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

Universe is [n]

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

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

<u>Sorted Array</u>

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

size

7

<u>Time Complexity:</u>
insert: n
median: 2
predecessor: log n

Universe is [n]

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

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

Sorted Array

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

size

7

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

Bit Vector

1	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

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

Universe is [n]

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

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

Sorted Array

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

size

7

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

Bit Vector

1	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**

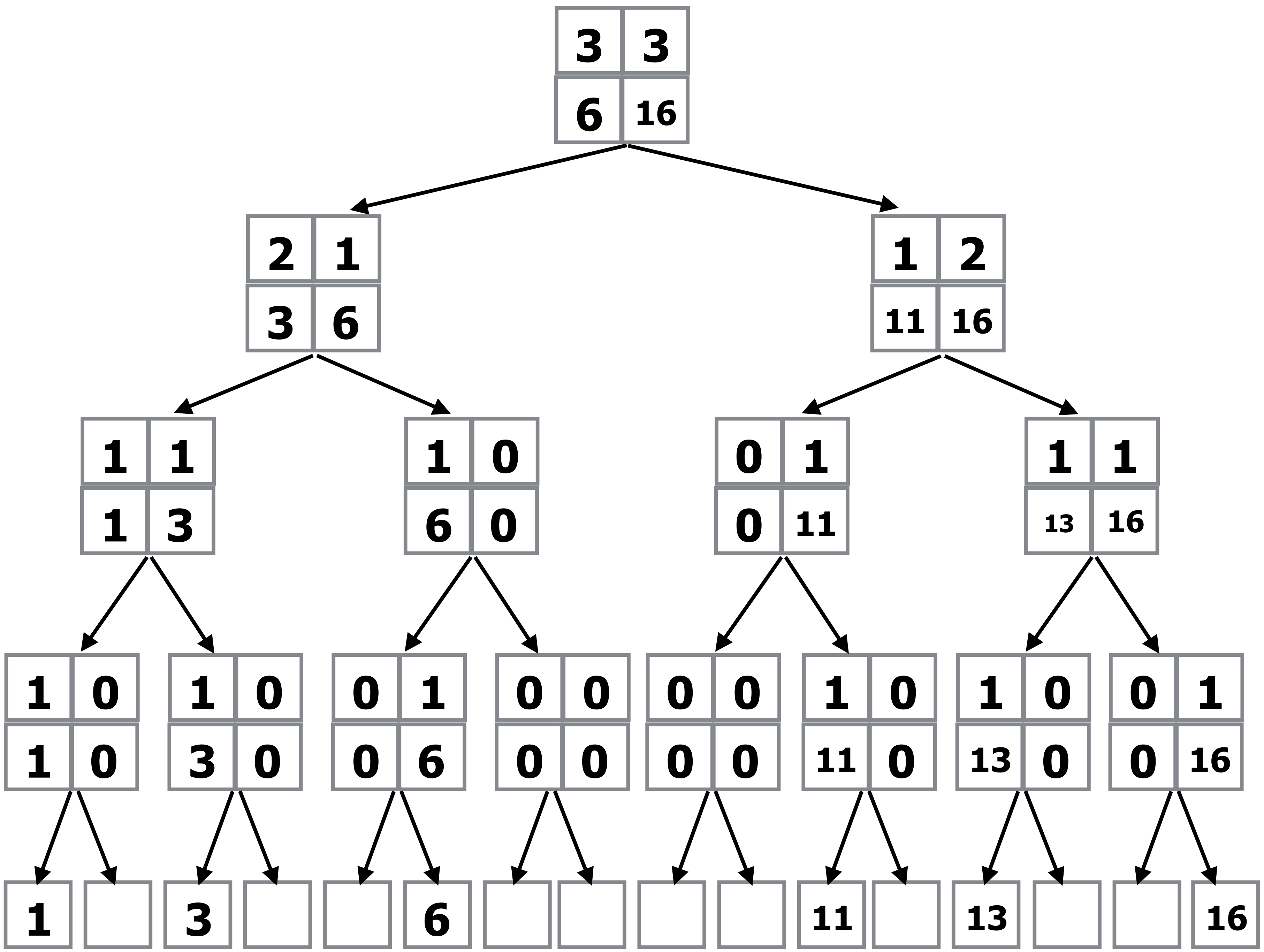
Time Complexity:
insert: 1
median: n
predecessor: n

Predecessor Array

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

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

Binary Search Trees



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

#left	#right
maxL	maxR

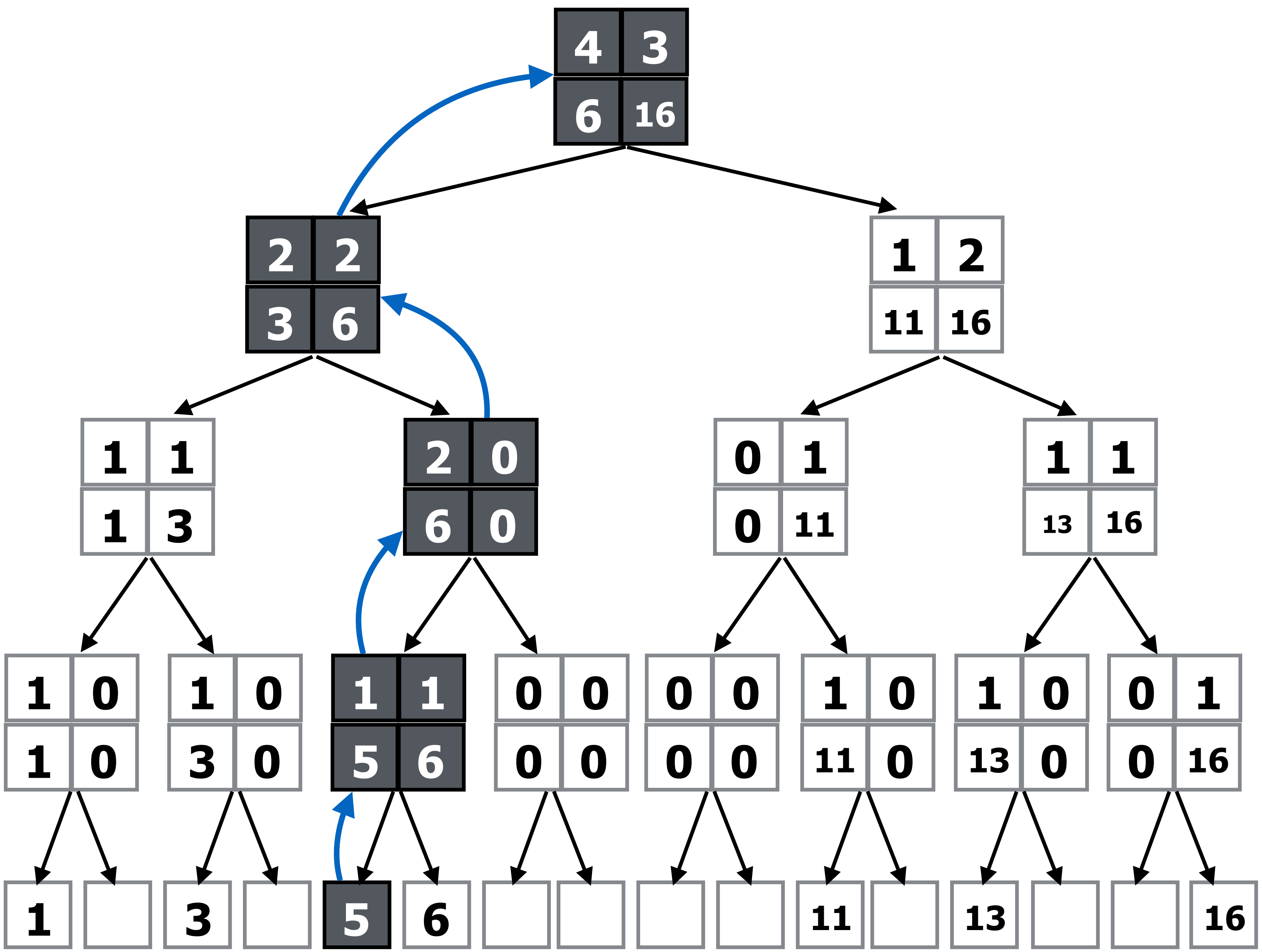
<u>Operations:</u>

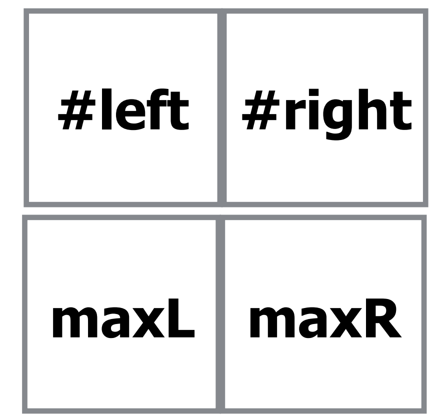
insert 5

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

Binary Search Trees

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



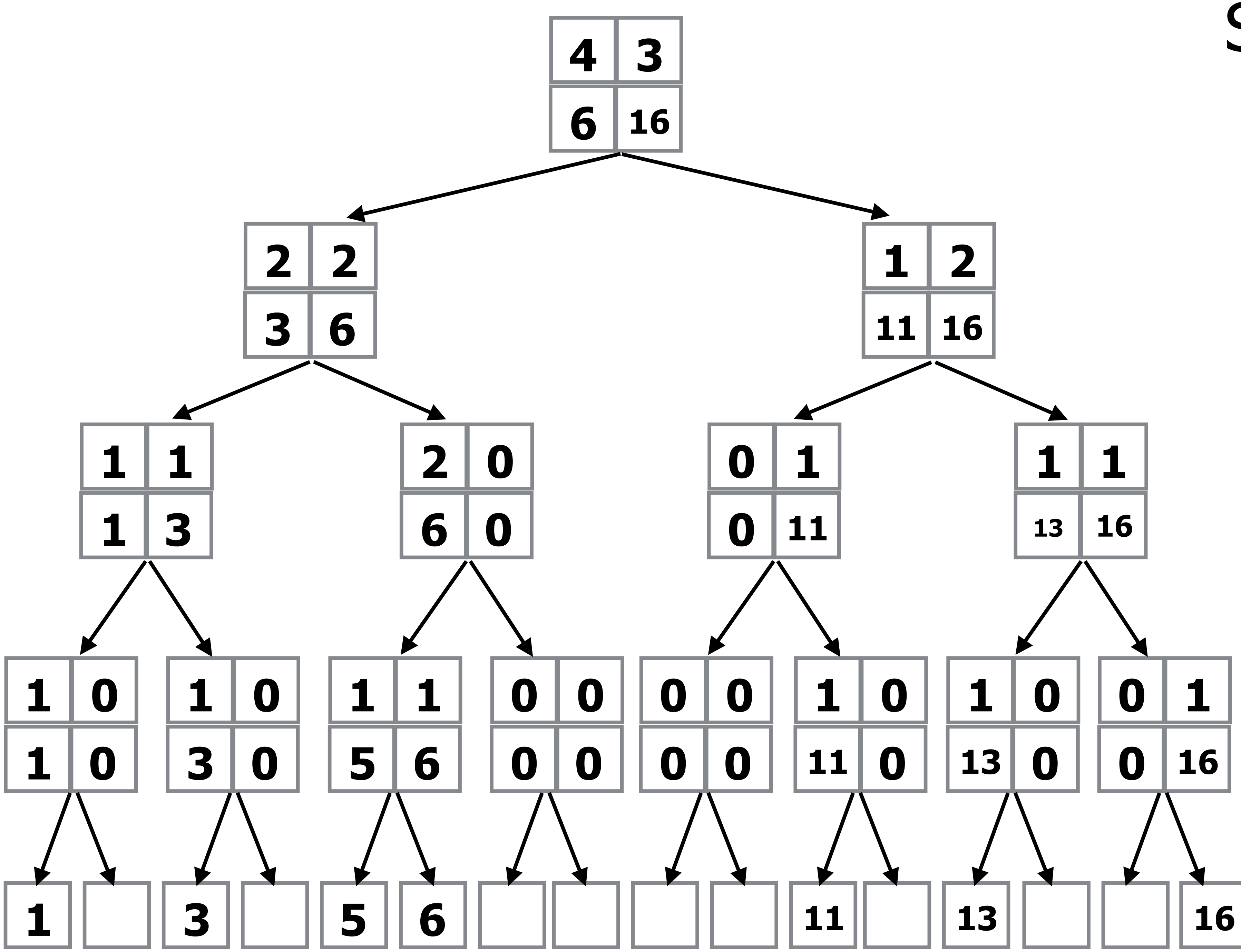


Operations:

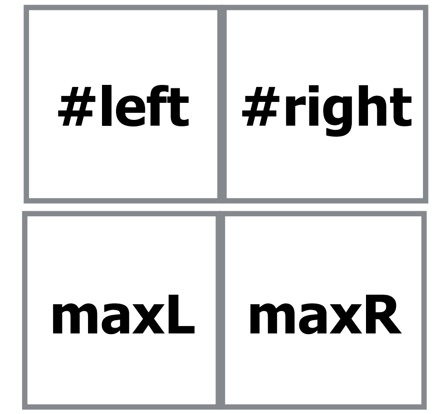
insert 5

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

Binary Search Trees



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

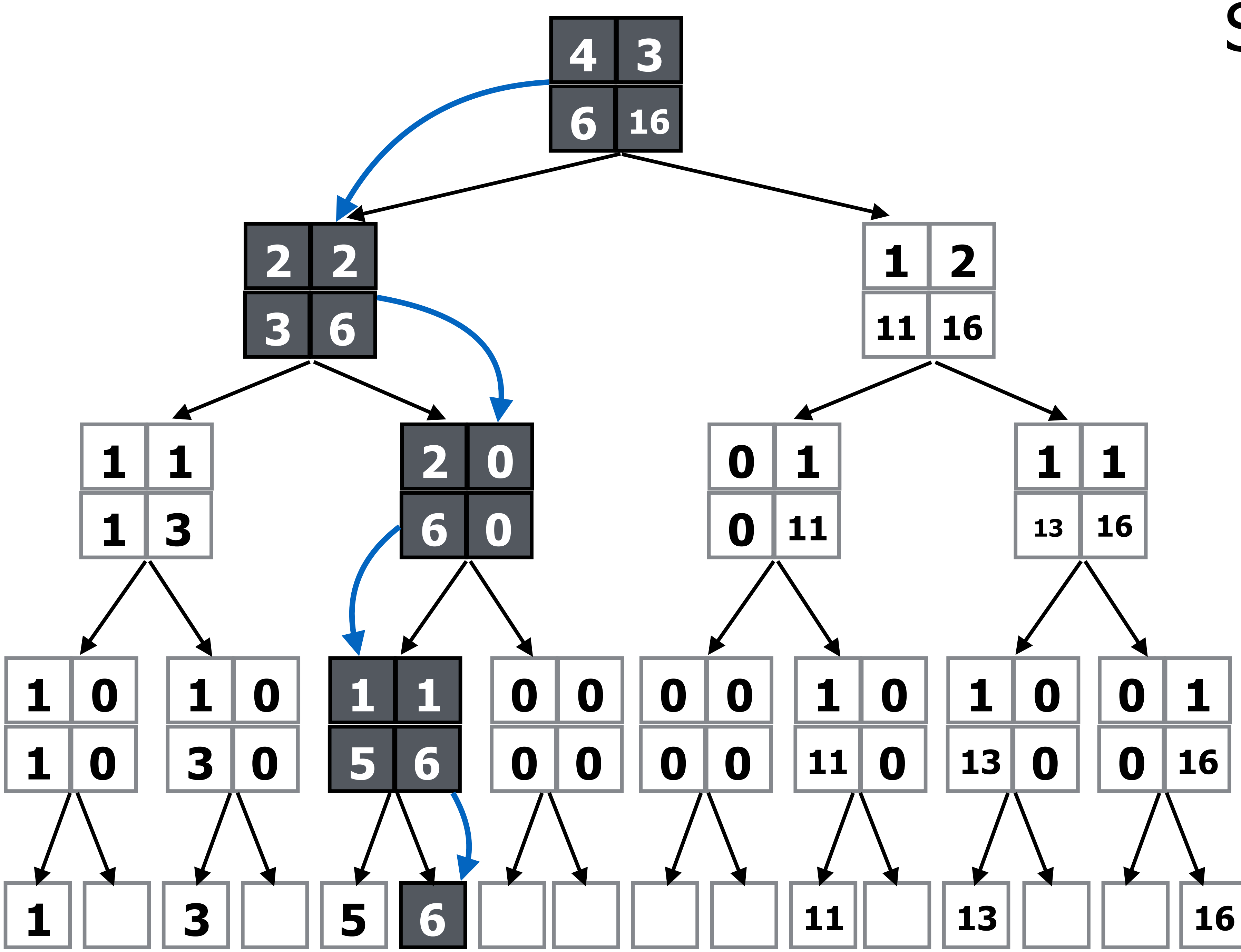


Operations:

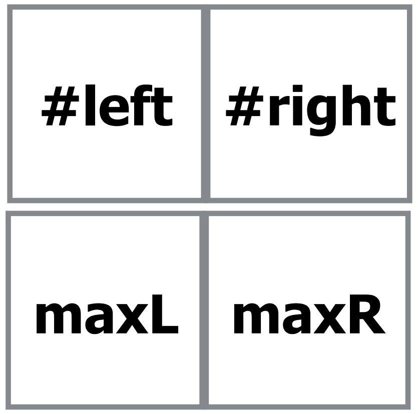
median

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

Binary Search Trees



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



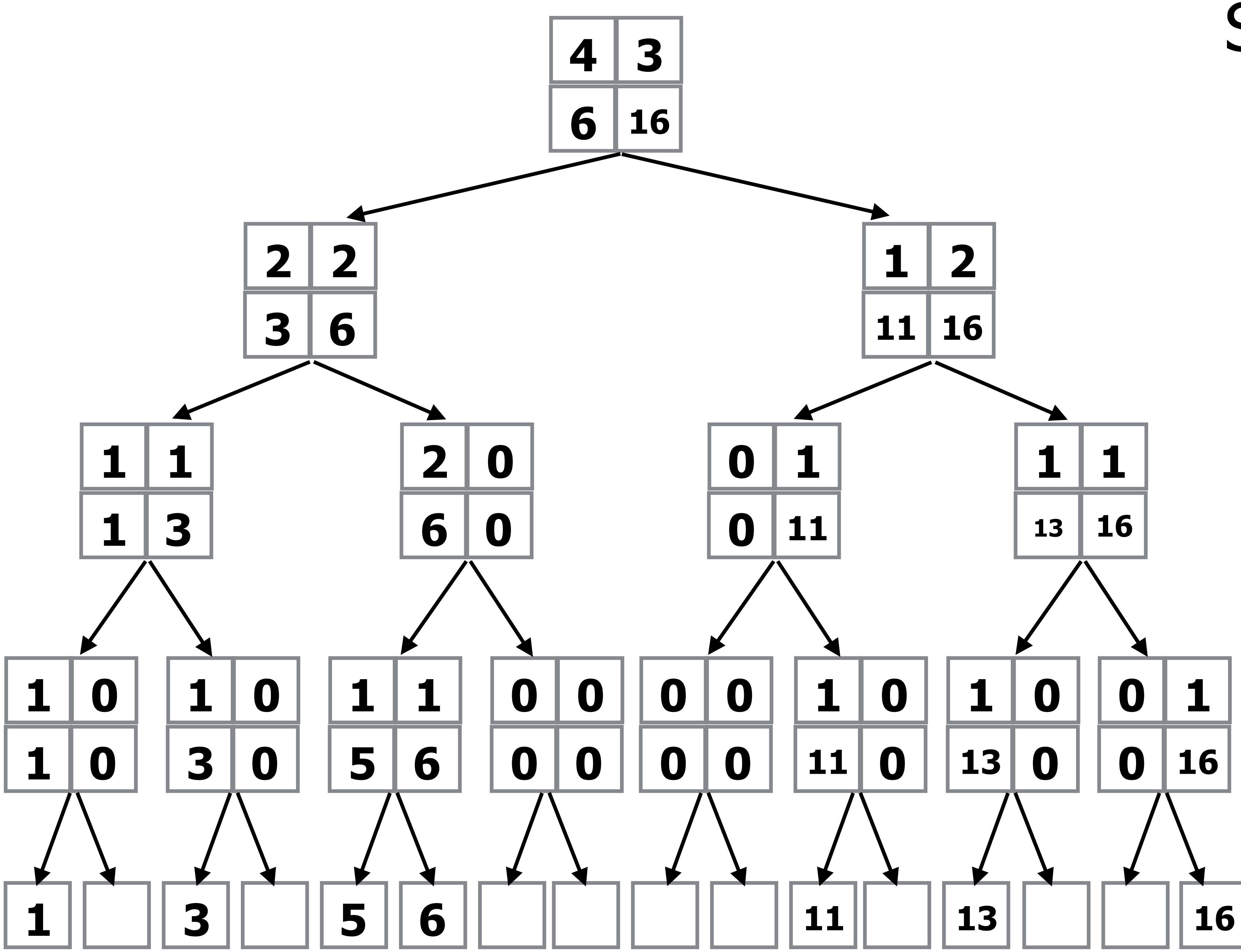
<u>Operations:</u>

median

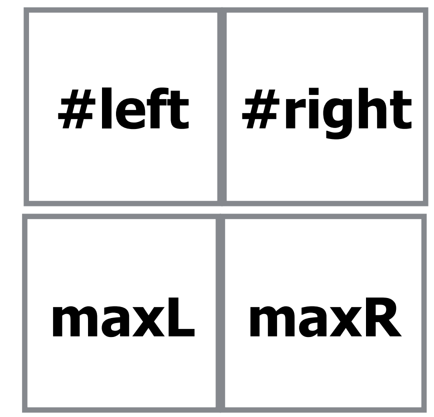
output = 6

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

Binary Search Trees



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

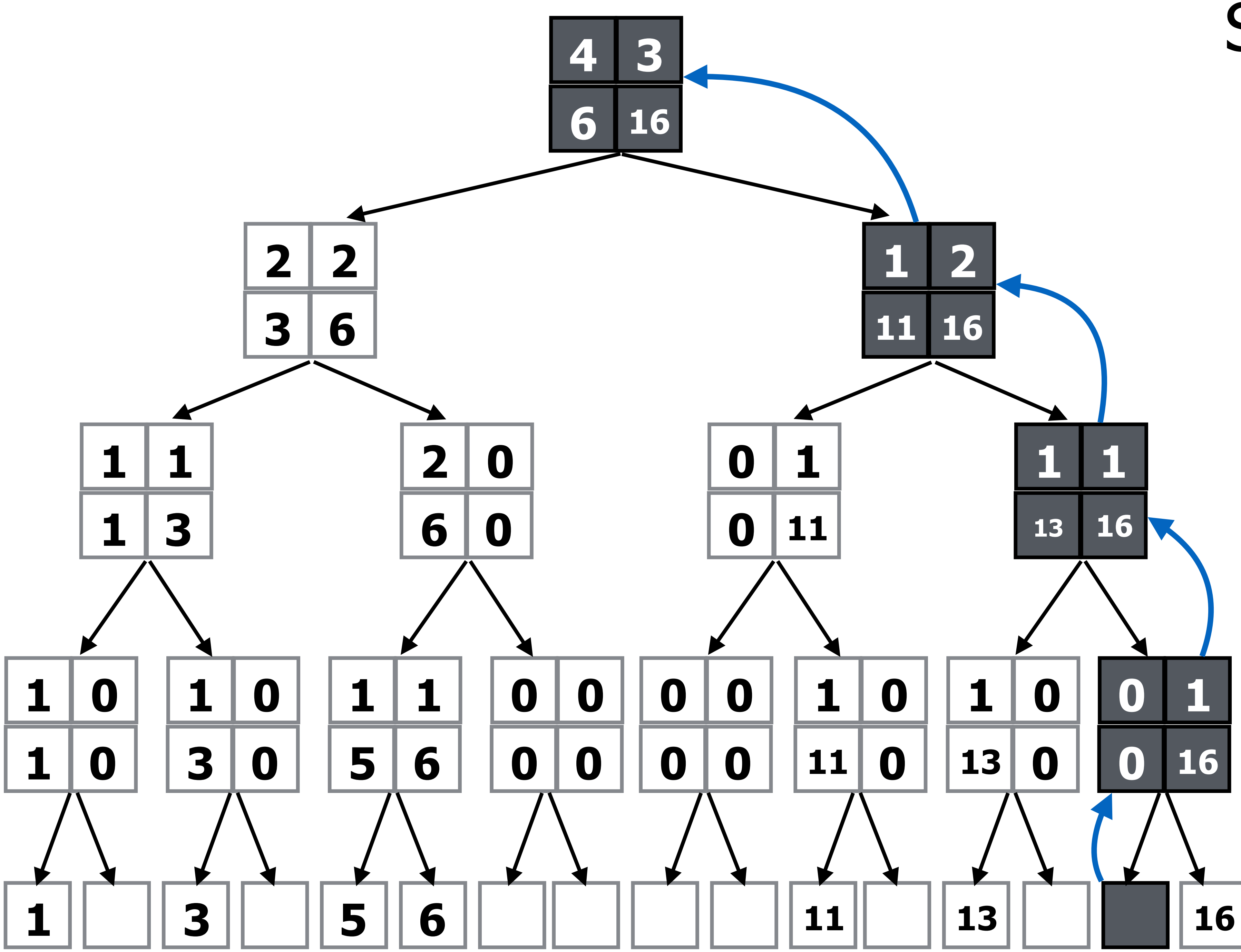


Operations:

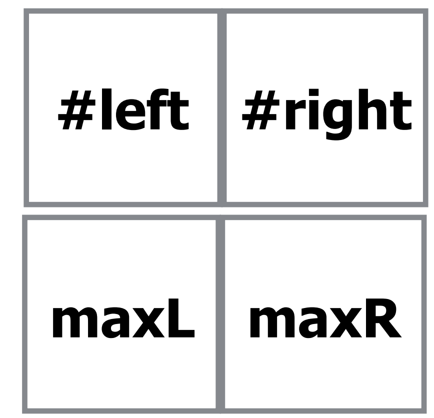
predecessor(15)

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

Binary Search Trees



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



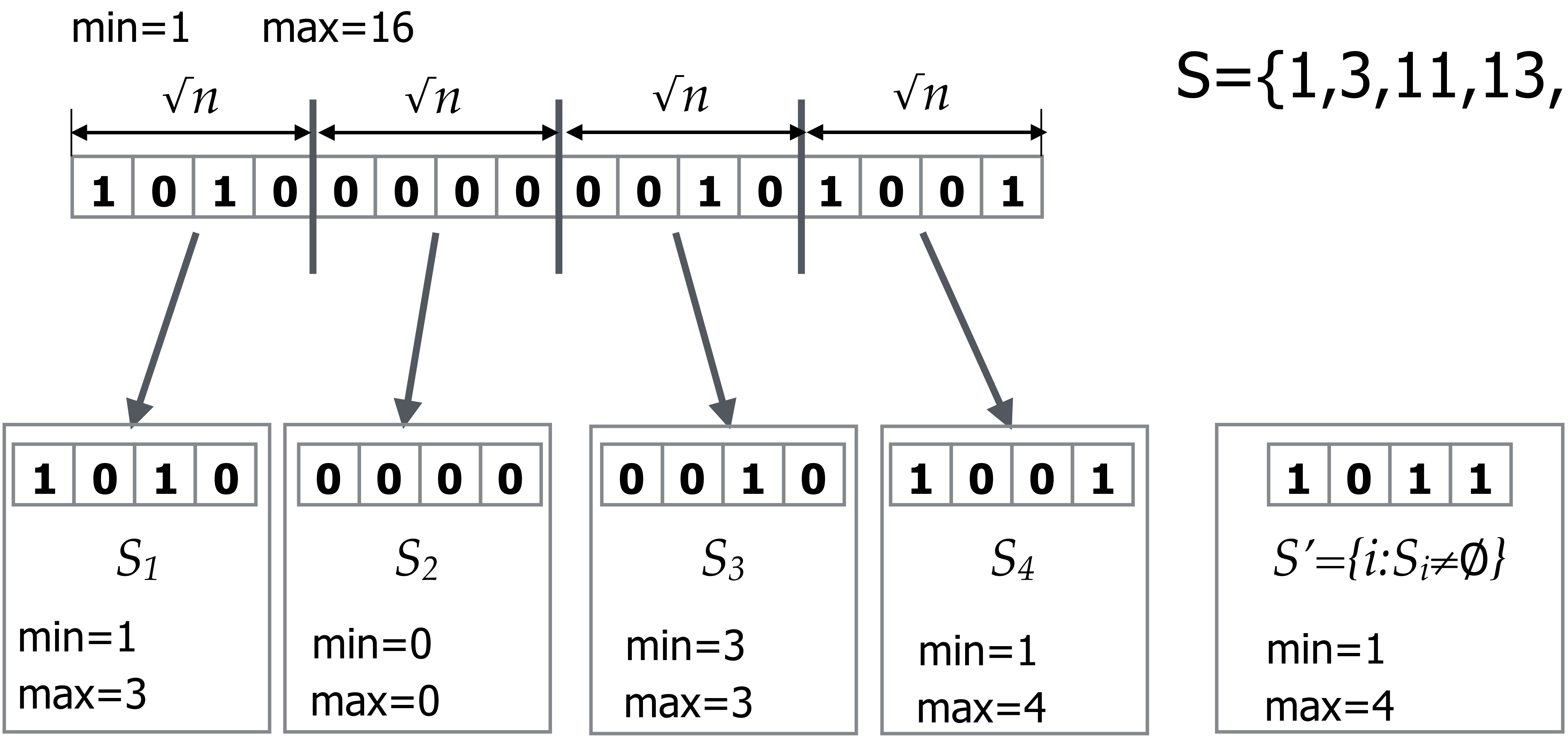
<u>Operations:</u>

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')

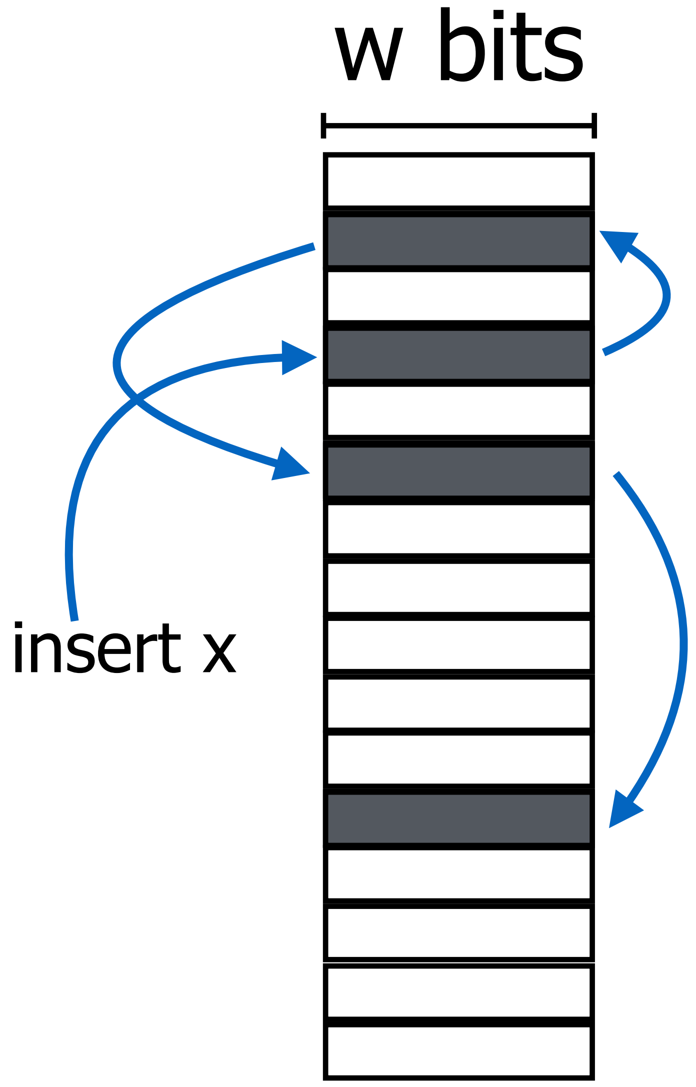
insertions, median and predecessor in O(log log n) time

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

$time$ = #cells accessed during update



Cell Probe Model [Yao81]

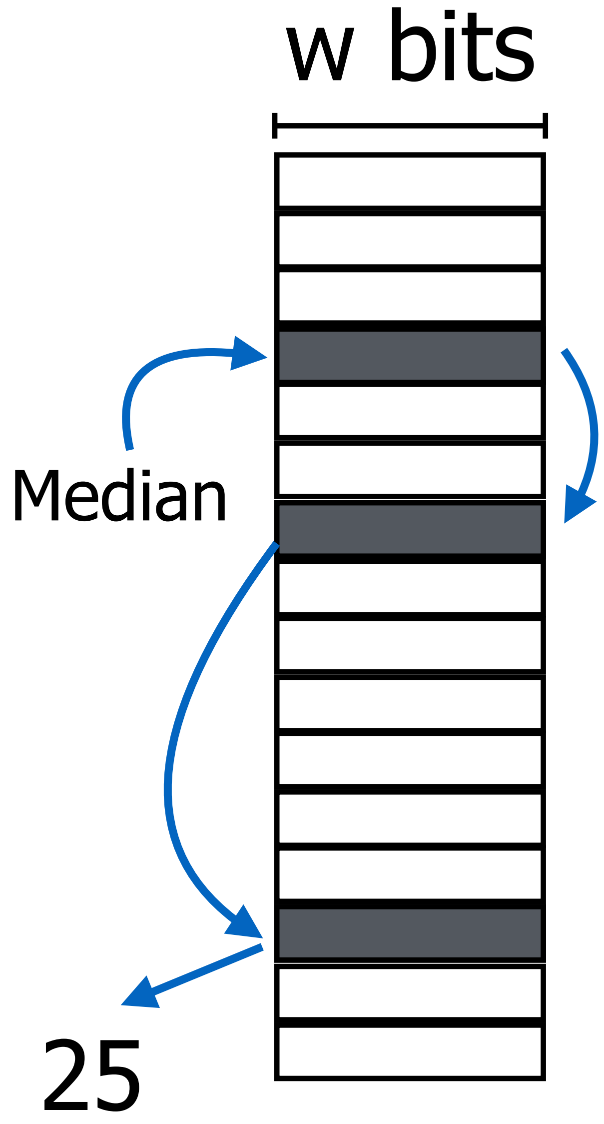
Data structure is a collection of cells that stores the data

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

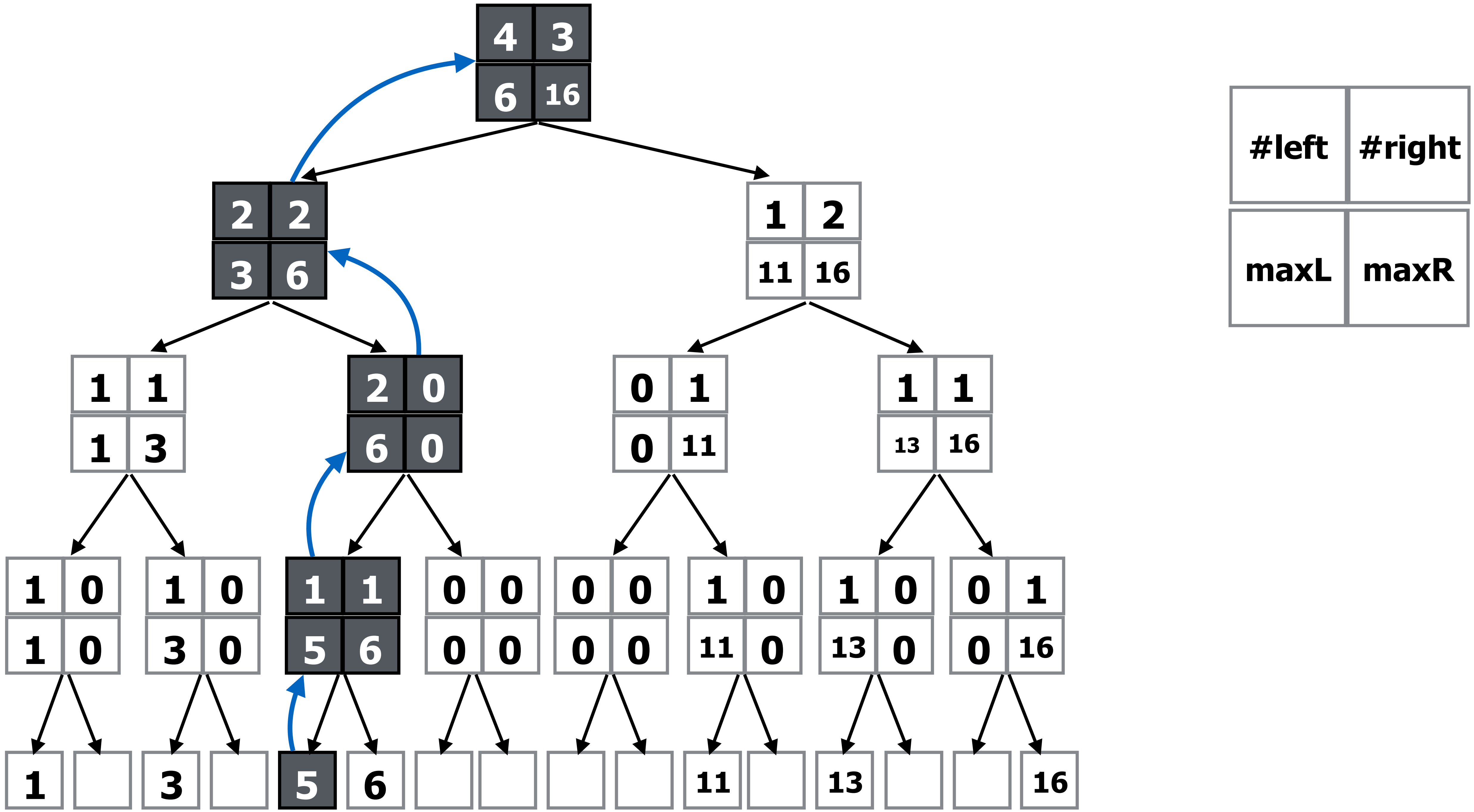
$time$ = #cells accessed during update

Query Algorithm:
access subset of cells and outputs answer

$time$ = #cells accessed during query

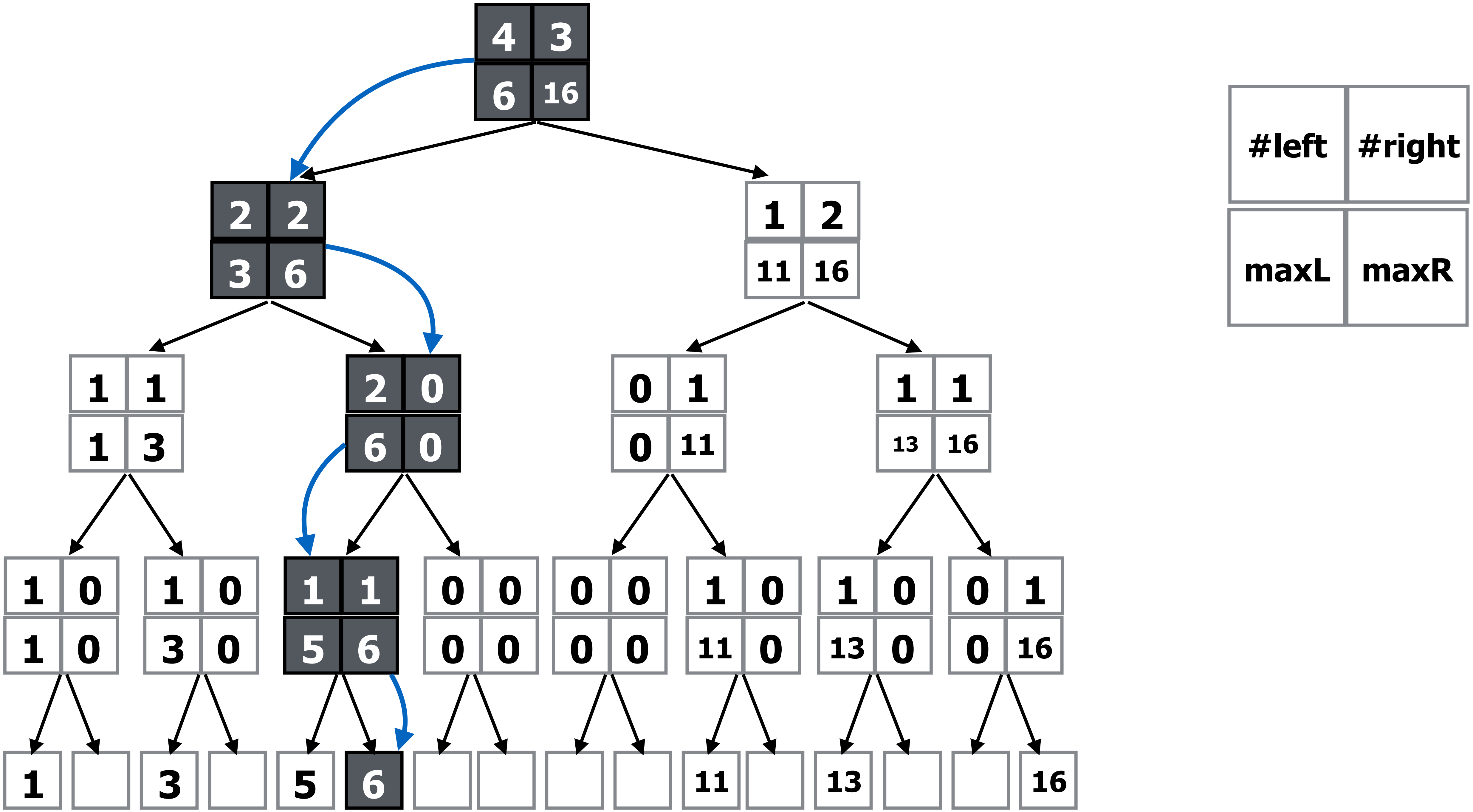


Non-Adaptive Inserts



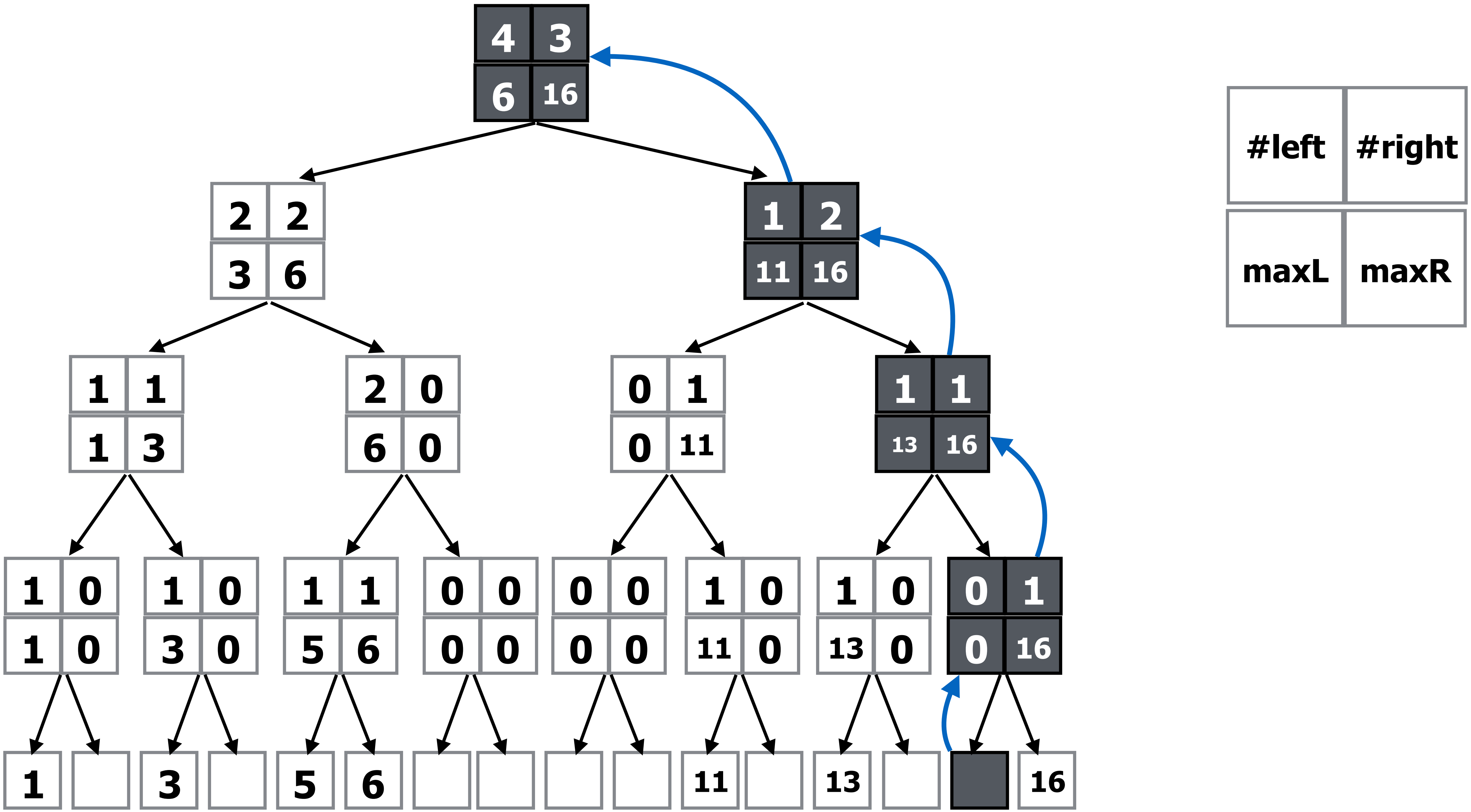
An <u>operation</u> is *non-adaptive* if locations of cells accessed depends only on the operation and not on contents of the cell

Adaptive Median



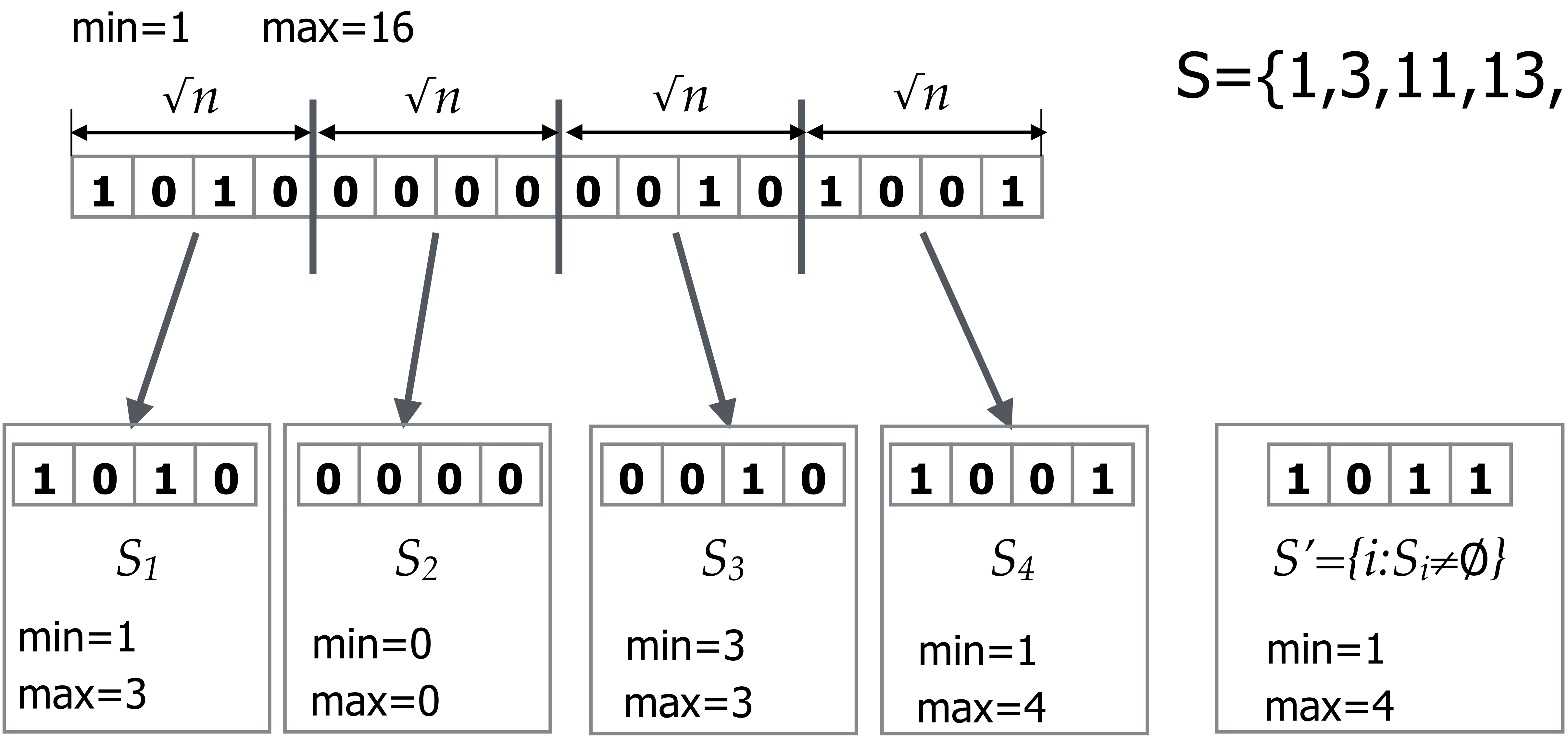
An operation is *non-adaptive* if locations of cells accessed depends only on the operation and not on contents of the cell

Non-Adaptive Predecessors



An <u>operation</u> 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

Adaptive updates and queries

Dynamic connectivity with edge insertions, Parity Sum	$t_q \geq \frac{\log n}{\log w t_u}$	[FredmanSaks89] chronogram technique
k[th] smallest	$t_q \geq \frac{\log n}{\log w t_u}$	[PatrascuThorup14] reduction to Parity Sum
Dynamic connectivity with edge insertions + deletions	$\max\{t_u, t_q\} \geq \log n$	[PatrascuDemaine04] information transfer technique
2D range counting	$t_q \geq \frac{\log^2 n}{\log^2 w t_u}$	[Larsen12] cell sampling + chronogram
2D range counting (amortized + large error)	$t_q \geq \frac{\log^2 n}{\log^2 w t_u}$	[WeinsteinYu16] a new communication model

Non-Adaptive

directed graph reachability with non-adaptive queries	$\max\{t_u, t_q\} \geq n/w$	[BrodyLarsen12]

Non-Adaptive Lower Bounds

Minimum with deletions

non-adaptive delete and minimum operations	$t_{\text{del}} + t_{\text{min}} \geq \frac{\log n}{\log\log n + \log w}$	this work

Median

non-adaptive insert and median operations	$t_i + t_m \geq \frac{\log n}{\log\log n + \text{l}}$	
non-adaptive inserts	$t_m \geq \frac{n^{\frac{1}{t_i+1}}}{w^2 \cdot t_i^2}$	this work
adaptive insert and median operations	even super constant open	

Bit vectors: t_i=1, t_m=n

...essor Se...

	...or $t_p \geq \frac{\text{mi}}{}$	...endent ...gerBrody Kephart17]
	$\frac{\text{...g } n}{\text{...}n + \log w}$ or $t_i \geq \frac{t_p \cdot n^{\frac{1}{2(t_p+1)}}}{\log n}$	this work
adaptive insert and predecessor operation	$t_i + t_p \geq \log\log n$	[A88][BF02][SV04] [PatrascuThorup06]

Binary trees: t_i=$\log n$, t_p=$\log n$

vEB: t_i=$\log\log n$, t_p=$\log\log n$

Predecessor Arrays: t_i=n, t_p=1

Results for Median in Other Models

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

- Streaming Algorithms.
  [ChakrabartiJayramPatrascu08]

- Data Structures in the Comparison Model.
  [BrodalChaudriRadhakrishnan96]

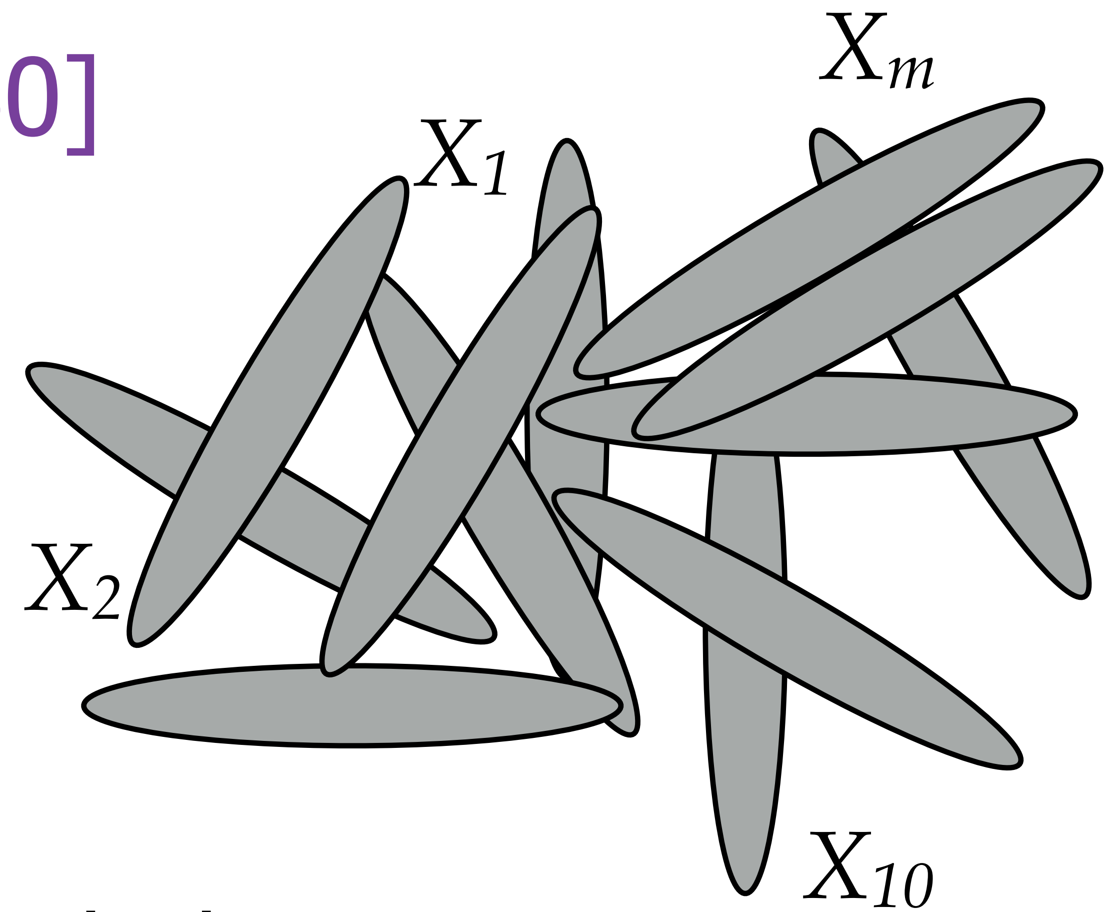
  - insert time $= t$,

  - median computation $\geq m/4^t$, median computation $\leq m/2^t$

The Key Technical Tool

Sunflower Lemma

[ErdosRado60]

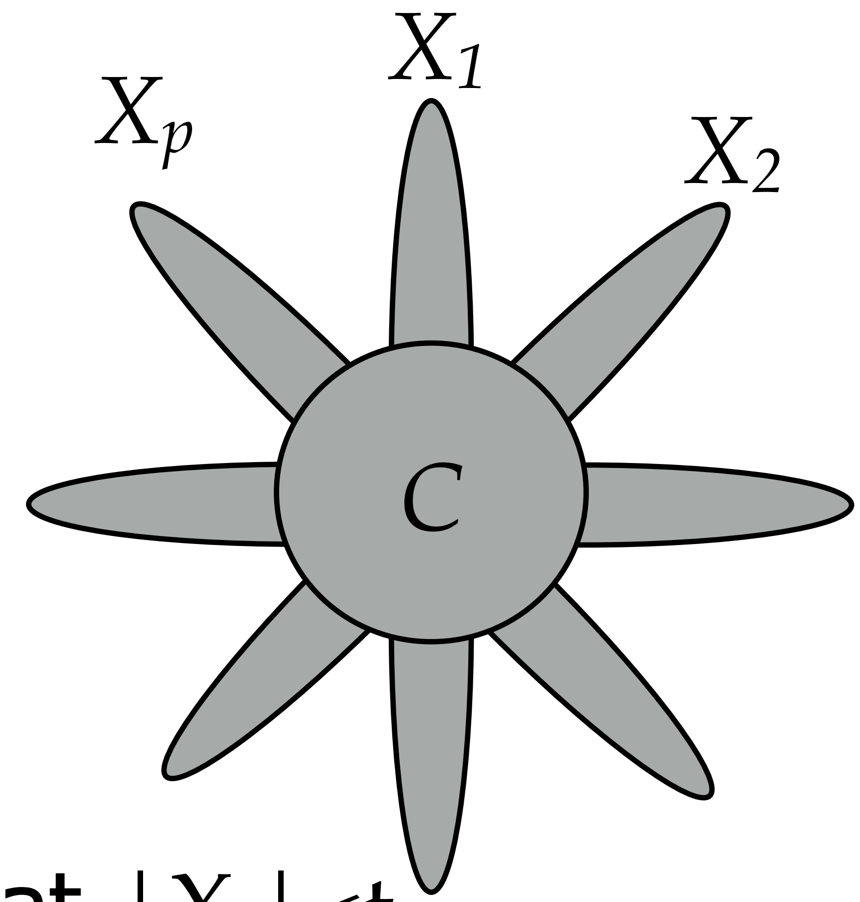


For every sequence $X = X_1, X_2, \ldots, X_n$ such that $|X_i| \leq t$,

if $n \geq t!(p-1)^{t+1}$

Sunflower Lemma

[ErdosRado60]



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

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

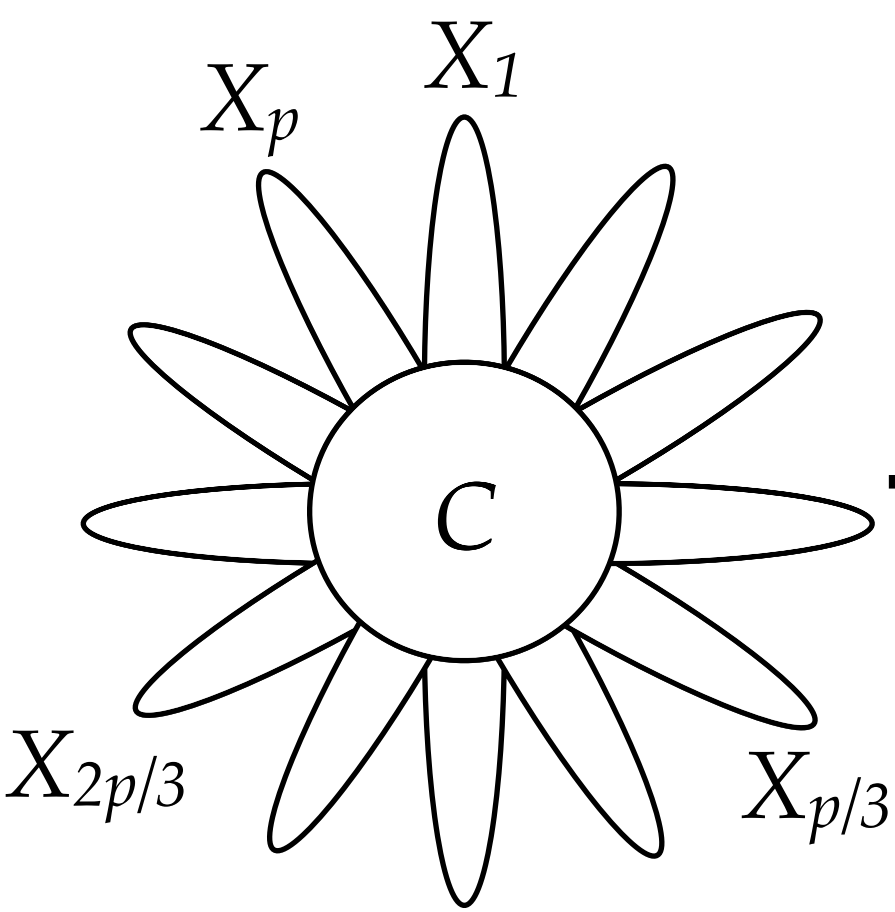
w: word size t_i: time for inserts t_m: time for median

Outline of Proof

1. Let $X = X_1, X_2, \ldots, X_n,$ where

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

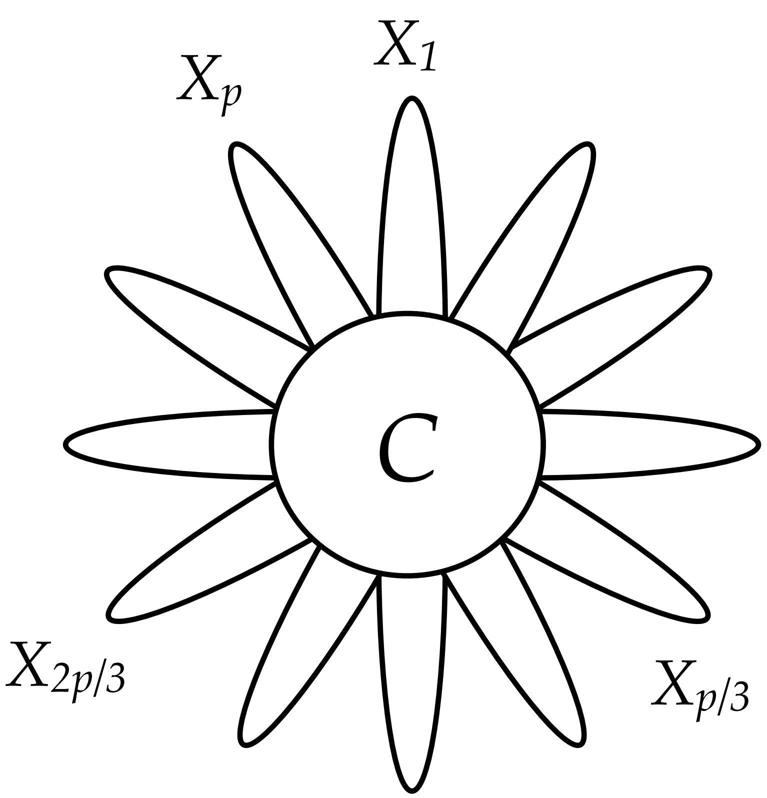
2. Identify a Sunflower in X with p petals



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

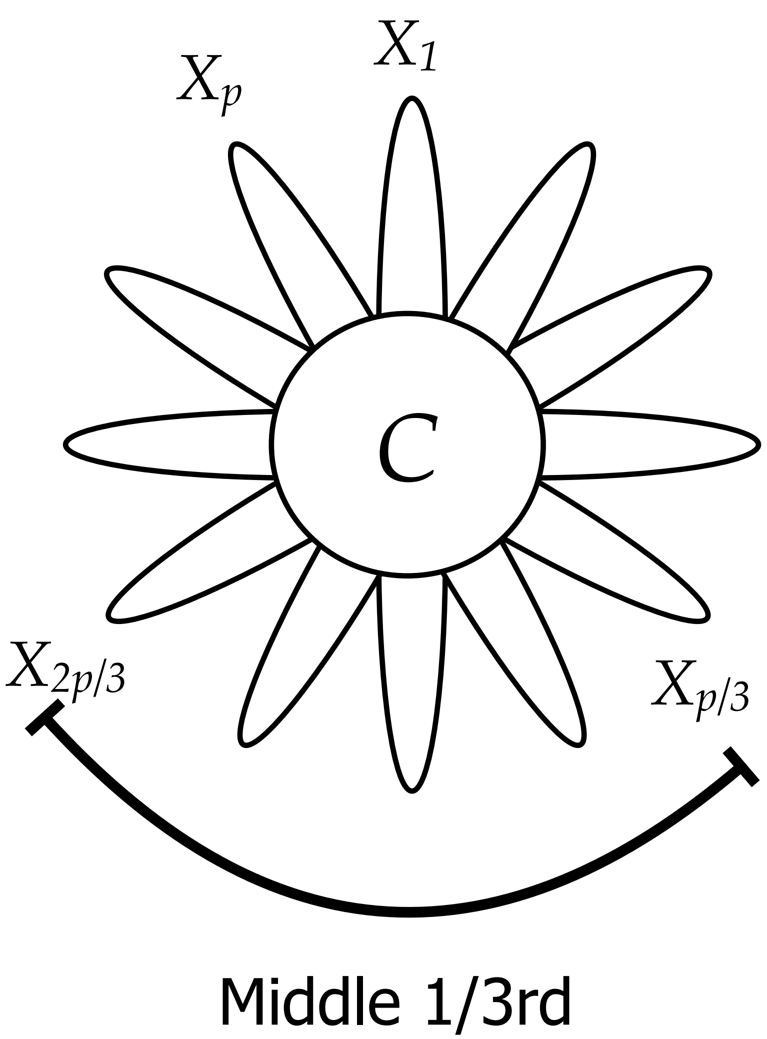
Proof of Theorem 1

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



Proof of Theorem 1

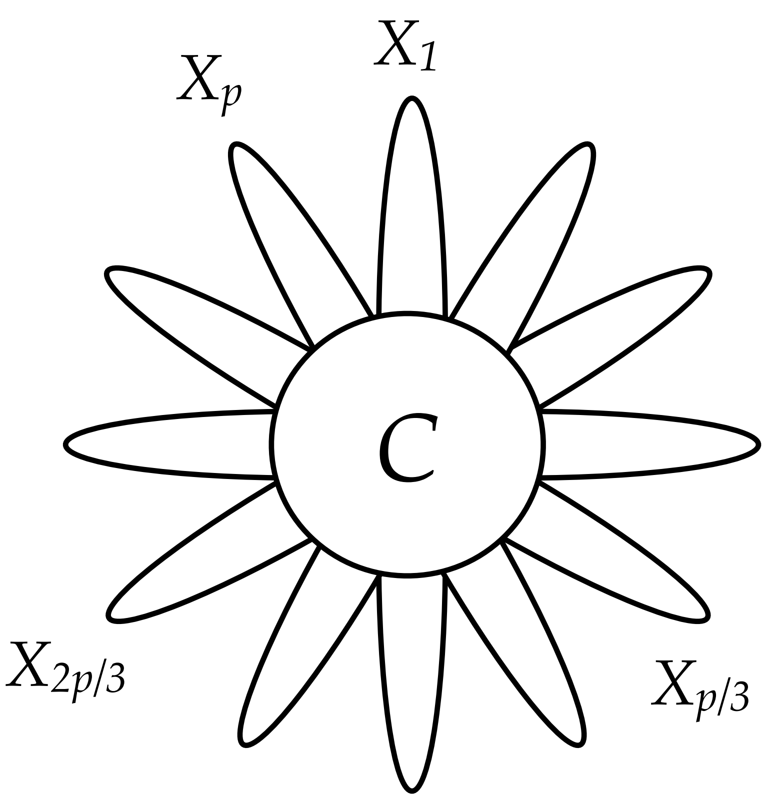
$X_i = \{j \mid j$ is accessed while inserting i or computing median$\}$



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

Proof of Theorem 1

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

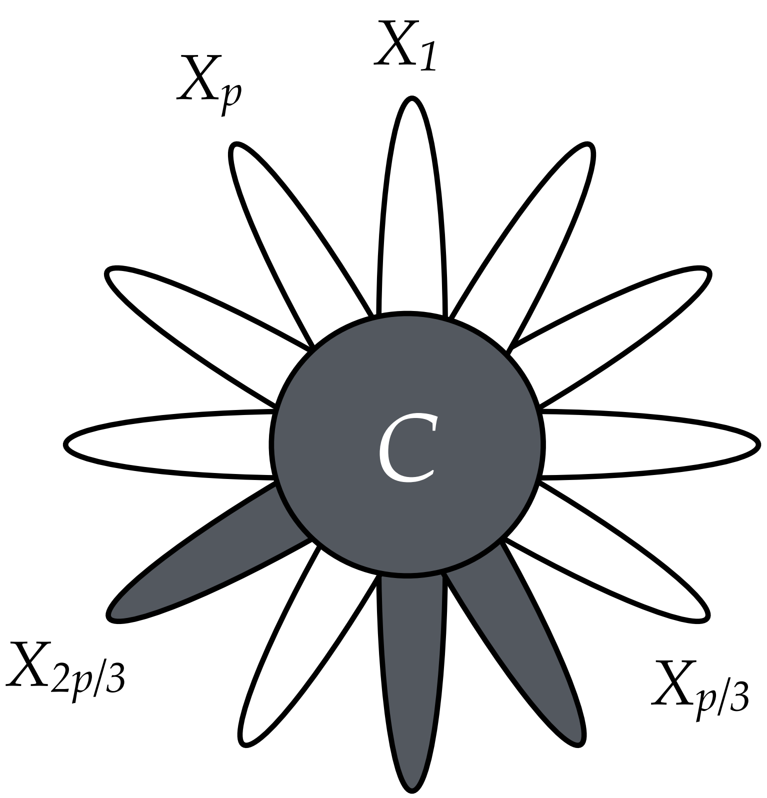


1. Let S be a uniformly random subset of middle 1/3rd elements

2. Insert elements of S

Proof of Theorem 1

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

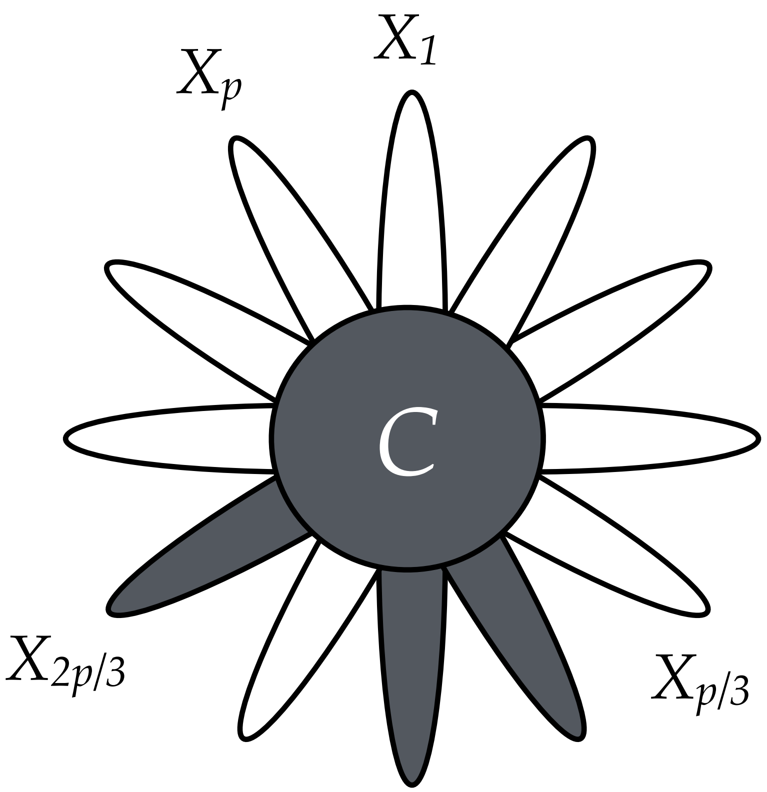


1. Let S be a uniformly random subset of middle 1/3rd elements

2. Insert elements of S

Proof of Theorem 1

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



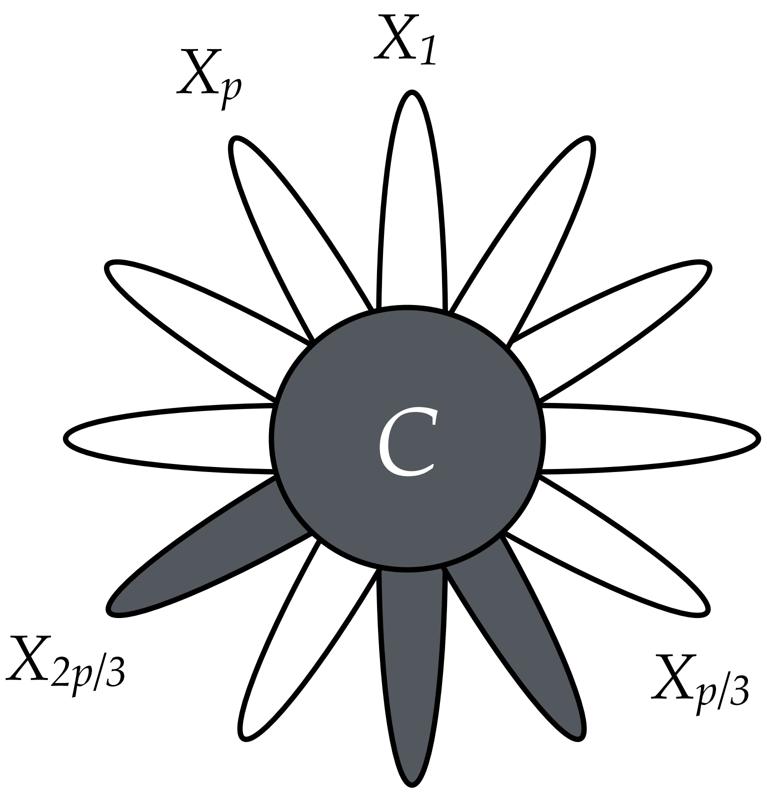
1. Let S be a uniformly random subset of middle 1/3rd elements

2. Insert elements of S

<u>Claim:</u> Contents of C encode S

Proof of Theorem 1

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



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

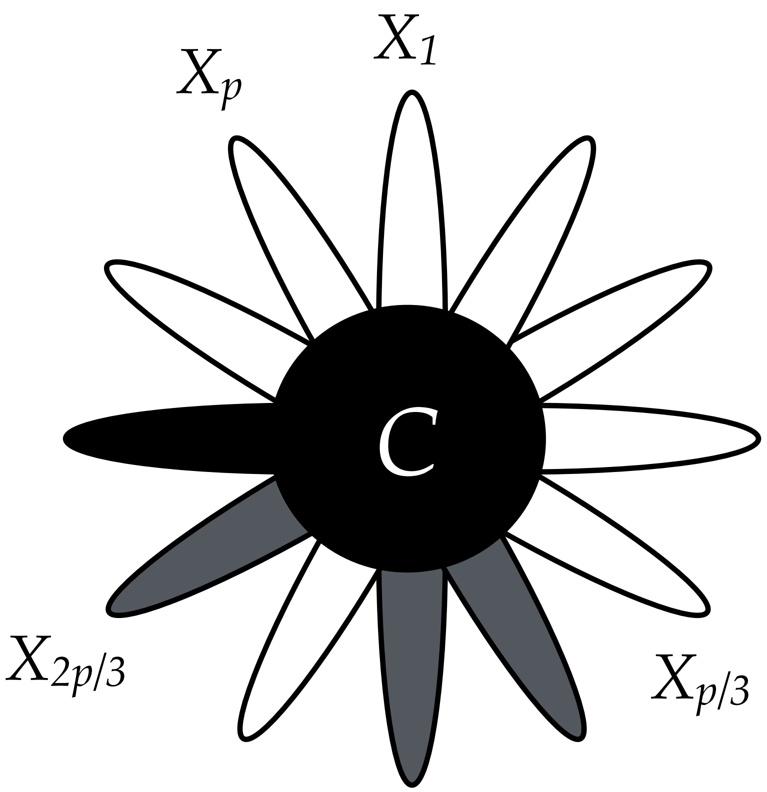
2. Insert elements of S

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.

Proof of Theorem 1

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



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

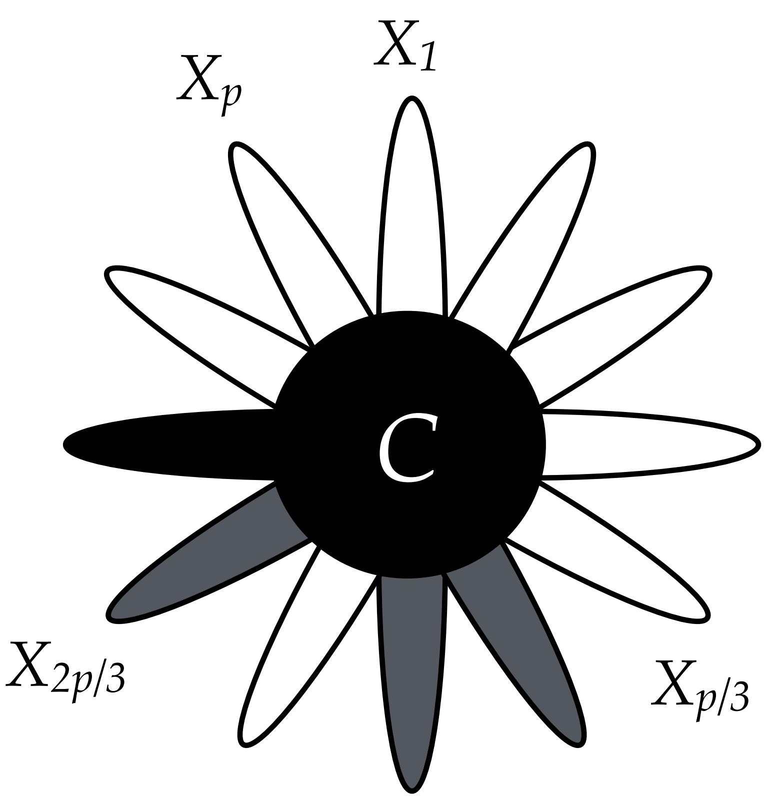
2. Insert elements of S

<u>Claim:</u> 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.

Proof of Theorem 1

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



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

2. Insert elements of S

<u>Claim:</u> 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.

Conclusion

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

Entropy ≤ length of encoding

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 t_i: time for inserts t_m: time for median

Second Lower Bound for Median

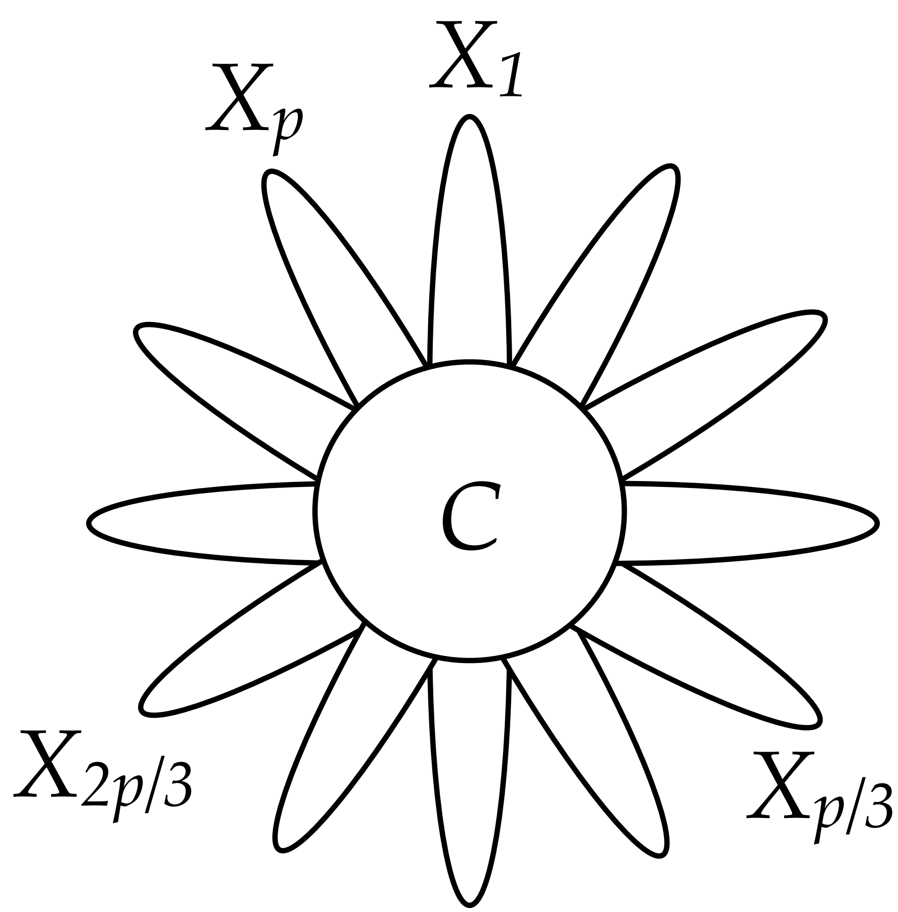
Theorem 2: Any data structure for Median on $[n]$ with non-adaptive insertions must have

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

w: word size t_i: time for inserts t_m: time for median

Outline of Proof

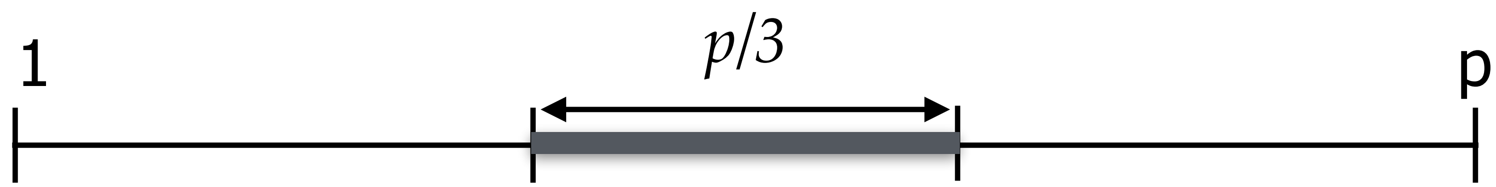
Step 1: Like before, identify a Sunflower among $X = X_1, X_2, \ldots, X_n$, where $X_i = \{j \mid j \text{ is accessed while inserting } i\}$



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

Step 3: lower bound for k^{th} smallest via chronogram approach

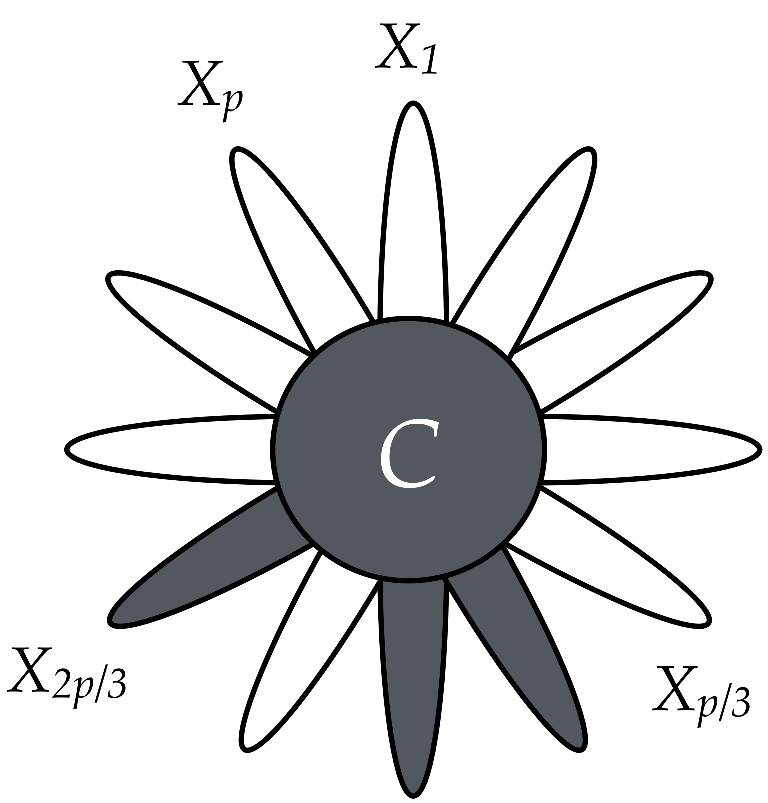
Reduction to kth smallest



(input set S is a subset of this interval)

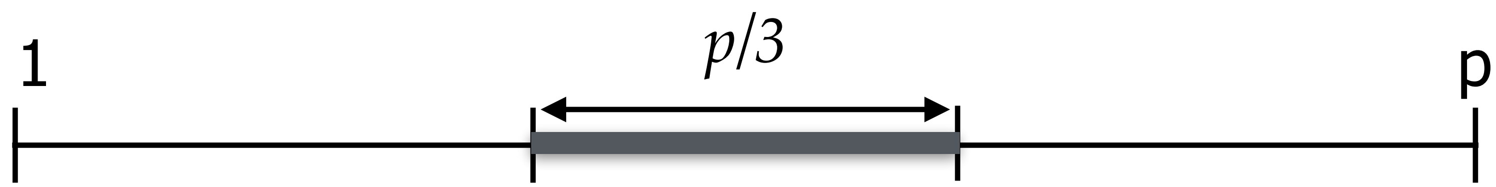
Claim: After fixing C and $|S|$, can find k^{th} smallest in time t_m

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



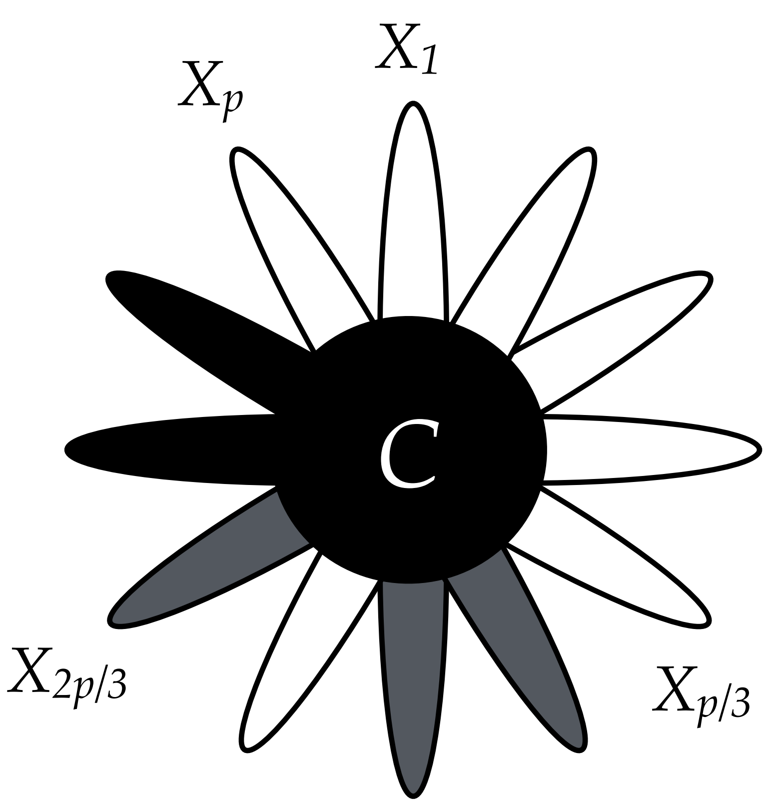
$X_i = \{j \mid j \text{ is accessed while inserting } i\}$

Reduction to kth smallest



(input set S is a subset of this interval)

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



$X_i = \{j \mid j$ is accessed while inserting $i\}$

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

The Chronogram Approach [FS89] [PT14]

1. Applying the "$log\, n$" lower bound,

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

2. Can be improved to

$$t_m \geq \frac{n^{\frac{1}{2(t_i+1)}}}{w t_i}$$

<u>Key insight</u>: 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

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

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.

Thank you