Palindrome Recognition In The Streaming Model

Petra Berenbrink[1], Funda Ergün[1,2], Frederik Mallmann-Trenn[1], and Erfan Sadeqi Azer[1]

1 **Simon Fraser University**
2 **Indiana University**

Abstract

A palindrome is defined as a string which reads forwards the same as backwards, like, for example, the string "racecar". In the *Palindrome Problem*, one tries to find all *palindromes* in a given string. In contrast, in the case of the *Longest Palindromic Substring Problem*, the goal is to find an arbitrary one of the longest palindromes in the string.
In this paper we present three algorithms in the streaming model for the the above problems, where at any point in time we are only allowed to use sublinear space. We first present a one-pass randomized algorithm that solves the *Palindrome Problem*. It has an *additive* error and uses $O(\sqrt{n})$ space. We also give two variants of the algorithm which solve related and practical problems. The second algorithm determines the exact locations of *all* longest palindromes using two passes and $O(\sqrt{n})$ space. The third algorithm is a one-pass randomized algorithm, which solves the *Longest Palindromic Substring Problem*. It has a *multiplicative* error using only $O(\log(n))$ space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Palindromes; Streaming Model; Complementary Palindrome

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

A palindrome is defined as a string which reads forwards the same as backwards, e.g., the string "racecar". In the *Palindrome Problem* one tries to find all *palindromes* (palindromic substrings) in an input string. A related problem is the *Longest Palindromic Substring Problem* in which one tries to find any one of the longest palindromes in the input.
In this paper we regard the streaming version of both problems, where the input arrives over time (or, alternatively, is read as a stream) and the algorithms are allowed space sub linear in the size of the input. Our first contribution is a one-pass randomized algorithm that solves the *Palindrome Problem*. It has an *additive* error and uses $O(\sqrt{n})$ space. The second contribution is a two-pass algorithm which determines the exact locations of all longest palindromes. It uses the first algorithm as the first pass and uses $O(\sqrt{n})$ space. The third is a one-pass randomized algorithm for the *Longest Palindromic Substring Problem*. It has a *multiplicative* error using $O(\log(n))$ space. We also give two variants of the first algorithm which solve other related practical problems. [1]
Palindromes have several relations to practical areas such as computational biology. As palindromic structures can frequently be found in proteins and identifying them gives researchers hints about the structure of nucleic acids.

[1] The full version of this paper can be accessed at arXiv:1308.3466.

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related work While palindromes are well-studied, to the best of our knowledge there are no results for the streaming model. Manacher [5] presents a linear time online algorithm that reports at any time whether all symbols seen so far form a palindrome. The authors of [1] show how to modify this algorithm in order to find all palindromic substrings in linear time (using a parallel algorithm).

Some of the techniques used in this paper have their origin in the streaming pattern matching literature. In the *Pattern Matching Problem*, one tries to find all occurrences of a given pattern P in a text T. The first algorithm for pattern matching in the streaming model was shown in [6] and requires $O(\log(m))$ space. The authors of [3] give a simpler pattern matching algorithm with no preprocessing, as well as a related streaming algorithm for estimating a stream's Hamming distance to p-periodicity. Breslauer and Galil [2] provide an algorithm which does not report false negatives and can also be run in real-time. All of the above algorithms in the string model take advantage of Karp-Rabin fingerprints [4].

Our results In this paper we present three algorithms, *ApproxSqrt*, *Exact*, and *ApproxLog* for finding palindromes and estimating their length in a given stream S of length n. We assume that the workspace is bounded while the output space is unlimited.

Given an index m in stream S, $P[m]$ denotes the palindrome of maximal length centered at index m of S. Our algorithms identify a palindrome $P[m]$ by its *midpoint* m and by its length $\ell(m)$. Our first algorithm outputs all palindromes in S and therefore solves the *Palindrome Problem*.

▶ **Theorem 1** (*ApproxSqrt*). *For any* $\varepsilon \in [1/\sqrt{n}, 1]$ Algorithm ApproxSqrt(S, ε) *reports for every palindrome* $P[m]$ *in* S *its midpoint* m *as well as an estimate* $\tilde{\ell}(m)$ *(of* $\ell(m)$*) such that w.h.p.*[2] $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$. *The algorithm makes one pass over* S, *uses* $O(n/\varepsilon)$ *time, and* $O(\sqrt{n}/\varepsilon)$ *space.*

The algorithm can easily be modified to report all palindromes $P[m]$ in S with $\ell(m) \geq t$ and no $P[m]$ with $\ell(m) < t - \varepsilon\sqrt{n}$ for some threshold $t \in \mathbb{N}$. For $t \leq \sqrt{n}$ one can modify the algorithm to report a palindrome $P[m]$ if and only if $\ell(m) \geq t$.

Our next algorithm, *Exact*, uses two-passes to solve the *Longest Palindromic Substring Problem*. It uses *ApproxSqrt* as the first pass. In the second pass the algorithm finds the midpoints of all palindromes of length exactly $\ell_{\max}$ where $\ell_{\max}$ is the (initially unknown) length of the longest palindrome in S.

▶ **Theorem 2** (*Exact*). Algorithm Exact *reports w.h.p.* ℓ_{max} *and* m *for all palindromes* $P[m]$ *with a length of* ℓ_{max}. *The algorithm makes two passes over* S, *uses* $O(n)$ *time, and* $O(\sqrt{n})$ *space.*

Arguably the most significant contribution of this paper is an algorithm which requires only logarithmic space. In contrast to *ApproxSqrt* (Theorem 1) this algorithm has a multiplicative error and it reports only one of the longest palindromes (see *Longest Palindromic Substring Problem*) instead of all of them due to the limited space.

▶ **Theorem 3** (*ApproxLog*). *For any* ε *in* $(0, 1]$, Algorithm ApproxLog *reports w.h.p. an arbitrary palindrome* $P[m]$ *of length at least* $\ell_{max}/(1+\varepsilon)$. *The algorithm makes one pass over* S, *uses* $O(\frac{n\log(n)}{\varepsilon\log(1+\varepsilon)})$ *time, and* $O(\frac{\log(n)}{\varepsilon\log(1+\varepsilon)})$ *space.*

We also show two practical generalisations of our algorithms which can be run simultaneously. These results are presented in the next observation and the next lemma.

[2] We say an event happens *with high probability* (w.h.p.) if its probability is at least $1 - 1/n$.

▶ Observation 4. For $\ell_{max} \geq \sqrt{n}$, there is an algorithm which reports w.h.p. the midpoints of all palindromes $P[m]$ with $\ell(m) > \ell_{max} - \varepsilon\sqrt{n}$. The algorithm makes one pass over S, uses $O(n/\varepsilon)$ time, and $O(\sqrt{n}/\varepsilon)$ space.

▶ **Lemma 5.** *For $\ell_{max} < \sqrt{n}$, there is an algorithm which reports w.h.p. ℓ_{max} and a $P[m]$ s.t. $\ell(m) = \ell_{max}$. The algorithm makes one pass over S, uses $O(n)$ time, and $O(\sqrt{n})$ space.*

In the full version of the paper we will show an almost matching bound for the additive error of *Algorithm ApproxSqrt*. In more detail, we will show that any randomized one-pass algorithm that approximates the length of the longest palindrome up to an additive error of $\varepsilon\sqrt{n}$ must use $\Omega(\sqrt{n}/\varepsilon)$ space.

2 Model and Definitions

Let $S \in \Sigma^n$ denote the input stream of length n over an alphabet Σ. For simplicity we assume symbols to be positive integers, i.e., $\Sigma \subset \mathbb{N}$. We define $S[i]$ as the symbol at index i and $S[i,j] = S[i], S[i+1], \ldots S[j]$. In this paper we use the streaming model: In one *pass* the algorithm goes over the whole input stream S, reading $S[i]$ in *iteration* i of the pass. In this paper we assume that the algorithm has a memory of size $o(n)$, but the output space is unlimited. We use the so-called word model where the space equals the number of $O(\log(n))$ registers (See [2]).

S contains an *odd palindrome* of length ℓ with midpoint $m \in \{\ell, \ldots, n-\ell\}$ if $S[m-i] = S[m+i]$ for all $i \in \{1, \ldots, \ell\}$. Similarly, S contains an *even palindrome* of length ℓ if $S[m-i+1] = S[m+i]$ for all $i \in \{1, \ldots, \ell\}$. In other words, a palindrome is odd if and only if its length is odd. For simplicity, our algorithms assume palindromes to be even - it is easy to adjust our results for finding odd palindromes by apply the algorithm to $S[1]S[1]S[2]S[2] \cdots S[n]S[n]$ instead of $S[1,n]$.

The maximal palindrome (the palindrome of maximal length) in $S[1,i]$ with midpoint m is called $P[m,i]$ and the maximal palindrome in S with midpoint m is called $P[m]$ which equals $P[m,n]$. We define $\ell(m,i)$ as the maximum length of the palindrome with midpoint m in the substring $S[1,i]$. The maximal length of the palindrome in S with midpoint m is denoted by $\ell(m)$. Moreover, for $z \in \mathbb{Z} \setminus \{1, \ldots, n\}$ we define $\ell(z) = 0$. Furthermore, for $\ell^* \in \mathbb{N}$ we define $P[m]$ to be an ℓ^*-palindrome if $\ell(m) \geq \ell^*$. Throughout this paper, $\tilde{\ell}()$ refers to an estimate of $\ell()$.

For many biological applications such as *nucleic acid secondary structure prediction*, one is interested in complementary palindromes which are defined in Definition 21 in the Appendix.

We use the KR-Fingerprint, which was first defined by Karp and Rabin [4] to compress strings and was later used in the streaming pattern matching problem (see [6], [3], and [2]). For a string S' we define the forward fingerprint (similar to [2]) and its reverse as follows. $\phi^F_{r,p}(S') = \left(\sum_{i=1}^{|S'|} S'[i] \cdot r^i\right) \mod p$ $\phi^R_{r,p}(S') = \left(\sum_{i=1}^{|S'|} S'[i] \cdot r^{l-i+1}\right) \mod p, <$ where p is an arbitrary prime number in $[n^4, n^5]$ and r is randomly chosen from $\{1, \ldots, p\}$. We write ϕ^F (ϕ^R respectively) as opposed to $\phi^F_{r,p}$($\phi^R_{r,p}$ respectively) whenever r and p are fixed. We define for $1 \leq i \leq j \leq n$ the fingerprint $F^F(i,j)$ as the fingerprint of $S[i,j]$, i.e., $F^F(i,j) = \phi^F(S[i,j]) = r^{-(i-1)}(\phi^F(S[1,j]) - \phi^F(S[1,i-1])) \mod p$. Similarly, $F^R(i,j) = \phi^R(S[i,j]) = \phi^R(S[1,j]) - r^{j-i+1} \cdot \phi^R(S[1,i-1]) \mod p$. For every $1 \leq i \leq n - \sqrt{n}$ the fingerprints $F^F(1, i-1-\sqrt{n})$ and $F^R(1, i-1-\sqrt{n})$ are called *Master Fingerprints*. Note that it is easy to obtain $F^F(i,j+1)$ by adding the term $S[j+1]r^{j+1}$ to $F^F(i,j)$. Similarly, we obtain $F^F(i+1,j)$ by subtracting $S[i]$ from $r^{-1} \cdot F^F(i,j)$. The authors of [2] observe useful properties which we state in Lemma 19 and Lemma 20.

3 *Algorithm Simple ApproxSqrt*

In this section, we introduce a simple one-pass algorithm which reports all midpoints and length estimates of palindromes in S. Throughout this paper we use i to denote the current index which the algorithm reads. *Simple ApproxSqrt* keeps the last $2\sqrt{n}$ symbols of $S[1,i]$ in the memory.

It is easy to determine the exact length palindromes of length less than $\sqrt{n}$ since any such palindrome is fully contained in memory at some point. However, in order to achieve a better time bound the algorithm only *approximates* the length of short palindromes. It is more complicated to estimate the length of a palindrome with a length of at least $\sqrt{n}$. However, *Simple ApproxSqrt* detects that its length is at least $\sqrt{n}$ and stores it as an R_S-entry (introduced later) in a list L_i. The R_S-entry contains the midpoint as well as a length estimate of the palindrome, which is updated as i increases.

In order to estimate the lengths of the long palindromes the algorithm designates certain indices of S as *checkpoints*. For every checkpoint c the algorithm stores a fingerprint $F^R(1,c)$ enabling the algorithm to do the following. For every midpoint m of a long palindrome: Whenever the distance from a checkpoint c to m (c occurs before m) equals the distance from m to i, the algorithm compares the substring from c to m to the reverse of the substring from m to i by using fingerprints. We refer to this operation as *checking* $P[m]$ against checkpoint c. If $S[c+1,m]^R = S[m+1,i]$, then we say that $P[m]$ was *sucessfully checked* with c and the algorithm updates the length estimate for $P[m]$, $\tilde{\ell}(m)$. The next time the algorithm possibly updates $\tilde{\ell}(m)$ is after d iterations where d equals the distance between checkpoints. This distance d gives the additive approximation. See Figure 1 in the Appendix for an illustration.

We need the following definitions before we state the algorithm: For $k \in \mathbb{N}$ with $0 \leq k \leq \lfloor \frac{\sqrt{n}}{\varepsilon} \rfloor$ checkpoint c_k is the index at position $k \cdot \lfloor \varepsilon\sqrt{n} \rfloor$ thus checkpoints are $\lfloor \varepsilon\sqrt{n} \rfloor$ indices apart. Whenever we say that an algorithm stores a checkpoint, this means storing the data belonging to this checkpoint. Additionally, the algorithm stores *Fingerprint Pairs*, fingerprints of size $\lfloor \varepsilon\sqrt{n} \rfloor, 2\lfloor \varepsilon\sqrt{n} \rfloor, \ldots$ starting or ending in the middle of the sliding window. In the following, we first describe the data that the algorithm has in its memory after reading $S[1,i-1]$, then we describe the algorithm itself. Let $R_S(m,i)$ denote the representation of $P[m]$ which is stored at time i. As opposed to storing $P[m]$ directly, the algorithm stores m, $\tilde{\ell}(m,i)$, $F^F(1,m)$, and $F^R(1,m)$.

Memory invariants. Just before algorithm *Simple ApproxSqrt* reads $S[i]$ it has stored the following information. Note that, for ease of referencing, during an iteration i data structures are indexed with the iteration number i. That is, for instance, L_{i-1} is called Li after $S[i]$ is read.

1. The contents of the sliding window $S[i-2\sqrt{n}-1, i-1]$.
2. The two *Master Fingerprints* $F^F(1,i-1)$ and $F^R(1,i-1)$.
3. A list of *Fingerprint Pairs*: Let r be the maximum integer s.t. $r \cdot \lfloor \varepsilon\sqrt{n} \rfloor < \sqrt{n}$. For $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$ the algorithm stores the pair $F^R((i-\sqrt{n})-j, (i-\sqrt{n})-1)$, and $F^F(i-\sqrt{n}, (i-\sqrt{n})+j-1)$. See Figure 2 for an illustration.
4. A list CL_{i-1} which consists of all fingerprints of prefixes of S ending at already seen checkpoints, i.e., $\mathrm{CL}_{i-1} = \left[F^R(1,c_1), F^R(1,c_2), \ldots, F^R\left(1, c_{\lfloor (i-1)/\lfloor \varepsilon\sqrt{n} \rfloor \rfloor}\right)\right]$
5. A list L_{i-1} containing representation of all $\sqrt{n}$-palindromes with a midpoint located in $S[1,(i-1)-\sqrt{n}]$. The j^{th} entry of L_{i-1} has the form $R_S(m_j, i-1) = (m_j, \tilde{\ell}(m_j, i-1), F^F(1,m_j), F^R(1,m_j))$ where
   (a) m_j is the midpoint of the j^{th} palindrome in $S[1,(i-1)-\sqrt{n}]$ with a length of at

least $\sqrt{n}$. Therefore, $m_j < m_{j+1}$ for $1 \le j \le |L_{i-1}| - 1$.
(b) $\tilde{\ell}(m_j, i-1)$ is the current estimate of $\ell(m_j, i-1)$.

In the following, we explain how the algorithm maintains the above invariants.
Maintenance. At iteration i the algorithm performs the following steps. It is implicit that L_{i-1} and $CLi-1$ become L_i and CL_i respectively.

1. Read $S[i]$, set $m = i - \sqrt{n}$. Update the sliding window to $S[m-\sqrt{n}, i] = S[m-2\sqrt{n}, i]$
2. Update the *Master Fingerprints* to be $F^F(1,i)$ and $F^R(1,i)$.
3. If i is a checkpoint (i.e., a multiple of $\lfloor \varepsilon\sqrt{n} \rfloor$), then add $F^R(1,i)$ to CL_i.
4. Update all *Fingerprint Pairs*: For $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, , \ldots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$
   - Update $F^R(m-j, m-1)$ to $F^R(m-j+1, m)$ and $F^F(m, m+j-1)$ to $F^F(m+1, m+j)$.
   - If $F^R(m-j+1, m) = F^F(m+1, m+j)$, then set $\tilde{\ell}(m,i) = j$.
   - If $\tilde{\ell}(m,i) < \sqrt{n}$, output m and $\tilde{\ell}(m,i)$.
5. If $\tilde{\ell}(m,i) \ge \sqrt{n}$, add *add* $R_S(m,i)$ to L_i:
   $L_i = L_i \circ (m, \tilde{\ell}(m,i), F^F(1,m), F^R(1,m))$.
6. For all c_k with $1 \le k \le \lfloor \frac{i}{\lfloor \varepsilon\sqrt{n} \rfloor} \rfloor$ and $R_S(m_j, i) \in L_i$ with $i - m_j = m_j - c_k$, check if $\tilde{\ell}(m_j, i)$ can be updated:
   - If the left side of m_j is the reverse of the right side of m_j (i.e., $F^R(c_k+1, m_j) = F^F(m_j+1, i)$) then update $R_S(m_j, i)$ by updating $\tilde{\ell}(m_j, i)$ to $i - m_j$.
7. If $i = n$, then report L_n.

In all proofs in this paper which hold w.h.p. we assume that fingerprints do not fail as we take less than n^2 fingerprints and by Lemma 20, the probability that a fingerprint fails is at most $1/n^4$. Thus, by applying the union bound the probability that no fingerprint fails is at least $1 - n^{-2}$. The following lemma shows that the *Simple ApproxSqrt* finds all palindromes along with the estimates as stated in Theorem 1. *Simple ApproxSqrt* does not fulfill the time and space bounds of Theorem 1; we will later show how to improve its efficiency. The proof can be found in the appendix.

▶ **Lemma 6.** *For any ε in $[1/\sqrt{n}, 1]$ $\mathrm{ApproxSqrt}(S, \varepsilon)$ reports for every palindrome $P[m]$ in S its midpoint m as well as an estimate $\tilde{\ell}(m)$ such that w.h.p. $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \le \ell(m)$.*

4 A space-efficient version

In this section, we show how to modify *Simple ApproxSqrt* so that it matches the time and space requirements of Theorem 1. The main idea of the space improvement is to store the lists L_i in a compressed form.
Compression It is possible in the simple algorithm for L_i to have linear length. In such cases S contains many overlapping palindromes which show a certain *periodic* pattern as shown in Corollary 11, which our algorithm exploits to compress the entries of L_i. This idea was first introduced in [6], and is used in [3], and [2]. More specifically, our technique is a modification of the compression in [2]. In the following, we give some definitions in order to show how to compress the list. First we define a *run* which is a sequence of midpoints of overlapping palindromes.

▶ **Definition 7** (ℓ^*–Run)**.** Let ℓ^* be an arbitrary integer and $h \ge 3$. Let $m_1, m_2, m_3, \ldots, m_h$ be consecutive midpoints of ℓ^*-palindromes in S. $m_1, \ldots, m_h$ form an *ℓ^*-run* if $m_{j+1} - m_j \le \ell^*/2$ for all $j \in \{1, \ldots, h-1\}$.

In Corollary 11 we show that $m_2 - m_1 = m_3 - m_2 = \cdots = m_h - m_{h-1}$. We say that a run is maximal if the run cannot be extended by other palindromes. More formally:

▶ **Definition 8** (Maximal ℓ^*−Run). An ℓ^*-run over $m_1, \ldots, m_h$ is *maximal* it satisfies both of the following: i) $\ell(m_1 - (m_2 - m_1)) < \ell^*$, ii) $\ell(m_h + (m_2 - m_1)) < \ell^*$.

Simple ApproxSqrt stores palindromes explicitly in L_i, i.e., $L_i = [R_S(m_1, i); \ldots ; R_S(m_{|L_i|}, i)]$ where $R_S(m_j, i) = (m_j, \tilde{\ell}(m_j, i), F^F(1, m_j), F^R(1, m_j))$, for all $j \in \{1, 2, \ldots, h\}$. The improved *Algorithm ApproxSqrt* stores these midpoints in a compressed way in list $\hat{L}_i$. *ApproxSqrt* distinguishes among three cases: Those palindromes which

1. are not part of a $\sqrt{n}$-run are stored explicitly as before. We call them R_S-entries. Let $P[m, i]$ be such a palindrome. After iteration i the algorithm stores $R_S(m, i)$.
2. form a maximal $\sqrt{n}$-run are stored in a data structure called R_F-entry. Let $m_1, \ldots, m_h$ be the midpoints of a maximal $\sqrt{n}$-run. The data structure stores the following information.
   - m_1, $m_2 - m_1$, h, $\tilde{\ell}(m_1, i)$, $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$, $\tilde{\ell}(m_h, i)$,
   - $F^F(1, m_1)$, $F^R(1, m_1)$, $F^F(m_1 + 1, m_2)$, $F^R(m_1 + 1, m_2)$
3. form a $\sqrt{n}$-run which is not maximal (i.e., it can possibly be extended) in a data structure called R_{NF}-entry. The information stored in an R_{NF}-entry is the same as in an R_F-entry, but it does not contain the entries: $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$, and $\tilde{\ell}(m_h, i)$.

The algorithm stores only the estimate (of the length) and the midpoint of the following palindromes explicitly.

- $P[m]$ for an R_S-entry (Therefore all palindromes which are not part of a $\sqrt{n}$-run)
- $P[m_1]$, $P[m_{\lfloor (h+1)/2 \rfloor}]$, $P[m_{\lceil (h+1)/2 \rceil}]$, and $P[m_h]$ for an R_F-entry
- $P[m_1]$ for an R_{NF}-entry.

In what follows we refer to the above listed palindromes as *explicitly stored* palindromes. We argue in Observation 14 that in any interval of length $\sqrt{n}$ the number of *explicitly stored* palindromes is bounded by a constant.

4.1 *Algorithm ApproxSqrt*

In this subsection, we describe some modifications of *Simple ApproxSqrt* in order to obtain a space complexity of $O(\frac{\sqrt{n}}{\varepsilon})$ and a total running time of $O(\frac{n}{\varepsilon})$. *ApproxSqrt* is the same as *Simple ApproxSqrt*, but it compresses the stored palindromes. *ApproxSqrt* uses the same memory invariants as *Simple ApproxSqrt*, but it uses $\hat{L}_i$ as opposed to L_i.
ApproxSqrt uses the first four steps of *Simple ApproxSqrt*. Step 5, Step 6, and Step 7 are replaced. The modified Step 5 ensures that there are at most two R_S-entries per interval of length $\sqrt{n}$. Moreover, Step 6 is adjusted since *ApproxSqrt* stores only the length estimate of explicitly stored palindromes.

5. If $\tilde{\ell}(m, i) \geq \sqrt{n}$, obtain $\hat{L}_i$ by adding the palindrome with midpoint $m(= i - \sqrt{n})$ to $\hat{L}_{i-1}$ as follows:
   a. The last element in $\hat{L}_i$ is the following R_{NF}-entry
   $\big(m_1, m_2 - m_1, h, \tilde{\ell}(m_1, i), F^F(1, m_1), F^R(1, m_1), F^F(m_1 + 1, m_2), F^R(m_1 + 1, m_2)\big)$.
      i. If the palindrome can be added to this run, i.e., $m = m_1 + h(m_2 - m_1)$, then we increment the h in the R_{NF}-entry by 1.
      ii. If the palindrome cannot be added: Store $P[m, i]$ as an R_S-entry: $\hat{L}_i = \hat{L}_i \circ (m, \tilde{\ell}(m, i), F^F(1, m), F^R(1, m))$. Moreover, convert the R_{NF}-entry into the R_F-entry by adding $\tilde{\ell}(m_{\lfloor \frac{1+h}{2} \rfloor}, i)$, $\tilde{\ell}(m_{\lceil \frac{1+h}{2} \rceil}, i)$ and $\tilde{\ell}(m_h, i)$: First we calculate $m_{\lfloor \frac{1+h}{2} \rfloor} =$

$m_1 + \left(\lfloor\frac{1+h}{2} - 1\rfloor\right)(m_2 - m_1)$. One can calculate $m_{\lceil\frac{1+h}{2}\rceil}$ similarly. For $m' \in \{m_{\lfloor\frac{1+h}{2}\rfloor}, m_{\lceil\frac{1+h}{2}\rceil}, m_h\}$ calculate $\tilde{\ell}(m', i) = \max_{i-2\sqrt{n}\le j\le i}\{j - m' \mid \exists c_k \text{ with } j-m' = m'-c_k \text{ and } F^R(c_k+1, m') = F^F(m'+1, j)\}$.

b. The last two entries in $\hat{L}_i$ are stored as R_S-entries and together with $P[m,i]$ form a $\sqrt{n}$-run. Then remove the entries of the two palindromes out of $\hat{L}_{i-1}$ and add a new R_{NF}-entry with all three palindromes to $\hat{L}_{i-1}$: $m_1, \tilde{\ell}(m_1,i), F^F(1,m_1), F^F(1,m_2), F^R(1,m_1), F^R(1,m_2), m_2 - m_1, h = 3$ Retrieve $F^F(m_1+1, m_2)$ and $F^R(m_1+1, m_2)$ as shown in Lemma 19.

c. Otherwise, store $P[m,i]$ as an R_S-entry: $\hat{L}_i = \hat{L}_i \circ (m, \tilde{\ell}(m,i), F^F(1,m), F^R(1,m))$

6. This step is similar to step 6 of *Simple ApproxSqrt* the only difference is that we check only for explicitly stored palindromes if they can be extended outwards. [3]

7. If $i = n$. If the last element in $\hat{L}_i$ is an R_{NF}-entry, then convert it into an R_F-entry as in 5(a)ii. Report L_n.

4.2 Structural Properties

In this subsection, we prove structural properties of palindromes. These properties allow us to compress (by using R_S-entries and R_F-entries) overlapping palindromes $P[m_1], \ldots, P[m_h]$ in such a way that at any iteration i all the information stored $R_S(m_1,i), \ldots, R_S(m_h,i)$ is available. The structural properties imply, informally speaking, that the palindromes are either far from each other, leading to a small number of them, or they are overlapping and it is possible to compress them. Lemma 10 shows this structure for short intervals containing at least three palindromes. Corollary 11 shows a similar structure for palindromes of a run which is used by *ApproxSqrt*. We first give the common definition of periodicity.

▶ **Definition 9** (*period*)**.** A string S' is said to have period p if it consists of repetitions of a block of p symbols. Formally, S' has period p if $S'[j] = S[j+p]$ for all $j = 1, \ldots, |S'| - p$. [4]

▶ **Lemma 10.** *Let $m_1 < m_2 < m_3 < \cdots < m_h$ be indices in S that are consecutive midpoints of ℓ^*-palindromes for an arbitrary natural number ℓ^*. If $m_h - m_1 \le \ell^*$, then*

(a) $m_1, m_2, m_3, \ldots, m_h$ are equally spaced in S, i.e., $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$

(b) $S[m_1+1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}} w & h \text{ is even} \end{cases}$ *, where $w = S[m_1+1, m_2]$.*

Proof. Given $m_1, m_2, \ldots, m_h$ and ℓ^* we prove the following stronger claim by induction over the midpoints $\{m_1, \ldots, m_j\}$. (a') $m_1, m_2, \ldots, m_j$ are equally spaced. (b') $S[m_1+1, m_j+\ell^*]$ is a prefix of ww^Rww^R... .
Base case $j = 2$: Since we assume m_1 is the midpoint of an ℓ^*-palindrome and $\ell^* \ge m_h - m_1 \ge m_2 - m_1 = |w|$, we have that $S[m_1 - |w| + 1, m_1] = w^R$. Recall that $\ell(m_2) \ge \ell^* \ge |w|$ and thus, $S[m_1+1, m_2+|w|] = ww^R$.
We can continue this argument and derive that $S[m_1+1, m_2+\ell^*]$ is a prefix of $ww^R \ldots ww^R$.
(a') for $j = 2$ holds trivially.
Inductive step $j-1 \to j$: Assume (a') and (b') hold up to m_{j-1}. We first argue that $|m_j - m_1|$ is a multiple of $|m_2 - m_1| = |w|$. Suppose $m_j = m_1 + |w| \cdot q + r$ for some integers

[3] This step is only important for the running time.
[4] Here, p is called a period for S' even if $p > |S'|/2$

$q \geq 0$ and $r \in \{1, \ldots, |w|-1\}$. Since $m_j \leq m_{j-1} + \ell^*$, the interval $[m_1 + 1, m_{j-1} + \ell^*]$ contains m_j. Therefore, by inductive hypothesis, $m_j - r$ is an index where either w or w^R starts. This implies that the prefix of ww^R(or $w^R w$) of size $2r$ is a palindrome and the string ww^R(or $w^R w$) has period $2r$. On the other hand, by consecutiveness assumption, there is no midpoint of an ℓ^*-palindrome in the interval $[m_1 + 1, m_2 - 1]$. does not have a period of $2p$, a contradiction. We derive that $m_j - m_1$ is multiple of $|w|$.
Hence, we assume $m_j = m_{j-1} + q \cdot |w|$ for some q. The assumption that m_j is a midpoint of an ℓ^*-palindrome beside the inductive hypothesis implies (b') for j. The structure of $S[m_{j-1}+|w|-\ell^*+1, m_{j-1}+|w|+\ell^*]$ shows that $m_{j-1}+|w|$ is a midpoint of an ℓ^*-palindrome. This means that $m_j = m_{j-1} + |w|$. This gives (a') and yields the induction step. ◀

Corollary 11 shows the structure of overlapping palindromes and is essential for the compression. The main difference between Corollary 11 and Lemma 10 is the required distance between the midpoints of a run. Lemma 10 assumes that every palindrome in the run overlaps with all other palindromes. In contrast, Corollary 11 assumes that every palindrome $P[m_j]$ overlaps with $P[m_{j-2}]$, $P[m_{j-1}]$, $P[m_{j+1}]$, and $P[m_{j+2}]$. It can be proven by an induction over the midpoints and using Lemma 10. The proof is in the appendix.

▶ **Corollary 11.** *If $m_1, m_2, \ldots, m_h$ form an ℓ^*-run for an arbitrary natural number ℓ^* then*

(a) $m_1, m_2, m_3, \ldots, m_h$ are equally spaced in S, i.e., $|m_2 - m_1| = |m_{k+1} - m_k|$ $\forall k \in \{1, \ldots, h-1\}$

(b) $S[m_1 + 1, m_h] = \begin{cases} (ww^R)^{\frac{h-1}{2}} & h \text{ is odd} \\ (ww^R)^{\frac{h-2}{2}} w & h \text{ is even} \end{cases}$ *, where $w = S[m_1 + 1, m_2]$.*

Lemma 12 shows the pattern for the lengths of the palindromes in each half of the run. This allows us to only store a constant number of length estimates per run. The proof can be found in the appendix.

▶ **Lemma 12.** *At iteration i, let $m_1, m_2, m_3, ..., m_h$ be midpoints of a maximal ℓ^*-run in $S[1, i]$ for an arbitrary natural number ℓ^*. For any midpoint m_j, we have:*

$$\ell(m_j, i) = \begin{cases} \ell(m_1, i) + (j-1) \cdot (m_2 - m_1) & j < \frac{h+1}{2} \\ \ell(m_h, i) + (h-j) \cdot (m_2 - m_1) & j > \frac{h+1}{2} \end{cases}$$

4.3 Analysis

We show that one can convert R_S-entries into a run and vice versa and *ApproxSqrt*'s maintenance of R_F-entries and R_{NF}-entries does not impair the length estimates. The following lemma shows that one can retrieve the length estimate of a palindrome as well as its fingerprint from an R_F-entry.

▶ **Lemma 13.** *At iteration i, the R_F-entry over $m_1, m_2, \ldots, m_h$ is a lossless compression of* $[R_S(m_1, i); \ldots; R_S(m_h, i)]$

Let *Compressed Run* be the general term for R_F-entry and R_{NF}-entry. We argue that in any interval of length $\sqrt{n}$ we only need to store at most two single palindromes and two *Compressed Runs*. Suppose there were three R_S-entries, then, by Corollary 11, they form a $\sqrt{n}$-run since they overlap each other. Therefore, the three R_S-entries would be stored in a *Compressed Run*. For a similar reason there cannot be more than two *Compressed Runs* in one interval of length $\sqrt{n}$. We derive the following observation.

▶ Observation 14. For any interval of length $\sqrt{n}$ there can be at most two R_S-entries and two *Compressed Runs* in L^*.

We now have what we need in order to prove Theorem 1; the proof is given in the appendix.

5 Algorithm Exact

This section describes *Algorithm Exact* which determines the exact length of the longest palindrome in S using $O(\sqrt{n})$ space and two passes over S.
For the first pass this algorithm runs *ApproxSqrt* $(S, \frac{1}{2})$ (meaning that $\varepsilon = 1/2$) and the variant of *ApproxSqrt* described in Lemma 5 simultaneously. The first pass returns ℓ_{max} (Lemma 5) if $\ell_{max} < \sqrt{n}$. Otherwise, the first pass (Theorem 1) returns for every palindrome $P[m]$, with $\ell(m) \geq \sqrt{n}$, an estimate satisfying $\ell(m) - \sqrt{n}/2 < \tilde{\ell}(m) \leq \ell(m)$ w.h.p..
The algorithm for the second pass is determined by the outcome of the first pass. For the case $\ell_{max} < \sqrt{n}$, it uses the sliding window to find all $P[m]$ with $\ell(m) = \ell_{max}$. If $\ell_{max} \geq \sqrt{n}$, then the first pass only returns an additive $\sqrt{n}/2$-approximation of the palindrome lengths. We define the *uncertain intervals* of $P[m]$ to be: $I_1(m) = S[m - \tilde{\ell}(m) - \sqrt{n}/2 + 1, m - \tilde{\ell}(m)]$ and $I_2(m) = S[m + \tilde{\ell}(m) + 1, m + \tilde{\ell}(m) + \sqrt{n}/2]$. The algorithm uses the length estimate calculated in the first pass to delete all R_S-entries (Step 3) which cannot be the longest palindromes. Similarly, the algorithm (Step 2) only keeps the middle entries of R_F-entries since these are the longest palindromes of their run (See Lemma 23). In the second pass, *Algorithm Exact* stores $I_1(m)$ for a palindrome $P[m]$ if it was not deleted. *Algorithm Exact* compares the symbols of $I_1(m)$ symbol by symbol to $I_2(m)$ until the first mismatch is found. Then the algorithm knows the exact length $\ell(m)$ and discards $I_1(m)$. The analysis will show, at any time the number of stored uncertain intervals is bounded by a constant.
First Pass Run the following two algorithms simultaneously:

1. *ApproxSqrt* $(S, 1/2)$. Let L be the returned list.
2. Variant of *ApproxSqrt* (See Lemma 5) which reports ℓ_{max} if $\ell_{max} < \sqrt{n}$.

Second Pass

- $\ell_{max} < \sqrt{n}$: Use a sliding window of size $2\sqrt{n}$ and maintain two fingerprints $F^R[i - \sqrt{n} - \ell_{max} + 1, i - \sqrt{n}]$, and $F^F[i - \sqrt{n} + 1, i - \sqrt{n} + \ell_{max}]$. Whenever these fingerprints match, report $P[i - \sqrt{n}]$.
- $\ell_{max} \geq \sqrt{n}$: In this case, the algorithm uses a preprocessing phase first.
  Preprocessing
  1. Set $\tilde{\ell}_{max} = \max\{\tilde{\ell}(m) \mid P[m]$ is stored in L as an R_F or an R_S entry$\}$.
  2. For every R_F-entry R_F in L with midpoints $m_1, \ldots, m_h$ remove R_F from L and add $R_s(m, i) = (m, \tilde{\ell}(m), F^F(1, m), F^R(1, m))$ to L, for $m \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}\}$. To do this, calculate $m_{\lfloor \frac{1+h}{2} \rfloor} = m_1 + (\lfloor \frac{1+h}{2} \rfloor - 1)(m_2 - m_1)$ and $m_{\lceil \frac{1+h}{2} \rceil} = m_1 + (\lceil \frac{1+h}{2} \rceil - 1)(m_2 - m_1)$. Retrieve $F^F(1, m)$ and $F^R(1, m)$ for $m \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}\}$ as shown in Lemma 19.
  3. Delete all R_S-entries $(m_k, \tilde{\ell}(m_k), F^F(1, m_k), F^R(1, m_k))$ with $\tilde{\ell}(m_k) \leq \tilde{\ell}_{max} - \sqrt{n}/2$ from L.
  4. For every palindrome $P[m] \in L$ set $I_1(m) := (m - \tilde{\ell}(m) - 1/2\sqrt{n}, m - \tilde{\ell}(m)]$ and set $finished(m) :=$ false.

  The resulting list is called L^*.
  String processing At iteration i the algorithm performs the following steps.
  1. Read $S[i]$. If there is a palindrome $P[m]$ such that $i \in I_1(m)$, then store $S[i]$.
  2. If there is a midpoint m such that $m + \tilde{\ell}(m) < i < m + \tilde{\ell}(m) + \frac{\sqrt{n}}{2}$, finished$(m) =$ false, and $S[m - (i - m) + 1] \neq S[i]$, then set finished$(m) :=$ true and $\ell(m) = i - m - 1$.
  3. If there is a palindrome $P[m]$ such that $i \geq \tilde{\ell}(m) + m + \frac{\sqrt{n}}{2}$, then discard $I_1(m)$.
  4. If $i = n$, then output $\ell(m)$ and m of all $P[m]$ in L^* with $\ell(m) = \ell_{max}$.

We analyse *Exact* in Section E of the appendix.

6 Algorithm ApproxLog

In this section, we present an algorithm which reports one of the longest palindromes and uses only logarithmic space. *ApproxLog* has a multiplicative error instead of an additive error term. Similar to *ApproxSqrt* we have special indices of S designated as checkpoints that we keep along with some constant size data in memory. The checkpoints are used to estimate the length of palindromes. However, this time checkpoints (and their data) are only stored for a limited time. Since we move from additive to multiplicative error we do not need checkpoints to be spread evenly in S. At iteration i, the number of checkpoints in any interval of fixed length decreases exponentially with distance to i. The algorithm stores a palindrome $P[m]$ (as an R_S-entry or R_{NF}-entry) until there is a checkpoint c such that $P[m]$ was checked unsuccessfully against c. A palindrome is stored in the lists belonging to the last checkpoint they with which is was checked successfully. In what follows we set $\delta \triangleq \sqrt{1+\varepsilon} - 1$ for the ease of notation. Every checkpoint c has an attribute called $level(c)$. It is used to determine the number of iterations the checkpoint data remains in the memory.
Memory invariants. After algorithm *ApproxLog* has processed $S[1, i-1]$ and before reading $S[i]$ it contains the following information:

1. Two *Master Fingerprints* up to index $i-1$, i.e., $F^F(1, i-1)$ and $F^R(1, i-1)$.
2. A list of checkpoints CL_{i-1}. For every $c \in \mathrm{CL}_{i-1}$ we have
   - $level(c)$ such that c is in CL_{i-1} iff $c \geq (i-1) - 2(1+\delta)^{level(c)}$.
   - $fingerprint(c) = F^R(1, c)$
   - a list L_c. It contains all palindromes which were successfully checked with c, but with no other checkpoint $c' < c$. The palindromes in L_c are either R_S-entries or R_{NF}-entries (See *Algorithm ApproxSqrt*).
3. The midpoint m^*_{i-1} and the length estimate $\tilde{\ell}(m^*_{i-1}, i-1)$ of the longest palindrome found so far.

The algorithm maintains the following property. If $P[m, i]$ was successfully checked with checkpoint c but with no other checkpoint $c' < c$, then the palindrome is stored in L_c. The elements in L_c are ordered in increasing order of their midpoint. The algorithm stores palindromes as R_S-entries and R_{NF}-entries. This time however, the length estimates are not maintained. Adding a palindrome to a current run works exactly (the length estimate is not calculated) as described in *Algorithm ApproxSqrt*.
Maintenance. At iteration i the algorithm performs the following steps.

1. Read S[i]. Update the *Master Fingerprints* to be $F^F(1, i)$ and $F^R(1, i)$.
2. For all $k \geq k_0 = {}^{\log(1/\delta)}/{}_{\log(1+\delta)}$(The algorithm does not maintain intervals of size 0.)
   a. If i is a multiple of $\lfloor \delta(1+\delta)^{k-2} \rfloor$, then add the checkpoint $c = i$ (along with the checkpoint data) to CL_i. Set $level(c) = k$, $fingerprint(c) = F^R(1, i)$ and $L_c = \emptyset$.
   b. If there exists a checkpoint c with $level(c) = k$ and $c < i - 2(1+\delta)^k$, then prepend L_c to $L_{c'}$ where $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \ and \ c'' > c\}$. Merge and create runs in L_c if necessary (Similar to step 5 of *ApproxSqrt*). Delete c and its data from CL_i.
3. For every checkpoint $c \in \mathrm{CL}_i$
   a. Let m_c be the midpoint of the first entry in L_c and $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \ and \ c'' < c\}$. If $i - m_c = m_c - c'$, then we *check* $P[m]$ against c' by doing the following:
      i. If the left side of m_c is the reverse of the right side of m_c (i.e., $F^R(c', m_c) = F^F(m_c, i)$) then move $P[m_c]$ from L_c to $L_{c'}$ by adding $P[m_c]$ to $L_{c'}$:
         A. If $|L_{c'}| \leq 1$, store $P[m_c]$ as a R_S-entry.

B. If $|L_{c'}| = 2$, create a run out of the R_S-entries stored in $L_{c'}$ and $P[m_c]$.
C. Otherwise, add $P[m_c]$ to the R_{NF}-entry in $L_{c'}$.

ii. If the left side of m_c is *not* the reverse of the right side of m_c, then remove m_c from L_c.

iii. If $i - m_c > \tilde{\ell}(m_i^*)$, then set $m_i^* = m_c$ and set $\tilde{\ell}(m_i^*) = i - m_c$.

4. If $i = n$, then report m_i^* and $\tilde{\ell}(m_i^*)$.

6.1 Analysis

ApproxLog relies heavily on the interaction of the following two ideas. The pattern of the checkpointing and the compression which is possible due to the properties of overlapping palindromes (Lemma 10). On the one hand the checkpoints are close enough so that the length estimates are accurate (Lemma 18). The closeness of the checkpoints ensures that palindromes which are stored at a checkpoint form a run (Lemma 17) and therefore can be stored in constant space. On the other hand the checkpoints are far enough apart so that the number of checkpoints and therefore the required space is logarithmic in n.
We start off with an observation to characterize the checkpointing. Step 2 of the algorithm creates a checkpoint pattern: Recall that the level of a checkpoint is determined when the checkpoint and its data are added to the memory. The checkpoints of every level have the same distance. A checkpoint (along with its data) is removed if its distance to i exceeds a threshold which depends on the level of the checkpoint. Note that one index of S can belong to different levels and might therefore be stored several times. The following observation follows from Step 2 of the algorithm.

▶ Observation 15. At iteration i, $\forall k \geq k_0 = {}^{\log(1/\delta)}/_{\log(1+\delta)}$: Let $C_{i,k} = \{c \in \mathrm{CL}_i \mid \mathit{level}(c) = k\}$. We have

1. $C_{i,k} \subseteq [i - 2(1+\delta)^k, i]$.
2. The distance between two consecutive checkpoints of $C_{i,k}$ is $\lfloor \delta(1+\delta)^{k-2} \rfloor$.
3. $|C_{i,k}| = \left\lceil \frac{2(1+\delta)^k}{\lfloor \delta(1+\delta)^{k-2} \rfloor} \right\rceil$.

This observation can be used to calculate the size the checkpoint data which the algorithm stores at any time. The proof can also be found in the appendix.

▶ **Lemma 16.** *At Iteration i of the algorithm the number of checkpoints is in $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$.*

The space bounds of Theorem 3 hold due to the following property of the checkpointing: If there are more than three palindromes stored in a list L_c for checkpoint c, then the palindromes form a run and can be stored in constant space as the following lemma shows.

▶ **Lemma 17.** *At iteration i, let $c \in CL_i$ be an arbitrary checkpoint. The list L_c can be stored in constant space.*

Proof. We fix an arbitrary $c \in \mathrm{CL}_i$. For the case that there are less than three palindromes belonging to L_c, they can be stored as R_S-entries in constant space. Therefore, we assume the case where there are at least three palindromes belonging to L_c and we show that they form a run. Let c' be the highest (index) checkpoint less than c, i.e., $c' = max\{c'' \mid c'' \in \mathrm{CL}_i \textit{ and } c'' < c\}$. We disregard the case that the index of c is 1. Let k be the minimum value such that $(1+\delta)^{k-1} < i - c \leq (1+\delta)^k$. Recall that L_c is the list of palindromes which the algorithm has successfully checked against c and not against c' yet. Let $P[m]$ be a palindrome in L_c. Since it was successfully checked against c we know that $i - m \geq m - c$. Similarly, since $P[m]$ was not checked against c' we have $i - m < m - c'$. Thus, for every $P[m]$ in L_c we have

$\frac{i+c'}{2} < m \leq \frac{i+c}{2}$. Therefore, all palindromes stored in L_c are in an interval of length less than $\frac{i+c}{2} - \frac{i+c'}{2} = \frac{c-c'}{2}$. If we show that $\ell(m) \geq \frac{c-c'}{2}$ for all $P[m]$ in L_c, then applying Lemma 10 with $\ell^* = \frac{c-c'}{2}$ on the palindromes in L_c implies that they are forming a run. The run can be stored in constant space in an R_{NF}-entry. Therefore, it remains to show that $\ell(m) \geq \frac{c-c'}{2}$. We first argue the following: $c - c' \underset{Obs.\ 15}{\leq} \delta(1+\delta)^{k-2} \underset{\delta\leq 1}{\leq} \frac{(1+\delta)^{k-1}}{2} \underset{Def.\ of\ k}{<} \frac{i-c}{2}$. Since $P[m]$ was successfully checked against c and since $m > \frac{i+c'}{2}$ we derive that $\ell(m) > \frac{i+c'}{2} - c$. Therefore, $\ell(m) > \frac{i+c'}{2} - c = \frac{i-c}{2} + \frac{c'-c}{2} \underset{(6.1)}{>} c - c' + \frac{c'-c}{2} = \frac{c-c'}{2}$. ◀

The following lemma shows that the checkpoints are sufficiently close in order to satisfy the multiplicative approximation.

▶ **Lemma 18.** ApproxLog *reports a midpoint m^* such that w.h.p.* $\frac{\ell_{max}}{(1+\varepsilon)} \leq \tilde{\ell}(m^*) \leq \ell_{max}$.

Proof. In step 4 of iteration i, *ApproxLog* reports the midpoint and length estimate of $P[m^*]$. We first argue that $\tilde{\ell}(m^*) \leq \ell(m^*) \leq \ell_{max}$. Let i' be the last time $\tilde{\ell}(m^*)$ was updated by step 3(a)iii of the algorithm. By the condition of step 3(a)iii, $S[c'+1, m^*]$ is the reverse of $S[m^*+1, i']$, where $c' = 2 \cdot m^* - i'$. Hence, we derive $\ell(m^*) \geq i' - m = \tilde{\ell}(m^*)$. Moreover, by the definition of ℓ_{max} we have $\ell_{max} \geq \ell(m^*)$.
We now argue $\frac{\ell_{max}}{(1+\varepsilon)} \leq \tilde{\ell}(m^*)$. Let $P[m_{max}]$ be a palindrome of maximum length, i.e., $\ell(m_{max}) = \ell_{max}$. Let k be an integer such that $(1+\delta)^{k-1} < \ell_{max} \leq (1+\delta)^k$. Consider $\tilde{\ell}(m^*)$ after the algorithm processed $S[1, i']$, where $i' = m_{max} + (1+\delta)^{k-1}$. By Observation 15, there is a checkpoint in interval $[i' - 2\cdot(1+\delta)^{k-3}, i' - 2\cdot(1+\delta)^{k-1} + \delta(1+\delta)^{k-3}]$. Let c denote this checkpoint. *ApproxLog* successfully checked $P[m_{max}]$ against this checkpoint and therefore the value $\tilde{\ell}(m^*)$ is set to at least $m_{max} - c$. We have $m_{max} - c \geq (1+\delta)^{k-1} - \delta(1+\delta)^{k-3}$. Thus, we have $\tilde{\ell}(m^*) \geq m_{max} - c \geq (1+\delta)^{k-1} - \delta(1+\delta)^{k-3}$. Therefore, $\frac{\ell_{max}}{\tilde{\ell}(m^*)} \leq \frac{(1+\delta)^k}{(1+\delta)^{k-1} - \delta(1+\delta)^{k-3}} = \frac{(1+\delta)^3}{(1+\delta)^2 - \delta} \leq (1+\delta)^2 = 1 + \varepsilon$. ◀

This concludes Theorem 3. A full proof can be found in the appendix.

References

1 Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(1–2):163 – 173, 1995.

2 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In Raffaele Giancarlo and Giovanni Manzini, editors, *Combinatorial Pattern Matching*, volume 6661 of *Lecture Notes in Computer Science*, pages 162–172. Springer Berlin Heidelberg, 2011.

3 Funda Ergun, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *Proceedings of the 13th international conference on Approximation, and 14 the International conference on Randomization, and combinatorial optimization: algorithms and techniques*, APPROX/RANDOM'10, pages 545–559, Berlin, Heidelberg, 2010. Springer-Verlag.

4 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, March 1987.

5 Glenn Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *J. ACM*, 22(3):346–351, July 1975.

6 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 315–323, Washington, DC, USA, 2009. IEEE Computer Society.

Appendix

A Useful fingerprint properties

▶ **Lemma 19.** *(Similar to Lemma 1 and Corollary 1 of [2]) Consider two substrings $S[i,k]$ and $S[k+1,j]$ and their concatenated string $S[i,j]$ where $1 \le i \le k \le j \le n$.*

- $F^F(i,j) = \left(F^F(i,k) + r^{k-i+1} \cdot F^F(k+1,j)\right)$ *mod* p.
- $F^F(k+1,j) = r^{-(k-i+1)}\left(F^F(i,j) - F^F(i,k)\right)$ *mod* p.
- $F^F(i,k) = \left(F^F(i,j) - r^{k-i+1} \cdot F^F(k+1,j)\right)$ *mod* p.

The authors of [2] show that, for appropriate choices of p and r, it is very unlikely that two different strings share the same fingerprint.

▶ **Lemma 20.** *(Theorem 1 of [2]) For two arbitrary strings s and s' with $s \ne s'$ the probability that $\phi^F(s) = \phi^F(s')$ is smaller than $1/n^4$.*

B Complementary Palindromes

▶ **Definition 21.** Let $f : \Sigma \to \Sigma$ be a function indicating a complement for each symbol in Σ. A string $S \in \Sigma^n$ with length n contains a *complementary palindrome* of length ℓ with midpoint $m \in \{\ell, \dots, n-\ell\}$ if $S[m-i+1] = f(S[m+i])$ for all $i \in \{1, \dots, \ell\}$.

The fingerprints can also be used for finding complementary palindromes: If one changes the forward *Master Fingerprints* to be $F_c^F(1,l) = \left(\sum_{i=1}^{l} f(S[i]) \cdot r^i\right) \mod p$ (as opposed to $F^F(1,l) = \left(\sum_{i=1}^{l} S[i] \cdot r^i\right) \mod p$) in all algorithms in this paper, then we obtain the following observation.

▶ Observation 22. All algorithms in this paper can be adjusted to recognize complementary palindromes with the same space and time complexity.

C *Simple ApproxSqrt*

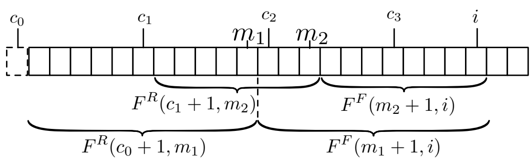

Figure 1 At iteration i two midpoints m_1 and m_2 are checked. Corresponding substrings are denoted by brackets. Note, the distance from c_0 to m_1 equals the distance from m_1 to i. Similarly, the distance from c_1 to m_2 equals the distance from m_2 to i.

Proof of Lemma 6. Fix an arbitrary palindrome $P[m]$. First we assume $\ell(m) < \sqrt{n}$. Then *Simple ApproxSqrt* reports m and $\tilde{\ell}(m)$ in step 4 of iteration $m + \sqrt{n}$ which is the iteration where the entire palindrome is in the sliding window. Furthermore, in the step 4, the algorithm checks for all $j \in \{\lfloor \varepsilon\sqrt{n} \rfloor, 2 \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \dots, r \cdot \lfloor \varepsilon\sqrt{n} \rfloor, \sqrt{n}\}$, where r is the maximum integer s.t. $r \cdot \lfloor \varepsilon\sqrt{n} \rfloor < \sqrt{n}$, if $F^R(m-j+1, m) = F^F(m+1, m+j)$, then sets $\tilde{\ell}(m,i) = j$.

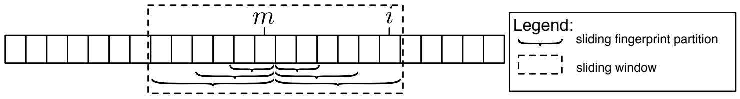

Figure 2 Illustration of the *Fingerprint Pairs* after iteration i of algorithm with $\sqrt{n} = 6$, $\varepsilon = 1/3$, and $m = i - \sqrt{n}$.

Let j_m be the maximum j such that $F^R(m-j+1, m) = F^F(m+1, m+j)$ is satisfied at iteration $m + \sqrt{n}$. Then $P[m]$ covers the *Fingerprint Pair* with length j_m but does not cover the *Fingerprint Pair* with length $j_m + \lfloor \varepsilon\sqrt{n} \rfloor$. Since *Simple ApproxSqrt* sets $\tilde{\ell}(m) = j_m$ we have $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \leq \ell(m)$.
Now we assume $\ell(m) \geq \sqrt{n}$. Step 5 of iteration $m + \sqrt{n}$ adds $R_S(m, i)$ to L_{i-1}. We show for every $i \geq m + \sqrt{n}$ that the following holds: After *Simple ApproxSqrt* read $S[1, i]$ (iteration i) we have $\ell(m, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m, i) \leq \ell(m, i)$. We first show the first inequality and afterwards the second. Define $i' \leq i$ to be the last iteration where the algorithm updated $\tilde{\ell}(m, i')$ in Step 6, i.e., it sets $\tilde{\ell}(m, i') = \ell(m, i') = i' - m$.

- $\ell(m, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m, i)$: We first show $\ell(m, i) < i' + \varepsilon\sqrt{n} - m$ by distinguishing between two cases:
  1. $i' > i - \varepsilon\sqrt{n}$: By definition of $\ell(m, i)$, we have $\ell(m, i) \leq i - m$. And thus $\ell(m, i) < i' + \varepsilon\sqrt{n} - m$.
  2. $i' \leq i - \varepsilon\sqrt{n}$: Since the estimate of m was updated at iteration i' we know that there is a checkpoint at index $2m - i'$ and therefore we know that due to step 3 there is a checkpoint at index $2m - (i' + \lfloor \varepsilon\sqrt{n} \rfloor)$. Since i' is the last iteration where the estimate of $P[m]$ was updated we infer that $S[2m - (i' + \lfloor \varepsilon\sqrt{n} \rfloor), m]$ was not the reverse of $S[m + 1, i' + \lfloor \varepsilon\sqrt{n} \rfloor]$. Hence, $\ell(m, i) < i' + \lfloor \varepsilon\sqrt{n} \rfloor - m \leq i' + \varepsilon\sqrt{n} - m$.

  With $\ell(m, i) < i' + \varepsilon\sqrt{n} - m$ we have

  $$\ell(m, i) < i' + \varepsilon\sqrt{n} - m = \ell(m, i') + \varepsilon\sqrt{n} = \tilde{\ell}(m, i') + \varepsilon\sqrt{n} = \tilde{\ell}(m, i) + \varepsilon\sqrt{n}.$$

  The last equality holds since i' was the last index where $\tilde{\ell}(m, i)$ was updated.
- $\tilde{\ell}(m, i) \leq \ell(m, i)$: Whenever $\tilde{\ell}(m, i)$ is updated to $\ell(m, i')$ by the algorithm, this means that $F^F(m+1, i') = F^R(2m - i', m)$ and since we assume that fingerprints do not fail we have that $S[m - \tilde{\ell}(m, i') + 1, m]$ is the reverse of $S[m + 1, m + \tilde{\ell}(m, i')]$. It follows that $\tilde{\ell}(m, i) = \tilde{\ell}(m, i')$ and $\ell(m, i) \geq \ell(m, i')$.

Furthermore, step 7 reports L_n at iteration n which includes m and $\tilde{\ell}(m)$. ◀

Simple ApproxSqrt requires linear space in the worst-case. The algorithm stores a list of all $\sqrt{n}$-palindromes For an S, containing a linear number of $\sqrt{n}$-palindromes. *Simple ApproxSqrt* requires linear worst-case space. As an example for such a stream consider $S = a^n$ with $a \in \Sigma$. All indices in the interval $[\sqrt{n}, n - \sqrt{n}]$ are midpoints of $\sqrt{n}$-palindromes.

D ApproxSqrt

Proof of Corollary 11. We prove this by induction. Suppose j_0 is the highest index where $m_{j_0} < m_1 + \ell^*$. By Definition 7, we have $j_0 \geq 3$. We start by proving the induction basis. By Lemma 10, the claim holds for $m_1, m_2, \ldots, m_{j_0}$, i.e., they are equally spaced and $S[m_1 + 1, m_{j_0}]$ is a prefix of $ww^R \ldots ww^R$. For the inductive step we assume that the claim holds for m_{j-1}. Consider the midpoints m_{j-1}, m_j, m_{j+1}. Since $m_{j+1} - m_{j-1} \leq \ell^*$, Lemma 10 shows that those midpoints fulfill the claimed structure. ◀

Proof of Lemma 12. We prove the first case where m_j is in the first half, i.e., $j < \frac{h+1}{2}$. The other case is similar. By Corollary 11, $S[m_1, m_h]$ is of the form ww^Rww^R... and $m_j = m_1 + (j-1) \cdot |w|$, where w is $S[m_1+1, m_2]$. Define m_0 to be the index $m_1 - |w|$. Since $\ell(m_1, i) \geq \ell^* \geq |w|$ we have $S[m_0+1, m_1] = w^R$.
By Corollary 11, we have that $S[m_0+1, m_j]^R = S[m_j+1, m_{2j}]$. This implies $\ell(m_j, i) \geq j \cdot |w|$. Define k to be $\ell(m_j, i) - j \cdot |w|$. We show that $k = \ell(m_0, i)$. By definition, k is the length of the longest suffix of $S[1, m_0]$ which is the reverse of the prefix of $S[m_{2j}+1, n]$. Corollary 11 shows that $S[m_{2j}+1, m_{2j}+\ell^*]$ is equal to $S[m_0+1, m_0+\ell^*]$ as both are prefixes of w^Rww^R.... Therefore, k is also the same as the length of the longest suffix of $S[1, m_0]$ which is the reverse of the prefix of $S[m_0+1, m_0+\ell^*]$, i.e., $k = max\{k' | S[m_0 - k' + 1, m_0]^R = S[m_0+1, m_0+k']\} = \ell(m_0, i)$. Thus, $\ell(m_j, i) = \ell(m_0, i) + j \cdot |w|$. ◀

Proof of Lemma 13. Fix an index j. We prove that we can retrieve $R_S(m_j, i)$ out of the R_F-entry representation. Corollary 11 gives a formula to retrieve m_j from the corresponding R_F-entry. Formally, $m_j = m_1 + (j-1) \cdot (m_1 - m_2)$.
Corollary 11 shows that $S[m_1+1, m_h]$ follows the structure $ww^Rw^R \cdots$ where $w = S[m_1 + 1, m_2]$.
This structure allows us to retrieve $F^F(1, m_j), F^R(1, m_j)$, since we have $F^F(1, m_j) = \phi^F(S[1, m_1]ww^Rww^R \cdots)$.
We know argue that the length estimates have the same accuracy as R_S-entries. Note that the proof of Lemma 6 shows that after iteration i and any $R_S(m, i)$ we have $\ell(m, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m, i) \leq \ell(m, i)$. We show that one can retrieve the length estimate for palindromes which are not stored explicitly by using the following equation. The equation is motived by Lemma 12.

$$\tilde{\ell}(m_j, i) = \begin{cases} \tilde{\ell}(m_1, i) + (j-1) \cdot (m_2 - m_1) & j < \frac{h+1}{2} \\ \tilde{\ell}(m_h, i) + (h-j) \cdot (m_2 - m_1) & j > \frac{h+1}{2} \end{cases}.$$

Let i' be the index where R_F was finished. We distinguish among three cases:

1. $m_j = m_1$: Since the algorithm treats $P[m_j]$ as an R_S-entry in terms of comparisons, $\ell(m_j, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i) \leq \ell(m_j, i)$ holds.
2. $m_j \in \{m_{\lfloor (h+1)/2 \rfloor}, m_{\lceil (h+1)/2 \rceil}, m_h\}$: At index i' *ApproxSqrt* executes step 5(a)ii and one can verify that $\ell(m_j, i') - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i') \leq \ell(m_j, i')$ holds. For all $i \geq i'$ palindrome $P[m_j]$ is treated as an R_S-entry in terms of comparisons. Thus, $\ell(m_j, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i) \leq \ell(m_j, i)$ holds.
3. Otherwise we assume WLOG. $1 < j < \lfloor \frac{h+1}{2} \rfloor$. Lemma 12 shows that $\ell(m_j, i) - \tilde{\ell}(m_j, i) = \ell(m_1, i) - \tilde{\ell}(m_1, i)$. We know $0 \leq \ell(m_1, i) - \tilde{\ell}(m_1, i) < \varepsilon\sqrt{n}$. Thus, $\ell(m_j, i) - \varepsilon\sqrt{n} < \tilde{\ell}(m_j, i) \leq \ell(m_j, i)$.

◀

Proof of Theorem 1. Similar to other proofs in this paper we assume that fingerprints do not fail as we take less than n^2 fingerprints and by Lemma 20, the probability that a fingerprint fails is at most $1/(n^4)$. Thus, by applying the union bound the probability that no fingerprint fails is at least $1 - n^{-2}$.
Correctness: Fix an arbitrary palindrome $P[m]$. For the case $\ell(m) < \sqrt{n}$ there is no difference between *Simple ApproxSqrt* and *ApproxSqrt*, so the correctness follows from Lemma 6. In the following, we assume $\ell(m) \geq \sqrt{n}$. Firstly, we argue that $R_S(m, n)$ is stored in $\hat{L}_n$. At index $i = m + \sqrt{n}$, *ApproxSqrt* adds $P[m]$ to $\hat{L}_i$. The algorithm does this by using the longest sliding fingerprint pair which guarantees that if $S[i - 2\sqrt{n}+1, i - \sqrt{n}]$ is the reverse of $S[i - \sqrt{n}+1, i]$, then the fingerprints of sliding window are equal. Moreover, a palindrome

is never removed from $\hat{L}_i$ for $1 \le i \le n$. Additionally, Lemma 13 shows how to retrieve the midpoint. Hence, $R_S(m,n)$ is stored in $\hat{L}_n$.
We now argue $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \le \ell(m)$. Palindromes are stored in an R_S-entry, R_F-entry and R_{NF}-entry. Since we are only interested in the estimate $\tilde{\ell}(m)$ after the n^{th} iteration of *Simple ApproxSqrt* and since the algorithm finishes an R_{NF}-entry at iteration n, we know that there are no R_{NF}-entries at after iteration n.

1. $R_S(m,n)$ is stored as an R_S-entry. Since R_S-entries are treated in the same way as in *Simple ApproxSqrt*, $\ell(m) - \varepsilon\sqrt{n} < \tilde{\ell}(m) \le \ell(m)$ holds by Lemma 6.
2. $R_S(m,n)$ is stored in the R_F-entry R_F. Then Lemma 13 shows the correctness.

Furthermore, the algorithm reports $\hat{L}_n$ step 7 of iteration n.
Space: The number of checkpoints equals $\lfloor n/\lfloor\varepsilon\sqrt{n}\rfloor\rfloor \le 2n/\varepsilon\sqrt{n} = O(\sqrt{n}/\varepsilon)$, since $\varepsilon \ge 1/\sqrt{n}$. Moreover, there are $O(n/\varepsilon)$ *Fingerprint Pairs* which can be stored in $O(n/\varepsilon)$ space. The sliding window requires $2\sqrt{n}$ space. The space required to store the information of all $\sqrt{n}$-palindromes is bounded by $O(\sqrt{n})$: By Observation 14, the number of R_S-entries and *Compressed Runs* in an interval of length $\sqrt{n}$ is bounded by a constant. Each *Compressed Run* and each R_S-entry can be stored in constant space. Thus, in any interval of length $\sqrt{n}$ we only need constant space and thus altogether $O(\sqrt{n})$ space for storing the information of palindromes.
Running time: The running time of the algorithm is determined by the number of comparisons done at lines 6 and 4. First we bound the number of comparisons corresponding to line 6. For all $\sqrt{n}$-palindromes we bound the total number of comparisons by $O(\frac{n}{\varepsilon})$: The *ApproxSqrt* checks only explicitly stored palindromes with checkpoints and therefore with R_S-entries and at most 4 midpoints per *Compressed Run*. As shown in Observation 14, there is at most a constant number of explicitly stored midpoints in every interval of length $\sqrt{n}$. In total, we have $O(\sqrt{n})$ explicitly stored midpoints and $O(\sqrt{n}/\varepsilon)$ fingerprints of checkpoints. We only check each palindrome at most once with each checkpoint [5]. Hence, the total number of comparisons is in order of $O(n/\varepsilon)$. Now, we bound the running time corresponding to Step 4. This step has two functions: There are $O(1/\varepsilon)$ *Fingerprint Pairs* which are updated every iteration. This takes $O(1/\varepsilon)$ time. Additionally, the middle of the sliding window is checked with at most $O(1/\varepsilon)$ *Fingerprint Pairs*. Thus, the time for Step 4 of the algorithm is bounded by $O(n/\varepsilon)$. ◀

E *Algorithm Exact*

The analysis of *Algorithm Exact* is based on the observation that, after removing palindromes which are definitely shorter then the longest palindrome, at any time the number of stored uncertain intervals is bounded by a constant. The following Lemma shows that only the palindromes in the middle are strictly longer than the other palindromes of the run. This allows us to remove all palindromes which are not in the middle of the run. The techniques used in the lemma are very similar to the ideas used in Lemma 12. Let $\hat{L}_n$ be the list after the first pass.

▶ **Lemma 23.** *Let $m_1, m_2, m_3, ..., m_h$ be midpoints of a maximal ℓ^*-run in S. For every $j \in \{1,\ldots,h\} \setminus \{\lfloor(h+1)/2\rfloor, \lceil(h+1)/2\rceil\}$,*

$$\ell(m_j) < max\{\ell(m_{\lfloor(h+1)/2\rfloor}), \ell(m_{\lceil(h+1)/2\rceil})\}.$$

[5] We can use an additional queue to store the index where the algorithm needs to *check* a checkpoint with a palindrome.

Proof. If h is even, the claim follows from Lemma 12. Therefore, we assume $h = 2d-1$ which means that $m_{\lfloor (h+1)/2 \rfloor} = m_{\lceil (h+1)/2 \rceil} = m_d$. Hence, we have to show that $\ell(m_j) < \ell(m_d)$. Define w exactly as it is defined in Lemma 12 to be $S[m_1+1, m_2]$. Note that $S[m_{h-1}+1, m_h] = w^R$. We need two claims:

1. $\ell(m_d) \geq (d-1)|w| + min\{\ell(m_1), \ell(m_h)\}$
   Proof: Suppose $\ell(m_d) < (d-1)|w| + min\{\ell(m_1), \ell(m_h)\}$. We know that $S[m_1+1, m_d]$ is the reverse of $S[m_d+1, m_h]$. Therefore, $\ell(m_d) = (d-1)|w| + k$ where k is the length of the longest suffix of $S[1, m_1]$ which is the reverse of the prefix of $S[m_h+1, n]$. Formally, $k = max\{k' | S[m_1 - k' + 1, m_1]^R = S[m_h+1, m_h+k']\}$.
   Define $\ell' \triangleq min\{\ell(m_1), \ell(m_h)\}$. Thus, it suffices to show that $k \geq min\{\ell(m_1), \ell(m_h)\} = \ell'$. Observe, $\ell' < \ell^* + |w|$ since otherwise $m_1 - |w|$ or $m_h + |w|$ would be a part of the run. Since m_1 is the midpoint of a palindrome of length $\ell' < \ell^* + |w|$, by Corollary 11, the left side of m_1, i.e., $S[m_1 - \ell' + 1, m_1]$ is a suffix of length ℓ' of $ww^R \dots ww^R$. Similarly, $S[m_h+1, m_h+\ell']$ is a prefix of length ℓ' of $ww^R \dots ww^R$. These two facts imply that $S[m_1 - \ell' + 1, m_1]$ is the reverse of $S[m_h+1, m_h+\ell']$ and thus $k \geq \ell'$. ☐
2. $|\ell(m_h) - \ell(m_1)| < |w|$
   Proof: WLOG., let $\ell(m_h) \geq \ell(m_1)$, then suppose $\ell(m_h) - \ell(m_1) \geq |w|$. This implies that $\ell(m_h) \geq |w| + \ell(m_1) \geq |w| + \ell^*$. By Lemma 11, we derive that the run is not maximal, i.e., there is a midpoint of a palindrome with length of ℓ^* at index $m_h + |w|$. A contradiction. ☐

Using these properties we claim that $\ell(m_d) > \ell(m_j)$:

1. For $j < d : \ell(m_d) \underset{Property\ 1}{\geq} (d-1)|w| + min\{\ell(m_1), \ell(m_h)\} \underset{Property\ 2}{>} (d-1)|w| + \ell(m_1) - |w| = (d-2)|w| + \ell(m_1) - |w| \geq (j-1)|w| + \ell(m_1) \underset{Lemma\ 12}{=} \ell(m_j)$.
2. For $j > d : \ell(m_d) \underset{Property\ 1}{\geq} (d-1)|w| + min\{\ell(m_1), \ell(m_h)\} \underset{Property\ 2}{>} (d-1)|w| + \ell(m_h) - |w| = (d-2)|w| + \ell(m_h) - |w| \geq (j-1)|w| + \ell(m_h) \underset{Lemma\ 12.}{=} \ell(m_j)$.

This yields the claim. ◀

We conclude this section by proving Theorem 2 covering the correctness of the algorithm as well as the claimed space and time bounds. We say a palindrome with midpoint m *covers* an index i if $|m-i| \leq \ell(m)$.

Proof of Theorem 2. In this proof, similar to Theorem 1, we assume that the fingerprints do not fail w.h.p. .
For the case $\ell_{max} < \sqrt{n}$ it is easy to see that the algorithm satisfies the theorem.
Therefore, we assume $\ell_{max} \geq \sqrt{n}$.
Correctness: After the first pass we know that due to Theorem 1 all $\sqrt{n}$-palindromes are in L. The algorithm removes some of those palindromes and we argue that a palindrome which is removed from L cannot be the longest palindrome. A palindrome removed in

- Step 2 is, by Lemma 23, strictly shorter than the palindromes of the middle of the run from which it was removed.
- Step 3 with midpoint m has a length which is bounded by $\tilde{\ell}_{max} - \tilde{\ell}(m) \geq \sqrt{n}/2$. We derive $\ell_{max} \geq \tilde{\ell}_{max} \geq \sqrt{n}/2 + \tilde{\ell}(m) > \ell(m)$ where the last inequality follows from $\ell(m) - \sqrt{n}/2 < \tilde{\ell}(m) \leq \ell(m)$.

Therefore, all longest palindromes are in L^*. Furthermore, the exact length of $P[m]$ is determined at iteration $m+\ell(m)+1$ since this is the first iteration where $S[m-(\ell(m)+1)+1] \neq S[m+\ell(m)+1]$. In Step 2, the algorithm sets the exact length $\ell(m)$. If $\ell(m) = \ell_{max}$, then

the algorithm reports m and ℓ_{max} in step 4 of iteration n.
Space: For every palindrome we have to store at most one uncertain interval. At iteration i, the number of uncertain intervals we need to store equals the number of palindromes which cover index i. We prove in the following that this is bounded by 4. We assume ε to be $1/2$ and $\ell_{max} \geq \sqrt{n}$. Define $\tilde{\ell}_{min}$ to be the length of the palindrome in L^* for which the estimate is minimal. All palindromes in L^* have a length of at least $\sqrt{n}$, thus $\tilde{\ell}_{min} \geq \sqrt{n}$. We define the following intervals:

- $\mathcal{I}_1 = (i - \tilde{\ell}_{min} - \sqrt{n}, i - \tilde{\ell}_{min}]$
- $\mathcal{I}_2 = (i - \tilde{\ell}_{min}, i]$

Recall that the algorithm removes all palindromes which have a length of at most $\tilde{\ell}_{max} - \sqrt{n}/2$ and thus $\tilde{\ell}_{min} \geq \tilde{\ell}_{max} - \sqrt{n}/2$. Additionally, we know by Theorem 1 that $\ell_{max} - \tilde{\ell}_{max} < \sqrt{n}/2$. We derive: $\ell_{max} - \tilde{\ell}_{min} \leq \ell_{max} - \tilde{\ell}_{max} + \sqrt{n}/2 < \sqrt{n}/2 + \sqrt{n}/2 = \sqrt{n}$ and therefore $\ell_{max} < \sqrt{n} + \tilde{\ell}_{min}$. Hence, there is no palindrome which covers i and has a midpoint outside of the intervals $\mathcal{I}_1$ and $\mathcal{I}_2$. It remains to argue that the number palindromes which are centered in $\mathcal{I}_1$ and $\mathcal{I}_2$ and stored in L^* is bounded by four: Suppose there were at least four palindromes in $\mathcal{I}_1$ and $\mathcal{I}_2$ which cover i. Lemma 10 shows that in any interval of length $\tilde{\ell}_{min}$ either the number of palindromes is bounded by two or they form an $\tilde{\ell}_{min}$-run. Thus there has to be at least one $\tilde{\ell}_{min}$-run. Recall that step 2 keeps for all R_F-entries only the midpoints in the middle of the run. The first pass does not create R_F-entries for all runs where the difference between two consecutive midpoints is more than $\sqrt{n}/2$ (See Definition 8 and step 5(a)i of *ApproxSqrt*). Thus, there is a run where the distance between two consecutive midpoints is greater than $\sqrt{n}/2$. By Lemma 12, the difference between $\ell(m)$ for distinct midpoints m of one side of this run is greater than $\sqrt{n}/2$. Since the checkpoints are equally spaced with consecutive distance of $\sqrt{n}/2$, the difference between $\tilde{\ell}(m)$ for distinct midpoints m of one side of this run is greater than $\sqrt{n}/2$ as well. This means that just the palindrome(s) in the middle of the run, say $P[m]$, satisfy the constraint $\tilde{\ell}(m) \geq \tilde{\ell}_{max} - \sqrt{n}/2$ and therefore the rest of the palindromes of this run would have been deleted. A contradiction. Thus, there are at most two palindromes in both intervals and thus in total at most four palindromes and four uncertain intervals. This yields the space bound of $O(\sqrt{n})$.
Running time: As shown in Theorem 1 and Lemma 5 the running time is in $O(n)$. The preprocessing and the last step of the second pass can be done in $O(n)$. If the first pass returned an $\ell_{max} < \sqrt{n}$ then the running time of the second pass is trivially in $O(n)$. Suppose $\ell_{max} \geq \sqrt{n}$. The preprocessing and the last step of *Algorithm Exact* can be done in $O(n)$ time and they are only executed once. The remaining operations can be done in constant time per symbol. This yields the time bound of $O(n)$. ◀

Proof of Lemma 16. The distance between consecutive checkpoints of level k is $\lfloor \delta(1+\delta)^{k-2} \rfloor$ and thus the number of checkpoints per stage is bounded by

$$\left\lceil \frac{2(1+\delta)^k}{\lfloor \delta(1+\delta)^{k-2} \rfloor} \right\rceil \leq \left\lceil \frac{4(1+\delta)^k}{\delta(1+\delta)^{k-2}} \right\rceil \leq \frac{4(1+\delta)^2}{\delta} + 1 = \frac{4(1+\varepsilon)}{\sqrt{1+\varepsilon}-1} + 1$$

$$= \frac{4(1+\varepsilon)(\sqrt{1+\varepsilon}+1)}{\varepsilon} + 1 \leq \frac{24}{\varepsilon} + 1$$

where the first inequality comes from the fact that $k \geq k_0$. The number of levels is $\lceil \log_{1+\delta}(n) \rceil = \lceil 2\log_{1+\varepsilon}(n) \rceil$ and the number of checkpoints is therefore bounded by $(2\log_{1+\varepsilon}(n)+1)(24/\varepsilon + 1) = O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$. The required space to store a checkpoint along with its data is constant. ◀

Proof of Theorem 3. In this proof, similar to Theorems 1 and 2, we assume that the fingerprints do not fail w.h.p. as *ApproxLog*, similar to *ApproxSqrt*, does not take more than n^2 fingerprints during the processing of any input of length n.
Correctness: The correctness of the algorithm follows from Lemma 18. It remains to argue that the space and time are not exceeded.
Space: The space required by *ApproxLog* is dominated by space needed to store the palindromes (corresponding midpoints and fingerprints) in L_c for all $c \in \mathrm{CL}_i$. Lemma 17 shows that for any $c \in \mathrm{CL}_i$ the list L_c can be stored in constant space. Furthermore, Lemma 16 shows that there are $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$ elements in CL_i.
Running time: The running time is determined by step 2b and 3. The algorithm goes in every iteration through all checkpoints in CL_i which has $O\left(\frac{\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$ elements as Lemma 16 shows. For each checkpoint the steps (2b and 3) take only constant time. Thus, the required time to process the whole input is $O\left(\frac{n\log(n)}{\varepsilon \log(1+\varepsilon)}\right)$. ◀

F Variants of *Algorithm ApproxSqrt*

In this section we present two variants of *ApproxSqrt*. The first variant is similar to *ApproxSqrt*, but instead of reporting all palindromes it reports a palindromes $P[m]$ iff $\ell(m) > \ell_{max} - \varepsilon\sqrt{n}$ assuming that $\ell_{max} \geq \sqrt{n}$. If the algorithm is run on an input where $\ell_{max} < \sqrt{n}$, then the algorithm realizes this and does not report any palindromes. It require $\omega(\frac{\sqrt{n}}{\varepsilon})$ space to output all palindromes of size ℓ_{max} if $\ell_{max} < \sqrt{n}$. The described variant can be implemented in the following way: Run *ApproxSqrt* and before returning the final list L_n trim the list L_n by removing all short palindromes. For details see the preprocessing of *Algorithm Exact* (which is introduced in Section 5).This leads to Observation 4.
The second variant reports one of the longest palindromes and the precise ℓ_{max} if $\ell_{max} < \sqrt{n}$. In case $\ell_{max} \geq \sqrt{n}$, the algorithm detects this, but does not report the precise ℓ_{max}. The reason is that we can store a small palindrome in our memory and it is not possible to report $\ell_{max}(\geq \sqrt{n})$ in space $o(n)$. In following we present the proof of Lemma 5.

Proof of Lemma 5. The algorithm uses a sliding window of size $2\sqrt{n}$. At iteration i, the middle of the sliding window is $m = i - \sqrt{n}$. Let $\ell_{i-1} < \sqrt{n}$ be the length of the longest palindrome at index $i-1$. The algorithm does the following.
Algorithm: Initialize $k = 0$. At iteration i compare $F^R(m-k+1, m)$ with $F^F(m+1, m+k)$ for $k = \ell_{i-1}+1$ to $\sqrt{n}$ until they are unequal: Set ℓ_i to be the highest value for k such that $F^R(m-k+1, m)$ equals $F^F(m+1, m+k)$. If $\ell_i > \ell_{i-1}$, then set $longest = P[m]$. If ℓ_i is smaller than ℓ_{i-1}, set $\ell_i = \ell_{i-1}$. After iteration n output ℓ_{max} and *longest*.
Correctness: For $k \in \mathbb{N}$ if $S[i-k+1, i]$ is the reverse of $S[i+1, i+k]$, then the fingerprints are equal. Let m be the first palindrome in S with $\ell(m) = \ell_{max}$, then at iteration $m + \sqrt{n}$ the algorithm compares the fingerprint $F^R(i - (\ell_{max}+1)+1, i)$ with $F^F(i+1, i+\ell_{max}+1)$ and sets $\ell_m = \ell_{max}$. The value of ℓ_m is not changed afterwards.
Space: The algorithm stores two fingerprints, the longest palindrome found so far, and the sliding window of size $2\sqrt{n}$ which results in $O(\sqrt{n})$ space.
Running time: Sliding the two fingerprints takes $O(n)$ time in total. The fingerprints are extended at most $\sqrt{n}$ times. Storing the longest palindrome takes at most $O(\sqrt{n})$ time and this is done at most $\sqrt{n}$ times. ◀