Lower Bounds on Non-Adaptive Data Structures Maintaining Sets of Numbers, from Sunflowers

Sivaramakrishnan Natarajan Ramamoorthy[1]
Paul G. Allen School for Computer Science & Engineering, University of Washington, Seattle, USA
sivanr@cs.washington.edu

Anup Rao[2]
Paul G. Allen School for Computer Science & Engineering, University of Washington, Seattle, USA
anuprao@cs.washington.edu

Abstract

We prove new cell-probe lower bounds for dynamic data structures that maintain a subset of $\{1, 2, ..., n\}$, and compute various statistics 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. For any such data structure that can compute the median of the set, we prove that:

$$t_{\mathsf{med}} \geq \Omega\left(\frac{n^{\frac{1}{t_{\mathsf{ins}}+1}}}{w^2 \cdot t_{\mathsf{ins}}^2}\right),$$

where t_{ins} is the number of memory locations accessed during insertions, t_{med} is the number of memory locations accessed to compute the median, and w is the number of bits stored in each memory location. When the data structure is able to perform deletions non-adaptively and compute the minimum non-adaptively, we prove

$$t_{\mathsf{min}} + t_{\mathsf{del}} \geq \Omega\left(\frac{\log n}{\log w + \log\log n}\right),$$

where t_{min} is the number of locations accessed to compute the minimum, and t_{del} is the number of locations accessed to perform deletions. For the predecessor search problem, where the data structure 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

$$\text{either } t_{\mathsf{pred}} \geq \Omega\left(\frac{\log n}{\log\log n + \log w}\right), \text{ or } t_{\mathsf{ins}} \geq \Omega\left(n^{\frac{1}{2(t_{\mathsf{pred}}+1)}}\right),$$

where t_{pred} is the number of locations accessed to compute predecessors.

These bounds are nearly matched by Binary Search Trees in some range of parameters. Our results follow from using the Sunflower Lemma of Erdős and Rado [11] together with several kinds of encoding arguments.

2012 ACM Subject Classification Theory of computation → Cell probe models and lower bounds

Keywords and phrases Non-adaptive data structures, Sunflower lemma

Digital Object Identifier 10.4230/LIPIcs.CCC.2018.27

[1] Supported by the National Science Foundation under agreement CCF-1420268 and CCF-1524251
[2] Supported by the National Science Foundation under agreement CCF-1420268 and CCF-1524251

33rd Computational Complexity Conference (CCC 2018).
Editor: Rocco A. Servedio; Article No. 27; pp. 27:1–27:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank Paul Beame for useful discussions and bringing to our attention a variant of the Sunflower lemma from [2]. We thank Pavel Hrubeš for observing that one could insert a set using deletions, to prove Theorem 1. We thank Kasper Green Larsen and Mikkel Thorup for helpful conversations. We thank an anonymous reviewer for a suggestion that helped improve a bound in Theorem 3.

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 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 *minimum*, *median* 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 these statistics.

The performance of data structures is usually measured with Yao's *cell-probe* model [32]. 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 accessed 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 $S \subseteq \{1, 2, \dots, n\}$ 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 [29] 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 [14] 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 [13] 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 [24, 23, 26, 18, 33, 30]. Notably, Pătraşcu and Thorup [26] proved lower bounds on data structures that can compute the k'th smallest number of the set for every k via a reduction from Parity Sum for which [13] used the chronogram technique to prove a lower bound. This shows that computing the k'th smallest element takes strictly more time than just computing the median. Some of our own results also use the chronogram technique of Fredman-Saks.

Lower bounds on data structures for computing single statistics like the median or minimum have been particularly elusive. Computing statistics like the median and the minimum are very fundamental in algorithm design. The best known upper bounds require $O(\log\log n)$ time for insertions, and median and minimum computations. It is surprising that no previous lower bounds were known in the cell-probe model. We prove the first lower bounds on the performance of data structures computing the median and minimum. Brodal, Chaudhuri and Radhakrishnan [7] showed that if the data structure is only allowed to compare the contents of cells, and perform no other computation with the cells, then we must have $t_{\mathsf{min}} \geq \Omega\left(n/4^{t_{\mathsf{ins}}}\right)$, where t_{min} is the number of comparisons used to compute the minimum, and t_{ins} is the number of comparisons used to insert numbers into the set. Moreover, [7] gave a data structure matching these bounds. The same bounds apply for computing the median as well. It remains an interesting open problem to prove a lower bound of $t_{\mathsf{ins}} + t_{\mathsf{med}} \geq \Omega(\log\log n)$ in the cell-probe model when $w = O(\log n)$, where t_{ins} is the time for insertions and t_{med} is the time to compute the median. We note here that there is a long sequence of works proving lower bounds on computing the median in the context of branching programs [10, 21, 5, 9].

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 [1, 20, 19, 4, 28, 25]. In particular, [4, 28] 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 [25]. Still, it remains open to understand the full trade-off between the time complexity of inserting elements and the time complexity of computing predecessors[3].

In our work, we prove new lower bounds on *non-adaptive* data structures that allow for computing the median, minimum, and predecessors of elements. 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.

Perhaps the most widely known and basic dynamic data structure for maintaining sets of numbers is the binary search tree (see Appendix A for a description). Both insertions and deletions into a binary search tree are non-adaptive operations. Indeed, all of the assumptions regarding non-adaptivity made in our lower bounds are satisfied by binary search trees—so the models we consider here are both well motivated and quite natural. Non-adaptive data structures tend 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 multiple times.

Several past works have proved lower bounds on various computational models under the assumption of non-adaptivity (see for example [17]). In the context of data structures, Brody and Larsen [8] 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. [3, 16] proved non-adaptive lower bounds on static data structures for the dictionary problem in the bit probe model.

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

1.1 Our Results

We prove new lower bounds on non-adaptive data structures computing the minimum, median and predecessors. Our results are obtained via an application of the famous Sunflower Lemma of Erdős and Rado [11]. The Sunflower Lemma was used in the past to prove lower bounds on dynamic data structures by Frandsen and Milterson [12] and then again for static data structures by Gal and Milterson [15], and our use of it is similar. However, in the setting of non-adaptive data structures, we are able to leverage the lemma to get results even when the word size is large.

Our first result proves a lower bound when both deletions and minimum computations are non-adaptive[4]. Similar results hold for computing the median and predecessors as well, but they are subsumed by the theorems to follow.

▶ **Theorem 1.** *Any data structure that computes the minimum of a subset of* $\{1, 2, \ldots, n\}$ *while supporting non-adaptive delete operations and non-adaptive minimum computations must take time* $\Omega\left(\frac{\log n}{\log\log n + \log w}\right)$ *for some operation, where* w *is the word size of the cells.*

Our second result concerns non-adaptive data structures for computing the median. Here the lower bound holds even if the median computation is adaptive and the insertion operation is non-adaptive:

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

$$t_{\mathsf{med}} \geq \Omega\left(\frac{n^{\frac{1}{t_{\mathsf{ins}}+1}}}{w^2 \cdot t_{\mathsf{ins}}^2}\right),$$

where t_{med} *is the time required to compute the median,* t_{ins} *is the time required to insert elements, and* w *is the word size of the cells.*

Our last result concerns the predecessor search problem. Here the lower bound holds even if the insertion operation is adaptive, as long as the predecessor computations are non-adaptive:

▶ **Theorem 3.** *Any data structure that maintains a subset of* $\{1, 2, \ldots, n\}$ *while supporting non-adaptive predecessor operations must satisfy*

$$t_{\mathsf{pred}} \geq \Omega\left(\frac{\log n}{\log\log n + \log w}\right) \qquad \text{or} \qquad t_{\mathsf{ins}} \geq \Omega\left(n^{\frac{1}{2(t_{\mathsf{pred}}+1)}}\right),$$

where t_{ins} *is the time required for inserts,* t_{pred} *is the time required for computing predecessors and* w *is the word-size of the cells.*

Very recently, Boninger, Brody and Kephart [6] independently obtained some lower bounds on non-adaptive data structures computing predecessors. Among other results, they showed that any data structure with non-adaptive insertions and non-adaptive predecessor computations must have[5] $t_{\mathsf{ins}} \geq \Omega(\log n)$, or $t_{\mathsf{pred}} \geq \frac{\log n}{\log w + \log t_{\mathsf{ins}}}$. Our bounds do not require

[4] The analogous result for computing the maximum also holds. Its proof is nearly identical to the proof for theorem about the minimum.

[5] [6] consider the tradeoff with the size of the set being added, which allows them to prove lower bounds even when the data structure is only required to maintain small sets. The bound stated here is what they obtain when the size of the set is allowed to be arbitrary.

non-adaptivity for the insertion operations, and are quantitatively better when $t_{\mathsf{pred}} = o(\log n/\log\log n)$. We also note that the *cell sampling* technique ([22, 18]) does not give any meaningful lower bounds for these problems.

Our 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 and Theorem 3 show 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 both in time $O(\log\log n)$ with $w = \log n$.

The rest of the paper is organized as follows. After the preliminaries, we begin proving lower bounds in Section 3, where we give an introduction to our techniques by proving lower bounds for several problems when all operations are assumed to be non-adaptive. We prove Theorem 1 there. We then prove Theorem 2 in Section 4, and Theorem 3 in Section 5. We discuss a simple data structure based on binary search trees for these problems in Appendix A.

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 $a_{<i}$ 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 $\ell \in \mathbb{N}$.

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

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

The entropy satisfies some useful properties:

▶ **Proposition 4** (Chain Rule). $\mathsf{H}(A_1A_2|B) = \mathsf{H}(A_1|B) + \mathsf{H}(A_2|BA_1)$.

▶ **Lemma 5** (Subadditivity). $\mathsf{H}(A_1A_2|B) \leq \mathsf{H}(A_1|B) + \mathsf{H}(A_2|B)$.

▶ **Proposition 6.** *For every $a, b \geq 1$ and $c > 2$, 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,

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

where the last inequality follows from the fact that $c > 2$. This contradicts $a \log ab \geq c$, and therefore, $a \geq \frac{c}{\log c + \log b}$. ◀

▶ **Proposition 7.** *For $1 \leq k \leq n$, $\log \binom{n}{k} \leq k \cdot \log \frac{en}{k}$.*

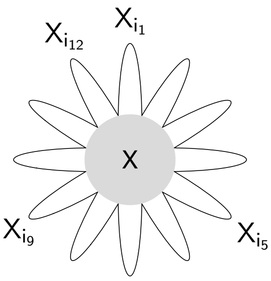


Figure 1 A Flower with 12 petals. X denotes the core of the Flower.

2.1 Sunflowers

Our proof relies on a variant[6] of the Sunflower lemma [11]. The lemma we need is almost identical to a lemma proved by [2], and we use their ideas to prove it.

▶ **Definition 8.** A sequence of sets $X_1, \cdots, X_p$ is called a t-*flower* with p petals if each set in the sequence is of size t, and there is a set X of size at most t such that for every i, j, $X_i \cap X_j \subseteq X$. X is called the *core* of the flower.

See Figure 1 for an illustration of a flower. Next, following [2], we show that a long enough sequence of sets must contain a flower.

▶ **Lemma 9** (Flower Lemma). *Let $X_1, \cdots, X_n$ be a sequence of sets each of size t. If $n > (p-1)^{t+1}$, then there is a subsequence that is a t-flower with p petals.*

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

Suppose that $t \geq 2$, and the sequence does not contain a t-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 $(p-2)^{t+1-|X|}$ since all of these sets have X in common, and any $(t-|X|)$-flower with $p-1$ petals yields a t-flower with p petals in our original sequence, by adding X_1 to the list of petals. Thus we get,

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

as desired. ◀

3 Lower Bounds when All Operations are Non-Adaptive

As a warm up, we prove some loose lower bounds when all operations in the data structure are non-adaptive. In the next section, we prove our final theorems where we only assume that some of the operations are non-adaptive.

[6] Using the Sunflower lemma would would give us bounds with the same asymptotics, but the Flower Lemma (Lemma 9) gives cleaner bounds.

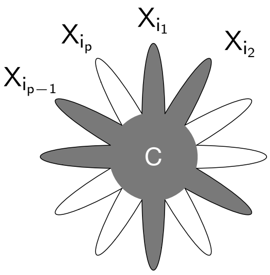


Figure 2 C denotes the core of the Flower, and the shaded cells are the only cells accessed when deleting $\{i_1, i_2, \cdots, i_p\} \setminus S$.

We start by proving Theorem 1, which gives a lower bound on the time for any data structure that computes minimum and deletions non-adaptively.

Proof of Theorem 1. Consider the sequence of sets $\mathcal{X} = X_1, \cdots, X_n$ where

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

If t is the time required for the operations of the data structure, then each set X_i is of size at most $2t$. Without loss of generality, we can assume that each X_i is of size *exactly* $2t$. The key observation is that there cannot be a large $2t$-flower in $\mathcal{X}$:

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

Proof. Suppose for the sake of contradiction that the sequence $X_{i_1}, \cdots, X_{i_p}$ is a $2t$-flower with $i_1 < i_2 < \cdots < i_p$, and $p = 2wt + 1$. Then let S be any subset of $\{i_1, i_2, \cdots, i_p\}$ and C denote the the contents of the core of the $2t$-flower after inserting the set $\{i_1, \ldots, i_p\}$ and then deleting the elements of $\{i_1, i_2, \ldots, i_p\} \setminus S$.

We show that C serves as an encoding of S. This is because C is all we need to reconstruct the execution of the following sequence of deletion and minimum operations: compute the minimum, delete the minimum, compute the minimum, delete the minimum, and so on. The answers to these computations determine the elements in S. The answer to the first minimum computation can be reconstructed from C, since C contains all cells used in this computation. If we attempt to delete i_j, then the only cells of X_{i_j} that were modified by a previous deletion operation are contained in C. Thus, every such deletion operation can be simulated with access to C (See Figure 2).

C can be described using at most $2t \cdot w$ bits, yet C encodes an arbitrary subset of p elements. This proves the claim. ◀

By the Flower-Lemma (Lemma 9), the sequence $\mathcal{X}$ has a $2t$-flower with $n^{\frac{1}{2t+1}}$ petals. So, we get

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

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

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

Proposition 6 implies the desired bound on t. ◀

Next we prove a similar result for computing the median.

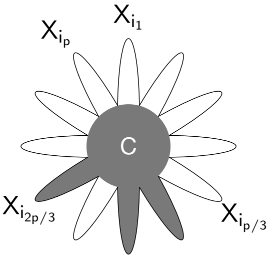


Figure 3 C denotes the core of the Flower, and the shaded cells are the only cells accessed when inserting S.

Theorem 11. *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 $2t$. Without loss of generality, we can assume that each X_i is of size *exactly* $2t$. The key observation is that there cannot be a large $2t$-flower in $\mathcal{X}$:

Claim 12. *If $\mathcal{X}$ has a $2t$-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 $2t$-flower with $i_1 < i_2 < \cdots < i_p$, and $p = 6wt + 3$. Then let S be any subset of $\{i_{p/3+1}, i_{p/3+2}, \cdots, i_{2p/3}\}$ and C denote the the contents of the core of the $2t$-flower after inserting elements of S into the data structure (see Figure 3).

We show that C serves as an encoding of S. This is because C 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. These operations determine the elements in S between its smallest element and median. By the definition of the flower, the only cells of $X_{i_1}, \ldots, X_{i_{p/3}}$ that were accessed when S was inserted are contained in C. Therefore, the sequence of operations can be simulated using C (see Figure 3). Similarly, executing the following operations helps retrieve elements in S 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.

C can be described using at most $2t \cdot w$ bits, yet C encodes a subset of $p/3 = (2tw + 1)$ elements. This proves the claim. ◀

By the Flower-Lemma (Lemma 9), the sequence $\mathcal{X}$ has a $2t$-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

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

Proposition 6 implies the desired bound on t. ◀

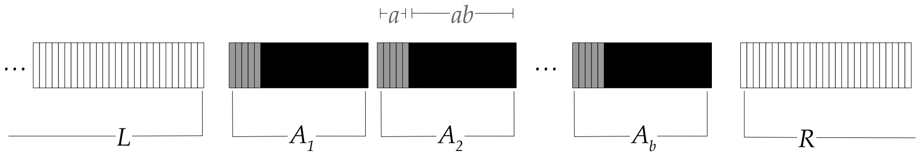


Figure 4 The elements corresponding to petals are partitioned into disjoint intervals $L, A_1, \ldots, A_q, R$. T is the set of black elements. S_i is a random subset of the i'th gray elements from each interval A_j.

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

▶ **Theorem 13.** *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 $2t$. Without loss of generality, we can assume that each X_i is of size *exactly* $2t$. We first show that the time complexity can be lower bounded in terms of the the number of petals in a $2t$-flower belonging to $\mathcal{X}$.

▶ **Claim 14.** *If $\mathcal{X}$ has a $2t$-flower with p petals, then $p \leq 4tw + 1$.*

Proof. Supposed for the sake of contradiction that the sequence $X_{i_1}, \cdots, X_{i_p}$ is a $2t$-flower and $i_1 < i_2 < \cdots < i_p$, and $p = 4tw + 2$. Let S be any subset of $\{i_1, i_3, \cdots, i_{p-1}\}$ and C denote the contents of the cells in the core after inserting the elements of S.

We show that C serves as an encoding of S. To reconstruct S, it suffices to compute the predecessors of the following elements: $i_2, i_4, \cdots, i_p$. By the definition of the $2t$-flower, the only cells accessed in $X_{i_2}, X_{i_4}, \ldots, X_{i_p}$ during the insertion operations are contained in the core of the $2t$-flower. Therefore, the sequence of predecessor operations can be simulated by access only to the cells in the core.

Hence C encodes S. Since there are 2^{2tw+1} possible sets S, and C can be described using $2tw$ bits, we must have $2tw \geq p/2$. This proves the claim. ◀

By the Flower Lemma 9, the sequence $\mathcal{X}$ has a $2t$-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

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

Proposition 6 implies the desired bound on t. ◀

4 Lower Bounds for Median when Insertions are Non-Adaptive

In this section, we prove Theorem 2. We start by giving an outline of the proof. As before, we first associate every element in $\{1, 2, \cdots, n\}$ with the set of cells that are accessed while inserting the element. We then identify a flower among these sets. Proving a lower bound on the time to compute the median is challenging as the computation is adaptive. We shall have to use the flower found above in a subtle way. We come up with a carefully chosen sequence of insertions, followed by a median computation that recovers the k'th smallest element of the set. The sequence of insertions are performed in batches, and every cell that is not in the core of the flower is associated with the batch number of the insertion operation that last accessed it. Ignoring the cells that belong to the core of the flower, we show that at least one cell associated with every batch is accessed with constant probability. Since these cells are disjoint, this will prove that the time to compute the median is at least a constant fraction of the number of batches. To make the above argument work, we use Shannon entropy to quantify the amount of information that the median computation must recover from the cells associated with each batch of insertions.

We now proceed with the formal proof. Define the sequence of sets $X = X_1, \ldots, X_n$, where

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

By the flower lemma (Lemma 9), this sequence of sets must contain a t_{ins}-flower with $p = n^{1/(t_{\mathsf{ins}}+1)}$ petals, and without loss of generality, we assume that the petals are $X_1, \ldots, X_p$. Let C denote the core of the t_{ins}-flower.

To carry out the proof, we need to carefully define a sequence of operations that insert a subset of the elements $\{1, 2, \ldots, p\}$[7]. For parameters a, b, let $L, A_1, \ldots, A_b, R \subseteq \{1, 2, \ldots, p\}$ be consecutive disjoint intervals in ascending order, such that L is of size $p/3$, R is of size $p/3$ and for each i, A_i is of size $a + ab$, and $b(a + ab) \leq p/3$. See Figure 4. Let $S_1, \ldots, S_a$ be independently sampled sets, such that S_i is a uniformly random subset of $\{j : j \text{ is the } i\text{'th element of } A_r \text{ for some } r\}$. So each S_i is a subset of the gray elements in Figure 4. Finally, let T be the set

$$T = \{j : \text{for some } i \in [b],\ j \in A_i \text{ and } j \text{ is not one of the first } a \text{ elements of } A_i\},$$

so T is the set of black elements in Figure 4. Let k be a uniformly random element of $\{a, a + (a + ab), a + 2(a + ab), \ldots, a + (b - 1)(a + ab)\}$.

Consider the following sequence of operations with the data structure:

1. Phase 1:
   a. Insert the elements of T.
   b. Insert the elements of S_1, then the elements of S_2, and so on, until S_a has been inserted.
2. Phase 2:
   a. Insert an appropriate number of elements into L or R so that the median of all the elements inserted is the k'th smallest element of $T \cup S_1 \cup S_2 \ldots \cup S_a$.
   b. Compute the median of the inserted set.

We shall prove that the expected number of cells accessed to compute the median must be close to a. In order to prove this, we use ideas inspired by the chronogram approach. Consider the cells accessed during Phase 1. We say that a cell *belongs to* S_i if it is in the set

$$\bigcup_{j : j \text{ is the } i\text{'th element of } A_r \text{ for some } r} X_j \setminus C$$

So, every cell of the data structure can belong to at most one of the sets $S_1, \ldots, S_a$. Moreover, every cell that is accessed when inserting S_i either belongs to S_i or is in the core of the t_{ins}-flower.

[7] This sequence of operations is inspired by an argument in [27]

Define

$$E_i = \begin{cases} 1 & \text{if a cell that belongs to } S_i \text{ is accessed in Phase 2,} \\ 0 & \text{otherwise.} \end{cases}$$

Observe that the insertions in Phase 2(a) never access a cell that belongs to S_i for any i. Since $E_i = 1$ whenever a cell that belongs to S_i is accessed, all such accesses must come from the median computation in Phase 2. Thus, $t_{\mathsf{med}} \geq \sum_{i=1}^{a} \mathbb{E}[E_i]$. We now proceed to lower bound $\sum_{i=1}^{a} \mathbb{E}[E_i]$.

Let C_i denote the contents of the core immediately after S_i was inserted. Let S_i^j denote the set $S_i \cap A_j$ and $S_i^{<j}$ denote the set $S_i^1 \cup S_i^2 \cup \ldots \cup S_i^{j-1}$. Recall that S_{-i} denotes $S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_a$.

▶ **Claim 15.** *The variables S_{-i}, C_i determine the contents, after Phase 1, of all cells that do not belong to S_i.*

Proof. If a cell does not belong to S_i, then there are three possibilities. If it belongs to a set $S_{i'}$ for $i' < i$, then its value can be reconstructed from $S_1, \ldots, S_{i'}$. If it belongs to $S_{i'}$ for $i' > i$, its value can be reconstructed from C_i and $S_{i+1}, \ldots, S_a$. If it does not belong to any set, then if it is in the core, it is determined by C_i and $S_{i+1}, \ldots, S_a$, and if it is not in the core, its value is fixed. ◀

Let $k = a + (j-1)(a+ab)$, so j is a uniformly random number from the set $\{1, 2, \ldots, b\}$.

▶ **Claim 16.** *The k'th smallest element of $T \cup S_1 \cup S_2 \ldots \cup S_a$ computed in Phase 2 and S_{-i} together determine $\sum_{\ell=1}^{j} |S_i^\ell|$.*

Proof. The k'th smallest element of $T \cup S_1 \cup S_2 \ldots \cup S_a$ is e if and only if the number of elements in $A_1 \cup A_2 \ldots \cup A_b$ that are less than e and missing in $T \cup S_1 \cup S_2 \ldots \cup S_a$ is exactly $e - k$. In other words, the k'th smallest element of $T \cup S_1 \cup S_2 \ldots \cup S_a$ is e if and only if

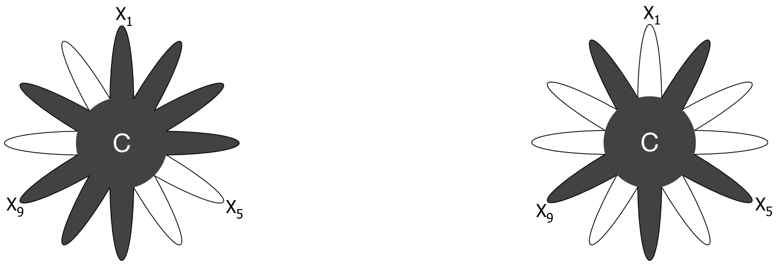
$$|\{j : j < e, j \in (A_1 \cup A_2 \ldots \cup A_b) \setminus (T \cup S_1 \cup S_2 \ldots \cup S_a)\}| = e - k.$$

Let α be the number of elements missing from the intervals $A_1, A_2, \cdots, A_b$, and e be the k'th smallest element of $T \cup S_1 \cup S_2 \ldots \cup S_a$. We know that $0 \leq \alpha \leq ab$, and hence $k \leq e \leq k + ab$. Therefore, the k'th smallest element must be the a'th smallest element in A_j or belong to $T \cap A_j$, and must determine the total number of elements missing before this point. This proves the claim. ◀

▶ **Claim 17.**

$$\mathbb{E}[E_i] \geq \mathop{\mathbb{E}}_{j}\left[\mathsf{H}\left(S_i^j | S_i^{<j}, S_{-i}, C_i, |S_i|\right)\right].$$

Proof. The intuition behind the proof is that in Phase 2, the algorithm starts out knowing only the size of the sets, and learns the k'th smallest element of the sets after computing the median. The contents of all cells needed to insert elements in Phase 2 are determined by S_{-i}, C_i, since these variables determine the cells in the core. By Claim 15, after fixing $S_i^{<j}, S_{-i}, C_i, |S_i|$, all the cells that do not belong to S_i are determined. Thus, after fixing $S_i^{<j}, S_{-i}, C_i, |S_i|$, the value of E_i is determined. Now if $E_i = 0$, then the k'th smallest element is determined, which means that $\mathsf{H}\left(S_i^j | S_i^{<j}, S_{-i}, C_i, |S_i|\right) = 0$. If $E_i = 1$, the inequality holds trivially. ◀



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

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

Figure 5 $S \subseteq \{1,5,9\}$. Cells in petal X_i are shaded black when $\mathsf{Pred}'(i) \neq \mathsf{Pred}(i)$.

Recall that $t_{\mathsf{med}} \geq \sum_{i=1}^{a} \mathbb{E}[E_i]$. Then by the above claim, linearity of expectation and the chain rule for entropy, we have:

$$\begin{aligned} t_{\mathsf{med}} \geq \sum_{i=1}^{a} \mathbb{E}[E_i] \geq \sum_{i=1}^{a} \mathbb{E}_{j}\left[\mathsf{H}(S_i^j | S_i^{<j}, C_i, S_{-i}, |S_i|)\right] &= (1/b)\sum_{i=1}^{a} \mathsf{H}(S_i | C_i, S_{-i}, |S_i|) \\ &\geq (1/b)\sum_{i=1}^{a} \mathsf{H}(S_i|S_{-i}) - \mathsf{H}(C_i, |S_i|) \\ &\geq a \cdot \left(1 - \frac{t_{\mathsf{ins}} w + \log b}{b}\right), \end{aligned} \tag{1}$$

where the last inequality follows from the facts that

$$\mathsf{H}(S_i|S_{-i}) = \mathsf{H}(S_i) = b, \text{ and } \mathsf{H}(C_i, |S_i|) \leq \mathsf{H}(C_i) + \mathsf{H}(|S_i|) \leq w t_{\mathsf{ins}} + \log b.$$

Set $b = 4wt_{\mathsf{ins}}$ and a to be the largest integer such that $a \leq \frac{p}{3b(b+1)}$. Since $b \geq 4$, $\frac{\log b}{b} \leq \frac{1}{2}$. Now, (1) implies that

$$t_{\mathsf{med}} \geq a/4 \geq \Omega\left(\frac{n^{1/(t_{\mathsf{ins}}+1)}}{w^2 \cdot t_{\mathsf{ins}}^2}\right),$$

where the last inequality follows from the fact that $a \geq \frac{p}{3b(b+1)} - 1$.

5 Lower Bounds for Predecessor Search when Predecessors are Non-Adaptive

In this section we prove Theorem 3. 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 9), $\mathcal{X}$ contains a t_{pred}-flower with $n^{\frac{1}{t_{\mathsf{pred}}+1}}$ petals. Let a be the largest even integer such that $a(a+1) \leq n^{\frac{1}{t_{\mathsf{pred}}+1}}$. Note that $a \geq \frac{n^{\frac{1}{2(t_{\mathsf{pred}}+1)}}}{2}$. For ease of notation, we assume that $X_1, X_2 \cdots, X_{a(a+1)}$ are the promised t_{pred}-flower.

Let S be any subset of $\{i | i = (j-1)(a+1)+1 \text{ for some } j \in [a]\}$. Insert all elements of S. For $j \in [a(a+1)]$, let $\mathsf{Pred}'(j)$ be the value obtained by simulating the predecessor computation assuming that the cells outside the core were never accessed when S was inserted. Note that $\mathsf{Pred}'(j)$ can be computed from the cells in the core. Let $\mathsf{Pred}(j)$ be the predecessor of j. For every $i \in [a]$, define

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

Figure 5 shows an example with $a = 3$. Since $|S| \leq a$ and the total number of cells accessed while inserting S is atleast $\frac{a}{2} \cdot \sum_{i=1}^{a} Z_i$,

$$\sum_{i=1}^{a} Z_i \cdot (a/2) \leq t_{\mathsf{ins}} \cdot a. \tag{2}$$

Let C denote the contents of the core after inserting elements of S, the names of the elements i with $Z_i = 1$, and whether or not $i \in S$ for every element with $Z_i = 1$. In other words, C encodes the core, the set $\{i : Z_i = 1\}$ and the set $S \cap \{i : Z_i = 1\}$.

▶ **Lemma 18.** *C encodes S.*

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

We now analyze the length of the encoding of C. The contents of the core can be described with wt_{pred} bits. It takes at most $2\log a$ bits to encode $|\{i|Z_i = 1\}|$ and $|S \cap \{i|Z_i = 1\}|$. Given their sizes, it takes $\log\binom{a}{\sum_{i=1}^{a} Z_i}$ bits to encode $\{i|Z_i = 1\}$, and $\log\binom{a}{|S\cap\{i|Z_i=1\}|}$ bits to encode $S \cap \{i|Z_i = 1\}$. Therefore, the length of the encoding is at most

$$wt_{\mathsf{pred}} + \log\binom{a}{\sum_{i=1}^{a} Z_i} + \log\binom{a}{|S\cap\{i|Z_i=1\}|} + 2\log a.$$

Since there are 2^a possible sets S, we must have

$$a \leq wt_{\mathsf{pred}} + \log\binom{a}{\sum_{i=1}^{a} Z_i} + \log\binom{a}{|S\cap\{i|Z_i=1\}|} + 2\log a. \tag{3}$$

Observe that either $t_{\mathsf{ins}} \geq \frac{a}{64}$ or not. In the former case, since $a \geq \frac{n^{\frac{1}{2(t_{\mathsf{pred}}+1)}}}{2}$, we can conclude that $t_{\mathsf{ins}} \geq \Omega\left(n^{\frac{1}{2(t_{\mathsf{pred}}+1)}}\right)$. In the latter case, Equation 2 implies that $\sum_{i=1}^{a} Z_i \leq a/32$. Note that $\binom{a}{|S\cap\{i|Z_i=1\}|} \leq \binom{a}{\sum_{i=1}^{a} Z_i}$ when $\sum_{i=1}^{a} Z_i \leq a/32$. Using Proposition 7, we get

$$\log\binom{a}{\sum_{i=1}^{a} Z_i} + \log\binom{a}{|S\cap\{i|Z_i=1\}|} \leq 2\left(\sum_{i=1}^{a} Z_i\right) \cdot \log\frac{ea}{\sum_{i=1}^{a} Z_i} \leq \frac{a}{2},$$

where the last inequality follows from the fact that $\sum_{i=1}^{a} Z_i \leq a/32$. After rearranging (3), the previous inequality implies that $t_{\mathsf{pred}} \geq \frac{a}{2w} - \frac{2\log a}{w}$. Since $a \geq \frac{n^{\frac{1}{2(t_{\mathsf{pred}}+1)}}}{2}$, we can conclude that $t_{\mathsf{pred}} \cdot \log(wt_{\mathsf{pred}}) \geq \Omega(\log n)$. Using Proposition 6, we obtain the desired lower bound of $t_{\mathsf{pred}} \geq \Omega\left(\frac{\log n}{\log\log n + \log w}\right)$.

References

1 Miklós Ajtai. A lower bound for finding predecessors in yao's call probe model. *Combinatorica*, 8(3), 1988.

2 Noga Alon and Ravi B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

3 Noga Alon and Uriel 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.

4 Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *JCSS: Journal of Computer and System Sciences*, 65, 2002.

5 Paul Beame, Vincent Liew, and Mihai Patrascu. Finding the median (obliviously) with bounded space. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 103–115, 2015.

6 Joseph Boninger, Joshua Brody, and Owen Kephart. Non-adaptive data structure bounds for dynamic predecessor. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, pages 20:1–20:12, 2017.

7 Gerth Stølting Brodal, Shiva Chaudhuri, and Jaikumar Radhakrishnan. The randomized complexity of maintaining the minimum. In *SWAT: Scandinavian Workshop on Algorithm Theory*, 1996.

8 Joshua Brody and Kasper Green Larsen. Adapt or die: Polynomial lower bounds for non-adaptive dynamic data structures. *Theory of Computing*, 11:471–489, 2015.

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

10 Timothy M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6(2), 2010.

11 Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *Journal of London Mathematical Society*, 35:85–90, 1960.

12 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. `doi:10.1145/256303.256309`.

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

14 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *JCSS: Journal of Computer and System Sciences*, 47, 1993.

15 Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. *Theor. Comput. Sci.*, 379(3):405–417, 2007. `doi:10.1016/j.tcs.2007.02.047`.

16 Mohit Garg and Jaikumar Radhakrishnan. Set membership with non-adaptive bit probes. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 38:1–38:13, 2017.

17 Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2000.

18 Kasper Green 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.

19 Peter Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. 57:37–49, 1 1998.

20 Peter Bro 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, 1994. ACM Press.

21 J. Ian Munro and Venkatesh Raman. Selection from read-only memory and sorting with minimum data movement. *TCS: Theoretical Computer Science*, 165, 1996.

22 Rina Panigrahy, Kunal Talwar, and Udi Wieder. Lower bounds on near neighbor search via metric expansion. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 805–814, Washington, DC, USA, 2010. IEEE Computer Society.

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

24 Mihai Pătraşcu and Erik 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.

25 Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 232–240, 2006.

26 Mihai Pătraşcu and Mikkel 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.

27 Mihai Patrascu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 166–175, 2014.

28 Pranab Sen and Srinivasan Venkatesh. Lower bounds for predecessor searching in the cell probe model. *J. Comput. Syst. Sci*, 74(3):364–385, 2008.

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

30 Omri Weinstein and Huacheng Yu. Amortized dynamic cell-probe lower bounds from four-party communication. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 305–314, 2016.

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

32 Andrew Yao. Should tables be sorted? *JACM: Journal of the ACM*, 28, 1981.

33 Huacheng 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.

A A Data Structure 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 [31]). This data structure matches many of the lower bounds in our proofs.

▶ **Theorem 19.** *There is a data structure that maintains a subset of $\{1, 2, \ldots, n\}$ and supports insertions, deletions and computing the median, minimum, 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.*

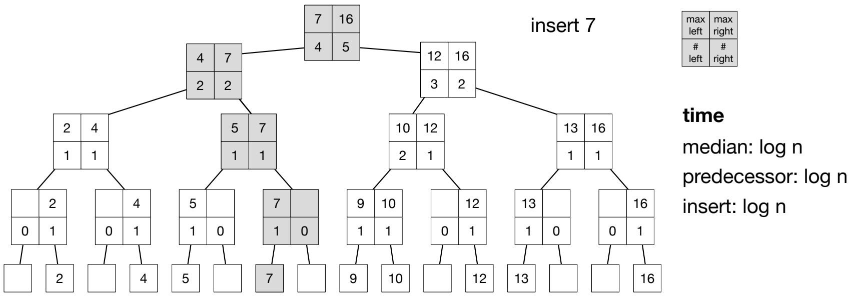


Figure 6 A data structure based on binary search trees storing the set $\{2, 4, 5, 7, , 9, 10, 12, 13, 16\}$.

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 6 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 or minimum, we read the cells associated with the root to determine if the desired value 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 or minimum. 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. ◀