's ingenuity lies in two key ideas. First, it transforms the input string (e.g.,

aba becomes ^#a#b#a#$) to handle both odd- and even-length palindromes uniformly; in the transformed string, every maximal palindrome has an odd length and

a unique center. Second, it maintains the center and rightmost boundary of the palindrome found so far. When calculating the palindromic radius for a new center, it uses the information from its "mirror" position relative to the rightmost-reaching palindrome to avoid redundant character comparisons. This allows it to achieve linear time complexity. Manacher's algorithm is not merely background; it serves as a fundamental building block for the RBS detection algorithm proposed later, providing an efficient way to identify all base-level (RBS-0) structures. The concept of a palindrome's center, whether a single character or the space between two, is also critical for organizing the search for higher-order structures.[19]

2.2 Gapped Palindromes and Repetitive Structures

The RBS structure uvuR is a specific type of gapped palindrome. Research into algorithms for finding and counting gapped palindromes provides crucial context for the expected complexity of our problem. Popa and Popa [13] have presented several efficient algorithms for

counting gapped palindromes under different constraints:

- For counting all occurrences of uvuR in a string of length N without any constraints on u or v, an O(N) time algorithm exists. This is achieved using a suffix tree on the string S#SR.
- If the length of the gap v is constrained to an interval, i.e., $g \le |v| \le G$, the complexity increases to O(NlogN). This algorithm requires more sophisticated data structures, such as AVL trees, to manage the length constraints during the traversal of the suffix tree.
- For a more general case with position-dependent gap constraints, the complexity further rises to O(Nlog2N).

The relationship between constraints and complexity is informative. The RBS definition imposes a *structural* constraint on the gap v—namely, that v must itself be an RBS structure of a lower order. This is a far more intricate constraint than a simple length restriction. By analogy to the work on gapped palindromes, it is reasonable to anticipate that solving the RBS detection problem will require a complexity greater than the unconstrained O(N) case. This context helps in evaluating the "optimality" of the proposed solution; an algorithm with polynomial complexity, such as O(N2), would represent an efficient solution for such a constrained problem, especially if the size of

the output can be shown to be quadratic in the worst case. This class of problems is also related to other studies of repetitive structures in strings, such as borders (a proper prefix that is also a suffix), periods, and covers.[20]

2.3 Essential Data Structures for Advanced String Processing

To move beyond naive comparisons and achieve an optimized solution, advanced data structures are indispensable. The core operation in identifying the arms u and uR of an RBS structure is to check if a prefix of one substring is the reverse of a suffix of another. This can be rephrased as a Longest Common Extension (LCE) query.

An LCE query, also known as a Longest Common Prefix (LCP) query between two suffixes, asks for the length of the longest common prefix of two suffixes of a string, S[i..N] and S[j..N]. With appropriate preprocessing, these queries can be answered in O(1) time. A standard method involves constructing a suffix tree or a suffix array for the string.

A suffix tree is a compressed trie of all suffixes of a string, which can be built in O(N) time using algorithms like Ukkonen's.[13] Once built, it can be augmented with a data structure for answering Lowest Common Ancestor (LCA) queries, which in turn allows for

O(1) LCE queries. A more space-efficient alternative is the suffix array, which is a sorted array of all suffixes, combined with an LCP array. This combination, along with a Range Minimum Query (RMQ) data structure, also supports O(1) LCE queries after O(N) or O(NlogN) preprocessing.[21]

For the RBS problem, we need to compare a substring of S with a substring of its reverse, SR. By constructing a generalized suffix tree or suffix array for the concatenated string T=S#SR (where # is a unique delimiter not in the alphabet), we can answer LCE queries between any substring of S and any substring of SR in O(1) time. This capability is the key to accelerating the search for the symmetric arms u and uR, forming the cornerstone of the optimized algorithm presented in Part II.

Part II: The RBS Detection Algorithm

Section 3: A Formal Framework for Recursive Bilateral Symmetry

3.1 The RBS Hierarchy

The inductive definition of RBS gives rise to a rich hierarchy of nested symmetric structures. To build intuition, consider the following examples in the binary alphabet {0,1}:

- **RBS Order 0:** Any standard non-empty palindrome.
  - 101101 is a palindrome, so it has RBS order 0.
  - 0000 is a palindrome, so it has RBS order 0.
- **RBS Order 1:** A string of the form uvuR where v is an RBS-0 string (a palindrome).
  - Consider the string 10(111)01. Here, u=10 and v=111. Since v is a palindrome (RBS-0), the full string 1011101 has RBS order 1.
  - Consider 11(101101)11. Here, u=11 and v=101101. Since v is a palindrome (RBS-0), the full string has RBS order 1.
- **RBS Order 2:** A string of the form uvuR where v is an RBS-1 string.
  - Consider 01(11(101)11)10. Here, u=01. The inner part is v=11(101)11. As shown above, this v has RBS order 1. Therefore, the full string 011110111110 has RBS order 2.

A given string may admit multiple decompositions into the uvuR form. The **maximal RBS order** of a string is defined as the maximum order achievable over all possible valid decompositions. For instance, the string 11111 is a palindrome (RBS-0). It can also be decomposed as u=1, v=111, where v is also a palindrome (RBS-0). This decomposition gives it an RBS order of 1+0=1. The maximal RBS order is therefore 1. The algorithm must be designed to find this maximum order for every substring.

3.2 Connections to Formal Languages and Generative Systems

The structure of RBS can be elegantly described using the formalism of language theory. The detection of an RBS string is equivalent to a parsing problem for a specific type of grammar. An L-system, as noted earlier, is a parallel rewriting system used to *generate* fractal strings.[16] For example, the rule

A⟶ABA starting from the axiom A generates strings that correspond to the construction of the Cantor set.

Our RBS detection algorithm can be viewed as the inverse of this process: a recognition or parsing system. The recursive definition of RBS corresponds to a set of production rules in a formal grammar, specifically a context-sensitive grammar.

- RBS_0 -> P (where P is the set of all palindromes)
- RBS_k -> u RBS_{k-1} u^R (for any non-empty string u)

An RBS string can thus be represented by a parse tree. Each internal node in the tree corresponds to a symmetric frame u...uR, and its child is the parse tree of the inner component v. The depth of this parse tree corresponds to the RBS order of the string. This perspective highlights a powerful duality: while L-systems build complex strings from simple symbols, an RBS detection algorithm deconstructs a string to reveal its underlying hierarchical, symbolic structure. This connection elevates the problem from a simple pattern-matching task to one with deeper ties to the theory of formal languages and computation.

Section 4: Algorithmic Design for RBS Detection

We now present the design of an algorithm to solve the RBS detection problem. We begin with a straightforward dynamic programming formulation to establish a baseline, then develop an optimized version that leverages the advanced data structures discussed in Section 2.

4.1 A Foundational Dynamic Programming Approach

A natural way to solve problems involving optimal substructure, like RBS, is dynamic programming. Let RBS_Order[i][j] store the maximum RBS order of the substring S[i..j].

- **State:** RBS_Order[i][j] for $1 \le i \le j \le N$.
- **Initialization:** For all i,j, initialize RBS_Order[i][j]=−1.
- **Base Cases (Order 0):** First, identify all palindromic substrings. For each substring S[i..j] that is a palindrome, set RBS_Order[i][j]=0. This can be done in O(N2) time by checking every substring.
- Recursive Step (Order k>0): To compute RBS_Order[i][j], we must check all possible decompositions of S[i..j] into uvuR.
  $RBS\_Order[i][j] = 1 \le k < (j-i+1)/2 \max\{1 + RBS\_Order[i+k][j-k]\}$

  This maximum is taken over all values of k such that the "arm" condition holds: $S[i..i+k-1]R = S[j-k+1..j]$.

A direct implementation of this recurrence involves iterating through all substrings S[i..j] (in increasing order of length), and for each, iterating through all possible arm lengths k. The check for the arm condition takes O(k) time. This leads to an overall time complexity of O(N3), which is too slow for large inputs.

4.2 The Optimized Algorithm: DP with LCE Queries

The bottleneck in the naive DP approach is the repeated, costly check of the arm condition. This is eliminated by pre-processing the string to answer these queries instantly. The core strategy of the implemented algorithm is to perform an iterative deepening search, finding all RBS structures of order k and using them to discover all structures of order k+1.

Algorithm: Detect-RBS

1. Preprocessing Phase:
   a. LCE Data Structure: Construct the string T = S + '#' + S_reversed + '$', where S_reversed is the reverse of S, and # and $ are unique delimiters. Build a suffix array for T (the implementation uses a standard sorting-based approach), an LCP array using Kasai's algorithm, and a sparse table to handle Range Minimum Queries (RMQ). This preprocessing allows any LCE query on T to be answered in O(1) time.

b. Base Case Identification: Run Manacher's algorithm on S to find all maximal palindromic substrings.

2. Initialization Phase:
   a. DP Table: Create an N×N integer array, RBS_Order, and initialize all entries to -1.
   b. Worklists: Create an array of lists, Worklist[k] for k=0,1,...,N−1.
   c. Populate Base Cases: Use the output from Manacher's algorithm, which is adapted to find all palindromic substrings, not just the maximal ones. For each such substring S[i..j], set RBS_Order[i][j] = 0 and add the tuple (i, j) to Worklist.
3. Iterative Deepening Phase:
   a. Iterate for k from 0 to N−2:
   i. While Worklist[k] is not empty:
   - Dequeue a tuple (i, j). This represents an inner substring v=S[i..j] that has a maximal RBS order of k.
   - If this core is at the boundary of the original string (i.e., i=0 or j=N-1), it cannot be extended, so we skip it.
   - We now search for all symmetric arms u that can frame this v. This is equivalent to finding the longest string that is simultaneously a suffix of S[0..i−1] and the reverse of a prefix of S[j+1..N].
   - This length, let's call it Lmax, can be found with a single O(1) LCE query on the pre-processed string T. The query finds the longest common prefix between the suffix of T starting at index j+1 (representing the right-side arm) and the suffix of T starting at index N + 1 + (N - i) (representing the reversed left-side arm).
   - For each possible arm length l from 1 to Lmax:
   - Let the new, larger substring be p=S[i−l..j+l]. Its start index is i′=i−l and end index is j′=j+l.
   - The RBS order of this new string is at least k+1. We update its order if we have found a better one:
   - If RBS_Order[i'][j'] < k + 1:
   - Set RBS_Order[i'][j'] = k + 1.
   - If k+1 < N, enqueue the tuple (i', j') into Worklist[k+1].

After the loops complete, the RBS_Order table will contain the maximal RBS order for every substring of S that possesses such a structure.

Section 5: Analysis and Evaluation

5.1 Correctness Proof

The correctness of the Detect-RBS algorithm can be established by induction on the RBS order k.

- **Base Case (k=0):** The preprocessing step uses Manacher's algorithm, which is proven to correctly identify all maximal palindromic substrings in O(N) time.[18] The initialization phase correctly populates the RBS_Order table and Worklist with all substrings that are palindromes (RBS order 0). Therefore, the algorithm correctly identifies all RBS-0 structures.
- **Inductive Hypothesis:** Assume that after the completion of the main loop for order k−1, the algorithm has correctly identified all substrings with a maximal RBS order of k−1, and that Worklist[k-1] contains all such substrings.
- **Inductive Step:** Consider the iteration of the main loop for order k. The loop processes every tuple (i, j) from Worklist[k-1], which by the inductive hypothesis corresponds to a substring v=S[i..j] of order k−1. For each such v, the algorithm seeks to form a new string p=uvuR. The LCE query correctly and exhaustively finds the maximum possible length, Lmax, of a symmetric arm u that can frame v. The inner loop then considers every possible arm length l from 1 to Lmax. For each such arm, a new string p=S[i−l..j+l] is formed. By the definition of RBS, this string p has an RBS order of at least k+1. The algorithm updates RBS_Order[i-l][j+l] to k+1 and enqueues it for the next level. Because this process is performed for every RBS-(k-1) core and for every possible arm length, all valid RBS-(k+1) strings are discovered. The condition if RBS_Order[i'][j'] < k + 1 ensures that we only update if we find a higher-order structure and that each substring is enqueued at most once per order level, preventing redundant work and cycles. Thus, the algorithm correctly computes the maximal RBS order for all substrings.

5.2 Complexity Analysis

- **Time Complexity:**
  1. **Preprocessing:** Constructing the LCE data structure involves several steps. The implemented suffix array construction, which relies on standard sorting, takes O(N2logN) time in the worst case. Building the LCP array with Kasai's

algorithm takes O(N), and building the RMQ sparse table takes O(NlogN). Manacher's algorithm is O(N). The preprocessing phase is therefore dominated by the suffix array construction, resulting in a complexity of O(N2logN).

2. **Initialization:** Populating the RBS_Order table for all O(N2) palindromes takes O(N2) time.
3. **Iterative Deepening:** The total number of substrings is O(N2). A given substring (i, j) is enqueued at most once per order level due to the update condition. The key operations inside the loop are the dequeue, the O(1) LCE query, and the inner loop over arm lengths. The total work can be amortized. The total number of updates to the RBS_Order table is at most O(N2). The work is proportional to the sum of lengths of all maximal arms found, which can be bounded by O(N2). The complexity of this phase is therefore O(N2).

The overall time complexity of the implemented algorithm is the sum of its phases, which is dominated by the O(N2logN) preprocessing step. It is important to note that if a more advanced linear-time suffix array construction algorithm (like SA-IS) were used, the preprocessing would become O(NlogN), and the overall complexity would be O(N2), which is likely optimal as the output table is of size O(N2).

- **Space Complexity:**
  1. The LCE data structure requires O(N) space for the suffix array, LCP array, and RMQ table.
  2. The RBS_Order table requires O(N2) space.
  3. The worklists, in the worst case (e.g., for the string 0^N), might need to store O(N2) entries.
     The dominant factor is the DP table, leading to a total space complexity of O(N2).

5.3 Comparative Analysis

The proposed algorithm carves out a new niche in the landscape of algorithms for symmetric structures. The following table compares its properties to those of related, foundational algorithms.

Table 1: Comparison of Algorithmic Approaches for Detecting Symmetric Structures
| Algorithm/Approach | Structure Detected | Time Complexity (Worst-Case) | Space Complexity | Key Data Structures |

| :--- | :--- | :--- | :--- | :--- |
| Expand from Center 18 | All Maximal Palindromes |
O(N2) | O(1) | None |
| Manacher's Algorithm 18 | All Maximal Palindromes |
O(N) | O(N) | Palindrome Radii Array |
| Gapped Palindrome Counting 13 |
uvuR (unconstrained) | O(N) | O(N) | Suffix Tree |
| Gapped Palindrome Counting 13 |
uvuR (length-constrained) | O(NlogN) | O(N) | Suffix Tree, AVL Trees |
| Proposed Algorithm (Detect-RBS) | Recursive Bilateral Symmetry (uvuR where v is RBS) |
O(N2logN) | O(N2) | DP Table, LCE Data Structure |
This comparison highlights that while algorithms for simpler structures like standard or unconstrained gapped palindromes can achieve linear or near-linear time, the complex, recursive structural constraint of RBS necessitates a different approach. The polynomial complexity is a direct consequence of the problem's need to find an optimal hierarchical decomposition for every substring, a task more intricate than simple counting or finding maximal instances.

Part III: Conclusion and Future Directions

Section 6: Conclusion and Future Work

6.1 Summary of Contributions

This report has introduced and formalized the concept of Recursive Bilateral Symmetry (RBS) as a novel, hierarchical structure in strings. By synthesizing the notions of palindromicity, gapped palindromes, and fractal self-similarity, we have provided a rigorous, inductive definition that captures a rich class of nested symmetric patterns.

The primary contribution is the design, presentation, and analysis of the first known

algorithm for detecting RBS. The Detect-RBS algorithm employs a dynamic programming strategy accelerated by modern stringology techniques. By leveraging Manacher's algorithm for efficient base-case identification and a precomputed LCE data structure for constant-time verification of symmetric arms, the algorithm provides an effective solution to this new and complex combinatorial pattern matching problem.

6.2 Avenues for Future Research

The formalization of RBS and the development of a foundational algorithm open up several promising avenues for future investigation.

- **Optimization and Lower Bounds:** A key open question is whether the O(N2) complexity barrier can be broken. The current implementation runs in O(N2logN) due to a non-optimal suffix array construction; implementing a linear-time construction would achieve an O(N2) runtime. Could techniques from the fastest gapped palindrome counting algorithms, which use heavy-path decomposition on suffix trees [13], be adapted to reduce the time complexity further, perhaps to O(NlogN) or O(Nlog2N)? Conversely, can a formal proof be constructed to show an Ω(N2) lower bound on the time complexity for this problem?
- **Approximate RBS:** In many real-world applications, particularly bioinformatics, symmetric structures are rarely perfect. An important extension would be to develop algorithms for detecting approximate RBS, where the palindromic and symmetric arm conditions are relaxed to allow for a certain number of mismatches (Hamming distance) or insertions/deletions (edit distance). This aligns with research on streaming algorithms that find approximate palindromes [22] and would greatly enhance the practical applicability of the concept.
- **Generalization to Larger Alphabets:** The current algorithm is presented for binary strings but is directly applicable to constant-size alphabets. Analyzing its performance and potential adaptations for large or integer alphabets would be a valuable theoretical extension.
- **Applications:** The potential applications of RBS detection are broad and merit exploration:
    - **Bioinformatics:** Hierarchical, symmetric structures are known to exist in DNA (e.g., hairpin loops within larger stable regions) and protein structures. An algorithm for RBS could serve as a tool for identifying these complex functional or structural motifs.

- **Data Compression:** Compression algorithms are fundamentally about identifying and exploiting redundancy. RBS represents a highly structured form of redundancy. New compression schemes could be designed to explicitly find and encode RBS structures, potentially offering high compression ratios for data rich in such patterns.
- **Computer Graphics and Algorithmic Art:** The connection to fractals and L-systems is direct. The Detect-RBS algorithm could be used to analyze the structural complexity of strings generated by L-systems or to generate novel fractal-like patterns by controlling the RBS order of a descriptive string.

Works cited

1. Special Issue : Symmetries, and Symmetry Breaking in String Theory - MDPI, accessed June 20, 2025, https://www.mdpi.com/journal/symmetry/special_issues/3H9BCI1886
2. String theory - Wikipedia, accessed June 20, 2025, https://en.wikipedia.org/wiki/String_theory
3. M⇔Bst > SUSY Hierarchal Spin Equalizer - American Butterfly, accessed June 20, 2025, http://americanbutterfly.org/pt3/the-network-on-a-string/susy-hierarchal-spin-equalizer
4. [2506.15194] Cosmic Strings in Multi-Step Symmetry Breaking - arXiv, accessed June 20, 2025, https://arxiv.org/abs/2506.15194
5. Cosmic Strings in Multi-Step Symmetry Breaking - arXiv, accessed June 20, 2025, https://arxiv.org/pdf/2506.15194
6. On symmetry in strings, sequences and languages - Taylor & Francis Online, accessed June 20, 2025, https://www.tandfonline.com/doi/abs/10.1080/00207169408804333
7. Algorithms-on-Strings.pdf - ResearchGate, accessed June 20, 2025, https://www.researchgate.net/profile/Maxime-Crochemore/publication/220693689_Algorithms_on_Strings/links/55e3276d08ae2fac47211401/Algorithms-on-Strings.pdf
8. www.olabs.edu.in, accessed June 20, 2025, https://www.olabs.edu.in/?sub=97&brch=48&sim=492&cnt=1#:~:text=A%20palindrome%20string%20is%20a,it%20down%20into%20smaller%20subproblems.
9. Using recursion to determine whether a word is a palindrome (article ..., accessed June 20, 2025, https://www.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/using-recursion-to-determine-whether-a-word-is-a-palindrome
10. recursive definition of a palindrome help - Math Stack Exchange, accessed June 20, 2025, https://math.stackexchange.com/questions/1783238/recursive-definition-of-a-palindrome-help
11. Recursively Defining Languages Homework - RIT, accessed June 20, 2025,

https://www.cs.rit.edu/~jmg/courses/cs380/20021/slides/recursivelang.pdf
12. Efficient Algorithms for Counting Gapped Palindromes - DROPS - Schloss Dagstuhl, accessed June 20, 2025, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CPM.2021.23
13. Efficient Algorithms for Counting Gapped Palindromes - DROPS, accessed June 20, 2025, https://drops.dagstuhl.de/storage/00lipics/lipics-vol191-cpm2021/LIPIcs.CPM.2021.23/LIPIcs.CPM.2021.23.pdf
14. Fractals in C/C++ | GeeksforGeeks, accessed June 20, 2025, https://www.geeksforgeeks.org/fractals-in-cc/
15. Chapter 8: Fractals - Nature of Code, accessed June 20, 2025, https://natureofcode.com/fractals/
16. L-system - Wikipedia, accessed June 20, 2025, https://en.wikipedia.org/wiki/L-system
17. Fractals in Computer Graphics - GeeksforGeeks, accessed June 20, 2025, https://www.geeksforgeeks.org/fractals-in-computer-graphics/
18. 647. Palindromic Substrings - In-Depth Explanation - AlgoMonster, accessed June 20, 2025, https://algo.monster/liteproblems/647
19. Palindromes In Sturmian Strings - arXiv, accessed June 20, 2025, https://arxiv.org/pdf/1002.4606
20. Shortest cover after edit - arXiv, accessed June 20, 2025, https://arxiv.org/pdf/2402.17428
21. [Literature Review] Shortest cover after edit - Moonlight, accessed June 20, 2025, https://www.themoonlight.io/review/shortest-cover-after-edit
22. Tight tradeoffs for approximating palindromes in streams - SciSpace, accessed June 20, 2025, https://scispace.com/pdf/tight-tradeoffs-for-approximating-palindromes-in-streams-vwy9w6mg7t.pdf