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

Sivaramakrishnan Natarajan Ramamoorthy, Anup Rao
University of Washington
What is the most efficient way to maintain a set of numbers from \( \{1, 2, \ldots, n\} \)?

- insert new elements
- compute the median
- compute the predecessors
Universe is \([n]\)
\(S=\{1,3,5,6,11,13,16\}\)

Predecessor\((x) = \text{Max } \{y \leq x\} \text{ subject to } y \in S\)
Universe is $[n]$
$S=\{1,3,5,6,11,13,16\}$

**Sorted Array**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>13</td>
<td>16</td>
</tr>
</tbody>
</table>

$\text{size} = 7$

**Time Complexity:**
- insert: $n$
- median: $2$
- predecessor: $\log n$

$\text{Predecessor}(x) = \text{Max } \{y \leq x\} \text{ subject to } y \in S$
Universe is \([n]\)
\(S=\{1,3,5,6,11,13,16\}\)

Sorted Array

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>6</th>
<th>11</th>
<th>13</th>
<th>16</th>
</tr>
</thead>
</table>

size 7

Predecessor \(x\) = \(\text{Max}\ \{y \leq x\} \text{ subject to } y \in S\)

Bit Vector

| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

Time Complexity:
- insert: \(n\)
- median: 2
- predecessor: \(\log n\)

Time Complexity:
- insert: 1
- median: \(n\)
- predecessor: \(n\)
Universe is [n]
S={1,3,5,6,11,13,16}

Sorted Array

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>5</th>
<th>6</th>
<th>11</th>
<th>13</th>
<th>16</th>
</tr>
</thead>
</table>
size | 7 |

Bit Vector

1 0 1 0 1 1 0 0 0 0 0 1 0 1 0 0 1

Predecessor Array

1 1 3 3 5 6 6 6 6 6 6 11 11 13 13 13 16

Predecessor(x) = Max \{y \leq x\} subject to y \in S

Time Complexity:
insert: n
median: 2
predecessor: log n

Time Complexity:
insert: 1
median: n
predecessor: n

Time Complexity:
insert: n
median: n
predecessor: 1
Binary Search Trees

insertions, median and predecessor in O(log n) time
Binary Search Trees

Insertions, median and predecessor in O(log n) time
Binary Search Trees

S={1,3,5,6,11,13,16}

Operations:
median
insertions, median and predecessor in $O(\log n)$ time
Binary Search Trees

S = \{1, 3, 5, 6, 11, 13, 16\}

Operations:
- median
- output = 6

insertions, median and predecessor in \(O(\log n)\) time
Binary Search Trees

$S = \{1, 3, 5, 6, 11, 13, 16\}$

Operations:
- predecessor(15)

insertions, median and predecessor in $O(\log n)$ time
Binary Search Trees

S = \{1,3,5,6,11,13,16\}

Operations:
predecessor(15) output = 13

insertions, median and predecessor in O(log n) time
van Emde Boas Algorithm[77]

$S = \{1, 3, 11, 13, 16\}$

Recursion: $T(n) = T(\sqrt{n}) + O(1)$ (at every level, recurse on $S_i$ or $S'$)

insertions, median and predecessor in $O(\log \log n)$ time
Cell Probe Model [Yao81]

Data structure is a collection of cells that stores the data

**Update Algorithm:**
access subset of cells, modifying some to reflect changes

\[ time = \#\text{cells accessed during update} \]
Cell Probe Model [Yao81]

Data structure is a collection of cells that stores the data

**Update Algorithm:**
access subset of cells, modifying some to reflect changes

\[
time = \#\text{cells accessed during update}
\]

**Query Algorithm:**
access subset of cells and outputs answer

\[
time = \#\text{cells accessed during query}
\]
An operation is non-adaptive if locations of cells accessed depends only on the operation and not on contents of the cell.
An operation is non-adaptive if locations of cells accessed depends only on the operation and not on contents of the cell.
An operation is *non-adaptive* if locations of cells accessed depends only on the operation and not on contents of the cell.
Adaptive Queries and Updates

\[ S = \{1, 3, 11, 13, 16\} \]

Insertions, median and predecessor in \( O(\log \log n) \) time
Advantages of Non-Adaptivity

• Non-adaptive data structures are simpler.

• An implementation perspective: can load all relevant cells into cache at one go.
### Some Known Dynamic Data Structure Lower Bounds

<table>
<thead>
<tr>
<th>Adaptive updates and queries</th>
<th>Dynamic connectivity with edge insertions, Parity Sum</th>
<th>$t_q \geq \frac{\log n}{\log wt_u}$</th>
<th>[FredmanSaks89] chronogram technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k^{\text{th}}$ smallest</td>
<td>$t_q \geq \frac{\log n}{\log wt_u}$</td>
<td>[PatrascuThorup14] reduction to Parity Sum</td>
<td></td>
</tr>
<tr>
<td>Dynamic connectivity</td>
<td>$\max{t_u, t_q} \geq \log n$</td>
<td>[PatrascuDemaine04] information transfer technique</td>
<td></td>
</tr>
<tr>
<td>with edge insertions +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deletions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2D range counting</td>
<td>$t_q \geq \frac{\log^2 n}{\log^2 wt_u}$</td>
<td>[Larsen12] cell sampling + chronogram</td>
<td></td>
</tr>
<tr>
<td>2D range counting (amortized</td>
<td>$t_q \geq \frac{\log^2 n}{\log^2 wt_u}$</td>
<td>[WeinsteinYu16] a new communication model</td>
<td></td>
</tr>
<tr>
<td>+ large error)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-Adaptive</th>
<th>directed graph reachability with non-adaptive queries</th>
<th>$\max{t_u, t_q} \geq n/w$</th>
<th>[BrodyLarsen12]</th>
</tr>
</thead>
</table>
## Non-Adaptive Lower Bounds

### Minimum with deletions

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Lower Bound</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-adaptive delete and</td>
<td>[ t_{\text{del}} + t_{\text{min}} \geq \frac{\log n}{\log \log n + \log w} ]</td>
<td>This work</td>
</tr>
<tr>
<td>minimum operations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Median

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Lower Bound</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-adaptive insert and median</td>
<td>[ t_i + t_m \geq \frac{\log n}{\log \log n + \log w} ]</td>
<td></td>
</tr>
<tr>
<td>operations</td>
<td></td>
<td>PatrascuThorup06</td>
</tr>
</tbody>
</table>

### Predecessor Search

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Lower Bound</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-adaptive inserts</td>
<td></td>
<td>This work</td>
</tr>
</tbody>
</table>

### Bit vectors

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Lower Bound</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum with deletions</td>
<td></td>
<td>BoningerBrodyKephart17</td>
</tr>
</tbody>
</table>

### Binary trees

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Lower Bound</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### vEB

<table>
<thead>
<tr>
<th>Operation Type</th>
<th>Lower Bound</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### This work

- Adaptive insert and predecessor operation
- Even super constant open

---

**Bit vectors:** \( t_i = 1, t_m = n \)

**Predecessor Arrays:**

\[ t_i = n, t_p = 1 \]

**Binary trees:** \( t_i = \log n, t_p = \log n \)

**vEB:** \( t_i = \log \log n, t_p = \log \log n \)
Results for Median in Other Models

• Time Space trade offs.
  [MunroRaman96, Chan10], [BeameLiewPatrascu15]

• Streaming Algorithms.
  [ChakrabartiJayramPatrascu08]

• Data Structures in the Comparison Model.
  [BrodalChaudriRadhakrishnan96]
  
  • insert time = $t$,

  • median computation $\geq m/4^t$, median computation $\leq m/2^t$
The Key Technical Tool
Sunflower Lemma

For every sequence $X=X_1, X_2, \ldots, X_n$ such that $|X_i| \leq t$, if $n \geq t!(p - 1)^{t+1}$
For every sequence $X=X_1, X_2, \ldots, X_n$ such that $|X_i| \leq t$, if $n \geq t!(p - 1)^{t+1}$, then there is a Sunflower with $p$ petals: after renaming, $\exists C$ such that $X_i \cap X_j = C$ for distinct $i, j \in [p]$.
Sunflowers in Complexity

• Monotone Circuit Lower bounds:
  [Razborov85] [AlonBoppana87]

• Dynamic data structures with $w=1$:
  [FrandsenMiltersenSkyum93]

• Static data structures with $w=1$:
  [GalMiltersen07] (succinct representations)
First Lower Bound for Median

**Theorem 1:** Any data structure for Median on \([n]\) with non-adaptive insert and median operations must have

\[ t_i + t_m \geq \frac{\log n}{\log \log n + \log w} \]

\(w\): word size \hspace{1cm} t_i: \text{time for inserts} \hspace{1cm} t_m: \text{time for median}
Outline of Proof

1. Let $X = X_1, X_2, \ldots, X_n$, where
   
   $X_i = \{ j \mid j \text{ is accessed while inserting } i \text{ or computing median} \}$

2. Identify a Sunflower in $X$ with $p$ petals

3. Use $C$ to encode a random subset of $[p/3, 2p/3]$
Proof of Theorem 1

\[ X_i = \{ j \mid j \text{ is accessed while INSERTING } i \text{ or computing median} \} \]
Proof of Theorem 1

\[ X_i = \{ j \mid j \text{ is accessed while inserting } i \text{ or computing median} \} \]

1. Let \( S \) be a uniformly random subset of middle 1/3rd elements
Proof of Theorem 1

\[ X_i = \{ j \mid j \text{ is accessed while inserting } i \text{ or computing median} \} \]

1. Let \( S \) be a uniformly random subset of middle 1/3rd elements

2. Insert elements of \( S \)
Proof of Theorem 1

$X_i = \{j \mid j \text{ is accessed while inserting } i \text{ or computing median}\}$

1. Let $S$ be a uniformly random subset of middle $1/3$rd elements

2. Insert elements of $S$
Proof of Theorem 1

1. Let $S$ be a uniformly random subset of middle $1/3$rd elements

2. Insert elements of $S$

Claim: Contents of $C$ encode $S$

$X_i = \{j \mid j$ is accessed while inserting $i$ or computing median$\}$
Proof of Theorem 1

1. Let $S$ be a uniformly random subset of middle $1/3$rd elements

2. Insert elements of $S$

Claim: Contents of $C$ encode $S$

a) If $x \in [1,p/3] \cup [2p/3+1,p]$, $x$ can be inserted with access only to $C$. 

$X_i = \{j \mid j$ is accessed while inserting $i$ or computing median$\}$
Proof of Theorem 1

1. Let $S$ be a uniformly random subset of middle 1/3rd elements

2. Insert elements of $S$

**Claim:** Contents of $C$ encode $S$

a) If $x \in [1,p/3] \cup [2p/3+1,p]$, $x$ can be inserted with access only to $C$. 

$X_i = \{ j \mid j$ is accessed while inserting $i$ or computing median $\}$
Proof of Theorem 1

1. Let $S$ be a uniformly random subset of middle $1/3$rd elements

2. Insert elements of $S$

Claim: Contents of $C$ encode $S$

a) If $x \in [1, p/3] \cup [2p/3+1, p]$, $x$ can be inserted with access only to $C$.

b) Repeatedly insert elements from $[1, p/3] \cup [2p/3+1, p]$ and compute the median to recover $S$. 

$X_i = \{j \mid j$ is accessed while inserting $i$ or computing median$\}$
Conclusion

1. If $S$ is a uniformly random subset of $[p/3, 2p/3]$, then $H(S) = p/3$

2. Since contents of C encode $S$, $H(S) \leq w(t_i + t_m)$

3. Therefore, $t_i + t_m \geq p/3w$
First Lower Bound for Median

**Theorem 1:** Any data structure for Median on $[n]$ with non-adaptive insert and median operations must have

$$t_i + t_m \geq \frac{\log n}{\log \log n + \log w}$$

$w$: word size \quad $t_i$: time for inserts \quad $t_m$: time for median
Second Lower Bound for Median

**Theorem 2:** Any data structure for Median on \([n]\) with non-adaptive insertions must have

\[
t_m \geq \frac{n^{t_i+1}}{w^2 \cdot t_i^2}
\]

\(w\): word size \quad t_i: \text{time for inserts} \quad t_m: \text{time for median}
Outline of Proof

Step 1: Like before, identify a Sunflower among
\( X = X_1, X_2, \ldots, X_n \), where \( X_i = \{ j \mid j \text{ is accessed while inserting } i \} \)

\[ X_1 \quad X_p \]
\[ X_{2p/3} \quad X_{p/3} \]

Step 2: median helps recover the \( k^{th} \) smallest

Step 3: lower bound for \( k^{th} \) smallest via chronogram approach
Reduction to $k^{th}$ smallest

Claim: After fixing $C$ and $|S|$, can find $k^{th}$ smallest in time $t_m$

1. If $x \in [1, p/3] \cup [2p/3, p]$, inserting $x$ requires only access to $C$.

$X_i = \{j \mid j$ is accessed while inserting $i\}$
Reduction to $k^{th}$ smallest

Claim: After fixing $C$ and $|S|$, can find $k^{th}$ smallest in time $t_m$.

1. If $x \in [1,p/3] \cup [2p/3,p]$, inserting $x$ requires only access to $C$.

2. Insert elements $\in [1,p/3] \cup [2p/3,p]$ and compute median.

$X_i = \{j \mid j$ is accessed while inserting $i\}$
1. Applying the "log n" lower bound,

\[ t_i \cdot t_m \geq \frac{\log n}{\log wt_i} \]

2. Can be improved to

\[ t_m \geq \frac{n^{\frac{1}{2(t_i+1)}}}{wt_i} \]

Key insight: epoch analysis ([FS89]) exploiting sunflower structure
Open Question

Is the following true?

Any data structure for Median on \([n]\) with \(w = O(\log n)\) and adaptive insert and median operations must have

\[ t_i + t_m \geq \frac{\log \log n}{\log \log \log \log n} \]

**Remark 1:** Proving a super constant lower bound on \(t_i + t_m\) is also open (worst case times assumed).

**Remark 2:** Proving lower bounds for minimum and maximum under deletions is also open.