# Exploring Differential Obliviousness

**Amos Beimel**[1]
Dept. of Computer Science, Ben-Gurion University, Israel
amos.beimel@gmail.com

**Kobbi Nissim**
Dept. of Computer Science, Georgetown University, Washington, D.C., USA
kobbi.nissim@georgetown.edu

**Mohammad Zaheri**
Dept. of Computer Science, Georgetown University, Washington, D.C., USA
mz394@georgetown.edu

## Abstract

In a recent paper, Chan et al. [SODA '19] proposed a relaxation of the notion of (full) memory obliviousness, which was introduced by Goldreich and Ostrovsky [J. ACM '96] and extensively researched by cryptographers. The new notion, *differential obliviousness*, requires that any two neighboring inputs exhibit similar memory access patterns, where the similarity requirement is that of differential privacy. Chan et al. demonstrated that differential obliviousness allows achieving improved efficiency for several algorithmic tasks, including sorting, merging of sorted lists, and range query data structures.

In this work, we continue the exploration of differential obliviousness, focusing on algorithms that do not necessarily examine all their input. This choice is motivated by the fact that the existence of logarithmic overhead ORAM protocols implies that differential obliviousness can yield at most a logarithmic improvement in efficiency for computations that need to examine all their input. In particular, we explore property testing, where we show that differential obliviousness yields an almost linear improvement in overhead in the dense graph model, and at most quadratic improvement in the bounded degree model. We also explore tasks where a non-oblivious algorithm would need to explore different portions of the input, where the latter would depend on the input itself, and where we show that such a behavior can be maintained under differential obliviousness, but not under full obliviousness. Our examples suggest that there would be benefits in further exploring which class of computational tasks are amenable to differential obliviousness.

**2012 ACM Subject Classification** Security and privacy → Privacy-preserving protocols

**Keywords and phrases** Differential Obliviousness, Differential Privacy, Oblivious RAM, Graph Property Testing

**Digital Object Identifier** 10.4230/LIPIcs.APPROX-RANDOM.2019.65

**Category** RANDOM

**Related Version** A full version of the paper is available at `https://arxiv.org/abs/1905.01373`.

**Funding** Work supported by NSF grant No. 1565387 TWC: Large: Collaborative: Computing Over Distributed Sensitive Data. A.B. is additionally supported by ISF grant no. 152/17, a grant from the Cyber Security Research Center at Ben-Gurion University, and ERC grant 742754 (project NTSC).

## 1 Introduction

A program's memory access pattern can leak significant information about the private information used by the program even if the memory content is encrypted. Such leakage can turn into a data protection problem in various settings. In particular, where data

[1] Work done while A.B. was visiting Georgetown University.

is outsourced to be stored on an external server, it has been shown that access pattern leakage can be exploited in practical attacks and lead to the compromise of the underlying data [20, 4, 29, 21, 23]. Such leakages can also be exploited when a program is executed in a secure enclave environment but needs to access memory that is external to the enclave.

Memory access pattern leakage can be avoided by employing a strategy that makes the sequence of memory accesses (computationally or statistically) independent of the content being processed. Beginning with the seminal work of Goldreich and Ostrovsky, it is well known how to transform any program running on a random access memory (RAM) machine to one with an *oblivious* memory access pattern while retaining efficiency by using an Oblivious RAM protocol (ORAM) [10, 30, 13]. Current state-of-the-art ORAM protocols achieve logarithmic overhead [2], matching a recent lowerbound by Larsen and Nielsen [24], and protocols with $O(1)$ overhead exist when the server is allowed to perform computation and large blocks are retrieved [6, 28]. To further reduce the overhead, oblivious memory access pattern protocols have been devised for specific tasks, including graph algorithms [3, 17], geometric algorithms [8] and sorting [16, 25]. The latter is motivated by sorting being a fundamental and well researched computational task as well as its ubiquity in data processing.

## 1.1 Differential Obliviousness

Full obliviousness is rather a strong requirement: any two possible inputs (of the same size) should exhibit identical or indistinguishable sequences of memory accesses. Achieving full obliviousness via a generic use of ORAM protocols requires a setup phase with running time (at least) linear in the memory size and then a logarithmic overhead per each memory access.

A recent work by Chan, Chung, Maggs, and Shi [5] put forward a relaxation of the obliviousness requirement where indistinguishability is replaced with differential privacy. Intuitively, this means that any two possible neighboring inputs should exhibit memory access patters that are similar enough to satisfy differential privacy, but may still be too dissimilar to be "cryptographically" indistinguishable. It is not a priori clear whether differential obliviousness can be achieved without resorting to full obliviousness. However, the recent work Chan et al. showed that differential obliviousness does allow achieving improved efficiency for several algorithmic tasks, including sorting (over very small domains), merging of sorted lists, and range query data structures.

Also of relevance are the works by He et al. [19] and Mazloom and Gordon [27], which study protocols for secure multiparty computation in which the parties are allowed to learn information from the computation as long as this information preserves the differential privacy of the input. He et al. and Mazloom and Gordon demonstrate that this leakage is useful: He et al. construct protocols for the private record linkage problem for two databases; Mazloom and Gordon present protocols for histograms, PageRank, and matrix factorization.

Furthermore, even the use of ORAM protocols may be insufficient for preventing leakage in cases where the number of memory probes is input dependent. In fact, Kellaris et al. [21] show that such leakage can result in a complete reconstruction in the case of retrieving elements specified by range queries, as the number of records returned depends on the contents of the data structure. Full obliviousness would require that the sequence of memory accesses would be padded to a maximal one to avoid such leakage, a solution that would have a dire effect on the efficiency of many algorithms. Differential obliviousness may in some cases allow achieving meaningful privacy while maintaining efficiency. Examples of such protocols include the combination of ORAM with differentially private sanitization by Kellaris et al. [22] and the recent work of Chan et al. [5] on range query data structures, which avoids using ORAM.

## 1.2 This Work: Exploring Differential Obliviousness

Noting that the existence of logarithmic overhead ORAM protocols implies that differential obliviousness can yield at most a logarithmic improvement in efficiency for computations that need to examine all their input, we explore tasks where this is not the case. In particular, we focus on property testing and on tasks where the number of memory accesses can depend on the input.

**Property testing.** As evidence that differential obliviousness can provide a significant improvement over full obliviousness, we show in Section 3 that property testers in the dense graph model, where the input is in the adjacency matrix representation [12], can be made differentially oblivious. This result captures a large set of testable graph properties [12, 1] including, e.g., graph bipartitness and having a large clique. Testers in this class probe a uniformly random subgraph and hence are fully oblivious without any modification, as their access pattern does not depend on the input graph. However, this is not the case if the tester reveals its output to the adversary, as this allows learning information about the specific probed subgraph. A fully oblivious tester would need to access a linear-sized subgraph, whereas we show that a differentially oblivious tester only needs to apply the original tester $O(1)$ times.[2]

We also consider property testing in the bounded degree model, where the input is in the incidence lists model [14]. In this model we provide negative results, demonstrating that adaptive testers cannot, generally, be made differentially oblivious without a significant loss in efficiency. In particular, in Section 4 we consider differentially oblivious property testers for connectivity in graphs of degree at most two. For non-oblivious testers, it is known that constant number of probes suffice when the tester is adaptive [14].[3] It is also known that any non-adaptive tester for this task requires probing $\Omega(\sqrt{n})$ nodes [32]. We show that this lowerbound extends to differentially oblivious testers, i.e., any differentially oblivious tester for connectivity in graphs of maximal degree 2 requires $\Omega(\sqrt{n})$ probes. While this still improves over full obliviousness, the gap between full and differential obliviousness is in this case diminished.

**Locating an Object Satisfying a Property.** Here, our goal is to check whether a given data set of objects includes an object that satisfies a specified property. Without obliviousness requirements, a natural approach is to probe elements in a random order until an element satisfying the property is found or all elements were probed. If a $p$ fraction of the elements satisfy the property, then the expected number of probes is $1/p$. This algorithm is in fact instance optimal when the data set is randomly permuted.[4]

A fully oblivious algorithm would require $\Omega(n)$ probes on any dataset even when $p = 1$. In contrast, we demonstrate in Section 5 that with differential obliviousness instance optimality can, to a large extent, be preserved. Our differentially oblivious algorithm always returns a correct answer and makes at most $m$ probes with probability at least $1 - e^{-O(mp)}$.

**Prefix Sum.** Our last example considers a sorted dataset (possibly, the result of an earlier phase in the computation). Our goal is to compute the sum of all records in the (sorted) dataset that are less than or equal to a given value $a$ (see Section 6 for the definition of privacy).

[2] We omit dependencies on privacy and accuracy parameters from this introductory description.
[3] In an adaptive tester at least one choice of a node to probe should depend on information gathered from incidence lists of previously probed nodes.
[4] Our treatment of instance optimality is rather informal. The concept was originally presented in [9].

Without obliviousness requirements, one can find the greatest record less than or equal to value $a$, say, using binary search, and then compute the prefix sum by a quick scan through all records appearing before this record. This algorithm is in fact nearly instance optimal, as it can be shown that any algorithm which returns the correct exact answer with non-negligible probability must probe all entries greater than $a$. However, fully oblivious algorithms would have to probe the entire dataset.

In Section 6, we give our nearly instance optimal differentially oblivious prefix sum algorithm. As the probes of a binary search would leak information about the memory content, we introduce a differentially oblivious "simulation" of the binary search. Our differentially oblivious binary search runs in time $O(\log^2 n)$.

We also address the scenario where there are multiple prefix sum queries to the same database. If the number of queries is bounded by some integer $t$, then each differentially oblivious binary search will run in time $O(t \log^2 n)$ (as we need to run the search algorithm with a smaller privacy parameter $\varepsilon$). Using ORAM, one can answer such queries with $O(n \log n)$ prepossessing time and $O(\log^2 n)$ time per query. Combining our algorithm and ORAM, we can amortize the pre-processing time over $O(\sqrt{n})$ queries, that is, without any pre-processing, the running of time of answering the $i$-th query is $O(i \log^4 n)$ for the first $O(\sqrt{n})$ queries and $O(\log^2 n)$ for any further query.

## 1.3 Background Work

The papers by Chan, Chung, Maggs, and Shi [5], He, Machanavajjhala, Flynn, and Srivastava [19], and by Mazloom and Gordon [27] mentioned above are most relevant for this article. As mentioned above, Kellaris et al. [22] examined a similar concept with the goal of preventing reconstruction attacks in secure remote databases. Goldreich, Goldwasser, and Ron [12] initiated the research on graph property testing. Persiano and Yeo [31] showed that the $O(\log n)$ lowerbound for ORAM of [24] also holds when the security requirement is relaxed to differetial privacy. Goldreich's book on property testing [11] gives sufficient background for our discussion. Dwork, McSherry, Nissim, and Smith [7] defined differential privacy. For more details on ORAM and a list of relevant papers, the reader can consult [2].

## 2 Definitions

### 2.1 Model of Computation

We consider the standard Random Access Memory (RAM) model of computation that consists of a CPU and a memory. The CPU executes a program and is allowed to perform two types of memory operations: read a value from a specified physical address, and write a value to a specified physical address. We assume that the CPU has a private cache of where it can store $O(1)$ values (and/or a polylogarithmic number of bits). As an example, in the setting of a client storing its data on the cloud, the client plays the role of the CPU and the cloud server plays the role of the memory.

We assume that a program's sequence of read and write operations may be visible to an adversary. We will call this sequence the program's access pattern. We will further assume that the memory content is encrypted so that no other information is leaked about the content read from and stored in memory location. The program's access pattern may depend on the program's input, and may hence leak information about it.

**Algorithm 1** Experiment $\mathsf{Exp}_b^{A,M}$ for defining differential obliviousness.

```
$(\mathbf{x}_0, \mathbf{x}_1, st) \leftarrow_{\$} A_1(\lambda, n)$
$b' \leftarrow_{\$} A_2^{\mathcal{M}(\mathbf{x}_b,\cdot)}(st)$
Return $b'$

Oracle $\mathcal{M}(\mathbf{x}, q)$
$\mathsf{out} \leftarrow_{\$} M(\mathbf{x}, q, state)$
Return $\mathsf{Access}^M(\mathbf{x}, q, state)$
```

## 2.2 Oblivious Algorithms

There are various works focused on oblivious algorithms [8, 15, 26] and Oblivious RAM (ORAM) constructions [13]. These works adopt "full obliviousness" as a privacy notion. Suppose that $M(\lambda, \mathbf{x})$ is an algorithm that takes in two inputs, a security parameter $\lambda$ and an input dataset denoted $\mathbf{x}$. We denote by $\mathsf{Access}^M(\lambda, \mathbf{x})$, the ordered sequence of memory accesses the algorithm $M$ makes on the input $\lambda$ and $\mathbf{x}$.

▶ **Definition 1** (Fully Oblivious Algorithms)**.** *Let $\delta$ be a function in a security parameter $\lambda$. We say that algorithm $M$ is $\delta$-statistically oblivious, iff for all inputs $\mathbf{x}$ and $\mathbf{y}$ of equal length, and for all $\lambda$, it holds that $\mathsf{Access}^M(\lambda, \mathbf{x}) \approx^{\delta(\lambda)} \mathsf{Access}^M(\lambda, \mathbf{y})$ where $\approx^{\delta(\lambda)}$ denotes that the two distributions have at most $\delta(\lambda)$ statistical distance. We say that $M$ is perfectly oblivious when $\delta = 0$.*

## 2.3 Differentially Oblivious Algorithms

Suppose that $M(\lambda, \mathbf{x}, q)$ is a (stateful) algorithm that takes in three inputs, a security parameter $\lambda > 0$, an input dataset denoted by $\mathbf{x}$ and a value $q$. We slightly change the definition of differentially oblivious algorithms given in [5]:

▶ **Definition 2** (Neighbor-respecting)**.** *We say that two input datasets $\mathbf{x}$ and $\mathbf{y}$ are neighboring iff they are of the same length and differ in exactly one entry. We say that $A = (A_1, A_2)$ is neighbor-respecting adversary iff for every $\lambda$ and every $n$, $A_1$ outputs neighboring datasets $\mathbf{x}_0, \mathbf{x}_1$, with probability 1.*

▶ **Definition 3.** *Let $\varepsilon, \delta$ be privacy parameters. Let $M$ be a (possibly stateful) algorithm described as above. To an adversary $A$ we associate the experiment in Algorithm 1, for every $\lambda \in \mathbb{N}$. We say that $M$ is $(\varepsilon, \delta)$-adaptively differentially oblivious if for all (computationally unbounded) stateful neighbor-respecting adversary $A$ we have*

$$\Pr[\mathsf{Exp}_0^{A,M}(\lambda, n) = 1] \leq e^{\varepsilon} \cdot \Pr[\mathsf{Exp}_1^{A,M}(\lambda, n) = 1] + \delta.$$

*In Algorithm 1, $\mathsf{Access}^M(\mathbf{x}, q, state)$ denotes the ordered sequence of memory accesses the algorithm $M$ makes on the inputs $\mathbf{x}, q$ and state.*

▶ Remark 4. The notion of adaptivity here is different from the one defined in [5]. We require that the dataset $\mathbf{x}$ remain the same through the experiment whereas in [5] the adaptive adversary can add or remove entries from the dataset.

As with differential privacy, we usually think about $\varepsilon$ as a small constant and require that $\delta = o(1/n)$ where $n = |\mathbf{x}|$ [7]. Observe that if $M$ is $\delta$-statistically oblivious then it is also $(0, \delta)$-differentially oblivious.

The following simple lemma will be useful to analyze our algorithms. The proof of the lemma appears in Appendix A.

▶ **Lemma 5.** *Let $\mathcal{A}$ be an $(\varepsilon, 0)$-differentially oblivious algorithm and $\mathcal{B}$ be an algorithm such that for every dataset $\mathbf{x}$ the statistical distance between $\mathcal{A}(\mathbf{x})$ and $\mathcal{B}(\mathbf{x})$ is at most $\gamma$ (that is, $|\Pr[\mathcal{A}(\mathbf{x}) \in S] - \Pr[\mathcal{B}(\mathbf{x}) \in S]| \leq \gamma$ for every $S$). Then, $\mathcal{B}$ is an $(\varepsilon, (1 + e^{\varepsilon})\gamma)$-differentially oblivious algorithm.*

## 3 Differentially Oblivious Property Testing of Dense Graphs Properties

In this section, we present a differentially oblivious property tester for dense graphs properties in the adjacency matrix representation model. A property tester is an algorithm that decides whether a given object has a predetermined property or is far from any object having this property by examining a small random sample of its input. The correctness requirement of property testers ignores objects that neither have the property nor are far from having the property. However, the privacy requirement is "worst case" and should hold for any two neighboring graphs. For the definition of privacy we say that two graphs $G, G'$ of size $n$ are neighbors if one can get $G'$ by changing the neighbors of exactly one node of $G$.

Property testing of graph properties in the adjacency matrix representation was introduced in [12]. A graph $G = (V, E)$ is represented by the predicate $f_G : V \times V \to \{0, 1\}$ such that $f_G(u, v) = 1$ if and only if $u$ and $v$ are adjacent in $G$. The distance between graphs is defined to be the number of different matrix entries over $|V|^2$. This model is most suitable for dense graphs where the number of edges is $O(|V|^2)$. We define a property $\mathcal{P}$ of graphs to be a subset of the graphs. We write $G \in \mathcal{P}$ to show that graph $G$ has the property $\mathcal{P}$. For example, we can define the bipartiteness property, where $\mathcal{P}$ is the set of all bipartite graphs.[5] We say that an $n$-vertex $G$ is $\gamma$-far from $\mathcal{P}$ if for every $n$-vertex graph $G' = (V', E') \in \mathcal{P}$ it holds that the symmetric difference between $E$ and $E'$ is greater than $\gamma n^2$. We define the property testing in this model as follows:

▶ **Definition 6** ([12])**.** *A $(\beta, \gamma)$-tester for a graph property $\mathcal{P}$ is a probabilistic algorithm that, on inputs $n, \beta, \gamma$, and an adjacency matrix of an $n$-vertex graph $G = (V, E)$:*

1. *Outputs 1 with probability at least $\beta$, if $G \in \mathcal{P}$.*
2. *Outputs 0 with probability at least $\beta$, if $G$ is $\gamma$-far from $\mathcal{P}$.*

We say a tester has one-sided error, if it accepts every graph in $\mathcal{P}$ with probability 1. We say a tester is non-adaptive if it determines all its queries to adjacency matrix only based on $n, \beta, \gamma$, and its randomness; otherwise, we say it is adaptive.

▶ **Example 7** ([12])**.** Consider the following $(2/3, \gamma)$-tester for bipartiteness: Choose a random subset $A \subset V$ of size $\tilde{O}(1/\gamma^2)$ with uniform distribution and output 1 iff the graph induced by $A$ is bipartite. Clearly, if $G$ is bipartite, then the tester will always return 1. Goldreich et al. [12] proved that if $G$ is $\gamma$-far from a bipartite graph, then the probability that the algorithm returns 1 is at most 1/3.

Recall that in the graph property testing, the tester $\mathcal{T}$ chooses a random subset of the graph with uniform distribution to test the property $\mathcal{P}$. Given the access pattern of the tester $\mathcal{T}$, an adversary will learn nothing since it is uniformly random. Thus, the access pattern by itself does not reveal any information about the input graph. However, we assume that the adversary also learns the tester's output and can hence learn some information

[5] Recall that an undirected graph is bipartite (or 2-colorable) if its vertices can be partitioned into two parts, $V_1$ and $V_2$, such that each part is an independent set (i.e., $E \subseteq \{(u, v) : (u, v) \in V_1 \times V_2\}$).

**Algorithm 2** Differentially Oblivious Property Tester $\mathsf{Tester}_{\mathcal{T}}$ for Dense Graphs.

```
Input: graph $G = (V, E)$
 1: Let $c \leftarrow 0$ and $T \leftarrow \frac{\ln(1/2\delta)}{\varepsilon}$
 2: for $i = 1$ to $4T$ do
 3:     if $\mathcal{T}(G) = 1$ then
 4:         $c \leftarrow c + 1$
 5:     end if
 6:     Let $A$ be the subset of vertices chosen by tester $\mathcal{T}$
 7:     Update graph $G$ to be the induced sub-graph on $V \backslash A$
 8: end for
 9: $\hat{T} \leftarrow 3T + \mathrm{Lap}(\frac{1}{\varepsilon})$
10: if $c \geq \min(\hat{T}, 4T)$ then
11:     output 1
12: else
13:     output 0
14: end if
```

about the input graph based on the output of the tester. To protect this information, we run tester $\mathcal{T}$ for constant number of times and output 1 iff the number of times $\mathcal{T}$ outputs 1 exceed a (randomly chosen) threshold.

Let $\mathcal{T}$ be a $(\beta, \gamma)$-tester for a graph property $\mathcal{P}$ where $\beta \leq 1/4$. We write $c_{\beta,\gamma}$ for the number of nodes that $\mathcal{T}$ samples. Note that $c_{\beta,\gamma}$ is constant in the graph size and a function of $\beta$ and $\gamma$. For simplicity, we only consider property testers with one-sided error. In Algorithm 2, we describe a $(\beta', \gamma')$-tester $\mathsf{Tester}_{\mathcal{T}}$ that outputs 1 with probability 1 if $G \in \mathcal{P}$ and outputs 0 with probability at least $\beta'$, if $G$ is $\gamma'$-far from $\mathcal{P}$, where $\beta'$ and $\gamma'$ are defined below.

▶ **Theorem 8.** *Let $\varepsilon, \delta > 0$ and $\gamma' = \gamma - \frac{4\ln(1/2\delta)c_{\beta,\gamma}}{n\varepsilon}$. Algorithm $\mathsf{Tester}_{\mathcal{T}}$ is an $(\varepsilon, \delta(1+e^{\varepsilon}))$-differentially oblivious algorithm that outputs 1 with probability 1 if $G \in \mathcal{P}$, and output 0 with probability at least $1 - \delta - (2\delta)^{\frac{1}{3\varepsilon}}$ if $G$ is $\gamma'$-far from $\mathcal{P}$.*

The proof of Theorem 8 appears in Appendix A.2.

## 4 Lower Bounds on Testing Connectivity in the Incidence Lists Model

We now consider differentially oblivious testing of connectivity in the incidence lists model [14]. In this model a graph has a bounded degree $d$ and is represented as a function $f : V \times [d] \rightarrow V \cup \{0\}$, where $f(v, i)$ is the $i$-th neighbor of $v$ (if no such neighbor exists, then $f(v, i) = 0$). In this model, the relative distance between graphs is normalized by $dn$ – the maximal number of edges in the graph. Formally, for two graphs with $n$ vertices,

$$\mathrm{dist}_d(G_1, G_2) \triangleq \frac{|\{(v, i) : v \in V, i \in [d], f_{G_1}(v, i) \neq f_{G_2}(v, i)\}|}{dn}.$$

A $(\beta, \gamma)$-tester in the incidence lists model is defined as in Definition 6, where a property $\mathcal{P}$ is a set of graphs whose maximal degree is $d$ and the distance to a property is defined with respect to $\mathrm{dist}_d$.

Goldreich and Ron [14] showed how to test if a graph is connected in the incidence list model in time $\tilde{O}(1/\gamma)$. Raskhodnikova and Smith [32] showed that a tester for connectivity (or any non-trivial property) with run-time $o(\sqrt{n})$ has to be adaptive, that is, the nodes that the algorithm probes should depend on the neighbors of nodes the algorithm has already probed (e.g., the algorithm probes some node $u$, discovers that $v$ is a neighbor and $u$, and probes $v$). We strengthen their results by showing that any tester for connectivity in graphs of maximal degree 2 and run-time $o(\sqrt{n})$ cannot be a differentially oblivious algorithm. We stress that adaptivity alone is not a reason for inefficiency with differential obliviousness. In fact, there exist differentially oblivious algorithms that are adaptive (e.g., our algorithm in Section 6).

▶ **Theorem 9.** *Let $\varepsilon, \delta > 0$ such that $e^{4\varepsilon}\delta < 1/16n$. Every $(\varepsilon, \delta)$-differentially private $(3/4, 1/3)$-tester for connectivity in graphs with maximal degree 2 runs in time $\Omega(\sqrt{n}/e^{2\varepsilon})$.*

**Proof.** Let Tester be a $(3/4, 1/3)$-tester for connectivity in graphs of degree at most 2. We somewhat relax the definition of probes and assume that once the tester probes a node, it sees all edges adjacent to this node. We prove that if Tester probes less than $c\sqrt{n}/e^{2\varepsilon}$ nodes (for some constant $c$), then it is not $(\varepsilon, \delta)$-oblivious. Assume that $n \equiv 0 \pmod 3$. Let $G_1 = (V, E_1)$ be a cycle of length $n$ and $G_2 = (V, E_1)$ consist of $n/3$ disjoint triangles. Clearly, $G_1$ is connected and $G_2$ is $1/3$-far from a connected graph. For a permutation $\pi : V \to V$, define $\pi(G_i) = (V, \pi(E_i))$, where $\pi(E_i) = \{(\pi(u), \pi(v)) : (u, v) \in E_i\}$, and let $\mathrm{perm}(G_i)$ be a random graph isomorphic to $G_i$, that is, $\mathrm{perm}(G_i) = \pi(G_i)$ for a permutation $\pi$ chosen with uniform distribution.[6] On the random graph $\mathrm{perm}(G_)$ Tester has to say "yes" with probability at least $3/4$ and on the random graph $\mathrm{perm}(G_2)$ Tester has to say "no" with probability at least $3/4$.

▶ **Observation 10.** *If* Tester *does not probe two distinct nodes whose distance is at most two, then* Tester *sees a collection of paths of length two and cannot know if the graph is* $\mathrm{perm}(G_1)$ *or* $\mathrm{perm}(G_2)$.

▷ Claim 11. Given the random graph $\mathrm{perm}(G_1)$, the tester has to probe two distinct nodes whose distance is at most 2 with probability at least $1/2$.

Proof. Consider Tester's answer when it sees a collection of paths of length 2. Assume first that the tester returns "No" with probability at least half in this case and let $p$ be the probability that Tester probes two distinct nodes whose distance is at most two on the random graph $\mathrm{perm}(G_1)$. The probability that Tester returns "Yes" on $\mathrm{perm}(G_1)$ is at most $p + 0.5(1 - p) = 0.5 + 0.5p$. Thus, $0.5 + 0.5p \geq 3/4$, i.e., $p \geq 0.5$.

If the tester returns "Yes" with probability at least half, then, by symmetric arguments, with probability at least $1/2$ Tester has to probe two nodes whose distance is at most two on the random graph $\mathrm{perm}(G_2)$. For a permutation $\pi$, if the distance between two nodes in $\pi(G_2)$ is at most 2, then the distance between these two nodes in $\pi(G_1)$ is at most 2. Thus, by Observation 10,

$$\begin{aligned}&\Pr[\text{Tester probes 2 nodes whose distance is 1 or 2 on } \mathrm{perm}(G_1)]\\ &\qquad\geq \Pr[\text{Tester probes 2 nodes whose distance is 1 or 2 } \mathrm{perm}(G_2)] \geq 1/2.\end{aligned}$$ ◁

[6] When we permute a graph, we also permute its incident list representation, i.e., if $(u, v) \in \pi(E)$, then with probability half $v$ will be the first neighbor of $u$ and with probability half it will be the second.

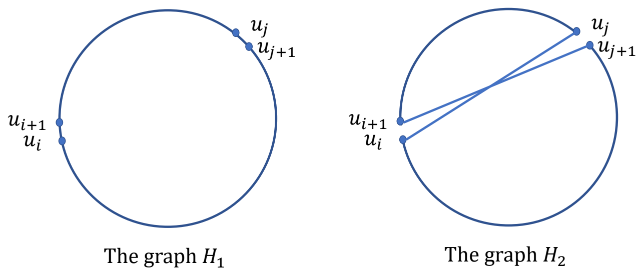

**Figure 1** The graphs $H_1$ and $H_2$.

Denote the nodes of $G_1$ by $V = \{v_0, \dots, v_{n-1}\}$ and define a distribution on pairs of graphs $H_1, H_2$, obtained by the following process:

- Choose a permutation $\pi : V \to V$ with uniform distribution and let $H_1 = \pi(G_1)$.
- Denote $H_1 = (V, E_1)$ and $u_j = \pi(v_j)$ for $j \in [n]$.
- Choose with uniform distribution two indices $i, j$ such that $j \in \{i+4, i+3, \dots, i-3\}$ (where the addition is done modulo $n$).
- Let $H_2 = (V, E_2)$, where $E_2 = E_1 \setminus \{(u_i, u_{i+1}), (u_j, u_{j+1})\} \cup \{(u_i, u_j), (u_{i+1}, u_{j+1})\}$.

The graphs are described in Figure 1. Note that $H_2$ is also a a random graph isomorphic to $G_1$, thus, given $H_2$ one cannot know which pair of non-adjacent nodes $u_i, u_j$ was used to create $H_2$.

Observe that $H_1$ and $H_2$ differ on 4 nodes. Since Tester is $(\varepsilon, \delta)$-differentially oblivious, for every algorithm $\mathcal{A}$,

$$\begin{aligned}\Pr[\mathcal{A}(H_1, &H_2, \mathsf{Access}^{\text{TESTER}}(H_1)) = 1] \\ &\le e^{4\varepsilon} \cdot \Pr[\mathcal{A}(H_1, H_2, \mathsf{Access}^{\text{TESTER}}(H_2)) = 1] + 4e^{4\varepsilon}\delta. \end{aligned} \tag{1}$$

Consider the following algorithm $\mathcal{A}$:

If $u_i$ and at least one of $u_{i+1}, u_{i+2}$ is probed by Tester$(H)$ prior to seeing any other pair of nodes of distance at most 2 in $H_1$ or $H_2$, then return 1 otherwise return 0.

▷ Claim 12. Let $i \in \{1, 2\}$. Suppose that Tester probes at most $q$ nodes. Pick at random with uniform distribution two nodes in $V$ with distance at least 3 in $H_i$. The probability that Tester$(H_i)$ probes both $u$ and $v$ prior to seeing any two nodes of distance at most 2 in $H_i$ is $O(q^2/n^2)$ (where the probability is over the random choice of $u, v$ and the randomness of Tester).

Proof. The node $u$ is a uniformly distributed node in $H_i$ and $v$ is any node of distance at least 3 from $v$, thus there are $n(n-5)/2$ options for $\{u, v\}$. Given a collection of paths of length at most 2 in $H_i$ all options are equally likely.

Let $w_1, \dots, w_k$ be the nodes probed in some execution of Tester. Fix some pair of indices $k_1 < k_2$. The probability that $\{u_i, u_{i+1}\} = \{w_{k_1}, w_{k+2}\}$ is at most $1/n(n-5)$. Thus, the probability that $u$ and $v$ are probed is at most $\frac{\binom{q}{2}}{n(n-5)/2} = O(q^2/n^2)$. ◁

▷ Claim 13. Assume that Tester probes at most $q$ nodes. The probability that $\mathcal{A}(H_1) = 1$ is at least $1/2n - O(q^2/n^2)$.

*Proof.* By Claim 11, the probability that Tester probes at least one pair of nodes with distance at most 2 is at least 1/2. Given that this event occurs, the probability that the random $u_i$ (chosen with uniform distribution) has the smallest index in the first such pair in $H_1$ (i.e., the first pair is either $(u_i, u_{i+1})$ or $(u_i, u_{i+2})$) is at least $1/n$.

Clearly, given these events no two nodes with distance at most 2 in $H_1$ were probed prior to probing the pair containing $u_i$. Furthermore, there are $O(1)$ pairs of nodes that are of distance at most 2 in $H_2$ and are of distance greater than 2 in $H_1$. By Claim 12, the probability that such pair is probed prior to Tester probing a pair of distance at most 2 in $H_1$ is $O(q^2/n^2)$. ◁

▷ Claim 14. Suppose that Tester probes at most $q$ nodes. The probability that $\mathcal{A}(H_2) = 1$ is $O(q^2/n^2)$.

*Proof.* The node $u_i$ is a uniformly distributed node in $H_2$. Furthermore, the nodes $u_{i+1}$ is a uniformly distributed node of distance at least 3 from $u_i$ in $H_2$, thus by Claim 12, the probability that Tester probes both $u_i$ and $u_{i+1}$ prior to seeing any pair of distance at least 2 in $H_2$ is $O(q^2/n^2)$. This probability can only decrease if we require that Tester probes both $u_i$ and $u_{i+1}$ prior to seeing any pair of distance at least 2 in $H_1$ and in $H_2$.

By the same arguments, the probability that Tester probes both $u_i$ and $u_{i+2}$ prior to seeing any pair of distance at least 2 in $H_1$ and in $H_2$ is $O(q^2/n^2)$. ◁

To conclude the proof of Theorem 9, we note that by (1) and Claim 13 and 14

$$\frac{1}{2n} - O(q^2/n) \le \Pr[\mathcal{A}(H_1) = 1] \le e^{4\varepsilon} \Pr[\mathcal{A}(H_2) = 1] + e^{4\varepsilon}\delta \le e^{4\varepsilon} O(q^2/n^2) + e^{4\varepsilon}\delta.$$

Since $e^{4\varepsilon}\delta \le 1/4n$, it follows that $q = \Omega(\sqrt{n}/e^{2\varepsilon})$. ◀

## 5 Differentially Oblivious Algorithm for Locating an Object

Given a dataset of objects $\mathbf{x}$ our goal is to locate an object that satisfies a property $\mathcal{P}$, if one exists. E.g., given a dataset consisting of employee records, find an employee with income in the range \$35,000–\$70,000 if such an employee exists in the dataset.

Absent privacy requirements, a simple approach is to probe elements of the dataset in a random order until an element satisfying the property is found or all elements were probed. If a $p$ fraction of the dataset entries satisfy $\mathcal{P}$ then the expected number of elements sampled by the non-private algorithm is $1/p$. However, a perfectly oblivious algorithm would require $\Omega(n)$ probes on any dataset, in particular on a dataset where all elements satisfy $\mathcal{P}$, where non-privately one probe would suffice. To see why, let $\mathcal{P}(x) = 1$ if $x = 1$ and $\mathcal{P}(x) = 0$ otherwise and let $\mathbf{x}$ include exactly one 1-entry in a uniformly random location. Observe that in expectation it requires $\Omega(n)$ memory probes to locate the 1-entry in $\mathbf{x}$. Perfect obliviousness would hence imply an $\Omega(n)$ probes on any input.

We give a nearly instance optimal differentially oblivious algorithm that always returns a correct answer. Except for probability $e^{-\Omega(mp)}$ the algorithm halts after $m$ steps.

**Our Algorithm.** Given the access pattern of the non-private algorithm, an adversary can learn that the last probed element satisfies $\mathcal{P}$. To hide this information, we change the stopping condition to having probed at least a (randomly chosen) threshold of elements

**Algorithm 3** Differentially Oblivious Locate Algorithm $\mathsf{Locate}_{\mathcal{P}}$.

```
Input: dataset $\mathbf{x} = (x_1, \ldots, x_n)$
1: Let $c \leftarrow 0$, $\varepsilon' = \frac{\varepsilon}{2\log(2/\delta)}$, and $T \leftarrow \frac{1}{\varepsilon'} \ln \frac{\log n}{\delta}$
2: for $i = 1$ to $n/2$ do
3:   Choose $j \in [n]$ with uniform distribution
4:   if $\mathcal{P}(x_j) = 1$ then
5:     $c \leftarrow c + 1$
6:   end if
7:   if $i$ is an integral power of 2 then
8:     $\hat{T} \leftarrow T + \mathrm{Lap}(\frac{1}{\varepsilon'})$
9:     if $c > \max(\hat{T}, 0)$ then
10:      output 1
11:    end if
12:  end if
13: end for
14: Scan the entire dataset, if there is an element satisfying $\mathcal{P}$ then output 1, else output 0
```

satisfying $\mathcal{P}$. If after $n/2$ probes the number of elements satisfying $\mathcal{P}$ is below the threshold the entire dataset is scanned. Our algorithm $\mathsf{Locate}_{\mathcal{P}}$ is described in Algorithm 3. On a given array $\mathbf{x}$, algorithm $\mathsf{Locate}_{\mathcal{P}}$ outputs 1 iff there exists an element in $\mathbf{x}$ satisfying the property $\mathcal{P}$.

We remark that Algorithm $\mathsf{Locate}_{\mathcal{P}}$ uses a mechanism similar to the the sparse vector mechanism of [18]. However, in our case instead of using a single noisy threshold across all steps, Algorithm $\mathsf{Locate}_{\mathcal{P}}$ generates in each step a noisy threshold $\hat{T} = T + \mathrm{Lap}(\frac{1}{\varepsilon'})$. The value of $T$ ensures that with high probability $\hat{T} > 0$. The proof of Theorem 15 is given in Appendix A.3.

▶ **Theorem 15.** *Algorithm $\mathsf{Locate}_{\mathcal{P}}$ is an $(\varepsilon, \delta(1 + e^{\varepsilon}))$-differentially oblivious algorithm that outputs 1 iff there exists an element in the array that satisfies property $\mathcal{P}$. For $m = \Omega(T/p \log(T/p))$, with probability $1 - e^{-\Omega(mp)}$ it halts in time at most $m$, where $T = \frac{2\log(2/\delta)}{\varepsilon} \ln \frac{\log n}{\delta}$.*

## 6 Differentially Oblivious Prefix Sum

Suppose that there is a dataset consisting of sorted sensitive user records, and one would like to compute the sum of all records in the (sorted) dataset that are less than or equal to a value $a$ in a way that respects individual user's privacy. We call this task differentially oblivious prefix sum. For the definition of privacy we say that two datasets of size $n$ are neighbors if they agree on $n - 1$ elements (although, as sorted arrays they can disagree on many indices). For example, $(1, 2, 3, 4)$ and $(1, 3, 4, 5)$ are neighbors and should have similar access pattern.

Without privacy one can find the greatest record less than or equal to value $a$, and then compute the prefix sum by a quick scan through all records appearing before such record. Any perfectly secure algorithm must read the entire dataset (since it is possible that all elements are smaller than $a$). Here, we give a differentially oblivious prefix sum algorithm that for many instances is much faster than any perfectly oblivious algorithm.

**Algorithm 4** Differentially Oblivious Search Algorithm SEARCH.

```
Input: a dataset $\mathbf{x} = (x_1, \dots, x_n)$ and a value $a$
1: Let $\varepsilon' \leftarrow \frac{\varepsilon}{2.5 \log n}$, $\delta' \leftarrow \frac{\delta}{2.5 \log n}$, $k \leftarrow \lceil \frac{4 \log(1/\delta')}{\varepsilon'} \rceil$, $\min \leftarrow 0$, and $\max \leftarrow n$
2: while $\max - \min > k$ do
3:     $c \leftarrow \lfloor (\max - \min)/k \rfloor$
4:     Let $\mathbf{y} = (y_1, \dots, y_k)$, where $y_i = x_{\min + i \cdot c}$ for every $i \in [k]$
5:     Scan the entire dataset $\mathbf{y}$ and find the maximal index $I$ such that $y_I \leq a$; if there is
       no such element then $I \leftarrow 0$
6:     noise $\leftarrow \mathrm{Lap}(\frac{1}{\varepsilon'})$
7:     $\min = \max\{0, \min + \lfloor (I + \text{noise} - \frac{\log 1/\delta'}{\varepsilon'}) \cdot c \rfloor\}$ and $\max = \min\{n, \min + \lfloor (I + \text{noise} +$
       $\frac{\log 1/\delta'}{\varepsilon'} + 1) \cdot c \rfloor\}$
8: end while
9: Scan the entire dataset $\mathbf{x}$ between min and max and return the the maximal index $I$
   such that $x_I \leq a$; if there is no such element then $I \leftarrow 0$
```

**Intuition.** Absent privacy requirements, using binary search, one can find the greatest element less than or equal to $a$, and then compute the prefix sum by a quick scan through all records that appear before such record. However, the binary search access pattern allows the adversary to gain sensitive information about the input. Our main idea is to approximately simulate the binary search and obfuscate the memory accesses to obtain differential obliviousness. In order to do that, we first divide the input array into $k$ chunks (where $k$ is polynomial in $1/\varepsilon, \log 1/\delta$, and $\log n$). Then, we find the chunk that contains the greatest element less than or equal to $a$ by comparing the first element (hence, the smallest element) of each chunk to $a$. Let $I$ be the index of such chunk. Next, we compute a noisy interval that contains $I$ using the Laplacian distribution. We iteratively repeat this process on the noisy interval, where in each step we eliminate more than a quarter of the elements of the interval. We continue until the size of the array is less than or equal to $k$. Next, we scan all elements in the remaining array and find the index of the greatest element smaller than or equal to $a$. Let $i$ be the index of such element; we compute the prefix sum by scanning the array $\mathbf{x}$ until index $i$.

**The Search Algorithm.** We present a search algorithm in Algorithm 4; on input $\mathbf{x} = (x_1, \dots, x_n)$ and $a$ this algorithm finds the largest index $I$ such that $x_I \leq a$. To compute the prefix sum, we compute $\hat{I} = I + \mathrm{Lap}(1/\varepsilon) + \frac{\log 1/\delta}{\varepsilon}$ and scan the first $\hat{I}$ elements of the dataset, summing only the first $I$. We show in Theorem 17 that our search algorithm is $(\varepsilon, \delta)$-differentially oblivious.

▶ Remark 16. We prove that algorithm SEARCH is an $(\varepsilon, 0)$-differentially private algorithm that returns a correct index with probability at least $1 - \beta$. We could change it to an $(\varepsilon, \delta)$-differentially private algorithm that never errs. This is done by truncating the noise to $\frac{\log 1/\delta'}{\varepsilon'}$.

▶ **Theorem 17.** *Let $\beta < 1/n$ and $\varepsilon < \log^2 n$. Algorithm* SEARCH *is an $(\varepsilon, 0)$-differentially oblivious algorithm that, for any input array with size $n$ and $a \in \mathbb{R}$, returns a correct index with probability at least $1 - \beta$. The running time of Algorithm* SEARCH *is $O(\frac{1}{\varepsilon} \log^2 n \log \frac{1}{\beta})$.*

Theorem 17 is proved in Appendix A.4.

**Algorithm 5** Differentially Oblivious Search Algorithm MULTISEARCH for Multiple Queries.

```
Input: a dataset $\mathbf{x} = (x_1, \dots, x_n)$
1: $t \leftarrow 1$ and $M \leftarrow 0$
2: for every query $a$ do
3:     if the greatest element in the ORAM is greater than $a$ or all records are in the ORAM (that is $M = n$) then
4:         answer the query using the ORAM
5:     else
6:         execute algorithm SEARCH with privacy parameter $\frac{\varepsilon}{t \log n}$ and accuracy parameter $\beta/\sqrt{n}$ for the database starting at record $M+1$ and let $I$ the largest index in this database such that $x_I \leq a$
7:         insert the first $\max\{I, 2t\}$ elements of this database to the ORAM; for each element also insert the sum of all elements in the array up to this element
8:         $t \leftarrow t+1$, $M \leftarrow M + \max\{I, 2t\}$
9:     end if
10: end for
```

## 6.1 Dealing with Multiple Queries

We extend our prefix sum algorithm to answer multiple queries. We can answer a bounded number of queries by running the differentially oblivious prefix sum algorithm multiple times. That is, when we want an $(\varepsilon, 0)$-oblivious algorithm correctly answering $t$ queries with probability at least $1 - \beta$, we execute algorithm SEARCH $t$ times with privacy parameter $\varepsilon/t$ and error probability $\beta/t$ (each time also computing the appropriate prefix sum). Thus, the running time of the algorithm is $O(\frac{t^2}{\varepsilon} \log^2 n \log \frac{t}{\beta})$ (excluding the scan time for computing the sum).

On the other hand, we can use an ORAM to answer unbounded number of queries. That is, in a pre-processing stage we store the $n$ records and for each record we store the sum of all records up to this record. Thereafter, answering each query will require one binary search. Using the ORAM of [2], the pre-processing will take time $O(n \log n)$ and answering each query will take time $O(\log^2 n)$. Thus, the ORAM algorithm is more efficient when $t \geq \sqrt{n}$.

We use ORAM along with our differentially oblivious prefix sum algorithm to answer unbounded number of queries while preserving privacy, combining the advantages of both of the previous algorithms.

▶ **Theorem 18.** *Algorithm* MULTISEARCH*, described in Algorithm 5, is an $(\varepsilon, 0)$-oblivious algorithm, which executes Algorithm* SEARCH *at most $O(\sqrt{n})$ times, where the run time of the $t$-th execution is $O(\frac{t}{\varepsilon} \log^3 n \log \frac{n}{\beta})$, scans the original database at most once, and in addition each query run time is at most $O(\log^2 n)$.*

**Proof.** First note that we only pay for privacy in the executions of algorithm SEARCH (reading and writing to the ORAM is perfectly private). In the $t$-th execution of algorithm SEARCH, we insert at least $2t$ elements to the ORAM, thus after $\sqrt{n}$ executions we inserted at least $\sum_{t=1}^{\sqrt{n}} 2t = n$ elements to the ORAM.

By simple composition, algorithm MULTISEARCH is $(\varepsilon', 0)$-differentially private, where

$$\varepsilon' = \sum_{t=1}^{\sqrt{n}} \frac{\varepsilon}{t \log n} \leq \frac{\varepsilon}{\log n} (\ln \sqrt{n} + 1) \leq \varepsilon,$$

where the last inequality is implied by the sum of the harmonic series. ◀

## References

1 Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. Efficient Testing of Large Graphs. *Combinatorica*, 20(4):451–476, 2000. doi:10.1007/s004930070001.

2 Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, and Elaine Shi. OptORAMa: Optimal Oblivious RAM. *IACR Cryptology ePrint Archive*, 2018:892, 2018. URL: https://eprint.iacr.org/2018/892.

3 Marina Blanton, Aaron Steele, and Mehrdad Aliasgari. Data-oblivious graph algorithms for secure computation and outsourcing. In Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng, editors, *8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13*, pages 207–218. ACM, 2013. doi:10.1145/2484313.2484341.

4 David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015*, pages 668–679. ACM, 2015.

5 T.-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. Foundations of Differentially Oblivious Algorithms. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2448–2467. SIAM, 2019.

6 Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion ORAM: A Constant Bandwidth Blowup Oblivious RAM. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A*, volume 9563 of *Lecture Notes in Computer Science*, pages 145–174. Springer, 2016. doi:10.1007/978-3-662-49099-0_6.

7 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

8 David Eppstein, Michael T. Goodrich, and Roberto Tamassia. Privacy-preserving data-oblivious geometric algorithms for geographic data. In Divyakant Agrawal, Pusheng Zhang, Amr El Abbadi, and Mohamed F. Mokbel, editors, *18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010*, pages 13–22. ACM, 2010. doi:10.1145/1869790.1869796.

9 Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In Peter Buneman, editor, *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 102–113. ACM, 2001.

10 Oded Goldreich. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 182–194. ACM, 1987. doi:10.1145/28395.28416.

11 Oded Goldreich. *Introduction to Property Testing.* Cambridge University Press, 2017. doi:10.1017/9781108135252.

12 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property Testing and its Connection to Learning and Approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

13 Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3):431–473, 1996. doi:10.1145/233551.233553.

14 Oded Goldreich and Dana Ron. Property Testing in Bounded Degree Graphs. *Algorithmica*, 32(2):302–343, 2002.

15 M. T. Goodrich, O. Ohrimenko, and R. Tamassia. Data-oblivious graph drawing model and algorithms. *CoRR*, 2012.

16 Michael T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014*, pages 684–693. ACM, 2014. doi:10.1145/2591796.2591830.

17 Michael T. Goodrich and Joseph A. Simons. Data-Oblivious Graph Algorithms in Outsourced External Memory. In Zhao Zhang, Lidong Wu, Wen Xu, and Ding-Zhu Du, editors, *Combinatorial Optimization and Applications - 8th International Conference, COCOA 2014*, volume 8881 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2014. doi:10.1007/978-3-319-12691-3_19.

18 Moritz Hardt and Guy N. Rothblum. A Multiplicative Weights Mechanism for Privacy-Preserving Data Analysis. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010*, pages 61–70. IEEE Computer Society, 2010.

19 Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. Composing Differential Privacy and Secure Computation: A Case Study on Scaling Private Record Linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1389–1406. ACM, 2017. doi:10.1145/3133956.3134030.

20 Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Inference attack against encrypted range queries on outsourced databases. In Elisa Bertino, Ravi S. Sandhu, and Jaehong Park, editors, *Fourth ACM Conference on Data and Application Security and Privacy, CODASPY'14*, pages 235–246. ACM, 2014. doi:10.1145/2557547.2557561.

21 Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic Attacks on Secure Outsourced Databases. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340. ACM, 2016. doi:10.1145/2976749.2978386.

22 Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Accessing Data while Preserving Privacy. *CoRR*, abs/1706.01552, 2017. arXiv:1706.01552.

23 Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved Reconstruction Attacks on Encrypted Data Using Range Query Leakage. In *2018 IEEE Symposium on Security and Privacy, SP 2018*, pages 297–314. IEEE Computer Society, 2018. doi:10.1109/SP.2018.00002.

24 Kasper Green Larsen and Jesper Buus Nielsen. Yes, There is an Oblivious RAM Lower Bound! In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*, volume 10992 of *Lecture Notes in Computer Science*, pages 523–542. Springer, 2018. doi:10.1007/978-3-319-96881-0_18.

25 Wei-Kai Lin, Elaine Shi, and Tiancheng Xie. Can We Overcome the $n \log n$ Barrier for Oblivious Sorting? In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2419–2438. SIAM, 2019. doi:10.1137/1.9781611975482.148.

26 Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A Programming Framework for Secure Computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 359–376. IEEE Computer Society, 2015.

27 Sahar Mazloom and S. Dov Gordon. Secure Computation with Differentially Private Access Patterns. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 490–507. ACM, 2018. doi:10.1145/3243734.3243851.

28 Tarik Moataz, Travis Mayberry, and Erik-Oliver Blass. Constant Communication ORAM with Small Blocksize. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 862–873. ACM, 2015.

29 Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference Attacks on Property-Preserving Encrypted Databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security 2015*, pages 644–655. ACM, 2015.

30 Rafail Ostrovsky. Efficient Computation on Oblivious RAMs. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 514–523. ACM, 1990. doi:10.1145/100216.100289.

31 Giuseppe Persiano and Kevin Yeo. Lower Bounds for Differentially Private RAMs. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 404–434. Springer, 2019. doi:10.1007/978-3-030-17653-2_14.

32 Sofya Raskhodnikova and Adam D. Smith. A Note on Adaptivity in Testing Properties of Bounded Degree Graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(089), 2006. URL: http://eccc.hpi-web.de/eccc-reports/2006/TR06-089/index.html.

## A Missing Proofs

### A.1 Proof of Lemma 5

**Proof.** Let $\mathbf{x}$ and $\mathbf{y}$ be two neighboring datasets and $S$ be a sets of outputs. Then,

$$\begin{aligned}\Pr[\mathcal{B}(\mathbf{x}) \in S] &\le \Pr[\mathcal{A}(\mathbf{x}) \in S] + \gamma\\ &\le e^{\varepsilon} \Pr[\mathcal{A}(\mathbf{y}) \in S] + \gamma\\ &\le e^{\varepsilon}(\Pr[\mathcal{B}(\mathbf{y}) \in S] + \gamma) + \gamma\\ &= e^{\varepsilon} \Pr[\mathcal{B}(\mathbf{y}) \in S] + (1 + e^{\varepsilon})\gamma.\end{aligned}$$ ◀

### A.2 Proof of the Correctness and Privacy of Algorithm Tester$_{\mathcal{T}}$

Theorem 8 is implied by the following lemmas.

▶ **Lemma 19.** *Algorithm Tester$_{\mathcal{T}}$ is $(\varepsilon, \delta(1 + e^{\varepsilon}))$-differentially oblivious.*

**Proof.** We first analyze a variant of Tester$_{\mathcal{T}}$, denoted by $\textsc{Tester}'_{\mathcal{T}}$, in which Step 10 is replaced by "If $c > \hat{T}$ then output 1" (that is, the algorithm does not check if $c > \min\{4T, \hat{T}\}$ before deciding in the positive).

Let $G = (V, E)$ and $G' = (V', E')$ be two neighboring graphs such that they differ on node $v \in V$. Fix the random choices of subsets $A$ in Step 6 and observe that after the execution of for loop, the count $c$ can differ by at most 1 between the executions on $G$ and $G'$. Let $\tilde{T}$ be the smallest integer greater than $\hat{T}$. Since algorithm $\textsc{Tester}'_{\mathcal{T}}$ uses the Laplace mechanism $e^{-\varepsilon} \Pr[\tilde{T} < a] \le \Pr[\tilde{T} < a - 1] \le e^{\varepsilon} \Pr[\tilde{T} < a]$ for every $a$. Thus,

$$\begin{aligned}\Pr[\textsc{Tester}'_{\mathcal{T}}(G) = 1] &= \sum_a \Pr[\tilde{T} = a] \Pr[c(G) > a]\\ &\le \sum_a \Pr[\tilde{T} = a] \Pr[c(G') > a - 1]\\ &\le e^{\varepsilon} \sum_a \Pr[\tilde{T} = a - 1] \Pr[c(G') > a - 1]\\ &\le e^{\varepsilon} \Pr[\textsc{Tester}'_{\mathcal{T}}(G') = 1].\end{aligned}$$

Similarly, $\Pr[\textsc{Tester}'_{\mathcal{T}}(G) = 1] \ge e^{-\varepsilon} \Pr[\textsc{Tester}'_{\mathcal{T}}(G') = 1]$. Hence, $\textsc{Tester}'_{\mathcal{T}}$ is $(\varepsilon, 0)$-differentially oblivious.

We next prove that Tester$_{\mathcal{T}}$ is $(\varepsilon, \delta(1 + e^{\varepsilon}))$-differentially oblivious using Lemma 5, that is we prove that for every graph $G$, the statistical distance between Tester$_{\mathcal{T}}(G)$ and $\textsc{Tester}'_{\mathcal{T}}(G)$ is at most $\delta$. Let $E$ be the event that $\hat{T} > 4T$ and observe that the probability

$E$ occurs is at most $\delta$.[7] We have that $\left|\Pr[\text{Tester}_{\mathcal{T}}(G)=1]-\Pr[\text{TESTER}'_{\mathcal{T}}(G)=1]\right| \leq \left|\Pr[\text{Tester}_{\mathcal{T}}(G)=1|E]-\Pr[\text{TESTER}'_{\mathcal{T}}(G)=1|E]\right|\Pr[E] \leq \Pr[E] \leq \delta$. Thus, by Lemma 5, algorithm $\text{Tester}_{\mathcal{T}}$ is $(\varepsilon, \delta(1+e^{\varepsilon}))$-differentially oblivious. ◀

Observe that Algorithm $\text{Tester}_{\mathcal{T}}$ never errs when $G \in \mathcal{P}$ as in that case after the for loop is executed $c = 4T$ and hence in Step 10 $\text{Tester}_{\mathcal{T}}$ outputs 1. The next lemma analyses the error probability when $G$ is $\gamma'$-far from $\mathcal{P}$.

▶ **Lemma 20.** *Algorithm $\text{Tester}_{\mathcal{T}}$ is $(1-\delta-(2\delta)^{\frac{1}{3\varepsilon}}, \gamma')$-tester for the graph property $\mathcal{P}$.*

**Proof.** Observe that on Step 7 of the algorithm, we are eliminating at most $n \cdot c_{\beta,\gamma}$ edges. Thus, we are eliminating at most $4Tnc_{\beta,\gamma}$ edges in total. Then, when $G$ is $\gamma'$-far from $\mathcal{P}$, it is also $\gamma$-far from $\mathcal{P}$ after the removal of the observed nodes in each step of the for loop. We next prove that Algorithm $\text{Tester}_{\mathcal{T}}$ fails with probability at most $2\delta^{\frac{1}{3\varepsilon}}$. Observe that if Algorithm $\text{Tester}_{\mathcal{T}}$ fails on $G$ then $c \geq 2T$ or $\text{Lap}(\frac{1}{\varepsilon}) \leq -T$. We define $Z_i$ to be output of $\mathcal{T}(G)$ in the $i$-th step of the for loop. Let $Z = \sum_i Z_i$. Observe that all $Z_i$ are independent and $\mathbb{E}[Z] \leq T$. Using the Chernoff Bounds[8], we obtain that $\Pr[Z \geq 2T] \leq e^{-T/3} = (2\delta)^{\frac{1}{3\varepsilon}}$. We also know $\Pr[\text{Lap}(\frac{1}{\varepsilon}) \leq -\frac{\ln(1/2\delta)}{\varepsilon}] = 0.5e^{-\ln(1/2\delta)} = \delta$. Therefore, Algorithm $\text{Tester}_{\mathcal{T}}$ fails with probability $\delta + (2\delta)^{\frac{1}{3\varepsilon}}$. ◀

## A.3 Proof of the Correctness and Privacy of Algorithm $\text{Locate}_{\mathcal{P}}$

The proof of Theorem 15 follows from the following claim and lemmas.

▷ Claim 21. Let $\ell \geq \log n/\log\log n$. The probability that there exists an element $j \in [n]$ such that algorithm $\text{Locate}_{\mathcal{P}}$ samples the element $j$ in Step 3 more than $2\ell$ times is less that $2^{-\ell}$.

Proof. Fix an index $j$. The probability that the element $j$ is sampled more than $2\ell$ times is less than $\binom{n/2}{2\ell}\frac{1}{n^{2\ell}} \leq \left(\frac{en}{4\ell}\right)^{2\ell}\frac{1}{n^{2\ell}} < \ell^{-2\ell} < 2^{-2\log n+2} < 2^{2-\ell}/n$. The claim follows by the union bound. ◁

▶ **Lemma 22.** *Let $\delta < 1/n$. Algorithm $\text{Locate}_{\mathcal{P}}$ is $(\varepsilon, \delta(1+e^{\varepsilon}))$-differentially oblivious.*

**Proof.** We first analyze a variant of $\text{Locate}_{\mathcal{P}}$, denoted by $\text{LOCATE}'_{\mathcal{P}}$, in which Step 9 is replaced by "If $c > \hat{T}$ then output 1" (that is, the algorithm does not check if $\hat{T} > 0$) and no element is sampled more than $2\log(2/\delta)$ times. We analyze the privacy of $\text{LOCATE}'_{\mathcal{P}}(\mathbf{x}')$ similarly to the analysis of the sparse vector mechanism in [18].

Let $\mathbf{x}$ and $\mathbf{x}'$ be two neighboring datasets that such that $\mathcal{P}(x_j) = 1$ and $\mathcal{P}(x'_j) = 0$ for some $j$. Denote by $\tau = (\tilde{T}_1, \ldots, \tilde{T}_{\log n})$ the values of the thresholds in an execution of $\text{LOCATE}'_{\mathcal{P}}$, where each threshold is rounded up to the smallest integer greater than $\hat{T}$. Furthermore, let $\ell_\tau \in [\log n]$ be the index such that $\text{LOCATE}'_{\mathcal{P}}$ on input $\mathbf{x}$ outputs 1 when $i = 2^{\ell_\tau}$ (if no such $i$ exists, then $\ell_\tau \in \lceil \log n \rceil + 1$). Observe that in each execution of Step 9 the count $c$ on input $\mathbf{x}$ is at least the count on input $\mathbf{x}'$ and can exceed it by at most $2\log(2/\delta)$ (since $j$ is sampled at most $2\log(2/\delta)$ times). Thus, $\text{LOCATE}'_{\mathcal{P}}$ on input $\mathbf{x}'$ with thresholds $\tau' = (\tilde{T}_1, \ldots, \tilde{T}_{\ell_\tau - 1}, \tilde{T}_{\ell_\tau} - 2\log(2/\delta), \tilde{T}_{\ell_\tau+1}, \ldots, \tilde{T}_{\log n})$ outputs 1 when $i = 2^{\ell_\tau}$. Since algorithm $\text{LOCATE}'_{\mathcal{P}}$ uses the Laplace mechanism with $\varepsilon' = \varepsilon/(2\log(1/\delta))$,

$$e^{-\varepsilon}\Pr[\tilde{T}_{\ell_\tau} = a] \leq \Pr[\tilde{T}_{\ell_\tau} = a - 2\log(2/\delta)] \leq e^{\varepsilon}\Pr[\tilde{T}_{\ell_\tau} = a]$$

[7] $\Pr[\text{Lap}(\frac{1}{\varepsilon}) \geq t/\varepsilon] = \frac{1}{2}e^{-t}$ for every $t > 0$. Thus, $\Pr[E] = \Pr[\text{Lap}(\frac{1}{\varepsilon}) \geq \frac{\ln(1/2\delta)}{\varepsilon}] = \delta$.

[8] $\Pr[Z \geq (1+\eta)\mu] \leq e^{-\eta^2\mu/(2+\eta)}$ for any $\eta > 0$ where $\mu$ is the expectation of $Z$.

for every $a$. Thus,

$$\begin{aligned}
&\Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S] \\
&\quad = \sum_{\tau=(\tilde{T}_1,\ldots,\tilde{T}_{\log n})} \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S \mid \tilde{T}_1,\ldots,\tilde{T}_{\log n}] \Pr[\tilde{T}_1,\ldots,\tilde{T}_{\log n}] \\
&\quad = \sum \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}') \in S \mid \tilde{T}_1,\ldots,\tilde{T}_{\ell_\tau-1}, \tilde{T}_{\ell_\tau} - 2\log(2/\delta), \tilde{T}_{\ell_\tau+1},\ldots,\tilde{T}_{\log n}] \\
&\qquad\qquad \cdot \Pr[\tilde{T}_1,\ldots,\tilde{T}_{\log n}] \\
&\quad \le e^{\varepsilon} \sum \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}') \in S \mid \tilde{T}_1,\ldots,\tilde{T}_{\ell_\tau-1}, \tilde{T}_{\ell_\tau} - 2\log(2/\delta), \tilde{T}_{\ell_\tau+1},\ldots,\tilde{T}_{\log n}] \\
&\qquad\qquad \cdot \Pr[\tilde{T}_1,\ldots,\tilde{T}_{\ell_\tau-1}, \tilde{T}_{\ell_\tau} - 2\log(2/\delta), \tilde{T}_{\ell_\tau+1},\ldots,\tilde{T}_{\log n}] \\
&\quad = e^{\varepsilon} \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}') \in S].
\end{aligned}$$

Similarly,

$$\begin{aligned}
&\Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S] \\
&\quad \ge e^{-\varepsilon} \sum \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}') \in S \mid \tilde{T}_1,\ldots,\tilde{T}_{\ell_\tau-1}, \tilde{T}_{\ell_\tau} - 2\log(2/\delta), \tilde{T}_{\ell_\tau+1},\ldots,\tilde{T}_{\log n}] \\
&\qquad\qquad \cdot \Pr[\tilde{T}_1,\ldots,\tilde{T}_{\ell_\tau-1}, \tilde{T}_{\ell_\tau} - 2\log(2/\delta), \tilde{T}_{\ell_\tau+1},\ldots,\tilde{T}_{\log n}] \\
&\quad = e^{-\varepsilon} \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}') \in S].
\end{aligned}$$

We next prove that $\text{Locate}_{\mathcal{P}}$ is $(\varepsilon, \delta(1+e^{\varepsilon}))$-differentially oblivious using Lemma 5. I.e, we prove that for every dataset $\mathbf{x}$, the statistical distance between $\mathsf{Access}^{\text{Locate}_{\mathcal{P}}}(\mathbf{x})$ and $\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x})$ is at most $\delta$. Notice that if all the thresholds are positive and all elements are sampled at most $2\log(2/\delta)$ times then $\text{Locate}_{\mathcal{P}}(\mathbf{x})$ and $\textsc{Locate}'_{\mathcal{P}}(\mathbf{x})$ have the same access pattern. By Claim 21, the probability that there exists a $j$ that is sampled more than $2\log(2/\delta)$ is at $2^{-\log(2/\delta)} = \delta/2$. We next observe that the probability that a threshold $\hat{T} = T + \text{Lap}(\frac{1}{\varepsilon'})$ is negative is at most $\delta/2$. Recall that $\Pr[\text{Lap}(\frac{1}{\varepsilon'}) \le -t/\varepsilon'] = \frac{1}{2}e^{-t}$ for every $t > 0$. Thus, $\Pr[\hat{T} \le 0] = \Pr[\text{Lap}(\frac{1}{\varepsilon'}) \le -\frac{1}{\varepsilon}\ln(\frac{\log n}{\delta})] = \frac{\delta}{2\log n}$. Let $A$ be the event that at least one of the $\log n$ thresholds $\hat{T}$ is at most 0 or some $j$ is sampled more that $2\log(2/\delta)$ times. By the union bound the probability of $A$ is at most $\delta$. Therefore, for every set of access patterns $S$

$$\begin{aligned}
&|\Pr[\mathsf{Access}^{\text{Locate}_{\mathcal{P}}}(\mathbf{x}) \in S] - \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S]| \\
&\quad = \Big| \Pr[\mathsf{Access}^{\text{Locate}_{\mathcal{P}}}(\mathbf{x}) \in S|A] \Pr[A] + \Pr[\mathsf{Access}^{\text{Locate}_{\mathcal{P}}}(\mathbf{x}) \in S|\bar{A}] \Pr[\bar{A}] \\
&\qquad - \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S|A] \Pr[A] - \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S|\bar{A}] \Pr[\bar{A}] \Big| \\
&\quad = \Big| \Pr[\mathsf{Access}^{\text{Locate}_{\mathcal{P}}}(\mathbf{x}) \in S|A] - \Pr[\mathsf{Access}^{\textsc{Locate}'_{\mathcal{P}}}(\mathbf{x}) \in S|A] \Big| \Pr[A] \\
&\quad \le \Pr[A] \le \delta.
\end{aligned}$$

Thus, by Lemma 5, algorithm $\text{Locate}_{\mathcal{P}}$ is $(\varepsilon, \delta(1+e^{\varepsilon}))$-differentially oblivious. ◀

We next analyze the running and probe complexity of our algorithm. Let $p$ be the probability that a uniformly chosen element in $\mathbf{x}$ satisfies $\mathcal{P}$. The non-private algorithm that samples elements until it finds an element satisfying $\mathcal{P}$ has expected running time $1/p$ and the probability that it does not stop after $m$ steps is $(1-p)^m = ((1-p)^{1/p})^{mp} \le e^{-mp}$. We show that $\text{locate}_{\mathcal{P}}$ has a similar behavior.

▶ **Lemma 23.** *Let $p$ be the probability that a uniformly chosen element in $\mathbf{x}$ satisfies $\mathcal{P}$. Then, for every integral power of two $m$ the probability that algorithm locate$_{\mathcal{P}}$ probes more than $m$ memory locations is less than $\delta/\log n + e^{-(m-2T)p+2T\ln m}$. In particular, for $m = \Omega(\frac{T}{p}\log(\frac{T}{p}))$, the probability is less than $\delta/\log n + e^{-O(mp)}$.*

**Proof.** Let $t = 2^i$. The probability that $\hat{T} \geq 2T$ is $\Pr[\mathrm{Lap}(\frac{1}{\varepsilon'}) \geq \frac{1}{\varepsilon'}\ln\frac{\log n}{\delta}] = 0.5e^{-\ln(\log n/\delta)} = \frac{\delta}{\log n}$. Assuming that $\hat{T} \geq 2T$, the probability that the algorithm does not halt after $m = 2^i$ steps is less than

$$\binom{m}{2T}(1-p)^{m-2t} \leq m^{2T}e^{-(m-2T)p} \leq e^{-(m-2T)p+2T\ln m}.$$ ◀

## A.4 Proof of the Correctness and Privacy of Algorithm Search

Theorem 17 is proved in the next 3 claims. We start by analyzing the running time of the algorithm.

▷ Claim 24. Let $\beta < 1/n$ and $\varepsilon < \log^2 n$. The while loop in Algorithm Search is executed at most $2.5\log n$ time. Furthermore, the total running time of the algorithm is $O(\frac{1}{\varepsilon}\log^2 n\log\frac{1}{\beta})$.

Proof. Let $\min_0, \max_0$ and $\min_1, \max_1$ be the values of $\min, \max$ before and after an execution of a step of the while loop in Algorithm Search. Note that

$$\max_1 - \min_1 \leq 1 + (2\cdot\frac{\log 1/\beta'}{\varepsilon'} + 1)\cdot\frac{\max_0 - \min_0}{\frac{4\log(1/\beta')}{\varepsilon'}} \leq 3\cdot\frac{\log 1/\beta'}{\varepsilon'}\cdot\frac{\max_0 - \min_0}{\frac{4\log(1/\beta')}{\varepsilon'}} = \frac{3(\max_0 - \min_0)}{4}.$$

Therefore, algorithm Search eliminates more than a quarter of the elements in each step of the while loop and the algorithm will halt after less than $2.5\log n$ steps.

Moreover, observe that Algorithm Search makes $k$ memory accesses in each step of the while loop and additional $k$ memory accesses after the loop. Thus, its running time is $O(\frac{1}{\varepsilon}\log^2 n(\log\log n + \log\frac{1}{\beta})) = O(\frac{1}{\varepsilon}\log^2 n\log\frac{1}{\beta})$ (since $\beta < 1/n$). ◁

▷ Claim 25. Algorithm Search returns the correct index with probability at least $1-\beta$.

Proof. Let $\bar{I}$ be the maximal index such that $x_{\bar{I}} \leq a$ (i.e., $\bar{I}$ is the index that algorithm Search should return). We prove by induction that if all Laplace noises in the algorithm satisfy $|\mathrm{Lap}(\frac{1}{\varepsilon'})| < \frac{\log 1/\beta'}{\varepsilon'}$ then in each step of the algorithm $\min \leq \bar{I} \leq \max$, hence the algorithm will return $\bar{I}$ in its last scan of $\mathbf{x}$ between min and max.

The basis of the induction is trivial since $0 \leq \bar{I} \leq n$. For the induction step, let $\min_0, \max_0$ and $\min_1, \max_1$ be the values of $\min, \max$ before and after an execution of a step of the while loop in Algorithm Search. By the induction hypothesis, $\min_0 \leq \bar{I} \leq \max_0$. The algorithm finds an index $I$ such that $\min_0 + Ic \leq \bar{I} \leq \min_0 + (I+1)c$. By our assumption on the Laplace noise, $\min_1 \leq \min_0 + Ic$, thus, $\min_1 \leq \bar{I}$. Similarly, $\max_1 \geq \min_0 + (I+1)c$, thus, $\max_1 \geq \bar{I}$.

Recall that $\Pr[|\mathrm{Lap}(\frac{1}{\varepsilon'})| \geq t/\varepsilon'] = e^{-t}$ for every $t > 0$. Thus, by Claim 24 and the union bound, the probability that one of the Laplace noises is greater than $\frac{\log 1/\beta'}{\varepsilon'}$ is at most $(2.5\log n)\cdot\beta' = \beta$. Hence, the probability that algorithm Search returns the correct index $\bar{I}$ is at least $1-\beta$. ◁

Next, we show that algorithm Search is $(\varepsilon, 0)$-differentially oblivious.

▷ Claim 26. Algorithm Search is an $(\varepsilon, 0)$-differentially oblivious algorithm.

Proof. We show below that each step of the while loop in algorithm Search is $(\varepsilon', 0)$-differentially oblivious. Applying the basic composition theorem and Claim 24, we obtain that the Search algorithm is $(\varepsilon = (2.5 \log n)\varepsilon', 0)$-differentially oblivious.

Fix a step of the loop and view it as an algorithm that returns min and max (given these values the access pattern of the next step is fixed). Let $\mathbf{x}$ and $\mathbf{x}'$ be two neighboring datasets such that for some $j$ we have $x_j > x'_j$ and $x_i = x'_i$ for all $i < j$. It holds that $x_{i-1} \leq x'_i \leq x_i$ for every $i$. Let $I(\mathbf{x})$ and $I(\mathbf{x}')$ be the values computed in step 5 of the algorithm on inputs $\mathbf{x}$ and $\mathbf{x}'$ respectively. Thus, the value $I(\mathbf{x})$ is at least the value $I(\mathbf{x}')$ and can exceed it by one. Intuitively, since algorithm Search uses the Laplace mechanism, the probabilities of returning a value min on $\mathbf{x}$ and $\mathbf{x}'$ are at most $e^{\pm\varepsilon'}$ apart. Formally, if $\mathrm{Lap}(1/\varepsilon') + I(\mathbf{x}) = \mathrm{Lap}(1/\varepsilon') + I(\mathbf{x})$ (where we consider two independent noises), then the algorithm returns the same value of min on both inputs. The lemma follows since for every set $A$:

$$e^{-\varepsilon'} \leq e^{-|I(\mathbf{x})-I(\mathbf{x}')|\varepsilon'} \leq \frac{\Pr[\mathrm{Lap}(1/\varepsilon') + I(\mathbf{x}) \in A]}{\Pr[\mathrm{Lap}(1/\varepsilon') + I(\mathbf{x}') \in A]} \leq e^{|I(\mathbf{x})-I(\mathbf{x}')|\varepsilon'} \leq e^{\varepsilon'}. \qquad \triangleleft$$