Electronic Colloquium on Computational Complexity, Report No. 40 (2017)

Non-Adaptive Data Structure Lower Bounds for Median and Predecessor Search from Sunflowers

Sivaramakrishnan Natarajan Ramamoorthy* Anup Rao†

March 4, 2017

Abstract

We prove new cell-probe lower bounds for data structures that maintain a subset of $\{1, 2, ..., n\}$, and compute the median of the set. The data structure is said to handle insertions *non-adaptively* if the locations of memory accessed depend only on the element being inserted, and not on the contents of the memory. We prove that any such data structure must satisfy:

$$t_m \geq \Omega\left(\frac{n^{\frac{1}{2(t_i+1)}}}{w \cdot t_i}\right),$$

where t_i is the number of memory locations accessed during insertions, t_m is the number of memory locations accessed to compute the median, and w is the number of bits stored in each memory location. Our lower bounds are nearly matched by Binary Search Trees.

For the predecessor search problem, where the algorithm is required to compute the predecessor of any element in the set, we prove that if computing the predecessors can be done non-adaptively, then

$$t_p \geq \Omega\left(\frac{\log n}{\log\log n + \log w}\right) \qquad \text{or} \qquad t_i \geq \Omega\left(\frac{t_p \cdot n^{\frac{1}{2(t_p+1)}}}{\log n}\right),$$

were t_p is the number of locations accessed to compute predecessors. Again, these bounds prove that Binary Search Trees have essentially optimal parameters for the predecessor search problem.

Our results follow from a novel application of the Sunflower Lemma of Erdős and Rado [ER60] to these questions.

1 Introduction

Data structures are algorithmic primitives to efficiently manage data. They are used widely in computer systems, and not just to maintain large data sets; these primitives play a fundamental role in many algorithmic tasks. For example, the *heap* data structure is a crucial component of the best algorithms for computing shortest paths in weighted graphs, and the *union-find* data structure

*Computer Science and Engineering, University of Washington, sivanr@cs.washington.edu. Supported by the National Science Foundation under agreement CCF-1420268 and CCF-1524251

†Computer Science and Engineering, University of Washington, anuprao@cs.washington.edu. Supported by the National Science Foundation under agreement CCF-1420268 and CCF-1524251.

ISSN 1433-8092

is vital to algorithms for computing minimum spanning trees in graphs. In both of these examples, the running times of these algorithms depend on the performance of the underlying data structures. In this paper, we study data structures that maintain a set of numbers S and allow for quickly computing the *median* of the set or *predecessors* of the set. The median is the middle number of the set in sorted order, and the predecessor of a number x is the largest element in S that is at most x. We give new lower bounds on data structures computing the median and predecessors.

The performance of data structures is usually measured with Yao's *cell-probe* model [Yao81]. A *dynamic data structure* in this model is a collection of *cells* that stores the data, along with an algorithm that makes changes to the data or retrieves information about it by reading from and writing to some of the cells. The *word-size* of the data structure, denoted w throughout this paper, is the number of bits stored in each cell of the data structure. The time complexity for performing a particular operation is the number of cells that are touched when the operation is carried out. Usually, there is a trade-off between the time for performing different operations. For example, if we maintain a set $S \subseteq \{1, 2, \ldots, n\}$ by storing its indicator vector (with $w = 1$), then elements can be inserted and deleted from the set in time 1, but computing the median of the set could take time $\Omega(n)$ in the worst case. However, if we maintained the set by storing its elements in sorted order (with $w = \log n$), and the size of the set, then the median can be computed in time 2, but inserting elements into the set would take time $\Omega(n)$. Binary search trees are a well-known data structure that maintain sets and allow one to compute the median and predecessors in time $O(\log n)$, when $w = \log n$. One can also use a very clever data structure due to van Emde Boas [vEB77] that brings down the time required for all operations to $O(\log \log n)$, when $w = \log n$. The Fusion trees data structure of Fredman and Willard [FW93] takes $O(\log n / \log w)$ time for all operations.

Proving lower bounds on the performance of dynamic data structures is usually challenging. In their landmark paper, Fredman and Saks [FS89] were the first to establish tight lower bounds for several dynamic data structure problems. They invented the *chronogram* technique and leveraged it to prove several lower bounds. Since then, researchers have built on their techniques to prove lower bounds on many other dynamic data structure problems [PD06, Păt07, PT11, Lar12, Yu16, WY16]. Some of our own results also use the chronogram approach of Fredman-Saks.

Lower bounds for the median problem have been particularly elusive. The lone result is due to Brodal, Chaudhuri and Radhakrishnan [BCR96]* who showed that if the algorithm is only allowed to compare the contents of cells, and perform no other computation with the cells, then we must have $t_m \geq \Omega\left(n/4^{t_i}\right)$, where t_m is the number of comparisons used to compute the median, and t_i is the number of comparisons used to insert numbers into the set. Moreover, [BCR96] gave a data structure matching these bounds. As far as we know, there may be a data structure that maintains a set and allows for computing the median, with all operations taking time $O(1)$. We note here that there is a long sequence of works proving lower bounds on computing the median in the context of branching programs [Cha10, MR96, BLP15, CJP08].

Past work had found more success with understanding the complexity of the predecessor search problem. A long sequence of works has proved lower bounds here [Ajt88, Mil94, MNSW98, BF02, SV03, PT06]. In particular, [BF02, SV03] showed that some operation must take time $\Omega\left(\log \log n / \log \log \log n\right)$, when $w = \log n$, and this was improved to $\Omega\left(\log \log n\right)$ by [PT06]. Still, it remains open to understand the full trade-off between the time complexity of inserting elements

*They actually discuss the problem of computing the minimum rather than the median, but the ideas can be extended to prove lower bounds for the median problem.

and the time complexity of computing predecessors[†].

In our work, we prove new lower bounds on *non-adaptive* data structures for the median and predecessor search problems. A data structure is said to perform an operation non-adaptively if the locations of memory accessed depend only on the operation being performed, and not on the contents of the memory that are read while the operation is executing. Non-adaptive data structures turn out to be simple, and faster in practice. This is because a practical implementation can load all of the cells required to perform the operation into a local cache in a single step, rather than having to fetch cells from the memory or storage multiple times.

Several past works have proved lower bounds on various computational models under the assumption of non-adaptivity (see for example [KT00]). In the context of data structures, Brody and Larsen [BL12] showed polynomial lower bounds for various dynamic problems in the non-adaptive setting. Among other results, they showed that any data structure for reachability in directed graphs that non-adaptively checks for reachability between pairs of vertices must take time $\Omega(n/w)$, where n is the size of the underlying graph. Alon and Feige [AF09] proved non-adaptive lower bounds on static data structures for the dictionary problem.

1.1 Our Results

We prove new lower bounds on non-adaptive data structures computing the median and predecessors. Our results are obtained via a novel application of the famous Sunflower Lemma of Erdős and Rado [ER60]. The Sunflower Lemma has been used in the past to prove lower bounds on the size of monotone Boolean circuits [Raz85, AB87], but this is the first time it has found an application to understanding data structures.

Our first result concerns non-adaptive data structures for computing the median:

Theorem 1. *Any data structure that computes the median of a subset of $\{1, 2, \ldots, n\}$ while supporting non-adaptive insert operations must satisfy*

$$t_m \geq \Omega\left(\frac{n^{\frac{1}{2(t_i+1)}}}{w \cdot t_i}\right),$$

where t_m is the time required to compute the median, t_i is the time required to insert elements, and w is the word size of the cells.

Our second result concerns the predecessor search problem:

Theorem 2. *Any data structure that maintains a subset of $\{1, 2, \ldots, n\}$ while supporting non-adaptive predecessor operations must satisfy*

$$t_p \geq \Omega\left(\frac{\log n}{\log\log n + \log w}\right) \qquad \text{or} \qquad t_i \geq \Omega\left(\frac{t_p \cdot n^{\frac{1}{2(t_p+1)}}}{\log n}\right),$$

where t_i is the time required for inserts, t_p is the time required for computing predecessors and w is the word-size of the cells.

[†]We thank Mikkel Thorup for bringing this question to our attention.

These two theorems are complemented by the observation that a variant of Binary Search trees gives a data structure that can insert and delete elements non-adaptively, compute predecessors non-adaptively, and perform all operations in time $O(\log n)$, with $w = \log n$. Theorem 2 shows that there is a gap between adaptive and non-adaptive data structures computing the median and predecessors, since we know that the van Emde Boas data structure can compute predecessors in time $O(\log \log n)$ with $w = \log n$.

The rest of the paper is organized as follows. After discussing some preliminaries, we discuss some simple data structures for the median problem and the predecessor search problem in Section 3. We begin proving lower bounds In Section 4, where we demonstrate the power of our techniques by giving lower bounds for the median problem and the predecessor search problem when all operations are assumed to be non-adaptive. We then prove Theorems 1 and 2 in Sections 5 and 6 respectively.

2 Preliminaries

Unless otherwise stated, logarithms in this article are computed base two. Given $a = a_1, a_2, \ldots, a_n$, we write $a_{\leq i}$ to denote $a_1, \ldots, a_i$. We define $a_{>i}$ and $a_{\leq i}$ similarly. Similarly, we write a_{-i} to denote $a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n$. $[\ell]$ denotes the set $\{1, 2, \ldots, \ell\}$, for $\ell \in \mathbb{N}$. If ℓ is not an integer but a real number greater than 1, then $[\ell]$ denotes the set $\{1, 2, \cdots, \lfloor \ell \rfloor\}$, where $\lfloor \ell \rfloor$ denotes the largest integer less than or equal to ℓ. We denote the expected value of a random variable A to be $\mathbb{E}[A]$.

We often work with subsets of cells written by the data structure and sequence of insertions. We often write C to denote the description of the locations and contents of the cells and $|C|$ as the number of cells. Similarly, when U is a sequence of insertions, we write $|U|$ to denote the number of insert operations and U to denote the description of every insert operation with the order in which they were performed.

The *entropy* of a discrete random variable A, is defined to be

$$\mathsf{H}(A) = \sum_a \Pr[A = a] \cdot \log \frac{1}{\Pr[A = a]}.$$

For two random variables A, B, the entropy of A conditioned on B is defined as

$$\mathsf{H}(\mathsf{A}|\mathsf{B}) = \sum_{a,b} \Pr[A = a, B = b] \cdot \log \frac{1}{\Pr[A = a | B = b]}.$$

The entropy satisfies some useful properties:

Proposition 3 (Chain Rule)**.** $\mathsf{H}(A_1 A_2 | B) = \mathsf{H}(A_1|B) + \mathsf{H}(A_2|BA_1)$.

Lemma 4 (Subadditivity)**.** $\mathsf{H}(A_1 A_2|B) \leq \mathsf{H}(A_1|B) + \mathsf{H}(A_2|B)$.

Proposition 5. *For every $a, b, c > 1$, if $a \log ab \geq c$, then $a \geq \frac{c}{\log c + \log b}$.*

Proof. Suppose that $a < \frac{c}{\log c + \log b}$. We then have,

$$\begin{aligned} a \log ab &< \frac{c}{\log c + \log b} \cdot (\log b + \log c - \log(\log c + \log b)) \\ &< c, \end{aligned}$$

which contradicts the inequality $a \log ab \geq c$. Therefore, $a \geq \frac{c}{\log c + \log b}$. □

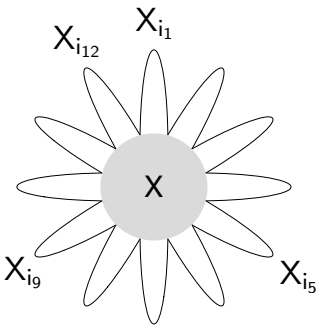


Figure 1: A Flower with 12 petals. X denotes the core of the Flower.

2.1 Sunflowers

Our proof relies on a variant[‡] of the Sunflower lemma [ER60]. The lemma we need is almost identical to a lemma proved by [AB87], and we use their ideas to prove it.

Definition 6. *A sequence of sets $X_1, \cdots, X_p$, each of size at most s is said to be a* flower *with p petals if there is a set X of size at most s such that for every i, j, $X_i \cap X_j \subseteq X$. X is called the* core *of the flower (See Figure 1).*

Next we show that a long enough sequence of sets must contain a flower.

Lemma 7 (Flower Lemma)**.** *Let $X_1, \cdots, X_k$ be a sequence of sets each of size at most s. If $k > (p-1)^{s+1}$, then there is a subsequence that is a flower with p petals.*

Proof. We prove the bound by induction on s, p. When $s = 1$, if $k > (p-1)^2$, either there are p sets that are the same or p sets that are distinct. Either way, we obtain a flower with p petals. When $p = 1$ the statement is trivially true.

Suppose that $s \geq 2$, and the sequence does not contain a flower with p petals. For each set $X \subseteq X_1$, we get a subsequence by restricting our attention to the sets X_i such that $X_i \cap X_1 = X$ and $i > 1$. By induction, the length of this subsequence can be at most $(p-2)^{s+1-|X|}$ since all of these sets have X in common, and any flower with $p-1$ petals yields a flower with p petals in our original sequence, by adding X_1 to the list of petals. Thus we get,

$$\begin{aligned} k &\leq 1 + \sum_{X \subseteq X_1} (p-2)^{s+1-|X|} \\ &= 1 + (p-2) \cdot \sum_{X \subseteq X_1} (p-2)^{s-|X|} \\ &\leq 1 + (p-2) \cdot (p-2+1)^s \leq (p-1)^{s+1}, \end{aligned}$$

as desired. □

[‡]Using the Sunflower lemma would would give us bounds with the same asymptotics, but the Flower Lemma (Lemma 7) gives cleaner bounds.

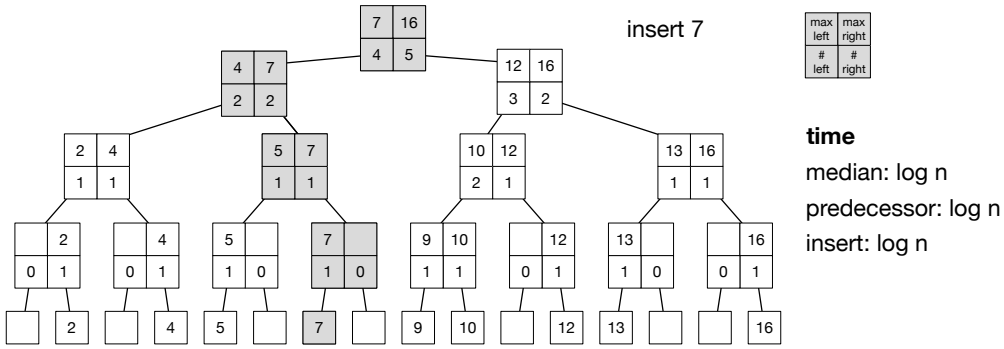


Figure 2: A data structure based on binary search trees storing the set $\{2, 4, 5, 7, , 9, 10, 12, 13, 16\}$.

3 A Data Structure for Median and Predecessor Search based on Binary Search Trees

Here we describe a data structure that maintains a subset of $\{1, \ldots, n\}$ allowing non-adaptive inserts, non-adaptive predecessor computations and adaptive median computations. The data structure builds on the well known binary search tree on $\{1, \ldots, n\}$ and is very close to the *x-fast trie* (see [Wil83]). This data structure matches many of the lower bounds in our proofs.

Theorem 8. *There is a data structure that maintains a subset of* $\{1, 2, \ldots, n\}$ *and supports insertions, deletions and computing medians and predecessors. All operations take time* $O(\log n)$*, the word size is* $\log n$*, and all operations except for the median operation are non-adaptive.*

Proof. Without loss of generality, we may assume that n is a power of 2. We maintain a balanced binary tree of height $\log n$. Every leaf is assigned an element from the universe.

There is a memory cell associated with every leaf and four memory cells associated with every internal node of the tree. The cells corresponding to each internal node store the number of elements in the left subtree rooteed at that node, the number of elements stored in the right subtree, the maximum element of the left subtree and the maximum element of the right subtree. Figure 2 shows an example of the data structure.

To insert an element into the set, we only need to access the cells associated with each node on the path from the root to the corresponding leaf. These are the only cells that need to be modified to make the data structure consistent with the new set. Deletions can be performed in the same way. The time required for these operations is $O(\log n)$, and they are non-adaptive.

To compute the median, we read the cells associated with the root to determine if the median belongs to the left or the right sub tree. Accordingly, we read the cells associated with either the left or the right child and recurse to find the median. The time required for this operation is $O(\log n)$, but it is adaptive.

To compute the predecessor of an element, we only need to access the cells associated with every node on the path from the root to the corresponding leaf in the tree. The predecessor is the maximum of last non-empty left-subtree seen on this path. Again, we see that this operation takes $O(\log n)$ time, and is non-adaptive. □

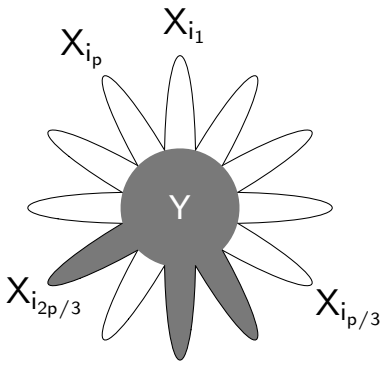


Figure 3: Y denotes the core of the Flower, and the shaded cells are the only cells accessed when inserting A.

4 Lower Bounds when Insert, Median and Predecessor Operations are Non-Adaptive

As a warm up, we prove lower bounds when all operations in the data structure are non-adaptive.

Theorem 9. *Any data structure with non-adaptive insertions and median computations must take time* $\Omega\left(\frac{\log n}{\log\log n+\log w}\right)$ *for some operation.*

Proof. Consider the sequence of sets $\mathcal{X} = X_1, \cdots, X_n$ where

$$X_i = \{j | \text{cell } j \text{ is accessed while inserting } i, \text{ or when computing the median}\}.$$

If t is the time required for the operations of the data structure, then each set X_i is of size at most $2t$. The key observation is that there cannot be a large flower in $\mathcal{X}$:

Claim 10. *If* $\mathcal{X}$ *has a flower with* p *petals, then* $p \leq 6wt + 2$.

Proof. Suppose for the sake of contradiction that the sequence $X_{i_1}, \cdots, X_{i_p}$ is a flower with $i_1 < i_2 < \cdots < i_p$, and $p = 6wt+3$. Then let A be a uniformly random subset of $\{i_{p/3+1}, i_{p/3+2}, \cdots, i_{2p/3}\}$ and Y denote the the contents of the core of the flower after inserting elements of A into the data structure (see Figure 3).

We show that Y serves as an encoding of A. This is because Y is all we need to reconstruct the execution of the following sequence of insert and median operations: insert i_1, compute the median, insert i_2, compute the median,$\cdots$, insert $i_{p/3}$, compute the median. The answers to these median computations determine the elements in A between its smallest element and median. Every insert operation requires access to a set of cells within the core and outside of it. By the definition of the flower, the cells outside of the core remain unchanged after inserting elements of A. Therefore, the sequence can be simulated by access only to the core (see Figure 3). Similarly, executing the following operations helps retrieve elements in A between its median and largest element: insert $i_{2p/3+1}$, compute the median, insert $i_{2p/3+2}$, compute the median,$\cdots$, insert i_p, compute the median.

Hence $\mathsf{H}(Y) \leq 2t \cdot w$. Also, $\mathsf{H}(Y) \geq p/3$, which proves the claim. □

By the Flower-Lemma (Lemma 7), the sequence $\mathcal{X}$ has a flower with $n^{\frac{1}{2t+1}}$ petals. Then we get

$$t \geq \frac{p-2}{6w} \geq \frac{n^{\frac{1}{2t+1}}-2}{6w},$$

where the last inequality follows from the choice of p. After rearranging, we get

$$t \cdot \log wt \geq \Omega\left(\log n\right).$$

Proposition 5 implies the desired bound on t. □

Next we prove a similar lower bound for the predecessor search problem.

Theorem 11. *Any data structure for the predecessor problem with non-adaptive insert operations and non-adaptive predecessor operations must have time* $\Omega\left(\frac{\log n}{\log\log n+\log w}\right)$.

Proof. Let $\mathcal{X} = X_1, \cdots, X_n$, where

$$X_i = \{j | \text{cell } j \text{ is accessed while inserting } i \text{ or computing the predecessor of } i\}.$$

It t is the time required for the operations of the data structure, then each set X_i is of size at most $2t$. We first show that the time complexity can be lower bounded in terms of the the number of petals in a flower belonging to $\mathcal{X}$.

Claim 12. *If $\mathcal{X}$ has a flower with p petals, then $p \leq 4tw + 1$.*

Proof. Let $X_{i_1}, \cdots, X_{i_p}$ form a flower and $i_1 < i_2 < \cdots < i_p$, and $p = 4tw + 2$. Let A be a random subset of $\{i_1, i_3, \cdots, i_{p-1}\}$ and Y denote the contents of the cells in the core after inserting elements of A.

We show that Y serves as an encoding of A. This is because to reconstruct Y, it suffices to compute the predecessors of the following elements: $i_2, i_4, \cdots, i_p$. Every predecessor operation requires access to the core and the cells outside of it. By the definition of the flower, the cells outside of the core remain unchanged after inserting the elements of A. Therefore, the sequence of predecessor operations can be simulated by access only to the core.

Hence $\mathsf{H}(Y) \leq 2t \cdot w$. Moreover, $\mathsf{H}(Y) \geq p/2$, which implies the claimed inequality. □

By the Flower Lemma 7, the sequence $\mathcal{X}$ has a flower with $n^{\frac{1}{2t+1}}$ petals. So $t \geq \frac{p-1}{4w} \geq \frac{n^{\frac{1}{2t+1}}-1}{4w}$, which follows from the choice of p. After rearranging, we get

$$t \cdot \log wt \geq \Omega\left(\log n\right).$$

Proposition 5 implies the desired bound on t.

□

5 Lower Bounds when Insertions are Non-Adaptive and Computing the Median is Adaptive

In this section, we prove Theorem 1. Consider the sequence $\mathcal{X} = X_1, X_2, \cdots, X_n$, where

$$X_i = \{j | \text{cell } j \text{ is accessed while inserting } i\}.$$

Let p be the largest integer such that $p \leq m^{\frac{1}{t_i+1}}$ and p is divisible by 3. Let q be the largest integer such that $q(q+1) < p/3$. By the Flower Lemma 7, $\mathcal{X}$ has a flower with $m^{\frac{1}{t_i+1}}$ petals. Note that $p \geq m^{\frac{1}{t_i+1}} - 3$ and $q \geq \frac{\sqrt{p}}{2\sqrt{3}}$. For ease of notation, we assume that $X_1, \cdots, X_p$ are the promised flower.

We start by showing that the data structure gives a data structure for computing the k^{th} smallest element of S. For any data structure D with non-adaptive inserts, define the sequence $\mathcal{X}_D = X_1, X_2, \cdots, X_n$, where

$$X_i = \{j | \text{cell } j \text{ is accessed by } D \text{ while inserting } i\}.$$

Lemma 13. *Let $a = \lceil \log m/w \rceil$. If there is a data structure D for the median problem on $\{1, 2, \ldots, p\}$ such that $\mathcal{X}_D$ is a Flower, and D supports non-adaptive inserts in time t_i and computes the median in time t_m, then there is a data structure D' on $[q(q+1)]$ such that $\mathcal{X}_{D'}$ is a Flower, and D' supports non-adaptive inserts in time $t_i + a$ and computes the k^{th} smallest element in time $t_i + t_m + a$.*

Proof. Without loss of generality, after renaming, the universe for the median problem can be assumed to be $\{-p/3+1, \cdots, 0, 1, \cdots, 2p/3\}$. Let D' be the new data structure that maintains a subset of $[q(q+1)]$ and computes the k^{th} smallest element. Let $c_1, c_2, \cdots, c_a$ be a sequence of cells not used by D. $c_1, \cdots, c_a$ together store the size of the set. The insert operation of D' executes the insert operation of D followed by incrementing the size of the set by accessing $c_1, \cdots, c_a$. The time for the new insert operation is $t_i + a$.

Let S be the set that D' stores. We have, $|S| \leq q \cdot (q+1) < p/3$. We now compare $|S|$ to the number of elements in $\{-p/3+1, \cdots, 0, 1, \cdots, 2p/3\}$ that are greater than $q(q+1)$ and smaller than 1. We have,

$$|\{q(q+1)+1, \cdots, 2p/3\}| \geq 2p/3 - q(q+1) > 2p/3 > |S|.$$

Similarly, $|\{-p/3+1, \cdots, 0\}| > |S|$. For every $k > |S|/2$, there exists an integer $k' \leq |S|$, which is a function of $|S|, k$, such that the new median after inserting $\{q(q+1)+1, q(q+1)+2, \cdots, q(q+1)+k'\}$ equals the k^{th} smallest element in S. Similarly, for every $k \leq |S|/2$, there exists an integer $k' \leq |S|$, which is a function of $|S|, k$, such that the new median after inserting $\{-p/3+1, -p/3+2, \cdots, -p/3+k'\}$ equals the k^{th} smallest element in S. Inserting an element from $\{-p/3+1, \cdots, 0\} \cup \{q(q+1)+1, \cdots 2p/3\}$ requires access to the core and its outside. Since the cells outside the core are yet to be accessed, all the insertions can be simulated by access only to the core.

Therefore, the time to compute the k^{th} smallest element equals the time to access the core and $c_1 \cdots, c_a$, and perform a median operation, which is at most $t_i + t_m + a$. □

Let $a = \lceil \log m/w \rceil$. By Lemma 13, there exists a data structure on universe $[q(q+1)]$ that supports non-adaptive inserts in time $t_i + a$ and computes the k^{th} smallest element in time $t_i + t_m + a$.

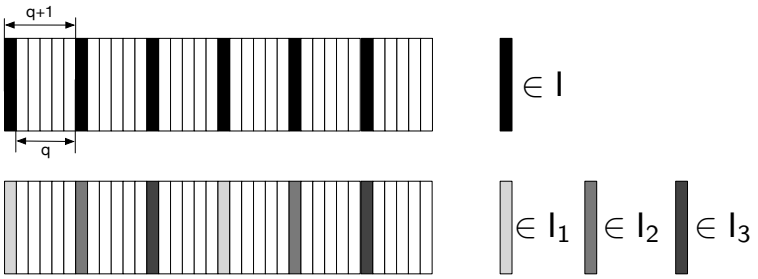


Figure 4: The figure describes the sets I, I_1, I_2, I_3. The number of insertion batches is 3.

Let $I = \{i | i = (j-1)(q+1)+1 \text{ for some } j \in [q]\}$ and $S_0 = [q(q+1)] \setminus I$.

- Insert the elements of S_0.

For $i \in \left[\frac{q}{4wt}\right]$, let

$$I_i = \{((j-1)4wt + i - 1) \cdot (q+1) + 1 \mid j \in [4wt]\}.$$

- i^{th} batch of insertions correspond to inserting elements from a set $A_i \subseteq I_i$ chosen uniformly at random. (See Figure 4)

Figure 5: Hard Distribution for median with parameters p, q, S.

We now proceed to lower bound $t_m + t_i + a$. For ease of notation, set $t = t_i + a$. We use the chronogram technique of [FS89] using a distribution suggested in [PT14], but with an improved analysis that exploits the underlying Flower structure.

In what is to follow, we maintain a set $S \subseteq [q(q+1)]$. S is empty to begin with. Figure 5 defines the distribution on the insertions. The sequence of insertions is followed by an operation to compute the $(k \cdot q)^{\text{th}}$ smallest element in S, where k is chosen uniformly at random from $[q]$.

For $i \in \left[\frac{q}{4wt}\right]$, let Y_i denote the random variable that stores the contents of the core at the end of the i^{th} batch of insertions.

Let $T = (T_1, \cdots, T_q)$, where T_k is the parity of the $(k \cdot q)^{\text{th}}$ smallest element in S for $k \in [q]$. Recall that A_i denotes the i'th batch of inserts into the set, and A_{-i} denotes all the inserts except for the i'th batch. We have

$$\begin{aligned} \mathsf{H}(T|A_{-i}Y_i) &= \mathsf{H}(T|A_{-i}) - \mathsf{H}(Y_i|A_{-i}) \\ &\geq \mathsf{H}(T|A_{-i}) - wt, \end{aligned} \tag{1}$$

where the last inequality followed from the fact that $\mathsf{H}(Y_i) \leq wt$. Note that A_i, A_{-i} determine T, and A_i is determined by T, A_{-i}. Hence, the chain rule of entropy implies that

$$\mathsf{H}(T|A_{-i}) = \mathsf{H}(A_i|A_{-i}) = \mathsf{H}(A_i) = 4wt.$$

For a uniformly random coordinate $L \in [q]$,

$$\begin{aligned}\mathsf{H}(T_L|A_{-i}Y_iL) &= \sum_{l=1}^{q} \frac{\mathsf{H}(T_l|A_{-i}Y_i)}{q} = \sum_{l=i}^{q} \frac{\mathsf{H}(T_l|A_{-i}Y_i)}{q} \\ &\geq \sum_{l=i}^{q-i} \frac{\mathsf{H}(T_l|A_{-i}Y_i)}{q}. \end{aligned} \tag{2}$$

Observe that for every $j, j' \in [q]$ such that $(i-1)\cdot 4wt + i - 1 \leq j < j' < i \cdot 4wt + i - 1$, conditioned on $A_{-i}Y_i$, we have that T_j determines $T_{j'}$ and vice versa. Therefore,

$$\begin{aligned}\sum_{l=i}^{q-i} \frac{\mathsf{H}(T_l|A_{-i}Y_i)}{q} &\geq \frac{1}{4wt} \cdot \mathsf{H}(T_i \cdots T_{q-i}|A_{-i}Y_i) \\ &= \frac{\mathsf{H}(T_i \cdots T_q|A_{-i}Y_i) - \mathsf{H}(T_{q-i+1} \cdots T_q|T_{<q-i+1}A_{-i}Y_i)}{4wt}, \end{aligned} \tag{3}$$

where the inequality follows from subadditivity of entropy.

Note that $\mathsf{H}(T|A_{-i}Y_i) = \mathsf{H}(T_i \cdots T_q|A_{-i}Y_i)$. Combining equations Equations 2, 3,

$$\begin{aligned}\mathsf{H}(T_L|A_{-i}Y_iL) &\geq \frac{\mathsf{H}(A_i) - wt - 1}{4wt} \\ &\geq 1 - \frac{1+wt}{4wt}, \end{aligned} \tag{4}$$

where the first inequality follows from Equation 1 and the fact that

$$\mathsf{H}(T_{q-i+1} \cdots T_q|T_{<q-i+1}A_{-i}Y_i) \leq 1.$$

Let Q_i be the indicator random variable denoting whether a cell that was last accessed while inserting elements from the set A_i is accessed when computing the $(L \cdot q)^{\text{th}}$ smallest element. Upon fixing A_{-i},Y_i, we can simulate the updates following the i^{th} batch of insertions. This is because for every element $r \in A_{>i}$, the cells of X_r outside the core are yet to be accessed. Therefore, Y_i is sufficient to simulate all insertions.

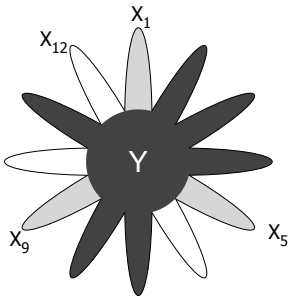
For $l \in [q]$, conditioned on A_{-i}, Y_i, either T_l is determined or Q_i takes the value 1. Therefore $\mathsf{H}(T_l|A_{-i}Y_i) \leq Q_i$. We now have,

$$\mathsf{H}(T_L|A_{-i}Y_iL) \leq \mathbb{E}\left[Q_i\right]. \tag{5}$$

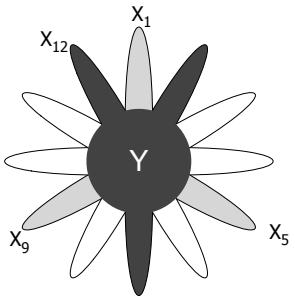
Combining Equations 4,5 we get that for all $i \in \left[\frac{q}{4wt}\right]$, $\mathbb{E}\left[Q_i\right] \geq 1 - \frac{1+wt}{4wt}$. Since, $wt \geq 1$, $\mathbb{E}\left[Q_i\right] \geq \frac{1}{2}$. By Lemma 13, $t_m + t_i + a \geq \sum_{i=1}^{\frac{q}{4wt}-1} \mathbb{E}\left[Q_i\right]$. Therefore,

$$t_m + t_i + a \geq \Omega\left(\frac{q}{wt}\right),$$

which implies the desired bound.



(a) $\sum_{i=1}^{3} Z_i = 2$.



(b) $\sum_{i=1}^{3} Z_i = 0$.

Figure 6: Any petal X_i that is shaded black indicate that $\mathsf{Pred}'(i) \neq \mathsf{Pred}(i)$. The unshaded petals indicate that their cells outside the core were not overwritten while inserting the set A. The petals shaded grey indicate the universe.

6 Lower Bounds when Insertions are Adaptive and Predecessor Operations are Non-Adaptive

In this section we prove Theorem 2. Consider the sequence $\mathcal{X} = X_1, \cdots, X_n$, where

$$X_i = \{j | \text{cell } j \text{ is accessed while computing the predecessor of } i\}.$$

By the Flower Lemma (Lemma 7), $\mathcal{X}$ contains a Flower with $n^{\frac{1}{t_p+1}}$ petals. Let q be the largest even integer such that $q(q+1) \leq n^{\frac{1}{t_p+1}}$. For ease of notation, we assume that $X_1, X_2 \cdots, X_{q(q+1)}$ are the promised Flower. Note that $q \geq \frac{n^{\frac{1}{2(t_p+1)}}}{2}$.

Let $A \subset \{i | i = (j-1)(q+1)+1 \text{ for some } j \in [q]\}$ be a uniformly random subset. Insert all elements of A. We have,

$$\mathsf{H}(A) = q. \tag{6}$$

For $i \in [q(q+1)]$, let $\mathsf{Pred}'(i)$ be the value obtained by simulating the predecessor computation assuming that the cells outside the core were never accessed when A was inserted. Note that $\mathsf{Pred}'(i)$ can be computed from the cells in the core. Let $\mathsf{Pred}(i)$ be the predecessor of i. For every $i \in [q]$, define

$$Z_i = \begin{cases} 1, & \left|\left\{j \in \{i(q+1)-q+1, \cdots, i(q+1)\} \middle| \mathsf{Pred}(j) \neq \mathsf{Pred}'(j)\right\}\right| > q/2 \\ 0, & \text{otherwise.} \end{cases}$$

Since the total number of cells accessed while inserting A is atleast $\frac{q \cdot \sum_{l=1}^{q} Z_l}{2}$,

$$\sum_{l=1}^{q} Z_l \cdot (q/2) \leq t_i \cdot q, \tag{7}$$

where the inequality follows from the fact that $|A| \leq q$. Let Y denote the contents of the core after inserting elements of A, the names of the elements i with $Z_i = 1$, and whether or not $i \in A$ for every element with $Z_i = 1$. In other words, Y encodes the core, the set $\{i : Z_i = 1\}$ and the set $A \cap \{i : Z_i = 1\}$.

Lemma 14. *Y encodes A.*

Proof. It suffices to come up with a decoding procedure that given Y recovers A. The decoding algorithm first recovers elements of A in $\{i|Z_i = 1\}$ from the description of Y. By definition, if $i \in A$ and $i \notin \{i|Z_i = 1\}$, then $\mathsf{Pred}'(j) = i$ for the majority values of $j \in \{i(q+1) - q + 1, \cdots, i(q+1)\}$. If $i \notin \{i|Z_i = 1\}$, then the decoding algorithm computes $\mathsf{Pred}'(j)$ for every $j \in \{i(q+1) - q + 1, \cdots, i(q+1)\}$. If the majority of the answers equal i, then the decoding algorithm infers that $i \in A$. Otherwise, it infers that $i \notin A$. This determines whether or not $i \in A$. □

We now analyze the length of the encoding of Y. The contents of the core can be described with wt_p bits. The number of bits required to encode the rest of Y is at most $(\log q + 1) \cdot \sum_{l=1}^{q} Z_l$. This is because it takes $\log q$ bits to encode each element in $\{i|Z_i = 1\}$ and an extra bit to indicate its membership in A. Therefore, the length of the encoding is at most $wt_p + (\log q + 1) \cdot \sum_{l=1}^{q} Z_l$. We now have the following upper bound on the entropy of A.

$$\mathsf{H}(A) \leq wt_p + (\log q + 1) \cdot \sum_{l=1}^{q} Z_l \leq wt_p + 2 \cdot (\log q + 1) \cdot t_i, \tag{8}$$

where the last inequality follows from Equation 7. Combining Equations 6, 8, we get that

$$t_p \geq \frac{q}{w} - \frac{2 \cdot (\log q + 1) \cdot t_i}{w}. \tag{9}$$

Observe that either $t_i \geq \frac{q}{4 \cdot (\log q + 1)}$ or not. In the former case, since $q \geq \frac{n^{\frac{1}{2(t_p+1)}}}{2}$, we can conclude that $t_i \geq \Omega\left(\frac{n^{\frac{1}{2(t_p+1)}} \cdot t_p}{\log n}\right)$. In the latter case, Equation 9 implies that $t_p \geq \frac{q}{2w}$. Since $q \geq \frac{n^{\frac{1}{2(t_p+1)}}}{2}$, we can conclude that $t_p \log(wt_p) \geq \Omega(\log n)$. Using Proposition 5, we obtain the desired bound of $t_p \geq \Omega\left(\frac{\log n}{\log \log n + \log w}\right)$.

Acknowledgments

We thank Paul Beame for useful discussions and bringing to our attention a variant of the Sunflower lemma from [AB87]. We thank Mikkel Thorup for a helpful conversation.

References

[AB87] N. Alon and R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[AF09] N. Alon and U. Feige. On the power of two, three and four probes. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*. SIAM, 2009.

[Ajt88] M. Ajtai. A lower bound for finding predecessors in yao's call probe model. *Combinatorica*, 8(3), 1988.

[BCR96] G. S. Brodal, S. Chaudhuri, and J. Radhakrishnan. The randomized complexity of maintaining the minimum. In *SWAT: Scandinavian Workshop on Algorithm Theory*, 1996.

[BF02] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *JCSS: Journal of Computer and System Sciences*, 65, 2002.

[BL12] J. Brody and K. G. Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. *CoRR*, abs/1208.2846, 2012.

[BLP15] P. Beame, V. Liew, and M. Patrascu. Finding the median (obliviously) with bounded space. *CoRR*, abs/1505.00090, 2015.

[Cha10] T. M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6(2), 2010.

[CJP08] A. Chakrabarti, T. S. Jayram, and M. Patrascu. Tight lower bounds for selection in randomly ordered streams. In *Proc. 19th Symp. on Discrete Algorithms (SODA)*, pages 720–729. ACM/SIAM, 2008.

[ER60] P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of London Mathematical Society*, 35:85–90, 1960.

[FS89] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1989.

[FW93] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *JCSS: Journal of Computer and System Sciences*, 47, 1993.

[KT00] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2000.

[Lar12] K. G. Larsen. The cell probe complexity of dynamic range counting. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 85–94, 2012.

[Mil94] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proceedings of the 26th Annual Symposium on the Theory of Computing*, pages 625–634, New York, May 1994. ACM Press.

[MNSW98] P. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57:37–49, 1 1998.

[MR96] J. I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *TCS: Theoretical Computer Science*, 165, 1996.

[Păt07] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th ACM Symposium on Theory of Computing (STOC)*, pages 40–46, 2007.

[PD06] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also STOC'04, SODA'04.

[PT06] M. Patrascu and M. Thorup. Time-space trade-offs for predecessor search. *CoRR*, 2006.

[PT11] M. Pătraşcu and M. Thorup. Don't rush into a union: Take time to find your roots. In *Proc. 43rd ACM Symposium on Theory of Computing (STOC)*, pages 559–568, 2011. See also arXiv:1102.1783.

[PT14] M. Patrascu and M. Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. *CoRR*, abs/1408.3045, 2014.

[Raz85] A. A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akademii Nauk SSSR*, 281, 1985.

[SV03] P. Sen and S. Venkatesh. Lower bounds for predecessor searching in the cell probe model. *CoRR*, cs.CC/0309033, 2003.

[vEB77] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, June 1977.

[Wil83] D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, pages 81–84, 1983.

[WY16] O. Weinstein and H. Yu. Amortized dynamic cell-probe lower bounds from four-party communication. *Electronic Colloquium on Computational Complexity (ECCC)*, 23, 2016.

[Yao81] A. Yao. Should tables be sorted? *JACM: Journal of the ACM*, 28, 1981.

[Yu16] H. Yu. Cell-probe lower bounds for dynamic problems via a new communication model. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 362–374, 2016.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il