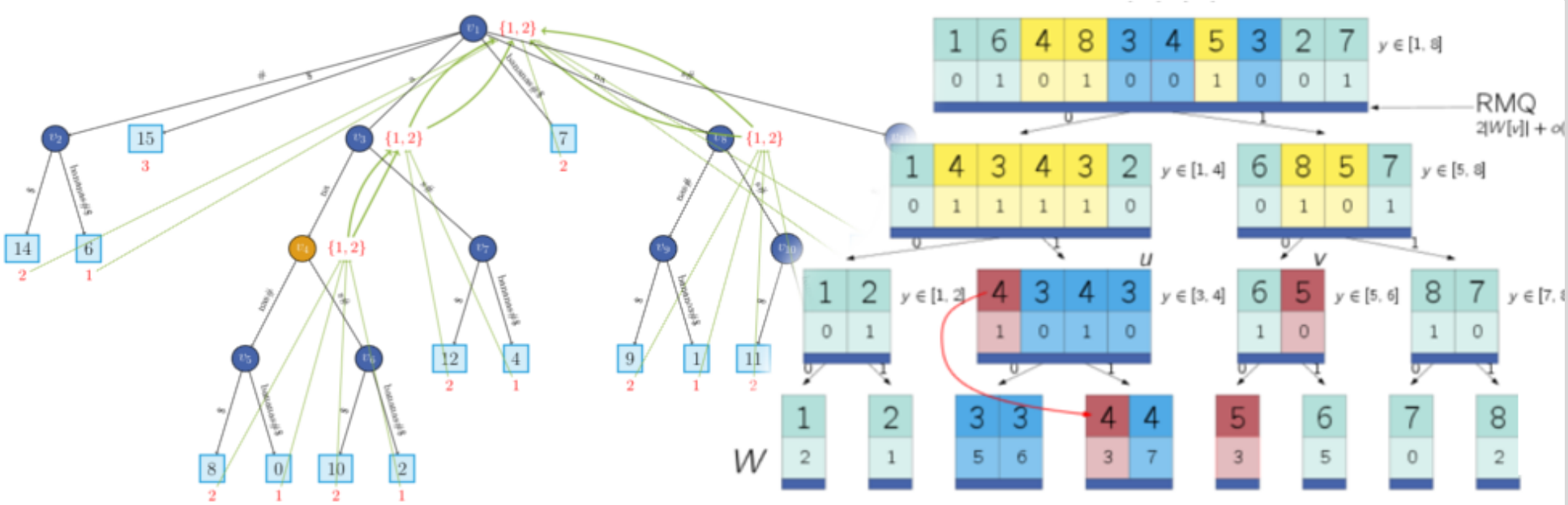
Karlsruhe Institute of Technology

Practical Range Minimum Queries Revisited

Niklas Baumstark, Simon Gog, Tobias Heuer, Julian Labeit | August 21, 2019

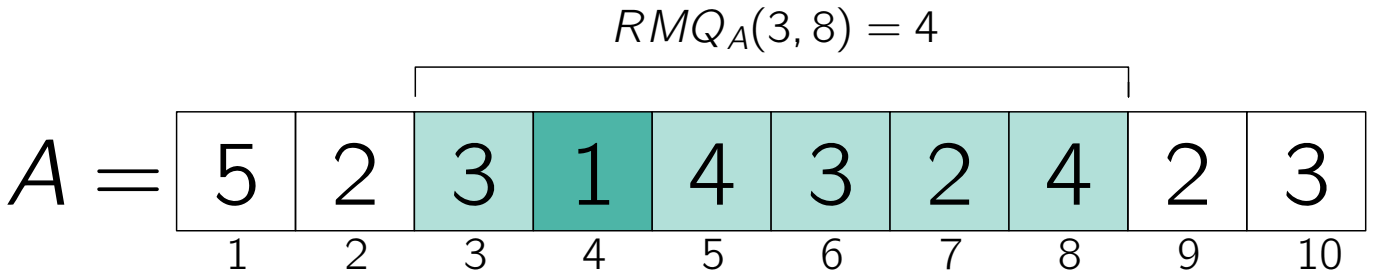
INSTITUTE FOR THEORETICAL INFORMATICS, PROFESSOR SANDERS



Range Minimum Query Problem

Definition

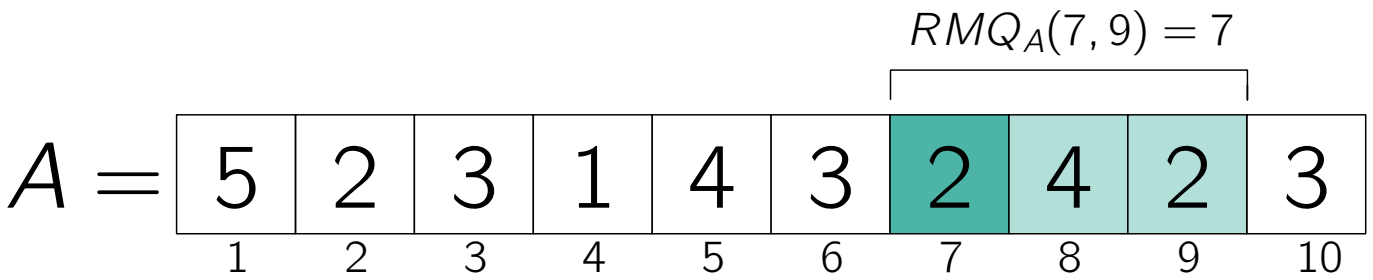
Given an array **A[1..n]** of n numbers. The **range minimum query** (RMQ) problem is to find an index structure that returns for **any range A[i..j]** the **position of the leftmost minimum**.



Range Minimum Query Problem

Definition

Given an array **A**[**1**..**n**] of n numbers. The **range minimum query** (RMQ) problem is to find an index structure that returns for **any range A**[**i**..**j**] the **position of the leftmost minimum**.



Applications

RMQs have a wide range of applications [FH11]

- Text-Indexing, Pattern Matching
- String mining
- Text compression
- Document Retrieval
- Tree, Graphs, Bioinformatics, etc.

Applications

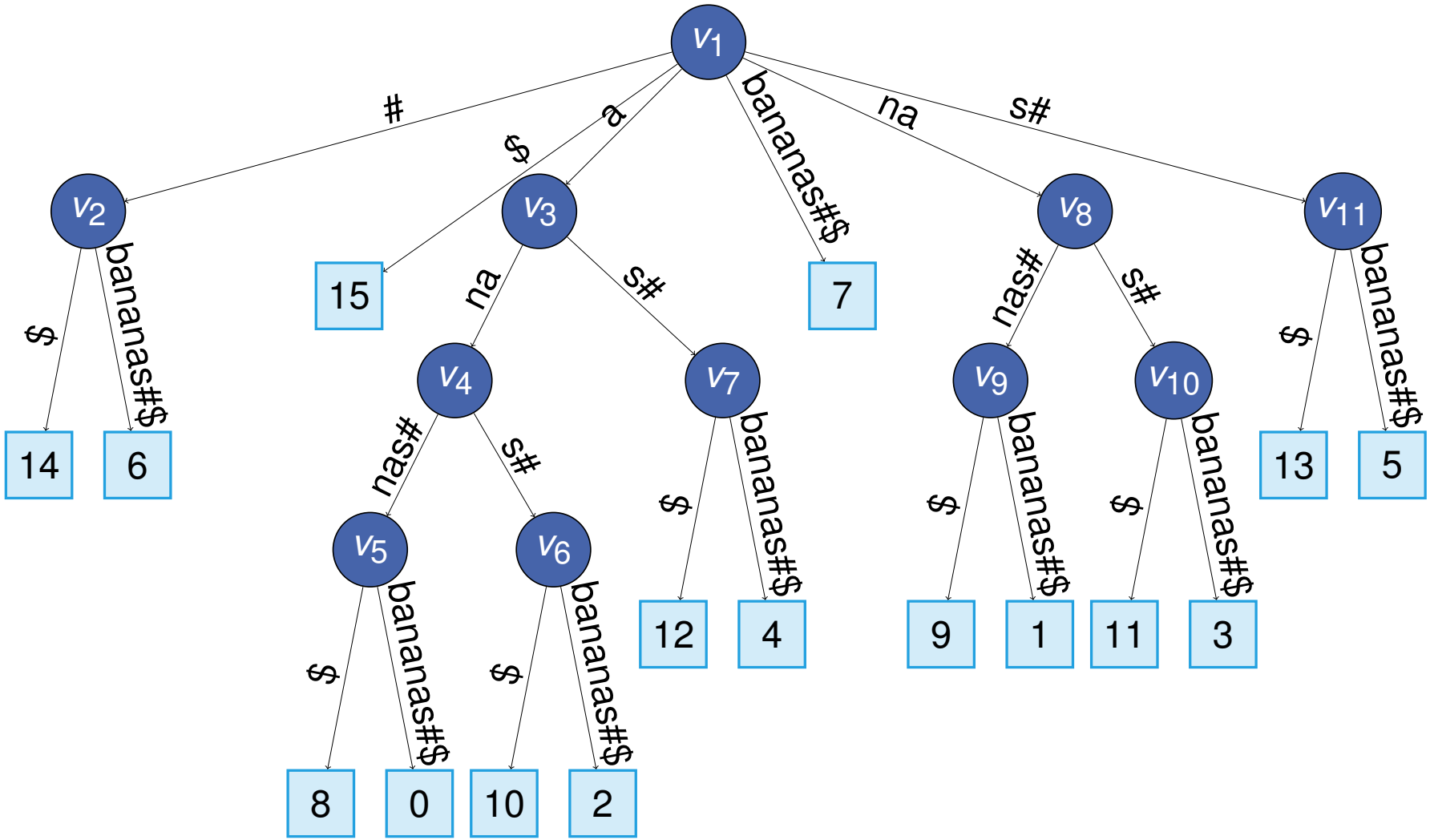
Document Listing

Document 1	Document 2	Search Pattern
ananas	bananas	ana

Offset	Doc	Suffix
14	2	#$
6	1	#bananas#$
15		$
8	2	ananas#$
0	1	ananas#bananas#$
10	2	anas#$
2	1	anas#bananas#$
12	2	as#$
4	1	as#bananas#$
7	2	bananas#$
9	2	nanas#$
1	1	nanas#bananas#$
11	2	nas#$
3	1	nas#bananas#$
13	2	s#$
5	1	s#bananas#$

Applications

Suffix Tree Traversal



Overview

	Preprocessing	Space	Query Time	Reference
PRECOMPUTE	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$ words	$\mathcal{O}(1)$	
SCAN	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	
SPARSETABLE	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$ words	$\mathcal{O}(1)$	[BF00]
Bender & Farach-Colton	$\mathcal{O}(n)$	$\mathcal{O}(n)$ words	$\mathcal{O}(1)$	[BF00]
Sadakane	$\mathcal{O}(n)$	4n + o(n) bits	$\mathcal{O}(1)$	[Sad07]
Fischer & Heun	$\mathcal{O}(n)$	2n + o(n) bits	$\mathcal{O}(1)$	[FH11]
Ferrada & Navarro	$\mathcal{O}(n)$	2n + o(n) bits	$\mathcal{O}(1)$	[FN16]

Note, succinct solution are only **constant in theory**.

In practice **highly engineered logarithmic solutions** are used to lower the $o(n)$ space term.

$\Rightarrow$ **Time-Space Tradeoff**

Overview

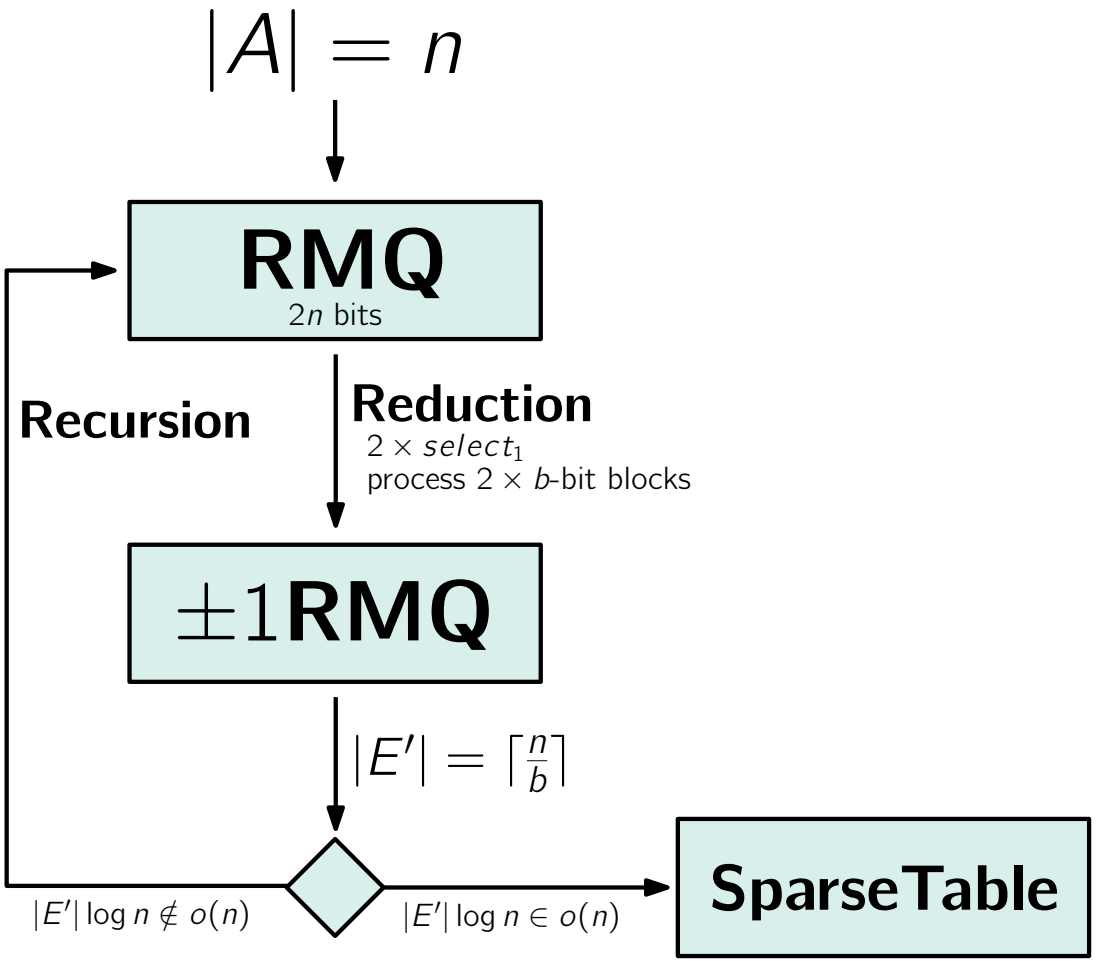
	Preprocessing	Space	Query Time	Reference
PRECOMPUTE	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$ words	$\mathcal{O}(1)$	
SCAN	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	
SPARSETABLE	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$ words	$\mathcal{O}(1)$	[BF00]
Bender & Farach-Colton	$\mathcal{O}(n)$	$\mathcal{O}(n)$ words	$\mathcal{O}(1)$	[BF00]
Sadakane	$\mathcal{O}(n)$	4n + o(n) bits	$\mathcal{O}(1)$	[Sad07]
Fischer & Heun	$\mathcal{O}(n)$	2n + o(n) bits	$\mathcal{O}(1)$	[FH11]
Ferrada & Navarro	$\mathcal{O}(n)$	2n + o(n) bits	$\mathcal{O}(1)$	[FN16]

Note, succinct solution are only **constant in theory**.

In practice **highly engineered logarithmic solutions** are used to lower the $o(n)$ space term.

$\Rightarrow$ **Time-Space Tradeoff**

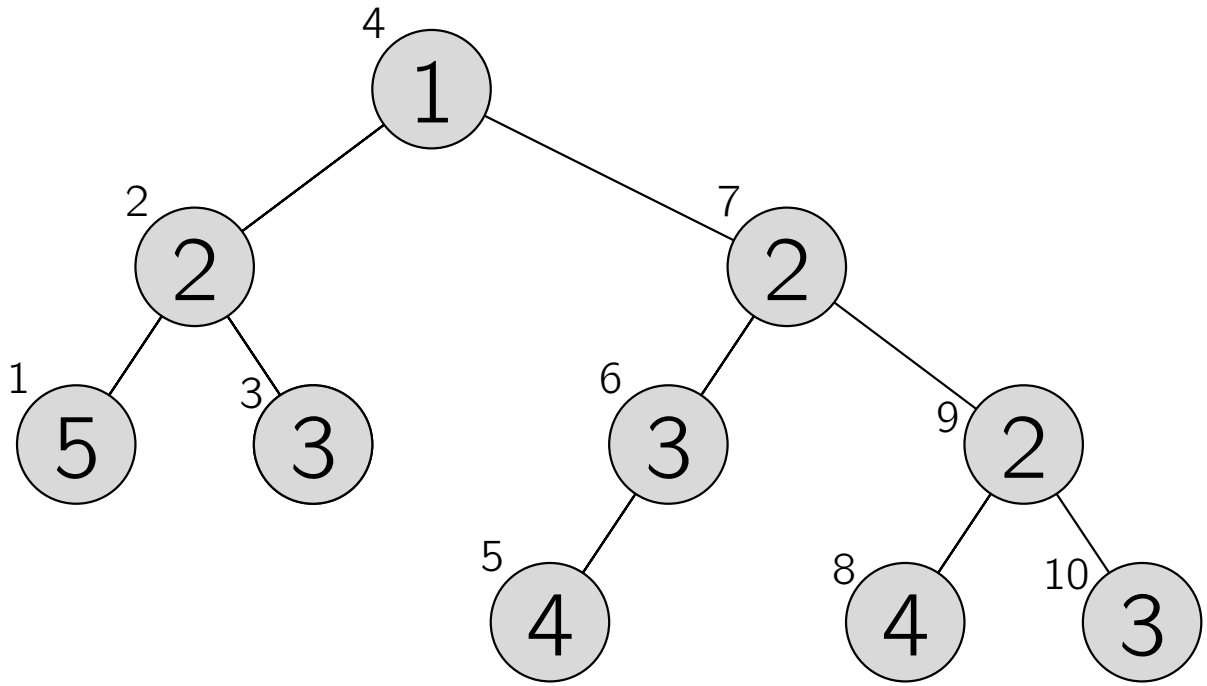
Contributions



1. Using a **recursive** data layout
2. New SELECT_1 implementation on special bit vectors
3. Height minimization strategy for *Generalized Cartesian Trees*

$\Rightarrow$ **Factor 3** faster than the current fastest solution

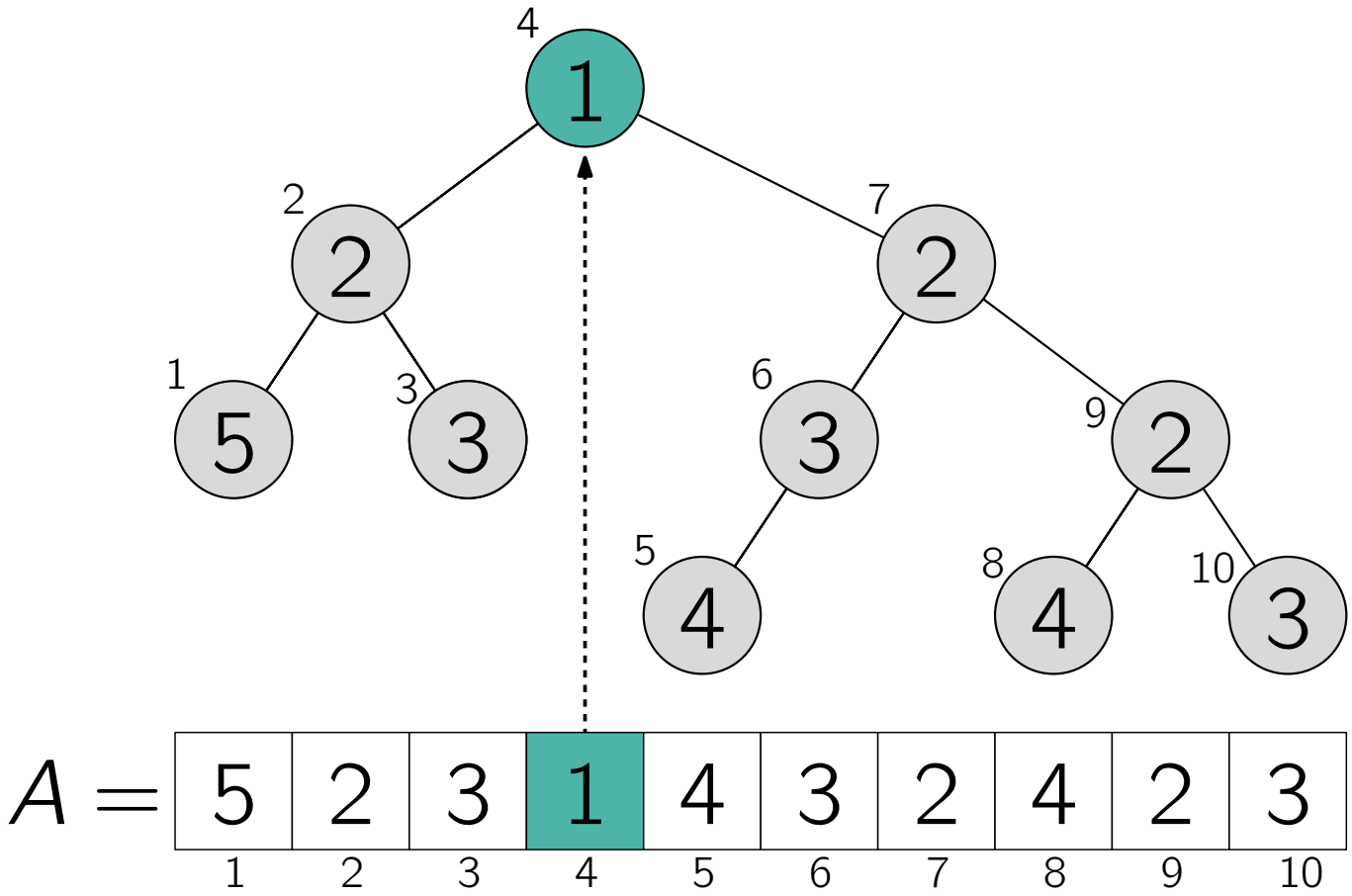
The Cartesian Tree [Vui80]



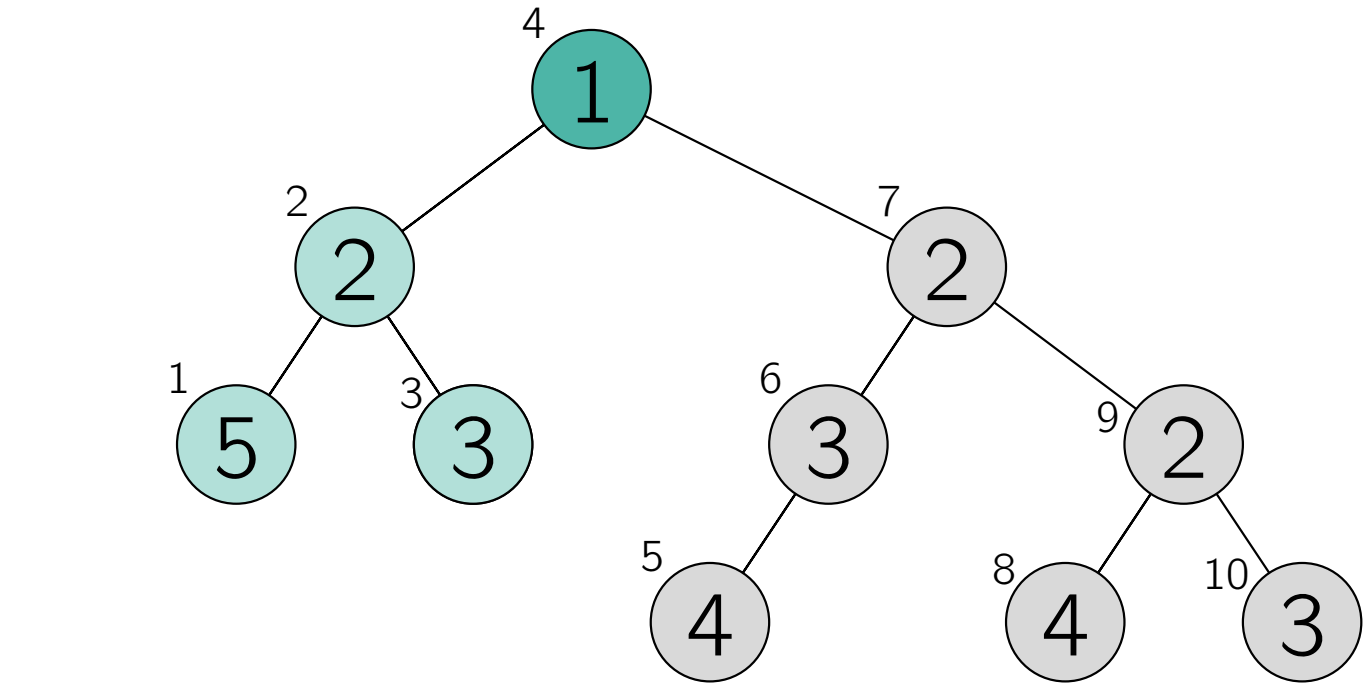
$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

The Cartesian Tree [Vui80]



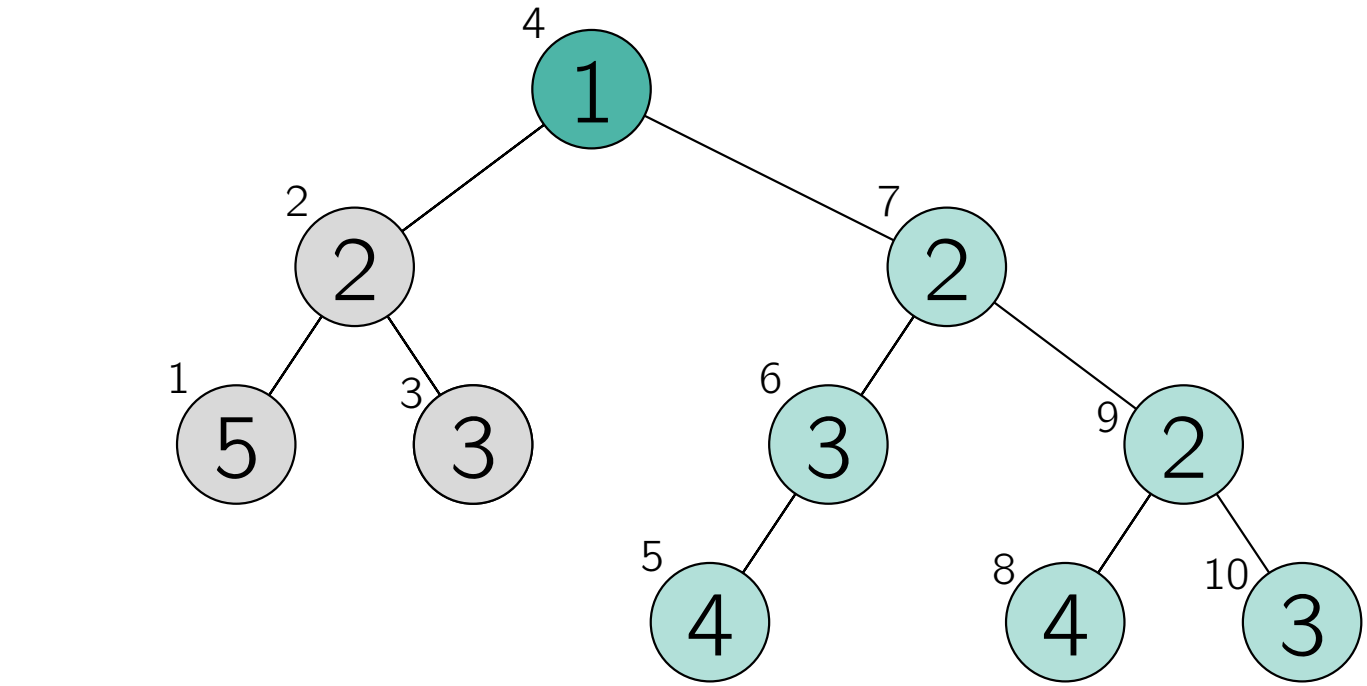
The Cartesian Tree [Vui80]



$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

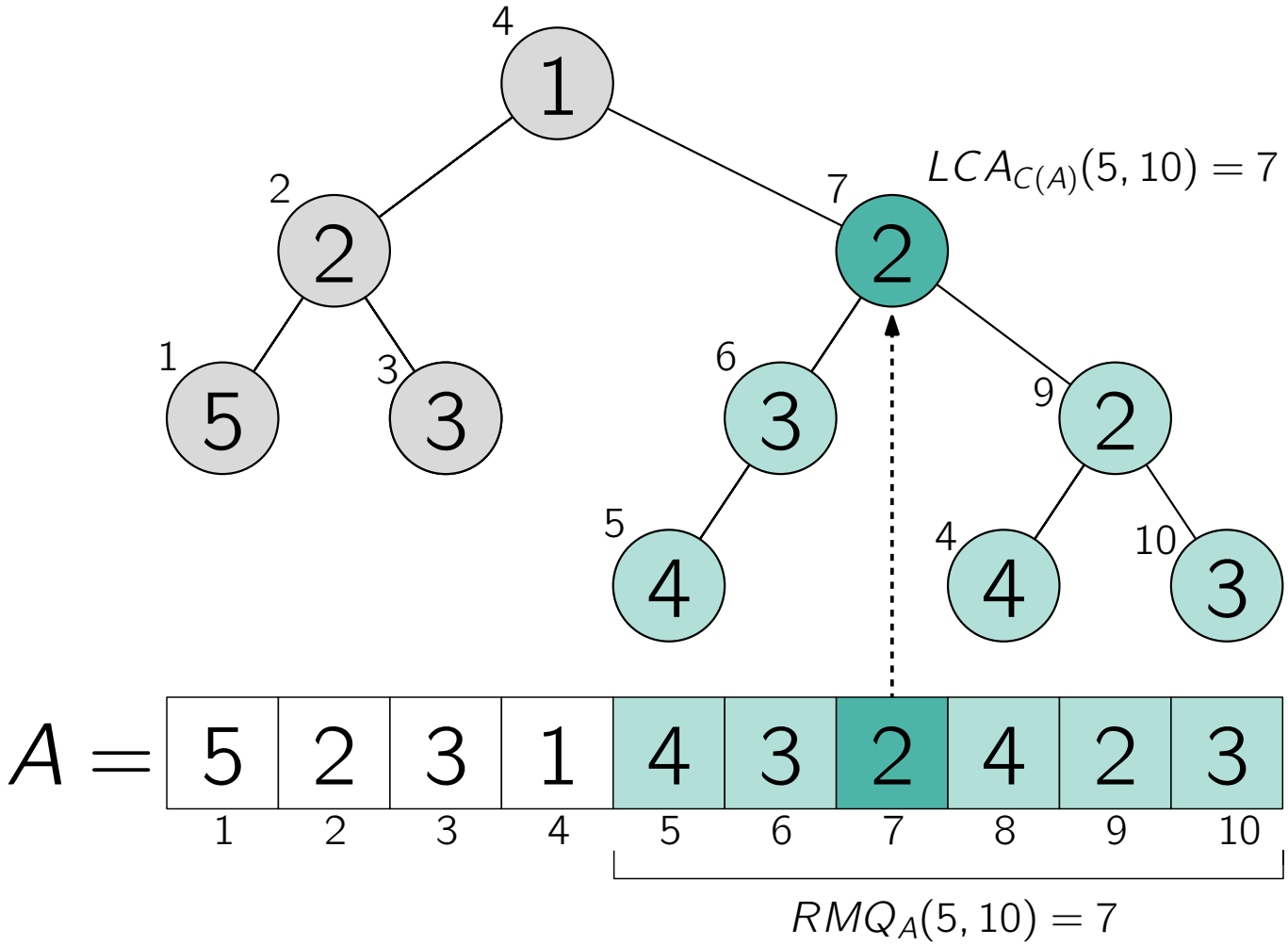
The Cartesian Tree [Vui80]



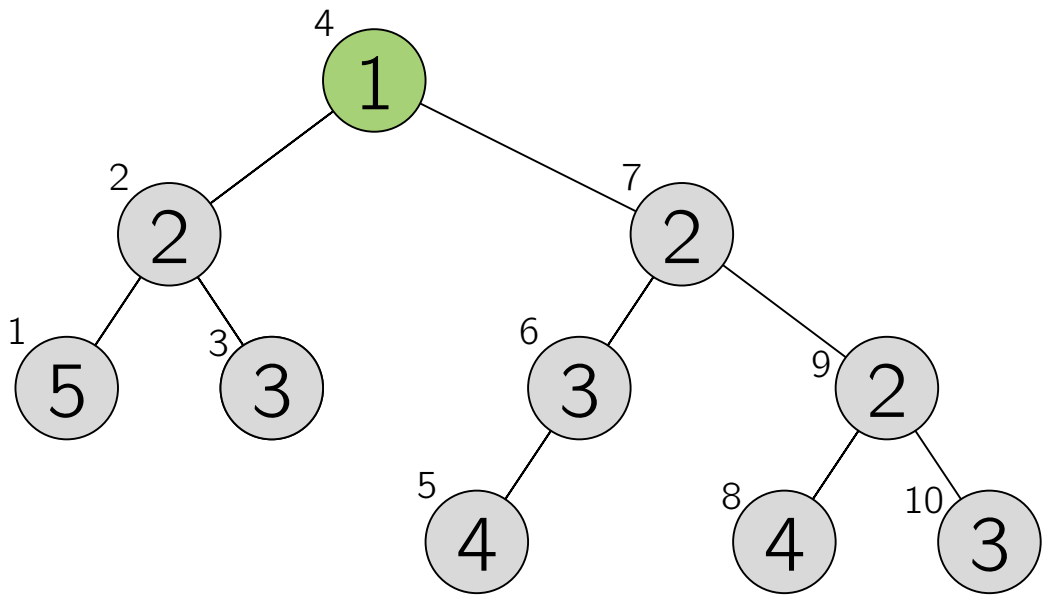
$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

Reducing RMQ to LCA [GBT84]



Balanced Parentheses [MR01]

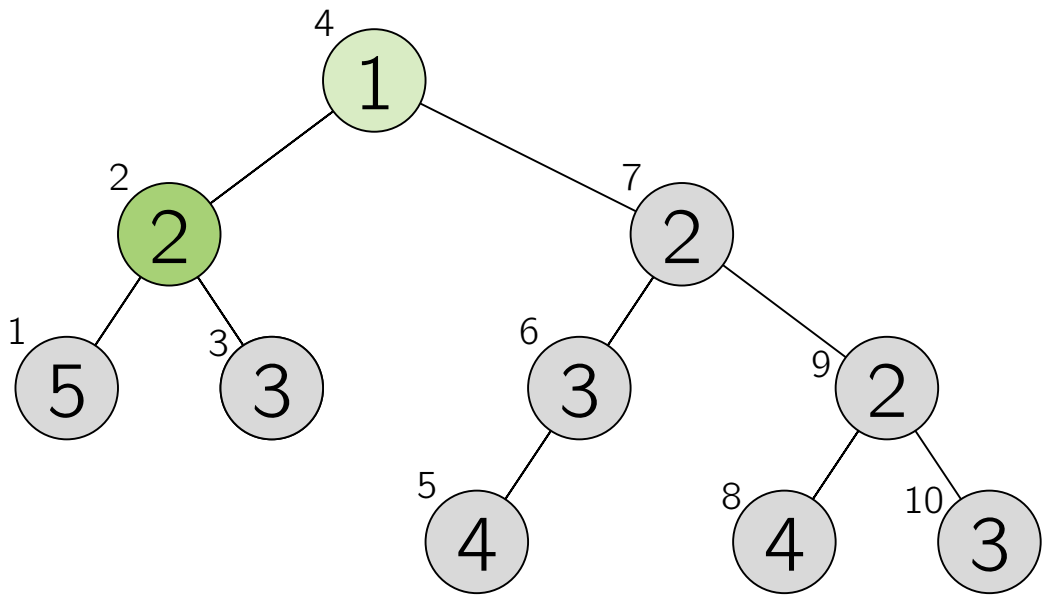


$BP = \overset{1}{(}$

$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

Balanced Parentheses [MR01]

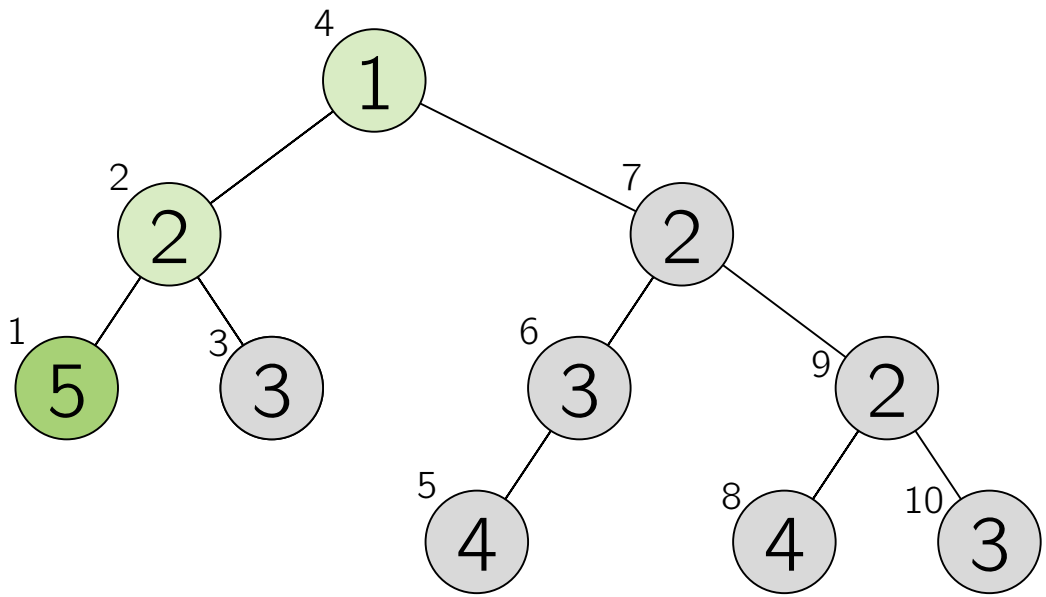


$$BP = \overset{1}{(}\ \overset{2}{(}$$

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 2 & 3 & 1 & 4 & 3 & 2 & 4 & 2 & 3 \\ \hline \end{array}$$

1 2 3 4 5 6 7 8 9 10

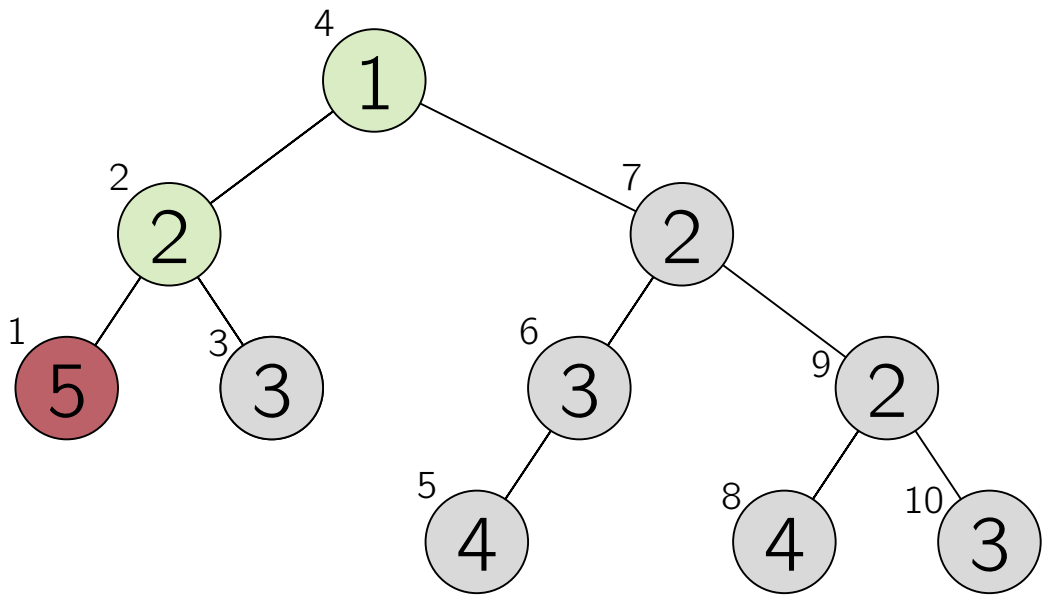
Balanced Parentheses [MR01]



$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}$$

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 2 & 3 & 1 & 4 & 3 & 2 & 4 & 2 & 3 \\ \hline \end{array}$$
$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array}$$

Balanced Parentheses [MR01]

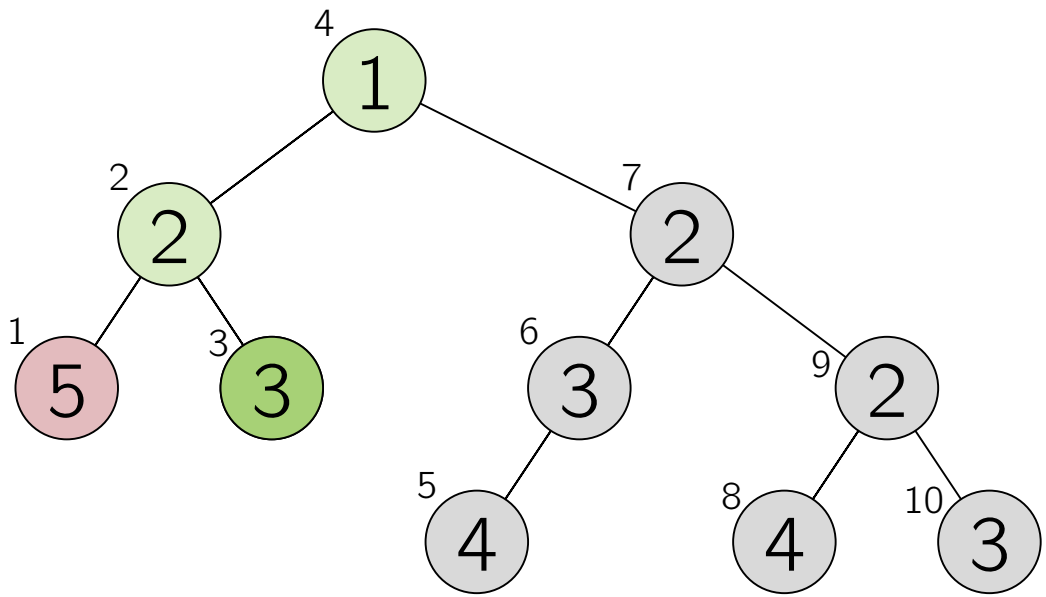


$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}\ )$$

$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

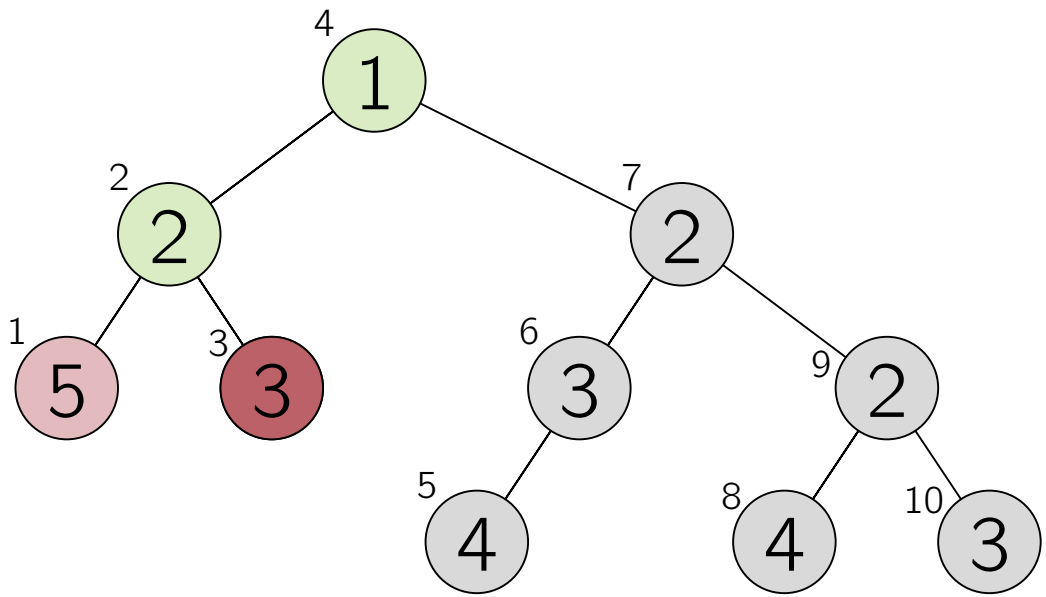
Balanced Parentheses [MR01]



$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}\ )\ \overset{4}{(}$$

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 2 & 3 & 1 & 4 & 3 & 2 & 4 & 2 & 3 \\ \hline \end{array}$$
$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array}$$

Balanced Parentheses [MR01]

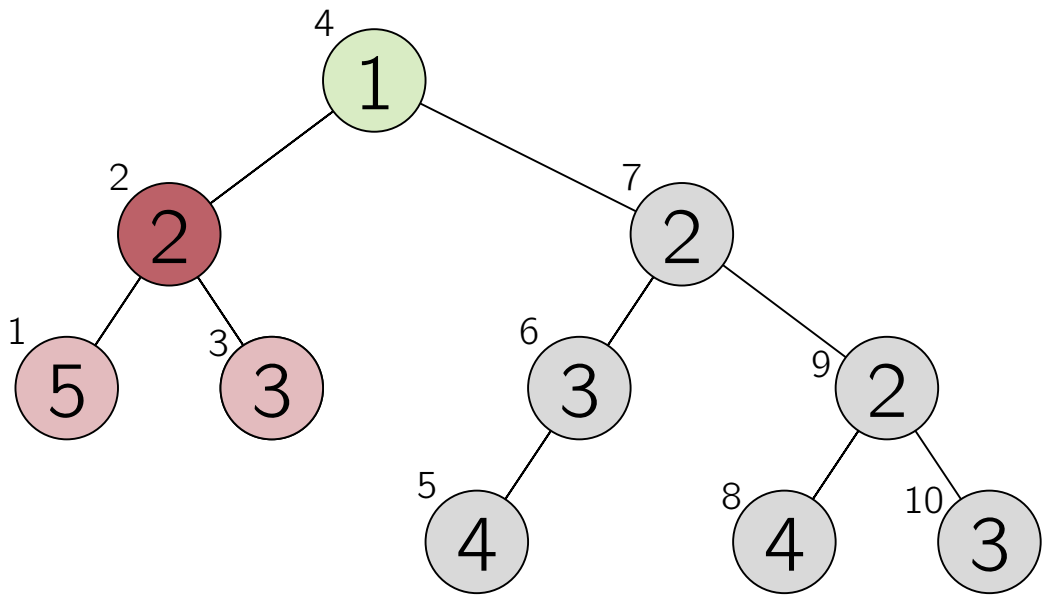


$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}\ )\ \overset{4}{(}\ )$$

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 2 & 3 & 1 & 4 & 3 & 2 & 4 & 2 & 3 \\ \hline \end{array}$$

1 2 3 4 5 6 7 8 9 10

Balanced Parentheses [MR01]

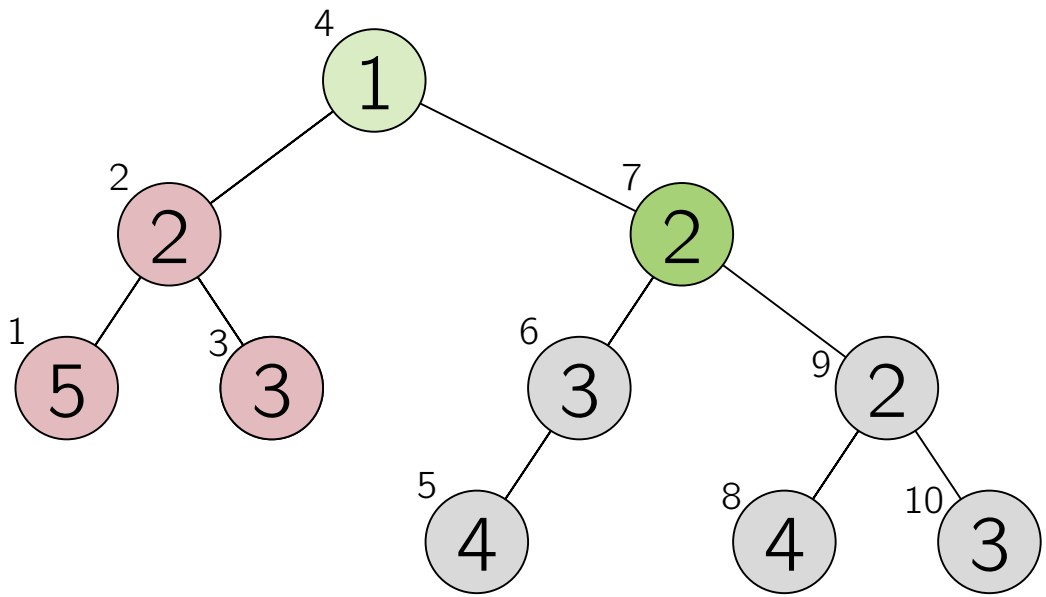


$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}\ )\ \overset{4}{(}\ )\ )$$

$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

Balanced Parentheses [MR01]

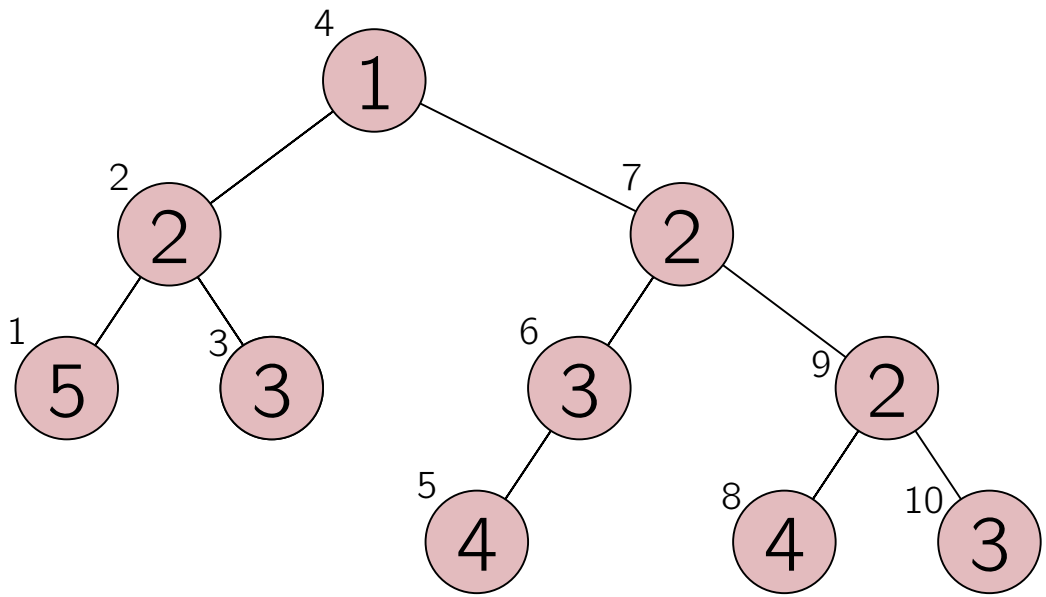


$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}\ )\ \overset{4}{(}\ )\ )\ \overset{5}{(}$$

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 2 & 3 & 1 & 4 & 3 & 2 & 4 & 2 & 3 \\ \hline \end{array}$$

1 2 3 4 5 6 7 8 9 10

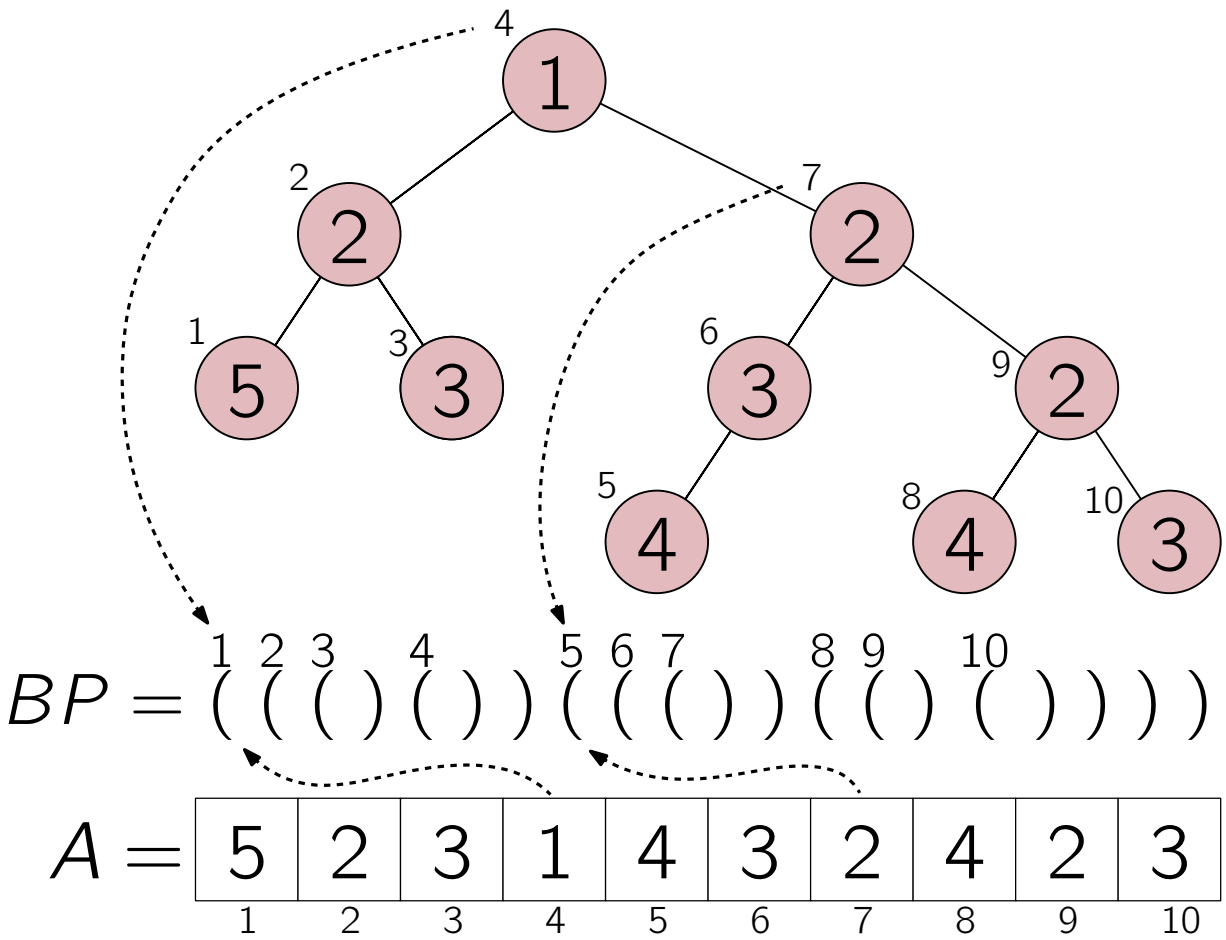
Balanced Parentheses [MR01]



$$BP = \overset{1}{(}\ \overset{2}{(}\ \overset{3}{(}\ )\ \overset{4}{(}\ )\ )\ \overset{5}{(}\ \overset{6}{(}\ \overset{7}{(}\ )\ )\ \overset{8}{(}\ \overset{9}{(}\ )\ \overset{10}{(}\ )\ )\ )$$

$$A = \begin{array}{|c|c|c|c|c|c|c|c|c|c|}\hline 5 & 2 & 3 & 1 & 4 & 3 & 2 & 4 & 2 & 3 \\ \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array}$$

Balanced Parentheses [MR01]



A node with INORDER i, which is equal to its index in A, is mapped to its PREORDER position in the BP sequence.

Bitvector - General Operations

Given a bitvector $B[1..n]$ (where $B[i] \in \{0,1\}\ \forall [1,n]$)

- $\text{RANK}_1(i, B) =$ counts the number of 1's in $B[1..i]$
- $\text{SELECT}_1(i, B) =$ return the index of the i-th 1 in B

$\Rightarrow$ solvable in $\mathcal{O}(1)$ time and $o(n)$ bits space [Jac88]

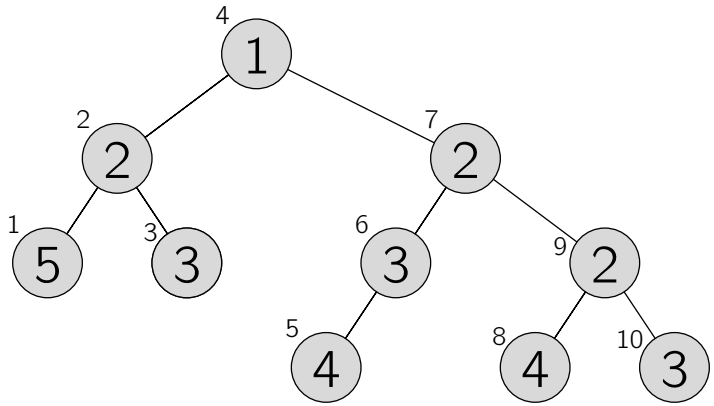
Bitvector - General Operations

Given a bitvector $B[1..n]$ (where $B[i] \in \{0,1\}\ \forall [1,n]$)

- $\text{RANK}_1(i, B) =$ counts the number of 1's in $B[1..i]$
- $\text{SELECT}_1(i, B) =$ return the index of the i-th 1 in B

$\Rightarrow$ solvable in $\mathcal{O}(1)$ time and $o(n)$ bits space [Jac88]

Reducing LCA to ±1RMQ [BV93]



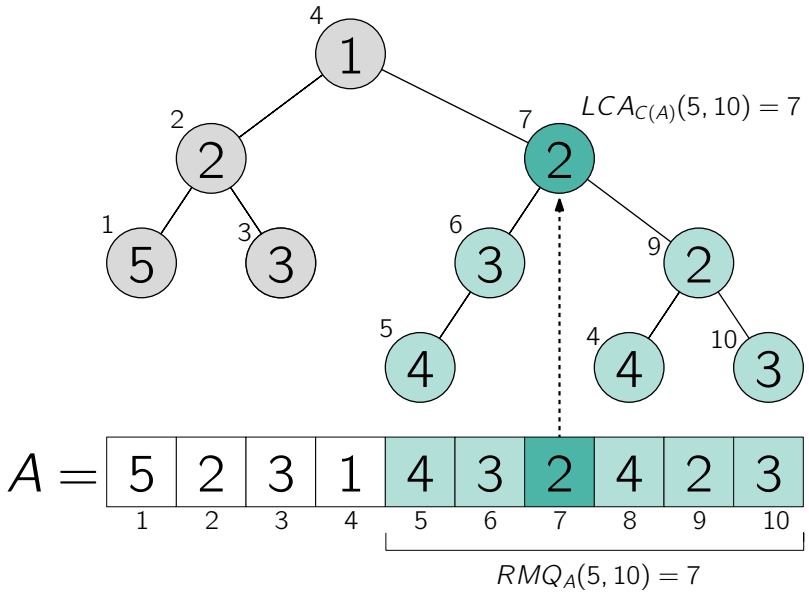
$A =$

5	2	3	1	4	3	2	4	2	3
1	2	3	4	5	6	7	8	9	10

$$L[i] = Depth(v_i)$$

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L	=	1	2	3	2	3	2	1	2	3	4	3	2	3	4	3	4	3	2	1	0
BP	=	(	(	(	)	(	)	)	(	(	(	)	)	(	(	)	(	)	)	)	)

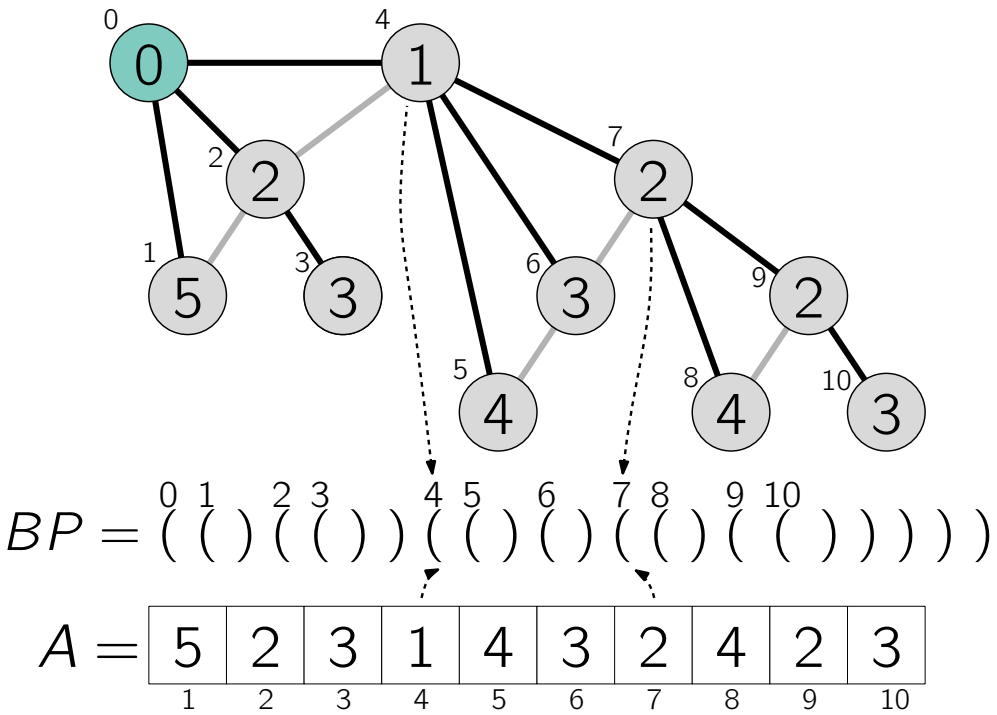
Reducing LCA to ±1RMQ [BV93]



$$\Rightarrow L[i] =: \text{EXCESS}(i, BP) = 2\text{RANK}_1(i, BP) - i$$

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L	=	1	2	3	2	3	2	1	2	3	4	3	2	3	4	3	4	3	2	1	0
BP	=	(	(	(	)	(	)	)	(	(	(	)	)	(	(	)	(	)	)	)	)

Generalized Cartesian Tree [MR01]



Leftmost Path Mapping

- Add a pseudo root to the tree
- Connect the pseudo root with all nodes on the **leftmost** path of the root of $C(A)$
- Continue recursively with the childs of the pseudo root

$\Rightarrow$ A node with INORDER i in $C(A)$ corresponds to a node with PREORDER i in $C^L(A)$.

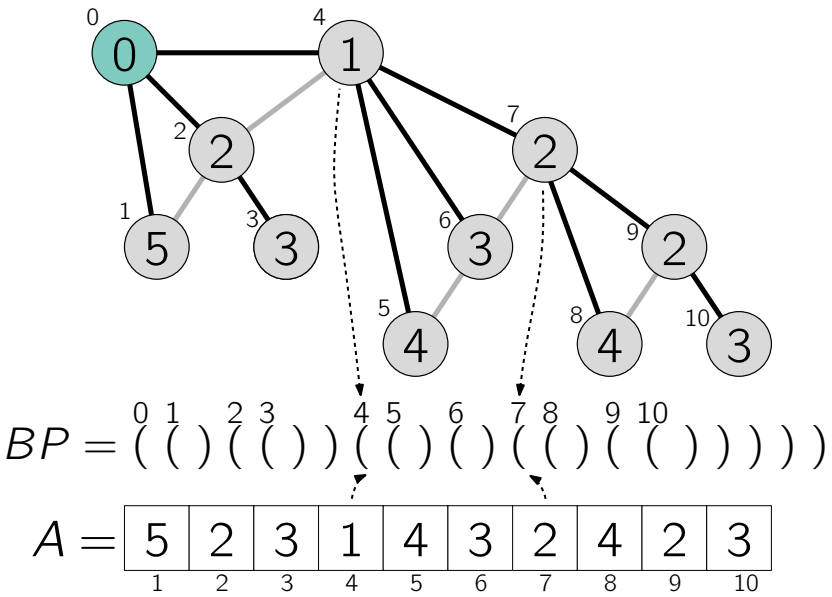
How to solve the RMQ problem? [FN16]

$$\mathrm{RMQ}_A(i,j) = \mathrm{RANK}_1(\mathrm{RRMQ}^{\pm}_{BP(C^L)}(\mathrm{SELECT}_1(i+1)-1, \mathrm{SELECT}_1(j+1)))$$

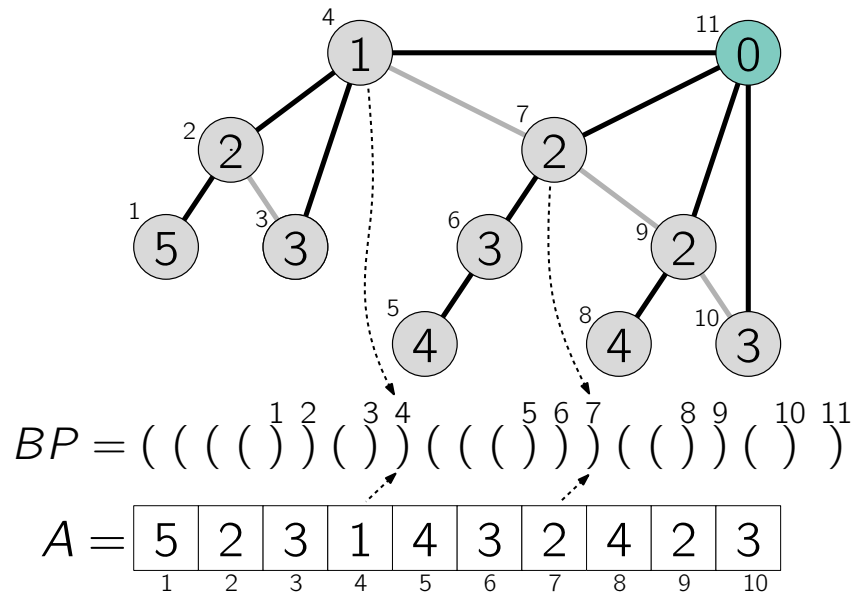
Note: $\pm$1RRMQ has to return the **rightmost** minimum EXCESS value in BP

Generalized Cartesian Tree [MR01]

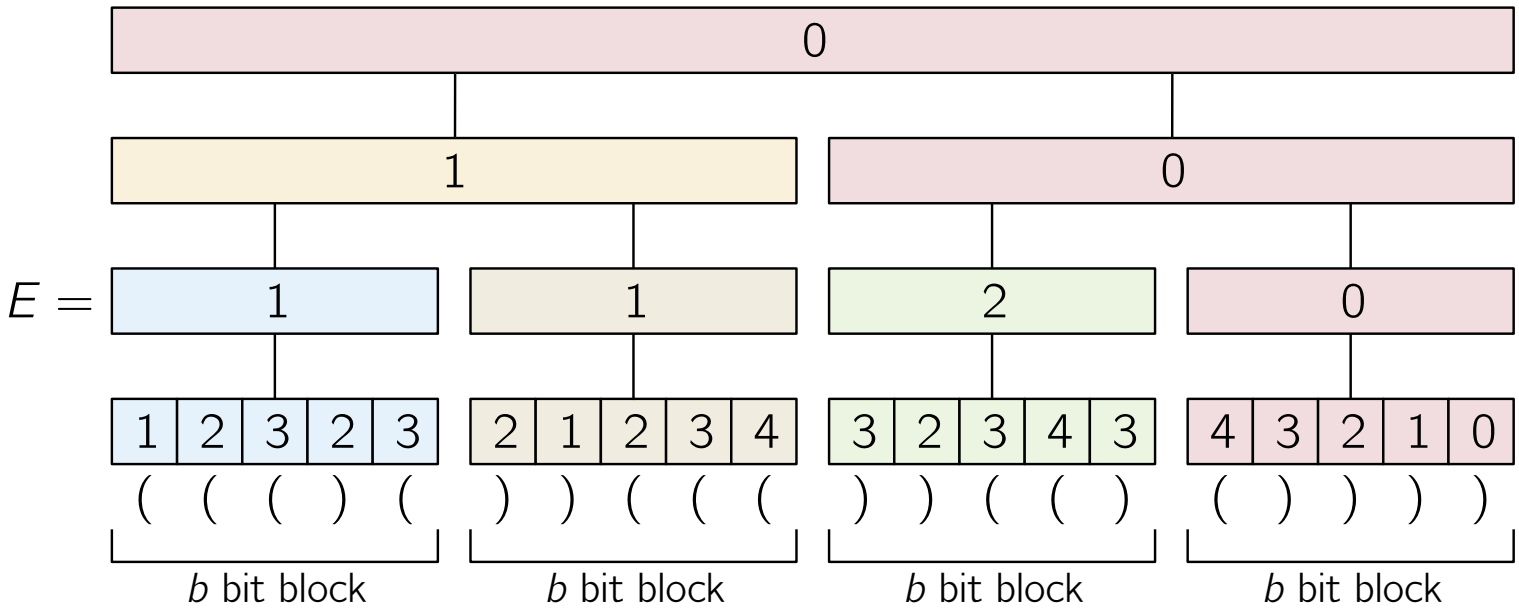
Leftmost Path Mapping



Rightmost Path Mapping

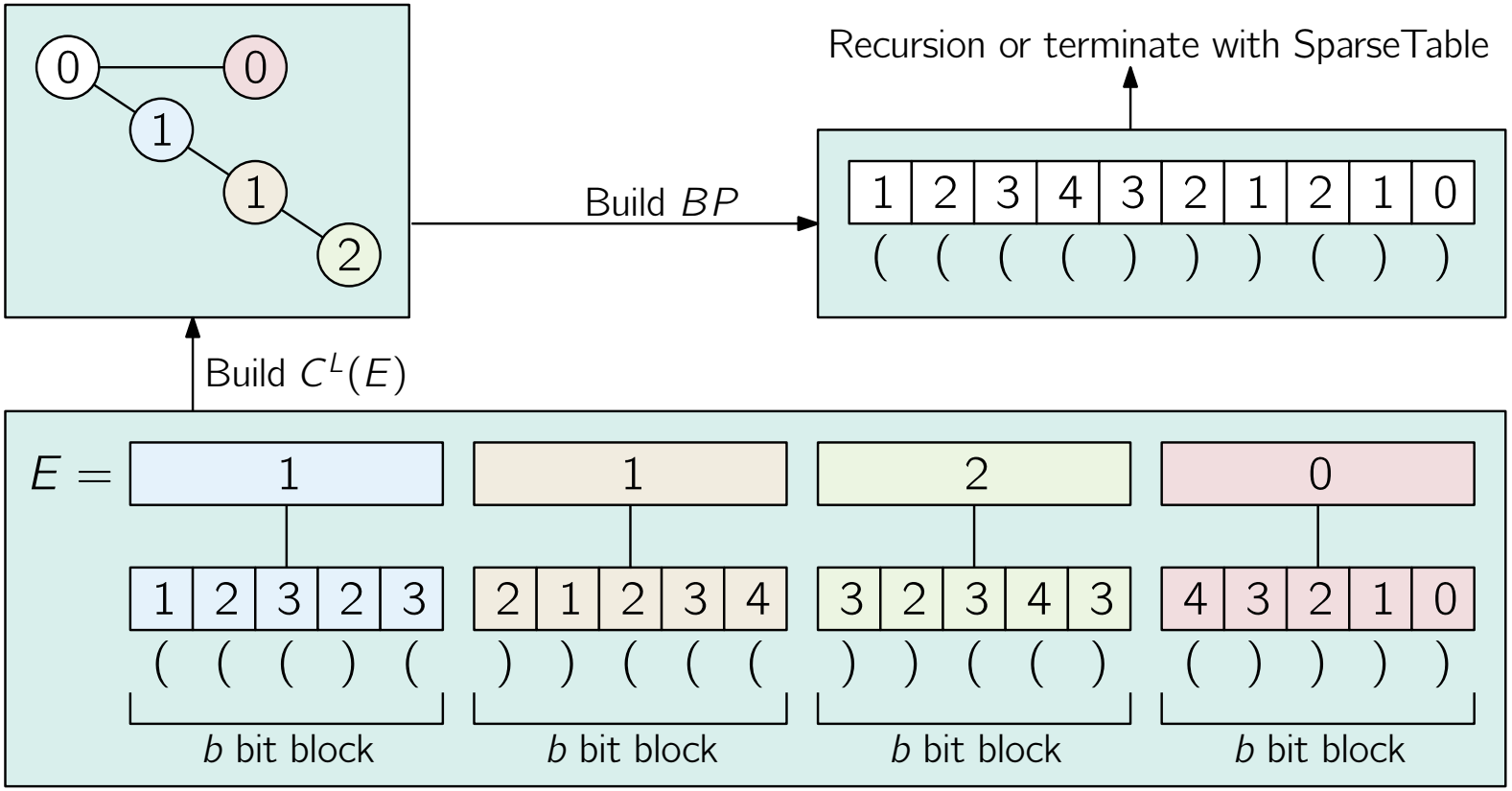


Solving $\pm 1\mathrm{RMQ}$ in practice - *rmM-Tree*

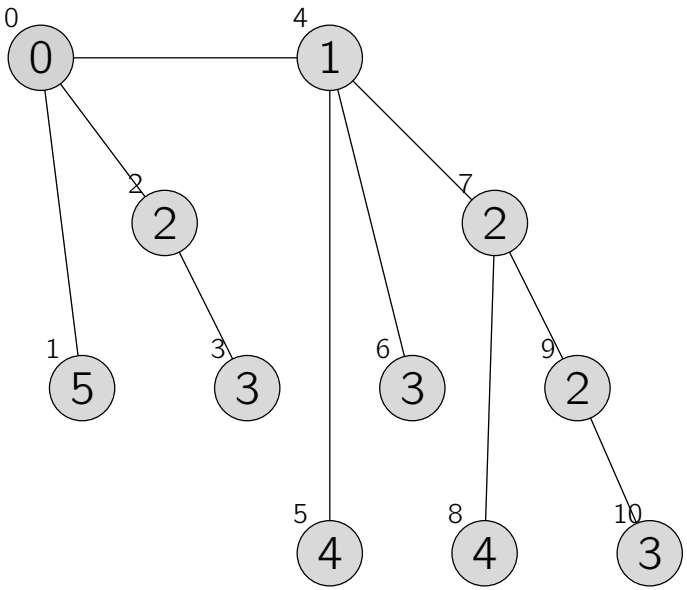


Nodes are enhanced with additional informations to solve EXCESS and SELECT [FN16] $\Rightarrow o(n)$ extra space and $\mathcal{O}(\log n)$ running time

Solving ±1RMQ - *Recursive Approach*



Improve SELECT

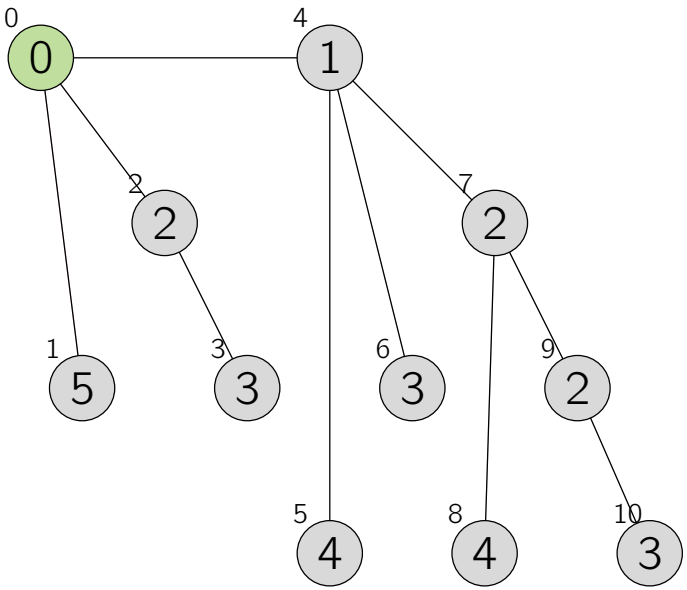


$BP =$

Lemma

The position of the representing opening parenthesis in BP of a node $v \in T$ *with* $\textsc{Preorder}(v) = i$ *is* $\textsc{select}_1(i, BP) = 2i - \textsc{depth}(v_i)$, *where* $\textsc{depth}(v) \geq 0$ *denotes the distance of* v *to the root node.*

Improve SELECT



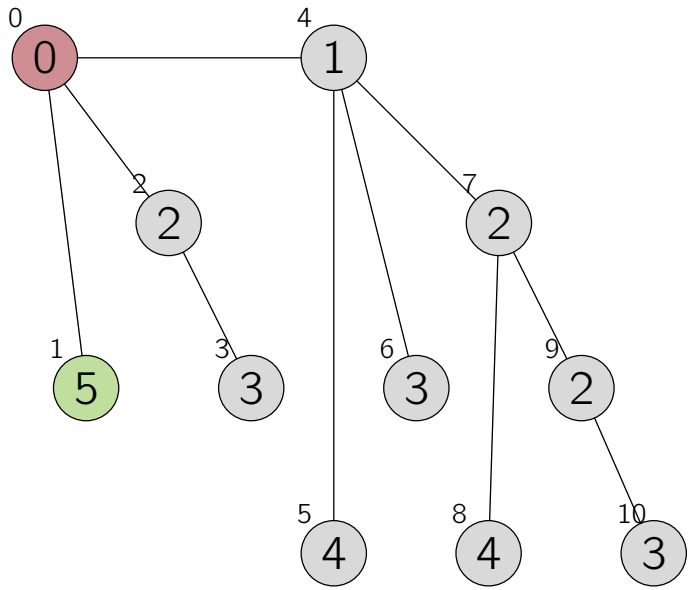
$$BP = \overset{0}{(}$$

$$\text{SELECT}_1(0, BP) = 2 \cdot 0 - 0 = 0$$

Lemma

The position of the representing opening parenthesis in BP of a node $v \in T$ *with* $\text{PREORDER}(v) = i$ *is* $\text{SELECT}_1(i, BP) = 2i - \text{DEPTH}(v_i)$, *where* $\text{DEPTH}(v) \geq 0$ *denotes the distance of* v *to the root node.*

Improve SELECT



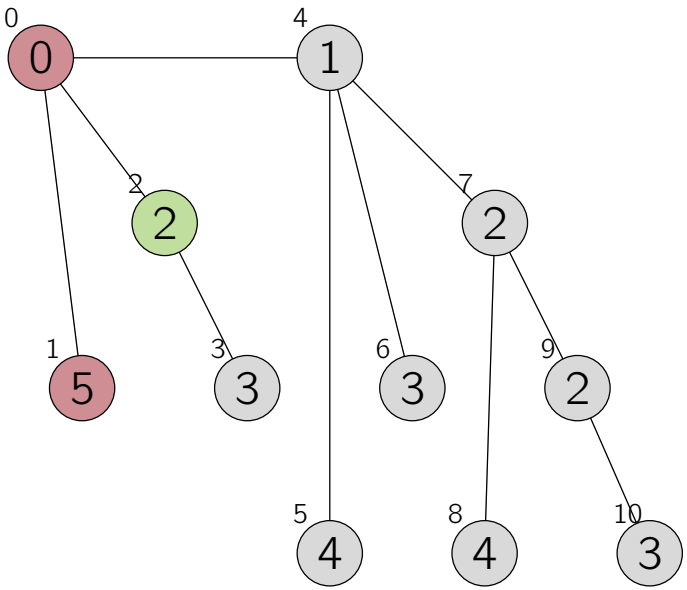
$$BP = \overset{0}{(}\ \overset{1}{(}$$

$$\text{SELECT}_1(1, BP) = 2 \cdot 1 - 1 = 1$$

Lemma

The position of the representing opening parenthesis in BP of a node $v \in T$ *with* $\text{PREORDER}(v) = i$ *is* $\text{SELECT}_1(i, BP) = 2i - \text{DEPTH}(v_i)$, *where* $\text{DEPTH}(v) \geq 0$ *denotes the distance of* v *to the root node.*

Improve SELECT



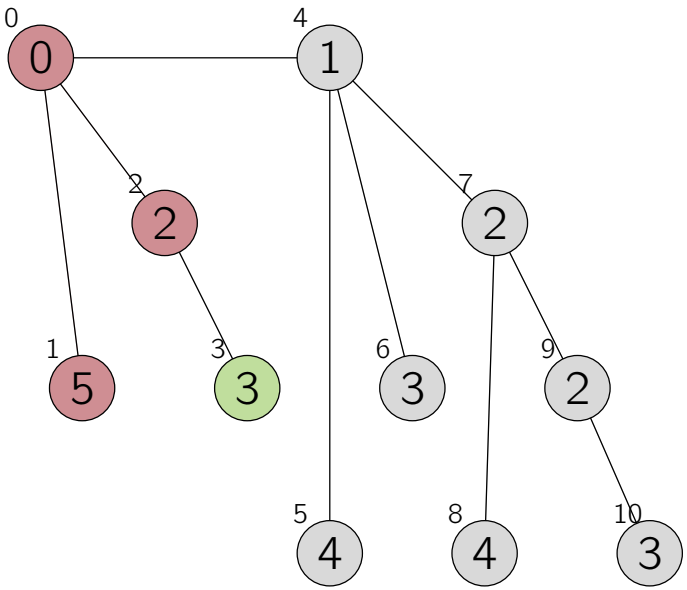
$$BP = \overset{0}{(}\ \overset{1}{(}\ )\ \ \overset{3}{(}$$

$$\text{SELECT}_1(2, BP) = 2 \cdot 2 - 1 = 3$$

Lemma

The position of the representing opening parenthesis in BP of a node $v \in T$ *with* $\text{PREORDER}(v) = i$ *is* $\text{SELECT}_1(i, BP) = 2i - \text{DEPTH}(v_i)$, *where* $\text{DEPTH}(v) \geq 0$ *denotes the distance of* v *to the root node.*

Improve SELECT



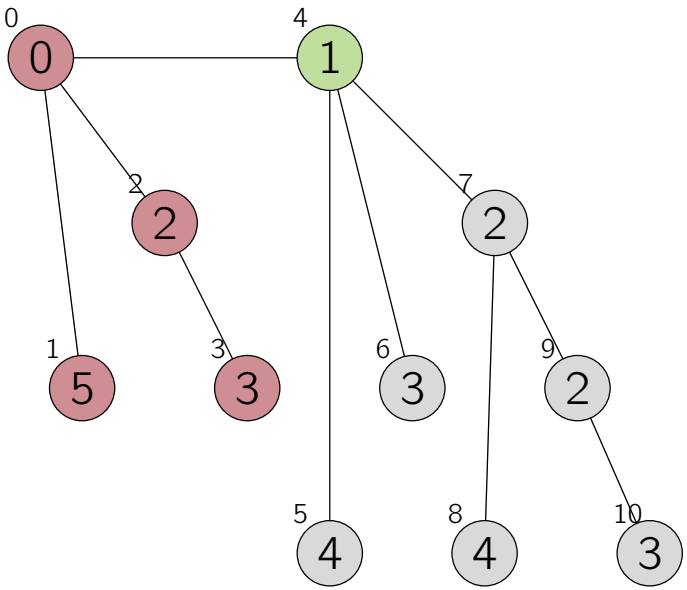
$$BP = \overset{0\ 1\ \ \ 3\ 4}{(\ (\ )\ \ (\ (}$$

$$\text{SELECT}_1(3, BP) = 2 \cdot 3 - 2 = 4$$

Lemma

The position of the representing opening parenthesis in BP of a node $v \in T$ *with* $\text{PREORDER}(v) = i$ *is* $\text{SELECT}_1(i, BP) = 2i - \text{DEPTH}(v_i)$, *where* $\text{DEPTH}(v) \geq 0$ *denotes the distance of* v *to the root node.*

Improve SELECT



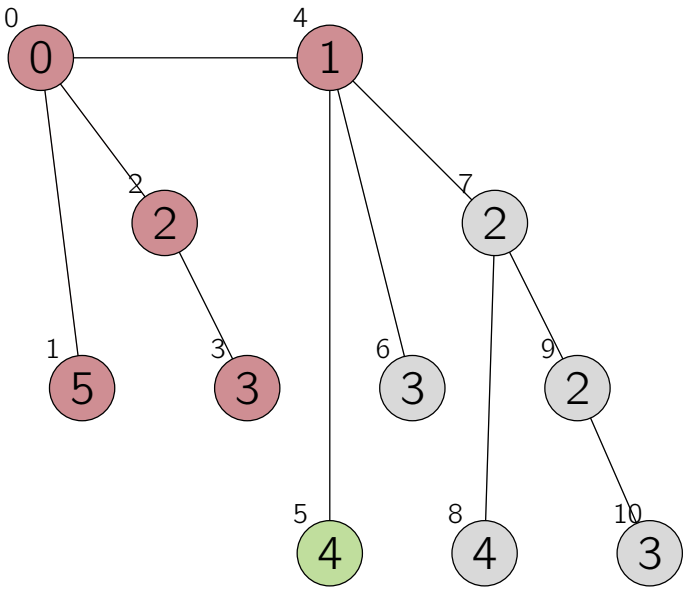
$$BP = \overset{0}{(}\ \overset{1}{(}\ )\ \overset{3}{(}\ \overset{4}{(}\ )\ )\ \overset{7}{(}$$

$$\text{SELECT}_1(4, BP) = 2 \cdot 4 - 1 = 7$$

Lemma

The position of the representing opening parenthesis in BP of a node $v \in T$ with $\text{PREORDER}(v) = i$ is $\text{SELECT}_1(i, BP) = 2i - \text{DEPTH}(v_i)$, where $\text{DEPTH}(v) \geq 0$ denotes the distance of v to the root node.

Improve SELECT



$$BP = \overset{0}{(}\ \overset{1}{(}\ )\ \overset{3}{(}\ \overset{4}{(}\ )\ )\ \overset{7}{(}\ \overset{8}{(}$$

$$\text{SELECT}_1(5, BP) = 2 \cdot 5 - 2 = 8$$

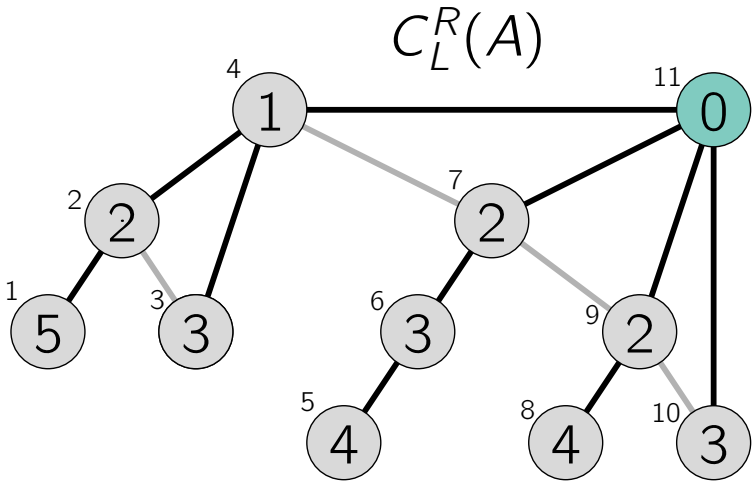
Lemma

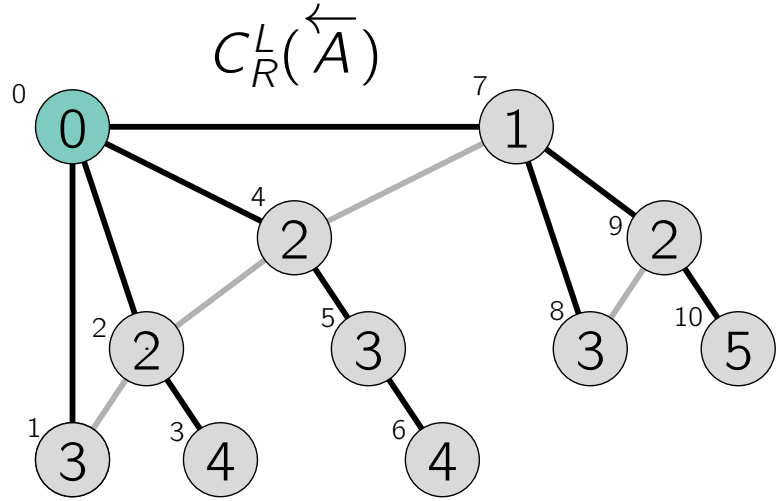
The position of the representing opening parenthesis in BP of a node $v \in T$ *with* $\text{PREORDER}(v) = i$ *is* $\text{SELECT}_1(i, BP) = 2i - \text{DEPTH}(v_i)$, *where* $\text{DEPTH}(v) \geq 0$ *denotes the distance of* v *to the root node.*

Consequences

- Solve $\text{SELECT}_1(i, BP)$ with binary search in interval $[2i - \text{MAX\_DEPTH}, 2i]$ with $\text{RANK}_1(i, BP)$

Minimize the depth of the Cartesian Tree





$A =$

1	2	3	4	5	6	7	8	9	10
5	2	3	1	4	3	2	4	2	3

$\overleftarrow{A} =$

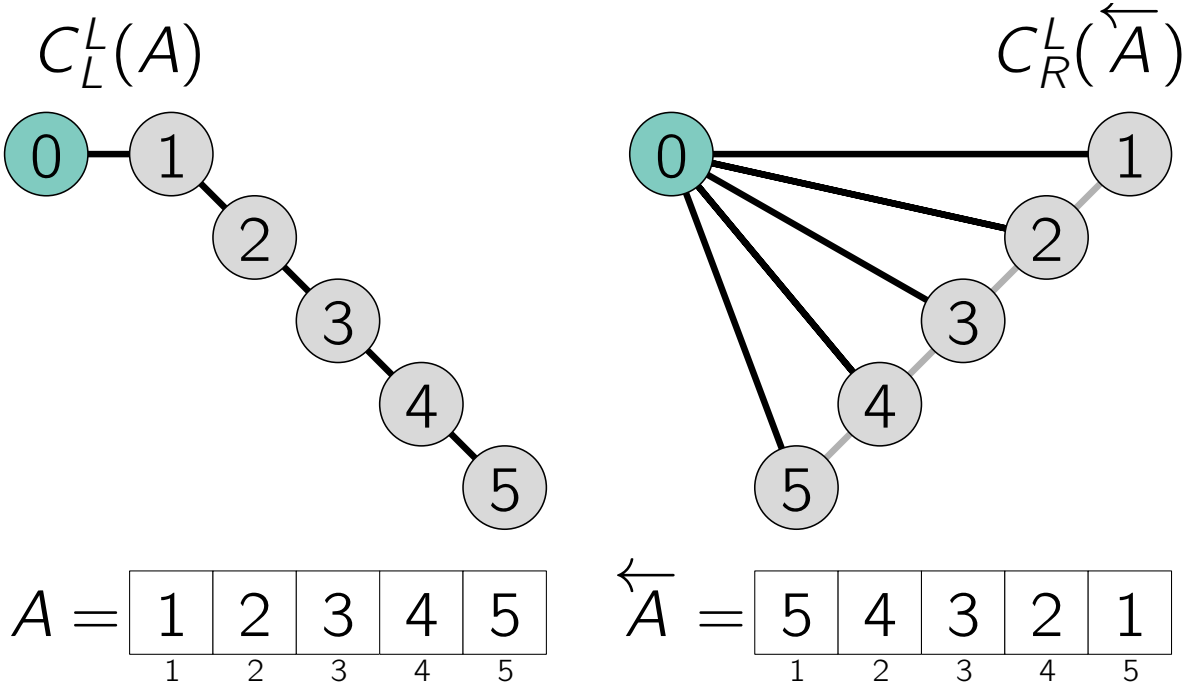
1	2	3	4	5	6	7	8	9	10
3	2	4	2	3	4	1	3	2	5

- *Cartesian Tree* over A with **leftmost** minimum as root
- Use **rightmost** path mapping for *Generalized Cartesian Tree*

- *Cartesian Tree* over $\overleftarrow{A}$ with **rightmost** minimum as root
- Use **leftmost** path mapping for *Generalized Cartesian Tree*

$\Rightarrow C_L^R(A)$ and $C_R^L(\overleftarrow{A})$ are isomorph.

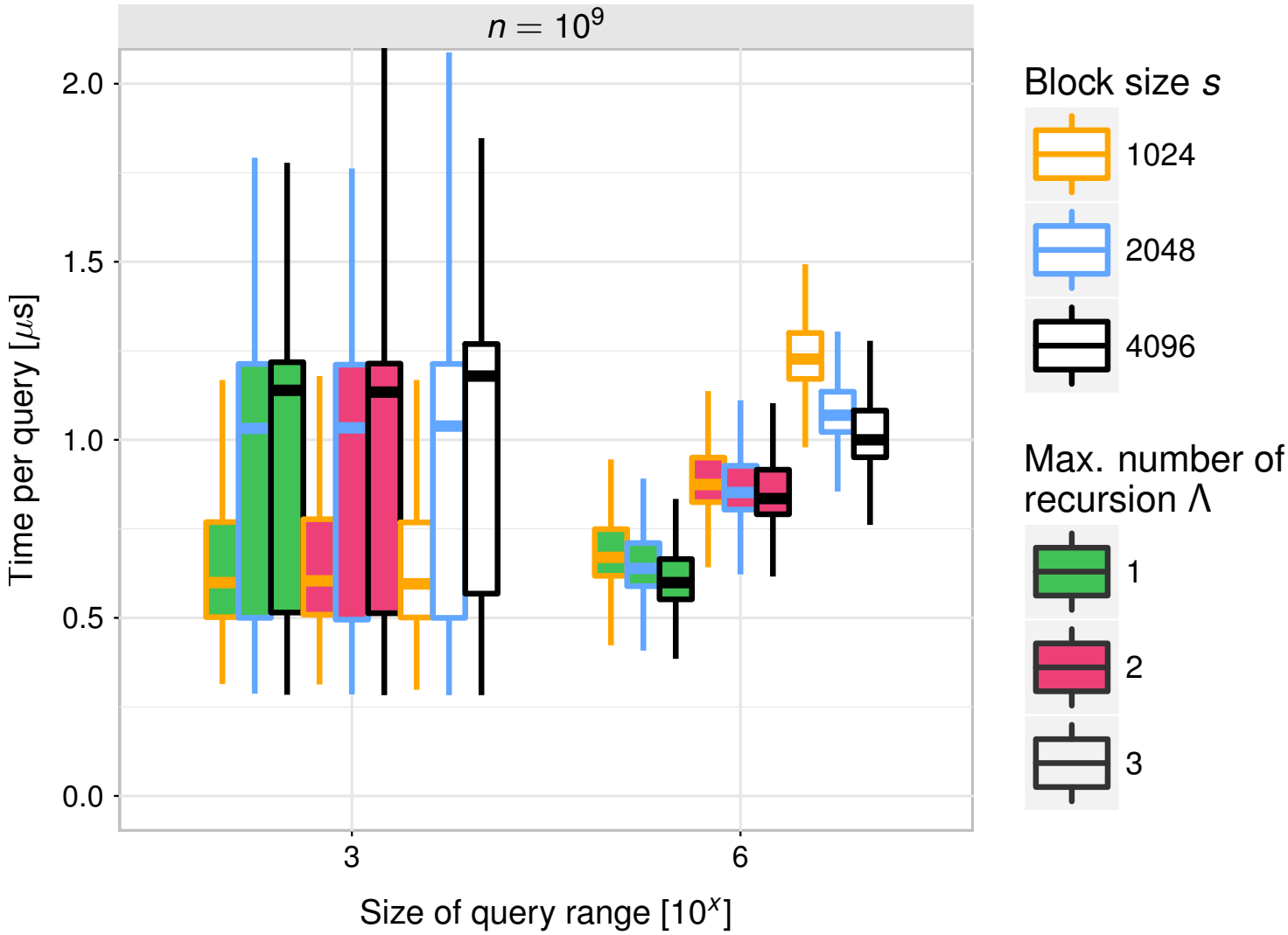
Minimize the depth of the Cartesian Tree



- Build $C_L^L(A)$ and $C_R^L(\overleftarrow{A}) \Rightarrow$ Continue with *Cartesian Tree* with minimal height
- Query logic does not change for $\pm$1RMQ$\Rightarrow$ only mirror result of a RMQ call on $C_R^L(\overleftarrow{A})$
- Accelerates SELECT

Parameter Tuning

Query Time



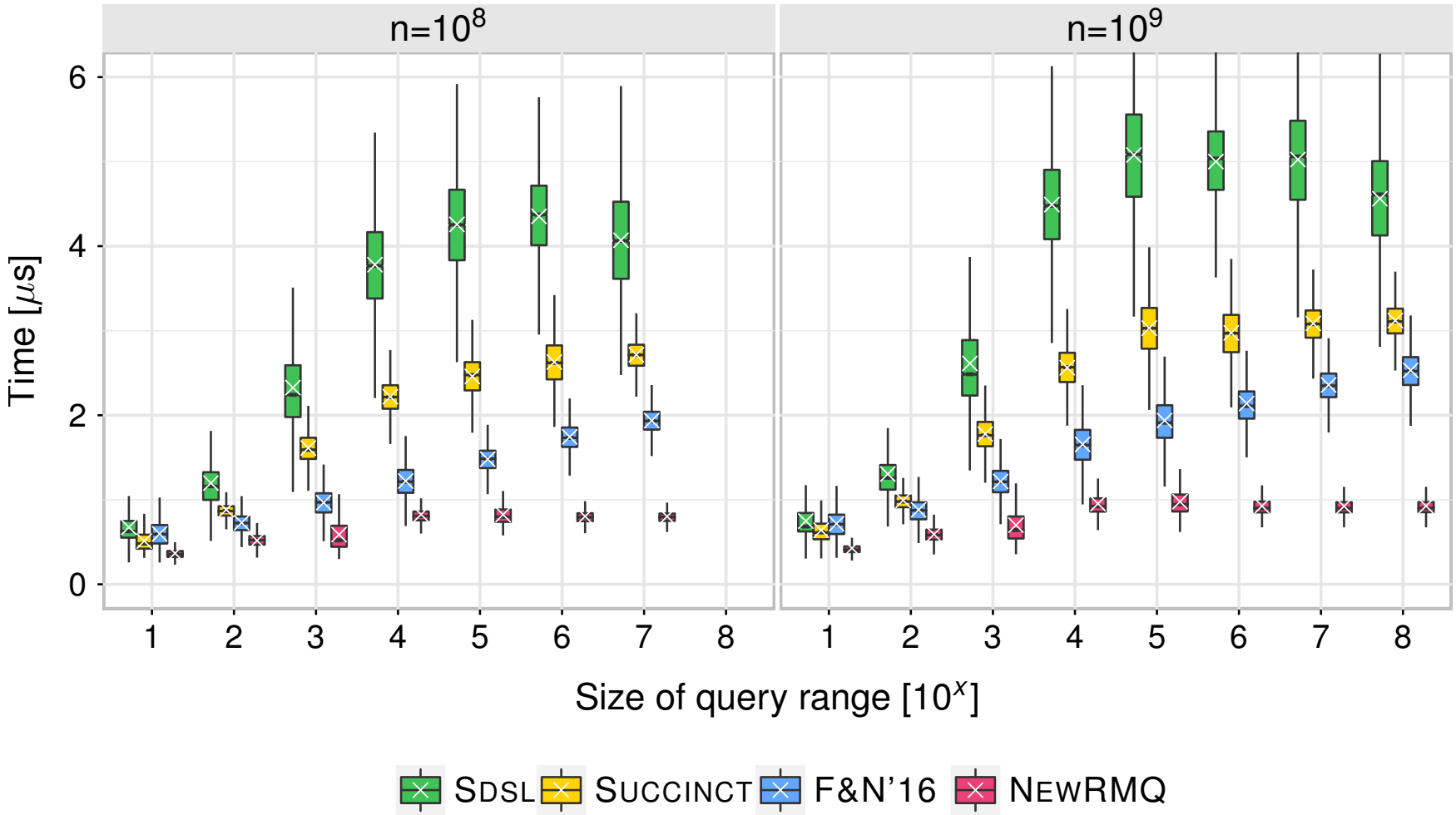
Parameter Tuning

Space

Implementation	Space in bits per element with varying n			
	$n = 10^6$	$n = 10^7$	$n = 10^8$	$n = 10^9$
$s = 1024,\ \Lambda = 1$	2.37	2.44	2.54	2.65
$s = 2048,\ \Lambda = 1$	2.24	2.27	2.32	2.37
$s = 4096,\ \Lambda = 1$	2.18	2.19	2.21	2.24
$s = 1024,\ \Lambda = 2$	2.16	2.16	2.16	2.17
$s = 2048,\ \Lambda = 2$	2.14	2.14	2.14	2.15
$s = 4096,\ \Lambda = 2$	2.14	2.14	2.14	2.14
$s = 1024,\ \Lambda = 3$	2.16	2.16	2.16	2.16
$s = 2048,\ \Lambda = 3$	2.14	2.14	2.14	2.14
$s = 4096,\ \Lambda = 3$	2.14	2.13	2.13	2.13

Comparison with other Implementations

Random Input



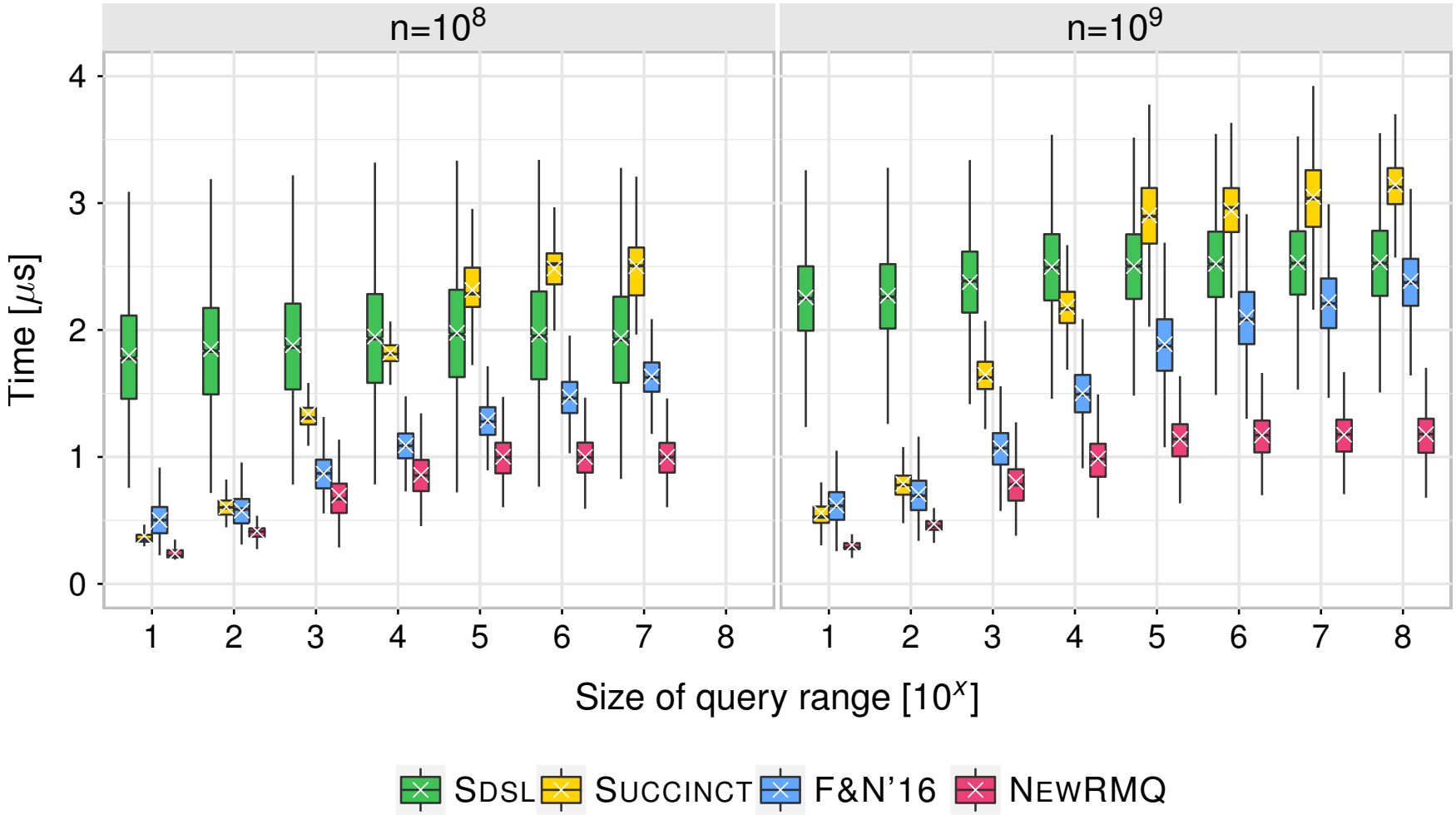
Comparison with other Implementations

Random Input

Implementation	Space in bits per element with varying n			
	$n = 10^6$	$n = 10^7$	$n = 10^8$	$n = 10^9$
SDSL	2.61	2.55	2.54	2.54
SUCCINCT	2.70	2.71	2.71	2.70
F&N'16	2.10	2.09	2.10	2.10
NEWRMQ	2.16	2.16	2.16	2.17

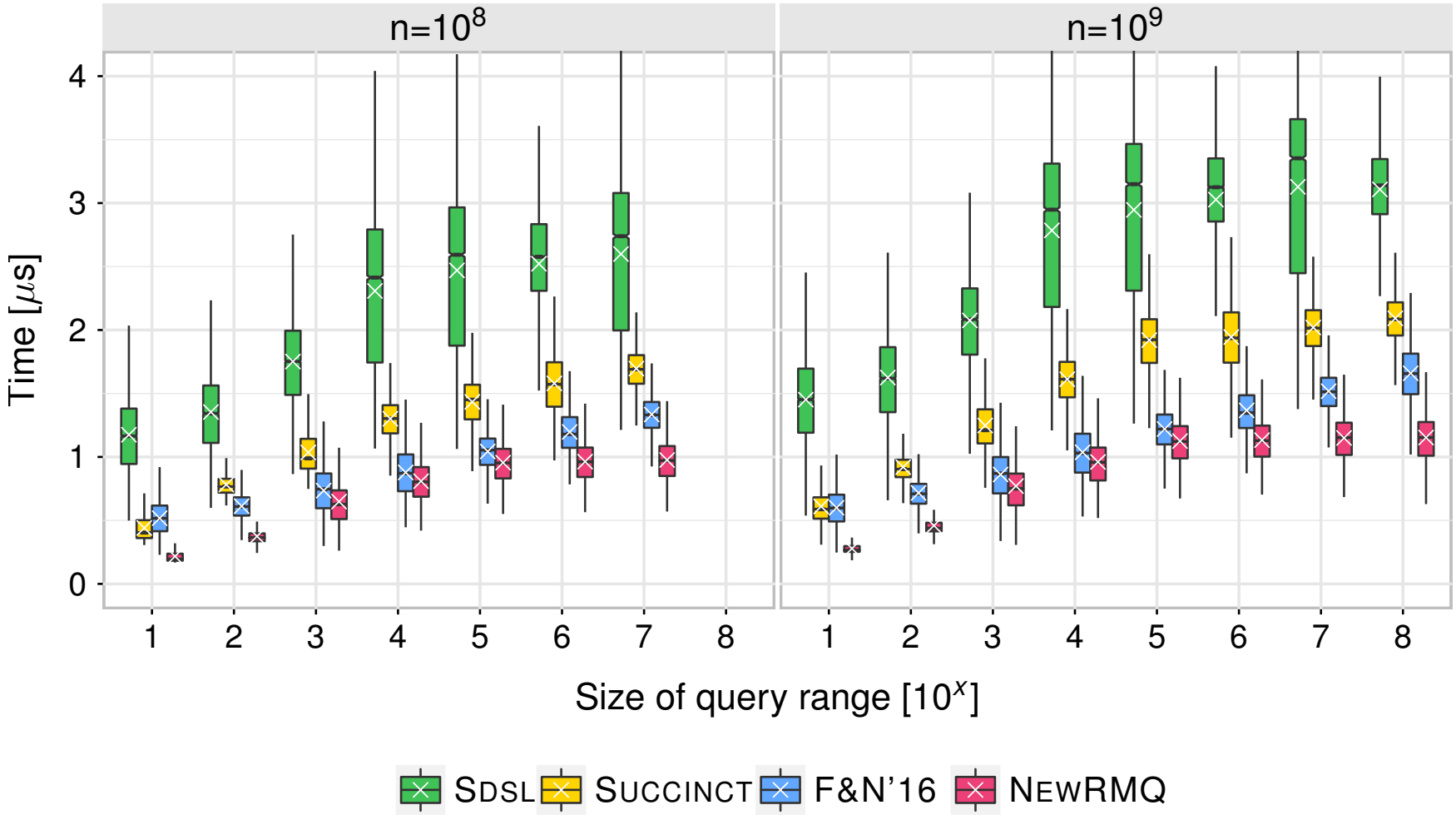
Comparison with other Implementations

Increasing Input

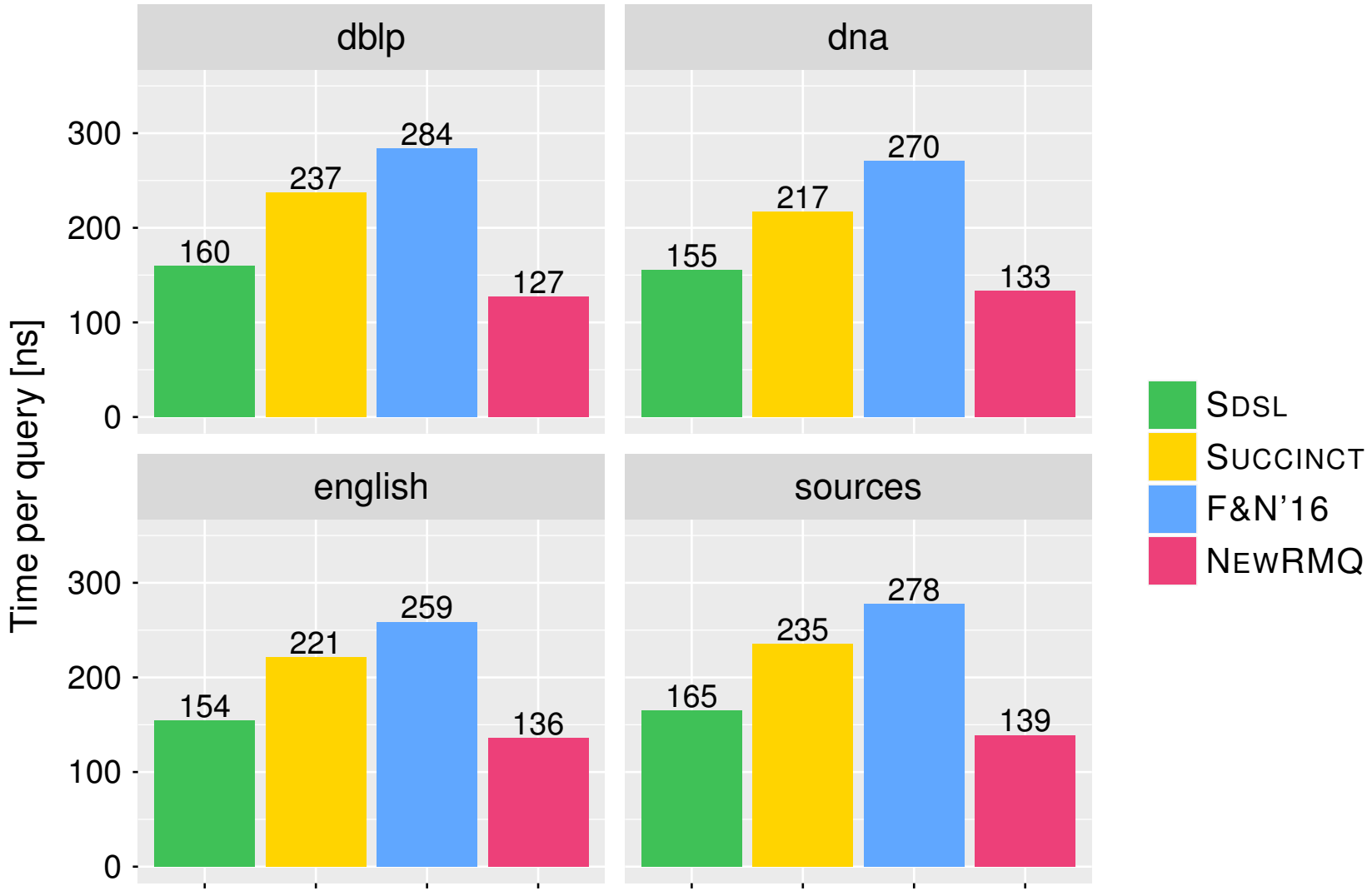


Comparison with other Implementations

Deacreasing Input



Traversing the *Suffix Tree* with RMQ



References I

M. Bender and M. Farach-Colton. “The LCA problem revisited”. In: *Proc. LATIN*. Springer. 2000, pp. 88–94.

O. Berkman and U. Vishkin. “Recursive star-tree parallel data structure”. In: *SIAM Journal on Computing* 22.2 (1993), pp. 221–242.

J. Fischer and V. Heun. “Space-efficient preprocessing schemes for range minimum queries on static arrays”. In: *SIAM Journal on Computing* 40.2 (2011), pp. 465–492.

H. Ferrada and G. Navarro. “Improved range minimum queries”. In: *Proc. DCC*. 2016, pp. 516–525.

H. Gabow, J. Bentley, and R. Tarjan. “Scaling and related techniques for geometry problems”. In: *Proc. STOC*. ACM. 1984, pp. 135–143.

References II


Guy Joseph Jacobson. “Succinct static data structures”. In: (1988).


J. Munro and V. Raman. “Succinct representation of balanced parentheses and static trees”. In: *SIAM Journal on Computing* 31.3 (2001), pp. 762–776.


K. Sadakane. “Compressed suffix trees with full functionality”. In: *Theory of Computing Systems* 41.4 (2007), pp. 589–607.


J. Vuillemin. “A unifying look at data structures”. In: *Communications of the ACM* 23.4 (1980), pp. 229–239.

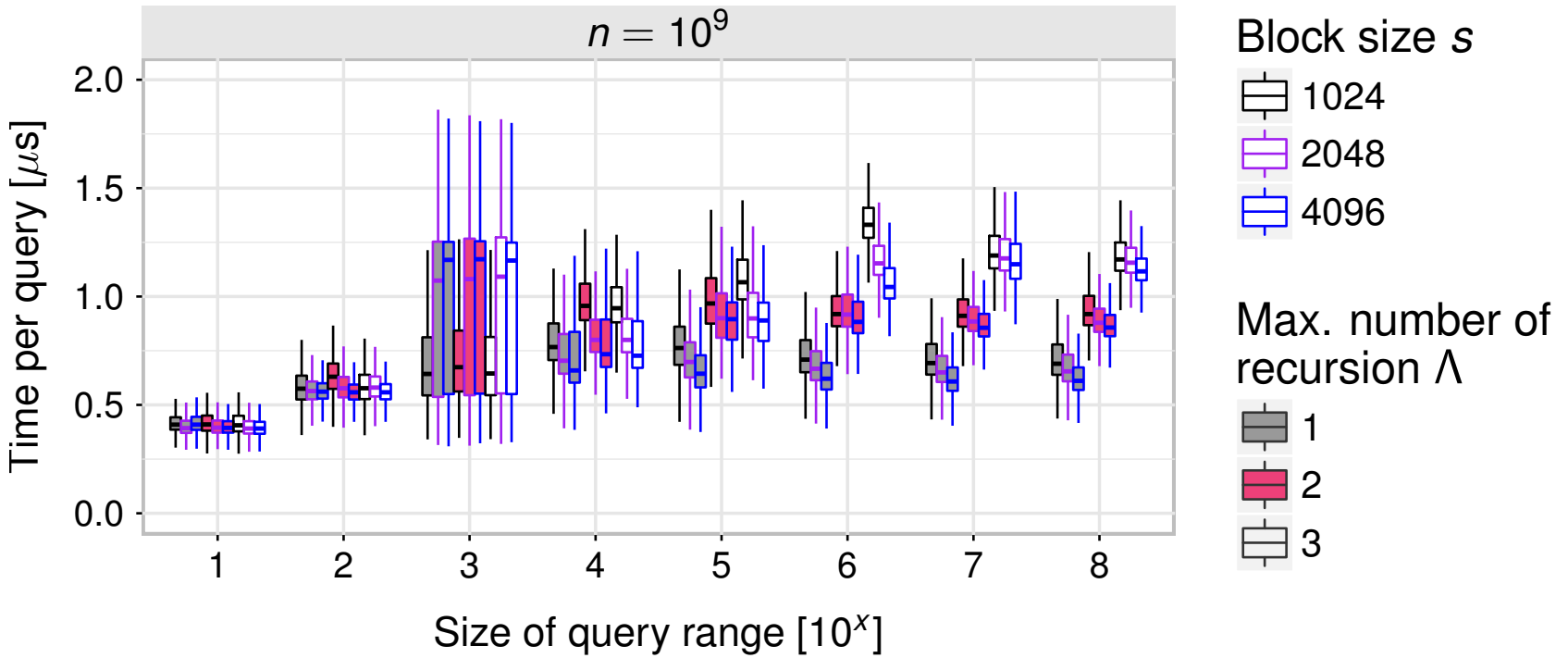
Additional Implementations Details

$$\mathrm{RMQ}_A(i,j) = \mathrm{RANK}_1(\mathrm{RRMQ}^{\pm}_{BP(C^L)}(\mathrm{SELECT}_1(i+1)-1, \mathrm{SELECT}_1(j+1)))$$

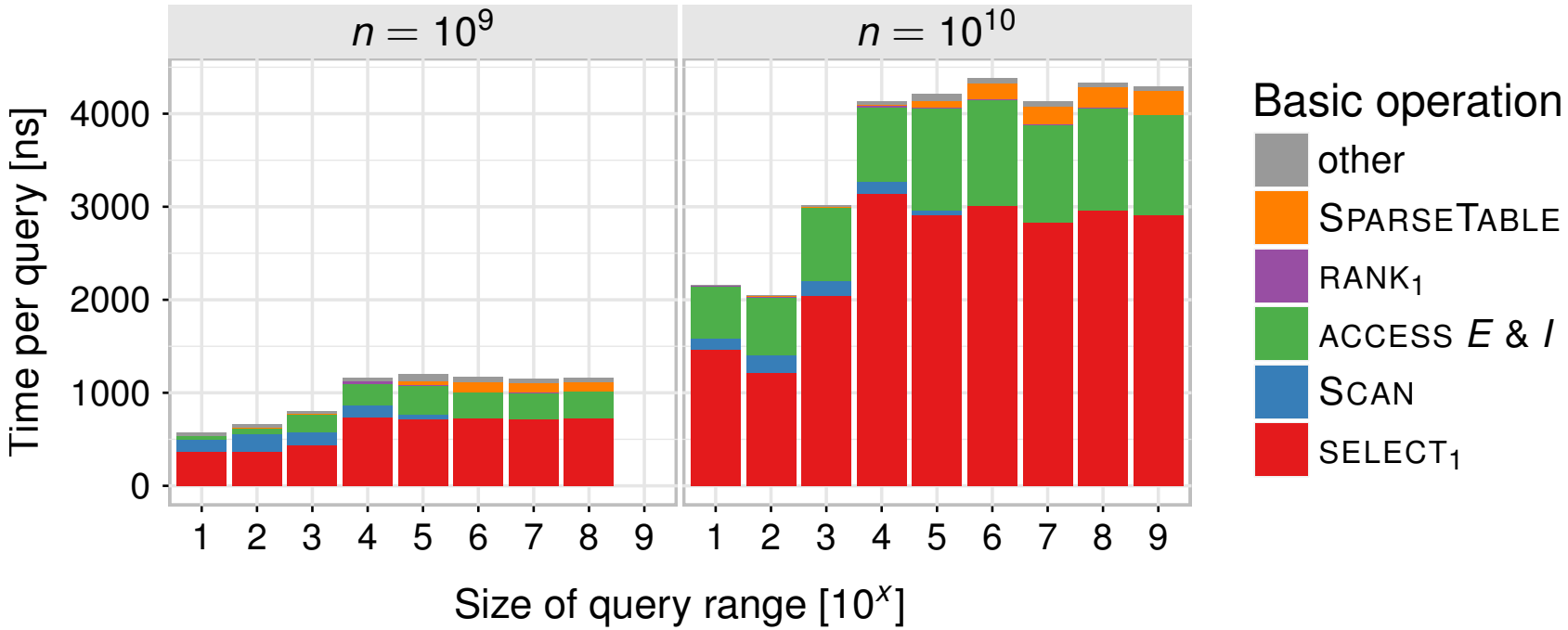
- If $j - i \leq \log n$, then try to avoid second SELECT by simple scan of the next $2 \log n$ bits starting at position $\mathrm{SELECT}_1(i+1)$
- For large tree sizes use sampling scheme ($o(n)$ space) for SELECT to prevent $\mathcal{O}(\log n)$ running time

Parameter Tuning

Query Time

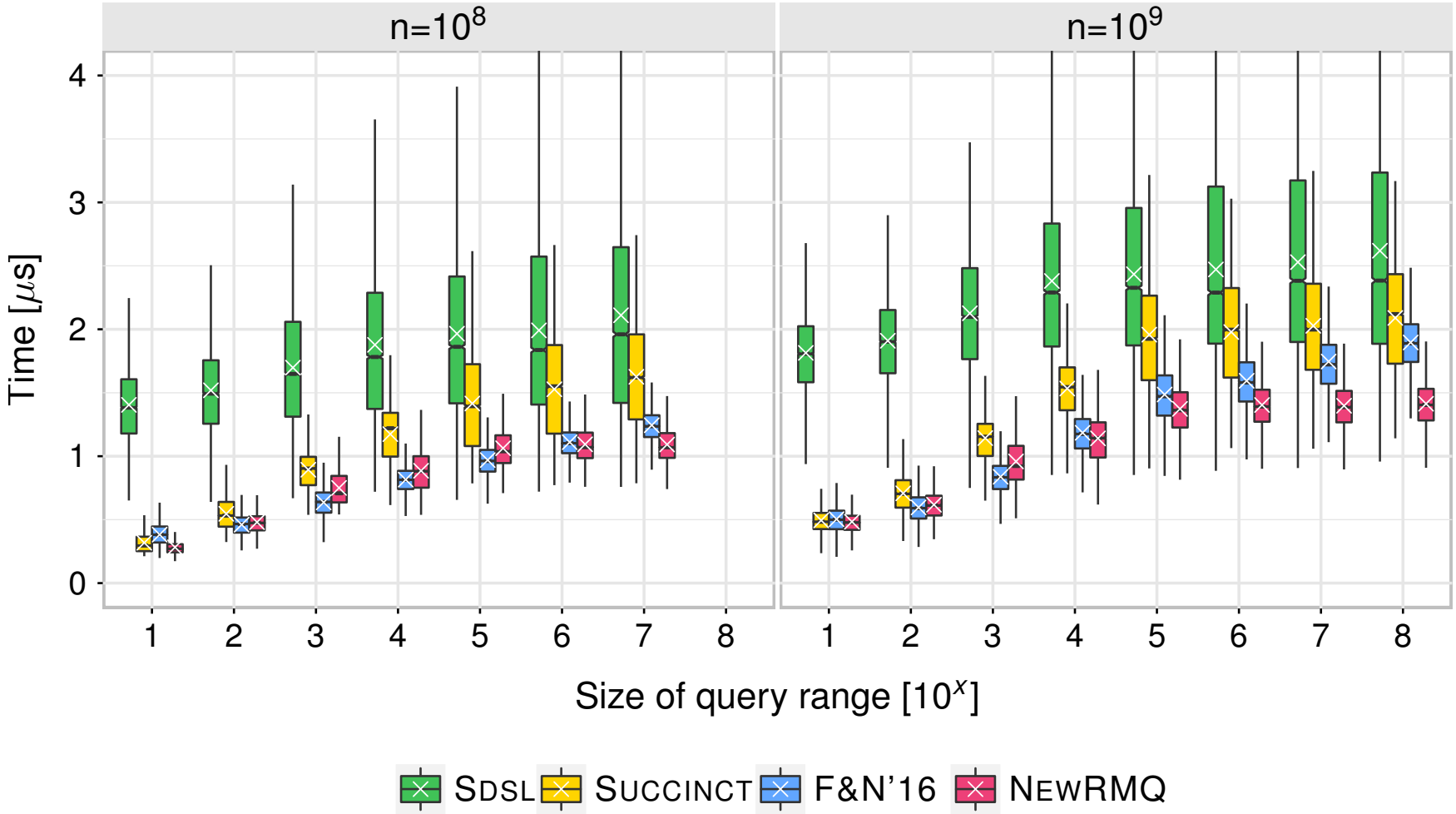


Query Time Breakdown



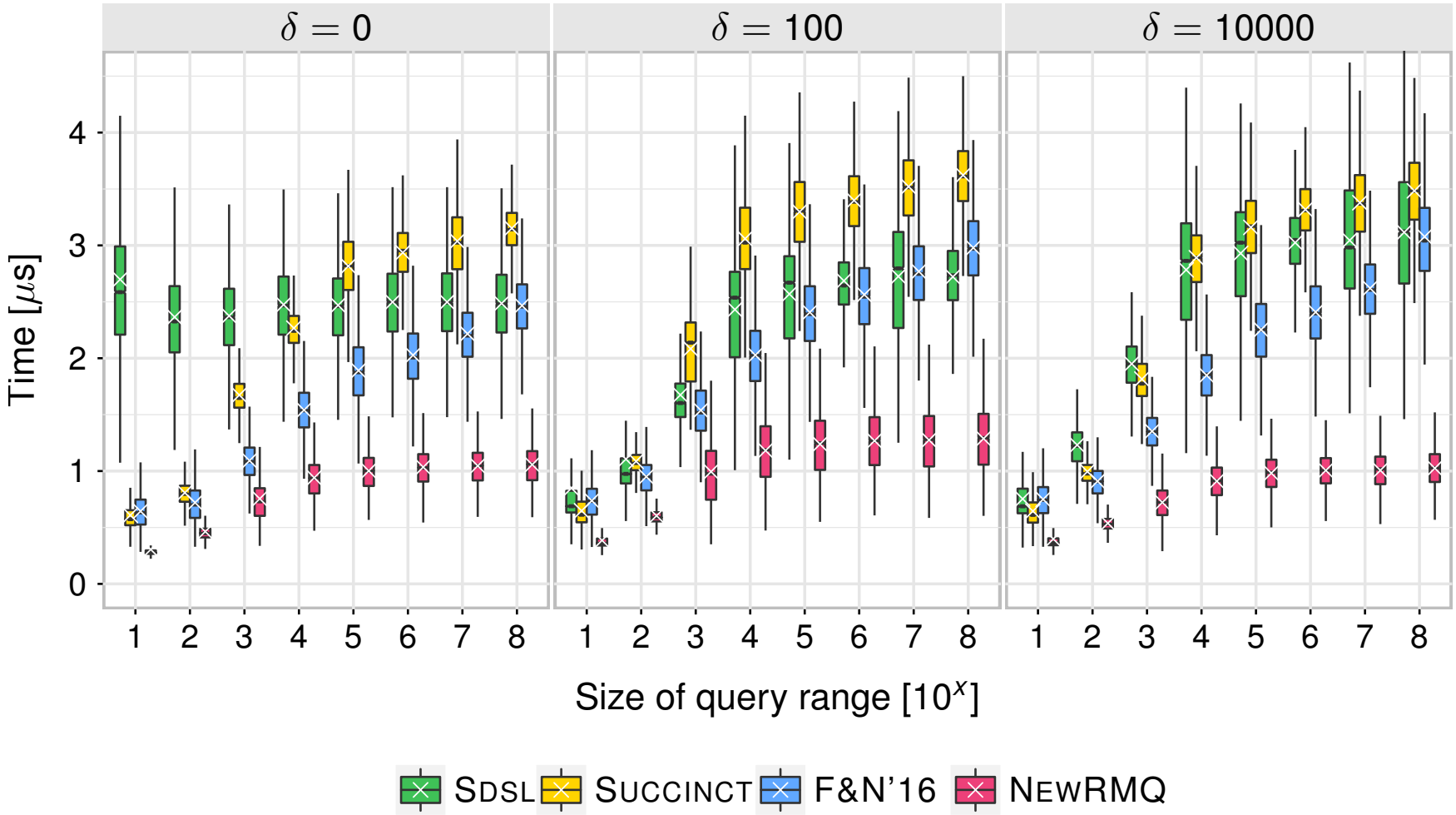
Comparison with other Implementations

Worst Case Input



Comparison with other Implementations

Increasing Input



Comparison with other Implementations

Deacreasing Input