Tail Bound Analysis of Probabilistic Recurrence Relations via Markov's Inequality

ANONYMOUS AUTHOR(S)

Probabilistic recurrence relations (PRRs) are a standard formalism to analyze the runtime of randomized algorithms. We consider the classical problem of deriving tail bounds on the runtime of randomized algorithms modeled as systems of probabilistic recurrence relations (SPRRs). Specifically, given a time limit κ that depends on the input size n, we study the tail probability $\Pr[T \geq \kappa]$ that the runtime T exceeds the time limit. The key research question in tail bound analysis is to find an upper-bound function $\mathit{upper}(\kappa) \geq \Pr[T \geq \kappa]$ such that $\mathit{upper}(\kappa)$ converges rapidly to zero as κ tends to infinity. The classical and most well-known approach to this problem is the "cookbook" method by Karp (JACM 1994). Since then, there has been no general approach that could improve the tail bounds derived from this classical method, except for ad-hoc manual analysis for specific PRRs such as that of QuickSort.

In this work, we propose a novel method to derive tail bounds for probabilistic recurrence relations via Markov's inequality. First, we focus on creating a theoretical foundation for the tail-bound analysis and present approaches for deriving exponentially and polynomially-decreasing tail bounds, which are the two most common types of bounds in the runtime analysis of randomized algorithms. We show that our theoretical approaches can handle instances which are beyond the reach of Karp's. Specifically, we can tackle systems of recurrence relations, instead of a single relation, and can also handle PRRs with randomized preprocessing time. However, there are also instances which can be handled by Karp's method but not ours. Hence, the two approaches are incomparable in theory. Additionally, we show that our theoretical approach can prove a new and asymptotically-improved tail bound for the classical randomized algorithm QuickSelect, which was not previously known in the literature.

We then turn our attention to automating our theoretical approaches and provide efficient algorithms based on Taylor's expansion and approximations of higher moments of logarithmic terms. The trade-off for the attained automation and efficiency is that our algorithmic approaches make extra assumptions and are hence not as general as our theoretical results. Nevertheless, they can handle a wide variety of real-world SPRRs as demonstrated in our experimental results. Compared with Karp's method, our algorithmic approaches find significantly tighter tail bounds for classical randomized algorithms such as QuickSort, QuickSelect and DiameterComputation. Notably, they also find tail bounds for various randomized algorithms in computational geometry, such as the LP algorithm by Seidel, to which no previous methods were applicable.

1 INTRODUCTION

Recurrence analysis in programming languages. A fundamental problem that has received significant attention in programming language research is the automated complexity analysis of programs (see e.g. the COSTA project [Albert et al. 2009], resource-aware OCAML [Hoffmann et al. 2017], complexity analysis of (probabilistic) programs [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2017a, 2018; Kura et al. 2019; Wang et al. 2021a,b], etc.). In this line of research, recurrence analysis is an important subject. Recurrence analysis is the study of automatically analyzing recurrence relations that describe the behavior of their corresponding programs. In program analysis, an important method is to extract a recurrence relation as an abstraction from the underlying program and perform an analysis on the extracted recurrence relation. Recurrence analysis has been studied in both classical (non-probabilistic) setting [Albert et al. 2009, 2008; Bagnara et al. 2005; Breck et al. 2020; Chatterjee et al. 2017a; Farzan and Kincaid 2015; Grobauer 2001; Kincaid et al. 2017, 2018] and probabilistic setting [Chatterjee et al. 2017b; Karp 1994; Tassarotti

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/

2017; Tassarotti and Harper 2018]. In this work, we consider the fundamental problem of tail bound analysis of recurrence relations in the probabilistic setting (i.e., probabilistic recurrence relations).

Probabilistic recurrence relations. Probabilistic recurrence relations (PRRs) are the probabilistic extension of classical recurrence relations and are widely used in the runtime analysis of recursive randomized algorithms. The random variables in PRRs model samplings and sizes of the instances passed to recursive procedures in a randomized algorithm. The runtime behavior of many randomized algorithms, such as QuickSort [Hoare 1961a], QuickSelect [Hoare 1961b], and DiameterComputation [Motwani and Raghavan 1995, Chapter 9] are captured by PRRs. Thus, the runtime analysis of a randomized algorithm is often reduced to that of its corresponding PRR. Hence, formal analysis of PRRs is a central and classical problem in complexity analysis of randomized algorithms [Bazzi and Mitter 2003; Chatterjee et al. 2017b; Chaudhuri and Dubhashi 1997; Dubhashi and Panconesi 2009; Karp 1994; McDiarmid and Hayward 1996; Motwani and Raghavan 1995; Tassarotti 2017; Tassarotti and Harper 2018]. Two fundamental problems in the formal analysis of PRRs are *expected runtime* and *tail bound* analysis.

Expected runtime analysis. Expected runtime analysis aims at deriving tight upper bounds on the expected runtime of a given PRR. Expected runtime is a key performance criterion that examines for how long the randomized algorithm will execute in expectation. This is important since many randomized algorithms perform poorly in the worst case but are efficient on average. For example, the well-known QuickSort algorithm [Hoare 1961a] takes $\Omega(n^2)$ time in the worst case but has an expected runtime of $O(n \log n)$. Compared with the classical runtime analysis of non-probabilistic recurrences that can typically be solved using the Master theorem [Cormen et al. 2009, Chapter 4][Akra and Bazzi 1998], the expected runtime analysis of PRRs is more intricate, and there is no straightforward solution similar to the Master theorem, except for certain special cases [Bazzi and Mitter 2003]. Instead, the expected runtime of PRRs is obtained either through the traditional methods of Harmonic series and generating functions via computer algebra [Flajolet et al. 1991; Zimmermann and Zimmermann 1989] or through automated guess-and-check algorithms [Chatterjee et al. 2017b].

Tail bound analysis. In this work, we focus on tail bound analysis of PRRs. The tail bound analysis [Dubhashi and Panconesi 2009] considers upper bounds on the probability that a given PRR does not terminate within a prescribed time limit. The goal is to show that this probability converges rapidly to zero as the time limit tends to infinity. The most classical approach to this problem is that of [Karp 1994]. Since then, finding approaches to obtain tighter bounds than Karp's method and automating such approaches has been a long-standing unresolved problem. Instead of a general approach, ad-hoc manual analysis has been performed to improve bounds for specific recurrences, such as QuickSort [McDiarmid and Hayward 1996].

Comparison of the two analyses. Compared with the expected runtime analysis, the tail bound analysis is a more refined problem. It examines the probability that the runtime of a given PRR deviates significantly from its expected value. Unsurprisingly, methods used for tail bound analysis are typically much more involved than expected runtime analysis. For instance, expected runtime analysis allows a template-based approach through guess-and-check functions [Chatterjee et al. 2017b], while there is no such template-based approach for the tail bound analysis in the literature. Furthermore, in some cases, such as QuickSort, it is possible to manually derive the exact formula of expected runtime of probabilistic recurrences by applying the generating-function method, but no such approach is known when it comes to tail bounds.

Current state-of-the-art. The classical method to obtain tail bounds of PRRs was proposed in [Karp 1994] where a "cookbook" formula similar to Master theorem is presented. This method requires the solution to the expected runtime of a deterministic version of the input PRR, and then

outputs a tail bound for the original PRR by a fixed mathematical formula. The two main merits of Karp's method are that (i) it is based on a very simple formula, and (ii) it often provides a good approximation of the general asymptotic behavior of the PRR. Nevertheless, it is well-known that the tradeoff of this simplicity is that the method usually leads to loose tail bounds. Based on Karp's method, the work [Chaudhuri and Dubhashi 1997] proposed an automated approach that weakens several conditions required in [Karp 1994] and obtains bounds that are slightly worse than [Karp 1994]. For the QuickSort PRR, the work [McDiarmid and Hayward 1996] performed an involved manual analysis to derive the asymptotically optimal tail bound. Karp's method is also extended to parallel randomized algorithms [Karpinski and Zimmermann 1991; Tassarotti 2017]. Recently, the work [Tassarotti and Harper 2018] mechanized the proof for the tail bound of QuickSort in the theorem prover Coq [Bertot and Castéran 2004].

Challenges and gaps. Although the tail bound analysis for PRRs is a classical problem that has been studied for around thirty years, there are still several key challenges. First, while the bounds derived from [Karp 1994] were known to be coarse, as exemplified by [McDiarmid and Hayward 1996] and the results of this work, in the decades since there have been no approaches that (a) can be automated, and (b) can improve these classical results. All previous improvements lacked automation and were obtained on a fixed PRR, i.e. they considered only a specific recurrence, such as QuickSort. Second, in several application domains, such as computational geometry, the preprocessing time in each recurrence equation is randomized rather than deterministic, and the analysis involves a system of PRRs rather than a single recurrence. Unfortunately, existing approaches all assumed a single PRR and that the preprocessing time is deterministic, and are therefore inapplicable. Third, all existing results consider tail bounds that are *exponentially* decreasing and other forms of tail bounds (e.g. polynomially-decreasing) have not been investigated.

Our contribution. In this work, we address the gaps and challenges above and provide the following contributions:

- Based on Markov's inequality, we propose theoretical results to derive exponentially and polynomially-decreasing tail bounds. In comparison to Karp's method, we establish the following: (i) our results can derive polynomially-decreasing tail bounds, while Karp's method can only derive exponentially-decreasing tail bounds; (ii) our results can handle randomized preprocessing time and systems of PRRs while Karp's method cannot; (iii) our results can derive asymptotically tighter tail bounds than Karp's method for the classical QuickSelect algorithm; (iv) we provide an example that can be handled by Karp's method but not by ours. As such, the two methods are technically incomparable. Another notable result is that our theoretical approach can obtain a novel asymptotically-improved tail bound for QuickSelect. Despite QuickSelect being a classical and widely-studied randomized algorithm, this bound was not previously known in the literature.
- Based on our theoretical results, we develop prototype template-based algorithms to automatically synthesize exponentially and polynomially-decreasing tail bounds for specific families of SPRRs, with pseudo-polynomial templates, that appear commonly in the runtime analysis of recursive randomized algorithms. Our main tools in the synthesis process are Taylor's expansion and tight approximations of expectation of moments with logarithmic terms.
- We provide experimental results over various classical randomized algorithms. Compared with Karp's cookbook method, our prototype algorithmic approaches can already derive much tighter tail bounds over classical randomized algorithms such as QuickSort, QuickSelect, RandomizedSearch and DiameterComputation. For example, in the case of QuickSelect, we find a bound for $\Pr[T \geq 20 \cdot n]$ that is 10^6 times tighter. Moreover, for QuickSort and

RandomizedSearch, our bounds become arbitrarily tighter than Karp's as n tends to infinity. See Section 5 for details.

- Our experiments illustrate that our implementation can also handle randomized algorithms that are beyond the scope of [Karp 1994] and other existing methods such as [Chaudhuri and Dubhashi 1997; Tassarotti 2017]. First, we can handle PRRs with a randomized preprocessing phase, while other approaches require the preprocessing phase to be deterministic. Randomized preprocessing is required in several algorithms from computational geometry [Motwani and Raghavan 1995, Chapter 9]. Second, our algorithms can handle *systems* of PRRs such as those arising from the SeidelLP [Seidel 1991] and the SmallestEnclosingDisk problems [Welzl 1991]. To our best knowledge, no existing method can handle tail bounds for systems of PRRs.
- Finally, our experiments demonstrate that our approach is extremely efficient in practice and able to find much tighter tail bounds for each benchmark in less than 4 seconds.

In summary, our contributions include significant advances towards all the mentioned key challenges for tail bound analysis of PRRs.

Key technical aspects. The key technical aspects of our results are two-fold: theoretical and algorithmic. First, we establish novel theoretical results for characterizing exponentially and polynomially-decreasing tail bounds of PRRs through Markov's inequality. Then, we provide synthesis algorithms, based on pseudo-polynomial templates, Taylor's expansion and over-approximation of expectation of moments with logarithmic terms, that automatically derive such tail bounds. A key novelty is that while these techniques are quite simple, they surprisingly lead to first significant improvements in classical decades-old bounds for fundamental and well-studied randomized algorithms. Furthermore, our approach can even handle classical randomized algorithms that are beyond the reach of all previous methods.

Limitations. While our theoretical results are applicable to deriving exponentially and polynomially-decreasing tail bounds for general SPRRs, our automated synthesis algorithms have to make extra assumptions and are therefore more limited than the theory. This is a price we pay as a tradeoff for automation and efficiency. We now briefly discuss the limitations of our algorithms. For details, see Section 4.1.

Templates. Our algorithms are based on certain pseudo-polynomial templates and thus cannot handle PRRs with non-pseudo-polynomial runtime behavior. Note that most classical randomized algorithms have pseudo-polynomial runtime. Thus, our algorithms are applicable to a wide variety of real-world instances.

Distributions. We assume that the probability distributions in a PRR are either discrete distributions with a bounded number of probabilistic choices or a (piece-wise) uniform distribution over a contiguous range of sizes passed to sub-procedures. In classical randomized divide-and-conquer algorithms, the random sizes passed to sub-procedures always come from such distributions. We can extend our algorithms to support other distributions, assuming that their higher moments with logarithms are computable/approximable, but this will cost us in terms of efficiency and has no real-world motivation.

Recursive calls. We assume that each recurrence equation in a PRR involves at most two sub-procedure calls to itself, i.e. each procedure in the randomized algorithm calls itself recursively for at most two times. Moreover, when there are exactly two sub-procedure calls, we assume the most common case of divide-and-conquer. Note that all previous methods impose an equivalent or similar restriction. For example, [Karp 1994] assumes a side condition that specifies a non-increasing property on the exact expected runtime whose validity requires an involved manual inspection. In our algorithms, we instead consider the syntactical restriction of divide-and-conquer.

Non-mutuality. We disallow mutual recursions, which is equivalent to assuming that the call graph of the randomized algorithm modeled by the input SPRR is acyclic (except for self-loops). Note that our setting is strictly more general than all previous methods [Chaudhuri and Dubhashi 1997; Karp 1994], which consider only a single PRR and assume the call graph has a single vertex and is hence non-mutual by definition.

Natural limitations. There are also some natural limitations arising from the problem itself. For example, the existence of a tail bound means that the PRR must terminate almost-surely, i.e. with probability 1. We therefore assume almost-sure termination. This assumption is also shared with all previous approaches.

In summary, while our theoretical results are general, our algorithmic approaches are only applicable to a certain special class of SPRRs that capture common patterns of existing randomized algorithms and the well-known classical examples from the literature, including benchmarks that are beyond the reach of all previous methods. Note that even this special class is more general than the settings studied by the previous methods. For example, we consider systems of PRRs instead of PRRs with a single recurrence. We also show that our method significantly improves over Karp's on classical examples and can obtain bounds that are *asymptotically* tighter. Finding more general algorithmic approaches for more expressive families of SPRRs is an interesting direction for future work.

2 PRELIMINARIES

2.1 Probabilistic Recurrence Relations

We first briefly review necessary concepts in probability theory and then define our problem. We refer to standard textbooks (e.g. [Billingsley 1995; Williams 1991]) for a more detailed treatment.

Discrete probability distributions. A *discrete probability distribution* (DPD) over a countable set U is a function $\eta : U \to [0,1]$ such that $\sum_{u \in U} \eta(u) = 1$. The *support* of η is defined as $supp(\eta) := \{u \in U \mid \eta(u) > 0\}$. We write FSDPD as a short-hand for finite-support DPD.

Probability spaces. A *probability space* is a triple $(\Omega, \mathcal{F}, \Pr)$ where Ω is a non-empty set (called the *sample space*), $\mathcal{F}$ is a *σ-algebra* over Ω (i.e. a collection of subsets of Ω that contains the empty set $\emptyset$ and is closed under complementation and countable union) and $\Pr$ is a *probability measure* on $\mathcal{F}$, i.e. a function $\Pr\colon \mathcal{F} \to [0,1]$ such that (i) $\Pr(\Omega) = 1$, and (ii) for all pairwise-disjoint set-sequences $A_1, A_2, \cdots \in \mathcal{F}$ (i.e. $A_i \cap A_j = \emptyset$ whenever $i \neq j$) it holds that $\Pr\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \Pr(A_i)$. A *random variable* X from a probability space $(\Omega, \mathcal{F}, \Pr)$ is an $\mathcal{F}$-measurable function $X\colon \Omega \to \mathbb{R} \cup \{-\infty, +\infty\}$, i.e. a function such that for all $d \in \mathbb{R} \cup \{-\infty, +\infty\}$, the set $\{\omega \in \Omega \mid X(\omega) < d\}$ lies in $\mathcal{F}$. We denote the expected value of X by $\mathbb{E}[X]$.

Filtrations. A *filtration* of a probability space $(\Omega, \mathcal{F}, \Pr)$ is an infinite sequence $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ of sigma-algebras over Ω such that $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$ for all $n \in \mathbb{N}$. Intuitively, a filtration models the information $\mathcal{F}_n$ available at a specific time point $n \geq 0$.

Discrete-time stochastic processes. A *discrete-time stochastic process* is an infinite sequence $\Gamma = \{X_n\}_{n \in \mathbb{N}}$ of random variables where the random variables X_n are all from the same probability space $(\Omega, \mathcal{F}, \Pr)$. The process Γ is *adapted to* a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ if for all $n \in \mathbb{N}$, X_n is $\mathcal{F}_n$-measurable. Intuitively, the random variable X_n measures the value at the n-th step of the process Γ.

Stopping times. Given a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ with respect to a probability space $(\Omega, \mathcal{F}, \Pr)$, a *stopping time* ρ is a random variable $\rho\colon \Omega \to \{0, 1, 2, 3, \cdots, \infty\}$, such that for every $n \in \{0, 1, 2, \cdots, \infty\}$, the event $\{\omega \mid \rho(\omega) \leq n\}$ is measurable in $\mathcal{F}_n$. In other words, $\{\omega \mid \rho(\omega) \leq n\} \in \mathcal{F}_n$.

Martingales and supermartingales. A discrete-time stochastic process $\Gamma = \{X_n\}_{n \in \mathbb{N}}$ adapted to a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ is a *martingale* (resp. *supermartingale*) if for every $n \in \mathbb{N}$, $\mathbb{E}[|X_n|] < \infty$

and $\mathbb{E}[X_{n+1}|X_0,\ldots,X_n] = X_n$ (resp. $\mathbb{E}[X_{n+1}|X_0,\ldots,X_n] \leq X_n$). Intuitively, a martingale (resp. supermartingale) is a discrete-time stochastic process in which for an observer who has seen the values of $X_0,\ldots,X_n$, the expected value at the next step, i.e. $\mathbb{E}[X_{n+1} \mid X_0,\ldots,X_n]$, is equal to (resp. no more than) the last observed value X_n. Some of our proofs use martingale theory, but the main ideas are accessible to readers unfamiliar with martingales. See Appendix A for a formal treatment.

Then we present the notion of systems of probabilistic recurrence relations (SPRRs). An SPRR is a collection of probabilistic recurrence equations that describe the running-time behavior of a randomized algorithm. Each recurrence equation corresponds to a procedure/function.

SPRR. An SPRR Δ is a finite collection of probabilistic recurrence relations $eq_1,\ldots,eq_m$, which model the running time of a randomized algorithm with m procedures, for which each recurrence equation eq_r $(1 \leq r \leq m)$ corresponds to a procedure in the algorithm (which we shall denote by $proc_r$). Each eq_r $(1 \leq r \leq m)$ takes the form:

$$T_r(n) = \begin{cases} Cost_r(n) + Rec_r(n) & \text{if } n > c_r \\ 0 & \text{if } n \leq c_r \end{cases} \qquad (eq_r)$$

where

$$Rec_r(n) := \left[\sum_{j=1}^{S_r} T_r(h_{r,j}(n))\right] + \left[\sum_{j=1}^{O_r} T_{q_{r,j}}(g_{r,j}(n))\right]$$

specifies the recursion terms, in which:

- n is a variable that represents a non-negative integer modeling the size of the input passed to the procedure $proc_r$, and c_r is a non-negative integer constant that serves as a threshold below which the procedure $proc_r$ halts immediately.
- For each r, $T_r(n)$ is a random variable modelling the running time of $proc_r$ on an input of size n.
- $Cost_r(n)$ is a random variable that models the preprocessing time of $proc_r$ on input size n. We will later treat this preprocessing time as a triggered cost.
- S_r is a constant that represents the number of procedure calls in $proc_r$ to $proc_r$ itself. Among these procedure calls, each $h_{r,j}(n)$ $(0 \leq h_{r,j}(n) \leq n)$ represents the random size of the instance passed to the j-th recursive procedure call to $proc_r$ itself.
- O_r is a constant that represents the number of procedure calls in $proc_r$ to procedures other than $proc_r$. We denote the callee in the j-th such call by $q_{r,j}$. Note that $q_{r,j} \neq r$. Moreover, $g_{r,j}(n)$ $(0 \leq g_{r,j}(n) \leq n)$ is a random variable associated with input size n that models the parameter size passed to the j-th call.

Since $h_{r,j}(n)$ and $g_{r,j}(n)$ both model input sizes to procedure calls, we assume that they observe discrete probability distributions over non-negative integers. Also note that we distinguish between the calls to the procedure itself, i.e. $T_r(h_{r,j}(n))$, and to other procedures, i.e. $T_{q_{r,j}}(g_{r,j}(n))$. Based on (eq_r), we define *non-mutual* SPRRs as SPRRs satisfying the condition that for all $1 \leq r \leq m$, we have $q_{r,j} < r$. Intuitively, these SPRRs do not have mutual recursive calls involving two or more procedures, but they can still have recursion. We assume that the probabilistic dependence and independence between $h_{r,j}(n), g_{r,j}(n)$ and $Cost_r(n)$ are given as part of the SPRR.

Next, we present several examples that model classical randomized algorithms as SPRRs. See Appendix B for more examples and details of the algorithms leading to these SPRRs.

Example 2.1 (QuickSelect). Consider the problem of finding the d-th smallest element in an unordered array of n distinct integers. A classical randomized algorithm for solving this problem is QuickSelect [Hoare 1961b] that provides a simple randomized variant matching the best-known $O(n)$ deterministic runtime. Since there is only one recursive procedure in QuickSelect, we model

the algorithm as the following SPRR with one recurrence equation (where we have $m = 1$, $T_1 = T$, $Cost_1(n) = n$, $c_1 = 1$, $S_1 = 1$ and $O_1 = 0$):

$$T(n) = \begin{cases} n + T(h(n)) & \text{if } n > 1 \\ 0 & \text{if } n \leq 1 \end{cases}.$$

Here, $T(n)$ represents the number of comparisons performed by QuickSelect over an input of size n, and $h(n)$ is the random variable that captures the size of the remaining array that has to be searched recursively, i.e. $h(n) = \max\{i, n-1-i\}$ where i is sampled uniformly from $\{0, \ldots, n-1\}$. □

Example 2.2 (QuickSort). Consider the classical problem of sorting an array of n distinct elements. A well-known randomized algorithm for solving this problem is QuickSort [Hoare 1961a]. As before, there is only one recursive procedure in QuickSort. Thus, we model the algorithm as the following SPRR with one recurrence equation (where we have $m = 1$, $T_1 = T$, $Cost_1(n) = n$, $c_1 = 1$, $S_1 = 2$ and $O_1 = 0$):

$$T(n) = \begin{cases} n + T(h_1(n)) + T(h_2(n)) & \text{if } n > 1 \\ 0 & \text{if } n \leq 1 \end{cases}.$$

Here, $h_1(n)$ and $h_2(n)$ are random variables that capture the sizes of the two sub-arrays, i.e. $h_1(n) = i$ and $h_2(n) = n - 1 - i$ where i is uniformly sampled from $\{0, \ldots, n-1\}$. □

Example 2.3 (SeidelLP). Consider a linear programming problem in d dimensions, i.e. with d variables $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d)$ and n constraints $\mathbf{a}_1^T \cdot \mathbf{x} \leq b_1, \ldots, \mathbf{a}_n^T \cdot \mathbf{x} \leq b_n$. Suppose the objective is to minimize $\mathbf{c}^T \cdot \mathbf{x}$. SeidelLP [Motwani and Raghavan 1995, Chapter 9.10] is a well-known randomized incremental algorithm that handles this problem. To model the algorithm by an SPRR, we set up a system with d equations $eq_1, eq_2, \ldots, eq_d$, where eq_r corresponds to the r-dimensional recursive subinstances of the algorithm. We have $c_r = r$ and for every $n > r$:

$$T_r(n) = Cost_r(n) + T_r(h_{r,1}(n)) + T_{r-1}(g_{r,1}(n))$$

where $h_{r,1}(n) = n - 1$. Moreover, $Cost_r$ and $g_{r,1}$ are dependent and jointly observe the following discrete distribution: with probability $\frac{r}{n}$, $Cost_r(n) = r \cdot n$ and $g_{r,1}(n) = n - 1$, and with probability $1 - \frac{r}{n}$, $Cost_r(n) = r$ and $g_{r,1}(n) = 0$. □

Semantics of SPRRs. Here we describe the formal semantics that treat each random variable $T_r(n)$ as the *accumulated cost* of a stochastic process derived from the underlying SPRR. Below, we fix an SPRR $\Delta = \{eq_1, \ldots, eq_m\}$ where each recurrence equation eq_r takes the form (eq_r). We represent a procedure call σ as an ordered pair (r, n) of non-negative integers where r represents the r-th procedure and n is the size of the parameter passed to this procedure call.

We first introduce several necessary concepts for the semantics. A *stack* β is a finite word $\sigma_1 \ldots \sigma_k$ in which each symbol $\sigma_j = (r_j, n_j)$ is a procedure call. σ_1 is the top of the stack and σ_k is the bottom. We use ε to denote the empty stack. For a stack $\beta = (r, n)\beta'$, we define $Cost(\beta)$ as an independent copy of $Cost_r(n)$, i.e. $Cost(\beta)$ is the preprocessing time of the call at its top. We further define the cost for an empty stack as 0, i.e., $Cost(\varepsilon) := 0$. A *stack transition function* describes the stochastic effect of popping the procedure call at the top of the stack and pushing all its sub-procedure calls. Formally, the *stack transition function* is a function κ that maps every procedure call $\sigma = (r, n)$ to the discrete probability distribution $\kappa(\sigma)$ over stacks such that for each stack β, $\kappa(\sigma)(\beta)$ is the probability that exactly all the procedure calls in β are pushed to the stack while respecting the order they appear in the word β[†]. The exact form of κ is completely determined by the summation

[†]We fix an arbitrary order that is followed when these procedure calls are pushed to the stack.

$Rec_r(n)$ in (eq_r) so it can be calculated exactly from the joint probability distributions of the random variables $h_{r,j}(n)$ and $g_{r,j}(n)$.

Markov chain semantics of SPRRs. The semantics of an SPRR Δ is a discrete-time Markov chain $(S, \mathbf{P})$. The set S of Markov chain states is the set of all stacks. For any two stacks β, β', the transition probability $\mathbf{P}(\beta, \beta')$ is determined by the following two cases:

Case 1: If $\beta = \varepsilon$, then $\mathbf{P}(\beta, \varepsilon) := 1$ and $\mathbf{P}(\beta, \beta') := 0$ for all other stacks $\beta' \neq \varepsilon$.

Case 2: $\beta = \sigma\beta''$ with the top procedure call σ. Then $\mathbf{P}(\beta, \beta') := 0$ if there is no stack γ such that $\beta' = \gamma\beta''$, and $\mathbf{P}(\beta, \beta') := \kappa(\sigma)(\gamma)$ if $\beta' = \gamma\beta''$ for a unique stack γ.

Informally, if the current stack is empty, i.e. the whole SPRR has terminated, which corresponds to Case 1, then the SPRR always stays at the empty stack. Otherwise (Case 2), the SPRR pops the top procedure call of the current stack and stochastically pushes sub-procedure calls back to the top of the stack. Given the Markov chain $(S, \mathbf{P})$ of an SPRR Δ and an initial stack β^*[‡], the probability space $(\Omega_\Delta, \mathcal{F}_\Delta, \Pr_{\Delta,\beta^*})$ for Δ is the one induced by the Markov chain of Δ [Baier and Katoen 2008, Chapter 10]. We denote the expectation under the probability space $(\Omega_\Delta, \mathcal{F}_\Delta, \Pr_{\Delta,\beta^*})$ as $\mathbb{E}_{\Delta,\beta^*}$, and omit the subscript Δ whenever it is clear from the context.

Tail-bound analysis of SPRRs. We now formally define the tail-bound analysis problem over SPRRs in the most general form. Based on our Markov-chain semantics, we first define the random variable C as the accumulated cost along an execution (i.e., an infinite sequence of stacks) of the Markov chain $(S, \mathbf{P})$ of an SPRR Δ. To be more precise, for each infinite sequence of stacks $\rho = \beta_0\beta_1\ldots$, we have $C_k(\rho) := \sum_{n=0}^{k} Cost(\beta_k)$, i.e., $C_k(\rho)$ models the accumulated cost for preprocessing time in the first k steps. We further define the total accumulated cost $C(\rho) := C_\infty(\rho)$. It is straightforward to observe that when the initial stack consists of a single main procedure call (r, n), the random variable C represents the total execution time $T_r(n)$ of the SPRR Δ.

The formal problem statement is as follows: Consider an SPRR Δ, a given initial main procedure r^* and a purely symbolic time limit $\kappa(\alpha, n^*)$, where we symbolically represent n^* as initial input size, and α as a coefficient in the time limit. We also accept concrete time limits as inputs, where we fill the numerical value to the coefficient α.The goal of the tail-bound analysis is to infer an upper-bound function $upper(\alpha, n^*)$, such that for every initial input size n^* and any plausible value α, we have that

$$\Pr_{\Delta,(r^*,n^*)}\left(C \geq \kappa(\alpha, n^*)\right) \leq upper(\alpha, n^*).$$

Informally, $upper(\alpha, n^*)$ bounds the probability that the runtime of the randomized algorithm exceeds the time limit $\kappa(\alpha, n^*)$. Our goal is to find an $upper(\alpha, n^*)$ that tends to 0 as either $n^* \to \infty$ or $\alpha \to \infty$. Note that we introduce the extra parameter α to accommodate the situation that $upper(\alpha, n^*)$ may depend only on α. To compare the asymptotical magnitude of two upper bounds $upper_1(\alpha, n^*)$ and $upper_2(\alpha, n^*)$, we follow the intuitive way. We first treat α as some fixed constant and compare the magnitude over n^*, and if the magnitude over n^* is identical between two bounds, we will take a finer comparison between the coefficient α in two upper bounds and compare the coefficient's magnitude over α.

2.2 Karp's CookBook Method

We now provide a short overview of the classical method in [Karp 1994]. Karp's method shows that tail bounds of a wide class of probabilistic recurrences can be directly computed by a mathematical formula. It solves probabilistic recurrences with a single recurrence equation and deterministic preprocessing time. In our notation, this class of probabilistic recurrences can be formulated as

$$T(n) = Cost(n) + \sum_{i=1}^{S} T(h_i(n)) \tag{1}$$

[‡]Note that β^* is typically a single main procedure call σ^*.

where $Cost(n)$ is the deterministic preprocessing time (i.e., a non-negative function in n), and each $h_i(n)$ represents the random size passed to the i-th sub-procedure call (to the same procedure). It is also assumed that $h_i(n) \leq n$. We further denote by $m_i(n)$ an upper bound for $\mathbb{E}[h_i(n)]$, and treat each m_i as a function on $[0, \infty)$. Using these ingredients as input, Karp obtained the following theorems:

Theorem 2.4 ([Karp 1994, Theorems 1.1–1.2]). *Suppose that the probabilistic recurrence (1) satisfies the prerequisites that (a) $S = 1$, (b) $0 \leq h_1(n) \leq n$ for every n, (c) $m_1(x)$ is continuous and $m_1(x)/x$ is non-decreasing over real values of x, and (d) $Cost(x)$ is continuous and stricly increasing on $\{x \in [0, \infty) \mid Cost(x) > 0\}$. Then, we have*

$$\Pr[T(n) \geq u(n) + w \cdot Cost(n)] \leq \left(\frac{m_1(n)}{n}\right)^w$$

for all $w \in \mathbb{N}$, where $u(n)$ is the solution to the non-probabilistic recurrence $T(n) = Cost(n) + T(m_1(n))$. Moreover, if the condition (d) is replaced by the statement that there exists $d > 0$ such that $Cost(n) = 1$ for $n \geq d$, and $Cost(n) = 0$ for $n < d$, then we have

$$\Pr[T(n) \geq u(n) + w] \leq \left(\frac{m_1(n)}{n}\right)^{w-1} \cdot \frac{m_1(n)}{\min\{n' \mid u(n') \geq u(n)\}} \ .$$

Example 2.5. We now illustrate how theorem 2.4 is applied to derive tails bounds for QuickSelect (Example 2.1). In this example, we can choose $m_1(n) = \frac{3}{4} \cdot n$ as $\mathbb{E}[h_1(n)] = \frac{3}{4} \cdot n$, and we have $u(n) = 4 \cdot n$. It follows straightforwardly that the prerequisites of Theorem 2.4 are satisfied. Thus by the theorem, we obtain that $\Pr[T(n) \geq (4+w) \cdot n] \leq (3/4)^w$. If we are interested in the tail probability $\Pr[T(n) \geq 12 \cdot n]$, we could simply plug in $w = 8$ to obtain $\Pr[T(n) \geq 12 \cdot n] \leq (3/4)^8 \approx 0.1$. □

Theorem 2.6 ([Karp 1994, Theorem 1.5]). *Suppose the probabilistic recurrence (1) satisfies the prerequisite (†) that for every possible combination of concrete sub-instance sizes $(y_1, y_2, \ldots, y_S)$ for $(h_1(n), h_2(n), \ldots, h_S(n))$, we have $\mathbb{E}[T(n)] \geq \sum_{i=1}^{S} \mathbb{E}[T(y_i)]$. Then, for every $w \geq 0$, we have:*

$$\Pr[T(n) \geq (w+1) \cdot \mathbb{E}[T(n)]] \leq \exp(-w) \ .$$

Example 2.7. We illustrate how Theorem 2.6 is applied to derive tail bounds for QuickSort (Example 2.2). By calculating the exact expected runtime of QuickSort through a Harmonic series, one can manually verify that the prerequisite (†) of Theorem 2.6 is satisfied. Thus by the theorem, we have $\Pr[T(n) \geq (w+1) \cdot \mathbb{E}[T(n)]] \leq \exp(-w)$. Suppose we are interested in $\Pr[T(n) \geq 9 \cdot n \ln n]$. Using the fact that $\mathbb{E}[T(n)]$ is slightly less than $2 \cdot n \cdot \ln n$, we can plug in $w = 4$, and obtain $\Pr[T(n) \geq 9 \cdot n \cdot \ln n] \leq \exp(-4) \approx 0.018$. □

Note that Karp's method is only applicable to the case where there is only one recurrence equation, i.e. $m = 1$, in (eq_r), and requires that the preprocessing time $Cost(n)$ is deterministic. In this work, we present approaches through Markov's inequality that lift these restrictions. Our new theory can derive tail bounds that are much tighter, sometimes even asymptotically tighter, than Karp's method and is also able to handle randomized preprocessing time and multiple recurrence equations, i.e. SPRRs.

3 DERIVING TAIL BOUNDS VIA MARKOV'S INEQUALITY

In this section, we present our theoretical approaches for solving the tail-bound problem via Markov's inequality. We first present the backround for Markov's inequality, then our main theorems to derive exponentially and polynomially-decreasing tail bounds, and finally prove these theorems using a general result through Taylor's expansion.

3.1 Background on Markov's Inequality

Markov's inequality provides a way to obtain tail bounds for a non-negative random variable.

Theorem 3.1 (Markov's inequality [Williams 1991, Chapter 6]). *For any non-negative random variable X and real number $a > 0$, we have* $\Pr(X \geq a) \leq \frac{\mathbb{E}(X)}{a}$.

In this work, we use a simple extension of Theorem 3.1 as follows (proof in Appendix C).

Lemma 3.2. *Consider any random variable X and monotonically non-decreasing function $\theta : \mathbb{R} \to [0, \infty)$. For any real number d such that $\theta(d) > 0$, we have* $\Pr(X \geq d) \leq \mathbb{E}[\theta(X)]/\theta(d)$.

To apply Lemma 3.2, we can choose the function θ as an exponential function $\theta(x) := \exp(t \cdot x)$ where $t > 0$. This leads to an exponentially-decreasing bound $\Pr(X \geq d) \leq \mathbb{E}[\exp(t \cdot X)]/\exp(t \cdot d)$ if we can bound $\mathbb{E}[\exp(t \cdot X)]$ from above. We can also set θ to be a monomial function of the form $\theta(x) := x^k$ where k is a non-negative integer, which leads to the polynoimally-decreasing tail bound $\Pr(X \geq d)) \leq \mathbb{E}[X^k]/d^k$ for a non-negative random variable X.

3.2 Main Theorems for Exponentially and Polynomially-Decreasing Tail Bounds

In this section, we first illustrate the main ideas of our approach on a simplified situation where the SPRR contains one recurrence equation with a single recursive call (i.e. $m = 1$, $S_1 = 1$ and $O_1 = 0$ in (eq_r)). We present a result for exponentially-decreasing tail bounds in this simplified case. Then, we extend this simplified situation to the general case and present our main theoretical result for deriving exponentially-decreasing tail bounds. Finally, we present a similar result for polynomially-decreasing tail bounds.

Below, we fix an SPRR Δ with m recurrence equations where the rth recurrence equation is in the form of (eq_r). We also denote the main procedure call by (r^*, n^*).

Theorem 3.3. *Suppose the SPRR Δ involves exactly one recurrence equation with one recursive call:*

$$T(n) = \begin{cases} Cost(n) + T(h(n)) & \text{if } n > c \\ 0 & \text{if } n \leq c \end{cases}. \tag{2}$$

If there are two functions $f, t : \mathbb{N} \to [0, \infty)$ such that

$$\exp(t(n) \cdot f(n)) \geq \mathbb{E}\left[\exp\left(t(n) \cdot (Cost(n) + f(h(n)))\right)\right] \tag{3}$$

for all $c < n \leq n^*$ (where the expectation is taken jointly over the random variables $Cost(n)$ and $h(n)$), then we have the following tail bound for a given time limit $\kappa(\alpha, n^*)$:

$$\Pr_{\Delta,\beta}(C \geq \kappa(\alpha, n^*)) \leq \exp(t_\star \cdot f(n^*) - t_\star \cdot \kappa(\alpha, n^*)) \tag{4}$$

where $t_\star := \min_{1 \leq n \leq n^*} t(n)$.

The intuition behind this theorem is as follows: First, we consider two functions f and t, for which f is the base function for the tail bound (i.e. one needs to choose $\kappa(\alpha, n^*) > f(n^*)$ to make the tail bound less than 1) and t is the function to scale the exponent in the tail bound which is needed in many situations (e.g. in the case $\kappa(\alpha, n^*) = \alpha \cdot n^*$ and the tail bound is only exponentially decreasing in α one needs the scaling function $t(n) = \frac{1}{n}$). Second, the condition (3) states that, with the base function f and the scaling function t, the expected value of $\exp(t(n) \cdot f(n))$ after expanding the recurrence equation one step further is no more than its current value. Third, based on (3), we can conclude that $\mathbb{E}(\exp(t_\star \cdot C)) \leq \exp(t_\star \cdot f(n))$, where $t_\star$ is the minimum value of $t(n)$ over $1 \leq n \leq n^*$. Finally, by applying Lemma 3.2, we obtain the tail bound.

Below we show how Theorem 3.3 can be applied to QuickSelect (Example 2.1).

Example 3.4. Consider the QuickSelect SPRR of Example 2.1. We choose the base function $f(n) := 6 \cdot n$ and the scaling function $t(n) := \frac{1}{n}$. We now verify that (3) holds for all $2 \leq n \leq n^*$:

$$\exp(6) \geq \exp(1) \cdot \tfrac{1}{n} \cdot \left(\textstyle\sum_{i=\lceil n/2 \rceil}^{n-1} \exp\left(\frac{6 \cdot i}{n}\right) + \sum_{i=\lfloor n/2 \rfloor}^{n-1} \exp\left(\frac{6 \cdot i}{n}\right)\right).$$

By calculating the RHS through geometric series, we obtain

$$\exp(1) \cdot \tfrac{1}{n} \cdot \tfrac{2 \cdot \exp(6) - \exp(\frac{6}{n} \cdot \lceil n/2 \rceil) - \exp(\frac{6}{n} \cdot \lfloor n/2 \rfloor)}{\exp(6) - 1}.$$

From the convexity of $\exp(\cdot)$, we have $\exp(\frac{6}{n} \cdot \lceil n/2 \rceil) + \exp(\frac{6}{n} \cdot \lfloor n/2 \rfloor) \geq 2 \cdot \exp(3)$, thus we strengthen the inequality to

$$403.43 \approx \exp(6) \geq \exp(1) \cdot \tfrac{1}{n} \cdot \tfrac{2 \cdot \exp(6) - 2 \cdot \exp(3)}{\exp(6) - 1} \approx \tfrac{5.17}{n}.$$

Hence, (3) holds for all $2 \leq n \leq n^*$. Finally, plugging into (4) with $\kappa(\alpha, n^*) = \alpha \cdot n^*$, we obtain

$$\Pr(T(n^*) \geq \alpha \cdot n^*) \leq \exp(\tfrac{1}{n^*} \cdot (6 - \alpha) \cdot n^*) = \exp(6 - \alpha)$$

where $t_\star = \min_{1 \leq n \leq n^*} t(n) = \frac{1}{n^*}$. This bound is much tighter than that from Karp's method (see Example 2.5). For example, when one has $\alpha = 12$, we obtain $\Pr(T(n^*) \geq 12 \cdot n^*) \leq \exp(-6) \approx 0.002$, which is 50 times better than the bound from Karp's method. □

The previous example shows that our theoretical approach can derive a much tighter tail bound for QuickSelect, but the tail bound is not asymptotically tighter at the exponent. Below we show that by choosing a more suitable $f(n)$ in the application of Theorem 3.3, we can further derive a tail bound that is asymptotically tighter at the exponent.

Example 3.5. Suppose that our goal is to bound the tail probability $\Pr[T(n^*) \geq \alpha \cdot n^*]$ for QuickSelect when $\alpha \geq 8$. We choose the base function $f(n) := \frac{2 \cdot \alpha}{\ln \alpha} \cdot n$, and the scaling function $t(n) := \frac{\ln \alpha}{n}$. The condition (3) could be phrased as follows

$$\exp(2 \cdot \alpha) \geq \exp(\ln \alpha) \cdot \tfrac{1}{n} \cdot \left(\textstyle\sum_{i=\lceil n/2 \rceil}^{n-1} \exp\left(\frac{2 \cdot \alpha \cdot i}{n}\right) + \sum_{i=\lfloor n/2 \rfloor}^{n-1} \exp\left(\frac{2 \cdot \alpha \cdot i}{n}\right)\right).$$

Then we firstly calculate two geometric series in the right-hand-side, simplify $\exp(\ln \alpha)$ as α, and finally remove floors and ceils through convexity of $\exp(\cdot)$: $\exp(\frac{2 \cdot \alpha \cdot \lceil n/2 \rceil}{n}) + \exp(\frac{2 \cdot \alpha \cdot \lfloor n/2 \rfloor}{n}) \geq 2 \cdot \exp(\frac{2 \cdot \alpha}{2}) = 2 \cdot \exp(\alpha)$, obtaining the rephrased condition as follows:

$$\exp(2 \cdot \alpha) \geq \alpha \cdot \tfrac{2}{n} \cdot (\exp(2 \cdot \alpha) - \exp(\alpha))) \cdot \tfrac{1}{\exp(2 \cdot \alpha / n) - 1}.$$

By dividing $\exp(2 \cdot \alpha)$ on both sides, we have that:

$$1 \geq \tfrac{2 \cdot \alpha}{n} \cdot (1 - \exp(-\alpha)) \cdot \tfrac{1}{\exp(2 \cdot \alpha / n) - 1}.$$

We could further strengthen the condition by $\exp(2 \cdot \alpha / n) - 1 \geq 2 \cdot \alpha / n$, obtaining that:

$$1 \geq (1 - \exp(-\alpha)).$$

which trivially holds for every $\alpha \geq 8$. Hence we have that:

$$\Pr[T(n^*) \geq \alpha \cdot n^*] \leq \exp(2 \cdot \alpha - \alpha \ln \alpha) = \exp(-\Theta(\alpha \ln \alpha)).$$

Recall that Karp's method only derives a tail bound of $\exp(-\Theta(\alpha))$, thus is asympotically worse than our approach in the exponent by a $\ln \alpha$ factor. □

Remark. The example above shows that our approach can obtain bounds that are considerably tighter than Karp's method. Moreover, we will now extend the results to systems of PRRs consisting of several recursive equations (modeling recursive algorithms with several procedures). This is also beyond the reach of Karp's method and all previous methods. However, our approach is unable to handle a program, called CouponCollector[Motwani and Raghavan 1995, Chapter 3.6] as follows.

Example 3.6. Consider the Coupon-Collector problem 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 the rounds continue until all the n types of coupons are collected. We model the rounds as a recurrence relation with two variables n, m, where n represents the total number of coupon types and m represents the remaining number of uncollected coupon types. The recurrence relation is as follows:

$$T(n, 0) = 0$$
$$T(n, m) = T(n, h(m)) + 1$$

where $h(m)$ observes as $m - 1$ with probability m/n, and observes as m with probability $1 - m/n$.

Our approach fails in case since it is hard to apply Markov's inequality to derive exponentially decreasing tail bounds. However, Karp's method could produce the following:

$$\Pr[T(n, n) \geq n \ln n + \alpha \cdot n] \leq \exp(-\alpha).$$

□

To extend Theorem 3.3 to an SPRR in general form, we first generalize the condition (3) by incorporating multiple recursive calls at the RHS of (3). Then, we have a condition in the form of (3) at every recurrence equation in the SPRR, so we can use $2 \cdot m$ functions $t_1, \ldots, t_m$ and $f_1, \ldots, f_m$ (for the m recurrence equations in an SPRR) where each t_r and f_r serve as the functions t, f in (3) for the rth recurrence equation. This amounts to an extension as follows:

Theorem 3.7 (Exponentially-Decreasing Tail Bounds). *Consider $2 \cdot m$ functions $f_1, \ldots, f_m$ and $t_1, \ldots, t_m$, all from $\mathbb{N}$ into $[0, \infty)$. Suppose that for every $1 \leq r \leq m$ and $c_r < n \leq n^*$ we have*

$$\exp(t_r(n) \cdot f_r(n)) \geq \mathbb{E}\left[\exp\left(t_r(n) \cdot (Cost_r(n) + Rec'_r(n))\right)\right] \tag{5}$$

where $Rec'_r(n)$ is the term derived from $Rec_r(n)$ by substituting each occurrence of T_r with f_r, i.e.

$$Rec'_r(n) := \textstyle\sum_{i=1}^{S_r} f_r(h_{r,i}(n)) + \sum_{i=1}^{O_r} f_{q_{r,i}}(g_{r,i}(n)).$$

Then, we have the tail bound

$$\Pr_{\Delta,\beta}(C \geq \kappa(\alpha, n^*)) \leq \exp(t_\star \cdot f_{r^*}(n^*) - t_\star \cdot \kappa(\alpha, n^*))$$

where $t_\star := \min_{1 \leq r \leq m} \min_{1 \leq n \leq n^*} t_r(n)$.

The intuition for Theorem 3.7 is similar to Theorem 3.3, except for that we have the whole term $Rec'_r(n)$ in (5) instead of a single $f(h(n))$ in (3), and we need to have the condition (3) for every recurrence equation.

We now turn to polynomially-decreasing tail bounds. As mentioned below Lemma 3.2, we choose the function θ in the application of Lemma 3.2 to be $\theta(x) = x^k$ for some non-negative integer k. Based on the choice of k, our theorem for polynomially-decreasing tail bounds considers the expectation of all the higher moments of degree no more than k for each procedure T_r (i.e., $\mathbb{E}[T_r(n)^i]$ for all $0 \leq i \leq k$). This is because all monomials of degree k or less appear in the Taylor expansion of $(x + a)^k$. Specifically, we consider $m \cdot (k + 1)$ functions $f_{r,i}$ ($1 \leq r \leq m, 0 \leq i \leq k$) for which each $f_{r,i}(n)$ intuitively represents an over-approximation of $\mathbb{E}[T_r(n)^i]$. We have the following theorem:

Theorem 3.8 (Polynomially-Decreasing Tail Bounds). *Let k be a non-negative integer that represents the maximal degree of a monomial. Consider $m \cdot (k + 1)$ functions $f_{r,i} : \mathbb{N} \to [0, \infty)$ ($1 \leq r \leq m, 0 \leq i \leq k$). Suppose that for every $1 \leq r \leq m$ and $1 \leq i \leq k$, the following conditions hold for all $c_r + 1 \leq n \leq n^*$:*

$$f_{r,i}(n) \geq \mathbb{E}\left[Binom_{r,i}(n, f)\right] \tag{6}$$
$$f_{r,0}(n) \geq 1 \tag{7}$$

where the expression $Binom_{r,i}(n, f)$ is obtained by first expanding the i-th power of the RHS of (eq_r) (i.e. $(Cost_r(n) + Rec_r(n))^i$) into a summation of products of $Cost_r(n)$, $T_r(h_{r,i}(n))$ and $T_{q_{r,i}}(g_{r,i}(n))$, and then substituting each occurrence of $T^j_{r'}$ with its corresponding over-approximation $f_{r',j}$ in the summation obtained above. Then we have the tail bound

$$\Pr(T(n^*) \geq \kappa(\alpha, n^*)) \leq f_{r^*,k}(n^*)/\kappa(\alpha, n^*)^k.$$

We now present the intuition behind Theorem 3.8. First, (6) states that under the substitution of each $T^j_{r'}$ by its over-approximation $f^j_{r'}$, the expected value of the i-th power of the binomial expansion of the RHS of (eq_r) is no more than the current value of $f_{r,i}(n)$. Note that in the binomial expansion $Binom_{r,i}(n)$ of the i-th power, one needs to further consider $f_{r,j}(n)$'s for all $0 \leq j \leq i$. Second, the condition (7) imposes a boundary condition to ensure that $f_{r,0}(n)$ over-approximates $\mathbb{E}[T_r(n)^0]$, which is 1. The two conditions ensure that $\mathbb{E}(T_{r^*}(n^*)) \leq f_{r^*,k}(n^*)$. Finally, similarly to the exponential case, we apply Lemma 3.2 to get the tail bound with $\theta(x) = x^k$.

3.3 Proof Sketches

In this section, we provide short proof sketches for our main theorems, i.e. Theorems 3.7 and 3.8. The proofs are carried out by first establishing an auxiliary proposition that tackles tail bounds in the form of a non-negative Taylor function, and then applying the proposition to exponential and monomial functions. Below we fix an SPRR Δ.

We say that a function $\theta : [0, \infty) \to [0, \infty)$ is a *non-negative Taylor function* if it can be expanded into the Taylor series at zero with non-negative coefficients. Note that any non-negative Taylor function θ is monotone, and can be applied to derive tail bounds through Lemma 3.2. It is also worth noting that both exponential functions of the form $\theta(x) = \exp(t \cdot x)$ $(t > 0)$ and monomial functions of the form $\theta(x) = x^k$ $(k \in \mathbb{N})$ are non-negative Taylor functions.

We define the notion of *expectation non-increasing Taylor function sequences* (ENTFS's) as follows. An ENTFS is an infinite sequence Ψ of functions $\Psi = \Psi_0, \Psi_1, \Psi_2, \ldots$ such that (i) each Ψ_k maps each stack β of Δ to a non-negative real value $\Psi_k(\beta)$ and (ii) for every stack β and $k \geq 0$, we have

$$\Psi_k(\beta) \geq \mathbb{E}\left[\sum_{j=0}^{\infty} \frac{\Psi_{k+j}(next_\beta)}{j!} \cdot Cost^j(\beta)\right]. \tag{8}$$

where $next_\beta$ is a random variable that represents the random stack at the next step. i.e. $next_\beta$ observes the probability distribution $\mathbf{P}(\beta, -)$.

The intuition behind (8) is that, with a slight misuse of notation, the accumulated cost starting from the stack β is informally equal to "$next_\beta + Cost(\beta)$" interpreted as the sum of the accumulated cost starting from $next_\beta$ and the cost $Cost(\beta)$ triggered at the current step. The RHS in (8) corresponds to the Taylor expansion of $\Psi_k(next_\beta + Cost(\beta))$ at $next_\beta$. Hence, the whole condition (8) specifies that the expectation of expanding Ψ_k into its Taylor series at the next random stack $next_\beta$ with the preprocessing time $Cost(\beta)$ of the current stack β is no more than its current value $\Psi_k(\beta)$. In other words, (8) combines a supermartingale (non-increasing expectation) condition with the Taylor expansion. We now present our general proposition for deriving tail bounds from an ENTFS.

Proposition 3.9. *Consider a non-negative Taylor function θ and an ENTFS $\Psi = \{\Psi_k\}_{k \geq 0}$. Suppose*

$$\Psi_k(\varepsilon) \geq \theta^{(k)}(0) \text{ for every } k \geq 0. \tag{9}$$

Then for every stack β and $k \geq 0$, we have $\Psi_k(\beta) \geq \mathbb{E}_\beta\left[\theta^{(k)}(C)\right]$. By Lemma 3.2, we obtain the tail bound

$$\Pr_{\sigma^*}(C \geq \kappa) \leq \frac{\mathbb{E}_{\sigma^*}[\theta(C)]}{\theta(\kappa)} \leq \frac{\Psi_0(\sigma^*)}{\theta(\kappa)}$$

for any initial procedure call σ^*.

This theorem states that if we can construct an ENTFS that bounds the derivatives $\theta^{(k)}(0)$ as in (9), then we can bound $\mathbb{E}_\beta\left[\theta^{(k)}(C)\right]$ by $\Psi_k(\beta)$, and hence bound $\mathbb{E}_\beta\left[\theta(C)\right]$ by $\Psi_0(\beta)$. The intuition for (9) is that it specifies the boundary condition where the k-th order derivative of θ at zero should be bounded by the value of Ψ_k at the empty stack. A proof sketch is as follows:

Proof Sketch. We prove the result using well-known theorems from martingale theory [Williams 1991, Chapter 10]. Let $\{\widehat{\beta_k}\}_{k\geq 0}$ be the random trace of the SPRR where each $\widehat{\beta_k}$ is the random stack representing the stack at the kth step in the execution of the SPRR. We first construct a sequence $\{Y_{k,i}\}_{i\geq 0}$ of random variables defined as $Y_{k,i} := \sum_{j\geq 0} \frac{\Psi_{k+j}(\widehat{\beta_i})}{j!}\cdot C_i^j$. From (8), we know that $\{Y_{k,i}\}_{i\geq 0}$ is a supermartingale, i.e. for each i, the expected value of $Y_{k,i+1}$ given the value of $Y_{k,i}$ is no greater than $Y_{k,i}$. Next, by applying the Optional Stopping Theorem [Williams 1991, Chapter 10.10], we prove that $\mathbb{E}[Y_{k,0}] \geq \mathbb{E}[Y_{k,\tau}]$, where τ is a stopping time representing the total number of pre-processing stages in an execution of the SPRR. i.e., $\tau := \inf\{k : \widehat{\beta_k} = \varepsilon\}$, Note that $Y_{k,0} = \Psi_k(\beta)$ and

$$Y_{k,\tau} = \sum_{j=0}^{\infty} \frac{\Psi_{k+j}(\varepsilon)}{j!}\cdot C^j \geq \sum_{j=0}^{\infty} \frac{\theta^{(k+j)}(0)}{j!}\cdot C^j = f^{(k)}(C)$$

from (9). Finally, by Lemma 3.2, we obtain the desired result. See Appendix C for a detailed proof and the background from the theory of martingales. □

We now show how to instantiate Proposition 3.9 to prove our main theorems.

Proof Sketch of Theorem 3.3 and 3.7. Suppose that (3) holds. We construct an ENTFS $\{\Psi_i\}_{i\geq 0}$ that satisfies (8) and (9) with respect to the exponential function $\theta(x) = \exp(t_\star \cdot x)$. The ENTFS $\{\Psi_i\}_{i\geq 0}$ is constructed as follows: given any stack $\beta = (r, n_1)(r, n_2)\ldots(r, n_k)$ where $r = 1$, since we only have one recurrence equation, we define $\Psi_i(\beta) := t_\star^i \cdot \exp(t_\star \cdot \sum_{j=1}^{k} f(n_j))$. The boundary condition (9) can be verified straightforwardly. For proving the expectation-nonincreasing condition (8), we use (3) and the convexity of $\exp(\cdot)$. This shows that the Ψ_i's are indeed an ENTFS. Thus by Proposition 3.9, we have $\mathbb{E}(\exp(t_\star \cdot C)) \leq \Psi_0(\sigma^*) = \exp(t_\star \cdot f(n^*))$. Finally, we apply Lemma 3.2 to obtain the desired result.

The proof of Theorem 3.7 is almost the same with Theorem 3.3. We only need to simply extend the stack β and the ENTFS Ψ_i's to the case where there are more than one recurrence relations, and the derivation is the same. □

Proof Sketch of Theorem 3.8. The proof for Theorem 3.8 is similar to Theorem 3.3, with the major difference that we construct an ENTFS for $\theta(x) = x^k$ in the application of Proposition 3.9. □

Due to space limitations, detailed proofs of Theorems 3.3, 3.7 and 3.8 are provided in Appendix C.

4 ALGORITHMIC APPROACHES

Our theoretical approaches proposed in the previous section require an involved manual analysis if they are used directly. To resolve this issue, we propose automated algorithmic approaches in this section. To obtain automated and efficient approaches, we first make practical assumptions on the input SPRRs. Based on these assumptions, we illustrate our algorithms which are based on pseudo-polynomial templates, i.e. univariate polynomials extended with logarithmic terms.

A *pseudo-monomial* is a single product of the form $d \cdot n^a \cdot \log^b n$ (e.g. $n\log n$, $n^2\log n$, $n\log^2 n$) where d is a constant coefficient, $a, b \in \mathbb{Z}$, and n is the variable that represents the input instance size. A *pseudo-polynomial* is a finite sum of pseudo-monomials. We always use the natural logarithm in pseudo-polynomials. We employ pseudo-polynomials as templates in our algorithms. The main reason why we utilize pseudo-polynomials is that the runtime behavior of classical randomized algorithms can often be captured by pseudo-polynomials.

4.1 Practical Assumptions

We make assumptions on both the structure of the SPRRs and the probability distributions that appear in them.

First, we have *structural* assumptions on the input SPRR as follows.

- *Non-mutuality*: We prohibit mutual recursion and assume that the call graph among the procedures of the SPRR is acyclic, except for potential recursive calls made by a procedure to itself. Formally, for every $q_{r,j}$ in a recurrence equation (eq_r), we assume $q_{r,j} < r$.
- *Single recursion or divide-and-conquer*: We assume that for each recurrence equation (eq_r), either (i) it involves only one self-recursive call or (ii) its self-recursion part is a divide-and-conquer that involves two recursive calls to itself for which the sum of the passed sizes in these two recursive calls is equal to the original passed size subtracted by one. Note that we do not impose any restriction on the recursive calls to other procedures here. Formally, we assume in each recurrence equation (eq_r) either $S_r = 1$, or $S_r = 2$ and $h_{r,1}(n) + h_{r,2}(n) = n - 1$.

Note that most randomized algorithms fulfill this assumption. For non-mutuality, many randomized algorithms either contain only one procedure and thus are non-mutual by definition, or fix a hierarchy on the call graph of the procedures so that the resultant SPRR is naturally non-mutual. For single recursion/divide-and-conquer, mostly a procedure in the randomized algorithm calls to itself only once or twice and this is done in the divide-and-conquer fashion.

Second, we make assumptions on the *probability distributions* in the input SPRR. We consider common probability distributions that arise in classical randomized algorithms. Roughly speaking, we consider either discrete probability distributions whose support contains a fixed number of elements, or (piecewise) uniform distributions over the range $\{0, \ldots, n-1\}$. This covers most commonly-used probability distributions in randomized algorithms. The details are as follows.

- We assume $Cost_r(n)$ is sampled from an FSDPD, whose every element (in its support) is a pseudo-polynomial. Moreover, the probability assigned to each element in the distribution takes the form of either a constant, $\frac{c}{n}$, or $1 - \frac{c}{n}$ where c is a constant.
- Our algorithms consider three types of probability distributions for $h_{r,j}(n)$'s and $g_{r,j}(n)$'s in every recurrence equation (eq_r) below.
  - The first type is that of an FSDPD in which every element is either a constant or one of the expressions $n, n-1, \lceil n/2 \rceil, \lfloor n/2 \rfloor$. Moreover, the probability assigned to each element is either a constant or an expression from $\frac{c}{n}$ or $1 - \frac{c}{n}$ where c is a constant.
  - The second type is the uniform distribution over $\{0, 1, \ldots, n-1\}$.
  - The third type is that the value sampled from the distribution is $\max\{i, n-i-1\}$ for which the index i is distributed uniformly over $\{0, 1, \ldots, n-1\}$. This distribution is a piecewise uniform distribution and we call it the Muniform distribution.

Third, we follow the convention of *independence* assumptions in the input SPRR. In the literature [Chaudhuri and Dubhashi 1997; Karp 1994; Karpinski and Zimmermann 1991; Tassarotti 2017], only a single recurrence equation is considered and the independence between the preprocessing time and the sub-instance sizes (of the self-recursion) is naturally assumed. In our algorithms, we assume the independence between (i) $h_{r,j}(n)$'s and (ii) $Cost_r(n)$, $g_{r,j}(n)$'s in each (eq_r), and we allow a certain level of dependence between the preprocessing time $Cost_r(n)$ and the sub-instance sizes $g_{r,j}(n)$'s. For the latter, we formallly assume that either $Cost_r(n)$ is independent of the $g_{r,j}$'s, or $Cost_r(n)$ and the $g_{r,j}$'s are dependent but the probability distributions they observe are FSDPDs. This assumption allows our algorithms to handle several randomized algorithms in computational geometry (e.g. [Seidel 1991]) that require such dependence.

Finally, we assume *almost-sure termination* for input SPRRs. This assumption is natural since the existence of a tail bound implies almost-sure termination. Theorem 3.7 and Theorem 3.8 implicitly guarantee almost-sure termination as each of the conditions (5)–(7) implies $\Pr(T(n) < \infty) = 1$.

4.2 ExpSyn: An Algorithm for Deriving Exponentially-Decreasing Bounds

Based on Theorem 3.7, we propose an algorithm ExpSyn to derive exponentially-decreasing tail bounds. Recall that to apply the theorem, it suffices to synthesize $2 \cdot m$ functions $f_1, \ldots, f_m$ and $t_1, \ldots, t_m$ that satisfy the condition (5). To highlight the main ideas, we first present our algorithm in the simplified case of a single recurrence with only one recursive call to itself, and then generalize it to all SPRRs that fall into our practical assumptions.

The Simplified Case. Consider an input SPRR in the form of (2), i.e. only one recurrence equation with a single recursive call to itself. Our algorithm also reads the input time limit $\kappa(\alpha, n)$ in the form of a pseudo-polynomial with the (only) key coefficient parameter α. We apply Theorem 3.3 and try to synthesize functions $f(n)$ an $t(n)$ satisfying (3):

$$\exp(t(n) \cdot f(n)) \geq \exp(t(n) \cdot (\mathit{Cost}(n) + f(h(n)))).$$

We define $X(n) := -f(n) + \mathit{Cost}(n) + f(h(n))$, and simplify our inequality to:

$$\mathbb{E}\left[\exp(t(n) \cdot X(n))\right] \leq 1. \tag{10}$$

The goal of our algorithm is to solve (10) using the steps below.

Step 1: Obtaining $f(n)$. In the first step, the algorithm receives an extra input function $f(n)$ in the form of a pseudo-monomial. Alternatively, the choice of $f(n)$ can be easily automated by exhaustively searching a monomial around the magnitude of the input $\kappa(\alpha, n)$ and then repeatedly raising the coefficient of the monomial until the function $f(n)$ satisfies (3) or certain threshold for the coefficient is exceeded. The automated search for a $f(n)$ is guaranteed by the following lemma.

Lemma 4.1. *Suppose that the condition* (3) *holds for a non-negative and monotonically-nondecreasing function* $f(n)$. *Then for every* $\beta \geq 1$, *we have that the condition* (3) *holds for the function* $\beta \cdot f(n)$.

Proof. We have the following:

$$\begin{aligned}
& \mathbb{E}[\exp(t(n) \cdot (\mathit{Cost}(n) + \beta \cdot f(h(n))))] \\
= \;& \text{§ from independence §} \\
& \mathbb{E}[\exp(t(n) \cdot \mathit{Cost}(n))] \cdot \mathbb{E}[\exp(\beta \cdot t(n) \cdot f(h(n)))] \\
= \;& \text{§ from } h(n) \leq n \text{ §} \\
& \mathbb{E}[\exp(t(n) \cdot \mathit{Cost}(n))] \cdot \left(\textstyle\sum_{k=0}^{n} \mathbb{P}(h(n) = k) \cdot (\exp(t(n) \cdot f(k)))^{\beta}\right) \\
\leq \;& \text{§ from the monotonicity of } f(n) \text{ and non-negativity of } t(n) \text{ §} \\
& \mathbb{E}[\exp(t(n) \cdot \mathit{Cost}(n))] \cdot (\exp(t(n) \cdot f(n)))^{\beta-1} \cdot \left(\textstyle\sum_{k=0}^{n} \mathbb{P}(h(n) = k) \cdot (\exp(t(n) \cdot f(k)))\right) \\
= \;& \mathbb{E}[\exp(t(n) \cdot \mathit{Cost}(n))] \cdot (\exp(t(n) \cdot f(n)))^{\beta-1} \cdot \mathbb{E}[\exp(t(n) \cdot f(h(n)))] \\
= \;& (\exp(t(n) \cdot f(n)))^{\beta-1} \cdot \mathbb{E}[\exp(t(n) \cdot (\mathit{Cost}(n) + f(h(n))))] \\
\leq \;& \text{§ from (3) §} \\
& (\exp(t(n) \cdot f(n)))^{\beta-1} \cdot \exp(t(n) \cdot f(n)) \\
= \;& \exp(t(n) \cdot \beta \cdot f(n))
\end{aligned}$$

□

The lemma above can be directly extended to (5) by applying almost the same proof and using the independence between $h_{r,j}(n)$'s and $Cost(n)$, $g_{r,j}$'s in (eq_r), and assuming in the case $S_r = 2$ that $f_r(k) + f_r(n-1-k) \leq f_r(n)$. Note that the last condition holds if $f_r(n)$ is a pseudo-monomial $n^a \cdot \ln^b n$ where $a \geq 1$ and $b \geq 0$ (Proposition D.2).

Example 4.2. We use QuickSelect (Example 2.1) as our running example and consider deriving a tail bound for $\Pr(T(n^*) \geq \alpha \cdot n^*)$ where $\kappa(\alpha, n) = \alpha \cdot n$ For the first step, we choose the function $f(n) := 6 \cdot n$ and we will show that this choice of $f(n)$ suffices to derive a tail bound much tighter than Karp's method. Alternatively, one could choose the magnitude $O(n)$ around $\kappa(\alpha, n)$ and repeatedly raising the coefficient of $f(n)$ to search for a suitable $f(n)$. □

Step 2: Calculating $X(n)$ ***and Bounding it by*** $B(n)$. Following the first step, the algorithm symbolically calculates $X(n)$ and finds a function $B(n)$ such that $|X(n)| \leq B(n)$. Since $f(n)$ is a pseudo-monomial, we naturally choose $B(n)$ also to be a pseudo-monomial. A bound $B(n)$ can be easily synthesized by examining the behavior of $X(n)$ when $n \to \infty$ using basic calculus. See Appendix D for details.

Example 4.3. For QuickSelect, we have $X(n) = -f(n) + Cost(n) + 6 \cdot h(n)$. We plug in $Cost(n) = n$ and $f(n) = 6 \cdot n$, and get $X(n) = -6 \cdot n + n + 6 \cdot h(n)$. We then bound $|X(n)|$ by $B(n) = 5 \cdot n$. □

Step 3: Setting up a template for $t(n)$. In this step, the algorithm sets up a template for the scaling function $t(n)$. We choose the template simply to be $t(n) := \lambda / B(n)$ with an unknown coefficient λ that we have to synthesize. The intuition is that $t(n)$ scales the bound function $B(n)$ to be a constant, so that we can apply Taylor expansion to over-approximate $\mathbb{E}[\exp(t(n) \cdot X(n))]$ in (10). Note that although the template is simple, the resolution of the unknown coefficient λ is still non-trivial since the constraint in (3) is exponential.

Example 4.4. We set the template $t(n) := \lambda / (5 \cdot n)$. □

Step 4: Binary search. The goal of this step is to find a concrete value for the unknown coefficient λ in $t(n)$ to satisfy (10). The algorithm simply performs a binary search within a prescribed range $[0, M]$ to find the largest possible value for λ. We find the largest possible value for λ since a larger λ leads to a tighter tail bound. The legitimacy of the binary search follows from Lemma 4.5 below. To aid the binary search, we need a verification procedure to check whether a concrete value for λ satisfies (10), which is resolved later.

Lemma 4.5 (Proof in Appendix H). *Consider two concrete values $\lambda_1 > \lambda_2$ for λ. If (10) holds by setting $\lambda := \lambda_1$, then it also holds when $\lambda := \lambda_2$.*

Example 4.6. Continuing with Example 4.4, the algorithm assumes a prescribed range for λ and perform binary search within this range. As an example, assume that the range is $[0, 16]$. To begin with the binary search, we first check whether (10) holds for $\lambda = 8$. If it indeed holds, then the range λ would be reduced to $[8, 16]$, otherwise the range would be reduced to $[0, 8]$. Through the binary search, our algorithm outputs an optimal λ which is roughly $\lambda \approx 7.65$. □

As mentioned previously, in order to complete the binary search, we need a verification procedure that decides whether a given concrete value for λ satisfies (10). This is the most involved part of our algorithm. Below we provide a sketch covering the main points and ideas, and relegate the technical details to Appendices E-H.

Verification procedure. Given a concrete value $\overline{\lambda}$ for λ so that $t(n) = \overline{\lambda} / B(n)$, the overall goal of the verification procedure is to check whether (10) holds for all $c + 1 \leq n \leq n^*$. The algorithm first over-approximates the exponential term in the LHS of (10) through Taylor expansion. From a

Taylor expansion with Lagrangian remainder of depth $L+1$, we have that for every $x \geq 0$ there exists $\xi \in [0, x]$ such that $\exp(x) = 1 + \sum_{i=1}^{L} \frac{x^i}{i!} + \frac{\exp(\xi)}{(L+1)!} \cdot x^{L+1}$. Suppose $-D \leq x \leq D$ and L is an odd number. Then, x^{L+1} is always non-negative and thus we have: $\exp(x) \leq 1 + \sum_{i=1}^{L} \frac{x^i}{i!} + \frac{\exp(D)}{(L+1)!} \cdot x^{L+1}$. Applying this inequality to $\exp(t(n) \cdot X(n))$ with $|t(n) \cdot X(n)| \leq t(n) \cdot B(n) = \overline{\lambda}$, we get

$$\exp(t(n) \cdot X(n)) \leq 1 + \sum_{i=1}^{L} \frac{t(n)^i}{i!} \cdot X(n)^i + \frac{\exp(\overline{\lambda}) \cdot t(n)^{L+1}}{(L+1)!} \cdot X(n)^{L+1}.$$

So, to guarantee $\mathbb{E}[\exp(t(n) \cdot X(n))] \leq 1$, it suffices to show

$$\mathbb{E}\left[1 + \sum_{i=1}^{L} \frac{t(n)^i}{i!} \cdot X(n)^i + \frac{\exp(\overline{\lambda}) \cdot t(n)^{L+1}}{(L+1)!} \cdot X(n)^{L+1}\right] \leq 1,$$

which is equivalent to

$$\sum_{i=1}^{L} \frac{t(n)^i}{i!} \cdot \mathbb{E}\left[X(n)^i\right] + \frac{\exp(\overline{\lambda}) \cdot t(n)^{L+1}}{(L+1)!} \cdot \mathbb{E}\left[X(n)^{L+1}\right] \leq 0. \tag{11}$$

We have now reduced the problem to deciding (11). The verification procedure first apply our assumptions on distribution that over-approximates each higher-moment expectation $\mathbb{E}\left[X(n)^i\right]$ up to any given error parameter $\epsilon > 0$ by a pseudo-polynomial expression $A_i(n)$*, and then checks whether the relaxed inequality (that involves only a comparison between pseudo-polynomials) holds.

The approximation of the higher-moment expectations $\mathbb{E}\left[X(n)^i\right]$ is composed of the following steps (Step A1 – Step A4):

Step A1: Binomial Expansion. We first apply binomial expansion to $X(n)^i$. Recall that $X(n) = -f(n) + Cost(n) + f(h(n))$. This results in a summation of products of $-f(n)$, $Cost(n)$ and $f(h(n))$. Using linearity of expectation, the approximation of $\mathbb{E}[X(n)^i]$ is obtained by separately approximating the expectation of each product term in the summation. Using the independence between $Cost(n)$ and $f(h(n))$, the expectation of each product term in the summation can be approximated by separately approximating the higher-moment expectations $\mathbb{E}[Cost(n)^j]$ and $\mathbb{E}[f(h(n))^j]$.

Example 4.7. Continue with Example 4.6. We show how the algorithm over-approximates $\mathbb{E}\left[X(n)^2\right]$. First, the algorithm expands $X(n)^2$ to $f(n)^2 + Cost(n)^2 + f(h(n))^2 - 2 \cdot f(n) \cdot Cost(n) - 2 \cdot f(n) \cdot f(h(n)) + 2 \cdot Cost(n) \cdot f(h(n))$. Then, plugging in $f(n) = 6 \cdot n$, it expands $\mathbb{E}[X(n)^2]$ as $\mathbb{E}[X(n)^2] = 36 \cdot n^2 + \mathbb{E}[Cost(n)^2] + \mathbb{E}[f(h(n))^2] - 12 \cdot \mathbb{E}[n \cdot f(h(n))] - 12 \cdot \mathbb{E}[n \cdot Cost(n)] + 2 \cdot \mathbb{E}[Cost(n) \cdot f(h(n))]$, and approximates each summand separately. By the independence, it rewrites $\mathbb{E}[Cost(n) \cdot f(h(n))]$ as $\mathbb{E}[Cost(n)] \cdot \mathbb{E}[f(h(n))]$ and approximates $\mathbb{E}[Cost(n)]$ and $\mathbb{E}[f(h(n))]$ independently. □

Step A2: Approximating $\mathbb{E}[Cost(n)^j]$. To approximate $\mathbb{E}[Cost(n)^j]$ for $j > 0$, we rely on the assumption that $Cost(n)$ observes an FSDPD. So, we can simply iterate over its finite support and compute $\mathbb{E}[Cost(n)^j]$ exactly as a pseudo-polynomial.

Example 4.8. In QuickSelect, $Cost(n) = n$. So we have $\mathbb{E}[X(n)^2] = \mathbb{E}[f(h(n))^2] - 10 \cdot n \cdot \mathbb{E}[f(h(n))] + 25 \cdot n^2$. □

Step A3: Approximating $\mathbb{E}[f(h(n))^j]$. Recall that we assumed $h(n)$ is either an FSDPD, or the uniform distribution over $\{0, \ldots, n-1\}$ or the Muniform distribution. The algorithm handles each of these three cases by a detailed calculation specific to the distribution. For an FSDPD, we can reuse the

*To be more precise, the approximation works only when n is large enough, and we simply calculate its exact value when n is small.

treatment for $Cost(n)$. For the uniform distribution, we observe that $\mathbb{E}[f(h(n))^j]$ is a consecutive summation of pseudo-monomials as

$$\frac{1}{n} \cdot \sum_{i=0}^{n-1} i^a \ln^b i. \tag{$*$}$$

since $f(n)$ is a pseudo-monomial. One can obtain an approximation by the corresponding integral $\int_0^n x^a \ln^b x \, dx$, which can be calculated exactly as a pseudo-polynomial using integration by parts. To derive precise upper and lower bounds for $\mathbb{E}[f(h(n))^j]$, we will further strengthen this approximation through an iteration-based approach. As the number of iterations increases, our algorithm could reach arbitrarily small additive error. See Appendix E for details. We treat Muniform similarly to the uniform distribution, expect that in this case we need to approximate a sum of two consecutive summations of pseudo-monomials:

$$\frac{1}{n} \cdot \left[\left(\sum_{i=\lceil n/2 \rceil}^{n-1} i^a \ln^b i \right) + \left(\sum_{i=\lfloor n/2 \rfloor}^{n-1} i^a \ln^b i \right) \right].$$

Our algorithm approximates the two summations separately.

Example 4.9. Following Example 4.8, we consider $\mathbb{E}[f(h(n))^2] = 36 \cdot \mathbb{E}[h(n)^2]$. Note that $h(n)$ observes a Muniform distribution and $\mathbb{E}[h(n)^2] = \frac{1}{n} \cdot \left(\sum_{i=\lceil n/2 \rceil}^{n-1} i^2 + \sum_{i=\lfloor n/2 \rfloor}^{n-1} i^2 \right)$. Our algorithm computes the tight approximation $-\frac{\lceil n/2 \rceil}{6 \cdot n} + \frac{\lceil n/2 \rceil^2}{2 \cdot n} - \frac{\lceil n/2 \rceil^3}{3 \cdot n} - \frac{\lfloor n/2 \rfloor}{6 \cdot n} + \frac{\lfloor n/2 \rfloor^2}{2 \cdot n} - \frac{\lfloor n/2 \rfloor^3}{3 \cdot n} + \frac{1}{3} - n + \frac{2}{3} \cdot n^2$. □

Dealing with floors and ceilings. When we have a Muniofrm distribution or FSDPD that involves $\lceil n/2 \rceil$ or $\lfloor n/2 \rfloor$, the moment approximation result may involve terms with floors and ceilings, e.g. $\ln(\lceil n/2 \rceil)$. Our algorithm performs a case analysis over the parity of n to remove ceilings and floors and then approximates their powers with pseudo-polynomials (see Appendix F). For example, for odd n, we have $\ln(\lceil n/2 \rceil) = \ln(n/2 + 1/2)$, which is then approximated as $\ln(n) - \ln 2 \leq \ln(n/2 + 1/2) \leq \ln(n) - \ln 2 + \frac{1}{n}$.

Example 4.10. In Example 4.9, since $h(n)$ observes a Muniform distribution, the approximation of $\mathbb{E}[h(n)^2]$ involves floors $\lfloor n/2 \rfloor$ and ceilings $\lceil n/2 \rceil$. For even n, both terms are equal to $n/2$. For odd n, we have $\lceil n/2 \rceil = n/2 + 1/2$ and $\lfloor n/2 \rfloor = n/2 - 1/2$. Hence we get $\mathbb{E}[f(h(n))^2] = \frac{7 \cdot n^2}{12} - \frac{3 \cdot n}{4} + \frac{1}{6}$ when n is even, and $\frac{7 \cdot n^2}{12} - \frac{3 \cdot n}{4} + \frac{1}{4 \cdot n} - \frac{1}{12}$ when n is odd. It follows that we have $A_2(n) = \mathbb{E}[X^2(n)] = 21 \cdot n^2 - 27 \cdot n + 6$ for even n, and $A_2(n) = 21 \cdot n^2 - 27 \cdot n + \frac{9}{n} - 3$ for odd n. □

Step A4: Non-positivity Checking. After the over-approximation of each $\mathbb{E}\left[X(n)^i\right]$ by a pseudo-polynomial expression $A_i(n)$, our algorithm obtains a strengthened version of (11) for sufficiently large n in this form:

$$\sum_{i=1}^{L} \frac{t_r(n)^i}{i!} \cdot A_i(n) + \frac{\exp(\overline{\lambda})^{L+1}}{(L+1)!} \cdot A_{L+1}(n) \leq 0.$$

The verification procedure now checks whether this strengthened inequality holds, i.e. whether the LHS is non-positive when n is sufficiently large. In short, it checks whether a given pseudo-polynomial is non-positive for all natural numbers above a given threshold. This can be decided using calculus techniques. We propose a sound and complete approach that decides this property by examining the behavior of the pseudo-polynomial when $n \to \infty$. See Appendix G for details.

Upon the final decision, if the verification procedure can confirm that the relaxed inequality holds, then the concrete value $\overline{\lambda}$ for λ fulfills (10). Otherwise, the verification procedure simply outputs "*fail*" which means that $\overline{\lambda}$ may violate (10), and the binary search at the upper-level should consider a smaller value for λ. The accuracy of our verification procedure is guaranteed by the fact that the error/deviation introduced from the original inequality (10) to the strengthened inequality tends to zero as we increase the Taylor expansion depth L and the precision of the approximation of $\mathbb{E}[f(h(n))^j]$. This concludes our algorithm for the simplified case of a single PRR.

We now show how these ideas and techniques can be generalized to any SPRR that satisfies our practical assumptions. There are two major factors to consider, namely the generalization to divide-and-conquer and multiple recurrence equations. Below we sketch the major generalization, and relegate the details to Appendix H.

Divide-and-conquer. The main difficulty of this case is that the over-approximation of the higher-moment expectations $\mathbb{E}[X(n)^i]$ may involve the approximation of expectations of non-pseudo-polynomials caused by the cross product of $\ln h_{r,1}(n)$ and $\ln h_{r,2}(n)$. We address the issue by soundly over-approximating $\ln h_{r,2}(n)$ as $\ln n$, so that the resultant expectations are still in pseudo-polynomial form.

Multiple recurrence equations. Given that the input SPRRs are assumed to be non-mutual, we can simply handle all recurrence equations in an SPRR one by one in order from eq_1 to eq_m, where eq_1 is the bottom procedure and eq_m is the main procedure. When solving for (eq_r), the algorithm already has the values for $\lambda_1, \ldots, \lambda_{r-1}$. Thus, it simply plugs them in and calculates the higher-moment expectations induced from the recursive calls to $proc_1, \ldots, proc_{r-1}$ in $Rec_r(n)$. It then solves for λ_r in the same way as for the single recurrence case.

Our ExpSyn algorithm is presented in pseudo-code form as Algorithm 1.

Algorithm 1: Algorithm ExpSyn

Input: an SPRR Δ with m equations s.t. Section 4.1, m pseudo-monomials $f_1, \ldots, f_m$.
Parameter: depth L for Taylor expansion, $M \in (0, \infty)$ for binary search, and error parameter ϵ for verification procedure

$r \leftarrow 1$.
repeat
  // Deal with equation (eq_r) in SPRR Δ.
  Step 1: Read $f_r(n)$ from the input (note that this step can be automated through Lemma 4.1).
  Step 2: Calculate the exponent $X_r(n)$ and bound it with $B_r(n)$.
  Step 3: Set up template $t_r(n) := \lambda_r / B_r(n)$ with the unknown coefficient λ_r.
  Step 4: Binary-search λ_r within $[0, M]$, during which verify whether the current choice $\overline{\lambda_r}$ satisfies (5) by over-approximating the condition (5) with depth $L+1$ Taylor expansion, rephrasing the expansion as (11) and the following steps.
  - *Step A1:* Binomial-expand each $X_r(n)^i$ in (11) into a summation of products of $-f_r(n)$, $Cost_r(n)$ and $f_r(h(n))$. Note that the functions $f_1, \ldots, f_{r-1}$ have already been solved.
  - *Step A2:* Compute each $\mathbb{E}[Cost_r(n)^j]$ from the binomial expansion as a pseudo-polynomial.
  - *Step A3:* If $S_r = 1$, then we will approximate each $\mathbb{E}[f_r(h(n))^j]$ from the binomial expansion with error parameter ϵ by the algorithm in Appendix E, otherwise if $S_r = 2$, we will approximate each $\mathbb{E}[f_r(h_1(n))^j \cdot f_r(h_2(n))^k]$ instead.
  - *Step A4:* Check the non-positivity of the whole pseudo-polynomial obtained from steps A1–A3 via the algorithm in Appendix G.

  $r \leftarrow r + 1$.
until $r > m$
return the bound by Theorem 3.7

4.3 PolySyn: An Algorithm for Deriving Polynomially-Decreasing Bounds

Algorithm 2: Algorithm PolySyn

Input:

- An SPRR Δ with m equations.
- The degree k that specifies the polynomial degree in tail bounds.
- $2 \cdot m \ldots (k+1)$ numbers $u_{r,i}$ and $v_{r,i}$ that specifies templates.

Parameter: Error parameter in binary search ϵ, and $M > 0$ for the range of binary search.

for $r \leftarrow 1$ **to** m **do**
  // Deal with SPRR equation (eq_r)
  for $i \leftarrow 0$ **to** k **do**
    // Solving the i-th function $f_{r,i}(n)$ that over-approximates $\mathbb{E}[T_r(n)^i]$
    Step 1: Setting up template $f_{r,i}(n) := \lambda_{r,i} \cdot n^{u_{r,i}} \cdot \ln n^{v_{r,i}}$ with unknown coefficient $\lambda_{r,i}$
    Step 2: Binary Searching $\lambda_{r,i}$ within range $[0, M]$ and with error parameter ϵ and verifiying if (6) and (7) as follows.
    - Compute $\mathbb{E}[Cost_r(n)^j]$ as a pseudo-polynomial.
    - Check the non-positivity of the whole pseduo-polynomial via the algorithm in Appendix G.
  end for
end for
return The final bound calculated by Theorem 3.8

Based on Theorem 3.8, we develop our algorithm PolySyn for polynomially-decreasing bounds. The pseudo-code of PolySyn is presented in Algorithm 2. Since the algorithm follows the same paradigm as in ExpSyn, we only focus on their differences.

Recall that to apply Theorem 3.8, we need to synthesize $m \cdot (k+1)$ functions $f_{r,i}(n)$, one for each equation (eq_r) in the SPRR and each degree $0 \le i \le k$, so that the conditions (6) and (7) are fulfilled. The main differences with ExpSyn are in the templates, the order to address the unknown coefficients, and the details of the binary search. Below we illsturate the difference in detail, we fix a number k for the maximum degree of monomials.

Step 1: Setting up Templates. The templates correspond to Theorem 3.8 and are different from those in ExpSyn. The algorithm PolySyn sets up a pseudo-monomial template $f_{r,i}(n) := \lambda_{r,i} \cdot n^{u_{r,i}} \cdot \ln^{v_{r,i}} n$ for every $1 \le r \le m$ and $0 \le i \le k$, where $u_{r,i}$ and $v_{r,i}$ are given from the input and $\lambda_{r,i}$ is the unknown coefficient. Assuming that $u_{r,i}$ and $v_{r,i}$ are given does not affect the generality of our algorithm since they are small in practice and we can simply enumerate all possible values from a given range.

Unknown Coefficients. Using the non-mutuality assumption, our algorithm solves $\lambda_{r,i}$'s one by one. The solving order is lexicographic on (r,i), i.e. smaller r first, and then smaller i when the r's are equal. This order must be followed since for every $1 \le r \le m$ and $0 \le i \le k$, the term $Binom_{r,i}$ that appears in the RHS of condition (6) only involves $\lambda_{r',i'}$'s such that $1 \le r' \le r$ and $0 \le i' \le i$. Thus, the optimal value for $\lambda_{r,i}$ only depends on the value of the $\lambda_{r',i'}$'s for $1 \le r' \le r$ and $0 \le i' \le i$.

Step 2: Binary Search. The binary search process for the value of $\lambda_{r,i}$ now tries to find the *smallest* possible value. It can be straightforwardly verified from Theorem 3.8 that a smaller value for $\lambda_{r,i}$ leads to a better tail bound. Moreover, the following lemma justifies the binary search:

Lemma 4.11 (Proof in Appendix H). *Suppose that the values of all the unknown coefficients $\lambda_{r',i'}$ for $1 \leq r' \leq r$ and $0 \leq i' \leq i-1$ are known and fixed. For any two values $\lambda_0 < \lambda_1$, if both the conditions (6) and (7) hold by setting $\lambda_{r,i} = \lambda_0$, then they still hold if one sets $\lambda_{r,i} = \lambda_1$.*

Apart from the three main differences above, there are two minor changes in our algorithm for polynomially-decreasing bounds: (i) we do not need to rely on a Taylor expansion since there is no exponential term, and (ii) we check the boundary condition (7). All other steps in our algorithm remain the same as in Algorithm ExpSyn. For example, we still use binomial expansion for the RHS of (6), we treat the probability distributions in exactly the same way, and we use the same verification procedure. See Appendix H for details.

Remark (Time Complexity)**.** Our algorithm ExpSyn runs in time polynomial in the size of the input SPRR, the input function $f(n)$ and the logarithm of the error parameter ϵ (i.e., $\log \epsilon^{-1}$), and in time pseudo-polynomial in the depth L of Taylor expansions. The latter can be observed from the fact that the binomial expansion with degree L can be done in time pseudo-polynomial in L. By similar reasons, our algorithm PolySyn runs in time polynomial in the size of the input SPRR and the template $f(n)$, and in time pseudo-polynomial in the degree k of the monomial. □

Remark (Possible Extensions)**.** Our algorithms can be extended to an even wider class of probabilistic recurrences. First, a main techique in our algorithms is the tight approximation of summations of pseudo-monomials that corresponds to expectation of probability distributions in (eq_r), e.g. $h(n)$, under the application of pseudo-monomials, i.e. $\mathbb{E}[h^i(n) \ln^j h(n)]$. Thus, our algorithms are easily extendable to probability distributions whose expectation under pseudo-monomials can be tightly approximated. If one only considers monomials, i.e. pseudo-monomials without logarithm, then only higher moments of these probabilistic distributions are required. Second, our algorithm could handle sizes passed to sub-procedures in the form of $\lfloor \frac{n}{c} \rfloor$, $\lceil \frac{n}{c} \rceil$ ($c \in \mathbb{N}, c \geq 3$) by considering the value of the size n modulo c. Note that in our algorithms, we distinguish the even/odd case of n for $c = 2$. Third, our divide-and-conquer setting requires the restriction that $h_{r,1}(n) + h_{r,2}(n) = n-1$. It is only used to efficiently approximate the cross product term $\mathbb{E}[h_{r,1}(n)^u \cdot h_{r,2}(n)^v]$ while approximating the higher-order moment $\mathbb{E}[X_r(n)^j]$. In detail, under this assumption, we could subtitute $h_{r,2}(n)$ with $n-1-h_{r,1}(n)$, and use the algorithm illustrated above. We believe that in many examples we could relax the setting to $h_{r,1}(n) + h_{r,2}(n) \leq n-1$ by taking $h_{r,1}(n) + h_{r,2}(n) = n-1$ as the worse case. Fourth, although we only allow divide-and-conquer in the case of multiple self-recursive calls, our algorithms solve this case by an over-approximation of logarithmic terms and thus can be extended to more complex cases, e.g. more than two self-recursive calls. □

5 EXPERIMENTAL RESULTS

In this section, we provide experimental results over SPRRs obtained from various classical randomized algorithms. These results demonstrate that our algorithmic approaches are capable of both obtaining much tighter tail bounds than [Karp 1994] and handling a wider family of SPRRs.

We evaluated the two algorithmic approaches of Section 4 over classical randomized algorithms such as QuickSort (Example 2.2), QuickSelect (Example 2.1), DiameterComputation [Motwani and Raghavan 1995, Chapter 9], RandomizedSearch [McConnell 2001, Chapter 9], ChannelConflictResolution [Kleinberg and Tardos 2006, Chapter 13], SeidelLP (Example 2.3) and SmallestEnclosingDisk [Welzl 1991]. We also considered a manually-crafted algorithm, SortingSelect, that sorts an array using a divide-and-conquer method for which the median pivot element is obtained through QuickSelect. Due to space constraints, benchmarks' details are provided in Appendix B.

We implemented our algorithms in C++ using the `miracl` library [Ltd. 2018] for high precision computation. For QuickSort, QuickSelect, DiameterComputation, RandomizedSearch, and

Benchmark	Task	$f(n)'s$	$B(n)$	$t(n)'s$	Our symbolic bound	Time (s)	Karp's bound
L1Diameter	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$2.5 \cdot n$	$1.5 \cdot n$	$\frac{\lambda}{n}$	$\exp(1.008 \cdot (2.5-\alpha))$	2.28	$(\frac{1}{2})^{\alpha-2}$
		$3 \cdot n$	$2 \cdot n$		$\exp(1.540 \cdot (3.0-\alpha))$	2.16	
		$3.5 \cdot n$	$2.5 \cdot n$		$\exp(1.774 \cdot (3.5-\alpha))$	2.08	
L2Diameter	$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*]$	$2.5 \cdot n \cdot \ln n$	$1.5 \cdot n \cdot \ln n$	$\frac{\lambda}{n \cdot \ln n}$	$\exp(1.008 \cdot (2.5-\alpha))$	3.09	$(\frac{1}{2})^{\alpha-2}$
		$3 \cdot n \cdot \ln n$	$2 \cdot n \cdot \ln n$		$\exp(1.338 \cdot (3.0-\alpha))$	3.10	
		$3.5 \cdot n \cdot \ln n$	$2.5 \cdot n \cdot \ln n$		$\exp(1.189 \cdot (3.5-\alpha))$	3.10	
QuickSelect	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$5.0 \cdot n$	$3 \cdot n$	$\frac{\lambda}{n}$	$\exp(1.007 \cdot (5.0-\alpha))$	1.92	$(\frac{3}{4})^{\alpha-4}$
		$6 \cdot n$	$3.67 \cdot n$		$\exp(1.364 \cdot (6.0-\alpha))$	1.98	
		$7 \cdot n$	$4.33 \cdot n$		$\exp(1.112 \cdot (7.0-\alpha))$	1.95	
SortSelect	$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^*]$	$5.0 \cdot n$	$3 \cdot n$	$\frac{\lambda_1}{n}$	$\exp(1.007 \cdot (15.0-\alpha) \cdot \ln n^*)$	1.96	-
		$15 \cdot n \cdot \ln n$	$4.20 \cdot n$				
		$6 \cdot n$	$3.67 \cdot n$	$\frac{\lambda_2}{n}$	$\exp(1.364 \cdot (18.0-\alpha) \cdot \ln n^*)$	1.98	
		$18 \cdot n \ln n$	$5.24 \cdot n$				
		$7 \cdot n$	$4.33 \cdot n$		$\exp(1.112 \cdot (21.0-\alpha) \cdot \ln n^*)$	2.14	
		$21 \cdot n \cdot \ln n$	$6.27 \cdot n$				
QuickSort	$\Pr[C_\tau \geq \alpha \cdot n^* \cdot \ln n^* + \beta \cdot n^*]$	$6 \cdot n \cdot \ln n$	$1.08 \cdot n$	$\frac{\lambda}{n}$	$\exp(1.123 \cdot ((6-\alpha) \cdot \ln n^* - \beta))$	2.54	$\exp(\frac{1-\alpha}{2})$
		$7 \cdot n \cdot \ln n$	$1.43 \cdot n$		$\exp(1.038 \cdot ((7-\alpha) \cdot \ln n^* - \beta))$	2.57	
		$8 \cdot n \cdot \ln n$	$1.78 \cdot n$		$\exp(0.944 \cdot ((8-\alpha) \cdot \ln n^* - \beta))$	2.53	
RandSearch	$\Pr[C_\tau \geq \alpha \cdot \ln n^*]$	$4.0 \cdot \ln n$	1.78	λ	$(n^*)^{0.446 \cdot (4.0-\alpha)}$	2.53	$(n^*)^{1-0.287 \cdot \alpha}$
		$4.3 \cdot \ln n$	1.98		$(n^*)^{0.441 \cdot (4.3-\alpha)}$	2.53	
		$4.5 \cdot \ln n$	2.12		$(n^*)^{0.432 \cdot (4.5-\alpha)}$	2.52	
Channel	$\Pr[C_\tau \geq \alpha \cdot n^*]$	$3 \cdot n$	2	λ	$\exp(0.887 \cdot (3.0-\alpha) \cdot n^*)$	0.49	$\exp(\frac{\alpha-e}{e})$
		$4 \cdot n$	3		$\exp(0.964 \cdot (3.0-\alpha) \cdot n^*)$	0.44	
		$5 \cdot n$	4		$\exp(0.975 \cdot (3.0-\alpha) \cdot n^*)$	0.45	

Table 1. Symbolic Exponentially-decreasing Tail Bounds. For each benchmark, we provide results based on several different choices of $f(n)$.

SortingSelect, we ran our algorithm of Section 4.2 that synthesizes exponentially-decreasing tail bounds. For SeidelLP and SmallestEnclosingDisk, we ran the algorithm of Section 4.3 that synthesizes polynomially-decreasing bounds. In our experiments, we expanded the Taylor series to a depth of at most 15, restricted the range of binary search in $[0, 20]$, and set the error parameter ϵ in verification procedure as 10^{-6}. All results were obtained on an Ubuntu 18.04 machine with an 8-Core Intel i7-7900x Processor (13.75M cache, up to 4.30 GHz) and 40 GB of RAM. Tables 1–3 illustrate our experimental results over the benchmarks.

Exponentially-decreasing tail bounds. Table 1 shows our symbolic exponential bounds, which were obtained using the ExpSyn algorithm. It also provides Karp's bounds for the same SPRRs, demonstrating that our bounds are much tighter. To further illustrate this point, we provide concrete bounds (with fixed values for the parameters α and β) in Table 2. As shown in this table, our bounds consistently beat previous results. Notably, in some cases, our bound gets infinitely tighter than Karp's as n^* grows. Additionally, Karp's method is not applicable for SortSelect because they either involve more than one recurrence equation. These examples show that our approach handles a wider class of SPRRs.

Polynomially-decreasing tail bounds. Table 3 shows our experimental results using the PolySyn algorithm to obtain polynomially-decreasing bounds. In all the benchmarks, we set $u_{r,i} = i$ and $v_{r,i} = 0$ in our templates for every $1 \leq r \leq m$ and $1 \leq i \leq k$. To the best of our knowledge, we are providing the first algorithm for synthesizing polynomially-decreasing tail bounds. As such, there are no previous methods to compare against. Moreover, previous approaches for exponential bounds do not yield any results on these benchmarks. We believe this is probably because the best possible tail bounds are polynomially decreasing.

As shown in Tables 1 and 3, our algorithms are extremely efficient in practice and can synthesize much tighter bounds within a few seconds. In our experiments, the maximum runtime was only 3.40 seconds, while the average runtime was 1.38 seconds.

6 RELATED WORKS

In this section, we compare our approach with the most related results in the literature.

Benchmark	Parameters	$f(n)$	$B(n)$	Our concrete bound	Karp's concrete bound	Ratio
L1Diameter	$\alpha = 6$	$2.5 \cdot n$	$1.5 \cdot n$	0.029	0.0625	2.13
	$\alpha = 8$	$3.5 \cdot n$	$2.5 \cdot n$	$3.41 \cdot 10^{-4}$	0.015	45.81
	$\alpha = 10$	$3 \cdot n$	$2 \cdot n$	$2.079 \cdot 10^{-5}$	0.0039	187.84
L2Diameter	$\alpha = 6$	$3.5 \cdot n$	$2.5 \cdot n$	0.051	0.0625	1.21
	$\alpha = 8$	$2.5 \cdot n$	$1.5 \cdot n$	0.0039	0.015	3.97
	$\alpha = 10$	$3 \cdot n$	$2 \cdot n$	$8.51 \cdot 10^{-5}$	0.0039	45.87
QuickSelect	$\alpha = 12$	$5 \cdot n$	$3 \cdot n$	$8.641 \cdot 10^{-4}$	0.1002	115.85
	$\alpha = 16$	$7 \cdot n$	$4.33 \cdot n$	$4.503 \cdot 10^{-5}$	0.0317	703.85
	$\alpha = 20$	$6 \cdot n$	$3.67 \cdot n$	$5.089 \cdot 10^{-9}$	0.0101	$1.984 \cdot 10^{6}$
QuickSort	$\alpha = 7, \beta = 6$	$6 \cdot n \cdot \ln n$	$1.079 \cdot n$	$\exp(-1.123 \cdot \ln n^* + 6.738)$	0.050	∞ (as $n^* \to \infty$)
	$\alpha = 8, \beta = 1$	$7 \cdot n \cdot \ln n$	$1.426 \cdot n$	$\exp(-1.038 \cdot \ln n^* + 1.038)$	0.03	
	$\alpha = 9, \beta = 3$	$8 \cdot n \cdot \ln n$	$1.772 \cdot n$	$\exp(-0.944 \cdot \ln n^* + 2.832)$	0.018	
RandSearch	$\alpha = 6.5$	$4 \cdot \ln n$	1.772	$(n^*)^{-1.113}$	$(n^*)^{-0.869}$	
	$\alpha = 6.8$	$4.3 \cdot \ln n$	1.98	$(n^*)^{-1.104}$	$(n^*)^{-0.956}$	
	$\alpha = 7.0$	$4.5 \cdot \ln n$	2.11	$(n^*)^{-1.084}$	$(n^*)^{-1.013}$	
Channel	$\alpha = 5$	$3 \cdot n$	3	$\exp(-1.773 \cdot n^*)$	0.432	
	$\alpha = 6$	$4 \cdot n$	4	$\exp(-1.928 \cdot n^*)$	0.300	
	$\alpha = 7$	$5 \cdot n$	5	$\exp(-1.951 \cdot n^*)$	0.207	

Table 2. Concrete Exponentially-decreasing Tail Bounds.

Benchmark	$\mathbb{E}[T_d(n^*)]$	Task	degree k	Symbolic bound	Time(s)
SeidelLP($d = 2$)	$8 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 8 \cdot n^*]$	$k = 5$	$3.15 \cdot \alpha^{-5}$	0.12
			$k = 6$	$5.24 \cdot \alpha^{-6}$	0.21
			$k = 7$	$9.33 \cdot \alpha^{-7}$	0.22
SeidelLP($d = 3$)	$36 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 36 \cdot n^*]$	$k = 5$	$3.30 \cdot \alpha^{-5}$	0.25
			$k = 6$	$5.55 \cdot \alpha^{-6}$	0.43
			$k = 7$	$10.01 \cdot \alpha^{-7}$	0.70
Disk($d = 3$)	$10 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 10 \cdot n^*]$	$k = 3$	$1.487 \cdot \alpha^{-3}$	0.15
			$k = 4$	$2.125 \cdot \alpha^{-4}$	0.37
			$k = 5$	$3.318 \cdot \alpha^{-5}$	1.75
Disk($d = 4$)	$41 \cdot n^*$	$\Pr[C_\tau \geq \alpha \cdot 41 \cdot n^*]$	$k = 2$	$1.137 \cdot \alpha^{-2}$	0.21
			$k = 3$	$1.439 \cdot \alpha^{-3}$	0.28
			$k = 4$	$2.000 \cdot \alpha^{-4}$	3.40

Table 3. Polynomially-decreasing Tail Bounds.

Karp's cookbook. On the theoretical level, the classical "cookbook" method by Karp [Karp 1994] establishes a fixed mathematical formula (See Section 2.2 for details) for deriving exponentially-decreasing tail bounds, in which the correctness of the formula is proved through either fixed-point theory or game theory. In contrast, our approach is based on the completely different method of Markov's inequality. Moreover, we have shown in Section 3.2 that (i) our theoretical approaches can derive polynomially-decreasing tail bounds and handle probabilistic recurrences with randomized preprocessing time and multiple recurrence equations, (ii) they can derive (asymptotically) tighter tail bounds, and (iii) they are generally incomparable with Karp's method. On a technical level, our algorithmic approaches are quite complex since they need to examine the detailed structure and probability distributions in the input SPRR. However, they can nevertheless handle a wide class of probabilistic recurrences, including those of classical randomized algorithms, and automatically derive much tighter tail bounds. It is worth noting that, to the best of our knowledge, it is infeasible to directly extend Karp's method to randomized preprocessing time or multiple recurrence equations, since its correctness proofs are restricted to the fixed formulas.

Other tail-bound results. The work [Chaudhuri and Dubhashi 1997] weakens the prerequisites in [Karp 1994] and obtains bounds that are slightly worse than [Karp 1994]. This work cannot handle probabilistic recurrences with randomized preprocessing time or multiple recurrence equations. Thus, the comparison between our approach and [Karp 1994] carries over to [Chaudhuri and Dubhashi 1997]. The work [McDiarmid and Hayward 1996] performs an involved ad-hoc manual analysis to derive an asymptotically optimal tail bound for QuickSort, while our approach is more general through Markov's inequality, can be automated, and algorithmically derives a tail bound for the QuickSort that is only slightly worse than [McDiarmid and Hayward 1996] ($\exp(-\Theta(\ln n^* \cdot \ln\ln n^*))$ vs. $\exp(-\Theta(\ln n^*))$). A recent result [Tassarotti 2017] introduces a new approach for solving work-and-span runtime of parallel randomized algorithms. This method requires manual derivation and can neither handle randomized preprocessing time nor systems of probabilistic recurrences. Moreover, our automated approach can derive much tighter tail bounds for QuickSort against this method: our automated approach generates the tail bound $\Pr[T(n) \geq 11 \cdot n^* \ln n^* + 12 \cdot n^*] \leq \exp(-5.615 \ln n^* - 13.476)$, while this method outputs $\Pr[T(n) \geq 11 \cdot n^* \ln n^* + 12 \cdot n^*] \leq \exp(-0.24 \cdot \ln n^* - 1.47)$.

Probabilistic programs. There are also several related results in probabilistic programs. These results are either based on martingale concentration inequalities [Chakarov and Sankaranarayanan 2013; Chatterjee et al. 2018, 2017c; Huang et al. 2018], Markov's inequality (for polynomially-decreasing tail bounds) [Chatterjee and Fu 2017; Kura et al. 2019; Wang et al. 2021a], or fixed-point theory [Wang et al. 2021b]. Compared with these results, our approaches target SPRRs instead of programs. Moreover, our SPRRs can have complicated recursion and handle various probability distributions, which are both beyond the reach of the results on probabilistic programs. Specifically, none of these results consider recursion except for [Wang et al. 2021a]. The recursion handled in [Wang et al. 2021a] is much more limited as it does not consider complicated samplings (e.g. uniform distribution over the passed sizes), and is hence unable to handle divide-and-conquer recurrences. Moreover, it can only obtain polynomially-decreasing tail bounds. Finally, we follow different algorithmic techniques that synthesize tail bounds in exponential and polynomial form by Taylor's expansion and tight approximation of expectation of moments with logarithmic terms. Thus, our techniques are also novel and differ from the related results on probabilistic programs.

Here we provide a detailed comparison with a recent work [Wang et al. 2021b] that also aims at deriving tight tail bounds over probabilistic programs. This work proceeds by either applying convex programming to the restricted case of *linear* exponents, or repulsing ranking supermartingales (RepRSMs) [Chatterjee et al. 2017c]. In contrast, our approach is based on Markov's inequality, handles nonlinear exponents, e.g. exponents with logarithmic terms, and addresses the detailed distributions by dedicated algorithmic techniques from Taylor's expansion and approximation of pseudo-polynomials. Hence, our approach and [Wang et al. 2021b] are quite different. Additionally, [Wang et al. 2021b] cannot handle our benchmarks because the exponents are non-linear and there is no suitable RepRSM for probabilistic recurrences.

7 CONCLUSION AND FUTURE WORK

In this work, we introduced novel theoretical and algorithmic approaches that derive exponentially and polynomially-decreasing tail bounds for probabilistic recurrences via Markov's inequality. A future direction would be to improve our algorithmic approaches to derive tighter tail bounds. Another future direction would be to explore the integration of our approaches and the classical cookbook method by Karp. A third direction would be to automate the process of obtaining SPRRs from probabilistic programs and randomized algorithms, similar to the COSTA project [Albert et al. 2009, 2008, 2007] that translates Java bytecode into non-probabilistic recurrences.

REFERENCES

Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. 2018. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *PACMPL* 2, POPL (2018), 34:1–34:32.

Mohamad A. Akra and Louay Bazzi. 1998. On the Solution of Linear Recurrence Equations. *Comput. Optim. Appl.* 10, 2 (1998), 195–210.

Elvira Albert, Puri Arenas, Samir Genaim, Miguel Gómez-Zamalloa, German Puebla, Diana V. Ramírez-Deantes, Guillermo Román-Díez, and Damiano Zanardini. 2009. Termination and Cost Analysis with COSTA and its User Interfaces. *Electr. Notes Theor. Comput. Sci.* 258, 1 (2009), 109–121.

Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. 2008. Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis. In *SAS 2008*. 221–237.

Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. 2007. Cost Analysis of Java Bytecode. In *ESOP 2007*. 157–172.

Roberto Bagnara, Andrea Pescetti, Alessandro Zaccagnini, and Enea Zaffanella. 2005. PURRS: Towards Computer Algebra Support for Fully Automatic Worst-Case Complexity Analysis. *CoRR* abs/cs/0512056 (2005). arXiv:cs/0512056 http://arxiv.org/abs/cs/0512056

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

Louay Bazzi and Sanjoy K. Mitter. 2003. The Solution of Linear Probabilistic Recurrence Relations. *Algorithmica* 36, 1 (2003), 41–57.

Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Springer.

P Billingsley. 1995. *Probability and Measure* (3rd ed.). Wiley.

Jason Breck, John Cyphert, Zachary Kincaid, and Thomas W. Reps. 2020. Templates and recurrences: better together. In *PLDI*, Alastair F. Donaldson and Emina Torlak (Eds.). ACM, 688–702.

Aleksandar Chakarov and Sriram Sankaranarayanan. 2013. Probabilistic Program Analysis with Martingales. In *CAV*. 511–526.

Krishnendu Chatterjee and Hongfei Fu. 2017. Termination of Nondeterministic Recursive Probabilistic Programs. *CoRR* abs/1701.02944 (2017).

Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. 2017a. Non-polynomial Worst-Case Analysis of Recursive Programs. In *CAV 2017*. 41–63.

Krishnendu Chatterjee, Hongfei Fu, and Aniket Murhekar. 2017b. Automated Recurrence Analysis for Almost-Linear Expected-Runtime Bounds. In *CAV*. 118–139.

Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. 2018. Algorithmic Analysis of Qualitative and Quantitative Termination Problems for Affine Probabilistic Programs. *TOPLAS* 40, 2 (2018), 7:1–7:45.

Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. 2017c. Stochastic invariants for probabilistic termination. In *POPL 2017*. 145–160.

Shiva Chaudhuri and Devdatt P. Dubhashi. 1997. Probabilistic Recurrence Relations Revisited. *Theoretical Computer Science* 181, 1 (1997), 45–56.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms (3. ed.)*. MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

Devdatt P. Dubhashi and Alessandro Panconesi. 2009. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press.

Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *FMCAD*, Roope Kaivola and Thomas Wahl (Eds.). IEEE, 57–64.

Philippe Flajolet, Bruno Salvy, and Paul Zimmermann. 1991. Automatic Average-Case Analysis of Algorithm. *Theor. Comput. Sci.* 79, 1 (1991), 37–109.

Bernd Grobauer. 2001. Cost Recurrences for DML Programs. In *ICFP*, Benjamin C. Pierce (Ed.). ACM, 253–264.

C. A. R. Hoare. 1961a. Algorithm 64: Quicksort. *Commun. ACM* 4, 7 (1961), 321.

C. A. R. Hoare. 1961b. Algorithm 65: find. *Commun. ACM* 4, 7 (1961), 321–322.

Jan Hoffmann, Ankush Das, and Shu-Chun Weng. 2017. Towards automatic resource bound analysis for OCaml. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 359–373. https://doi.org/10.1145/3009837.3009842

Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. 2018. New Approaches for Almost-Sure Termination of Probabilistic Programs. In *APLAS*. 181–201.

Richard M. Karp. 1994. Probabilistic Recurrence Relations. *Journal of the ACM* 41, 6 (1994), 1136–1150.

Marek Karpinski and Wolf Zimmermann. 1991. *Probabilistic Recurrence Relations for Parallel Divide-and-Conquer Algorithms*. Technical Report TR-91-067. https://www.icsi.berkeley.edu/ftp/global/pub/techreports/1991/tr-91-067.pdf

Zachary Kincaid, Jason Breck, Ashkan Forouhi Boroujeni, and Thomas W. Reps. 2017. Compositional recurrence analysis revisited. In *PLDI*, Albert Cohen and Martin T. Vechev (Eds.). ACM, 248–262.

Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. 2018. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.* 2, POPL (2018), 54:1–54:33.

Jon M. Kleinberg and Éva Tardos. 2006. *Algorithm design*. Addison-Wesley.

Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. 2019. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In *TACAS (LNCS, Vol. 11428)*, Tomás Vojnar and Lijun Zhang (Eds.). Springer, 135–153.

MIRACL Ltd. 2018. https://github.com/miracl/MIRACL.

Hosam Mahmoud. 2008. *Pólya urn models*. CRC press.

Jeffrey J. McConnell (Ed.). 2001. *The Analysis of Algorithms: An Active Learning Approach*. Jones & Bartlett Learning.

Colin McDiarmid and Ryan Hayward. 1996. Large Deviations for Quicksort. *Journal of Algorithms* 21, 3 (1996), 476–507.

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

Franco P. Preparata and David E. Muller. 1979. Finding the Intersection of n Half-Spaces in Time O(n log n). *Theor. Comput. Sci.* 8 (1979), 45–55. https://doi.org/10.1016/0304-3975(79)90055-0

Raimund Seidel. 1991. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry* 6, 3 (1991), 423–434.

Joseph Tassarotti. 2017. Probabilistic Recurrence Relations for Work and Span of Parallel Algorithms. *CoRR* abs/1704.02061 (2017). http://arxiv.org/abs/1704.02061

Joseph Tassarotti and Robert Harper. 2018. Verified Tail Bounds for Randomized Programs. In *ITP*. 560–578.

Di Wang, Jan Hoffmann, and Thomas W. Reps. 2021a. Central moment analysis for cost accumulators in probabilistic programs. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 559–573. https://doi.org/10.1145/3453483.3454062

Jinyi Wang, Yican Sun, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. 2021b. Quantitative analysis of assertion violations in probabilistic programs. In *PLDI*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 1171–1186.

Emo Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science, Proceedings [on occasion of H. Maurer's 50th birthday] (LNCS, Vol. 555)*, Hermann A. Maurer (Ed.). Springer, 359–370.

David Williams. 1991. *Probability with Martingales*. Cambridge university press.

Paul Zimmermann and Wolf Zimmermann. 1989. *The automatic complexity analysis of divide-and-conquer algorithms*. Technical Report. INRIA.