Institute of Science and Technology

Average-Case Analysis of Programs: Automated Recurrence Analysis for Almost-Linear Bounds

1 Anonymous and 2 Anonymous and 3 Anonymous

Technical Report No. IST-2016-619-v1+1
Deposited at 15 Jul 2016 09:04
https://repository.ist.ac.at/619/1/popl2017b.pdf

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

Average-Case Analysis of Programs: Automated Recurrence Analysis for Almost-Linear Bounds

Abstract

We consider the problem of developing automated techniques to aid the average-case complexity analysis of programs. Several classical textbook algorithms have quite efficient average-case complexity, whereas the corresponding worst-case bounds are either inefficient (e.g., QUICK-SORT), or completely ineffective (e.g., COUPON-COLLECTOR). Since the main focus of average-case analysis is to obtain efficient bounds, we consider bounds that are either logarithmic, linear, or almost-linear ($\mathcal{O}(\log n)$, $\mathcal{O}(n)$, $\mathcal{O}(n \cdot \log n)$, respectively, where n represents the size of the input). Our main contribution is a sound approach for deriving such average-case bounds for randomized recursive programs. Our approach is efficient (a simple linear-time algorithm), and it is based on (a) the analysis of recurrence relations induced by randomized algorithms, and (b) a guess-and-check technique. Our approach can infer the asymptotically optimal average-case bounds for classical randomized algorithms, including RANDOMIZED-SEARCH, QUICK-SORT, QUICK-SELECT, COUPON-COLLECTOR, where the worst-case bounds are either inefficient (such as linear as compared to logarithmic of average-case, or quadratic as compared to linear or almost-linear of average-case), or ineffective. We have implemented our approach, and the experimental results show that we obtain the bounds efficiently for various classical algorithms.

Categories and Subject Descriptors F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs

General Terms Program Verification, Average-Case Analysis

Keywords Randomized Recursive Programs, Average-Case Analysis

1. Introduction

Static analysis for quantitative bounds. Static analysis of programs aims to reason about programs without running them. The most basic properties for static analysis are qualitative properties, such as safety, termination, liveness, that for every trace of a program gives a Yes or No answer (such as assertion violation or not, termination or not). However, recent interest in analysis of resource-constrained systems, such as embedded systems, as well as for performance analysis, quantitative performance characteristics are necessary. For example, the qualitative problem of termination asks whether a given program always terminates, whereas the quantitative problem asks to obtain precise bounds on the number of steps, and is thus a more challenging problem. Hence the problem of automatically reasoning about resource bounds (such as time complexity bounds) of programs is both of significant theoretical as well as practical interest.

Worst-case bounds. The worst-case analysis of programs is the fundamental problem in computer science, which is the basis of algorithms and complexity theory. However, manual proofs of worst-case analysis can be tedious and also require non-trivial mathematical ingenuity, e.g., the book *The Art of Computer Programming* by Knuth presents a wide range of involved techniques to derive such precise bounds [31, 32]. There has been a considerable research effort for automated analysis of such worst-case bounds for programs, see [17, 18, 22, 23] for excellent expositions on the significance of deriving precise worst-case bounds and the automated methods to derive them. For the worst-case analysis there are several techniques, such as worst-case execution time analysis [39], resource analysis using abstract interpretation and type systems [3, 18, 22, 23, 29], ranking functions [5, 6, 9, 11, 36–38, 40], as well as recurrence relations [1–3, 15].

Average-case bounds. While several works have focused on deriving worst-case bounds for programs, quite surprisingly little work has been done to derive precise bounds for average-case analysis, with the exception of [13], which focuses on randomization in combinatorial structures (such as trees). This is despite the fact that average-case analysis is an equally important pillar of theoretical computer science, both in terms of theoretical and practical significance. For example, while for real-time systems with hard constraints worst-case analysis is necessary, for real-time systems with soft constraints the more relevant information is the average-case analysis. Below we highlight three key significance of average-case analysis.

1. *Simplicity and desired properties:* The first key aspect is *simplicity*: if the goal is to design algorithms with efficient average-case complexity as compared to worst-case complexity, then much simpler algorithms (and thus simple and efficient implementations) exist. A classic example is as follows: consider the classical MEDIAN-FIND problem, or in general the SELECTION problem that given a set of n numbers and $0 \leq k \leq n$, asks to find the k-th largest number (for median $k = n/2$). The classical linear-time algorithm for the problem (see [10, Chapter 9]) is quite involved, and its worst-case analysis to obtain linear time bound is rather complex. In contrast, a much simpler algorithm exists (namely, QUICK-SELECT, see Example 3 in Section 2.2) that has linear average-case complexity. A related advantage for randomized algorithms with average-case complexity is that such algorithms enjoy many desired properties, which deterministic algorithms do not have. A basic ex-

ample is CHANNEL-CONFLICT RESOLUTION (see Example 7, Section 2.4) where the simple randomized algorithm can be implemented in a distributed or concurrent setting, whereas deterministic algorithms are quite cumbersome. In summary, randomized algorithms with average-case analysis lead to simple, efficient algorithms with several desirable properties.

2. *Efficiency in practice:* The second key aspect is *efficiency in practice*. Since worst-case analysis concerns with corner cases that rarely arise, many algorithms and implementations have a much better average-case complexity as compared to the worst-case complexity, and they perform extremely well in practice. A classic example is the QUICK-SORT algorithm, that has quadratic worst-case complexity, but almost linear average-case complexity, and is one of the most efficient sorting algorithms in practice.

3. *Worst-case analysis ineffective:* The third key advantage of average-case analysis is that in several important cases the worst-case analysis is completely ineffective. For example, consider one of the textbook stochastic process, namely the COUPON-COLLECTOR problem, where there are n types of coupons to be collected, and in each round, a coupon type among the n types is obtained uniformly at random. The process stops when all types are collected. The COUPON-COLLECTOR process is one of the basic and classical stochastic processes, with numerous applications in network routing, load balancing, etc (see [34, Chapter 3] for applications of COUPON-COLLECTOR problems). For the worst-case analysis, the process might not terminate (thus has infinite worst-case bound), but the average-case analysis shows that the expected termination time is $\mathcal{O}(n \cdot \log n)$, i.e., quite efficient.

Challenges. The average-case analysis brings several new challenges as compared to the worst-case analysis.

1. First, for the worst-case complexity bounds, the most classical characterization for analysis of recurrences is the *Master theorem* (cf. [10, Chapter 1]). However, the average-case analysis problems give rise to recurrences that are not characterized by the Master theorem.

2. Second, techniques like ranking functions (linear or polynomial ranking functions) cannot derive efficient bounds such as $\mathcal{O}(\log n)$ or $\mathcal{O}(n \cdot \log n)$.

While average-case analysis has been considered for combinatorial structures using generating function [13], we are not aware of any automated technique to handle recurrences arising from randomized algorithms.

Analysis problem: Almost-linear bounds for univariate and separable bivariate recurrences. In this work we consider the algorithmic analysis problem of recurrences arising naturally for randomized recursive programs. More specifically we consider the following problem.

- We consider two classes of average-case recurrences: (a) *univariate* class with one variable (which represents the array length, or the number of input elements, as required in problems such as QUICK-SELECT, QUICK-SORT etc); and (b) *separable bivariate* class with two variables (where the two independent variables represent the total number of elements and total number of successful cases, respectively, as required in problems such as COUPON-COLLECTOR, CHANNEL-CONFLICT RESOLUTION). The above two classes capture a large class of average-case analysis problems, including all the classical ones mentioned above. Moreover, the main purpose of average-case analysis is to obtain efficient bounds. Hence we focus on the case of logarithmic, linear, and almost-linear bounds (i.e., bounds of form $\mathcal{O}(\log n)$, $\mathcal{O}(n)$ and $\mathcal{O}(n \cdot \log n)$, respectively, where n is the size of the input).

Thus the main problem we consider is to automatically derive such efficient bounds for randomized univariate and separable bivariate recurrence relations to aid the average-case analysis of programs.

Our contributions. Our main contribution is a sound approach for analysis of recurrences for average-case analysis. The details of our contributions are as follows:

1. *Efficient algorithm.* Our main contributions are efficient algorithms for our problem. We first present a linear-time algorithm for the univariate case. Our algorithm is based on simple comparison of leading terms of pseudo-polynomials. Second, we present a simple reduction for separable bivariate recurrence analysis to the univariate case. Our efficient (linear-time) algorithm can soundly infer logarithmic, linear, and almost-linear bounds for average-case recurrences involving one or two variables.

2. *Analysis of classical algorithms.* We show that for several classical algorithms, such as RANDOMIZED-SEARCH, QUICK-SELECT, QUICK-SORT, DIAMETER-COMPUTATION, COUPON-COLLECTOR, CHANNEL-CONFLICT RESOLUTION (see Section 2.2 and Section 2.4 for examples), our sound approach can obtain the asymptotically optimal average-case bounds. In all the cases above, either the worst-case bounds (i) do not exist (e.g., COUPON-COLLECTOR), or (ii) are quadratic when the average-case bounds are linear or almost-linear (e.g., QUICK-SELECT, QUICK-SORT); or (iii) are linear when the average-case bounds are logarithmic (e.g., RANDOMIZED-SEARCH). Thus in cases where the worst-case bounds are either not applicable, or grossly overestimate the average-case bounds, our technique is both efficient (linear-time) and can infer the optimal bounds.

3. *Implementation.* Finally, we have implemented our approach, and we present experimental results on the classical examples to show that we can efficiently achieve the automated average-case analysis of randomized recurrence relations.

Novelty and technical contribution. The key novelty of our approach is an automated method to analyze recurrences arising from randomized recursive programs, which are not covered by Master theorem. Our approach is based on a guess-and-check technique. We show that by over-approximating terms in a recurrence relation through integral and Taylor's expansion, we can soundly infer logarithmic, linear and almost-linear bounds using simple comparison between leading terms of pseudo-polynomials.

2. Recurrence Relations

In this section, we present our mini specification language for recurrence relations for average-case analysis. The language is designed to capture running time of recursive randomized algorithms which involve (i) only one function call whose average-case complexity is to be determined, (ii) at most two integer parameters, and (iii) involve randomized-selection or divide-and-conquer techniques. We present our language separately for the univariate and bivariate cases. In the sequel, we denote by $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Z}$, and $\mathbb{R}$ the sets of all positive integers, non-negative integers, integers, and real numbers, respectively.

2.1 Univariate Randomized Recurrences

Below we define the notion of univariate randomized recurrence relations. First, we introduce the notion of univariate recurrence expressions. Since we only consider single recursive function call, we use 'T' to represent the (only) function call. We also use '$\mathfrak{n}$' to represent the only parameter in the function declaration.

Univariate recurrence expressions. The syntax of *univariate recurrence expressions* $\mathfrak{e}$ is generated by the following grammar:

$$\begin{aligned}\mathfrak{e} ::= & \, c \mid \mathfrak{n} \mid \ln \mathfrak{n} \mid \mathfrak{n} \cdot \ln \mathfrak{n} \mid \frac{1}{\mathfrak{n}} \\ & \mid \mathrm{T}(\mathfrak{n}-1) \mid \mathrm{T}\left(\left\lfloor \frac{\mathfrak{n}}{2} \right\rfloor\right) \mid \mathrm{T}\left(\left\lceil \frac{\mathfrak{n}}{2} \right\rceil\right) \\ & \mid \frac{\sum_{\mathfrak{j}=1}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j})}{\mathfrak{n}} \mid \frac{1}{\mathfrak{n}} \cdot \left(\sum_{\mathfrak{j}=\lceil \frac{\mathfrak{n}}{2} \rceil}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j}) + \sum_{\mathfrak{j}=\lfloor \frac{\mathfrak{n}}{2} \rfloor}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j}) \right) \\ & \mid c \cdot \mathfrak{e} \mid \mathfrak{e} + \mathfrak{e}\end{aligned}$$

where $c \in [1, \infty)$ and $\ln(\centerdot)$ represents the natural logarithm function with base e. Informally, $\mathrm{T}(\mathfrak{n})$ is the (expected) running time of a recursive randomized program which involves only one recursive routine indicated by T and only one parameter indicated by $\mathfrak{n}$. Then each $\mathrm{T}(\centerdot)$-term in the grammar has a direct algorithmic meaning:

- $\mathrm{T}(\mathfrak{n}-1)$ may mean a recursion to a sub-array with length decremented by one;
- $\mathrm{T}\left(\left\lfloor \frac{\mathfrak{n}}{2} \right\rfloor\right)$ and $\mathrm{T}\left(\left\lceil \frac{\mathfrak{n}}{2} \right\rceil\right)$ may mean a recursion related to a divide-and-conquer technique;
- finally,

  $$\frac{\sum_{\mathfrak{j}=1}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j})}{\mathfrak{n}} \text{ and } \frac{1}{\mathfrak{n}} \cdot \left(\sum_{\mathfrak{j}=\lceil \frac{\mathfrak{n}}{2} \rceil}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j}) + \sum_{\mathfrak{j}=\lfloor \frac{\mathfrak{n}}{2} \rfloor}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j}) \right)$$

  may mean a recursion related to a randomized selection of an array index.

Substitution. Consider a function $h : \mathbb{N} \to \mathbb{R}$ and univariate recurrence expression $\mathfrak{e}$. The *substitution function*, denoted by $\mathsf{Sub}(\mathfrak{e}, h)$, is the function from $\mathbb{N}$ into $\mathbb{R}$ such that the value for n is obtained by evaluation through substituting h for T and n for $\mathfrak{n}$, respectively. Moreover, if $\mathfrak{e}$ does not involve the appearance of 'T', then we use the abbreviation $\mathsf{Sub}(\mathfrak{e})$ i.e., we omit h. For example, (i) if $\mathfrak{e} = \mathfrak{n} + \mathrm{T}(\mathfrak{n}-1)$, and $h : n \mapsto n \cdot \log n$, then $\mathsf{Sub}(\mathfrak{e}, h)$ is the function $n \mapsto n + (n-1) \cdot \log(n-1)$, and (ii) if $\mathfrak{e} = 2 \cdot \mathfrak{n}$, then $\mathsf{Sub}(\mathfrak{e})$ is the function $n \mapsto 2n$.

Univariate recurrence relation. A *univariate recurrence relation* $G = (\mathsf{eq}_1, \mathsf{eq}_2)$ is a pair of equalities as follows:

$$\begin{cases} \mathsf{eq}_1 : \ \mathrm{T}(\mathfrak{n}) = \mathfrak{e} \\ \mathsf{eq}_2 : \ \mathrm{T}(1) = c \end{cases} \tag{1}$$

where $c \in (0, \infty)$ and $\mathfrak{e}$ is a univariate recurrence expression.

Evaluation of univariate recurrence relation. Consider a univariate recurrence relation G

$$\begin{cases} \mathsf{eq}_1 : \ \mathrm{T}(\mathfrak{n}) = \mathfrak{e} \\ \mathsf{eq}_2 : \ \mathrm{T}(1) = c \end{cases} \tag{2}$$

where $c \in (0, \infty)$ and $\mathfrak{e}$ is a univariate recurrence expression. The *evaluation sequence* $\mathsf{Eval}(G)$ is as follows: $\mathsf{Eval}(G)(1) = c$, and for $n \geq 2$, given $\mathsf{Eval}(G)(i)$ for $1 \leq i < n$, for the value $\mathsf{Eval}(G)(n)$ we evaluate the expression $\mathsf{Sub}(\mathfrak{e}, \mathsf{Eval}(G))$, because in $\mathfrak{e}$ the parameter $\mathfrak{n}$ always decreases and thus this is well-defined.

Finite vs infinite solution. Note that the above description gives a computational procedure to compute $\mathsf{Eval}(G)$ for any finite n, in linear time in n through dynamic programming. The interesting question is to algorithmically analyze the infinite behavior. A function $T_G : \mathbb{N} \to \mathbb{R}$ is called a solution to G if $T_G(n) = \mathsf{Eval}(G)(n)$ for all $n \geq 1$. The function T_G is unique and explicitly defined as follows:

- *Base Step.* $T_G(1) := c$;
- *Recursive Step.* $T_G(n) := \mathsf{Sub}(\mathfrak{e}, T_G)(n)$ for all $n \geq 2$.

The interesting algorithmic question is to reason about the asymptotic infinite behaviour of T_G.

2.2 Motivating Classical Examples

In this section we present several classical examples of randomized programs whose recurrence relations belong to the class of univariate recurrence relations described in Section 2.1.

Example 1 (RANDOMIZED-SEARCH)**.** *Consider the Sherwood's* RANDOMIZED-SEARCH *algorithm (cf. [33, Chapter 9]) depicted in Fig. 1. The algorithm checks whether an integer value d is present within the index range $[i, j]$ ($0 \leq i \leq j$) in an integer array ar which is sorted in increasing order and is without duplicate entries. The algorithm outputs either the index for d in ar or -1 meaning that d is not present in the index range $[i, j]$ of ar.*

The description of the pseudo-code is as follows. The first four lines deal with the base case when there is only one index in the index range. The remaining lines deal with the recursive case: in line 6, an index k is uniformly sampled from $\{i, i+1, \ldots, j\}$; line 7–8 check whether k is the output; line 9–12 perform the recursive calls depending on whether $ar[k] < d$ or not; finally, line 13–14 handle the case when $d < ar[i]$ or $d > ar[j]$.

Let $T : \mathbb{N} \to \mathbb{N}$ be the function such that for any $n \in \mathbb{N}$, we have $T(n)$ is the supremum of the expected execution times upon all inputs (ar, i, j) with $j - i + 1 = n$. We derive a recurrence relation for T as follows. Let $n \in \mathbb{N}$ and $(ar, i, j), d$ be any input such that $n = j - i + 1$. We clarify two cases below:

1. *there exists an $i \leq k^* < j$ such that $ar[k^*] \leq d < ar[k^*+1]$, where $ar[j+1]$ is interpreted ∞ here;*
2. *$ar[j] \leq d$ or $d < ar[i]$.*

In both cases, we have $T(1) = 1$. In Case 1, we deduce from the pseudo-code in Fig. 1 that

$$T(n) \leq 6 + \frac{1}{n} \cdot \max_{1 \leq \ell^* < n} \left(\sum_{\ell=1}^{\ell^*} T(n-\ell) + \sum_{\ell=\ell^*+1}^{n} T(\ell-1) \right)$$

for all $n \geq 2$, where the maximum ranges over all $\ell^* := k^* - i + 1$'s. In Case 2, similarly we deduce that

$$T(n) \leq 6 + \frac{1}{n} \cdot \max \left\{ \sum_{\ell=1}^{n-1} T(n-\ell), \ \sum_{\ell=2}^{n} T(\ell-1) \right\}$$

Thus a preliminary version G' of the recurrence relation is $\mathrm{T}(1) = 1$ and

$$\mathrm{T}(n) = 6 + \frac{1}{n} \cdot \max_{1 \leq \ell^* < n} \left(\sum_{\ell=1}^{\ell^*} \mathrm{T}(n-\ell) + \sum_{\ell=\ell^*+1}^{n} \mathrm{T}(\ell-1) \right)$$

```
randsearch(ar, i, j, d) {
1:   if (i = j and ar[i] ≠ d)
2:     return −1;
3:   else if (i = j and ar[i] = d)
4:     return i;
5:   else
6:     k ← uniform(i, j);

7:     if (ar[k] = d)
8:       return k;
9:     else if (ar[k] < d and k < j)
10:      return randsearch(ar, k + 1, j, d);
11:    else if (ar[k] > d and i < k)
12:      return randsearch(ar, 1, k − 1, d);
13:    else
14:      return −1;
       end if
     end if
}
```

Figure 1. Pseudo-code for Sherwood's RANDOMIZED-SEARCH

for all $n \geq 2$. Let $T' : \mathbb{N} \to \mathbb{R}$ be the unique solution to G'. Then from the fact that $T'(2) \geq T'(1)$, by induction T' is monotonically increasing. Thus the maximum

$$\max_{1 \leq \ell^* < n} \left(\sum_{\ell=1}^{\ell^*} T'(n-\ell) + \sum_{\ell=\ell^*+1}^{n} T'(\ell-1) \right)$$

is attained at $\ell^* = \left\lfloor \frac{n}{2} \right\rfloor$ for all $n \geq 2$. Then G' is transformed into our final recurrence relation as follows:

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 6 + \frac{1}{\mathfrak{n}} \cdot \left(\sum_{\mathrm{j}=\lceil \frac{\mathfrak{n}}{2} \rceil}^{\mathfrak{n}-1} \mathrm{T}(\mathrm{j}) + \sum_{\mathrm{j}=\lfloor \frac{\mathfrak{n}}{2} \rfloor}^{\mathfrak{n}-1} \mathrm{T}(\mathrm{j}) \right) \\ \mathrm{T}(1) = 1 \end{cases} . \quad (3)$$

We note that the worst-case complexity for this algorithm is $\Theta(n)$.

Example 2 (QUICK-SORT)**.** *Consider the* QUICK-SORT *algorithm [10, Chapter 7] depicted in Fig. 2, where every input (ar, i, j) is assumed to satisfy that $0 \leq i \leq j$ and ar is an array of integers which does not contain duplicate numbers.*

The description of the pseudo-code is as follows: first, line 2 samples an integer uniformly from $\{i, \ldots, j\}$; then, line 3 calls a subroutine pivot *which (i) rearranges ar such that integers in ar which are less than $ar[k]$ come first, then $ar[k]$, and finally integers in ar greater than $ar[k]$, and (ii) outputs the new index m of $ar[k]$ in ar; and finally, lines 4–7 handle recursive calls to sub-arrays.*

From the pseudo-code, the following recurrence relation is easily obtained:

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 2 \cdot \mathfrak{n} + 2 \cdot \frac{\sum_{\mathrm{j}=1}^{\mathfrak{n}-1} \mathrm{T}(\mathrm{j})}{\mathfrak{n}} \\ \mathrm{T}(1) = 1 \end{cases} \quad (4)$$

where $\mathrm{T}(\mathfrak{n})$ represents the maximal expected execution time over all inputs (ar, i, j) such that $j - i + 1$ is a fixed constant represented by $\mathfrak{n}$, and the execution time of pivot *is represented by $2 \cdot \mathfrak{n}$. We note that the worst-case complexity for this algorithm is $\Theta(n^2)$.*

Example 3 (QUICK-SELECT)**.** *Consider the* QUICK-SELECT *algorithm (cf. [10, Chapter 9]) depicted in Fig. 3 which upon any input (ar, i, j) and d such that $0 \leq i \leq j$, $1 \leq d \leq j - i + 1$ and ar contains no duplicate integers, finds the d-th largest integer in ar. Note that for an array of size n, and $d = n/2$, we have the* MEDIAN-FIND *algorithm.*

```
quicksort(ar, i, j) {
1:  if (i < j)
2:    k ← uniform(i, j);
3:    m ← pivot(ar, i, j, ar[k]);

4:    if (i ≤ m − 1)
5:      quicksort(ar, i, m − 1);
      end if

6:    if (m + 1 ≤ j)
7:      quicksort(ar, m + 1, j);
      end if
    end if
}
```

Figure 2. Pseudo-code for Randomized QUICK-SORT

The description of the pseudo-code is as follows: line 1 handles the base case; line 3 starts the recursive case by sampling k uniformly from $\{i, \ldots, j\}$; line 4 rearranges ar and returns an m in the same way as pivot *in* QUICK-SORT *(cf. Example 2); line 5 handles the case when $ar[k]$ happens to be the d-th largest integer in ar; and finally, line 7–10 handle the recursive calls.*

Let $T : \mathbb{N} \to \mathbb{N}$ be the function such that for any $n \in \mathbb{N}$, we have $T(n)$ is the supremum of the expected execution times upon all inputs (ar, i, j) with $j - i + 1 = n$. By an analysis on where the d-th largest integer lies in ar which is similar to the analysis on d in Example 1, a preliminary recurrence relation is obtained such that $\mathrm{T}(1) = 1$ and

$$\mathrm{T}(n) = 4 + 2 \cdot n + \frac{1}{n} \cdot \max_{1 \leq \ell^* \leq n} \left(\sum_{\ell=1}^{\ell^*-1} \mathrm{T}(n-\ell) + \sum_{\ell=\ell^*}^{n} \mathrm{T}(\ell-1) \right).$$

By similar monotone argument in Example 1, the maximum of the right-hand-side expression above is attained at $\ell^* = \left\lfloor \frac{n+1}{2} \right\rfloor$ for all $n \geq 2$. By the fact that $\left\lfloor \frac{n+1}{2} \right\rfloor = \left\lceil \frac{n}{2} \right\rceil$ for all $n \geq 2$, the following recurrence relation is obtained:

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 4 + 2 \cdot \mathfrak{n} + \frac{1}{\mathfrak{n}} \cdot \left(\sum_{\mathrm{j}=\lfloor \frac{n}{2} \rfloor + 1}^{n-1} \mathrm{T}(\mathrm{j}) + \sum_{\mathrm{j}=\lceil \frac{n}{2} \rceil}^{\mathfrak{n}-1} \mathrm{T}(\mathrm{j}) \right) \\ \mathrm{T}(1) = 1 \end{cases}$$

To fit our univariate recurrence expression, we use over-approximation, and the final recurrence relation for this example is

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 4 + 2 \cdot \mathfrak{n} + \frac{1}{\mathfrak{n}} \cdot \left(\sum_{\mathrm{j}=\lfloor \frac{n}{2} \rfloor}^{n-1} \mathrm{T}(\mathrm{j}) + \sum_{\mathrm{j}=\lceil \frac{n}{2} \rceil}^{\mathfrak{n}-1} \mathrm{T}(\mathrm{j}) \right) \\ \mathrm{T}(1) = 1 \end{cases} . \quad (5)$$

We note that the worst-case complexity for this algorithm is $\Theta(n^2)$.

Example 4 (DIAMETER-COMPUTATION)**.** *Consider the* DIAMETER-COMPUTATION *algorithm (cf. [34, Chapter 9]) to compute the diameter of an input finite set S of three-dimensional points. A pseudo-code to implement this is depicted in Fig. 4.*

The description of the pseudo-code is as follows: line 1–2 handle the base case; line 3 samples a point p uniformly from S; line 4 calculates the maximum distance in S from p; line 5 calculates the

```
quickselect(ar, i, j, d) {
1:   if (i = j) return a[i];
2:   else
3:     k ← uniform(i, j);
4:     m ← pivot(ar, i, j, ar[k]);

5:     if (m − i + 1 = d)
6:       return ar[m];
7:     else if (m − i + 1 < d)
8:       return quickselect(ar, m + 1, j, d);
9:     else if (m − i + 1 > d)
10:      return quickselect(ar, i, m − 1, d);
       end if
     end if
}
```

Figure 3. Pseudo-code for Randomized QUICK-SELECT

intersection of all balls centered at points in S with uniform radius d; line 6 calculates the set of points outside U; lines 7–8 handle the situation $S' = \emptyset$ which implies that d is the diameter; lines 9–10 handle the recursive call to S'. Due to uniform choice of p at line 3, the size of S' is uniformly in $[0, |S| - 1]$; it then follows a pivoting (similar to that in Example 3 and Example 2) by line 5 w.r.t the linear order over $\{\max_{p' \in S} \mathit{dist}(p, p') \mid p \in S\}$.

Lines 5–6 can be done in $\mathcal{O}(|S| \cdot \log |S|)$ time for Euclidean distance, and $\mathcal{O}(|S|)$ time for L_1 metric [34]. Thus, we obtain two recurrence relations as follows:

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 2 + \mathfrak{n} + 2 \cdot \mathfrak{n} \cdot \ln \mathfrak{n} + \frac{\sum_{\mathfrak{j}=1}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j})}{\mathfrak{n}} \\ \mathrm{T}(1) = 1 \end{cases} \tag{6}$$

for Euclidean distance with the execution time for lines 5–6 being taken to be $2 \cdot \mathfrak{n} \cdot \ln \mathfrak{n}$, and

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 2 + \mathfrak{n} + 2 \cdot \mathfrak{n} + \frac{\sum_{\mathfrak{j}=1}^{\mathfrak{n}-1} \mathrm{T}(\mathfrak{j})}{\mathfrak{n}} \\ \mathrm{T}(1) = 1 \end{cases} \tag{7}$$

for L_1 metric with the execution time for lines 5–6 being taken to be $2 \cdot \mathfrak{n}$.

We note that the worst-case complexity for this algorithm is as follows: for Euclidean metric it is $\Theta(n^2 \cdot \log n)$ and for the L_1 metric it is $\Theta(n^2)$.

Example 5 (Sorting with QUICK-SELECT)**.** *Consider a sorting algorithm depicted in Fig. 5 which selects the median through the* QUICK-SELECT *algorithm. The recurrence relation is directly obtained as follows:*

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = 4 + T^*(\mathfrak{n}) + \mathrm{T}\left(\lfloor \frac{\mathfrak{n}}{2} \rfloor\right) + \mathrm{T}\left(\lceil \frac{\mathfrak{n}}{2} \rceil\right) \\ \mathrm{T}(1) = 1 \end{cases} \tag{8}$$

where $T^*(\cdot)$ is an upper bound on the expected running time of QUICK-SELECT *(cf. Example 3). We note that the worst-case complexity for this algorithm is $\Theta(n^2)$.*

2.3 Separable Bivariate Randomized Recurrences

We consider a generalization of the univariate recurrence relations to a class of bivariate recurrence relations called *separable bivariate recurrence relations*. Similar to the univariate situation, we use 'T' to represent the (only) function call and '$\mathfrak{n}$', '$\mathfrak{m}$' to represent namely the two integer parameters.

```
diameter(S) {
1:   if (|S| = 1)
2:     return 0;
     else
3:     p ← uniform(S);
4:     d ← max_{p' ∈ S} dist(p, p')
5:     U ← ⋂_{p' ∈ S} {p'' ∈ ℝ³ | dist(p'', p') ≤ d}
6:     S' ← S \ U

7:     if (S' = ∅)
8:       return d
9:     else
10:      return diameter(S')
       end if
     end if
}
```

Figure 4. Pseudo-code for DIAMETER-COMPUTATION

```
sortbyselect(ar, i, j) {
1:   if (i < j)
2:     m ← quickselect(ar, i, j, ⌊(j − i + 1)/2⌋);
3:     if (i < m − 1)
4:       sortbyselect(ar, i, m − 1);
       end if

5:     if (m + 1 < j)
6:       sortbyselect(ar, m + 1, j);
       end if
     end if
}
```

Figure 5. Pseudo-code for Sorting with QUICK-SELECT

Separable Bivariate Recurrence Expressions. The syntax of *separable bivariate recurrence expressions* is illustrated by $\mathfrak{e}, \mathfrak{h}$ and $\mathfrak{b}$ as follows:

$$\begin{aligned} \mathfrak{e} ::= {} & \mathrm{T}(\mathfrak{n}, \mathfrak{m} - 1) \mid \mathrm{T}\left(\mathfrak{n}, \left\lfloor \frac{\mathfrak{m}}{2} \right\rfloor\right) \mid \mathrm{T}\left(\mathfrak{n}, \left\lceil \frac{\mathfrak{m}}{2} \right\rceil\right) \\ & \mid \frac{\sum_{\mathfrak{j}=1}^{\mathfrak{m}-1} \mathrm{T}(\mathfrak{n}, \mathfrak{j})}{\mathfrak{m}} \mid \frac{1}{\mathfrak{m}} \cdot \left(\sum_{\mathfrak{j}=\lceil \frac{\mathfrak{m}}{2} \rceil}^{\mathfrak{m}-1} \mathrm{T}(\mathfrak{n}, \mathfrak{j}) + \sum_{\mathfrak{j}=\lfloor \frac{\mathfrak{m}}{2} \rfloor}^{\mathfrak{m}-1} \mathrm{T}(\mathfrak{n}, \mathfrak{j}) \right) \\ & \mid c \cdot \mathfrak{e} \mid \mathfrak{e} + \mathfrak{e} \end{aligned}$$

$$\mathfrak{h} ::= c \mid \ln \mathfrak{n} \mid \mathfrak{n} \mid \mathfrak{n} \cdot \ln \mathfrak{n} \mid c \cdot \mathfrak{h} \mid \mathfrak{h} + \mathfrak{h}$$

$$\mathfrak{b} ::= c \mid \frac{1}{\mathfrak{m}} \mid \ln \mathfrak{m} \mid \mathfrak{m} \mid \mathfrak{m} \cdot \ln \mathfrak{m} \mid c \cdot \mathfrak{b} \mid \mathfrak{b} + \mathfrak{b}$$

The differences are that (i) we have two independent parameters $\mathfrak{n}, \mathfrak{m}$, (ii) $\mathfrak{e}$ now represents an expression composed of only T-terms, and (iii) $\mathfrak{h}$ (resp. $\mathfrak{b}$) represents arithmetic expressions for $\mathfrak{n}$ (resp. for $\mathfrak{m}$). This class of separable bivariate recurrence expressions (often for brevity bivariate recurrence expressions) stresses a dominant role on $\mathfrak{n}$ and a minor role on $\mathfrak{m}$, and is intended to model randomized algorithms where some parameter (to be represented by $\mathfrak{n}$) does not change value.

Substitution. The notion of substitution is similar to the univariate case. Consider a function $h : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, and a bivariate recurrence expression $\mathfrak{e}$. The *substitution function*, denoted by $\mathsf{Sub}(\mathfrak{e}, h)$, is the function from $\mathbb{N} \times \mathbb{N}$ into $\mathbb{R}$ such that $\mathsf{Sub}(\mathfrak{e}, h)(n, m)$ is the real number evaluated through substituting h, n, m for $\mathrm{T}, \mathfrak{n}, \mathfrak{m}$, respectively. The substitution for $\mathfrak{h}, \mathfrak{b}$ is defined in a similar way, with the difference that they both induce a univariate function.

Bivariate recurrence relations. We consider *bivariate recurrence relations* $G = (\mathsf{eq}_1, \mathsf{eq}_2)$, which consists of two equalities of the following form:

$$\begin{cases} \mathsf{eq}_1 : \ \mathrm{T}(\mathfrak{n}, \mathfrak{m}) = \mathfrak{e} + \mathfrak{h} \cdot \mathfrak{b} \\ \mathsf{eq}_2 : \ \mathrm{T}(\mathfrak{n}, 1) = \mathfrak{h} \cdot c \end{cases} \tag{9}$$

where $c \in (0, \infty)$ and $\mathfrak{e}, \mathfrak{h}, \mathfrak{b}$ are from the grammar above.

Solution to bivariate recurrence relations. The evaluation of bivariate recurrence relation is similar to the univariate case. Similar to the univariate case, the unique solution $T_G : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ to a recurrence relation G taking the form (9) is a function defined recursively as follows:

- *Base Step.* $T_G(n, 1) := \mathsf{Sub}(\mathfrak{h})(n) \cdot c$ for all $n \in \mathbb{N}$;
- *Recursive Step.* $T_G(n, m) := \mathsf{Sub}(\mathfrak{e}, T_G)(n, m) + \mathsf{Sub}(\mathfrak{h})(n) \cdot \mathsf{Sub}(\mathfrak{b})(m)$ for all $n \in \mathbb{N}$ and $m \geq 2$.

Again the interesting algorithmic question is to reason about the asymptotic infinite behaviour of T_G.

2.4 Motivating Classical Examples

In this section we present two classical examples of randomized algorithms where the randomized recurrence relation belongs to the bivariate recurrence relations defined in Section 2.3.

Example 6 (COUPON-COLLECTOR)**.** *Consider the* COUPON-COLLECTOR *problem [34, Chapter 3] with n different types of coupons ($n \in \mathbb{N}$). The randomized process proceeds in rounds: at each round, a coupon is collected uniformly at random from the coupon types (i.e., each coupon type is collected with probability $\frac{1}{n}$); and the rounds continue until all the n types of coupons are collected.*

We model the rounds as a recurrence relation with two variables $\mathfrak{n}, \mathfrak{m}$, where $\mathfrak{n}$ represents the total number of coupon types and $\mathfrak{m}$ represents the remaining number of uncollected coupon types. The recurrence relation is as follows:

$$\begin{cases} \mathrm{T}(\mathfrak{n}, \mathfrak{m}) = \mathfrak{n} \cdot \frac{1}{\mathfrak{m}} + \mathrm{T}(\mathfrak{n}, \mathfrak{m} - 1) \\ \mathrm{T}(\mathfrak{n}, 1) = \mathfrak{n} \cdot 1 \end{cases} \tag{10}$$

where $\mathrm{T}(\mathfrak{n}, \mathfrak{m})$ is the expected number of rounds, $\frac{\mathfrak{n}}{\mathfrak{m}}$ represents the expected number of rounds to collect a new (i.e., not-yet-collected) coupon type when there are still $\mathfrak{m}$ type of coupons to be collected, and $\mathfrak{n}$ (for $\mathrm{T}(\mathfrak{n}, 1)$) represents the expected number of rounds to collect a new coupon type when there is only one new coupon type to be collected. We note that the worst-case complexity for this process is ∞.

Example 7 (CHANNEL-CONFLICT RESOLUTION)**.** *We consider two network scenarios in which n clients are trying to get access to a network channel. This problem is also called the* RESOURCE-CONTENTION RESOLUTION *[30, Chapter 13]. In this problem, if more than one client tries to access the channel, then no client can access it, and if exactly one client requests access to the channel, then the request is granted. While centralized deterministic algorithms exist (such as Round-Robin) for the problem, to be implemented in a distributed or concurrent setting, randomized algorithms are necessary.*

Distributed setting. *In the distributed setting, the clients do not share any information. In this scenario, in each round, every client requests an access to the channel with probability $\frac{1}{n}$. We are interested in the expected number of rounds until every client gets at least one access to the channel. At each round, let m be the number of clients who have not got any access. Then the probability that a new client (from the m clients) gets the access is $m \cdot \frac{1}{n} \cdot (1 - \frac{1}{n})^{n-1}$. Thus, the expected rounds that a new client gets the access is $\frac{n}{m} \cdot \frac{1}{(1-\frac{1}{n})^{n-1}}$. Since the sequence $\left\{(1 - \frac{1}{n})^{n-1}\right\}_{n \in \mathbb{N}}$ converges decreasingly to $\frac{1}{e}$ when $n \to \infty$, this expected time is no greater than $e \cdot \frac{n}{m}$. Then for this scenario, we obtain an over-approximating recurrence relation*

$$\begin{cases} \mathrm{T}(\mathfrak{n}, \mathfrak{m}) = \mathfrak{n} \cdot \frac{e}{\mathfrak{m}} + \mathrm{T}(\mathfrak{n}, \mathfrak{m} - 1) \\ \mathrm{T}(\mathfrak{n}, 1) = \mathfrak{n} \cdot 1 \end{cases} \tag{11}$$

for the expected rounds until which every client gets at least one access to the channel. Note that in this setting no client has any information about any other client.

Concurrent setting. *In the concurrent setting, the clients share one variable, which is the number of clients which has not yet been granted access. Also in this scenario, once a client gets an access the client does not request for access again. Moreover, the shared variable represents the number of clients m that have not yet got access. In this case, in reach round a client that has not access to the channel yet, requests access to the channel with probability $\frac{1}{m}$. Then the probability that a new client gets the access becomes $m \cdot \frac{1}{m} \cdot (1 - \frac{1}{m})^{m-1}$. It follows that the expected time that a new client gets the access becomes $\frac{1}{(1-\frac{1}{m})^{m-1}}$ which is smaller than e. Then for this scenario, we obtain an over-approximating recurrence relation*

$$\begin{cases} \mathrm{T}(\mathfrak{n}, \mathfrak{m}) = 1 \cdot e + \mathrm{T}(\mathfrak{n}, \mathfrak{m} - 1) \\ \mathrm{T}(\mathfrak{n}, 1) = 1 \cdot 1 \end{cases} . \tag{12}$$

We also note that the worst-case complexity for both is ∞.

2.5 From Programs to Recurrence Relations

The derivation of recurrence relations (as illustrated in our examples) for randomized recursive programs follows few simple main patterns. We describe these patterns below.

1. *Randomized index patterns:* A randomized index selection followed by a *pivoting* procedure w.r.t some linear order; and then followed by a sub-array/subset selection w.r.t the linear order and the element at the selected index (e.g., Example 2, Example 4); and more generally, also some input number independent of the array/set (e.g., Example 1, Example 3).
2. *Direct analysis and subproblems.* Direct analysis and solution of subcases (e.g., Example 5, where the subcases are other recurrence relations).
3. *Simple probabilistic bounds.* Simple derivations of probabilities and bounds on probabilities (e.g., Example 6, Example 7).

Since these patterns above are typical in randomized algorithms, by applying the patterns above, for a wide class of randomized recursive programs a pattern-based derivation of the recurrence relation can be easily achieved. The problem of deriving recurrence relations (such as for deterministic programs) directly from programs has already been considered in several works ([1–3, 15]). Hence, the crucial step that require automation is the analysis of the randomized recurrence relations. We will consider analyzing randomized recurrence relations algorithmically in our work.

3. Average-Case Analysis

In this work, we focus on synthesizing logarithmic, linear, and almost-linear asymptotic bounds for recurrence relations. Our goal is to decide and synthesize asymptotic bounds in the simple form as follows:

$$d \cdot \mathfrak{f} + \mathfrak{g}, \mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}.$$

Informally, $\mathfrak{f}$ is the major term for time complexity, d is the coefficient of $\mathfrak{f}$ to be synthesized, and $\mathfrak{g}$ is the time complexity for the base case specified in (1) or (9). In details, we study the following algorithmic problems, and in the algorithmic problem, w.l.o.g, we consider that every $\mathfrak{e}$ in (1) or (9) involves at least one $\mathrm{T}(\cdot)$-term and one non-$\mathrm{T}(\cdot)$-term.

Univariate Case: The univariate case problem is as follows:

- *Input:* a univariate recurrence relation G taking the form (1) and an expression $\mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}$.
- *Output: Decision problem.* Output "*yes*" if $T_G \in \mathcal{O}(\mathsf{Sub}(\mathfrak{f}))$, and "*fail*" otherwise.
- *Output: Quantitative problem.* A positive real number d such that

$$T_G(n) \leq d \cdot \mathsf{Sub}(\mathfrak{f})(n) + c \tag{13}$$

for all $n \geq 1$, or "*fail*" otherwise, where c is from (1).

Bivariate Case: The bivariate case problem is an extension of the univariate one, and hence the problem definitions are similar, and we present them succinctly below.

- *Input:* a bivariate recurrence relation G taking the form (9) and an expression $\mathfrak{f}$ (similar to the univariate case).
- *Output: Decision problem.* Output "*yes*" if $T_G \in \mathcal{O}(\mathsf{Sub}(\mathfrak{f}))$, and "*fail*" otherwise;
- *Output: Quantitative problem.* A positive real number d such that

$$T_G(n, m) \leq d \cdot \mathsf{Sub}(\mathfrak{f})(n, m) + c \cdot \mathsf{Sub}(\mathfrak{h})(n)$$

for all $n, m \geq 1$, or "*fail*" otherwise, where $c, \mathfrak{h}$ are from (9). Note that in the expression above the term $\mathfrak{b}$ does not appear as it can be captured with $\mathfrak{f}$ itself.

Recall that in the above algorithmic problems obtaining the finite behaviour of the recurrence relations is easy (through evaluation of the recurrences using dynamic programming), and the interesting aspect is to decide the asymptotic infinite behaviour.

4. The Synthesis Algorithm

In this section, we present our algorithms to synthesize asymptotic bounds for randomized recurrence relations.

Main ideas. The main idea is as follows. Consider as input a recurrence relation taking the form (1) and an univariate recurrence expression $\mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}$ which specifies the desired asymptotic bound. We first define the standard notion of a guess-and-check function in Section 4.1 which provides a sound approach for asymptotic bound. Based on the guess-and-check function, our algorithm executes the following steps for the univariate case.

1. First, the algorithm establishes a scalar variable d and then constructs the template h to be $n \mapsto d \cdot \mathsf{Sub}(\mathfrak{f})(n) + c$ for a univariate guess-and-check function.
2. Second, the algorithm computes an over-approximation $\mathsf{OvAp}(\mathfrak{e}, h)$ of $\mathsf{Sub}(\mathfrak{e}, h)$ through the techniques we develop in Sect. 4.2 such that the over-approximation $\mathsf{OvAp}(\mathfrak{e}, h)$ will involve terms from $\mathfrak{n}^k, \ln^\ell \mathfrak{n}$ ($k, \ell \in \mathbb{N}_0$) only. Note that k, ℓ may be greater than 1, so the above expressions are not necessarily linear (they can be quadratic or cubic for example).
3. Finally, the algorithm synthesizes a value for d such that $\mathsf{OvAp}(\mathfrak{e}, h)(n) \leq h(n)$ for all $n \geq 2$ through truncation of $[2, \infty) \cap \mathbb{N}$ into a finite range and a limit behaviour analysis (towards ∞).

Our algorithm for bivariate cases is a reduction to the univariate case.

4.1 Guess-and-Check Functions

We follow the standard guess-and-check technique to solve simple recurrence relations. Below we first fix a univariate recurrence relation G taking the form (1).

Definition 1 (Univariate Guess-and-Check Functions)**.** *Let G be a univariate recurrence relation taking the form (1). A function $h : \mathbb{N} \to \mathbb{R}$ is a* guess-and-check *function for G if there exists a natural number $N \in \mathbb{N}$ such that*

- (Base Condition) $T_G(n) \leq h(n)$ *for all* $1 \leq n \leq N$*, and*
- (Inductive Argument) $\mathsf{Sub}(\mathfrak{e}, h)(n) \leq h(n)$ *for all* $n > N$.

By an easy induction on n (starting from the N specified in Definition 1) we obtain the following result.

Theorem 1 (Guess-and-Check, Univariate Case)**.** *If a function $h : \mathbb{N} \to \mathbb{R}$ is a* guess-and-check *function for a univariate recurrence relation G taking the form (1), then $T_G(n) \leq h(n)$ for all $n \in \mathbb{N}$.*

We do not present explicitly present the definition for guess-and-check functions in the bivariate case, since we will present a reduction of the analysis of separable bivariate recurrence relations to that of the univariate ones (cf. Section 4.4).

4.2 Approximations for Recurrence Expressions

In this part, we develop tight approximations for logarithmic terms. In principle, we use Taylor's Theorem to approximate logarithmic terms such as $\ln(n-1), \ln \lfloor \frac{n}{2} \rfloor$, and integral to approximate summations of logarithmic terms. All the results in this section are technical and depends on basic calculus (the detailed proofs are in the Appendix A).

We have the following result using integral-by-part technique and Newton-Leibniz Formula.

Lemma 1. *For all $a, b \in (0, \infty)$ such that $a < b$, the following assertions hold:*

$$(1) \int_a^b \frac{1}{x}\,\mathrm{d}x = \ln x \Big|_a^b ;$$

$$(2) \int_a^b \ln x\,\mathrm{d}x = (x \cdot \ln x - x)\Big|_a^b ;$$

$$(3) \int_a^b x \cdot \ln x\,\mathrm{d}x = \left(\frac{1}{2} \cdot x^2 \cdot \ln x - \frac{1}{4} \cdot x^2\right)\Big|_a^b .$$

In the following lemma we provide a tight approximation for floored expressions, the proof of which is a simple case distinction between even and odd cases.

Lemma 2. *For all natural numbers n, we have*

$$\frac{n-1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2} \leq \left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2} .$$

The following lemma handles over-approximation of simple summations.

Lemma 3. *For any natural number $n \geq 2$ and real number c,*

$$\frac{\sum_{j=1}^{n-1} c}{n} \leq c \text{ and } \frac{\left(\sum_{j=\lceil \frac{n}{2} \rceil}^{n-1} c + \sum_{j=\lfloor \frac{n}{2} \rfloor}^{n-1} c\right)}{n} \leq c \ .$$

Based on Lemma 2 and Taylor's Theorem, we have the following two propositions.

Proposition 1. *For any natural number $n \geq 2$, we have:*

$$(1)\ \ln n - \ln 2 - \frac{1}{n-1} \leq \ln \left\lfloor \frac{n}{2} \right\rfloor \leq \ln n - \ln 2 \ ;$$

$$(2)\ \ln n - \ln 2 \leq \ln \left\lceil \frac{n}{2} \right\rceil \leq \ln n - \ln 2 + \frac{1}{n} \ .$$

Proposition 2. *For any natural number $n \geq 2$, we have:*

$$\ln n - \frac{1}{n-1} \leq \ln (n-1) \leq \ln n - \frac{1}{n} \ .$$

By investigating a thorough interaction between Taylor's Theorem and integral, we establish the following approximations for summation of logarithmic or reciprocal terms.

Proposition 3. *For any natural number $n \geq 2$, we have:*

$$(1)\ \int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1} \frac{1}{j} \in \left[-0.7552, -\frac{1}{6}\right] \tag{14}$$

$$(2)\ \int_1^n \ln x\,\mathrm{d}x - \left(\sum_{j=1}^{n-1} \ln j\right) - \frac{1}{2} \cdot \int_1^n \frac{1}{x}\,\mathrm{d}x \in \left[-\frac{1}{12}, 0.2701\right] \tag{15}$$

$$(3)\int_1^n x \cdot \ln x\,\mathrm{d}x - \left(\sum_{j=1}^{n-1} j \cdot \ln j\right) - \frac{1}{2} \cdot \int_1^n \ln x\,\mathrm{d}x + \frac{1}{12} \cdot \int_1^n \frac{1}{x}\,\mathrm{d}x - \frac{n-1}{2} \in \left[-\frac{19}{72}, 0.1575\right]. \tag{16}$$

From Proposition 3 and Lemma 1, we establish a tight approximation (with at most constant deviation) for summation of logarithmic or reciprocal terms.

Notations: $\Gamma_{(\cdot)}$. In the sequel, for any natural number $n \geq 2$, we use the following notation:

$$\Gamma_{\frac{1}{\mathfrak{n}}}(n) := \int_1^n \frac{1}{x}\,\mathrm{d}x = \ln n \ ;$$

$$\begin{aligned}\Gamma_{\ln \mathfrak{n}}(n) &:= \int_1^n \ln x\,\mathrm{d}x - \frac{1}{2} \cdot \int_1^n \frac{1}{x}\,\mathrm{d}x \\ &= n \cdot \ln n - n - \frac{\ln n}{2} + 1 \ ;\end{aligned}$$

$$\begin{aligned}\Gamma_{\mathfrak{n} \ln \mathfrak{n}}(n) &:= \int_1^n \left[x \cdot \ln x - \frac{\ln x}{2} + \frac{1}{12} \cdot \frac{1}{x}\right] \mathrm{d}x - \frac{n-1}{2} \\ &= \frac{n^2 \cdot \ln n}{2} - \frac{n^2}{4} - \frac{n \cdot \ln n}{2} + \frac{\ln n}{12} + \frac{1}{4} \ .\end{aligned}$$

Example 8. *Consider the summation*

$$\sum_{j=\lceil \frac{n}{2} \rceil}^{n-1} \ln j + \sum_{j=\lfloor \frac{n}{2} \rfloor}^{n-1} \ln j \quad (n \geq 4).$$

By Proposition 3, we can over-approximate it as

$$2 \cdot \left(\Gamma_{\ln \mathfrak{n}}(n) + \frac{1}{12}\right) - \left(\Gamma_{\ln \mathfrak{n}}\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Gamma_{\ln \mathfrak{n}}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) - 0.5402\right)$$

which is equal to

$$\begin{aligned}&2 \cdot n \cdot \ln n - 2 \cdot n - \ln n - \left\lceil \frac{n}{2} \right\rceil \cdot \ln \left\lceil \frac{n}{2} \right\rceil - \left\lfloor \frac{n}{2} \right\rfloor \cdot \ln \left\lfloor \frac{n}{2} \right\rfloor \\ &+ \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor + \frac{\ln \left\lfloor \frac{n}{2} \right\rfloor}{2} + \frac{\ln \left\lceil \frac{n}{2} \right\rceil}{2} + \frac{1}{6} + 0.5402.\end{aligned}$$

Then by Lemma 2 and Proposition 1, we can further obtain the following over-approximation

$$\begin{aligned}&2 \cdot n \cdot \ln n - 2 \cdot n - \ln n + 0.7069 \\ &- \frac{n}{2} \cdot (\ln n - \ln 2) - \frac{n-1}{2} \cdot \left(\ln n - \ln 2 - \frac{1}{n-1}\right) \\ &+ \frac{n+1}{2} + \frac{n}{2} + \frac{\ln n - \ln 2}{2} + \frac{\ln n - \ln 2 + \frac{1}{n}}{2}\end{aligned}$$

which is roughly

$$n \cdot \ln n - (1 - \ln 2) \cdot n + \frac{1}{2} \cdot \ln n + 0.6672 + \frac{1}{2 \cdot n}.$$

Remark 1. *We remark that although we do approximation for terms appearing in our recurrence relations only, our techniques are more general and can be applied to more general terms (e.g., a summation of logarithmic terms from $\lfloor \frac{n}{3} \rfloor$ to $\lceil \frac{n}{2} \rceil$).*

4.3 Algorithm for Univariate Recurrence Relations

In this section, we present our algorithm to synthesize a guess-and-check function in form (13) for univariate recurrence relations. Due to space restrictions some technical details are relegated to Appendix B.

We present our algorithm in two steps. First, we present the decision version, and then we present the quantitative version that synthesizes the associated constant. The two key aspects are over-approximation and use of pseudo-polynomials, and we start with over-approximation.

Definition 2 (Over-approximation)**.** *Let $\mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}$. Consider a univariate recurrence expression $\mathfrak{g}$, constants d and c, and the function $h = d \cdot \mathsf{Sub}(\mathfrak{f}) + c$. We define the* over-approximation function*, denoted $\mathsf{OvAp}(\mathfrak{g}, h)$, recursively as follows.*

- Base Step A. *If $\mathfrak{g}$ is one of the following:*

  $$c', \quad \mathfrak{n}, \quad \ln \mathfrak{n}, \quad \mathfrak{n} \cdot \ln \mathfrak{n}, \frac{1}{\mathfrak{n}}$$

  then $\mathsf{OvAp}(\mathfrak{g}, h) := \mathsf{Sub}(\mathfrak{g})$.

- Base Step B. *If $\mathfrak{g}$ is a single term which involves* T*, then we define $\mathsf{OvAp}(\mathfrak{g}, h)$ from over-approximations given in Lemma 2, Lemma 3 and Proposition 1– 3. In details, $\mathsf{OvAp}(\mathfrak{g}, h)$ is obtained from $\mathsf{Sub}(\mathfrak{g}, h)$ by first over-approximating any summation through Proposition 3 and Lemma 1 (i.e., through those $\Gamma_{(\cdot)}$ functions defined below Proposition 3), then over-approximating any*

  $$\ln (\mathfrak{n} - 1), \left\lfloor \frac{\mathfrak{n}}{2} \right\rfloor, \left\lceil \frac{\mathfrak{n}}{2} \right\rceil, \ln \left\lfloor \frac{\mathfrak{n}}{2} \right\rfloor, \ln \left\lceil \frac{\mathfrak{n}}{2} \right\rceil$$

  by Proposition 1 and Proposition 2. The details of the important over-approximations are illustrated explicitly in Table 1.

- Recursive Step. *We have two cases: (a) If $\mathfrak{g}$ is $\mathfrak{g}_1 + \mathfrak{g}_2$, then $\mathsf{OvAp}(\mathfrak{g}, h)$ is $\mathsf{OvAp}(\mathfrak{g}_1, h) + \mathsf{OvAp}(\mathfrak{g}_2, h)$. (b) If $\mathfrak{g}$ is $c' \cdot \mathfrak{g}'$, then $\mathsf{OvAp}(\mathfrak{g}, h)$ is $c' \cdot \mathsf{OvAp}(\mathfrak{g}', h)$.*

$\mathfrak{f}$, T-term	Over-approximation
$\ln \mathfrak{n}$, $\mathfrak{e}_1$	$\ln \mathfrak{n} - \frac{1}{\mathfrak{n}}$
$\ln \mathfrak{n}$, $\mathfrak{e}_2$	$\ln \mathfrak{n} - \ln 2$
$\ln \mathfrak{n}$, $\mathfrak{e}_3$	$\ln \mathfrak{n} - \ln 2 + \frac{1}{\mathfrak{n}}$
$\ln \mathfrak{n}$, $\mathfrak{e}_4$	$\ln \mathfrak{n} - 1 - \frac{\ln \mathfrak{n}}{2\cdot\mathfrak{n}} + \frac{13}{12}\cdot\frac{1}{\mathfrak{n}}$
$\ln \mathfrak{n}$, $\mathfrak{e}_5$	$\ln \mathfrak{n} - (1 - \ln 2) + \frac{\ln \mathfrak{n}}{2\cdot\mathfrak{n}} + \frac{0.6672}{\mathfrak{n}} + \frac{1}{2\cdot\mathfrak{n}^2}$
$\mathfrak{f}$, T-term	Over-approximation
$\mathfrak{n}$, $\mathfrak{e}_1$	$\mathfrak{n} - 1$
$\mathfrak{n}$, $\mathfrak{e}_2$	$\frac{\mathfrak{n}}{2}$
$\mathfrak{n}$, $\mathfrak{e}_3$	$\frac{\mathfrak{n}+1}{2}$
$\mathfrak{n}$, $\mathfrak{e}_4$	$\frac{\mathfrak{n}-1}{2}$
$\mathfrak{n}$, $\mathfrak{e}_5$	$\frac{3}{4}\cdot\mathfrak{n} - \frac{1}{4\cdot\mathfrak{n}}$
$\mathfrak{f}$, T-term	Over-approximation
$\mathfrak{n}\cdot\ln\mathfrak{n}$, $\mathfrak{e}_1$	$\mathfrak{n}\cdot\ln\mathfrak{n} - \ln\mathfrak{n} - 1 + \frac{1}{\mathfrak{n}}$
$\mathfrak{n}\cdot\ln\mathfrak{n}$, $\mathfrak{e}_2$	$\frac{1}{2}\cdot\mathfrak{n}\cdot\ln\mathfrak{n} - \frac{\ln 2}{2}\cdot\mathfrak{n}$
$\mathfrak{n}\cdot\ln\mathfrak{n}$, $\mathfrak{e}_3$	$\frac{\mathfrak{n}\cdot\ln\mathfrak{n}}{2} - \frac{\ln 2}{2}\cdot\mathfrak{n} + \frac{1-\ln 2}{2} + \frac{\ln\mathfrak{n}}{2} + \frac{1}{2\cdot\mathfrak{n}}$
$\mathfrak{n}\cdot\ln\mathfrak{n}$, $\mathfrak{e}_4$	$\frac{\mathfrak{n}\cdot\ln\mathfrak{n}}{2} - \frac{\mathfrak{n}}{4} - \frac{\ln\mathfrak{n}}{2} + \frac{\ln\mathfrak{n}}{12\cdot\mathfrak{n}} + \frac{0.5139}{\mathfrak{n}}$
$\mathfrak{n}\cdot\ln\mathfrak{n}$, $\mathfrak{e}_5$	$\frac{3}{4}\cdot\mathfrak{n}\cdot\ln\mathfrak{n} - 0.2017\cdot\mathfrak{n} - \frac{1}{2}\cdot\ln\mathfrak{n}$ $-0.2698 + \frac{\ln\mathfrak{n}}{8\cdot\mathfrak{n}} + \frac{1.6369}{\mathfrak{n}} + \frac{1}{2\cdot\mathfrak{n}\cdot(\mathfrak{n}-1)} + \frac{1}{4\cdot\mathfrak{n}^2}$

Table 1. Illustration for Definition 2, where
- $\mathfrak{e}_1 := \mathrm{T}(\mathfrak{n}-1)$;
- $\mathfrak{e}_2 := \mathrm{T}\left(\left\lfloor\frac{\mathfrak{n}}{2}\right\rfloor\right)$;
- $\mathfrak{e}_3 := \mathrm{T}\left(\left\lceil\frac{\mathfrak{n}}{2}\right\rceil\right)$;
- $\mathfrak{e}_4 := \frac{1}{\mathfrak{n}}\cdot\sum_{\mathfrak{j}=1}^{\mathfrak{n}-1}\mathrm{T}(\mathfrak{j})$; and
- $\mathfrak{e}_5 := \frac{1}{\mathfrak{n}}\cdot\left(\sum_{\mathfrak{j}=\lceil\frac{\mathfrak{n}}{2}\rceil}^{\mathfrak{n}-1}\mathrm{T}(\mathfrak{j}) + \sum_{\mathfrak{j}=\lfloor\frac{\mathfrak{n}}{2}\rfloor}^{\mathfrak{n}-1}\mathrm{T}(\mathfrak{j})\right)$.

Example 9. *Consider the recurrence relation for Sherwood's* RANDOMIZED-SEARCH *(cf. (3)). Choose* $\mathfrak{f} = \ln\mathfrak{n}$ *and then the template* h *becomes* $n \mapsto d\cdot\ln n + 1$*. From Example 8, we have that the over-approximation for*

$$6 + \frac{1}{\mathfrak{n}}\cdot\left(\sum_{\mathfrak{j}=\lceil\frac{\mathfrak{n}}{2}\rceil}^{\mathfrak{n}-1}\mathrm{T}(\mathfrak{j}) + \sum_{\mathfrak{j}=\lfloor\frac{\mathfrak{n}}{2}\rfloor}^{\mathfrak{n}-1}\mathrm{T}(\mathfrak{j})\right)$$

when $n \geq 4$ *is*

$$7 + d\cdot\left[\ln n - (1-\ln 2) + \frac{\ln n}{2\cdot n} + \frac{0.6672}{n} + \frac{1}{2\cdot n^2}\right]$$

where the second summand comes from an over-approximation of

$$\frac{1}{\mathfrak{n}}\cdot\left(\sum_{\mathfrak{j}=\lceil\frac{\mathfrak{n}}{2}\rceil}^{\mathfrak{n}-1} d\cdot\ln\mathfrak{j} + \sum_{\mathfrak{j}=\lfloor\frac{\mathfrak{n}}{2}\rfloor}^{\mathfrak{n}-1} d\cdot\ln\mathfrak{j}\right).$$

Pseudo-polynomials. Our next step is to define the notion of (univariate) pseudo-polynomials which extends normal polynomials with logarithm. This notion will be crucial to handle inductive arguments in the definition of (univariate) guess-and-check functions.

Definition 3 (Univariate Pseudo-polynomials)**.** *A univariate pseudo-polynomial (w.r.t logarithm) is a function* $p : \mathbb{N} \to \mathbb{R}$ *such that there exist non-negative integers* $k, \ell \in \mathbb{N}_0$ *and real numbers* a_i, b_i*'s such that for all* $n \in \mathbb{N}$*,*

$$p(n) = \sum_{i=0}^{k} a_i\cdot n^i\cdot\ln n + \sum_{i=0}^{\ell} b_i\cdot n^i. \tag{17}$$

W.l.o.g, we consider that in the form (17), it holds that (i) $a_k^2 + b_\ell^2 \neq 0$, (ii) either $a_k \neq 0$ or $k = 0$, and (iii) similarly either $b_\ell \neq 0$ or $\ell = 0$.

Degree of pseudo-polynomials. Given a univariate pseudo-polynomial p in the form (17), we define the *degree* $\deg(p)$ of p by:

$$\deg(p) = \begin{cases} k + \frac{1}{2} & \text{if } k \geq \ell \text{ and } a_k \neq 0 \\ \ell & \text{otherwise} \end{cases},$$

Intuitively, if the term with highest degree involves logarithm, then we increase the degree by $1/2$, else it is the power of the highest degree term.

Leading term $\overline{p}$*.* The *leading term* $\overline{p}$ of a pseudo-polynomial p in the form (17) is a function $\overline{p} : \mathbb{N} \to \mathbb{R}$ defined as follows:

$$\overline{p}(n) = \begin{cases} a_k\cdot n^k\cdot\ln n & \text{if } k \geq \ell \text{ and } a_k \neq 0 \\ b_\ell\cdot n^\ell & \text{otherwise} \end{cases}$$

for all $n \in \mathbb{N}$. Furthermore, we define C_p to be the (only) coefficient of $\overline{p}$.

With the notion of pseudo-polynomials, the inductive argument of guess-and-check functions can be soundly transformed into an inequality between pseudo-polynomials.

Lemma 4. *Let* $\mathfrak{f} \in \{\ln\mathfrak{n}, \mathfrak{n}, \mathfrak{n}\cdot\ln\mathfrak{n}\}$ *and* c *be a constant. For all univariate recurrence expressions* $\mathfrak{g}$*, there exists pseudo-polynomials* p *and* q *such that coefficients (i.e.,* a_i, b_i*'s in (17)) of* q *are all non-negative,* $C_q > 0$ *and the following assertion holds: for all* $d > 0$ *and for all* $n \geq 2$*, with* $h = d\cdot\mathsf{Sub}(\mathfrak{f}) + c$*, the inequality*

$$\mathsf{OvAp}(\mathfrak{g}, h)(n) \leq h(n) \tag{18}$$

is equivalent to

$$d\cdot p(n) \geq q(n). \tag{19}$$

Remark 2. *In the above lemma, though we only refer to existence of pseudo-polynomials* p *and* q*, they can actually be computed in linear time, because* p *and* q *are obtained by simple rearrangements of terms from* $\mathsf{OvAp}(\mathfrak{g}, h)$ *and* h*, respectively.*

Example 10. *Let us continue with Sherwood's* RANDOMIZED-SEARCH*. Again choose* $h = d\cdot\ln\mathfrak{n} + 1$*. From Example 9, we obtain that for every* $n \geq 4$*, the inequality*

$$d\cdot\ln n + 1 \geq 7 + d\cdot\left[\ln n - (1-\ln 2) + \frac{\ln n}{2\cdot n} + \frac{0.6672}{n} + \frac{1}{2\cdot n^2}\right]$$

resulting from over-approximation and the inductive argument of guess-and-check functions is equivalent to

$$d\cdot\left[(1-\ln 2)\cdot n^2 - \frac{n\cdot\ln n}{2} - 0.6672\cdot n - \frac{1}{2}\right] \geq 6\cdot n^2.$$

As is indicated in Definition 1, our aim is to check whether (18) holds for sufficiently large n. The following proposition provides a sufficient and necessary condition for checking whether (19) holds for sufficiently large n.

Proposition 4. *Let* p, q *be pseudo-polynomials such that* $C_q > 0$ *and all coefficients of* q *are non-negative. Then there exists a real number* $d > 0$ *such that* $d\cdot p(n) \geq q(n)$ *for sufficiently large* n *iff* $\deg(p) \geq \deg(q)$ *and* $C_p > 0$*.*

Note that by Definition 1 and the special form (13) for univariate guess-and-check functions, a function in form (13) needs only to satisfy the inductive argument in order to be a univariate guess-and-check function: once a value for d is synthesized for a sufficiently large N, one can scale the value so that the base condition

is also satisfied. Thus from the sufficiency of Proposition 4, our decision algorithm that checks the existence of some guess-and-check function in form (13) is presented below. Below we fix an input univariate recurrence relation G taking the form (1) and an input expression $\mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}$.

Algorithm *UniDec*: Our algorithm, namely *UniDec*, for the decision problem of the univariate case, has the following steps.

1. *Template.* The algorithm establishes a scalar variable d and sets up the template $d \cdot \mathfrak{f} + c$ for a univariate guess-and-check function.

2. *Over-approximation.* Let h denote $d \cdot \mathsf{Sub}(\mathfrak{f}) + c$. The algorithm calculates the over-approximation function $\mathsf{OvAp}(\mathfrak{e}, h)$, where $\mathfrak{e}$ is from (1).

3. *Transformation.* The algorithm transforms the inequality
$$\mathsf{OvAp}(\mathfrak{e}, h)(n) \leq h(n) \ \ (n \in \mathbb{N})$$
for inductive argument of guess-and-check functions through Lemma 4 equivalently into
$$d \cdot p(n) \geq q(n) \ \ (n \in \mathbb{N}),$$
where p, q are pseudo-polynomials obtained in linear-time through rearrangement of terms from $\mathsf{OvAp}(\mathfrak{e}, h)$ and h (see Remark 2).

4. *Coefficient Checking.* The algorithm examines cases on C_p. If $C_p > 0$ and $\deg(p) \geq \deg(q)$, then algorithm outputs "*yes*" meaning that "there exists a univariate guess-and-check function"; otherwise, the algorithm outputs "*fail*".

Theorem 2 (Soundness for *UniDec*)**.** *If UniDec outputs "yes", then there exists a univariate guess-and-check function in form (13) for the inputs G and $\mathfrak{f}$. The algorithm is a linear-time algorithm.*

Example 11. *Consider the recurrence relation for Sherwood's* RANDOMIZED-SEARCH *(cf. (3)) and $\mathfrak{f} = \ln \mathfrak{n}$ as the input. From Example 9 and Example 10, the algorithm directly asserts that the asymptotic behaviour of Sherwood's* RANDOMIZED-SEARCH *is $\mathcal{O}(\ln n)$.*

Remark 3. *From the tightness of our over-approximation (up to only constant deviation) and the sufficiency and necessity of Proposition 4, the UniDec algorithm can handle a large class of univariate recurrence relations. Moreover, the algorithm is quite simple and efficient (linear-time).*

Analysis of examples of Section 2.2. Our algorithm can decide the following bounds for the examples of Section 2.2.

1. For Example 1 we obtain an $\mathcal{O}(\log n)$ bound, whereas the worst-case bound is $\Theta(n)$.

2. For Example 2 we obtain an $\mathcal{O}(n \cdot \log n)$ bound, whereas the worst-case bound is $\Theta(n^2)$.

3. For Example 3 we obtain an $\mathcal{O}(n)$ bound, whereas the worst-case bound is $\Theta(n^2)$.

4. For Example 4 we obtain an $\mathcal{O}(n \cdot \log n)$ (resp. $\mathcal{O}(n)$) bound for Euclidean metric (resp. for L_1 metric), whereas the worst-case bound is $\Theta(n^2 \cdot \log n)$ (resp. $\Theta(n^2)$).

5. For Example 5 we obtain an $\mathcal{O}(n \cdot \log n)$ bound, whereas the worst-case bound is $\Theta(n^2)$.

In all cases above, our algorithm decides the asymptotically optimal bounds for the average-case analysis, whereas the worst-case analysis grossly over-estimate the average-case bounds.

Quantitative bounds. Above we have already established that our linear-time decision algorithm can establish the asymptotically optimal bounds for the recurrence relations of several classical algorithms. We now take the next step to obtain even explicit quantitative bounds, i.e., to synthesize the associated constants with the asymptotic complexity. To tackle these situations, we derive a following proposition which gives explicitly a threshold for "sufficiently large numbers". We first explicitly constructs a threshold for "sufficiently large numbers".

Definition 4 (Threshold $N_{\epsilon,p,q}$ for Sufficiently Large Numbers)**.** *Let p, q be two univariate pseudo-polynomials*
$$p(n) = \sum_{i=0}^{k} a_i \cdot n^i \cdot \ln n + \sum_{i=0}^{\ell} b_i \cdot n^i \ ,$$
$$q(n) = \sum_{i=0}^{k'} a'_i \cdot n^i \cdot \ln n + \sum_{i=0}^{\ell'} b'_i \cdot n^i$$
such that $\deg(p) \geq \deg(q)$ and $C_p, C_q > 0$. Then given any $\epsilon \in (0,1)$, the number $N_{\epsilon,p,q}$ is defined as the smallest natural number such that both x, y (defined below) is smaller than ϵ:
$$x = -1 + \sum_{i=0}^{k} |a_i| \cdot \frac{N^i \cdot \ln N}{\overline{p}(N)} + \sum_{i=0}^{\ell} |b_i| \cdot \frac{N^i}{\overline{p}(N)}$$
$$y = -\mathbf{1}_{\deg(p)=\deg(q)} \cdot \frac{C_q}{C_p} + \sum_{i=0}^{k'} |a'_i| \cdot \frac{N^i \cdot \ln N}{\overline{p}(N)} + \sum_{i=0}^{\ell'} |b'_i| \cdot \frac{N^i}{\overline{p}(N)}$$

Then we show that $N_{\epsilon,p,q}$ is indeed what we need.

Proposition 5. *Consider two univariate pseudo-polynomials p, q such that $\deg(p) \geq \deg(q)$, all coefficients of q are non-negative and $C_p, C_q > 0$. Then given any $\epsilon \in (0,1)$,*
$$\frac{q(n)}{p(n)} \leq \frac{\mathbf{1}_{\deg(p)=\deg(q)} \cdot \frac{C_q}{C_p} + \epsilon}{1 - \epsilon}$$
for all $n \geq N_{\epsilon,p,q}$ (for $N_{\epsilon,p,q}$ of Definition 4).

With Proposition 5, we describe our algorithm *UniSynth* which outputs explicitly a value for d (in (13)) if *UniDec* outputs yes. Below we fix an input univariate recurrence relation G taking the form (1) and an input expression $\mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}$. Moreover, the algorithm takes $\epsilon > 0$ as another input, which is basically a parameter to choose the threshold for finite behaviour. For example, smaller ϵ leads to large threshold, and vice-versa. Thus the algorithm we provide is a flexible one as the threshold can be varied with the choice of ϵ.

Algorithm *UniSynth*: Our synthesis algorithm for the quantitative problem has the following steps:

1. *Calling UniDec.* The algorithm calls *UniDec*, and if it returns "*fail*", then return "*fail*", otherwise execute the following steps. Obtain the following inequality
$$d \cdot p(n) \geq q(n) \ \ (n \in \mathbb{N})$$
from the transformation step of *UniDec*.

2. *Variable Solving.* The algorithm calculates $N_{\epsilon,p,q}$ for a given $\epsilon \in (0,1)$ (see Definition 4) and outputs the value of d as the least number such that the following two conditions hold: (i) for all $2 \leq n < N_{\epsilon,p,q}$
$$\mathsf{Eval}(G)(n) \leq d \cdot \mathsf{Sub}(\mathfrak{f})(n) + c$$

(recall $\mathsf{Eval}(G)(n)$ can be computed in linear time), and (ii) we have

$$d \geq \frac{\mathbf{1}_{\deg(p)=\deg(q)} \cdot \frac{C_q}{C_p} + \epsilon}{1-\epsilon}.$$

Theorem 3 (Soundness for *UniSynth*)**.** *If the algorithm UniSynth outputs a real number d, then $d \cdot \mathsf{Sub}(\mathfrak{f}) + c$ is a univariate guess-and-check function for G.*

Remark 4. *Note that even our quantitative algorithm requires quite simple computational steps, i.e., it is based on comparisons, and does not require any optimization (such as linear programming) procedure.*

Example 12. *Consider the recurrence relation for Sherwood's* RANDOMIZED-SEARCH *(cf. (3)) and $\mathfrak{f} = \ln \mathfrak{n}$ as the input. Consider that $\epsilon := 0.9$. From Example 9 and Example 10, the algorithm establishes the inequality*

$$d \geq \frac{6}{(1-\ln 2) - \frac{\ln n}{2 \cdot n} - \frac{0.6672}{n} - \frac{1}{2 \cdot n^2}}$$

and finds that $N_{0.9,p,q} = 6$. Then the algorithm finds $d = 204.5335$ through the followings:

- $\mathsf{Eval}(G)(2) = 7 \leq d \cdot \ln 2 + 1$*;*
- $\mathsf{Eval}(G)(3) = 11 \leq d \cdot \ln 3 + 1$*;*
- $\mathsf{Eval}(G)(4) = 15 \leq d \cdot \ln 4 + 1$*;*
- $\mathsf{Eval}(G)(5) = 17.8 \leq d \cdot \ln 5 + 1$*;*
- $d \geq \frac{\frac{6}{1-\ln 2}+0.9}{1-0.9}$.

Thus, by Theorem 1, the expected running time of Sherwood's Randomized-Search has an upper bound $204.5335 \cdot \ln n + 1$. Later in Section 5, we show that one can obtain a much better $d = 19.762$ through our algorithms by choosing $\epsilon := 0.01$, which is quite good since the optimal value lies somewhere in $[15.129, 19.762]$ (cf. the first item R.-SEAR. *in Table 2).*

4.4 Algorithm for Bivariate Recurrence Relations

In this part, we present our results for the separable bivariate recurrence relations. The key idea is to use separability to reduce the problem to univariate recurrence relations. There are two key steps which we describe below.

Step 1. The first step is to reduce a separable bivariate recurrence relation to a univariate one.

Definition 5 (From G to $\mathsf{Uni}(G)$)**.** *Let G be a separable bivariate recurrence relation taking the form (9). The univariate recurrence relation $\mathsf{Uni}(G)$ from G is defined by eliminating any occurrence of $\mathfrak{n}$ and replacing any occurrence of $\mathfrak{h}$ with 1.*

Informally, $\mathsf{Uni}(G)$ is obtained from G by simply eliminating the roles of $\mathfrak{h}, \mathfrak{n}$. The following example illustrates the situation for COUPON-COLLECTOR example.

Example 13. *Consider G to be the recurrence relation (10) for* COUPON-COLLECTOR *example. Then $\mathsf{Uni}(G)$ is as follows:*

$$\begin{cases} \mathrm{T}(\mathfrak{n}) = \frac{1}{\mathfrak{n}} + \mathrm{T}(\mathfrak{n}-1) \\ \mathrm{T}(1) = 1 \end{cases}.$$

Step 2. The second step is to establish the relationship between T_G and $T_{\mathsf{Uni}(G)}$, which is handled by the following proposition, whose proof is an easy induction on m.

Proposition 6. *For any separable bivariate recurrence relation G taking the form (9), the solution T_G is equal to $(n, m) \mapsto \mathsf{Sub}(\mathfrak{h})(n) \cdot T_{\mathsf{Uni}(G)}(m)$.*

Description of the Algorithm. With Proposition 6, the algorithm for separable bivariate recurrence relations is straightforward: simply compute $\mathsf{Uni}(G)$ for G and then call the algorithms for univariate case presented in Section 4.3.

Analysis of examples in Section 2.4. Our algorithm can decide the following bounds for the examples of Section 2.4.

1. For Example 6 we obtain an $\mathcal{O}(n \cdot \log m)$ bound, whereas the worst-case bound is ∞.
2. For Example 7 we obtain an $\mathcal{O}(n \cdot \log m)$ bound for distributed setting and $\mathcal{O}(m)$ bound for concurrent setting, whereas the worst-case bounds are both ∞.

Note that for all our examples, $m \leq n$, and thus we obtain $\mathcal{O}(n \cdot \log n)$ and $\mathcal{O}(n)$ upper bounds for average-case analysis, which are the asymptotically optimal bounds. In all cases above, the worst-case analysis is completely ineffective as the worst-case bounds are infinite. Moreover, consider Example 7, where the optimal number of rounds is n (i.e., one process every round, which centralized Round-Robin schemes can achieve). The randomized algorithm, with one shared variable, is a decentralized algorithm that achieves $O(n)$ expected number of rounds (i.e., the optimal asymptotic average-case complexity).

5. Experimental Results

We consider the classical examples illustrated in Section 2.2 and Section 2.4. In Table 2 for experimental results we consider the following recurrence relations G:

1. R.-SEAR. corresponds to the recurrence relation (3) for Example 1;
2. Q.-SORT corresponds to the recurrence relation (4) for Example 2;
3. Q.-SELECT corresponds to the recurrence relation (5) for Example 3;
4. DIAM. A (resp. DIAM. B) corresponds to the recurrence relation (6) (resp. the recurrence relation (7)) for Example 4;
5. SORT-SEL. corresponds to recurrence relation (8) for Example 5, where we use the result from setting $\epsilon = 0.01$ in Q.-SELECT;
6. COUPON corresponds to the recurrence relation (10) for Example 6;
7. RES. A (resp. RES. B) corresponds to the recurrence relation (11) (resp. the recurrence relation (12)) for Example 7.

In the table, $\mathfrak{f}$ specifies the input asymptotic bound, ϵ and Dec is the input which specifies either we use algorithm *UniDec* or the synthesis algorithm *UniSynth* with the given ϵ value, and d gives the value synthesized w.r.t the given ϵ (✓ for *yes*). We describe d_{100} below. Also we need approximation for constants such as e and $\ln 2$, and we use the interval $[2.7182, 2.7183]$ for tight approximation of e and $[0.6931, 0.6932]$ for tight approximation of $\ln 2$.

The value d_{100}. For our synthesis algorithm we obtain the value d. The optimal value of the associated constant with the asymptotic bound, denoted d^*, is defined as follows. For $z \geq 2$, let

$$d_z := \max \left\{ \frac{T_G(n) - c}{\mathsf{Sub}(\mathfrak{f})(n)} \mid 2 \leq n \leq z \right\}$$

(c is from (1)). Then the sequence d_z is increasing in z, and its limit is the optimal constant, i.e., $d^* = \lim_{z \to \infty} d_z$. We consider d_{100} as a lower bound on d^* to compare against the value of d we synthesize. In other words, d_{100} is the minimal value such that (13) holds for $1 \leq n \leq 100$, whereas for d^* it must hold for all n, and

Recur. Rel.	f	ϵ, *Dec*	d	d_{100}
R.-SEAR.	$\ln \mathfrak{n}$	*UniDec*	✓	15.129
		0.5	40.107	
		0.3	28.363	
		0.1	21.838	
		0.01	19.762	
Q.-SORT	$\mathfrak{n} \cdot \ln \mathfrak{n}$	*UniDec*	✓	3.172
		0.5	9.001	
		0.3	6.143	
		0.1	4.556	
		0.01	4.051	
Q.-SELECT	$\mathfrak{n}$	*UniDec*	✓	7.909
		0.5	17.001	
		0.3	11.851	
		0.1	9.001	
		0.01	8.091	
DIAM. A	$\mathfrak{n} \cdot \ln \mathfrak{n}$	*UniDec*	✓	4.525
		0.5	9.001	
		0.3	6.143	
		0.1	4.556	
		0.01	4.525	
DIAM. B	$\mathfrak{n}$	*UniDec*	✓	5.918
		0.5	13.001	
		0.3	9.001	
		0.1	6.778	
		0.01	6.071	
SORT-SEL.	$\mathfrak{n} \cdot \ln \mathfrak{n}$	*UniDec*	✓	16.000
		0.5	50.052	
		0.3	24.852	
		0.1	17.313	
		0.01	16.000	
COUPON	$\mathfrak{n} \cdot \ln \mathfrak{m}$	*UniDec*	✓	0.910
		0.5	3.001	
		0.3	1.858	
		0.1	1.223	
		0.01	1.021	
RES. A	$\mathfrak{n} \cdot \ln \mathfrak{m}$	*UniDec*	✓	2.472
		0.5	6.437	
		0.3	4.312	
		0.1	3.132	
		0.01	2.756	
RES. B	$\mathfrak{m}$	*UniDec*	✓	2.691
		0.5	6.437	
		0.3	4.312	
		0.1	3.132	
		0.01	2.756	

Table 2. Experimental results where all running times (averaged over 5 runs) are less than 0.02 seconds, between 0.01 and 0.02 in all cases.

hence $d^* \geq d_{100}$. Our experimental results show that the d values we synthesize for $\epsilon = 0.01$ is quite close to the optimal value.

We performed our experiments on Intel(R) Core(TM) i7-4510U CPU, 2.00GHz, 8GB RAM. All numbers in Table 2 are over-approximated up to 10^{-3}, and the running time of all experiments are less than 0.02 seconds. From Table 2, we can see that optimal d are effectively over-approximated. For example, for QUICK-SORT (Eq. (4)) (i.e, Q.-SORT in the table), our algorithm detects $d = 4.051$ and the optimal one lies somewhere in $[3.172, 4.051]$. The experimental results show that we obtain the results extremely efficiently (less than $1/50$-th of a second). For further experimental result details see Table 3 in Appendix C.

6. Related Work

The problem of deriving worst-case bounds have received a lot of attention. The following works consider various approaches for automated worst-case bounds, [20–22, 24–26, 28, 29] for amortized analysis, and the SPEED project [16–18] for non-linear bounds using abstract interpretation. All these works focus on the worst-case analysis, and do not consider average-case analysis.

Our main contribution is automated analysis of recurrence relations. Approaches for recurrence relations has also been considered in the literature, such as Grobauer [15] for generating recurrence relations from DML for the worst-case analysis, Flajolet *et al.* [14] for allocation problems, and Flajolet *et al.* [13] for solving recurrence relations for randomization of combinatorial structures (such as trees) through generating functions. Moreover, the COSTA project [1–3] transforms Java bytecode into recurrence relations and solves them through ranking functions. Our approach is quite different because we consider recurrence relations arising from randomized algorithms and average-case analysis, whereas previous approaches consider recurrence relations either for worst-case bounds or for combinatorial structures.

For intraprocedural analysis ranking functions have been widely studied [5, 6, 9, 11, 36–38, 40], which have then been extended to non-recursive probabilistic programs as ranking supermartingales [7, 8, 12]. Such approaches are related to almost-sure termination, and not deriving average-case analysis. Moreover, such approaches cannot produce bounds such as $\mathcal{O}(\log n)$ or $\mathcal{O}(n \log n)$.

Proof rules have also been considered for recursive (probabilistic) programs in [19, 27, 35], but these methods cannot be automated and require manual proofs.

7. Conclusion

Discussion. In this work we consider the problem of average-case analysis of randomized recursive programs. First, we present a sound and efficient (linear-time) method to derive logarithmic, linear, and almost-linear bounds for randomized recurrence relations for univariate recurrence relations. Our approach is applicable to a large class of classical textbook algorithms, such as QUICK-SORT, QUICK-SELECT, RANDOMIZED-SEARCH etc. Note that like Master Theorem, the univariate recurrence relations is already applicable to a large class of programs. Our second contribution is that we consider a subclass of bivariate recurrence relations, namely, separable bivariate recurrence relations, and show that this subclass can model recurrence relations of classical processes such as COUPON-COLLECTOR, CHANNEL-CONFLICT RESOLUTION. For this subclass we again present an efficient algorithm by a simple reduction to the univariate case, and thereby establish that the subclass is both expressive and efficiently analyzable.

Future work. Our work gives rise to a number of interesting questions. First, an interesting theoretical direction of future work would be to consider more general randomized recurrence relations (such as with more than two variables, or interaction between the variables). While the above problem is of theoretical interest, most interesting examples are already captured in our class of randomized recurrence relations as mentioned above. Another interesting practical direction of future work would be automated techniques to derive recurrence relations from randomized recursive programs.

References

[1] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost analysis of java bytecode. *ESOP 2007*, *Lecture Notes in Computer Science*, pages 157–172. Springer, 2007.

[2] E. Albert, P. Arenas, S. Genaim, and G. Puebla. Automatic inference of upper bounds for recurrence relations in cost analysis. *SAS 2008*, volume 5079 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2008.

[3] E. Albert, P. Arenas, S. Genaim, M. Gómez-Zamalloa, G. Puebla, D. V. Ramírez-Deantes, G. Román-Díez, and D. Zanardini. Termination and cost analysis with COSTA and its user interfaces. *Electr. Notes Theor. Comput. Sci.*, 258(1):109–121, 2009.

[4] R. G. Bartle and D. R. Sherbert. *Introduction to Real Analysis*. John Wiley & Sons, Inc., 4th edition, 2011.

[5] O. Bournez and F. Garnier. Proving positive almost-sure termination. In *RTA*, pages 323–337, 2005.

[6] A. R. Bradley, Z. Manna, and H. B. Sipma. Linear ranking with reachability. *CAV 2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 491–504. Springer, 2005.

[7] A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 511–526. Springer, 2013.

[8] K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. *POPl 2016*, pages 327–342. ACM, 2016.

[9] M. Colón and H. Sipma. Synthesis of linear ranking functions. *TACAS 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2001.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009. ISBN 978-0-262-03384-8. URL `http://mitpress.mit.edu/books/introduction-algorithms`.

[11] P. Cousot. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. *VMCAI 2005*, volume 3385 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2005.

[12] L. M. F. Fioriti and H. Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. *POPL 2015*, pages 489–501. ACM, 2015.

[13] P. Flajolet, B. Salvy, and P. Zimmermann. Automatic average-case analysis of algorithm. *Theor. Comput. Sci.*, 79(1):37–109, 1991.

[14] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[15] B. Grobauer. Cost recurrences for DML programs. *ICFP 2001*, pages 253–264. ACM, 2001.

[16] B. S. Gulavani and S. Gulwani. A numerical abstract domain based on expression abstraction and max operator with application in timing analysis. *CAV 2008*, volume 5123 of *Lecture Notes in Computer Science*, pages 370–384. Springer, 2008.

[17] S. Gulwani. SPEED: symbolic complexity bound analysis. *CAV 2009*, volume 5643 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2009.

[18] S. Gulwani, K. K. Mehra, and T. M. Chilimbi. SPEED: precise and efficient static estimation of program computational complexity. *POPL 2009*, pages 127–139. ACM, 2009.

[19] W. H. Hesselink. Proof rules for recursive procedures. *Formal Asp. Comput.*, 5(6):554–570, 1993.

[20] J. Hoffmann and M. Hofmann. Amortized resource analysis with polymorphic recursion and partial big-step operational semantics. *APLAS 2010*, volume 6461 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2010.

[21] J. Hoffmann and M. Hofmann. Amortized resource analysis with polynomial potential. *ESOP 2010*, volume 6012 of *Lecture Notes in Computer Science*, pages 287–306. Springer, 2010.

[22] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14, 2012.

[23] J. Hoffmann, K. Aehlig, and M. Hofmann. Resource aware ML. *CAV 2012*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer, 2012.

[24] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. *POPL 2003*, pages 185–197. ACM, 2003.

[25] M. Hofmann and S. Jost. Type-based amortised heap-space analysis. *ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 22–37. Springer, 2006.

[26] M. Hofmann and D. Rodriguez. Efficient type-checking for amortised heap-space analysis. *CSL 2009*, volume 5771 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2009.

[27] C. Jones. *Probabilistic Non-Determinism*. PhD thesis, The University of Edinburgh, 1989.

[28] S. Jost, H. Loidl, K. Hammond, N. Scaife, and M. Hofmann. "carbon credits" for resource-bounded computations using amortised analysis. *FM 2009*, volume 5850 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2009.

[29] S. Jost, K. Hammond, H. Loidl, and M. Hofmann. Static determination of quantitative resource usage for higher-order programs. *POPL 2010*, pages 223–236. ACM, 2010.

[30] J. Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2004. ISBN 0-321-29535-8.

[31] D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973. ISBN 0-201-03803-X.

[32] D. E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms, 2nd Edition*. Addison-Wesley, 1973.

[33] J. McConnell. *Analysis of Algorithms C An Active Learning Approach*. Jones and Bartlett Publishers, Inc., 2008.

[34] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. ISBN 0-521-47465-5.

[35] F. Olmedo, B. L. Kaminski, J.-P. Katoen, and C. Matheja. Reasoning about recursive probabilistic programs. In *LICS 2016*, 2016, to appear.

[36] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. *VMCAI 2004*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251. Springer, 2004.

[37] L. Shen, M. Wu, Z. Yang, and Z. Zeng. Generating exact nonlinear ranking functions by symbolic-numeric hybrid method. *J. Systems Science & Complexity*, 26(2):291–301, 2013. doi: 10.1007/s11424-013-1004-1.

[38] K. Sohn and A. V. Gelder. Termination detection in logic programs using argument sizes. *PODS 1991*, pages 216–226. ACM Press, 1991.

[39] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. B. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.

[40] L. Yang, C. Zhou, N. Zhan, and B. Xia. Recent advances in program verification through computer algebra. *Frontiers of Computer Science in China*, 4(1):1–16, 2010.

A. Proofs for Sect. 4.2

To prove lemmas in Sect. 4.2 we need the following well-known theorem.

Theorem 4 (Taylor's Theorem (with Lagrange's Remainder) [4, Chapter 6])**.** *For any function $f : [a, b] \to \mathbb{R}$ ($a, b \in \mathbb{R}$ and $a < b$), if f is $(k+1)$-order differentiable, then for all $x \in [a, b]$, there exists a $\xi \in (a, x)$ such that*

$$f(x) = \left(\sum_{j=0}^{k} \frac{f^{(j)}(a)}{j!} \cdot (x-a)^j \right) + \frac{f^{(k+1)}(\xi)}{(k+1)!} \cdot (x-a)^{k+1} \ .$$

We also need to recall that

$$\sum_{j=1}^{\infty} \frac{1}{j^2} = \frac{\pi^2}{6} \text{ and } \sum_{j=1}^{\infty} \frac{1}{j^3} = \alpha$$

where α is the Apéry's constant which lies in $[1.2020, 1.2021]$.

Proposition 1. For any natural number $n \geq 2$, we have

$$(1)\ \ln n - \ln 2 - \frac{1}{n-1} \leq \ln \left\lfloor \frac{n}{2} \right\rfloor \leq \ln n - \ln 2 \ ;$$

$$(2)\ \ln n - \ln 2 \leq \ln \left\lceil \frac{n}{2} \right\rceil \leq \ln n - \ln 2 + \frac{1}{n} \ .$$

Proof. Let $n \geq 2$ be a natural number. The first argument comes from the facts that

$$\ln \left\lfloor \frac{n}{2} \right\rfloor \leq \ln \frac{n}{2} = \ln n - \ln 2$$

and

$$\begin{aligned} \ln \left\lfloor \frac{n}{2} \right\rfloor &\geq \ln \frac{n-1}{2} \\ &= \ln \frac{n}{2} - \left(\ln \frac{n}{2} - \ln \frac{n-1}{2} \right) \\ &= \ln n - \ln 2 - \frac{1}{2} \cdot \frac{1}{\xi_n} \quad \left(\xi_n \in \left(\frac{n-1}{2}, \frac{n}{2} \right) \right) \\ &\geq \ln n - \ln 2 - \frac{1}{n-1} \end{aligned}$$

where we use the fact that

$$\left\lfloor \frac{n}{2} \right\rfloor \geq \frac{n-1}{2}$$

and ξ_n is obtained from Taylor's Theorem. The second argument comes from the facts that

$$\ln \left\lceil \frac{n}{2} \right\rceil \geq \ln \frac{n}{2} = \ln n - \ln 2$$

and

$$\begin{aligned} \ln \left\lceil \frac{n}{2} \right\rceil &\leq \ln \frac{n+1}{2} \\ &= \ln \frac{n}{2} + \left(\ln \frac{n+1}{2} - \ln \frac{n}{2} \right) \\ &= \ln n - \ln 2 + \left(\ln \frac{n+1}{2} - \ln \frac{n}{2} \right) \\ &= \ln n - \ln 2 + \frac{1}{2} \cdot \frac{1}{\xi'_n} \quad \left(\xi'_n \in \left(\frac{n}{2}, \frac{n+1}{2} \right) \right) \\ &\leq \ln n - \ln 2 + \frac{1}{n} \end{aligned}$$

where the first inequality is due to the fact that

$$\left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2}$$

and ξ'_n is obtained from Taylor's Theorem. □

Proposition 2. For any natural number $n \geq 2$, we have

$$\ln n - \frac{1}{n-1} \leq \ln (n-1) \leq \ln n - \frac{1}{n}.$$

Proof. The lemma follows directly from the fact that

$$\ln n - \ln (n-1) = \frac{1}{\xi}$$

for some $\xi \in (n-1, n)$, which can be obtained through Taylor's Theorem. □

Proposition 3. For any natural number $n \geq 2$, we have

$$(1)\ \int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1} \frac{1}{j} \in \left[-0.7552, -\frac{1}{6} \right] \quad (14)$$

$$(2)\ \int_1^n \ln x\,\mathrm{d}x - \left(\sum_{j=1}^{n-1} \ln j \right) - \frac{1}{2} \cdot \int_1^n \frac{1}{x}\,\mathrm{d}x \in \left[-\frac{1}{12}, 0.2701 \right] \quad (15)$$

$$(3) \int_1^n x \cdot \ln x\,\mathrm{d}x - \left(\sum_{j=1}^{n-1} j \cdot \ln j \right) - \frac{1}{2} \cdot \int_1^n \ln x\,\mathrm{d}x + \frac{1}{12} \cdot \int_1^n \frac{1}{x}\,\mathrm{d}x - \frac{n-1}{2} \in \left[-\frac{19}{72}, 0.1575 \right]. \quad (16)$$

Proof. Let n be a natural number such that $n \geq 2$. We first estimate the difference

$$\int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1} \frac{1}{j} \ .$$

To this end, we deduce the following equalities:

$$\begin{aligned} &\int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1} \frac{1}{j} \\ =& \sum_{j=1}^{n-1} \int_j^{j+1} \left[\frac{1}{x} - \frac{1}{j} \right] \mathrm{d}x \\ =& \sum_{j=1}^{n-1} \int_0^1 \left[\frac{1}{j+x} - \frac{1}{j} \right] \mathrm{d}x \\ =& \sum_{j=1}^{n-1} \int_0^1 \left[-\frac{1}{j^2} \cdot x + \frac{1}{\xi_{j,x}^3} \cdot x^2 \right] \mathrm{d}x \\ =& -\frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \frac{1}{j^2} \right) + \sum_{j=1}^{n-1} \int_0^1 \frac{1}{\xi_{j,x}^3} \cdot x^2\,\mathrm{d}x \ , \end{aligned}$$

where $\xi_{j,x}$ is a real number in $(j, j+x)$ obtained from Taylor's Theorem with Lagrange's Remainder. The first and fourth equalities come from the linear property of Riemann Integral; the second one follows from the variable substitution $x' = x - j$; the third one follows from Taylor's Theorem. Using the fact that $\xi_{j,x} \in (j, j+1)$, one obtains that

$$\int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1} \frac{1}{j} \leq -\frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \frac{1}{j^2} \right) + \frac{1}{3} \cdot \left(\sum_{j=1}^{n-1} \frac{1}{j^3} \right) \quad (20)$$

and

$$\int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1}\frac{1}{j} \geq -\frac{1}{2}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j^2}\right) + \frac{1}{3}\cdot\left(\sum_{j=2}^{n}\frac{1}{j^3}\right). \quad (21)$$

Then (14) follows from the facts that

$$\begin{aligned}\int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1}\frac{1}{j} &\leq \sum_{j=1}^{n-1}\left(-\frac{1}{2\cdot j^2}+\frac{1}{3\cdot j^3}\right)\\ &\leq -\frac{1}{2\cdot 1^2}+\frac{1}{3\cdot 1^3}\\ &= -\frac{1}{6}\end{aligned}$$

and

$$\begin{aligned}\int_1^n \frac{1}{x}\,\mathrm{d}x - \sum_{j=1}^{n-1}\frac{1}{j} &\geq \sum_{j=1}^{n-1}\left(-\frac{1}{2\cdot j^2}+\frac{1}{3\cdot j^3}\right) - \frac{1}{3} + \frac{1}{3\cdot n^3}\\ &\geq -\frac{\pi^2}{12}+\frac{\alpha}{3}-\frac{1}{3}\\ &\geq -0.7552\end{aligned}$$

where in both situations we use the fact that $2\cdot j^2 \leq 3\cdot j^3$ for all $j\in\mathbb{N}$.

Then we consider the difference

$$\int_1^n \ln x\,\mathrm{d}x - \sum_{j=1}^{n-1}\ln j\,.$$

First, we derive that

$$\begin{aligned}&\int_1^n \ln x\,\mathrm{d}x - \sum_{j=1}^{n-1}\ln j\\ =&\sum_{j=1}^{n-1}\int_j^{j+1}\left[\ln x - \ln j\right]\,\mathrm{d}x\\ =&\sum_{j=1}^{n-1}\int_0^1\left[\ln(j+x)-\ln j\right]\,\mathrm{d}x\\ =&\sum_{j=1}^{n-1}\int_0^1\left[\frac{1}{j}\cdot x - \frac{1}{2\cdot\xi_{j,x}^2}\cdot x^2\right]\,\mathrm{d}x\\ =&\frac{1}{2}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j}\right) - \sum_{j=1}^{n-1}\int_0^1\frac{1}{2\cdot\xi_{j,x}^2}\cdot x^2\,\mathrm{d}x\end{aligned}$$

where $\xi_{j,x}$ is a real number in $(j, j+1)$ obtained from Taylor's Theorem. Using the fact that $\xi_{j,x}\in(j,j+1)$, one can obtain that

$$\begin{aligned}&\int_1^n \ln x\,\mathrm{d}x - \sum_{j=1}^{n-1}\ln j \quad (22)\\ \leq&\frac{1}{2}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j}\right) - \frac{1}{6}\cdot\sum_{j=2}^{n}\frac{1}{j^2}\\ \leq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \frac{1}{4}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j^2}\right) - \frac{1}{6}\cdot\left(\sum_{j=2}^{n}\frac{1}{j^3}\right)\\ &\quad - \frac{1}{6}\cdot\sum_{j=2}^{n}\frac{1}{j^2}\\ =&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \sum_{j=1}^{n-1}\left(\frac{1}{12\cdot j^2}-\frac{1}{6\cdot j^3}\right)\\ &\quad + \frac{1}{3} - \frac{1}{6\cdot n^3} - \frac{1}{6\cdot n^2}\end{aligned}$$

where the second inequality follows from Inequality (21), and

$$\begin{aligned}&\int_1^n \ln x\,\mathrm{d}x - \sum_{j=1}^{n-1}\ln j \quad (23)\\ \geq&\frac{1}{2}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j}\right) - \frac{1}{6}\cdot\sum_{j=1}^{n-1}\frac{1}{j^2}\\ \geq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \frac{1}{12}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j^2}\right) - \frac{1}{6}\cdot\left(\sum_{j=1}^{n-1}\frac{1}{j^3}\right)\\ =&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \sum_{j=1}^{n-1}\left(\frac{1}{12\cdot j^2}-\frac{1}{6\cdot j^3}\right)\end{aligned}$$

where the second inequality follows from Inequality (20). Then from Inequality (22) and Inequality (23), one has that

$$\begin{aligned}&\int_1^n \ln x\,\mathrm{d}x - \sum_{j=1}^{n-1}\ln j\\ \leq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \sum_{j=1}^{\infty}\left(\frac{1}{12\cdot j^2}-\frac{1}{6\cdot j^3}\right) + \frac{1}{3}\\ \leq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \frac{\pi^2}{72} - \frac{\alpha}{6} + \frac{1}{3}\\ \leq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + 0.2701\end{aligned}$$

and

$$\begin{aligned}&\int_1^n \ln x\,\mathrm{d}x - \sum_{j=1}^{n-1}\ln j\\ \geq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x + \left(\frac{1}{12}-\frac{1}{6}\right)\\ \geq&\frac{1}{2}\cdot\int_1^n\frac{1}{x}\,\mathrm{d}x - \frac{1}{12}\end{aligned}$$

where in both situations we use the fact that $12\cdot j^2 \leq 6\cdot j^3$ for all $j\geq 2$. The inequalities above directly imply the inequalities in

(15). Finally, we consider the difference

$$\int_1^n x \cdot \ln x \,\mathrm{d}x - \sum_{j=m}^{n-1} j \cdot \ln j \ .$$

Following similar approaches, we derive that for all natural numbers $n \geq 2$,

$$\begin{aligned}
&\int_1^n x \cdot \ln x \,\mathrm{d}x - \sum_{j=1}^{n-1} j \cdot \ln j \\
=&\sum_{j=1}^{n-1} \int_j^{j+1} \left[x \cdot \ln x - j \cdot \ln j\right] \,\mathrm{d}x \\
=&\sum_{j=1}^{n-1} \int_0^1 \left[(j+x) \cdot \ln (j+x) - j \cdot \ln j\right] \,\mathrm{d}x \\
=&\sum_{j=1}^{n-1} \int_0^1 \left[(\ln j + 1) \cdot x + \frac{1}{2 \cdot \xi_{j,x}} \cdot x^2\right] \,\mathrm{d}x \\
=&\frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \ln j\right) + \frac{n-1}{2} + \sum_{j=1}^{n-1} \int_0^1 \frac{1}{2 \cdot \xi_{j,x}} \cdot x^2 \,\mathrm{d}x
\end{aligned}$$

where $\xi_{j,x} \in (j, j+1)$. Thus, one obtains that

$$\begin{aligned}
&\int_1^n x \cdot \ln x \,\mathrm{d}x - \sum_{j=1}^{n-1} j \cdot \ln j \leq \\
&\quad \frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \ln j\right) + \frac{n-1}{2} + \frac{1}{6} \cdot \sum_{j=1}^{n-1} \frac{1}{j}
\end{aligned} \tag{24}$$

and

$$\begin{aligned}
&\int_1^n x \cdot \ln x \,\mathrm{d}x - \sum_{j=1}^{n-1} j \cdot \ln j \geq \\
&\quad \frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \ln j\right) + \frac{n-1}{2} + \frac{1}{6} \cdot \sum_{j=1}^{n-1} \frac{1}{j} - \frac{1}{6} + \frac{1}{6 \cdot n}.
\end{aligned} \tag{25}$$

By plugging Inequalities in (21) and (23) into Inequality (24), one obtains that

$$\begin{aligned}
&\int_1^n x \cdot \ln x \,\mathrm{d}x - \sum_{j=1}^{n-1} j \cdot \ln j \\
\leq&\frac{1}{2} \cdot \left[\int_1^n \ln x \,\mathrm{d}x - \frac{1}{2} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x - \sum_{j=1}^{n-1} \left(\frac{1}{12 \cdot j^2} - \frac{1}{6 \cdot j^3}\right)\right] \\
&+ \frac{n-1}{2} \\
&+ \frac{1}{6} \cdot \left[\int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \frac{1}{j^2}\right) - \frac{1}{3} \cdot \left(\sum_{j=2}^{n} \frac{1}{j^3}\right)\right] \\
\leq&\frac{1}{2} \cdot \int_1^n \ln x \,\mathrm{d}x - \frac{1}{12} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{n-1}{2} \\
&+ \sum_{j=1}^{n-1} \left(\frac{1}{24 \cdot j^2} + \frac{1}{36 \cdot j^3}\right) + \frac{1}{18} - \frac{1}{18 \cdot n^3} \\
\leq&\frac{1}{2} \cdot \int_1^n \ln x \,\mathrm{d}x - \frac{1}{12} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{n-1}{2} + \frac{\pi^2}{144} + \frac{\alpha}{36} + \frac{1}{18} \\
\leq&\frac{1}{2} \cdot \int_1^n \ln x \,\mathrm{d}x - \frac{1}{12} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{n-1}{2} + 0.1575
\end{aligned}$$

for all natural numbers $n \geq 2$. Similarly, by plugging Inequalities in (20) and (22) into Inequality (25), one obtains

$$\begin{aligned}
&\int_1^n x \cdot \ln x \,\mathrm{d}x - \sum_{j=1}^{n-1} j \cdot \ln j \\
\geq&\frac{1}{2} \cdot \left[\int_1^n \ln x \,\mathrm{d}x - \frac{1}{2} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x - \sum_{j=1}^{n-1} \left(\frac{1}{12 \cdot j^2} - \frac{1}{6 \cdot j^3}\right) - \frac{1}{3}\right] \\
&+ \frac{n-1}{2} \\
&+ \frac{1}{6} \cdot \left[\int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{1}{2} \cdot \left(\sum_{j=1}^{n-1} \frac{1}{j^2}\right) - \frac{1}{3} \cdot \left(\sum_{j=1}^{n-1} \frac{1}{j^3}\right)\right] - \frac{1}{6} \\
\geq&\frac{1}{2} \cdot \int_1^n \ln x \,\mathrm{d}x - \frac{1}{12} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{n-1}{2} \\
&+ \sum_{j=1}^{n-1} \left(\frac{1}{24 \cdot j^2} + \frac{1}{36 \cdot j^3}\right) - \frac{1}{3} \\
\geq&\frac{1}{2} \cdot \int_1^n \ln x \,\mathrm{d}x - \frac{1}{12} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{n-1}{2} + \frac{1}{24} + \frac{1}{36} - \frac{1}{3} \\
=&\frac{1}{2} \cdot \int_1^n \ln x \,\mathrm{d}x - \frac{1}{12} \cdot \int_1^n \frac{1}{x} \,\mathrm{d}x + \frac{n-1}{2} - \frac{19}{72}
\end{aligned}$$

Then the inequalities in (16) are clarified. □

B. Proofs for Sect. 4.3

Lemma 4. Let $\mathfrak{f} \in \{\ln \mathfrak{n}, \mathfrak{n}, \mathfrak{n} \cdot \ln \mathfrak{n}\}$ and c be a constant. For all univariate recurrence expressions $\mathfrak{g}$, there exists pseudo-polynomials p and q such that coefficients (i.e., a_i, b_i's in (17)) of q are all non-negative, $C_q > 0$ and the following assertion holds: for all $d > 0$ and for all $n \geq 2$, with $h = d \cdot \mathsf{Sub}(\mathfrak{f}) + c$, the inequality

$$\mathsf{OvAp}(\mathfrak{g}, h)(n) \leq h(n) \tag{18}$$

is equivalent to

$$d \cdot p(n) \geq q(n) \quad (19).$$

Proof. From Definition 2, $n \mapsto n \cdot (n-1) \cdot \mathsf{OvAp}(\mathfrak{g}, h)(n)$ is a pseudo-polynomial. Simple rearrangement of terms of inequality (18) gives the desired pseudo-polynomials. Moreover, the fact that all coefficients in $\mathfrak{g}$ (from (1)) are positive, is used to derive that all coefficients of q are non-negative and $C_q > 0$. □

Proposition 4. Let p, q be pseudo-polynomials such that $C_q > 0$ and all coefficients of q are non-negative. Then there exists a real number $d > 0$ such that $d \cdot p(n) \geq q(n)$ for sufficiently large n iff $\deg(p) \geq \deg(q)$ and $C_p > 0$.

Proof. We present the two directions of the proof.

("*If*":) Suppose that $\deg(p) \geq \deg(q)$ and $C_p > 0$. Then the result follows directly from the facts that (i) $\frac{q(n)}{p(n)} > 0$ for sufficiently large n and (ii) $\lim_{n\to\infty} \frac{q(n)}{p(n)}$ exists and is non-negative.

("*Only-if*":) Let d be a positive real number such that $d \cdot p(n) \geq q(n)$ for sufficiently large n. Then $C_p > 0$, or otherwise $d \cdot p(n)$ is either constantly zero or negative for sufficiently large n. Moreover, $\deg(p) \geq \deg(q)$, since otherwise $\lim_{n\to\infty} \frac{q(n)}{p(n)} = \infty$. □

Proposition 5. Consider two univariate pseudo-polynomials p, q such that $\deg(p) \geq \deg(q)$, all coefficients of q are non-negative and $C_p, C_q > 0$. Then given any $\epsilon \in (0, 1)$,

$$\frac{q(n)}{p(n)} \leq \frac{\mathbf{1}_{\deg(p)=\deg(q)} \cdot \frac{C_q}{C_p} + \epsilon}{1-\epsilon}$$

for all $n \geq N_{\epsilon,p,q}$ (for $N_{\epsilon,p,q}$ of Definition 4).

Proof. Let p, q be given in Definition 4. Fix an arbitrary $\epsilon \in (0, 1)$ and let $N_{\epsilon,p,q}$ be given in Definition 4. Then for all $n \geq N_{\epsilon,p,q}$, (i) both $p(n), q(n)$ are positive and (ii)

$$\begin{aligned}\frac{q(n)}{\overline{p}(n)} &\leq \sum_{i=1}^{k'} a'_i \cdot \frac{N^i \cdot \ln N}{\overline{p}(N)} + \sum_{i=1}^{\ell'} b'_i \cdot \frac{N^i}{\overline{p}(N)} \\ &\leq \mathbf{1}_{\deg(p)=\deg(q)} \cdot \frac{C_q}{C_p} + \epsilon\end{aligned}$$

and

$$\begin{aligned}\frac{p(n)}{\overline{p}(n)} &\geq 1 - \left[-1 + \sum_{i=1}^{k} |a_i| \cdot \frac{N^i \cdot \ln N}{\overline{p}(N)} + \sum_{i=1}^{\ell} |b_i| \cdot \frac{N^i}{\overline{p}(N)}\right] \\ &\geq 1 - \epsilon.\end{aligned}$$

It follows that for all $n \geq N_{\epsilon,p,q}$,

$$\frac{q(n)}{p(n)} \leq \frac{\mathbf{1}_{\deg(p)=\deg(q)} \cdot \frac{C_q}{C_p} + \epsilon}{1-\epsilon} \ .$$

The desired result follows. □

Theorem 2.[Soundness for *UniDec*] If *UniDec* outputs "*yes*", then there exists a univariate guess-and-check function in form (13) for the inputs G and $\mathfrak{f}$. The algorithm is a linear-time algorithm.

Proof. From Definition 1 and the special form (13) for univariate guess-and-check functions, a function in form (13) which satisfies the inductive argument of Definition 1 can be modified to satisfy also the base condition of Definition 1 by simply raising d to a sufficiently large amount. Then the result follows from Theorem 1 and the sufficiency of Proposition 4. □

Theorem 3.[Soundness for *UniSynth*] If the algorithm *UniSynth* outputs a real number d, then $d \cdot \mathsf{Sub}(\mathfrak{f}) + c$ is a univariate guess-and-check function for G.

Proof. Directly from the construction of the algorithm, Theorem 1, Proposition 4 and Proposition 5. □

C. Detailed Experimental Results

The detailed experimental results are given in Table 3. We use ✓ to represent *yes* and × for *fail*. In addition to Table 2, we include values for $N_{\epsilon,p,q}$ in Definition 4. For the separable bivariate examples, recall that n does not change, and in these examples, the reduction to the univariate case is the function of m.

PROGRAM	f	UNIDEC	UNISYNTH(✓)			
			ϵ	$N_{\epsilon,p,q}$	d	d_{100}
R.-SEAR.	$\ln n$	✓	0.5	13	40.107	15.129
			0.3	25	28.363	
			0.1	97	21.838	
			0.01	1398	19.762	
Q.-SORT	$\ln n$	×	-	-	-	-
	n	×				
	$n \ln n$	✓	0.5	10	9.001	3.172
			0.3	21	6.143	
			0.1	91	4.556	
			0.01	1458	4.051	
Q.-SELECT	$\ln n$	×	-	-	-	-
	n	✓	0.5	33	17.001	7.909
			0.3	54	11.851	
			0.1	160	9.001	
			0.01	1600	8.091	
DIAM. A	$\ln n$	×	-	-	-	-
	n	×				
	$n \ln n$	✓	0.5	3	9.001	4.525
			0.3	3	6.143	
			0.1	4	4.556	
			0.01	4	4.525	
DIAM. B	$\ln n$	×	-	-	-	-
	n	✓	0.5	9	13.001	5.918
			0.3	14	9.001	
			0.1	40	6.778	
			0.01	400	6.071	
SORT-SEL.	$\ln n$	×	-	-	-	-
	n	×				
	$n \ln n$	✓	0.5	18	50.052	16.000
			0.3	29	24.852	
			0.1	87	17.313	
			0.01	866	16.000	
COUPON	$n \cdot \ln m$	✓	0.5	2	3.001	0.910
			0.3	2	1.858	
			0.1	2	1.223	
			0.01	2	1.021	
RES. A	$n \cdot \ln m$	✓	0.5	2	6.437	2.472
			0.3	2	4.312	
			0.1	2	3.132	
			0.01	2	2.756	
RES. B	$\ln m$	×	-	-	-	-
	m	✓	0.5	2	6.437	2.691
			0.3	2	4.312	
			0.1	2	3.132	
			0.01	2	2.756	

Table 3. Detailed experimental results where all running times (averaged over 5 runs) are less than 0.02 seconds (between 0.01 and 0.02 seconds).