Fast Rational Search via Stern-Brocot Tree

Connor Weyers
Department of Computer Science, Indiana University Bloomington, IN, USA

N. V. Vinodchandran
School of Computing, University of Nebraska-Lincoln, NE, USA

Abstract

We revisit the problem of rational search: given an unknown rational number $\alpha = \frac{a}{b} \in (0, \infty)$ with $a, b \leq n$, the goal is to identify α using comparison queries of the form "$\beta \leq \alpha$?". The problem has been studied several decades ago and optimal query algorithms are known. We present an algorithm for rational search based on a compressed traversal of the Stern–Brocot tree, which appeared to have been overlooked in the literature. This approach also naturally extends to two related problems that, to the best of our knowledge, have not been previously addressed: (i) unbounded rational search, where the bound n is unknown, and (ii) computing the *best* (in a precise sense) rational approximation of an *unknown* real number using only comparison queries. While the algorithm is simple and natural, one of our main contributions is its analysis: we give an upper and lower bound on its worst-case query complexity.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms; Theory of computation → Data structures design and analysis; Theory of computation → Sorting and searching

Keywords and phrases Rational number search, Continued fractions, Stern-Brocot tree, Rational approximation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.35

Related Version *Previous Version*: `https://arxiv.org/abs/2512.18036`

Funding *Connor Weyers*: Work done while at UNL School of Computing and supported in part by a UNL Grand Challenges Catalyst Competition Grant.
N. V. Vinodchandran: Supported in part by NSF grants 2342244, 2413848 and a UNL Grand Challenges Catalyst Competition Grant.

1 Introduction

Rational search is the problem of identifying an unknown fraction from a set of fractions. In the *bounded* version of the problem, the input is an unknown rational number $\alpha = \frac{a}{b} \in (0, \infty)$ with $a, b \leq n$, and the goal is to determine α using comparison queries of the form "$\beta \leq \alpha$?". The *unbounded* version assumes no knowledge of n: α could be any positive rational number. The efficiency of a rational search algorithm is typically measured by the number of such queries needed to uniquely identify the target fraction.

This problem was first studied several decades ago, and algorithms for the bounded version only were proposed in [7, 8], culminating in a query-optimal (up to an additive constant) algorithm by Kwek and Mehlhorn in 2003 [6]. The Kwek–Mehlhorn algorithm requires at most $2 \log_2 n + O(1)$ queries, which is optimal up to an additive constant, since the number of distinct fractions in $[0, 1]$ with denominator $\leq n$ is $\Omega(n^2)$; see, for example, [4].

In this work we present an algorithm for the rational search problem based on a compressed traversal of the classical *Stern–Brocot tree*. While the underlying idea is natural and may have been implicitly known to other researchers we were not aware of published prior work that formulates it explicitly for rational search or presents a fine analysis of its performance. Indeed, after our preprint was posted on arXiv, we were informed of a Chinese-language

blog post that independently presents a similar algorithm [10]. The algorithm we present is simple to describe and straightforward to implement; however, establishing its guarantees requires a more delicate analysis and that is one of the main contribution of the paper. In particular, analyzing its behavior requires understanding how mediant expansions evolve during the downward traversal of the tree. We prove an upper bound of $2.5849 \log_2 n + 2$ comparison queries in the worst case (where 2.5849 is an approximation to the exact constant $16/\log_2 73$ arising in the analysis). We also construct instances for which the algorithm requires $2.4189 \log_2 n$ comparisons (where 2.4189 approximates the irrational $8/\log_2(5+2\sqrt{6})$). Finding the exact worst-case complexity of the algorithm is open. Based on experiments we conducted, we conjecture that the correct upper bound is that given by our lower bound.

The algorithm is not worst-case query-optimal for the bounded rational search. However, an advantage is that our approach naturally solves the most general unbounded rational search problem of searching for any unknown positive rational number in $\mathbb{Q}$ using comparison queries without any additional knowledge. Moreover, it can be easily adapted to solve the problem of finding the *best* rational approximation of an *unknown* real number using only comparison queries, discussed next. No prior work appears to have addressed either of these two variations.

A problem related to the rational search problem is that of finding the *best* rational approximation to a given real (or irrational) number. Specifically, given a real number r and an approximation parameter δ, find the (unique) fraction $\frac{a}{b}$ so that $\frac{a}{b} \in [r \pm \delta]$ with the *smallest possible denominator* b. This problem is non-trivial even when r is given explicitly. In fact, Forišek proposed an efficient algorithm for this problem, when r is given, with the number of arithmetic operations linear in the input representation [3]. To the best of our knowledge, no efficient query-based algorithm is known for the case when the target real number is *unknown* and must be identified solely through comparison queries. An easy modification of our algorithm also solves this best rational approximation problem for an *unknown* real number. While our algorithm is similar to Forišek's approach, it relies on a combination of exponential search and binary search to navigate the Stern-Brocot tree efficiently only with comparison queries.

We conducted an experimental evaluation comparing our algorithm with the Kwek–Mehlhorn algorithm. The results show that, although the improvement is not large, our algorithm requires fewer queries on average[1] when given a denominator's upper bound n up to 10^{40}. We also implemented the rational-approximation variant of our algorithm and evaluated it on the real numbers $\pi, e, \sqrt{2}$, and $\sqrt{5}$, treating each as unknown and using only comparison queries. For various settings of the approximation parameter, we report the resulting approximations along with the corresponding query counts and runtimes.

2 Stern–Brocot Tree

The *Stern–Brocot tree*, introduced independently by Stern [9] and Brocot [2] in the 19th century, is an infinite binary tree structure that systematically enumerates all positive rational numbers in reduced form, each exactly once. The tree is constructed using the *mediant* operation, a simple binary operation on fractions. Given two fractions $\frac{a}{b}$ and $\frac{c}{d}$, the *mediant* is defined as: $\frac{a+c}{b+d}$. This new fraction lies strictly between $\frac{a}{b}$ and $\frac{c}{d}$. In general, this fraction is not necessarily in the reduced form, but in the Stern–Brocot tree construction, all mediants turn out to be in the reduced form.

[1] We allow the algorithm to terminate as soon as a comparison returns equality.

The tree is initialized with two *sentinel* fractions: $\frac{0}{1}$ and $\frac{1}{0} = \infty$. The root node is $1/1$ which is the mediant of $\frac{0}{1}$ and $\frac{1}{0}$. For a leaf $\frac{c}{d}$ of the partial tree constructed so far, let $\frac{a}{b}$ and $\frac{e}{f}$ be elements "left" and "right" (with respect to in-order traversal of the tree). Then left child of $\frac{c}{d}$ is $\frac{a+c}{b+d}$, the mediant of $\frac{a}{b}$ and $\frac{c}{d}$. The right child of $\frac{c}{d}$ is $\frac{c+e}{d+f}$, the mediant of $\frac{c}{d}$ and $\frac{e}{f}$. Figure 1 gives an illustration up to level 6. To build finite Stern–Brocot tree containing only fractions with denominators up to n, the construction halts at any pair $\frac{a}{b}, \frac{c}{d}$ for which the mediant $\frac{a+c}{b+d}$ satisfies $b + d > n$.

The Stern-Brocot tree contains all positive reduced rational numbers exactly once. Because of this, any rational can be uniquely represented by the unique path from the root to the rational number, for example as a sequence of L (left) and R (right). It is also well known that Stern-Brocot tree is tightly connected to *continued fractions*. Every path from the root of the Stern-Brocot tree to a rational number corresponds to its continued fraction representation, and vice versa. This is discussed further below in subsection 3.1. The literature on the Stern–Brocot tree and its connections to various number-theoretic concepts is both classical and extensive. An excellent starting point is the book *Concrete Mathematics* by Graham, Knuth, and Patashnik [4].

We are interested in the binary-search-tree property of the Stern-Brocot tree. Indeed, this structure provides a binary search mechanism for locating any unknown rational number. However, a naive search of walking down the tree one step at a time will result in n comparison queries in the worst case (for example to search for $1/n$) which is far from optimal. We present a compressed search algorithm that leads to an algorithm that is much more efficient.

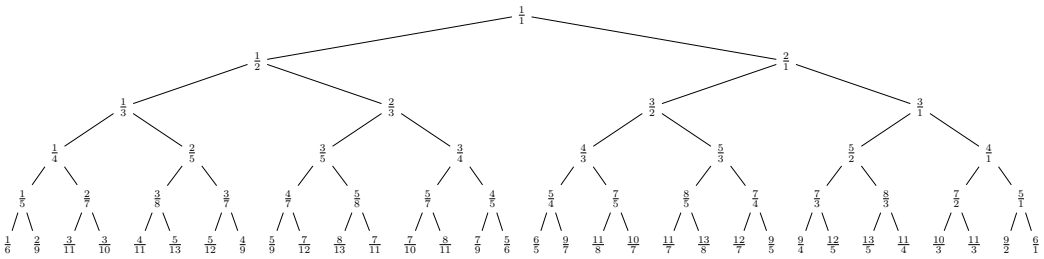

Figure 1 Stern-Brocot Tree up to level 6, starting at 1/1. As examples $\frac{1}{3} = \frac{0+1}{1+2}$ is the mediant of $\frac{0}{1}$ (not shown) and $\frac{1}{2}$, and $\frac{5}{7} = \frac{2+3}{3+4}$ is the mediant of $\frac{2}{3}$ and $\frac{3}{4}$.

3 Compressed Stern-Brocot Tree Search Algorithm

Our algorithm traverses the Stern-Brocot tree in a compressed manner to search for the unknown fraction α. As mentioned, any fraction $\alpha \in (0, \infty)$ can be uniquely represented as a sequence of "left" (L) and "right" (R) traversals of the Stern-Brocot tree starting at 0 (hence the first element of the sequence is R). This can be written in the exponential (or compressed) form as a string of alternating traversals: $R^{x_1}L^{x_2}R^{x_3}L^{x_4}\cdots D^{x_l}$ where $D \in \{L, R\}$. For example, $9/5$ is represented by R^2LR^3 (Figure 1).

If α is known to be between two fractions $\frac{p}{q}$ and $\frac{p'}{q'}$, a naive approach would take the mediant $\frac{p+p'}{q+q'}$ each step down and compare to α. However, this approach will result in $\Omega(n)$ queries in the worst case, as mentioned earlier. This is because, to reach the fraction corresponding to R^x, we are making x queries using this simple approach. Instead, we can do an *unbounded binary search* using comparisons to get to R^x. If we go to the "left" x steps by

using *repeated mediant*, the new fraction to compare will be $\frac{p'+xp}{q'+xq}$. For the unbounded binary search, Bentley and Yao give sophisticated options [1]. However, our algorithm employs the simplest and well known of them, performing an exponential search to find a bound and then do a binary search within the bound.

Suppose we computed the fraction corresponding to $R^{x_1}L^{x_2}\cdots D^{x_{i-1}}$. To compute the next fraction $R^{x_1}L^{x_2}\cdots D^{x_{i-1}}\bar{D}^{x_i}$ where $\bar{D}$ is the opposite direction of D, we must find the integer x_i. The exponential search involves making queries at $R^{x_1}L^{x_2}\cdots\bar{D}^{2^j-1}$ for each $j \in [1, k+1]$ such that $2^k - 1 < x_i \leq 2^{k+1} - 1$ where $k = \lfloor \log_2 x_i \rfloor$. At each iteration the fraction queried is $2^j - 1$ repeated mediant in the direction of traversal. Algorithm 1 describes this exponential search subroutine. The exponential search requires a maximum of $k+1$ queries with $k = \lfloor \log_2 x_i \rfloor$.

After we find k so that $x_i \in (2^k, 2^{k+1}]$, we do a binary search to find the x_i within this range. We stop the binary search when the α is located between the fraction at x_i and $x_i - 1$ steps. We do not write the pseudocode for this subroutine but it should be noted that it will find x_i by computing repeated mediants from the inputted rationals and the direction, as detailed earlier in this section. This binary search will take a maximum of $\lfloor \log_2 x_i \rfloor$ queries. Thus the exponential and binary search steps combined will take $2\lfloor \log_2 x_i \rfloor + 1$ queries.

If at any point, the queried fraction is equal to α, the search stops. Algorithm 2 describes this procedure. It calls Algorithm 1 as a subroutine. As written, it is an unbounded rational search algorithm where the bound on the numerator and denominator of the rational α we are searching for is not given. When searching $\alpha \in (0, \infty)$, the algorithm is initially called with parameters $(\alpha, \frac{0}{1}, \frac{1}{0}, \text{RIGHT})$ where α is accessed using comparison queries. For bounded search, the algorithm can be modified to slightly reduce the number of queries. This can be done by first verifying if the numerator and denominator are smaller than n before querying α.

Algorithm 1 Exponential Search.

Require: Comparisons to Rational α
Require: Rational $\frac{p}{q}$, Rational $\frac{p'}{q'}$, Direction $D \in$ {LEFT,RIGHT}
Ensure: $\frac{p}{q} < \frac{p'}{q'}$

```
1:  procedure EXPONENTIAL_SEARCH(α, p/q, p'/q', D)
2:      i ← 1
3:      if D is RIGHT then
4:          while (p+(2^i−1)p')/(q+(2^i−1)q') < α do
5:              i ← i + 1
6:          end while
7:      end if
8:      if D is LEFT then
9:          while (p'+(2^i−1)p)/(q'+(2^i−1)q) > α do
10:             i ← i + 1
11:         end while
12:     end if
13:     return i
14: end procedure
```

Algorithm 2 Compressed Stern-Brocot Tree Search.

Require: Comparisons to Rational α
Require: Rational $\frac{p}{q}$, Rational $\frac{p'}{q'}$, Direction $D \in \{\text{LEFT,RIGHT}\}$
Ensure: $\frac{p}{q} < \frac{p'}{q'}$

```
procedure RATIONAL_SEARCH(α, p/q, p'/q', D)
    k ← EXPONENTIAL_SEARCH(α, p/q, p'/q', D)
    x ← BINARY_SEARCH(α, p/q, p'/q', 2^{k-1} − 1, 2^k − 1, D)
    if D is LEFT then
        if α = (p'+xp)/(q'+xq) then
            return (p'+xp)/(q'+xq)
        end if
        result ← RATIONAL_SEARCH(α, (p'+xp)/(q'+xq), (p'+(x−1)p)/(q'+(x−1)q), RIGHT)
    else
        if α = (p+xp')/(q+xq') then
            return (p+xp')/(q+xq')
        end if
        result ← RATIONAL_SEARCH(α, (p+(x−1)p')/(q+(x−1)p'), (p+xp')/(q+xq'), LEFT)
    end if
    return result
end procedure
```

3.1 Connection to Continued Fractions

The compressed representation of a rational number has close connection to its *continued fraction representation*. Again, Graham, Knuth, and Patashnik [4] give an excellent treatment of the Stern-Brocot tree representation and the continued fraction representation with points to its history. Irwin [5] gives a geometric view of the continued fractions.

A *continued fraction* is an expression of the form $(a_0 + 1/(a_1 + 1/(a_2 + 1/(\cdots))))$ where $a_0 \in \mathbb{Z}$ and $a_i \in \mathbb{Z}_{>0}$ for all $i \geq 1$. It is customary to denote this expression in the compact form $[a_0; a_1, a_2, a_3, \ldots]$. Every real number admits a unique continued fraction expansion which is finite if and only if the number is rational. For example, $5/12$ is represented by $[0; 2, 2, 2]$ and $9/14$ is represented by $[0; 1, 1, 1, 4]$. For a (known) rational number p/q, its continued fraction representation can be efficiently computed by invoking the Euclid's algorithm.

For a fraction with continued fraction expansion $[a_0; a_1, a_2, \ldots, a_k]$, its compressed representation is $R^{a_0+1}L^{a_1}R^{a_2}L^{a_3}\cdots D^{a_k-1}$ truncated appropriately so that the final block uses $a_k - 1$ repetitions. For example for $9/14$ the compressed Stern-Brocot tree representation can be read out directly from its continued fraction representation as $R^1L^1R^1L^1R^3$ (the first R being following "right" from $\frac{0}{1}$) which matches what we observe from Figure 1.

Thus, if the rational number is known explicitly, Euclid's algorithm immediately yields this compressed representation. In contrast, our goal is to compute the representation using only comparison queries to the unknown fraction. In this setting, the problem becomes information-theoretic: we must iteratively reduce the set of possible values of the fraction using only comparison queries. We give such an algorithm and analyze the number of queries needed to compute the representation.

3.2 A $2.5849 \log_2 n + 2$ Upper Bound

While the algorithm is simple and intuitive, obtaining an accurate bound on the worst-case number of comparisons appears more complicated. We show a bound of $2.5849 \log_2 n + 2$. The constant 2.5849 is an approximation to the irrational number $16/(\log_2 73)$. We also show instances where $2.4189 \log_2 n$ comparisons are required. We will do the analysis for the unbounded version. Analysis of the bounded version will result in a slightly lower number of comparisons since the algorithm would not search past the bound. We will first analyze the algorithm performing a search on the rational numbers in the range $(0, 1)$ and then extend our analysis to all positive rationals.

▶ **Theorem 1.** *Let α be an unknown rational number in $(0, 1)$. The algorithm* Compressed Stern-Brocot Tree Search *outputs α with no more than $2.5849 \log_2 n$ queries of the form "$\beta \leq \alpha$" where n is the greater of the numerator and the denominator of α.*

To prove Theorem 1, we first introduce some notation and state some properties. The *unknown* rational $\alpha \in (0, 1)$ can be represented as a sequence $L^{x_1} R^{x_2} L^{x_3} R^{x_4} \dots D^{x_l}$ starting from $1/1$ for some l where $D \in \{L, R\}$ is the "direction" of the last *segment*. Note that here we are starting from sentinel fractions $0/1$ and $1/1$ and we consider the first traversal to be going to the left. We then get a slightly different sequence representation than when we introduced the concept. For an $i : 1 \leq i \leq l$, let d_i be the denominator of the fraction represented by $L^{x_1} R^{x_2} \dots D^{x_i}$. Let m_i be the denominator of the fraction at one less traversal than that for d_i: that of $L^{x_1} R^{x_2} \dots D^{x_i - 1}$. Define $d_0 = 1$ and $m_0 = 1$. We write some crucial properties as the following claims:

▷ Claim 2. (1) $d_0 = 1, d_l \leq n$ and $d_i = d_{i-1} + x_i m_{i-1}$ (2) $m_0 = 1$ and $m_i = d_{i-1} + (x_i - 1) m_{i-1}$ for all $1 \leq i \leq l$, (3) $d_i = m_i + m_{i-1}$ and $m_i/d_i \geq 1/2$ for all $1 \leq i \leq l$.

Proof of Claim 2. (1) and (2) follows from definitions. From the definition of d_i, it is clear that $d_i = d_{i-1} + x_i m_{i-1}$. As m_i is the denominator of the fraction at one less traversal, $m_i = d_{i-1} + (x_i - 1) m_{i-1}$. Proof of (3): Solving for d_{i-1} in $m_i = d_{i-1} + (x_i - 1) m_{i-1}$, we get $d_{i-1} = m_i - x_i m_{i-1} + m_{i-1}$. If we substitute this expression into the equation for d_i, we get $d_i = (m_i - x_i m_{i-1} + m_{i-1}) + x_i m_{i-1} = m_i + m_{i-1}$. For showing $m_i/d_i \geq 1/2$, consider the ratio:

$$\frac{m_i}{d_i} = \frac{d_{i-1} + (x_i - 1) m_{i-1}}{d_{i-1} + x_i m_{i-1}}.$$

As x_i grows larger, the ratio will increase and converge to 1. Therefore, the smallest value of the ratio will be when $x_i = 1$. From this, we have:

$$\frac{d_{i-1} + (x_i - 1) m_{i-1}}{d_{i-1} + x_i m_{i-1}} \geq \frac{d_{i-1}}{d_{i-1} + m_{i-1}}.$$

From the first part of our claim $d_i = m_i + m_{i-1}$. Since all these values are positive, $d_i \geq m_i$ for any i. Therefore, we have the following:

$$\frac{d_{i-1}}{d_{i-1} + m_{i-1}} \geq \frac{d_{i-1}}{d_{i-1} + d_{i-1}} = \frac{1}{2}.$$

Thus, we have our claim $m_i/d_i \geq 1/2$. ◁

▷ Claim 3. For any i, given d_{i-1} and m_{i-1}, the algorithm computes d_i and m_i (and x_i) using at most $2\lfloor \log_2 x_i \rfloor + 1$ queries.

Proof of Claim 3. Recall Algorithm 2 and its subroutine Algorithm 1. The next fraction in the sequence, corresponding to d_i will be less than or greater than α, depending on the direction of traversal. The fraction corresponding to m_i will be the opposite. For this proof, we will look at traversing to the right, but the proof for the left direction has the same logic. The subroutine, Algorithm 1, will compute $2^j - 1$ repeated mediants of the fractions corresponding to m_{i-1} and d_{i-1} until the fraction at $2^j - 1$ mediants is greater than α. This takes j queries. In the binary search subroutine, the search space is between $2^j - 1$ and $2^{j-1} - 1$ repeated mediants, or a range of 2^{j-1}. This takes $\lfloor \log_2(2^{j-1}) \rfloor = j - 1$ queries. The result of this search is a fraction that is greater than α and a fraction at one less traversal that is less than α. The denominator of the resulting fraction greater than α is d_i. The denominator of the resulting fraction less than α is m_i. The number of repeated mediants to get d_i is thus x_i. Therefore j is first integer value for which $2^j - 1 > x_i$ and thus $j = \lceil \log_2(x_i + 1) \rceil = \lfloor \log_2 x_i \rfloor + 1$. For our total queries from the exponential and binary search subroutines, we have $j + j - 1 = 2\lfloor \log_2 x_i \rfloor + 1$. ◁

In the following, for simplicity, we denote the function $2\lfloor \log_2 x \rfloor + 1$ as $G(x)$.

Warm-up: $3.1547 \log_2 n$ Upper Bound

We first show a simpler analysis that results in a weaker bound. Fix an arbitrary sequence $x_1, \ldots, x_l$ representing a rational as described before. We will show, for an appropriate constant C the following holds. We will choose C later but it will be greater than 3.

$$\sum_{i=1}^{l} G(x_i) \leq C \log_2 d_l$$

Since $d_l \leq n$, the bound follows. We will prove this using induction on l. For $l = 1$, $2\lfloor \log_2 x_1 \rfloor + 1 \leq C \log_2(x_1 + 1)$ for all $x_1 \geq 1$. Assume the hypothesis is true for $l - 1$. Then we have to show

$$\sum_{i=1}^{l-1} G(x_i) + 2\lfloor \log_2 x_l \rfloor + 1 \leq C \log_2 d_l.$$

Using the hypothesis, it is sufficient to show:

$$C \log_2 d_{l-1} + 2\lfloor \log_2 x_l \rfloor + 1 \leq C \log_2 d_l.$$

That is, we need a C so that

$$C \log_2 \left(\frac{d_l}{d_{l-1}} \right) = C \log_2 \left(\frac{d_{l-1} + x_l m_{l-1}}{d_{l-1}} \right) \geq 2\lfloor \log_2 x_l \rfloor + 1.$$

Using the fact that $\frac{m_{l-1}}{d_{l-1}} \geq \frac{1}{2}$, it is sufficient that

$$C \geq \frac{2\lfloor \log_2 x_l \rfloor + 1}{\log_2(1 + \frac{x_l}{2})}.$$

The function of the right-hand side takes maximum when $x_l = 4$ and corresponding C value is $5/\log_2 3$ which is ≤ 3.1547.

We can get an improved bound if we do the analysis 2 segments at a a time; that is, inductively assume the bound for $\sum_{i=1}^{l-2} G(x_i)$ and expand the last two segments. This will lead to an improved bound of $2.7024 \log_2 n$ (the exact irrational being $10/\log_2 13$). If we do a more complicated analysis taking 3 segments at a time, we will get the bound $2.6646 \log_2 n$. Here we give an analysis taking 4 segments at a time leading to an upper bound of $(16/\log_2 73) \log_2 n \leq 2.5849 \log_2 n$.

$2.5849 \log_2 n$ Upper Bound

Now we will do an analysis taking four segments at a time to find a better upper bound. The constant C comes out in the analysis as in the earlier case, but we will fix it beforehand to $C = 2.5849 > 16/\log_2 73$.

$$\sum_{i=1}^{l} G(x_i) \leq C \log_2 d_l. \tag{1}$$

We will prove (1) with induction. The arguments are similar to the warm-up case. However, since we have functions of 4 variables to deal with the analysis is somewhat messy and involves computational verification.

Base cases

Since we will be analyzing four segments at a time, we must consider the base cases for when there are four or less total segments l required to find the final rational.

$l = 1$. We must show:

$$G(x_1) \leq C \log_2 d_1.$$

From Claim 2, we find $d_1 = x_1 + 1$. The inequality can be verified that for the chosen C and $x_1 \geq 1$.

$$2\lfloor \log_2 x_1 \rfloor + 1 \leq C \log_2 (x_1 + 1)$$

$l = 2$. We should show the following:

$$G(x_1) + G(x_2) \leq C \log_2 d_2$$

From Claim 2, we get $d_2 = x_1 x_2 + x_1 + 1$. We can remove the floor functions in $G(x_1)$ and $G(x_2)$. It is sufficient to show

$$2 \log_2 x_1 x_2 + 2 \leq C \log_2 d_2.$$

Writing out d_2 explicitly in the expression, we get:

$$2 \log_2 x_1 x_2 + 2 \leq C \log_2 (x_1 x_2 + x_1 + 1). \tag{2}$$

For larger values of xs, it is sufficient to show $2 \log_2 x_1 x_2 + 2 \leq C \log_2 (x_1 x_2)$. In particular, for $x_1 x_2 \geq 2^{\frac{2}{C-2}}$, the inequality holds. In particular, plugging in $C = 2.5849$, we get that when $x_1 x_2 \geq 11$, the inequality holds. The case when $x_1 x_2 < 11$, we can explicitly test for all $\{x_1, x_2 \mid x_1, x_2 \geq 1 \text{ and } x_1 x_2 < 11\}$ and verify that all tuples satisfy (2). Since we will be using this method repeatedly, we wrote an exhaustive search program to automate this process. In particular the program takes a top value TOP, number of variables p, and two functions f and g. The program checks that $f \leq g$ for all tuples $\{\langle x_1, \ldots, x_p \rangle \mid x_i \geq 1 \text{ and } \prod_i x_i \leq \text{TOP}\}$. We verified this for the above case for two variables.

$l = 3$. We need to show

$$G(x_1) + G(x_2) + G(x_3) \leq C \log_2 d_3$$

where $d_3 = x_1x_2x_3 + x_1x_2 + x_1 + x_3 + 1$.

Once again, for larger values of x_is, it is sufficient to show: $2\log_2(x_1x_2x_3) + 3 \leq C\log_2(x_1x_2x_3)$. For the chosen value of C, the inequality holds when $x_1x_2x_3 \geq 35$. Again, for the smaller cases, the program verified that all the tuples $\{x_1, x_2, x_3 \mid x_1, x_2, x_3 \geq 1 \text{ and } x_1x_2x_3 < 35\}$ satisfy the original inequality $G(x_1) + G(x_2) + G(x_3) \leq C \log_2 d_3$.

$l = 4$. We need to show the following:

$$G(x_1) + G(x_2) + G(x_3) + G(x_4) \leq C \log_2 d_4$$

Again, using the same method, we find that the simplified inequality holds true when $x_1x_2x_3x_4 \geq 115$ for the chosen $C = 2.5849$. For the case when $x_1x_2x_3x_4 < 115$, we computationally verified that the full inequality holds true.

Inductive Step

For our inductive hypothesis, we assume the following holds true for any integer $p < l$:

$$\sum_{i=1}^{p} G(x_i) \leq C \log_2 d_p.$$

We want to show

$$\sum_{i=1}^{l} G(x_i) \leq C \log_2 d_l.$$

We can rewrite this as $\sum_{i=1}^{l-4} G(x_i) + G(x_{l-3}) + G(x_{l-2}) + G(x_{l-1}) + G(x_l) \leq C \log_2 d_l$. Assuming the hypothesis, it is sufficient to show:

$$C \log_2 d_{l-4} + G(x_{l-3}) + G(x_{l-2}) + G(x_{l-1}) + G(x_l) \leq C \log_2 d_l.$$

We can rewrite this as follows

$$G(x_{l-3}) + G(x_{l-2}) + G(x_{l-1}) + G(x_l) \leq C \log_2 \left(\frac{d_l}{d_{l-4}} \right). \tag{3}$$

We can expand out d_l up to four steps (using identities given in Claim 2) to be

$$\begin{aligned} d_l = {} & (d_{l-4} + x_{l-3}m_{l-4})\,(1 + x_{l-1} + x_lx_{l-1}) \\ & + (d_{l-4} + (x_{l-3} - 1)m_{l-4})\,(x_{l-2} + x_{l-1}x_{l-2} + x_lx_{l-1}x_{l-2} - x_{l-1} - x_lx_{l-1} + x_l) \end{aligned} \tag{4}$$

We can then write out d_l/d_{l-4} as

$$\begin{aligned} \frac{d_l}{d_{l-4}} = {} & \left(1 + x_{l-3}\frac{m_{l-4}}{d_{l-4}}\right)(1 + x_{l-1} + x_lx_{l-1}) \\ & + \left(1 + (x_{l-3} - 1)\frac{m_{l-4}}{d_{l-4}}\right)(x_{l-2} + x_{l-1}x_{l-2} + x_lx_{l-1}x_{l-2} - x_{l-1} - x_lx_{l-1} + x_l). \end{aligned} \tag{5}$$

From the fact that for all i, $x_i \geq 1$, we can see that

$$x_{l-3} > x_{l-3} - 1 \geq 0, \tag{6}$$

$$\begin{aligned} &x_{l-2} + x_{l-1}x_{l-2} + x_l x_{l-1} x_{l-2} - x_{l-1} - x_l x_{l-1} + x_l \\ &= x_l + x_{l-2} + x_{l-1}(x_{l-2} - 1) + x_l x_{l-1}(x_{l-2} - 1) \geq 0 \end{aligned} \tag{7}$$

From Claim 2, we have $\frac{m_{l-4}}{d_{l-4}} \geq \frac{1}{2}$. Equations 6 and 7 show that the terms that multiply $\frac{m_{l-4}}{d_{l-4}}$ are non-negative. Thus, we can write d_l/d_{l-4} as

$$\begin{aligned} \frac{d_l}{d_{l-4}} \geq &\left(1 + \frac{x_{l-3}}{2}\right)(1 + x_{l-1} + x_l x_{l-1}) \\ &+ \left(1 + \frac{x_{l-3} - 1}{2}\right)(x_{l-2} + x_{l-1}x_{l-2} + x_l x_{l-1} x_{l-2} - x_{l-1} - x_l x_{l-1} + x_l). \end{aligned} \tag{8}$$

From here we can substitute Equation 8 for d_l/d_{l-4} in Equation 3. With some further algebraic manipulation, we get that it is sufficient to show

$$\begin{aligned} G(x_{l-3}) + G(x_{l-2}) + G(x_{l-1}) + G(x_l) \leq C \log_2 \Bigg(&1 + \frac{x_l}{2} + \frac{x_{l-1}}{2} + \frac{x_{l-2}}{2} + \frac{x_{l-3}}{2} \\ &+ \frac{x_l x_{l-1}}{2} + \frac{x_{l-1}x_{l-2}}{2} + \frac{x_{l-2}x_{l-3}}{2} \\ &+ \frac{x_l x_{l-3}}{2} + \frac{x_l x_{l-1} x_{l-2}}{2} + \frac{x_{l-1}x_{l-2}x_{l-3}}{2} \\ &+ \frac{x_l x_{l-1} x_{l-2} x_{l-3}}{2} \Bigg) \end{aligned} \tag{9}$$

where the left-hand side is $2\left(\lfloor \log_2 x_{l-3} \rfloor + \lfloor \log_2 x_{l-2} \rfloor + \lfloor \log_2 x_{l-1} \rfloor + \lfloor \log_2 x_{l-l} \rfloor\right) + 4$.

At this point we explain how the constant C is chosen. Let $f(x_{l-3}, x_{l-2}, x_{l-1}, x_l)$ be the left-hand side function and $g(x_{l-3}, x_{l-2}, x_{l-1}, x_l)$ be the right-hand side function. Then we want $C \geq f/g$. By inspecting the function, we can conclude that for larger values of x_is, the inequality should hold for $C > 2$. Since the problematic case is when x_is are small, we can run an exhaustive search for the worst-case x_is in a bounded region. With sufficient search parameters, we find the values $x_l = 4, x_{l-1} = 2, x_{l-2} = 2, x_{l-3} = 4$ to have the maximum value for f/g. From Equation 9, we can plug in these values to get

$$5 + 3 + 3 + 5 \leq C \log_2 \left(1 + \frac{4}{2} + \frac{2}{2} + \frac{2}{2} + \frac{4}{2} + \frac{8}{2} + \frac{4}{2} + \frac{8}{2} + \frac{16}{2} + \frac{16}{2} + \frac{16}{2} + \frac{64}{2}\right)$$

Solving for C gives us the constant we use

$$C = \frac{16}{\log_2 73} \approx 2.5849.$$

We need to show the inequality holds with this chosen C. Since $G(x) = 2\lfloor \log_2 x \rfloor + 2 \leq 2 \log_2 x + 2$, for larger values of x_is, we can show that even

$$2 \log_2(x_{l-3}x_{l-2}x_{l-1}x_l) + 4 \leq C \log_2 \left(\frac{x_l x_{l-1} x_{l-2} x_{l-3}}{2}\right).$$

In particular, this holds for $x_{l-3}x_{l-2}x_{l-1}x_l \geq 2^{\frac{4+C}{C-2}}$. For $C = 2.5849$, the inequality holds true when $x_{l-3}x_{l-2}x_{l-1}x_l \geq 2450$. For the case when $x_{l-3}x_{l-2}x_{l-1}x_l < 2450$, there is only a finite number of tuples to test and we can use the aforementioned program to verify that the inequality given in Equation 9 holds for all possible combinations of such x_i.

▶ Remark. We can employ the same method as above with larger values of segments (with five for example) and we suspect that the bound can be brought closer to our lower bound of $2.4189 \log_2 n$. However, that is uninspiring and we do not know how to do a cleaner analysis to get the exact bound.

We have shown a bound for the rational numbers in $(0, 1)$. Next, we will extend our analysis to all positive rationals with a constant added to the bound.

▶ **Theorem 4.** *Let α be an unknown rational number in $(0, \infty)$. The algorithm* Compressed Stern-Brocot Tree Search *outputs α with no more than $2.5849 \log_2 n + 2$ queries of the form "$\beta \leq \alpha$" where n is the greater of the numerator and the denominator of α.*

For our first query, we determine whether $\alpha > 1$, $\alpha < 1$, or $\alpha = 1$. For $\alpha = 1$, we will conclude our search in a single query. For $\alpha < 1$, the search is now over the range $(0, 1)$, starting from $1/1$, and Theorem 1 applies. This gives a bound of $2.5849 \log_2 n + 1$ queries. For $\alpha > 1$, the algorithm then performs a search in the range $(1, \infty)$. This is the equivalent to performing a search for $1/\alpha \in (0, 1)$. In our algorithm, we have already started our search going to the right with $1/1$. Thus, Theorem 1 will not directly apply since we are not starting our search from $1/1$ but rather continuing the exponential search from $0/1$ (and inversely to the left from $1/0$ when searching for $1/\alpha$).

The number $1/\alpha$ can be represented by $L^{x_1} R^{x_2} ... D^{x_l}$ if we are starting from $1/1$. Since our algorithm is continuing its search to the left (if we are searching for $1/\alpha$) from $1/0$, this representation would be $L^{x_1+1} R^{x_2} ... D^{x_l}$. The number of repeated mediants in the first sequence would be $x_1 + 1$ and thus take $2\lfloor \log_2(x_1 + 1) \rfloor + 1$ queries. In our previous analysis, we showed the following bound on the number of queries when searching for fractions starting from $1/1$

$$\sum_{i=1}^{l} G(x_i) \leq 2.5849 \log_2 d_l.$$

The only difference in the number of queries when starting from $1/0$ would be in the first sequence. Thus, the number of queries would be

$$\sum_{i=2}^{l} G(x_i) + 2\lfloor \log_2(x_1 + 1) \rfloor + 1.$$

It is easy to see that $2\lfloor \log_2(x_1 + 1) \rfloor + 1 \leq G(x_1) + 2$. Using this fact and the bound from our previous analysis, we get

$$\sum_{i=2}^{l} G(x_i) + 2\lfloor \log_2(x_1 + 1) \rfloor + 1 \leq \sum_{i=1}^{l} G(x_i) + 2 \leq 2.5849 \log_2 d_l + 2.$$

Therefore, the number of queries required for any unknown rational number in $(0, \infty)$ is $2.5849 \log_2 d_l \leq 2.5849 \log_2 n$.

3.3 A $2.4189 \log_2 n$ Lower Bound Lower Bound

Here we prove a lower bound of approximately $2.4189 \log_2 n$ for the worst-case number of comparisons required by the algorithm. We will show this by finding the rate of increase of the denominators for a series of alternating left and right traversals in the Stern-Brocot tree. For two integers a and b, consider rationals represented by $(L^a R^b)^k$ for increasing values of k. Notice that the number of comparisons used by the algorithm for these rationals is $k \times (G(a) + G(b))$. We will express k as a function of n for large values of n.

Recall Claim 2: for any i, $d_i = d_{i-1} + x_i m_{i-1}$, $m_i = d_{i-1} + (x_i - 1)m_{i-1}$, and $d_i = m_i + m_{i-1}$. From this we have

$$d_i = d_{i-1} + x_i \left(d_{i-1} - \frac{d_{i-1} - d_{i-2}}{x_{i-1}} \right).$$

Dividing by d_{i-1} we get

$$\frac{d_i}{d_{i-1}} = 1 + x_i - \frac{x_i}{x_{i-1}} + \frac{d_{i-2}}{d_{i-1}} \cdot \frac{x_i}{x_{i-1}}.$$

Similarly, the ratio between d_{i-1} and d_{i-2} is

$$\frac{d_{i-1}}{d_{i-2}} = 1 + x_{i-1} - \frac{x_{i-1}}{x_{i-2}} + \frac{d_{i-3}}{d_{i-2}} \cdot \frac{x_{i-1}}{x_{i-2}}.$$

Assume that all x_i are alternating numbers $a, b, a, b, \ldots$. Let ϕ_a denote the ratio d_i/d_{i-1} in the limit for odd is and ϕ_b denote the ratio after b traversals in the limit for even is. Then we have thesystem of equations

$$\phi_a = 1 + a - \frac{a}{b} + \phi_b^{-1} \cdot \frac{a}{b} \text{ and } \phi_b = 1 + b - \frac{b}{a} + \phi_a^{-1} \cdot \frac{b}{a}.$$

Solving this system of equations for ϕ_a and ϕ_b will give

$$\phi_a = \frac{2 - 2\frac{b}{a} + ab + \sqrt{4ab + a^2b^2}}{2(1 + b - \frac{b}{a})} \text{ and } \phi_b = \frac{2 - 2\frac{a}{b} + ab + \sqrt{4ab + a^2b^2}}{2(1 + a - \frac{a}{b})}.$$

From this, $k = \log_{\phi_a \phi_b} n$ and the overall number of comparisons to reach a denominator at n for $(L^a R^b)^k$ is

$$(G(a) + G(b)) \log_{\phi_a \phi_b} n = \frac{(G(a) + G(b))}{\log_2 \phi_a \phi_b} \log_2 n.$$

To find a candidate for the worst case a and b, we tested all combinations of as and bs ranging from 1 to 1000. The worst case value was the sequences with $a = 8$ and $b = 1$. These values for a and b result in

$$\phi_8 = \frac{2 - 2(\frac{1}{8}) + 8(1) + \sqrt{4(8)(1) + 8^2 1^2}}{2(1 + 1 - \frac{1}{8})} = \frac{\frac{39}{4}\sqrt{96}}{\frac{15}{4}} = \frac{39}{15} + \frac{16}{15}\sqrt{6},$$

$$\phi_1 = \frac{2 - 2(8) + 1(8) + \sqrt{4(8)(1) + 8^2 1^2}}{2(1 + 8 - 8)} = \frac{-6 + \sqrt{96}}{2} = -3 + 2\sqrt{6}.$$

We then calculate the product to be $\phi_8 \cdot \phi_1 = 5 + 2\sqrt{6}$. The numerator is $G(8) + G(1) = 2\lfloor \log_2 8 \rfloor + 1 + 2\lfloor \log_2 8 \rfloor + 1 = 8$.. Thus, the worst case number comparisons to reach a numerator or denominator at n would be at least $\frac{8}{\log_2(5+2\sqrt{6})} \log_2 n > 2.4189 \log_2 n$ number of comparisons. Thus, searching for the fraction of the form $(L^8 R^1)^k$ takes at least $2.4189 \log_2 n$ number of comparisons.

3.4 Best Rational Approximation for Unknown Reals

We can modify the algorithm to solve the best rational approximation problem for an unknown real number r. Recall that the rational approximation problem for an unknown real number r is: given an approximation parameter δ, find the (unique) fraction $\frac{a}{b}$ so that $\frac{a}{b} \in [r \pm \delta]$ with smallest possible denominator b.

The modified algorithm works in much the same way as the rational search algorithm. When traversing to the left, fractions are queried the same to find some x so that $\frac{p'+xp}{q'+xq} \leq \alpha \leq \frac{p'+(x-1)p}{q'+(x-1)q}$. When traversing to the right, fractions are queried to find x so $\frac{p+xp'}{q+xq'} \geq \alpha \geq \frac{p+(x-1)p'}{q+(x-1)q'}$. This is done with exponential and binary search as in the rational search. Once we find x, we want to see if the fractions at x or $x-1$ traversals are in the range $[\alpha-\delta, \alpha+\delta]$. In the case of going left, we know $\frac{p'+xp}{q'+xq} < \alpha$. To see if $\frac{p'+xp}{q'+xq}$ is within the range, we can query to see if $\frac{p'+xp}{q'+xq} + \delta \geq \alpha$. For the fraction at $x-1$ traversals to the left, we can query to see if $\frac{p'+(x-1)p}{q'+(x-1)q} - \delta \leq \alpha$. When traversing to the right, similar queries can be called. If the fractions at x and $x-1$ traversals are not within the range, then we continue our search but switch to the opposite direction in the same manner as rational search. If the fraction at x traversals is within the range and not the fraction at $x-1$ traversals, then we return the fraction at x traversals. If the fraction at $x-1$ traversals is in range, then a binary search must be done from 1 to $x-1$ traversals to find the number of traversals z. The fraction at z traversals will be in the range $[\alpha \pm \delta]$ and the fraction at $z-1$ traversals will not. We return the fraction at z traversals.

▶ Remark. In the proof of Theorem 4, we consider the search for α in the range $(0, \infty)$ to be equivalent to the search for $1/\alpha \in (0, 1)$. For rational approximation, this would mean modifying our approximation parameter. For a rational β, we would test if it is in the range $\alpha \pm \delta$ by querying on $(\beta^{-1} \pm \delta)^{-1}$. From this, it is easy to see that a similar bound to Theorem 4 can be shown for the maximum queries of rational approximation.

3.5 Kwek–Mehlhorn Algorithm

We briefly describe Kwek–Mehlhorn rational search algorithm here. The algorithm has two phases. The first phase finds a μ such that $\alpha \in [\frac{\mu}{n^2}, \frac{\mu+1}{n^2}]$ using a standard binary search with comparison queries (which takes $2\log_2 n$ queries). The range between the two fractions $\frac{\mu}{n^2}$ and $\frac{\mu+1}{n^2}$ is $\frac{1}{n^2}$. They observe that for any two distinct fractions $(\frac{a}{b}, \frac{c}{d})$ such that $b, d \leq n$, the following holds: $\left|\frac{a}{b} - \frac{c}{d}\right| = \left|\frac{ad-bc}{bd}\right| \geq \frac{1}{n^2}$. From this, we can see that there must be only one fraction $\frac{a}{b}$ so that $b \leq n$ in the interval. In the second phase, they identify the unique fraction $\frac{a}{b}$ with $b \leq n$ in the range $[\frac{\mu}{n^2}, \frac{\mu+1}{n^2}]$. For this they use the continued fraction algorithm to find the fraction with the smallest denominator in the range using Euclid's GCD algorithm. Note that, as described, knowing the bound n on the denominators is critical for Kwek–Mehlhorn algorithm and it is an interesting open question whether we can design close to $2\log_2 n + O(1)$ query algorithm for unbounded rational search.

4 Experiments

We conducted experiments comparing the performance of the new algorithm with the Kwek–Mehlhorn algorithm. Each experiment consisted of 1000 iterations for each value of n, where n ranged over powers of 10. In each iteration, a random integer b was chosen such that $2 \leq b \leq n$, followed by a random integer a such that $1 \leq a < b$. The fraction $\frac{a}{b}$ and the corresponding value of n were used as inputs to both the Kwek–Mehlhorn algorithm and the Compressed Stern–Brocot Search algorithm. The output returned by each algorithm was checked against the input to verify correctness. For each iteration, the number of queries made was recorded. We report the average time, average number of queries, and maximum number of queries across all iterations for each value of n.

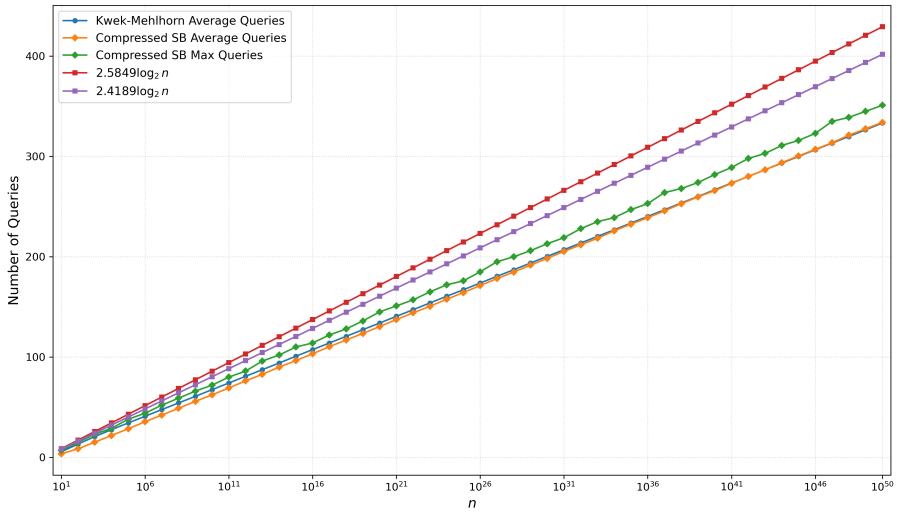

Figure 2 Queries taken by the Kwek-Mehlhorn and Compressed Stern-Brocot Tree Search algorithm for performing a rational search on a fraction $\frac{a}{b}$ such that $a < b \leq n$. Note that the maximum queries for the Kwek-Mehlhorn algorithm were not displayed in this graph because they are effectively the same as the average.

Table 1 summarizes the experimental results with odd powers included, and Figure 2 presents the query statistics graphically. Due to the high precision required for computations involving large n and rational values, Python's native floating-point arithmetic was insufficient. Therefore, we used Python's `decimal` module to ensure adequate numerical precision. The source code will be available upon request.

The rational search algorithm was also modified and implemented to solve the rational approximation problem for any unknown real number using comparison queries. We used Python's `math` module for the irrational numbers π, e, $\sqrt{2}$, and $\sqrt{5}$. For every $i \in [1, 15]$, the approximation for each irrational number was calculated with precision 10^{-i}. This solution was verified with our implementation of the Forišek approximation algorithm. See Table 2 for the values calculated. Each approximation was done 1000 times and the average time to calculate the solution is reported. The average time and number of queries to find the approximation can be found in Table 3.

All experiments were conducted on a cluster at the resident computing center. Each experiment was submitted as an independent job to a single node. Each node is equipped with two Intel Xeon Gold 6348 CPUs, providing a total of 56 cores. Memory allocation for each job was adjusted based on initial tests to ensure sufficient resources for subsequent runs.

When comparing the Kwek-Mehlhorn and Compressed Stern-Brocot Tree Search on the rational search problem, the Kwek-Mehlhorn algorithm scales better when n is really large. This aligns with the theoretical upper bounds on query complexity, although the Kwek-Mehlhorn algorithm does not outperform on average number of queries in our test distribution until $n \geq 10^{41}$. The average time for the Compressed Stern-Brocot Tree Search is consistently higher even when there is a lower average for the number of queries. This suggests improvements to the implementation are possible to lower the average time per query. However, in situations where implementation of a query takes substantial time, the savings in number of queries of typical inputs by the new algorithm could be of advantage. The maximum number of queries for the Kwek-Mehlhorn is the same as the average number of queries. Contrasting this, the Compressed Stern-Brocot Tree Search has a wide range

between its average number of queries and maximum number of queries. Even when the average number of queries is lower than that of the Kwek-Mehlhorn algorithm, the maximum number of queries is significantly higher. This result is in part due to our algorithm depending highly on the actual fraction while the Kwek-Mehlhorn algorithm asymptotically depends only on n. From this, any testing of our algorithm will depend heavily on the distribution of rationals we search for, rather than strictly the upper bound of numerators and denominators. Though the maximum number of queries for the Compressed Stern-Brocot Search is much higher than the average, it is lower than the upper bound of $2.5849 \log_2 n$ that we establish.

Table 1 The max queries, average queries, and average time of the Kwek-Mehlhorn and Compressed Stern-Brocot Search algorithm for performing a rational search on a fraction $\frac{a}{b}$ such that $a < b \leq n$, for even powers of n up to 10^{50}.

n	Kwek-Mehlhorn			Compressed Stern-Brocot		
	Max Queries	Average	Time (s)	Max Queries	Average	Time (s)
10^{2}	15	13.2	2.80×10^{-5}	15	8.5	4.42×10^{-5}
10^{4}	28	27.6	5.63×10^{-5}	29	21.8	9.57×10^{-5}
10^{6}	41	40.9	8.11×10^{-5}	44	35.5	1.43×10^{-4}
10^{8}	55	54.2	1.08×10^{-4}	59	48.9	1.92×10^{-4}
10^{10}	68	67.5	2.71×10^{-4}	72	62.3	3.78×10^{-4}
10^{12}	81	80.8	3.71×10^{-4}	86	76.2	5.01×10^{-4}
10^{14}	95	94.0	4.65×10^{-4}	102	89.9	6.15×10^{-4}
10^{16}	108	107.4	5.75×10^{-4}	114	103.3	7.51×10^{-4}
10^{18}	121	120.6	6.85×10^{-4}	128	117.0	8.95×10^{-4}
10^{20}	134	133.9	8.21×10^{-4}	145	130.4	1.05×10^{-3}
10^{22}	148	147.2	9.20×10^{-4}	157	144.1	1.21×10^{-3}
10^{24}	161	160.6	1.02×10^{-3}	172	157.6	1.36×10^{-3}
10^{26}	174	173.8	1.12×10^{-3}	185	171.3	1.52×10^{-3}
10^{28}	188	187.0	1.22×10^{-3}	200	184.7	1.67×10^{-3}
10^{30}	201	200.4	1.40×10^{-3}	213	198.4	1.90×10^{-3}
10^{32}	214	213.7	1.53×10^{-3}	228	211.8	2.09×10^{-3}
10^{34}	227	226.9	1.64×10^{-3}	239	225.9	2.25×10^{-3}
10^{36}	241	240.2	1.77×10^{-3}	253	239.0	2.44×10^{-3}
10^{38}	254	253.6	1.99×10^{-3}	268	252.9	2.72×10^{-3}
10^{40}	267	266.8	2.13×10^{-3}	282	265.9	2.93×10^{-3}
10^{42}	281	280.0	2.30×10^{-3}	298	280.2	3.17×10^{-3}
10^{44}	294	293.4	2.41×10^{-3}	311	293.8	3.34×10^{-3}
10^{46}	307	306.7	2.57×10^{-3}	323	307.1	3.57×10^{-3}
10^{48}	320	319.9	2.83×10^{-3}	339	321.2	3.90×10^{-3}
10^{50}	334	333.3	2.97×10^{-3}	351	334.0	4.10×10^{-3}

Table 2 Rational approximations returned by our algorithm for given δ and α.

δ	π	e	$\sqrt{2}$	$\sqrt{5}$
10^{-1}	16/5	8/3	3/2	7/3
10^{-2}	22/7	19/7	17/12	29/13
10^{-3}	201/64	87/32	41/29	38/17
10^{-4}	333/106	193/71	99/70	161/72
10^{-5}	355/113	1071/394	577/408	682/305
10^{-6}	355/113	2721/1001	1393/985	2207/987
10^{-7}	75948/24175	15062/5541	3363/2378	9349/4181
10^{-8}	100798/32085	23225/8544	19601/13860	12238/5473
10^{-9}	103993/33102	49171/18089	47321/33461	51841/23184
10^{-10}	312689/99532	419314/154257	114243/80782	219602/98209
10^{-11}	833719/265381	1084483/398959	275807/195025	710647/317811
10^{-12}	4272943/1360120	1084483/398959	1607521/1136689	3010349/1346269
10^{-13}	5419351/1725033	12496140/4597073	3880899/2744210	3940598/1762289
10^{-14}	58466453/18610450	28245729/10391023	9369319/6625109	16692641/7465176
10^{-15}	80143857/25510582	28245729/10391023	54608393/38613965	70711162/31622993

Table 3 Time and queries (columns abbreviated as Q) taken to compute the rational approximation for given δ and α by our algorithm.

δ	π		e		$\sqrt{2}$		$\sqrt{5}$	
	Time(s)	Q	Time(s)	Q	Time(s)	Q	Time(s)	Q
10^{-1}	7.036×10^{-6}	14	5.527×10^{-6}	12	3.889×10^{-6}	6	5.662×10^{-6}	11
10^{-2}	5.744×10^{-6}	11	6.728×10^{-6}	13	7.143×10^{-6}	10	7.758×10^{-6}	16
10^{-3}	1.049×10^{-5}	24	9.246×10^{-6}	17	8.588×10^{-6}	13	7.751×10^{-6}	16
10^{-4}	1.037×10^{-5}	24	1.252×10^{-5}	17	1.024×10^{-5}	14	9.539×10^{-6}	17
10^{-5}	9.513×10^{-6}	19	1.430×10^{-5}	26	1.358×10^{-5}	18	1.256×10^{-5}	24
10^{-6}	9.475×10^{-6}	19	1.603×10^{-5}	27	1.674×10^{-5}	21	1.537×10^{-5}	27
10^{-7}	1.978×10^{-5}	47	1.991×10^{-5}	34	1.732×10^{-5}	22	1.785×10^{-5}	32
10^{-8}	2.081×10^{-5}	47	1.992×10^{-5}	34	2.103×10^{-5}	26	1.786×10^{-5}	32
10^{-9}	1.918×10^{-5}	47	2.113×10^{-5}	33	2.221×10^{-5}	29	1.998×10^{-5}	33
10^{-10}	1.891×10^{-5}	39	2.591×10^{-5}	45	2.548×10^{-5}	30	2.313×10^{-5}	40
10^{-11}	2.133×10^{-5}	46	2.631×10^{-5}	45	2.540×10^{-5}	33	2.606×10^{-5}	43
10^{-12}	2.424×10^{-5}	50	2.709×10^{-5}	45	2.879×10^{-5}	37	2.841×10^{-5}	48
10^{-13}	2.360×10^{-5}	47	3.199×10^{-5}	54	3.234×10^{-5}	38	2.829×10^{-5}	48
10^{-14}	2.860×10^{-5}	60	3.332×10^{-5}	54	3.403×10^{-5}	41	2.954×10^{-5}	49
10^{-15}	2.853×10^{-5}	60	3.789×10^{-5}	63	3.660×10^{-5}	45	3.243×10^{-5}	56

References

1 Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information processing letters*, 5(SLAC-PUB-1679), 1976. doi:10.1016/0020-0190(76)90071-5.

2 Achille Brocot. Calcul des rouages par approximation, nouvelle méthode. *Revue Chronométrique*, 3:186–194, 1861.

3 Michal Forišek. Approximating rational numbers by fractions. In *International Conference on Fun with Algorithms*, pages 156–165. Springer, 2007.

4 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1994.

5 M. C. Irwin. Geometry of continued fractions. *The American Mathematical Monthly*, 96(8):696–703, October 1989.

6 Stephen Kwek and Kurt Mehlhorn. Optimal search for rationals. *Information Processing Letters*, 86(1):23–26, 2003. doi:10.1016/S0020-0190(02)00455-6.

7 Christos H. Papadimitriou. Efficient search for rationals. *Information Processing Letters*, 8(1):1–4, 1979. doi:10.1016/0020-0190(79)90079-6.

8 Steven Reiss. Rational search. *Information Processing Letters*, 8(2):89–90, 1979. doi:10.1016/0020-0190(79)90036-0.

9 M. A. Stern. Ueber eine zahlentheoretische funktion. *Journal für die reine und angewandte Mathematik*, 55:193–220, 1858.

10 Ke Sun. 基于 stern-brocot tree 的有理数集二分算法及其优化 [a binary search algorithm for the set of rational numbers based on stern-brocot tree and its optimization]. https://izard.space/2022/06/15/%E5%9F%BA%E4%BA%8EStern-Brocot%20tree%E7%9A%84%E6%9C%89%E7%90%86%E6%95%B0%E9%9B%86%E4%BA%8C%E5%88%86%E7%AE%97%E6%B3%95%E5%8F%8A%E5%85%B6%E4%BC%98%E5%8C%96/, 2022. Accessed: 2026-06-25.