Algorithmica (2017) 79:484–508
DOI 10.1007/s00453-016-0205-0

Efficient Sampling Methods for Discrete Distributions

Karl Bringmann[1] · **Konstantinos Panagiotou**[2]

Received: 29 July 2014 / Accepted: 20 August 2016 / Published online: 29 August 2016

Abstract We study the fundamental problem of the exact and efficient generation of random values from a finite and discrete probability distribution. Suppose that we are given n distinct events with associated probabilities $p_1, \ldots, p_n$. First, we consider the problem of sampling from the distribution where the i-th event has probability proportional to p_i. Second, we study the problem of sampling a subset which includes the i-th event independently with probability p_i. For both problems we present on two different classes of inputs—sorted and general probabilities—efficient data structures consisting of a preprocessing and a query algorithm. Varying the allotted preprocessing time yields a trade-off between preprocessing and query time, which we prove to be asymptotically optimal everywhere.

Keywords Sampling algorithm · Subset sampling · Distribution · Proportional sampling · Data structures

1 Introduction

Generating random variables from finite and discrete distributions has long been an important building block in many applications. For example, in computer simulations usually a huge number of random decisions based on prespecified or dynamically changing distributions is made. In this work we consider two fundamental compu-

A preliminary version of this paper with worse upper and lower bounds appeared at ICALP'12.

✉ Karl Bringmann
kbringma@mpi-inf.mpg.de

[1] Max Planck Institute for Informatics, Saarbrücken, Germany

[2] University of Munich, Munich, Germany

tational problems, namely sampling *from a distribution* and sampling *independent events*. We consider these problems on general probabilities as well as restricted to sorted probabilities. The latter case is motivated by the fact that many natural distributions, such as the geometric or binomial distribution, are unimodal, i.e., they change monotonicity at most once. After splitting up such a distribution at its only extremum, we obtain two sorted sequences of probabilities, see Sect. 5 for a thorough discussion. As we will see, there is a rich interplay in designing efficient algorithms that solve these different problem variants.

We present our results on the classical Real RAM model of computation [1,13]. In particular, we will assume that the following operations take constant time: (1) accessing the content of any memory cell, (2) generating a uniformly distributed real number in the interval [0, 1], and (3) performing basic arithmetical operations involving real numbers like addition, multiplication, division, comparison, truncation, and evaluating any fundamental function like exp and log. We argue in Sect. 5 that our algorithms can also be adapted to work on the Word RAM model of computation.

1.1 Proportional Sampling

We first focus on the classic problem of sampling from a given distribution. Given $\mathbf{p} = (p_1, \ldots, p_n) \in \mathbb{R}^n_{\geq 0}$, we define a random variable $Y = Y_{\mathbf{p}}$ that takes values in[1] $[n]$ such that $\Pr[Y = i] = p_i/\mu$, where $\mu = \sum_{i=1}^{n} p_i$ is assumed to be positive. Note that if $\mu = 1$ then $\mathbf{p}$ is indeed a probability distribution, otherwise we need to normalize first. We concern ourselves with the problem of sampling Y. We study this problem on two different classes of input sequences, sorted and general (i.e., not necessarily sorted) sequences; depending on the class under consideration we call the problem SortedProportionalSampling or UnsortedProportionalSampling.

A *single-sample algorithm* for SortedProportionalSampling or UnsortedProportionalSampling gets input $\mathbf{p}$ and outputs a number $s \in [n]$ that has the same distribution as Y. When we speak of "input $\mathbf{p}$" we mean that the algorithm gets to know n and can access every p_i in constant time. This can be achieved by storing all p_i's in an array, but also, e.g., by having access to an algorithm computing any p_i in constant time. In particular, the algorithm does not know the number of i's with $p_i = 0$. Moreover, the input format is not sparse. For this problem we prove the following result.

Theorem 1.1 *There is a single-sample algorithm for* SortedProportionalSampling *with expected time* $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ *and for* UnsortedProportionalSampling *with expected time* $\mathcal{O}(n)$. *Both bounds are asymptotically tight.*

We remark that all our lower bounds only hold for algorithms that work for all n and all (sorted) sequences $p_1, \ldots, p_n$. They are worst-case bounds over the input sequence $\mathbf{p}$ and asymptotic in n. For particular instances $\mathbf{p}$ there can be faster algorithms. To avoid any confusion, note that we mean worst-case bounds whenever we

[1] Throughout the paper, we abbreviate $[n] = \{1, \ldots, n\}$.

speak of *(running) time* and expected bounds whenever we speak of *expected (running) time*.

To obtain faster sampling times, we consider *sampling data structures* that support ProportionalSampling as a *query*. We view building the data structure as *preprocessing* of the input. More precisely, in this preprocessing-query variant we consider the interplay of two algorithms. First, the *preprocessing algorithm* P gets $\mathbf{p}$ as input and computes some auxiliary data $D = D(\mathbf{p})$. Second, the *query algorithm* Q gets input $\mathbf{p}$ and D, and samples Y, i.e., for any $s \in [n]$ we have $\Pr[Q(\mathbf{p}, D) = s] = \Pr[Y = s]$. For $\Pr[Q(\mathbf{p}, D) = s]$ the probability goes only over the random choices of Q, so that, after running the preprocessing once, running the query algorithm multiple times generates multiple independent samples. In this setting we prove the following tight result.

Theorem 1.2 *For any* $2 \leq \beta \leq \mathcal{O}(\frac{\log n}{\log\log n})$, SortedProportionalSampling *can be solved in preprocessing time* $\mathcal{O}(\log_\beta n)$ *and expected query time* $\mathcal{O}(\beta)$. *This is optimal, as there is a constant* $\varepsilon > 0$ *such that for all* $2 \leq \beta \leq \mathcal{O}(\frac{\log n}{\log\log n})$ SortedProportionalSampling *has no data structure with preprocessing time* $\varepsilon \log_\beta(n)$ *and expected query time* $\varepsilon\,\beta$.

Note that if we can afford a preprocessing time of $\mathcal{O}(\log n)$ then the query time is already $\mathcal{O}(1)$, which is optimal. Thus, larger preprocessing times cannot yield better query times. Moreover, for $\beta = \Theta(\frac{\log n}{\log\log n})$ the preprocessing time is equal to the query time. Thus, we may skip the preprocessing phase and run both the preprocessing and query algorithm for every sample. We obtain a single-sample algorithm with runtime $\mathcal{O}(\frac{\log n}{\log\log n})$. This shows that $\beta \gg \frac{\log n}{\log\log n}$ makes no sense and explains why we allow preprocessing time $\mathcal{O}(\log_\beta n)$ with $2 \leq \beta \leq \mathcal{O}(\frac{\log n}{\log\log n})$. Varying β yields a trade-off between preprocessing and query time; if one wants to have a large number of samples, one should set $\beta = 2$ to minimize query time, while a large β yields superior runtimes if one wants only a small number of samples. Note that we prove a matching lower bound for this trade-off for all β.

For general input sequences, ProportionalSampling can be solved by the technique known as *pairing* or *aliasing* [8,17]; see also Mihai Pătraşcu's blog [14] for an excellent exposition. Basically, we use $\mathcal{O}(n)$ preprocessing to distribute the probabilities of all elements over n urns such that any urn contains exactly $1/n$ probability mass, stemming from at most two elements. For querying we first choose an urn uniformly at random. Then we choose one of the two included elements randomly according to their probability mass in the urn, resulting in an $\mathcal{O}(1)$ (worst-case) query time. This result is not new, but will be used in the proofs of Theorem 1.5 and Theorem 1.6 below, so we include it for completeness.

Theorem 1.3 UnsortedProportionalSampling *can be solved in preprocessing time* $\mathcal{O}(n)$ *and query time* $\mathcal{O}(1)$. *This is optimal, as there is a constant* $\varepsilon > 0$ *such that* UnsortedProportionalSampling *has no data structure with preprocessing time* εn *and expected query time* εn.

Note that any data structure with preprocessing time t_p and query time t_q can be transformed into a single-sample algorithm with expected time $t_p + t_q$, so the single-

sample variant of the problem is also solved by the preprocessing-query variant. This argument proves that Theorem 1.1 follows from Theorems 1.2 and 1.3.

Related work The fundamental problem of the exact and efficient generation of random values from discrete and continuous distributions has been studied extensively in the literature. The seminal work [9] examines the power of several restricted devices, like finite-state machines; the articles [6,18] provide a further refined treatment of the topic. However, their results are not directly comparable to ours, since on the one hand they do not make any assumption on the sequence of probabilities and use unbiased coin flips as the only source of randomness, but on the other hand they cannot guarantee efficient precomputation on general sequences. Furthermore, [7] and [10] provided algorithms for a dynamic version of UNSORTEDPROPORTIONALSAMPLING, where the probabilities may change over time. In particular, under certain mild conditions their results guarantee the same bounds as in Theorem 1.3. Finally, there is a solution to UNSORTEDPROPORTIONALSAMPLING [3] that can be implemented on a WordRAM (i.e., the p_i's are each represented by w bits, and the usual arithmetic operations on w-bit integers take constant time) that improves upon Walker's technique and has optimal space and time requirements.

1.2 Subset Sampling

In the previous section we considered the problem of sampling from a distribution. In this section we give an algorithm to randomly pick a subset S of $\{1, \ldots, n\}$, where the values $p_i = \Pr[i \in S]$ are given as an input, and the events "$i \in S$" are independent. In other words, we are given $\mathbf{p} = (p_1, \ldots, p_n)$ as input and we want to sample a random variable $X \subseteq [n]$ with

$$\Pr[X = S] = \left(\prod_{i \in S} p_i\right) \cdot \left(\prod_{i \in [n] \setminus S} (1 - p_i)\right).$$

For shortcut we write $\mu = \mu_{\mathbf{p}} = \sum_{i=1}^{n} p_i = \mathbb{E}[X]$. We call the problem of sampling X SORTEDSUBSETSAMPLING or UNSORTEDSUBSETSAMPLING, if we consider it on sorted or general input sequences, respectively.

The motivation for this problems comes from sampling certain random graphs. Consider for instance the Chung-Lu random graph model [4]: We are given weights $w_1 \geq \cdots \geq w_n$ and sample a graph on vertex set $[n]$ where the edge $\{i, j\}$ is independently present with probability $\min\{1, \frac{w_i w_j}{\sum_k w_k}\}$. Note that for any fixed vertex i, the edge probabilities to vertices $j > i$ are descendingly sorted. Thus, sampling the set of neighbors of vertex i is an instance of SORTEDSUBSETSAMPLING. Solving these instances for all vertices i yields a Chung-Lu random graph, and our algorithms from this paper do this in total time $\mathcal{O}(n \log n + m)$, where m is the expected number of edges. This does not match the optimal $\mathcal{O}(n+m)$ [11], because we ignore the structure connecting the different arising instances. However, it serves as a motivating example.

As previously, we consider two variations of SUBSETSAMPLING. In the *single-sample* variant we are given $\mathbf{p}$ and we want to compute an output that has the same

distribution as X. Moreover, in the *preprocessing-query* variant we have a precomputation algorithm that is given $\mathbf{p}$ and computes some auxiliary data D, and a query algorithm that is given $\mathbf{p}$ and D and has an output with the same distribution as X; where the results of multiple calls to the query algorithm are independent.

Any query algorithm cannot run faster than $\mathcal{O}(1+\mu)$, as its expected output size is μ and any algorithm requires a running time of $\Omega(1)$. Whether this query time is achievable depends on μ and the allotted preprocessing time, as our results below make precise. Note that the single-sample variant of UNSORTEDSUBSETSAMPLING can be solved trivially in time $\mathcal{O}(n)$; we just toss a biased coin for every p_i. This algorithm is optimal, as shown by the following tight result.

Theorem 1.4 *There is a single-sample algorithm for* SORTEDSUBSETSAMPLING *with expected time*

$$t(n,\mu) = \begin{cases} \mathcal{O}(\mu), & \text{if } \mu \geq \frac{1}{2}\log n, \\ \mathcal{O}\left(1 + \frac{\log n}{\log(\frac{\log n}{\mu})}\right), & \text{otherwise}, \end{cases}$$

and for UNSORTEDPROPORTIONALSAMPLING *with expected time* $\mathcal{O}(n)$. *Both bounds are asymptotically tight for any fixed* $\mu = \mu(n)$.

Let us discuss what we mean by "asymptotically tight for any fixed $\mu = \mu(n)$". Fix any $\mu = \mu(n)$. Consider any single-sample algorithm for SORTEDSUBSETSAMPLING that, given any $\mathbf{p}$ (not necessarily with $\mu_{\mathbf{p}} = \mu$), correctly samples from the desired distribution. Then there exists an input $\mathbf{p}$ with $\mu_{\mathbf{p}} = \mu$ such that the expected time of the algorithm on input $\mathbf{p}$ is $\Omega(t(n,\mu))$, where $t(n,\mu)$ is defined in Theorem 1.4. This holds even if we allow the algorithm to have a very large runtime for all instances with $\mu_{\mathbf{p}} \neq \mu$. In particular, our runtime bound is not only tight for one infinite family of input $\mathbf{p}$ (realizing a particular function $\mu(n)$), but for every $\mu(n)$ we construct a hard family of inputs. A similar discussion applies to Theorems 1.5 and 1.6 below.

As for PROPORTIONALSAMPLING, the single-sample result Theorem 1.4 follows from our results on the preprocessing-query variant below.

Theorem 1.5 *For any* $2 \leq \beta < n$, SORTEDSUBSETSAMPLING *can be solved in preprocessing time* $\mathcal{O}(\log_\beta n)$ *and expected query time* $\mathcal{O}(t_q^\beta(n,\mu))$, *where*

$$t_q^\beta(n,\mu) = \begin{cases} \mu, & \text{if } \mu \geq \frac{1}{2}\log n, \\ 1+\beta\mu, & \text{if } \mu < \frac{1}{\beta}\log_\beta n, \\ \frac{\log n}{\log(\frac{\log n}{\mu})}, & \text{otherwise}. \end{cases}$$

In particular, the query time is always at most $\mathcal{O}(1+\beta\mu)$. *This is optimal, as there is a constant* $\varepsilon > 0$ *such that for all* $2 \leq \beta < n$ SORTEDSUBSETSAMPLING *has no data structure with preprocessing time* $\varepsilon \log_\beta n$ *and expected query time* $\varepsilon\, t_q^\beta(n,\mu)$ *for any fixed* $\mu = \mu(n)$.

Observe that setting $\beta = 2$ in the above result yields a preprocessing time of $\mathcal{O}(\log n)$ and an (optimal) expected query time of $\mathcal{O}(1 + \mu)$.

The next result addresses the case of general, i.e., not necessarily sorted, probabilities.

Theorem 1.6 UNSORTEDSUBSETSAMPLING *can be solved in preprocessing time* $\mathcal{O}(n)$ *and expected query time* $\mathcal{O}(1 + \mu)$. *This is optimal, as there is a constant* $\varepsilon > 0$ *such that* UNSORTEDSUBSETSAMPLING *has no data structure with preprocessing time* εn *and expected query time* εn *for any fixed* $\mu = \mu(n)$.

Both positive results in the previous theorems highly depend on each other. In particular, as is demonstrated in Sect. 2.2, we prove them by repeatedly reducing the instance size n and switching from the one problem variant to the other.

We also present a relation between PROPORTIONALSAMPLING and SUBSETSAMPLING that suggests that the classic problem PROPORTIONALSAMPLING is the easier of the two problems (or can be seen as a special case of SUBSETSAMPLING). Specifically, we present a reduction that allows one to infer the upper bounds for PROPORTIONALSAMPLING (Theorems 1.2 and 1.3) from the upper bounds for SUBSETSAMPLING (Theorems 1.5 and 1.6), see Sect. 4 for details.

Related work A classic algorithm solves SUBSETSAMPLING for $p_1 = \ldots = p_n = p$ in the optimal expected time $\mathcal{O}(1 + \mu)$, see, e.g., the monographs [5] and [8], where also many other cases are discussed. Indeed, observe that the index i_1 of the first sampled element is geometrically distributed, i.e., $\Pr[i_1 = i] = (1 - p)^{i-1} p$. Such a random value can be generated by setting $i_1 = \lfloor \frac{\log \text{rand}()}{\log(1-p)} \rfloor$. Moreover, after having sampled the index of the first element, we iterate the process starting at $i_1 + 1$ to sample the second element, and so on, until we arrive for the first time at an index $i_k > n$. In [16] the "orthogonal" problem is considered, where we want to uniformly sample a fixed number of elements from a stream of objects. The problem of UNSORTEDSUBSETSAMPLING was considered also in [15], where algorithms with linear preprocessing time and suboptimal query time $\mathcal{O}(\log n + \mu)$ were designed. Our results improve upon this running time, and provide matching lower bounds.

1.3 Notation and Organization

In the remainder, we will write $\ln x$ for the natural logarithm of x, $\log_t x = \ln x / \ln t$, and $\log x = \log_2 x$. Finally, we will write rand() for a uniform random number in $[0, 1]$.

The rest of the paper is structured as follows. In Sect. 2 we present our new algorithms, proving (the upper bounds of) Theorem 1.2 in Sect. 2.1 and Theorems 1.5 and 1.6 in Sect. 2.2. In Sect. 3 we present the lower bounds, proving (the lower bounds of) Theorems 1.3 and 1.6 in Sect. 3.1, Theorem 1.2 in Sect. 3.2, and Theorem 1.5 in Sect. 3.3. We present our reduction from PROPORTIONALSAMPLING to SUBSETSAMPLING in Sect. 4. We discuss relaxations to our input and machine model and possible extensions in Sect. 5.

2 Upper Bounds

2.1 A Simple Algorithm for Sorted Proportional Sampling

In this section, we prove the upper bound of Theorem 1.2 by presenting an algorithm for SortedProportionalSampling with $\mathcal{O}(\beta)$ expected query time after $\mathcal{O}(\log_\beta n)$ preprocessing, where $2 \le \beta \le \mathcal{O}(\frac{\log n}{\log\log n})$ is a parameter. We remark that our algorithm also works for $\beta \gg \frac{\log n}{\log\log n}$, but is not meaningful in this case, because then the preprocessing time is less than the query time.

Let $p_1, \ldots, p_n$ be an input sequence to SortedProportionalSampling. Consider the blocks $B_k := \{i \in [n] \mid \beta^k \le i < \beta^{k+1}\}$ with $0 \le k \le L := \lfloor \log_\beta n \rfloor$. Note that $B_0, \ldots, B_L$ partition $[n]$. For $i \in B_k$ we set $\overline{p}_i := p_{\beta^k}$, which is an upper bound for p_i. Let $\mu := \sum_i p_i$ and $\overline{\mu} := \sum_i \overline{p}_i$. We also set for $0 \le k \le L$

$$q_k := \sum_{i \in B_k} \overline{p}_i = |B_k| \cdot p_{\beta^k} = \left(\min(\beta^{k+1}, n+1) - \beta^k\right) \cdot p_{\beta^k}.$$

For preprocessing, we run the preprocessing of UnsortedProportionalSampling on $q_0, \ldots, q_L$. This takes time $\mathcal{O}(L) = \mathcal{O}(\log_\beta n)$ using Theorem 1.3, since q_k can be evaluated in constant time.

Our query algorithm consists of two steps. First, we sample an index i with distribution $\overline{p}_1, \ldots, \overline{p}_n$. To this end, we sample a block B_k proportional to the distribution $q_0, \ldots, q_L$ and then sample an index $i \in B_k$ uniformly at random. Second, with probability $1 - p_i/\overline{p}_i$ we reject i and repeat the whole process. Otherwise we return i. This culminates into Algorithm 1.

Algorithm 1 SortedProportionalSampling

Input: $p_1 \ge \ldots \ge p_n \ge 0$ and parameter $2 \le \beta \le \mathcal{O}(\frac{\log n}{\log\log n})$

Preprocessing:
- $L := \lfloor \log_\beta n \rfloor$
- $q_k := (\min\{\beta^{k+1}, n+1\} - \beta^k) \cdot p_{\beta^k}$
- Run preprocessing of UnsortedProportionalSampling$(q_0, \ldots, q_L)$

Querying:
- **Repeat**
  - $k :=$ UnsortedProportionalSampling$(q_0, \ldots, q_L)$
  - pick i uniformly at random in $\{\beta^k, \ldots, \min\{\beta^{k+1} - 1, n\}\}$
- **Exit loop with probability** p_i / p_{β^k}
- **Return** i

Note that we pick index $i \in B_k$ with probability proportional to $\overline{p}_i$ and do not reject it with probability $p_i/\overline{p}_i$. Thus, the probability of returning a particular index i is proportional to $\overline{p}_i \cdot p_i/\overline{p}_i = p_i$ and we obtained an exact sampling algorithm. Moreover, in any iteration of the loop the probability r of not rejecting, i.e., of leaving the loop, is

$$r = \frac{1}{\overline{\mu}} \sum_{i=1}^{n} \overline{p}_i \cdot p_i / \overline{p}_i.$$

In this equation, note the first step of sampling with respect to $\overline{p}_1, \ldots, \overline{p}_n$ ($\frac{1}{\overline{\mu}} \sum_{i=1}^{n} \overline{p}_i$) and the second step of rejection ($p_i/\overline{p}_i$). Clearly, this simplifies to $r = \mu/\overline{\mu}$. The following lemma shows that $\overline{\mu} \leq \beta \cdot \mu$, implying $r \geq 1/\beta$. Hence, the expected number of iterations of the loop is $\mathcal{O}(\beta)$, and in total querying takes expected time $\mathcal{O}(\beta)$.

Lemma 2.1 *We have* $\mu \leq \overline{\mu} \leq \beta \cdot \mu$.

Proof The first inequality follows from $p_i \leq \overline{p}_i$. Note that for $i \in B_k$ we have $\lceil i/\beta \rceil \leq \beta^k$. Thus, $p_{\lceil i/\beta \rceil} \geq p_{\beta^k}$. Hence,

$$\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i \leq \sum_{i=1}^{n} p_{\lceil i/\beta \rceil} \leq \beta \sum_{i=1}^{n} p_i = \beta \cdot \mu.$$

□

2.2 Subset Sampling

In this section we consider SortedSubsetSampling and UnsortedSubsetSampling and prove the upper bounds of Theorems 1.5 and 1.6. An interesting interplay between both of these problem variants will be revealed on the way.

We begin with an algorithm for unsorted probabilities that has a quite large preprocessing time, but will be used as a base case later. The algorithm uses Theorem 1.3.

Lemma 2.2 UnsortedSubsetSampling *can be solved in preprocessing time* $\mathcal{O}(n^2)$ *and expected query time* $\mathcal{O}(1 + \mu)$.

Proof For $i \in [n]$ let us denote by S_i the smallest sampled element that is at least i, or ∞, if no such element is sampled. Then S_i is a random variable such that

$$\Pr[S_i = j] = p_j \prod_{i \leq k < j} (1 - p_k) \quad \text{and} \quad \Pr[S_i = \infty] = \prod_{i \leq k \leq n} (1 - p_k).$$

All these probabilities can be computed on a Real RAM in time $\mathcal{O}(n)$ for any i, i.e., in time $\mathcal{O}(n^2)$ for all i. After having computed the distribution of the S_i's, we execute, for each $i \in [n]$, the preprocessing of Theorem 1.3, which allows us to quickly sample S_i later on. This preprocessing takes time $\mathcal{O}(n^2)$.

For querying, we start at $i = 1$ and iteratively sample the smallest element $j \geq i$ (i.e., sample S_i), output j, and start over with $i = j + 1$. This is done until $j = \infty$ or $i = n + 1$. Note that any sample of S_i can be computed in $\mathcal{O}(1)$ time with our preprocessing, so that sampling $S \subseteq [n]$ will be done in time $\mathcal{O}(1 + |S|)$. The expected runtime is, thus, $\mathcal{O}(1 + \mu)$. □

After having established this base case, we turn towards reductions between SORTEDSUBSETSAMPLING and UNSORTEDSUBSETSAMPLING. First, we give an algorithm for UNSORTEDSUBSETSAMPLING that reduces the problem to SORTEDSUBSETSAMPLING. For this, we roughly sort the probabilities so that we get good upper bounds for each probability. Then these upper bounds will be a sorted instance. After querying from this sorted instance, we use rejection (see, e.g., [8]) to sample with the original probabilities.

Lemma 2.3 *Assume that* SORTEDSUBSETSAMPLING *can be solved in preprocessing time* $t_p(n, \mu)$ *and expected query time* $t_q(n, \mu)$*, where* t_p *and* t_q *are monotonically increasing in* n *and* μ*. Then* UNSORTEDSUBSETSAMPLING *can be solved in preprocessing time* $\mathcal{O}(n + t_p(n, 2\mu + 1))$ *and expected query time* $\mathcal{O}(1 + \mu + t_q(n, 2\mu + 1))$.

Proof Let $\mathbf{p} = (p_1, \ldots, p_n)$ be an input sequence to UNSORTEDSUBSETSAMPLING. For preprocessing, we permute the input $\mathbf{p}$ so that it is approximately sorted, by partitioning it into buckets $U_k := \{i \in [n] \mid 2^{-k} \geq p_i > 2^{-k-1}\}$, for $k \in \{0, 1, \ldots, L-1\}$, and $U_L := \{i \in [n] \mid 2^{-L} \geq p_i\}$, where $L = \lceil \log n \rceil$. For each $i \in U_k$ we set $\overline{p}_i := 2^{-k}$, which is an upper bound on p_i. We sort the probabilities $\overline{p}_i$, $i \in [n]$, descendingly using bucket sort with the buckets U_k, yielding $\overline{p}'_1 \geq \ldots \geq \overline{p}'_n$. In this process we store the original index $\mathrm{ind}(i)$ corresponding to $\overline{p}'_i$, so that we can find $p_{\mathrm{ind}(i)}$ corresponding to $\overline{p}'_i$ in constant time. Then we run the preprocessing of SORTEDSUBSETSAMPLING on $\overline{p}'_1, \ldots, \overline{p}'_n$. Note that

$$\overline{\mu} := \sum_{i=1}^{n} \overline{p}'_i = \sum_{i=1}^{n} \overline{p}_i \leq \sum_{i=1}^{n} \max\left\{2p_i, \frac{1}{n}\right\} \leq 2\mu + 1.$$

Thus, the total preprocessing time is bounded by

$$\mathcal{O}(n) + t_p(n, \overline{\mu}) = \mathcal{O}(n + t_p(n, 2\mu + 1)),$$

establishing the first claim.

For querying, we query $\overline{p}'_1, \ldots, \overline{p}'_n$ using SORTEDSUBSETSAMPLING, yielding $S' \subseteq [n]$. We compute $S := \{\mathrm{ind}(i) \mid i \in S'\}$. Each $i \in S$ was sampled with probability $\overline{p}_i \geq p_i$. We use rejection to get this probability down to p_i. For this, we generate for each $i \in S$ a random number rand() and check whether it is smaller than or equal to $p_i/\overline{p}_i$. If this is not the case, we delete i from S. Note that we have thus sampled i with probability p_i, and all elements are sampled independently, so S has the desired distribution. Moreover, since the expected size of S' is $\overline{\mu}$, the expected query time is bounded by

$$t_q(n, \overline{\mu}) + \mathcal{O}(1 + \mathbb{E}[|S'|]) = \mathcal{O}(1 + \mu + t_q(n, 2\mu + 1)),$$

and the second claim is also established. □

We also give a reduction in the other direction, solving SORTEDSUBSETSAMPLING by UNSORTEDSUBSETSAMPLING.

Lemma 2.4 *Let $2 \le \beta < n$. Assume that* UnsortedSubsetSampling *can be solved in preprocessing time $t_p(n, \mu)$ and expected query time $t_q(n, \mu)$, where t_p and t_q are monotonically increasing in n and μ. Then* SortedSubsetSampling *can be solved in preprocessing time $\mathcal{O}(\log_\beta n + t_p(1+\log_\beta n, \beta\mu))$ and expected query time $\mathcal{O}(1+\beta\mu+t_q(1+\log_\beta n, \beta\mu))$. More precisely, our preprocessing computes a value $\overline{\mu}$ with $\mu \le \overline{\mu} \le \beta\mu$ and the expected query time is $\mathcal{O}(1+\overline{\mu}+t_q(1+\log_\beta n, \overline{\mu}))$.*

Proof Let $p_1, \ldots, p_n$ be an input sequence to SortedSubsetSampling. As in Sect. 2.1, we consider blocks $B_k = \{i \in [n] \mid \beta^k \le i < \beta^{k+1}\}$, with $k \in \{0, \ldots, L\}$ and $L := \lfloor \log_\beta n \rfloor$, and let $\overline{p}_i := p_{\beta^k}$ for $i \in B_k$. We will first sample with respect to the probabilities $\overline{p}_i$—call the sampled elements *potential*—and then use rejection. For this, let X_k be an indicator random variable for the event that we sample *at least one* potential element in B_k. Then

$$q_k := \Pr[X_k = 1] = 1 - (1 - p_{\beta^k})^{|B_k|}.$$

Moreover, let Y_k be a random variable for the index of the first potential element in block B_k, minus β^k. Let $Y_k = \infty$, if no element in B_k is sampled as a potential element. Then $\Pr[Y_k = i] = p_{\beta^k}(1 - p_{\beta^k})^i$ for $i \in \{0, \ldots, |B_k| - 1\}$, and $\Pr[Y_k = \infty] = \Pr[X_k = 0] = 1 - q_k$. We calculate

$$\Pr[Y_k = i \mid X_k = 1] = \frac{\Pr[Y_k = i]}{\Pr[X_k = 1]} = \frac{p_{\beta^k}}{q_k}(1 - p_{\beta^k})^i, \qquad i \in \{0, \ldots, |B_k| - 1\}.$$

Since this is a (truncated) geometric distribution, we can sample from it in constant time. Indeed, consider a geometric random variable Z with parameter p truncated at m, i.e., $\Pr[Z = i] = p(1-p)^i/q$ for $i \in \{0, \ldots, m-1\}$, where $q := 1 - (1-p)^m$. Then $\lfloor \log(1 - q \cdot \mathrm{rand}())/\log(1-p) \rfloor$ samples from Z; see also [8].

For preprocessing, we first compute the probabilities q_k, $k \in \{0, \ldots, L\}$. This can be done in time $\mathcal{O}(L) = \mathcal{O}(\log_\beta n)$ (as $a^b = \exp(b \ln a)$ can be computed in constant time on a Real RAM). Then we run the preprocessing of UnsortedSubsetSampling on them; note that the q_k's are in general not sorted. In total, the preprocessing time is at most

$$\mathcal{O}(\log_\beta n) + t_p(1 + \log_\beta n, \nu), \qquad \text{where} \qquad \nu = \sum_{i=0}^{\lfloor \log_\beta n \rfloor} q_k.$$

Using that $(1-x)^y \ge 1 - xy$ for $0 < x < 1$ and $y \ge 1$ we obtain

$$\nu = \sum_{i=0}^{\lfloor \log_\beta n \rfloor} 1 - (1 - p_{\beta^k})^{|B_k|} \le \sum_{i=0}^{\lfloor \log_\beta n \rfloor} p_{\beta^k}|B_k| = \sum_{i=1}^{n} \overline{p}_i = \overline{\mu}.$$

Using Lemma 2.1 we obtain $\nu \le \beta\mu$, and the bound $\mathcal{O}(\log_\beta n + t_p(1+\log_\beta n, \beta\mu))$ for the total preprocessing time follows immediately.

For querying, we query the blocks B_k that contain potential elements, using the query algorithm for UNSORTEDSUBSETSAMPLING on $q_0, \ldots, q_k$. Then, for each block B_k that contains a potential element, we sample all potential elements in this block. Note that the first of the potential elements in B_k is distributed as $\Pr[Y_k = i \mid X_k = 1]$, which is geometric, so we can sample from it in constant time, while all further potential elements are distributed as Y_k (but only on the remainder of the block), which is still geometric. Then, after having sampled a set $\overline{S}$ of potential elements, we keep each $i \in \overline{S}$ only if $\text{rand}() \leq p_i/\overline{p}_i$. This yields a random sample $S \subseteq \overline{S}$ with the desired distribution. The overall query time is then at most

$$t_q(1 + \log_\beta n, \nu) + \mathcal{O}(1 + |\overline{S}|) \leq t_q(1 + \log_\beta n, \overline{\mu}) + \mathcal{O}(1 + |\overline{S}|)$$

As the expected value of $|\overline{S}|$ is $\overline{\mu} \leq \beta\mu$ the proof is completed. □

Next, we put the above three lemmas together to prove the upper bounds of Theorems 1.5 and 1.6.

Proof of Theorem 1.6, upper bound To solve UNSORTEDSUBSETSAMPLING, we use the reduction Lemma 2.3 and then Lemma 2.4 (where we set $\beta = 2$), followed by the base case Lemma 2.2. This reduces the instance size from n to $\mathcal{O}(\log n)$, so that preprocessing costs $\mathcal{O}(n)$ for the invocation of the first lemma, $\mathcal{O}(\log n)$ for the second, and $\mathcal{O}(\log^2 n)$ for the third. Note that μ is increased only by constant factors, so that we indeed get the a query time of $\mathcal{O}(1 + \mu)$.

For SORTEDSUBSETSAMPLING we first prove a weaker statement than Theorem 1.5, which follows from simply putting together the reductions of this section.

Lemma 2.5 *Let $2 \leq \beta < n$. Then* SORTEDSUBSETSAMPLING *can be solved in preprocessing time $\mathcal{O}(\log_\beta n)$ and expected query time $\mathcal{O}(1 + \beta\mu)$. More precisely, our preprocessing computes a value $\overline{\mu}$ with $\mu \leq \overline{\mu} \leq \beta\mu$ and the expected query time is $\mathcal{O}(1 + \overline{\mu})$.*

Proof To solve SORTEDSUBSETSAMPLING, we use the reduction presented in Lemma 2.4 followed by the upper bound of Theorem 1.6 that we proved above. This reduces the instance size from n to $\mathcal{O}(\log_\beta n)$ while μ is increased to $\mathcal{O}(1 + \beta\mu)$. We obtain the desired preprocessing time $\mathcal{O}(\log_\beta n)$ and query time $\mathcal{O}(1 + \beta\mu)$. □

Proof of Theorem 1.5, upper bound Assume that we are allowed preprocessing time $\mathcal{O}(\log_{\tilde{\beta}} n)$ for some $2 \leq \tilde{\beta} < n$. Our algorithm for SORTEDSUBSETSAMPLING simply runs the preprocessing of Lemma 2.5 with $\beta = \tilde{\beta}$ to satisfy the preprocessing time constraint.

For querying, we improve upon the runtime of Lemma 2.5 as follows. For any $\beta \in \{2, \ldots, n\}$, let $\overline{\mu}(\beta)$ be the upper bound on μ computed by Lemma 2.5 given $\mathcal{O}(\log_\beta n)$ preprocessing time. Initially, we set $\beta := \tilde{\beta}$ so that $\overline{\mu}(\beta) = \overline{\mu}(\tilde{\beta})$ was computed by our preprocessing. If $1 + \overline{\mu}(\tilde{\beta}) \leq \log_{\tilde{\beta}} n$ then we run the query algorithm of Lemma 2.5 and are done. Otherwise, we repeatedly set $\beta := \lceil \beta^{1/2} \rceil$ and rerun the

preprocessing of Lemma 2.5, until $\beta = 2$ or $1 + \overline{\mu}(\beta) \leq \log_\beta n$. Then we run the query algorithm of Lemma 2.5.

It remains to analyze the runtime of this query algorithm. We consider three cases. (1) If $1 + \overline{\mu}(\tilde{\beta}) \leq \log_{\tilde{\beta}} n$ then the β-decreasing loop does not start and the query time is $\mathcal{O}(1+\overline{\mu}(\tilde{\beta})) \leq \mathcal{O}(1+\tilde{\beta}\mu)$. (2) If the β-decreasing loop breaks at $\beta = 2$, then since it did not stop at $\beta \in \{3, 4\}$ we have $1 + 4\mu > \log_4 n$, or $\mu = \Omega(\log n)$. In this case, the total query time is $\mathcal{O}(1 + \mu + \log n) = \mathcal{O}(\mu)$. (3) Otherwise the β-decreasing loop stopped at some β^* with $1 + \overline{\mu}(\beta^*) \leq \log_{\beta^*} n$. Using $\overline{\mu}(\beta) \leq \beta\mu$ and that we decrease β by taking its square root, we obtain $\beta^* \geq \gamma^{1/2}$, where $\gamma \geq 2$ satisfies

$$1 + \gamma\mu = \log_\gamma n.$$

The above equation solves to $\gamma = \Theta\big(\frac{\log n}{\mu} \big/ \log\big(\frac{\log n}{\mu}\big)\big)$. This yields a total query time of $\mathcal{O}(\log_{\beta^*} n) = \mathcal{O}(\log_\gamma n) = \mathcal{O}\big(\frac{\log n}{\log(\frac{\log n}{\mu})}\big)$, which proves the claimed query time. □

3 Lower Bounds

We prove most of our lower bounds by reducing the various sampling problems to the following fact, that searching in an unordered array of length m takes time $\Omega(m)$. A notable exception is Lemma 3.4.

Fact 3.1 *Consider problem* ArraySearch*: Given m and query access to an array $A \in \{0, 1\}^m$ consisting of m bits, with exactly one bit set to 1, find the position of this bit. Any randomized algorithm for* ArraySearch *needs $\Omega(m)$ accesses to A in expectation.*

3.1 Proportional Sampling on Unsorted Probabilities

The lower bound for Theorem 1.3 is provided by the following lemma that reduces ArraySearch to UnsortedProportionalSampling. Moreover, the same proof yields the lower bound of Theorem 1.6 for UnsortedSubsetSampling.

Lemma 3.2 *Any single-sample algorithm for* UnsortedProportionalSampling *has expected time $\Omega(n)$. Moreover, any single-sample algorithm for* UnsortedSubsetSampling *has expected time $\Omega(n)$.*

Proof Let A be an instance of ArraySearch of size n, say with 1-bit at position ℓ^*. We consider the instance

$$\mathbf{p}^A = (p_1^A, \ldots, p_n^A) \quad \text{with} \quad p_i^A = A[i].$$

Any sampling algorithm for UnsortedProportionalSampling returns ℓ^* on instance $\mathbf{p}^A$ with probability 1. Thus, simulating any algorithm for UnsortedProportionalSampling (by computing p_i^A on the fly) we obtain an algorithm for finding

the 1-bit of array A. Hence, by Fact 3.1, any algorithm for UNSORTEDPROPORTIONALSAMPLING takes expected time $\Omega(n)$.

Observe that on the same instance $\mathbf{p}^A$ any sampling algorithm for UNSORTEDSUBSETSAMPLING returns the set $\{\ell^*\}$ with probability 1. This needs expected time $\Omega(n)$ for the same reasons. With varying μ, no better bound is possible, either: If $\mu \geq 1$, consider an ARRAYSEARCH instance A of length $n-s$, where $s := \lceil \mu - 1 \rceil$. Let $p_i^A = A[i]$ for $1 \leq i \leq n-s$ and set the last s probabilities p_i^A to values that sum up to $\mu - 1$. Then we still need runtime $\Omega(n-\mu)$ by Fact 3.1. As we also need runtime $\Omega(\mu)$ for outputting the result, the lower bound of $\Omega(n)$ follows. Otherwise, if $\mu < 1$, then we consider $\tilde{p}_i^A := \mu \cdot A[i]$. Since the algorithm does not know μ, it behaves just as in the case $\mu = 1$ *until it reads* $p_{\ell^*}^A$. However, finding ℓ^* takes time $\Omega(n)$, which yields the result. □

3.2 Proportional Sampling on Sorted Probabilities

Here we present the proof of the lower bound of Theorem 1.2 for SORTEDPROPORTIONALSAMPLING.

Proof of Theorem 1.2, lower bound Let $n \in \mathbb{N}$ and $2 \leq \beta \leq \mathcal{O}(\frac{\log n}{\log \log n})$. Let $s_i := \sum_{j=0}^{i-1} \beta^j = (\beta^i - 1)/(\beta - 1)$. Let L be maximal with $s_L \leq n$ and note that $L = \Theta(\log_\beta n)$. Then $\beta \leq \mathcal{O}(\frac{\log n}{\log \log n})$ implies $\beta = \mathcal{O}(L)$. We consider blocks $B_i := \{s_i, s_i + 1, \ldots, s_i + \beta^{i-1} - 1\}$, for $i = 1, \ldots, L$, that partition $\{1, \ldots, s_L\}$.

Let A be an instance of ARRAYSEARCH of size L, say with 1-bit at position ℓ^*. To construct the instance $\mathbf{p} = \mathbf{p}^A = (p_1^A, \ldots, p_n^A)$ we set for any $\ell \in \{1, \ldots, L\}$ and $j \in B_\ell$

$$p_j^A := \beta^{-\ell + A[\ell]},$$

and $p_j^A := 0$ for $s_L < j \leq n$. As block B_ℓ has size β^ℓ, the total probability mass of B_ℓ is $\sum_{j \in B_\ell} p_j^A = \beta^{A[\ell]}$, i.e., it is β for $A[\ell] = 1$, and 1 otherwise. Observe that

$$\mu = \sum_{i=1}^{n} p_i^A = L + \beta - 1,$$

since block B_{ℓ^*} contributes β and each of the other $L - 1$ blocks contributes 1 as total probability mass. Furthermore, note that $p_1^A, \ldots, p_n^A$ is indeed sorted, as the probability of an element in block B_ℓ is smaller by a factor of (at least) β than the probability of an element in $B_{\ell-1}$, except if $\ell = \ell^*$, in which case these probabilities coincide.

In the following we will prove that there is no sampling algorithm where the preprocessing reads at most εL input values and the querying reads at most $\varepsilon\beta$ input values in expectation, for a sufficiently small constant $\varepsilon > 0$. Assume, for the sake of contradiction, that such an algorithm exists. On $\mathbf{p}^A$ we run the preprocessing and

then K times the query algorithm, sampling K numbers $X_1, \ldots, X_K \in \{1, \ldots, n\}$. Denote by Y_k the block of X_k, i.e., $X_k \in B_{Y_k}$. If $A[Y_k] = 1$ for some $1 \le k \le K$ then we return Y_k, otherwise we linearly search for the 1-bit of A.

This yields an algorithm for ARRAYSEARCH, let us analyze its expected number of accesses to A. Since the total probability mass of block B_{ℓ^*} is β, we have

$$\Pr[Y_k = \ell^*] = \frac{\beta}{\mu} = \frac{\beta}{L + \beta - 1} = \Omega\Big(\frac{\beta}{L}\Big),$$

since $\beta = \mathcal{O}(L)$. Thus, $\Pr[\nexists k\colon A[Y_k] = 1] = (1 - \Omega(\beta/L))^K = \exp(-\Omega(K\beta/L))$. Setting $K = \Theta(\log(1/\varepsilon)L/\beta)$ (with sufficiently large hidden constant), this probability is at most ε. Hence, the expected number of accesses to A of the constructed algorithm is (counting preprocessing, K queries, and a possible linear search through A)

$$\varepsilon L + K \cdot \varepsilon\beta + \Pr[\nexists k\colon A[Y_k] = 1] \cdot L \le \mathcal{O}(\log(1/\varepsilon)\varepsilon L).$$

For sufficiently small $\varepsilon > 0$ this contradicts Fact 3.1. □

Note that the same proof also works for single-sample algorithms. In this case the preprocessing reads no input values, and the only restriction is $\beta \le \mathcal{O}(L)$. Setting $\beta = \Theta(\log(n)/\log\log(n))$ this yields a lower bound of $\Omega(\log(n)/\log\log(n))$ on the expected runtime of any single-sample algorithm for SORTEDPROPORTIONALSAMPLING.

3.3 Subset Sampling on Sorted Probabilities

We first prove two lemmas proving lower bounds for SORTEDSUBSETSAMPLING in different situations. Then we show how the lower bound of Theorem 1.5 follows from these lemmas.

Lemma 3.3 *Let $\beta \in \{2, \ldots, n\}$. Consider any data structure for* SORTEDSUBSETSAMPLING *with preprocessing time $\varepsilon \log_\beta n$ (where $\varepsilon > 0$ is a sufficiently small constant) and query time $t_q(n, \mu)$. Then for any $\mu = \mu(n)$ with $\beta(1+\mu) = \mathcal{O}(\log_\beta n)$ we have $t_q(n, \mu) = \Omega(\beta\mu)$.*

Proof We closely follow the proof of the lower bound of Theorem 1.2 (Sect. 3.2). Let $s_i := \sum_{j=0}^{i-1} \beta^j = (\beta^i - 1)/(\beta - 1)$. Let L be maximal with $s_L \le n$ and note that $L = \Theta(\log_\beta n)$. We consider blocks $B_i := \{s_i, s_i + 1, \ldots, s_i + \beta^{i-1} - 1\}$, for $i = 1, \ldots, L$, that partition $\{1, \ldots, s_L\}$.

Note that our assumptions imply $\beta = \mathcal{O}(\log_\beta n)$, from which it follows that $\beta = \mathcal{O}(\log n)$ and thus $L = \Theta(\log_\beta n) = \Omega(\log n/\log\log n)$ grows with n. Since we can assume that n is sufficiently large, we thus can assume the same for L. By assumption we also have $\mu = \mathcal{O}(\log_\beta n) = \mathcal{O}(L)$. If $\mu > L$, then we introduce elements $p_1 = \ldots = p_{\lceil \mu - L \rceil} = 1$. Then on the remainder $p_{\lceil \mu - L \rceil + 1}, \ldots, p_n$ we have a probability mass $\mu - \lceil \mu - L \rceil$, which is at most L, but still $\Omega(\mu)$ (where we use that L is at least a

sufficiently large constant). Hence, it suffices to show that sampling from the remainder takes query time $\Omega(\beta\mu)$. Focussing on this remainder, without loss of generality we can from now on assume $\mu \leq L$.

Let A be an instance of ArraySearch of size L, say with 1-bit at position ℓ^*. To construct the instance $\mathbf{p} = \mathbf{p}^A = (p_1^A, \ldots, p_n^A)$, for some $0 \leq \alpha \leq 1$ we set for any $\ell \in \{1, \ldots, L\}$ and $j \in B_\ell$ the input to $p_j^A := \alpha \cdot \beta^{-\ell+A[\ell]}$, and for $s_L < j \leq n$ to $p_j^A := 0$. As block B_ℓ has size β^ℓ, the total probability mass of B_ℓ is $\sum_{j\in B_\ell} p_j^A = \alpha \cdot \beta^{A[\ell]}$. Observe that $\mu = \sum_{i=1}^n p_i^A = \alpha(L+\beta-1)$ indeed has a solution $0 \leq \alpha \leq 1$, since $\mu \leq L$. Furthermore, note that $p_1^A, \ldots, p_n^A$ is indeed sorted.

Assume for the sake of contradiction that there is a data structure for SortedSubsetSampling where the preprocessing reads at most $\varepsilon \log_\beta n$ input values and the querying reads at most $\varepsilon\beta\mu$ input values in expectation, for a sufficiently small constant $\varepsilon > 0$.

On $\mathbf{p}^A$ we run the preprocessing and then K times the query algorithm, sampling K sets $X_1, \ldots, X_K \subseteq \{1, \ldots, n\}$. For every $x \in \bigcup_{k=1}^K X_k$ we determine its block B_y and check whether $A[y] = 1$. If so, we have found the 1-bit of A. Otherwise we linearly search for the 1-bit of A.

This yields an algorithm for ArraySearch, let us analyze its expected number of accesses to A. Let ℓ^* be the position of the 1-bit in A. The probability of not sampling any $i \in B_{\ell^*}$ in any of the K queries is

$$\prod_{i\in B_{\ell^*}} (1-p_i)^K = (1-\alpha\cdot\beta^{-\ell^*+1})^{K\beta^{\ell^*}} \leq \exp(-K\alpha\beta).$$

This probability becomes at most ε by setting $K = \lceil \ln(1/\varepsilon)/(\alpha\beta)\rceil = \Theta(1 + \log(1/\varepsilon)/(\alpha\beta))$. Hence, the expected number of accesses to A of the constructed algorithm is (counting preprocessing, K queries, and a linear search through A with probability at most ε)

$$\begin{aligned}\mathcal{O}(\varepsilon L + K\cdot\varepsilon\beta\mu + \varepsilon\cdot L) &\leq \mathcal{O}(\varepsilon(L+\beta\mu+\log(1/\varepsilon)\mu/\alpha))\\ &\leq \mathcal{O}(\varepsilon(\log(1/\varepsilon)(L+\beta)+\beta\mu)),\end{aligned}$$

using $\mu = \alpha(L+\beta-1)$. Because of the condition $\beta(1+\mu) = \mathcal{O}(\log_\beta n)$ we can further bound the expected number of accesses to A by $\mathcal{O}(\log(1/\varepsilon)\varepsilon L)$, which contradicts Fact 3.1 for sufficiently small $\varepsilon > 0$. □

Lemma 3.4 *Consider any data structure for* SortedSubsetSampling *with preprocessing time $t_p(n)$ and expected query time $t_q(n,\mu)$. For any $\mu = \mu(n) \leq \frac{1}{2}$ we have*

$$t_p(n) + t_q(n,\mu) = \Omega\Big(\frac{\log n}{\log \frac{\log n}{\mu}}\Big).$$

Note that this lemma directly implies the lower bound of Theorem 1.4 for SortedSubsetSampling assuming $\mu \leq \frac{1}{2}$.

Proof Let (P, Q) be a preprocessing and a query algorithm, and let $\mathbf{p}$ be an instance. Let $D = P(\mathbf{p})$ be the result of the precomputation. By definition we have

$$\Pr[Q(\mathbf{p}, D) = \emptyset] = \prod_{i \in [n]} (1 - p_i) =: \Delta(\mathbf{p}),$$

where the probability goes only over the randomness of the query algorithm, not the preprocessing. If $\mu_{\mathbf{p}} \leq \frac{1}{2}$, since $p_i \leq \mu$ one can easily check that $1 - p_i \geq 4^{-p_i}$, which yields

$$\Delta(\mathbf{p}) \geq 4^{-\mu_{\mathbf{p}}} \geq 4^{-1/2} \geq \tfrac{1}{2}.$$

Let $\mathcal{P} \subseteq [n]$ be the positions $i \in [n]$ at which the preprocessing reads the value p_i during the computation of D, note that $|\mathcal{P}| \leq t_p = t_p(n)$. Without loss of generality, we can assume that $1, n \in \mathcal{P}$, i.e., that the preprocessing reads p_1 and p_n, as this adjustment of the algorithm does not increase its runtime asymptotically.

For an instance $\mathbf{p}$ and $\mathcal{Q} \subseteq [n]$, let $\Delta(\mathbf{p}, \mathcal{Q})$ be the probability that query algorithm Q (with input $\mathbf{p}$, D) reads exactly the values p_i with $i \in \mathcal{Q}$ before returning $\emptyset$. We clearly have

$$\sum_{\mathcal{Q} \subseteq [n]} \Delta(\mathbf{p}, \mathcal{Q}) = \Delta(\mathbf{p}). \tag{1}$$

Furthermore, if $\mu_{\mathbf{p}} \leq \frac{1}{2}$ and the expected query time is at most $t_q = t_q(n, \mu)$, we have

$$\sum_{\substack{\mathcal{Q} \subseteq [n] \\ |\mathcal{Q}| \leq 4t_q}} \Delta(\mathbf{p}, \mathcal{Q}) \geq \frac{1}{4}. \tag{2}$$

Indeed, since $|\mathcal{Q}|$ is a lower bound on the runtime of the query algorithm, denoting by $\mathcal{E}$ the event that algorithm Q on input $\mathbf{p}$, D runs for time at most $4t_q$ we have

$$\sum_{\substack{\mathcal{Q} \subseteq [n] \\ |\mathcal{Q}| \leq 4t_q}} \Delta(\mathbf{p}, \mathcal{Q}) \geq \Pr[Q(\mathbf{p}, D) = \emptyset \text{ and } \mathcal{E}] \geq \Pr[Q(\mathbf{p}, D) = \emptyset] + \Pr[\mathcal{E}] - 1.$$

Since $\Pr[Q(\mathbf{p}, D) = \emptyset] = \Delta(\mathbf{p}) \geq \frac{1}{2}$ and $\Pr[\text{not } \mathcal{E}] \leq \frac{1}{4}$ by Markov's inequality, we obtain (2).

By (2) and since the number of subsets of $[n]$ of size at most $4t_q$ is $\sum_{s=0}^{4t_q} \binom{n}{s} \leq \left(en/4t_q\right)^{4t_q} \leq n^{4t_q}/4$, there exists a set $\mathcal{Q}^* \subseteq [n]$, $|\mathcal{Q}^*| \leq 4t_q$, with

$$\Delta(\mathbf{p}, \mathcal{Q}^*) \geq \frac{1}{4} \cdot \left(\sum_{s=0}^{4t_q} \binom{n}{s} \right)^{-1} \geq n^{-4t_q}. \tag{3}$$

Now we fix the instance $\mathbf{p} = (p_1, \ldots, p_n)$ by setting

$$p_i := \frac{\alpha}{i},$$

for a parameter $\alpha > 0$ chosen such that $\sum_{i=1}^{n} p_i = \alpha H_n = \mu = \mu(n)$, implying $\alpha = \Theta(\mu / \log n)$. Fixing a set $\mathcal{Q}^*$ as above for this instance $\mathbf{p}$, we define a second instance $\mathbf{p}' = (p'_1, \ldots, p'_n)$ by setting

$$p'_i := \min\{p_j \mid i \geq j \in \mathcal{Q}^* \cup \mathcal{P}\}.$$

That is, $\mathbf{p}$ and $\mathbf{p}'$ agree on the read positions $\mathcal{Q}^*$ and $\mathcal{P}$, and at all other positions p'_i is as large as possible with $\mathbf{p}'$ still being sorted. This means that the preprocessing and the query algorithm cannot distinguish between both instances, implying a critical property we will use,

$$\Delta(\mathbf{p}', \mathcal{Q}^*) = \Delta(\mathbf{p}, \mathcal{Q}^*).$$

With this, we obtain

$$\Delta(\mathbf{p}') \overset{(1)}{\geq} \Delta(\mathbf{p}', \mathcal{Q}^*) = \Delta(\mathbf{p}, \mathcal{Q}^*) \overset{(3)}{\geq} n^{-4t_q}. \tag{4}$$

We next bound $\Delta(\mathbf{p}')$. Denote the read positions by $\mathcal{Q}^* \cup \mathcal{P} = \{i_1, \ldots, i_k\}$ with $i_1 \leq \ldots \leq i_k$. Note that $k \leq t_p + 4t_q$. By assumption, we have $i_1 = 1$, $i_k = n$, and we define $i_{k+1} := n + 1$. We obtain

$$\Delta(\mathbf{p}') = \prod_{i \in [n]} (1 - p'_i) = \prod_{\ell=1}^{k} (1 - p_{i_\ell})^{i_{\ell+1} - i_\ell}.$$

Using $1 - x \leq e^{-x}$ for $x \geq 0$ this yields

$$\Delta(\mathbf{p}') \leq \exp\left(- \sum_{\ell=1}^{k} p_{i_\ell} (i_{\ell+1} - i_\ell) \right) = \exp\left(-\alpha \sum_{\ell=1}^{k} \left(\frac{i_{\ell+1}}{i_\ell} - 1 \right) \right).$$

Using the arithmetic-geometric mean inequality we obtain

$$\frac{1}{k} \sum_{\ell=1}^{k} \frac{i_{\ell+1}}{i_\ell} \geq \left(\prod_{\ell=1}^{k} \frac{i_{\ell+1}}{i_\ell} \right)^{1/k} \geq n^{1/k},$$

which yields $\Delta(\mathbf{p}') \leq \exp\left(-\alpha k (n^{1/k} - 1)\right) \leq \exp(-\alpha(n^{1/k} - 1))$. Combining this with (4),

$$\exp\left(-\alpha(n^{1/k} - 1)\right) \geq n^{-4t_q} \geq \exp\left(-\mathcal{O}(\log^2 n)\right),$$

as $t_q = \mathcal{O}(\log n)$ (otherwise the claim follows directly). Taking the logarithm twice and rearranging,

$$k \geq \frac{\log n}{\log(1 + \mathcal{O}(\log^2(n)/\alpha))}.$$

Using $t_p + 4t_q \geq k$ and $\alpha = \Theta(\mu/\log n)$, we obtain

$$t_p + 4t_q \geq \frac{\log n}{\log(\mathcal{O}(\log^3(n)/\mu))},$$

and thus $t_p + t_q = \Omega(\frac{\log n}{\log(\log(n)/\mu)})$. □

A tedious case distinction now shows that the lower bound of Theorem 1.5 follows from the above two lemmas.

Proof of Theorem 1.5, lower bound We prove that any data structure for SORTED-SUBSETSAMPLING with $\varepsilon \log_\beta n$ preprocessing time (where $\varepsilon > 0$ is a sufficiently small constant) needs query time $\Omega(t_q^\beta(n, \mu))$ for any $\mu = \mu(n)$, where

$$t_q^\beta(n, \mu) = \begin{cases} \mu, & \text{if } \mu \geq \frac{1}{2}\log n, \\ 1 + \beta\mu, & \text{if } \mu < \frac{1}{\beta}\log_\beta n, \\ \frac{\log n}{\log(\frac{\log n}{\mu})}, & \text{otherwise.} \end{cases}$$

We consider six (sub-)cases depending on μ and β, in each case reducing the claim to Lemma 3.3 or 3.4.

Case 1, $\mu \geq \frac{1}{2}$: We split this into 3 subcases as follows.

Case 1.1, $\mu \geq \frac{1}{2}\log n$: As the expected output size is μ, the expected query time is always $\Omega(\mu)$, which is tight in this case.

Case 1.2, $\mu \geq \frac{1}{2}$ and $\frac{1}{\beta}\log_\beta n \leq \mu < \frac{1}{2}\log n$: In this case, we can choose $2 \leq \gamma \leq \beta$ such that $\mu = \Theta(\frac{1}{\gamma}\log_\gamma n)$. Solving for γ yields $\gamma = \Theta\big(\frac{\log n}{\mu} \big/ \log\frac{\log n}{\mu}\big)$. We have $\gamma \leq 2\gamma\mu \leq \mathcal{O}(\log_\gamma n)$, so Lemma 3.3 (applied with β replaced by γ) yields a lower bound of $\Omega(\gamma\mu) = \Omega\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big)$ for any data structure with preprocessing time $\mathcal{O}(\log_\beta n) \leq \mathcal{O}(\log_\gamma n)$.

Case 1.3, $\frac{1}{2} \leq \mu < \frac{1}{\beta}\log_\beta n$: These inequalities imply $\beta \leq 2\beta\mu \leq 2\log_\beta n$. Thus, Lemma 3.3 applies, showing that the query time is $\Omega(\beta\mu)$. As any algorithm takes time $\Omega(1)$, the query time is also bounded by $\Omega(1 + \beta\mu)$, as desired.

Case 2, $\mu < \frac{1}{2}$: We split this into three subcases as follows.

Case 2.1, $\frac{1}{\beta}\log_\beta n \leq \mu < \frac{1}{2}$: Note that $\mu \geq \frac{1}{\beta}\log_\beta n$ implies $\beta^2 \geq \beta\log\beta \geq \frac{1}{\mu}\log n$ so that $\log\beta = \Omega\big(\log\frac{\log n}{\mu}\big)$. Hence, the preprocessing time is $\varepsilon\log_\beta n = \mathcal{O}\big(\varepsilon\frac{\log n}{\log\frac{\log n}{\mu}}\big)$. For sufficiently small $\varepsilon > 0$, Lemma 3.4 now implies $t_q(n, \mu) = \Omega\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big)$, as desired.

Case 2.2, $\mu < \frac{1}{2}$ and $\frac{1}{\beta^3}\log n \le \mu < \frac{1}{\beta}\log_\beta n$: Then $\log\beta = \Omega\big(\log\frac{\log n}{\mu}\big)$ and $\log_\beta n = \mathcal{O}\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big)$. Hence, with $\varepsilon\log_\beta n$ preprocessing time and sufficiently small $\varepsilon > 0$, Lemma 3.4 implies that $t_q(n,\mu) = \Omega\big(\frac{\log n}{\log\frac{\log n}{\mu}}\big) \ge \Omega(\log_\beta n) \ge \Omega(\beta\mu)$, where the last inequality follows from $\mu < \frac{1}{\beta}\log_\beta n$. Since any algorithm takes time $\Omega(1)$, this yields a lower bound of $\Omega(1+\beta\mu)$, as desired.

Case 2.3, $\mu < \frac{1}{2}$ and $\mu < \frac{1}{\beta^3}\log n$: Note that $\mu < \frac{1}{\beta^3}\log n$ implies $\mu < \frac{1}{\beta}\log_\beta n$. Thus, if $\beta\mu < 1$ then our query time is $\mathcal{O}(1+\beta\mu) = \mathcal{O}(1)$, which is clearly optimal. Hence, assume $\beta\mu \ge 1$. Together with $\mu < \frac{1}{\beta^3}\log n$ this implies $\beta \le \sqrt{\log n}$. Hence,

$$\log_\beta n \ge \Omega\big(\tfrac{\log n}{\log\log n}\big) \gg \mathcal{O}\big(\sqrt{\log n}\big) \ge \beta \ge \Omega(\beta(1+\mu)),$$

where the last inequality uses $\mu < \frac{1}{2}$. Thus, Lemma 3.3 is applicable and we obtain a lower bound of $t_q(n,\mu) = \Omega(\beta\mu) = \Omega(1+\beta\mu)$, as desired. □

4 Reduction from Proportional Sampling to Subset Sampling

In this section, we present a reduction from (SORTED or UNSORTED) PROPORTIONALSAMPLING to (SORTED or UNSORTED) SUBSETSAMPLING. This yields an alternative proof of the upper bounds for PROPORTIONALSAMPLING (Theorems 1.2 and 1.3) using the upper bounds for SUBSETSAMPLING (Theorems 1.5 and 1.6). Moreover, it shows that the classic PROPORTIONALSAMPLING problem is easier than SUBSETSAMPLING (or the former can be seen as a special case of the latter).

We first present a reduction that works for $\mu \le 1$ and yields a query time proportional to $1/\mu$. Then we show how to ensure $1/\beta \le \mu \le 1$ after $\mathcal{O}(\log_\beta n)$ preprocessing, which together with the first reduction shows the main result of this section, Proposition 4.5.

4.1 Special Case

Let $\mathbf{p}$ be an instance to SORTEDPROPORTIONALSAMPLING or UNSORTEDPROPORTIONALSAMPLING. We assume $\mu \le 1$ and will obtain a running time proportional to $\frac{1}{\mu}$, which is most reasonable when μ comes from a small interval $[1/\beta, 1]$. Instead of $\mathbf{p}$ we consider $\mathbf{p}' = (p'_1,\dots,p'_n)$ with $p'_i := p_i/(1+p_i)$. Note that if $\mathbf{p}$ is sorted then $\mathbf{p}'$ is also sorted. Moreover, $\mu' := \sum_{i=1}^n p'_i$ is in the range $[\mu/2, \mu]$.

Let $Y = \text{PROPORTIONALSAMPLING}(\mathbf{p})$ be the random variable denoting proportional sampling on input $\mathbf{p}$, and $X = \text{SUBSETSAMPLING}(\mathbf{p}')$ be the random variable denoting subset sampling on input $\mathbf{p}'$. Then conditioned on sampling exactly one element $X = \{i\}$, this element i is distributed exactly as Y, as formulated by the following lemma.

Lemma 4.1 *We have for all $i \in [n]$*

$$\Pr[X = \{i\} \mid |X| = 1] = \Pr[Y = i].$$

Proof By applying Bayes' rule we infer that

$$\begin{aligned}\Pr[X = \{i\} \mid |X| = 1] &= \Pr[X = \{i\}] / \Pr[|X| = 1] \\ &= \left(\frac{p'_i}{1-p'_i}\prod_{k=1}^{n}(1-p'_k)\right) / \left(\sum_{j=1}^{n}\frac{p'_j}{1-p'_j}\prod_{k=1}^{n}(1-p'_k)\right) \\ &= \left(\frac{p'_i}{1-p'_i}\right) / \left(\sum_{j=1}^{n}\frac{p'_j}{1-p'_j}\right)\end{aligned}$$

Plugging in the definition of p'_i yields

$$\Pr[X = \{i\} \mid |X| = 1] = \frac{p_i}{\sum_{j=1}^{n} p_j} = \Pr[Y = i].$$

and the statement is shown. □

Moreover, the probability of sampling exactly one element is not too small, as shown in the following lemma. This bound is not best possible but sufficient for our purposes.

Lemma 4.2 *If $\mu \leq 1$ then we have*

$$\Pr[|X| = 1] \geq \mu/4.$$

Proof First, observe that by Markov's inequality

$$\Pr[|X| \geq 2] \leq \mathbb{E}[|X|]/2 = \mu'/2 \leq 1/2,$$

and thus, $\Pr[|X| \in \{0, 1\}] \geq 1/2$. Moreover, the definition of X implies that

$$\begin{aligned}\Pr[|X| = 0] &= \prod_{k=1}^{n}(1-p'_k) \quad \text{and} \quad \Pr[|X| = 1] \\ &= \sum_{j=1}^{n}\frac{p'_j}{1-p'_j}\prod_{k=1}^{n}(1-p'_k) = \mu \cdot \Pr[|X| = 0].\end{aligned}$$

By putting everything together we obtain that $\Pr[|X| = 1](1 + \frac{1}{\mu}) \geq 1/2$, and thus

$$\Pr[|X| = 1] \geq \mu \cdot \frac{1}{2(1+\mu)} \geq \frac{\mu}{4},$$

as claimed. □

We put these facts together to show the following result. We need $\mu \leq 1$, and we want μ as large as possible, since the obtained running time is proportional to $\frac{1}{\mu}$. In the next section we will see that we can assume $\frac{1}{\beta} \leq \mu \leq 1$ after preprocessing $\mathcal{O}(\log_\beta n)$.

Lemma 4.3 *Assume that (*SORTED *or* UNSORTED*)* SUBSETSAMPLING *can be solved in preprocessing time $t_p(n, \mu)$ and expected query time $t_q(n, \mu)$, where t_p and t_q are monotonically increasing in n and μ. Then (*SORTED *or* UNSORTED*, respectively)* PROPORTIONALSAMPLING *on instances with $\mu \leq 1$ can be solved in preprocessing time $\mathcal{O}(t_p(n, \mu))$ and expected query time $\mathcal{O}(\frac{1}{\mu} \cdot t_q(n, \mu))$.*

Proof For preprocessing, given input $\mathbf{p}$, we run the preprocessing of SUBSETSAMPLING on input $\mathbf{p}'$. This does not mean that we compute the vector $\mathbf{p}'$ beforehand, but if the preprocessing algorithm of SUBSETSAMPLING reads the i-th input value, we compute $p'_i = p_i/(1 + p_i)$ on the fly, so that preprocessing needs runtime $\mathcal{O}(t_p(n, \mu))$ (recall that $\mu' \leq \mu$). It allows to sample X later on in expected runtime $\mathcal{O}(t_q(n, \mu))$ using the same trick of computing $\mathbf{p}'$ on the fly.

For querying, we repeatedly sample X until we sample a set S of size one. Returning the unique element of S results in a proper sample according to SORTEDPROPORTIONALSAMPLING by Lemma 4.1. Moreover, by Lemma 4.2 and the fact that sampling X needs expected time $\mathcal{O}(t_q(n, \mu))$ after our preprocessing, the total expected query time is $\mathcal{O}(\frac{1}{\mu} \cdot t_q(n, \mu))$. □

4.2 General Case

In this subsection we reduce the general case with arbitrary μ to the special case $1/\beta \leq \mu \leq 1$. In the unsorted case, we simply compute μ exactly in time $\mathcal{O}(n)$, which shows the following proposition. In the sorted case, we approximate μ using an idea of Sect. 2.1, see Proposition 4.5.

Proposition 4.4 *Assume that* UNSORTEDSUBSETSAMPLING *can be solved in preprocessing time $t_p(n, \mu)$ and expected query time $t_q(n, \mu)$, where t_p and t_q are monotonically increasing in n and μ. Then* UNSORTEDPROPORTIONALSAMPLING *can be solved in preprocessing time $\mathcal{O}(n + t_p(n, 1))$ and expected query time $\mathcal{O}(t_q(n, 1))$.*

Note that plugging Theorem 1.6 into the above proposition yields the upper bound of Theorem 1.3.

Proof In the preprocessing we compute μ in time $\mathcal{O}(n)$, and set $\tilde{p}_i := p_i/\mu$ for $i \in [n]$. This rescaling ensures $\tilde{\mu} = \sum_i \tilde{p}_i = 1$. Then we run the algorithm guaranteed by Lemma 4.3 on $\tilde{p}_1, \ldots, \tilde{p}_n$. □

Proposition 4.5 *Let $\beta \in \{2, \ldots, n\}$. Assume that* SORTEDSUBSETSAMPLING *can be solved in preprocessing time $t_p(n, \mu)$ and expected query time $t_q(n, \mu)$, where t_p and t_q are monotonically increasing in n and μ. Then* SORTEDPROPORTIONALSAMPLING *can be solved in preprocessing time $\mathcal{O}(\log_\beta n + t_p(n, 1))$ and expected query time $\mathcal{O}(\max_{1/\beta \leq \nu \leq 1} \frac{1}{\nu} t_q(n, \nu))$.*

Note that plugging Theorem 1.5 into the above proposition yields the upper bound of Theorem 1.2 (to see the bound on the query time, note that we can set $t_q(n, \mu) = \mathcal{O}(1+\beta\mu)$ by Theorem 1.5 or Lemma 2.5, so that $\max_{1/\beta \le \nu \le 1} \frac{1}{\nu} t_q(n, \nu) = \mathcal{O}(\max_{1/\beta \le \nu \le 1} \frac{1}{\nu}(1 + \beta\nu)) = \mathcal{O}(\beta)$).

Proof Let $\mathbf{p}$ be an instance of SortedProportionalSampling with $\mu = \sum_{i=1}^{n} p_i$. As in Sect. 2.1 we consider the blocks $B_k := \{i \in [n] \mid \beta^k \le i < \beta^{k+1}\}$ with $0 \le k \le L := \lfloor \log_\beta n \rfloor$ and set $\overline{p}_i := p_{\beta^k}$ for $i \in B_k$. Then for $\overline{\mu} := \sum_{i=1}^{n} \overline{p}_i$ we have $\mu \le \overline{\mu} \le \beta \cdot \mu$ by Lemma 2.1. Note that we can compute $\overline{\mu}$ in time $\mathcal{O}(\log_\beta n)$, as

$$\overline{\mu} = \sum_{k=0}^{L} p_{\beta^k} \cdot \left(\min(\beta^{k+1}, n+1) - \beta^k\right).$$

With these observations at hand, for preprocessing, we compute $\overline{\mu}$ and consider $\mathbf{p}' = (p'_1, \ldots, p'_n)$ with $p'_i := p_i/\overline{\mu}$. Since $\mu \le \overline{\mu} \le \beta \cdot \mu$ we have $\mu' := \sum_{i=1}^{n} p'_i$ in the range $[1/\beta, 1]$. Thus, we can run the preprocessing of SortedProportionalSampling on $\mathbf{p}'$; Lemma 4.3 is applicable since $\mathbf{p}'$ has $\mu' \in [1/\beta, 1]$. We do this without computing the whole vector $\mathbf{p}'$. Instead, if the preprocessing algorithm reads the i-th input value, we compute p'_i on the fly. This way we need a total runtime for preprocessing of $\mathcal{O}(\log_\beta n + t_p(n, 1))$.

For querying, Lemma 4.3 allows us to query according to $\mathbf{p}'$ in expected runtime $\mathcal{O}(\frac{1}{\mu'} t_q(n, \mu')) \le \mathcal{O}(\max_{1/\beta \le \nu \le 1} \frac{1}{\nu} t_q(n, \nu))$, where we again compute values of $\mathbf{p}'$ on the fly as needed. As we want to sample proportionally to the input distribution, a sample with respect to $\mathbf{p}'$ has the same distribution as a sample with respect to $\mathbf{p}$, so that we simply return the sampled number. □

5 Relaxations

In this section we describe some natural relaxations for the input and machine model studied so far in this paper.

Large Deviations for the Running Times The query runtimes in Theorems 1.2, 1.5 and 1.6 are, in fact, not only small in expectation, but they are also concentrated, i.e., they satisfy large deviation estimates in the following sense. Let t be the expected runtime bound and T the actual runtime. Then

$$\Pr[T > kt] = e^{-\Omega(k)},$$

where the asymptotics are with respect to k. This is shown rather straightforwardly along the lines of our proofs of these theorems, except the fact that the size of the random set X in SubsetSampling is concentrated. Note that for any $a > 1$ the Chernoff bound shows that

$$\Pr[|X| > a\mu] < \left(\frac{e^{a-1}}{a^a}\right)^{\mu} \le \left(\frac{e}{a}\right)^{a\mu}.$$

For $\mu \ll 1$ this inequality does *not* show a tail bound of $e^{-\Omega(k)}$ for $\Pr[|X| > k\mu]$, and in fact such a tail bound does not hold. However, it suffices that $|X|$ is not much larger than $1+\mu$ to bound our algorithms' running times, and this indeed has an exponential tail bound, since by setting $a = k(\mu+1)/\mu$ we obtain

$$\Pr[|X| > k(\mu+1)] < \left(\frac{e\mu}{k(\mu+1)}\right)^{k(\mu+1)} \leq \left(\frac{k}{e}\right)^{-k}.$$

Partially Sorted Input The condition of sorted input for SORTEDSUBSETSAMPLING and SORTEDPROPORTIONALSAMPLING can easily be relaxed, as long as we have sorted upper bounds of the probabilities. Given input $\mathbf{p}$ and sorted $\overline{\mathbf{p}}$ with $p_i \leq \overline{p}_i$ for all $i \in [n]$, we simply sample according to $\overline{\mathbf{p}}$ and use rejection to get down to the probabilities $\mathbf{p}$. This allows for the optimal query time $\mathcal{O}(1+\mu)$ as long as $\overline{\mu} = \sum_{i=1}^{n} \overline{p}_i = \mathcal{O}(1+\mu)$, where $\mu = \sum_{i=1}^{n} p_i$.

Unimodular Input Many natural distributions $\mathbf{p}$ are not sorted, but unimodular, meaning that p_i is monotonically increasing for $1 \leq i \leq m$ and monotonically decreasing for $m \leq i \leq n$ (or the other way round). Knowing m, we can run the algorithms developed in this paper on both sorted halfs, and combine the return values, which gives an optimal query algorithm for unimodular inputs. Alternatively, if we have strong monotonicity, we can search for m in time $\mathcal{O}(\log n)$ using ternary search.

This can be naturally generalized to k-modular inputs, where the monotonicity changes k times.

Approximate Input In some applications it may be costly to compute the probabilities p_i exactly, but we are able to compute approximations $\overline{p}_i(\varepsilon) \geq p_i \geq \underline{p}_i(\varepsilon)$, with relative error at most ε, where the cost of computing these approximations depends on ε. We can still guarantee optimal query time, if the costs of computing these approximations are small enough, see e.g. [12].

We sketch this for SUBSETSAMPLING. We can surely sample a superset $\overline{S}$ with respect to the probabilities $\overline{p}_i(\frac{1}{2})$. Then we want to use rejection, i.e., for each element $i \in \overline{S}$ we want to compute a random number $r := \text{rand}()$ and delete i from $\overline{S}$ if $r \cdot \overline{p}_i(\frac{1}{2}) > p_i$, to get a sample set S. This check can be performed as follows. We initialize $k := 1$. If $r \cdot \overline{p}_i(\frac{1}{2}) > \overline{p}_i(2^{-k})$ we delete i from $\overline{S}$. If $r \cdot \overline{p}_i(\frac{1}{2}) \leq \underline{p}_i(2^{-k})$ we keep i and are done. Otherwise, we increase k by 1. This method needs an expected number of $\mathcal{O}(1)$ rounds of increasing k; the probability of needing k rounds is $\mathcal{O}(2^{-k})$. Hence, if the cost of computing $\overline{p}_i(\varepsilon)$ and $\underline{p}_i(\varepsilon)$ is $\mathcal{O}(\varepsilon^{-c})$ with $c < 1$, the expected overall cost is constant, and we get an optimal expected query time of $\mathcal{O}(1+\mu)$.

Word RAM Throughout the paper we worked in the Real RAM model of computation, where every memory cell can store a real number. In the more realistic Word RAM model each cell consists of $w = \Omega(\log n)$ bits and any reasonable operation on two words can be performed in constant time. Additionally to the standard repertoire of operations, we assume that we can generate a uniformly random word in constant time. It is known that in this model Bernoulli and geometric random variates can be drawn in constant time [2] and the classic aliasing method for UNSORTEDPROPOR-

TIONALSAMPLING still works [3]. This already allows one to translate large parts of the algorithms of this paper to the Word RAM. Unfortunately, terms like $\prod_{1\le k\le n}(1-p_k)$ (see Sect. 2.2) cannot be evaluated exactly on the Word RAM, as the result would need at least n bits. This difficulty can be solved by working with $\mathcal{O}(\log n)$ bit approximations and increasing the precision as needed, similarly to the generalization to *approximate input* that we discussed in the last paragraph. This way one can obtain a complete translation of our algorithms to the Word RAM. We omit the details.

Acknowledgements Open access funding provided by Max Planck Society (or associated institution if applicable).

References

1. Borodin, A., Munro, I.: The Computational Complexity of Algebraic and Numeric Problems. Elsevier Publishing Company, London (1975)
2. Bringmann, K., Friedrich, T.: Exact and efficient generation of geometric random variates and random graphs. In: Proceedings of 40th International Colloquium on Automata, Languages, and Programming (ICALP'13), pp. 267–278 (2013)
3. Bringmann, K., Green Larsen, K.: Succinct sampling from discrete distributions. In Proceedings of 45th Annual ACM Symposium on Theory of Computing (STOC'13), pp. 775–782 (2013)
4. Chung, Fan, Linyuan, Lu: The average distance in a random graph with given expected degrees. Internet Math. **1**(1), 91–113 (2004)
5. Devroye, L.: Nonuniform Random Variate Generation. Springer, New York (1986)
6. Flajolet, P., Saheb, N.: The complexity of generating an exponentially distributed variate. J. Algorithms **7**(4), 463–488 (1986)
7. Hagerup, T., Mehlhorn, K. and Munro, I.: Maintaining discrete probability distributions optimally. In: Proceedings of 20th International Colloquium on Automata, Languages, and Programming (ICALP '93), pp. 253–264 (1993)
8. Knuth, D.E.: The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd edn. Addison-Wesley Publishing Company, Boston (2009)
9. Knuth, D.E., Yao, A.C.: The complexity of nonuniform random number generation. In: Traub, J.F. (ed.) Algorithms and Complexity: New Directions and Recent Results, Proceedings of a Symposium, pp. 357–428. Carnegie-Mellon University, Computer Science Department, Academic Press, New York, NY (1976)
10. Matias, Y., Vitter, J.S., Ni, W.-C.: Dynamic generation of discrete random variates. Theory Comput Syst **36**(4), 329–358 (2003)
11. Miller, J.C., Hagberg, A.A.: Efficient generation of networks with given expected degrees. In: Proceedings of 8th International Workshop Algorithms and Models for the Web Graph (WAW'11), pp. 115–126 (2011)
12. Nacu, Ş., Peres, Y.: Fast simulation of new coins from old. Ann. Appl. Probab. **15**(1A), 93–115 (2005)
13. Preparata, F.P., Shamos, M.I.: Computational Geometry. Texts and Monographs in Computer Science. Springer, New York (1985)
14. Pătraşcu, M.: WebDiarios de Motocicleta, Sampling a discrete distribution. http://infoweekly.blogspot.com/2011/09/sampling-discrete-distribution.html (2011)
15. Tsai, M.-T., Wang, D.-W., Liau, C.-J., Hsu, T.-S.: Heterogeneous subset sampling. In: Proceedings of 16th Annual International Computing and Combinatorics Conference (COCOON '10), pp. 500–509 (2010)
16. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1), 37–57 (1985)

17. Walker, A.J.: New fast method for generating discrete random numbers with arbitrary distributions. Electron. Lett. **10**, 127–128 (1974)
18. Yao, A.C.: Context-free grammars and random number generation. In: Combinatorial Algorithms on Words **12**, 357–361 (1985)