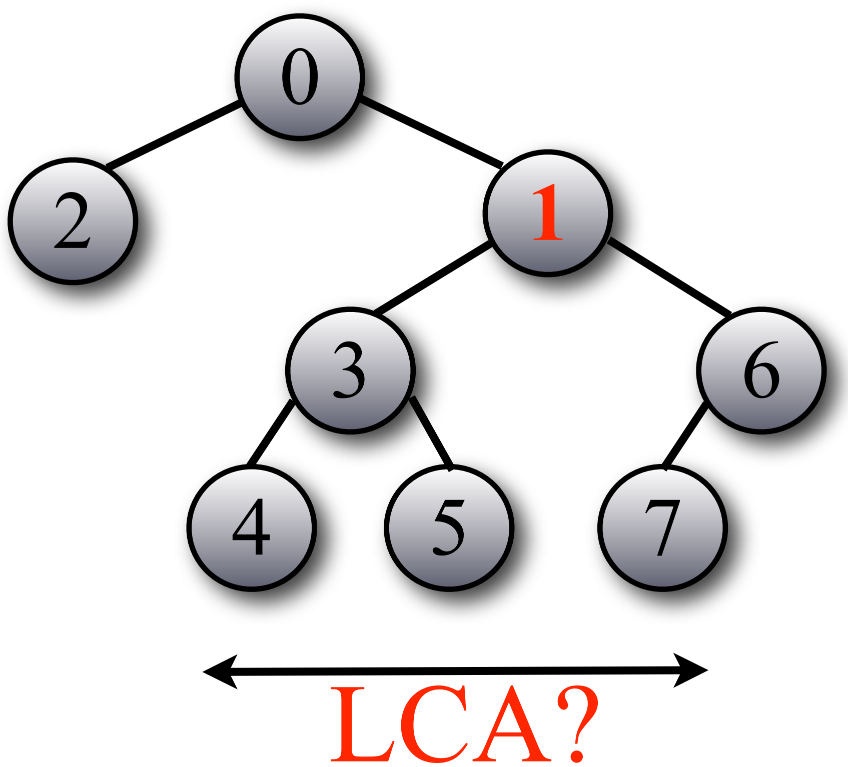
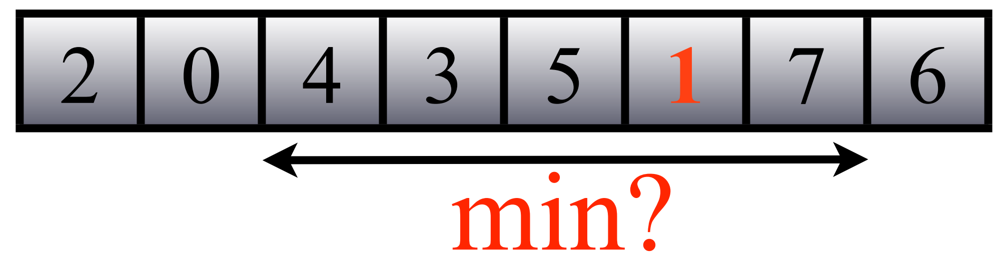
On Cartesian Trees, Lowest Common Ancestors, and Range Minimum Queries



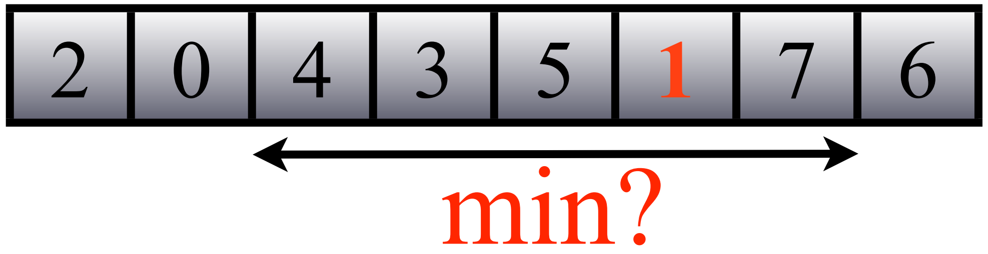


Parallel Computing Day Ben-Gurion University

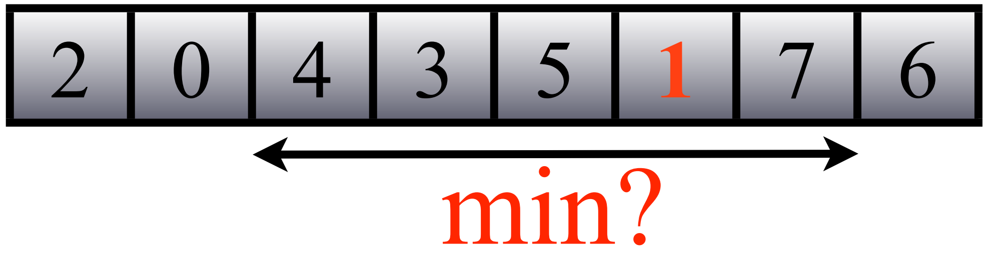
RMQ

2	0	4	3	5	1	7	6

RMQ



RMQ



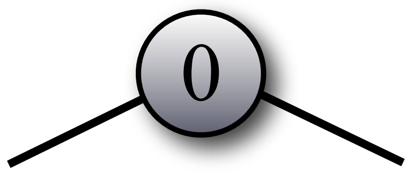
- <u>Applications:</u>
  - String Processing & Computational Biology
  - Search Engines and Document Retrieval
  - Equivalence to LCA
  - Database Queries

⋮

RMQ
& Cartesian Trees

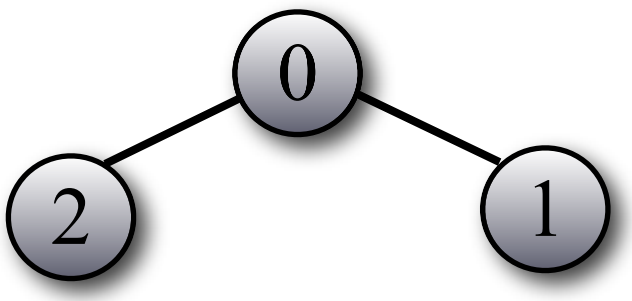
2	0	4	3	5	1	7	6

RMQ
& Cartesian Trees



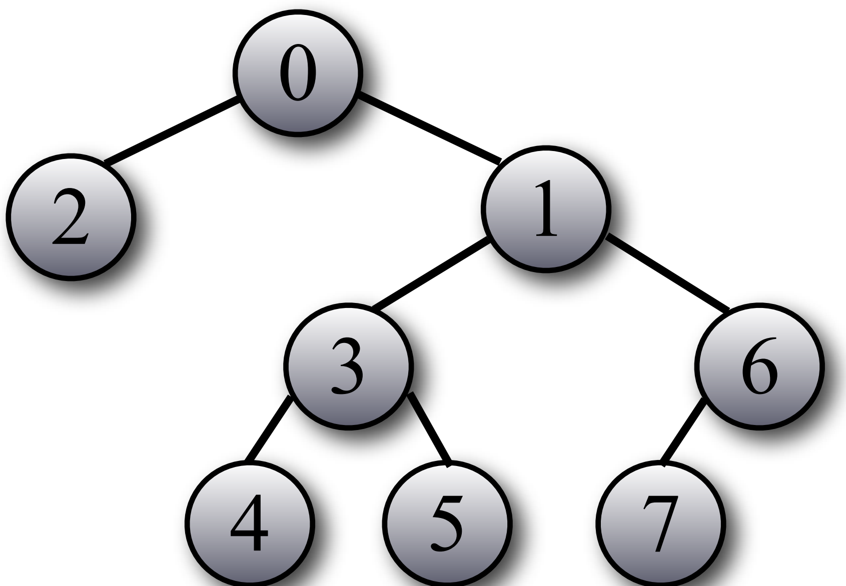
2	0	4	3	5	1	7	6

RMQ
& Cartesian Trees



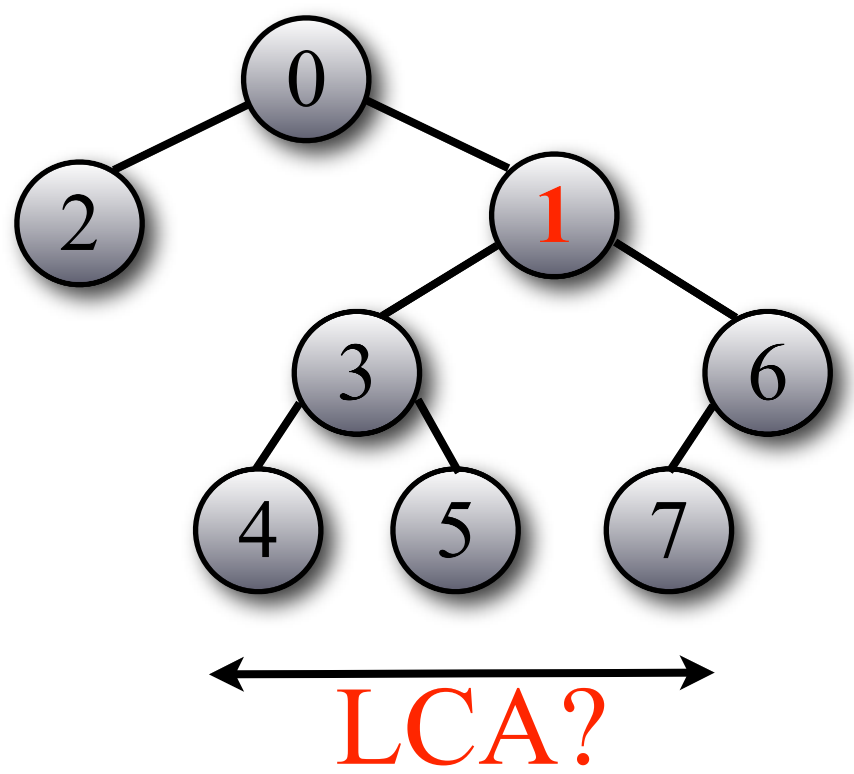
2	0	4	3	5	1	7	6

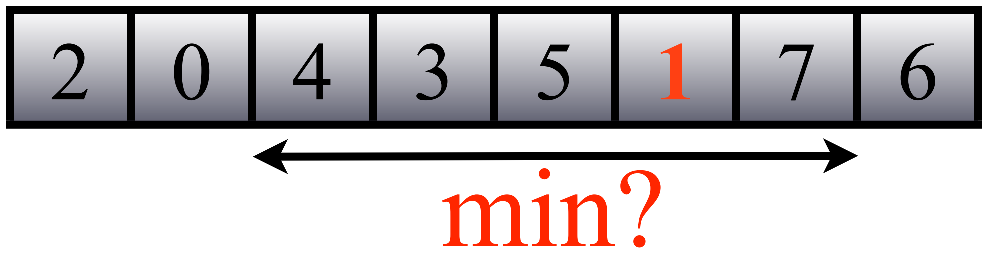
RMQ
& Cartesian Trees



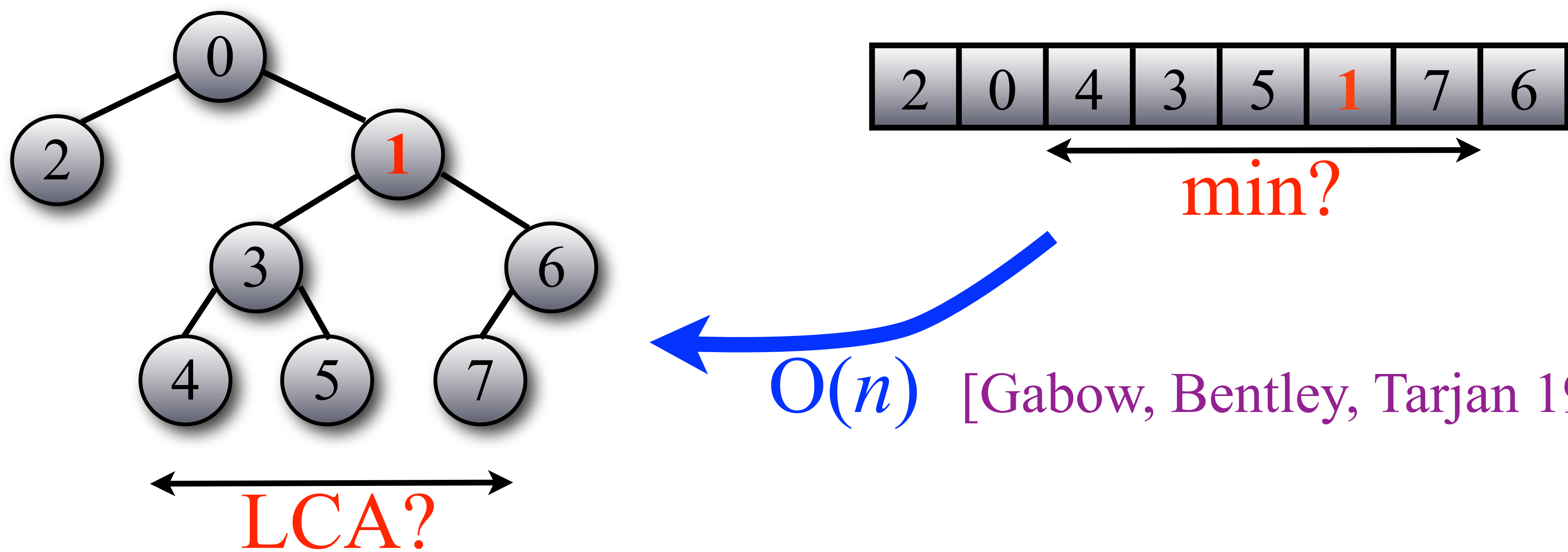
2	0	4	3	5	1	7	6

RMQ
& Cartesian Trees



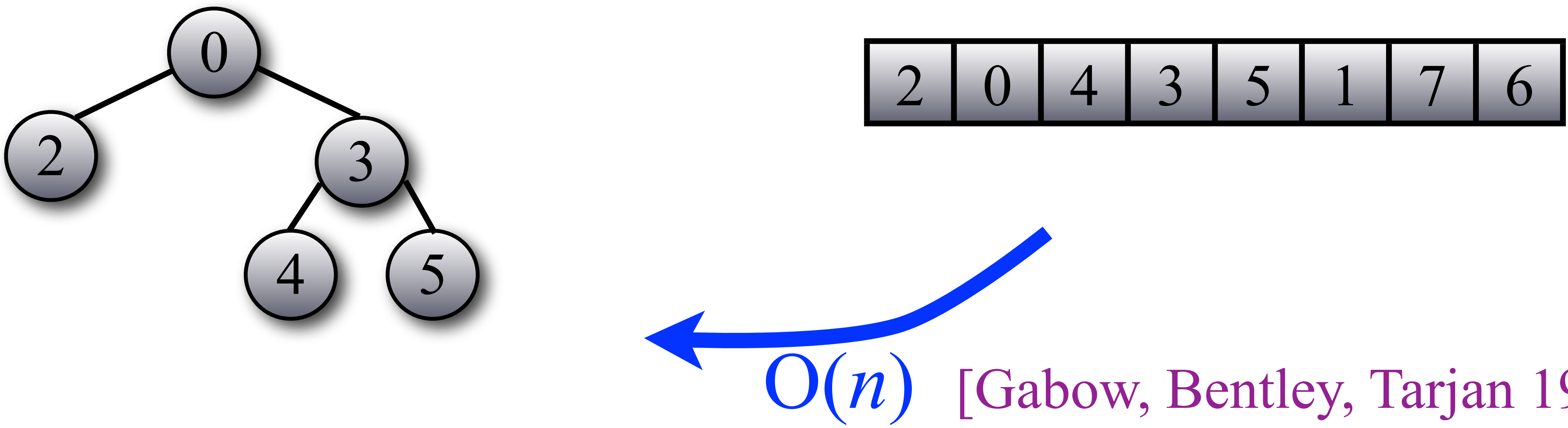


RMQ
& Cartesian Trees



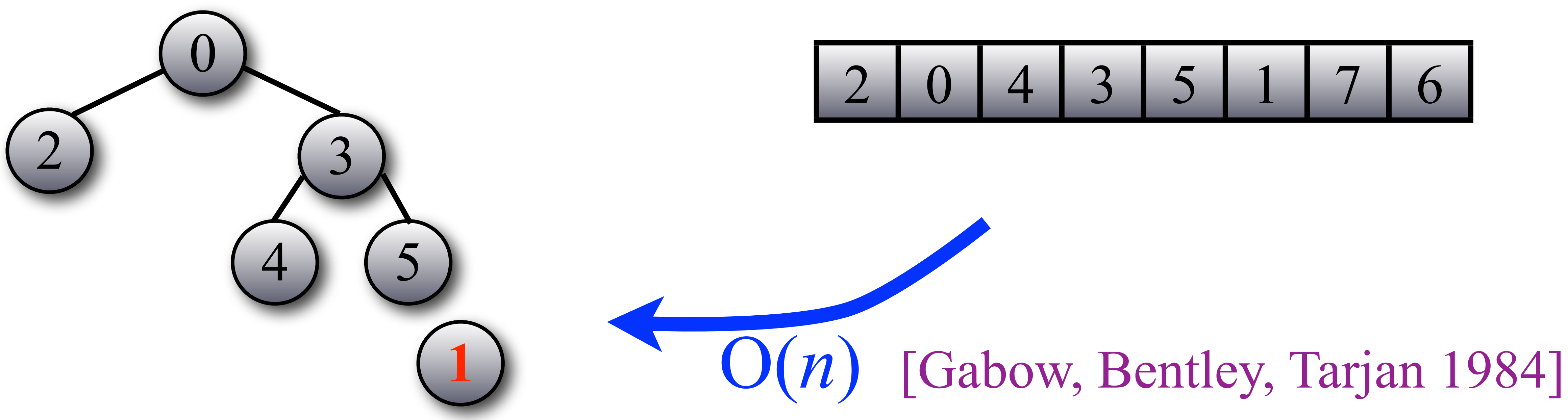
[Gabow, Bentley, Tarjan 1984]

RMQ
& Cartesian Trees

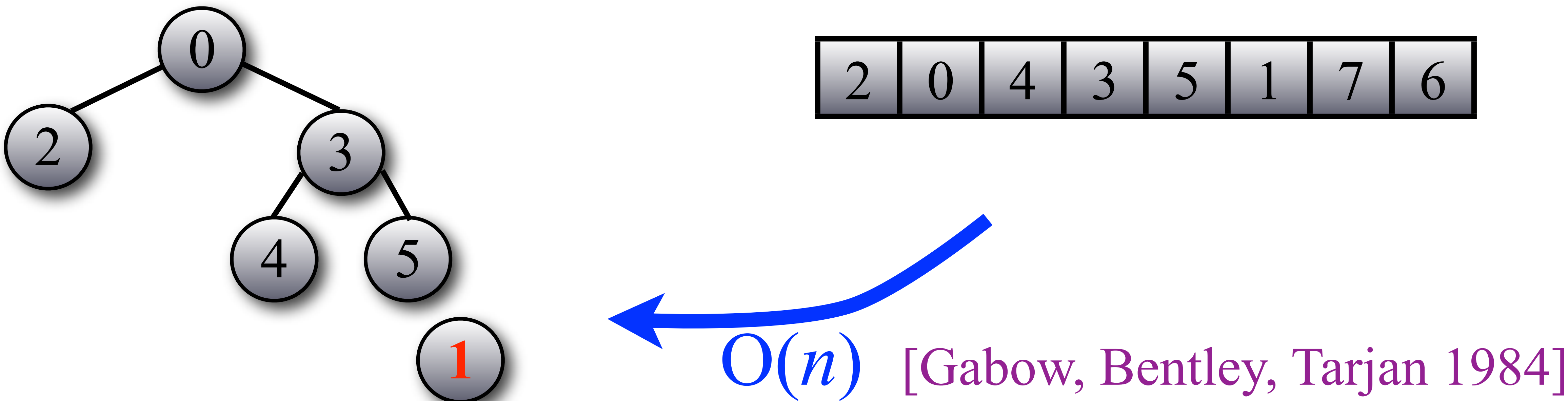


[Gabow, Bentley, Tarjan 1984]

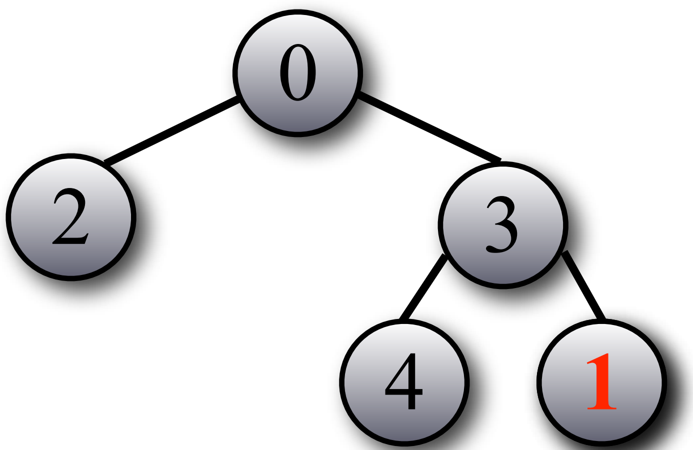
RMQ
& Cartesian Trees



RMQ
& Cartesian Trees



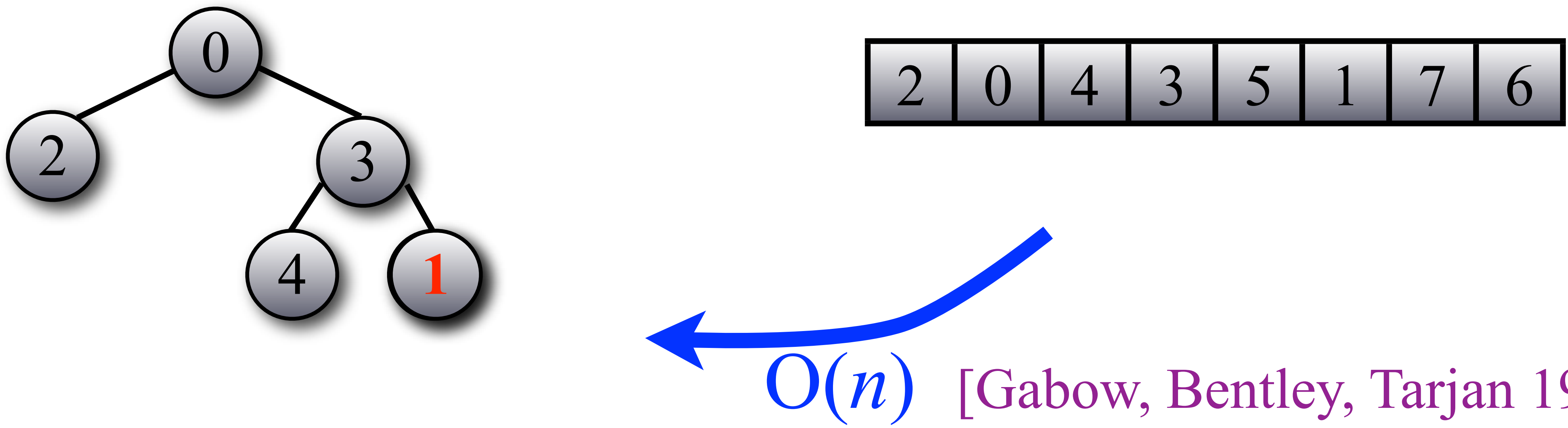
RMQ
& Cartesian Trees



2	0	4	3	5	1	7	6

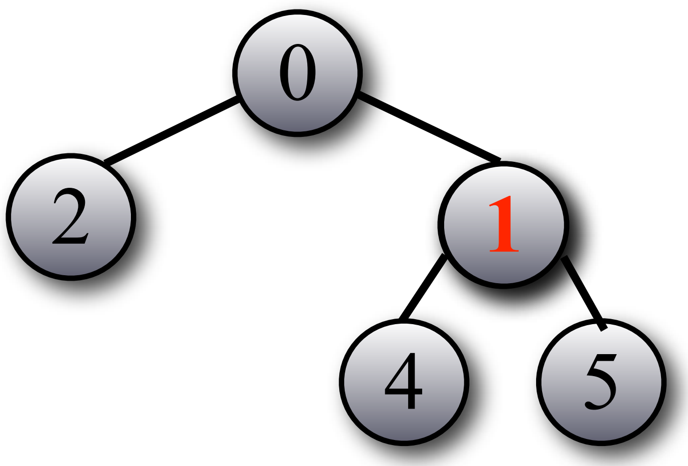
$O(n)$ [Gabow, Bentley, Tarjan 1984]

RMQ
& Cartesian Trees



[Gabow, Bentley, Tarjan 1984]

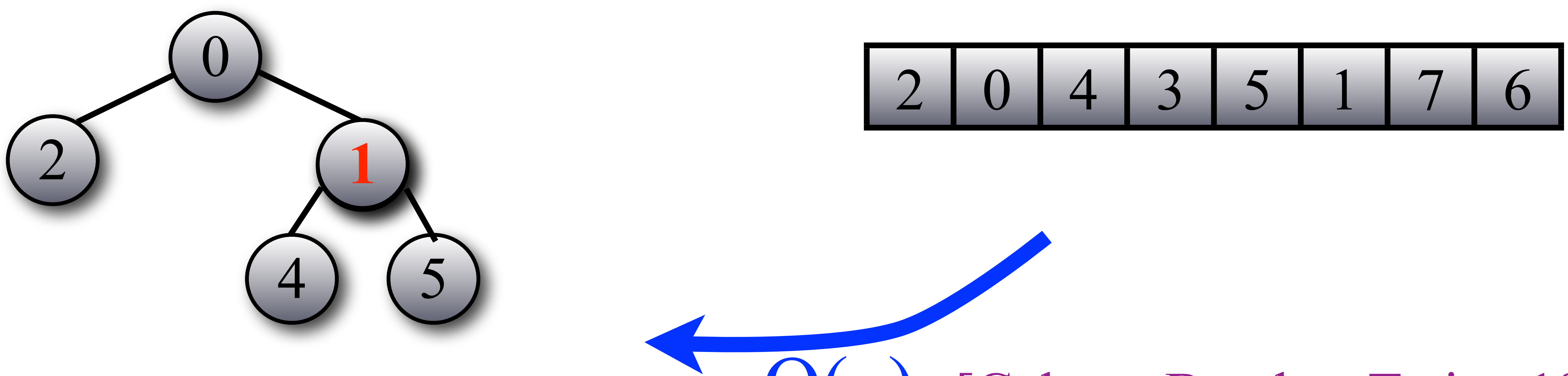
RMQ
& Cartesian Trees



2	0	4	3	5	1	7	6

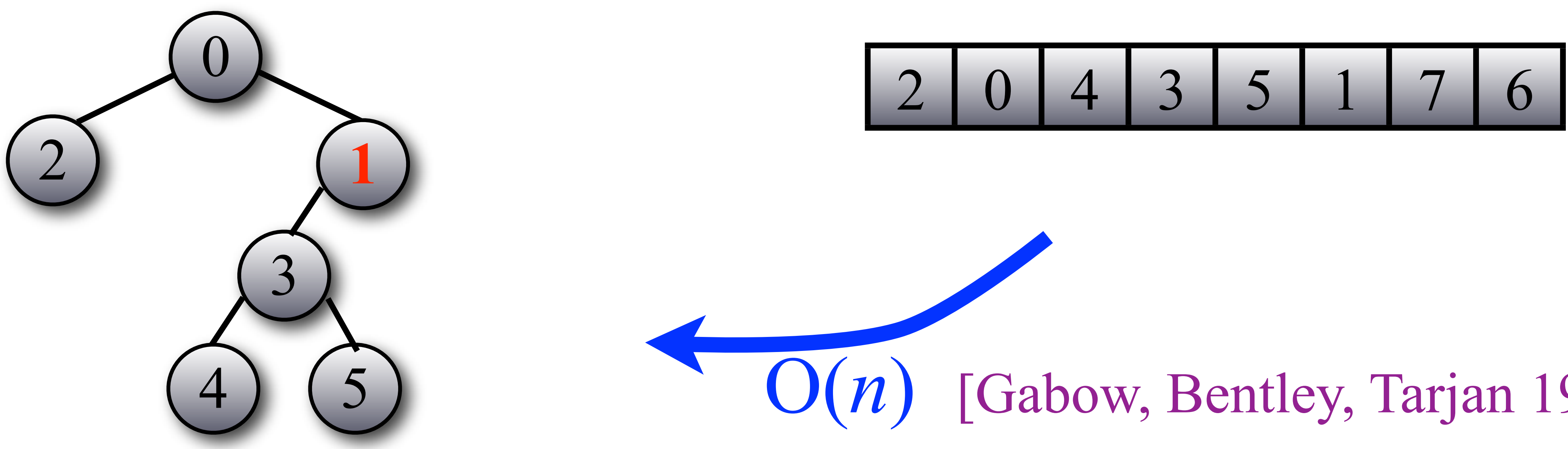
$O(n)$ [Gabow, Bentley, Tarjan 1984]

RMQ
& Cartesian Trees



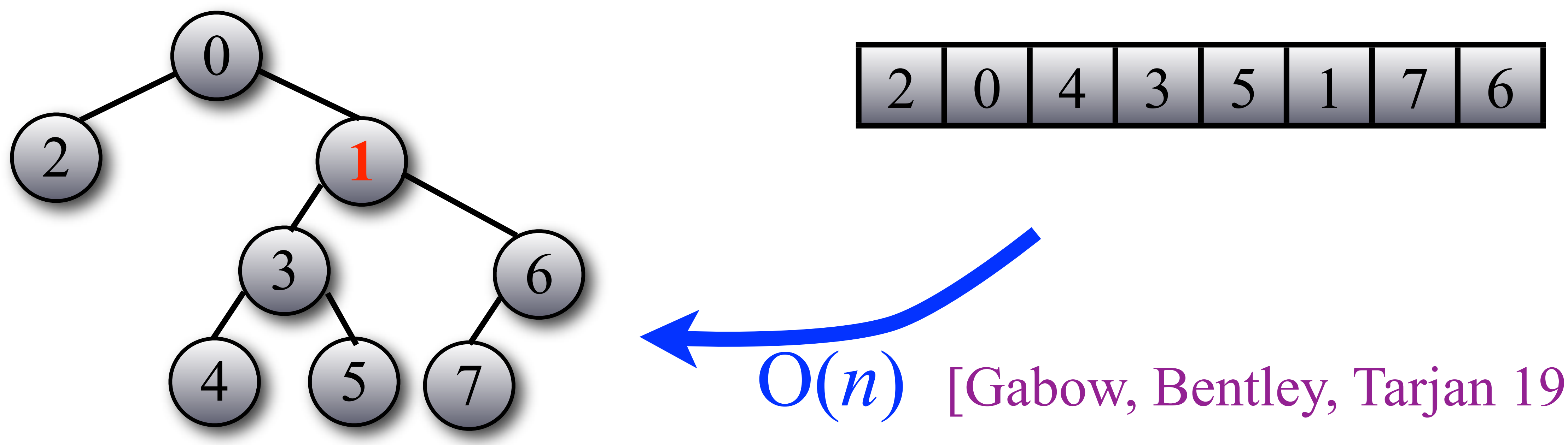
$O(n)$ [Gabow, Bentley, Tarjan 1984]

RMQ
& Cartesian Trees



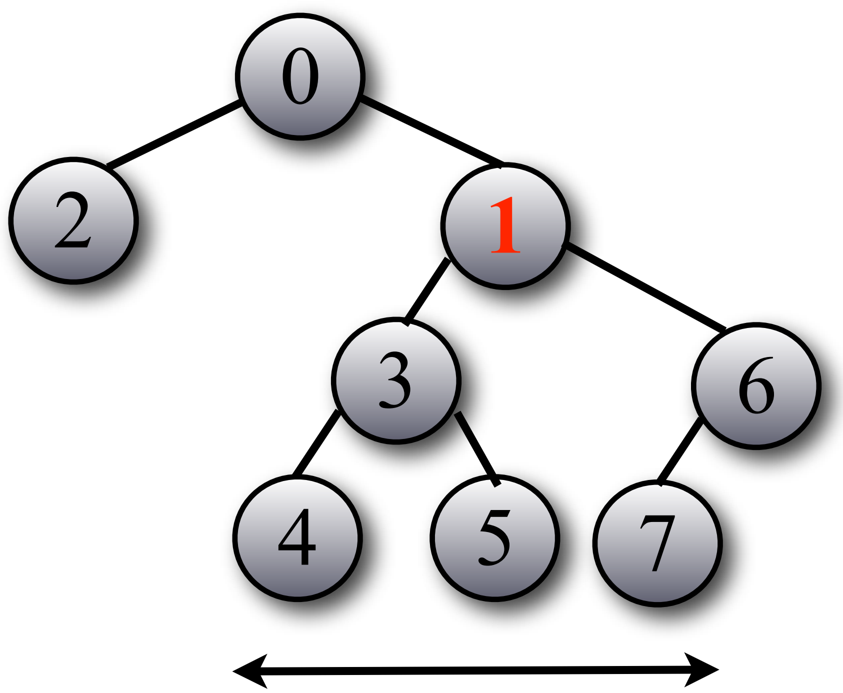
O(n) [Gabow, Bentley, Tarjan 1984]

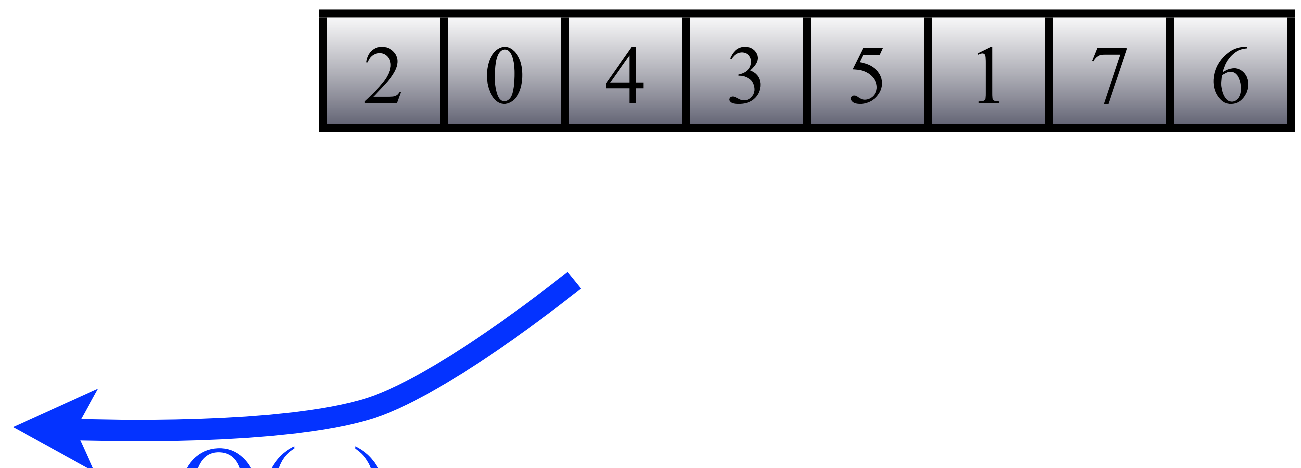
RMQ
& Cartesian Trees



$O(n)$ [Gabow, Bentley, Tarjan 1984]

RMQ
& Cartesian Trees

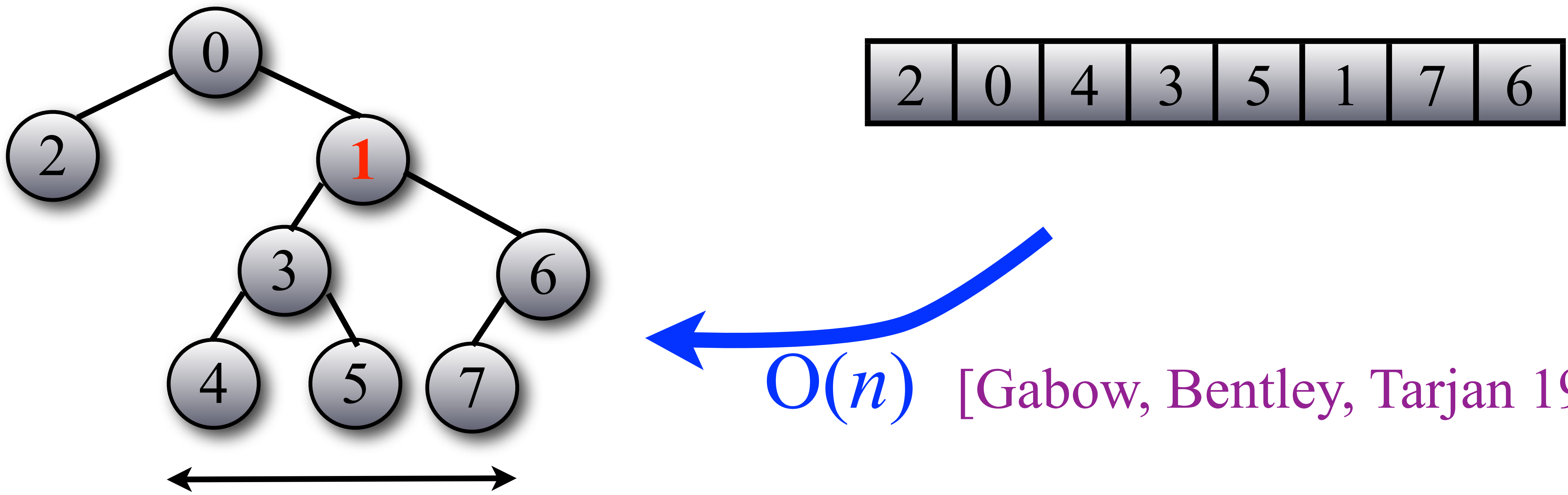




O(n) [Gabow, Bentley, Tarjan 1984]

LCA: O(n) prep. O(1) query [Harel, Tarjan 1984]

RMQ
& Cartesian Trees



$O(n)$ [Gabow, Bentley, Tarjan 1984]

LCA: $O(n)$ prep. $O(1)$ query [Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

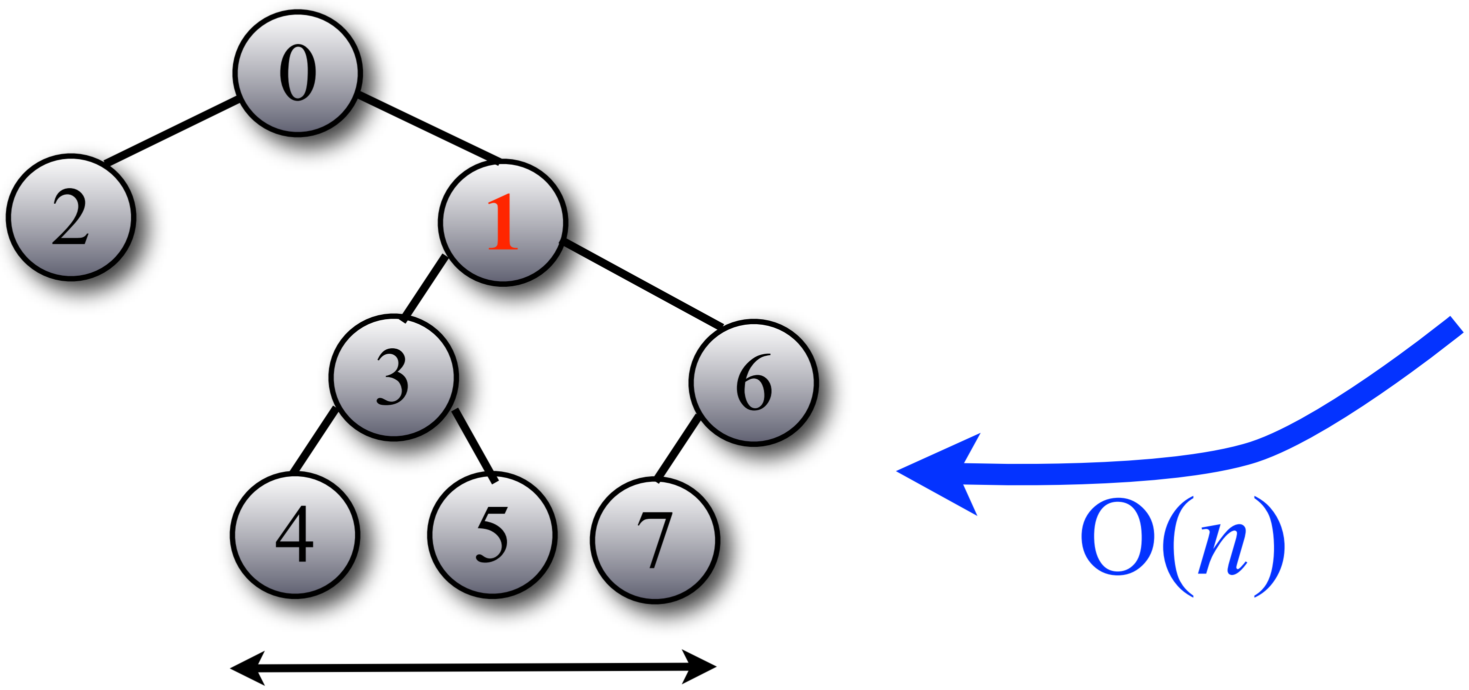
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ
& Cartesian Trees



LCA: O(*n*) prep. O(1) query

[Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

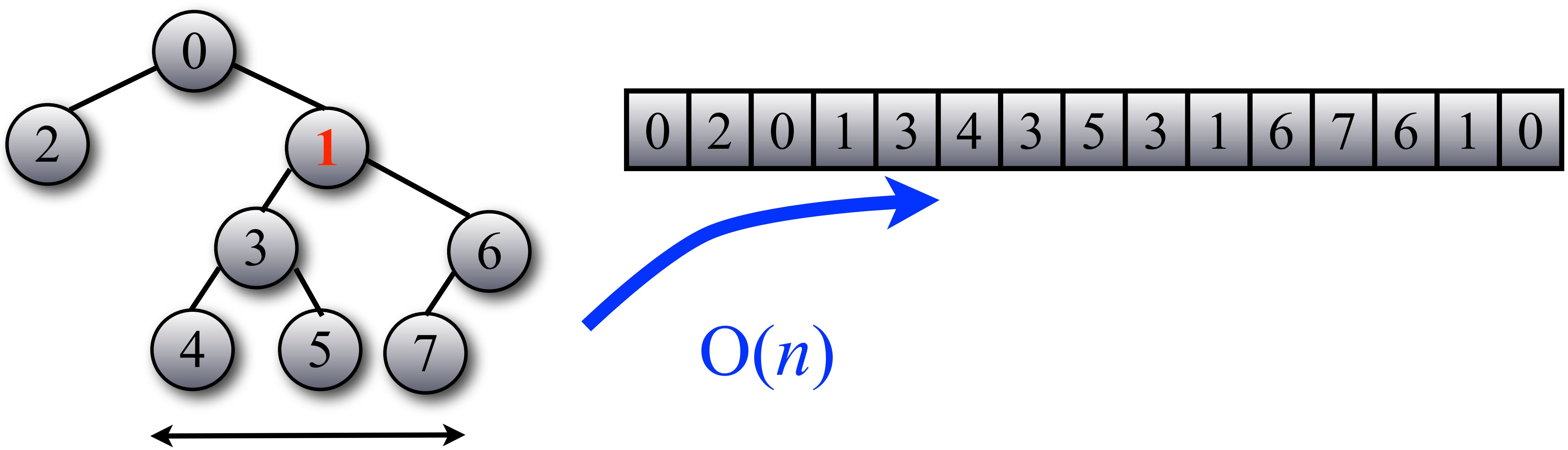
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ
& Cartesian Trees



LCA: O(*n*) prep. O(1) query [Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

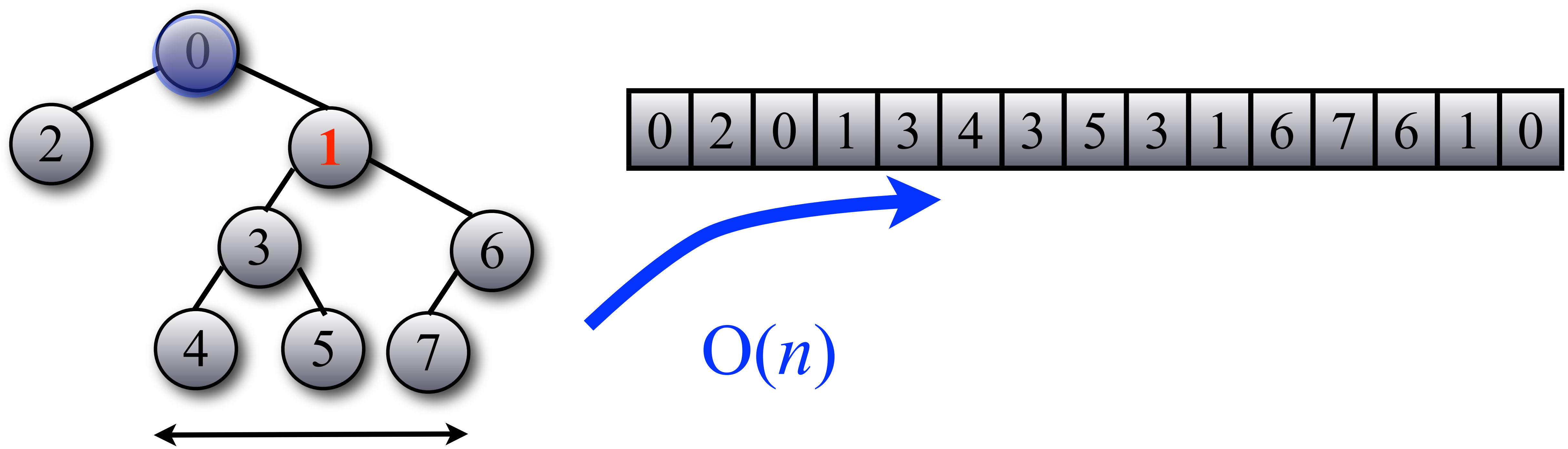
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ
& Cartesian Trees



LCA: O(*n*) prep. O(1) query [Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

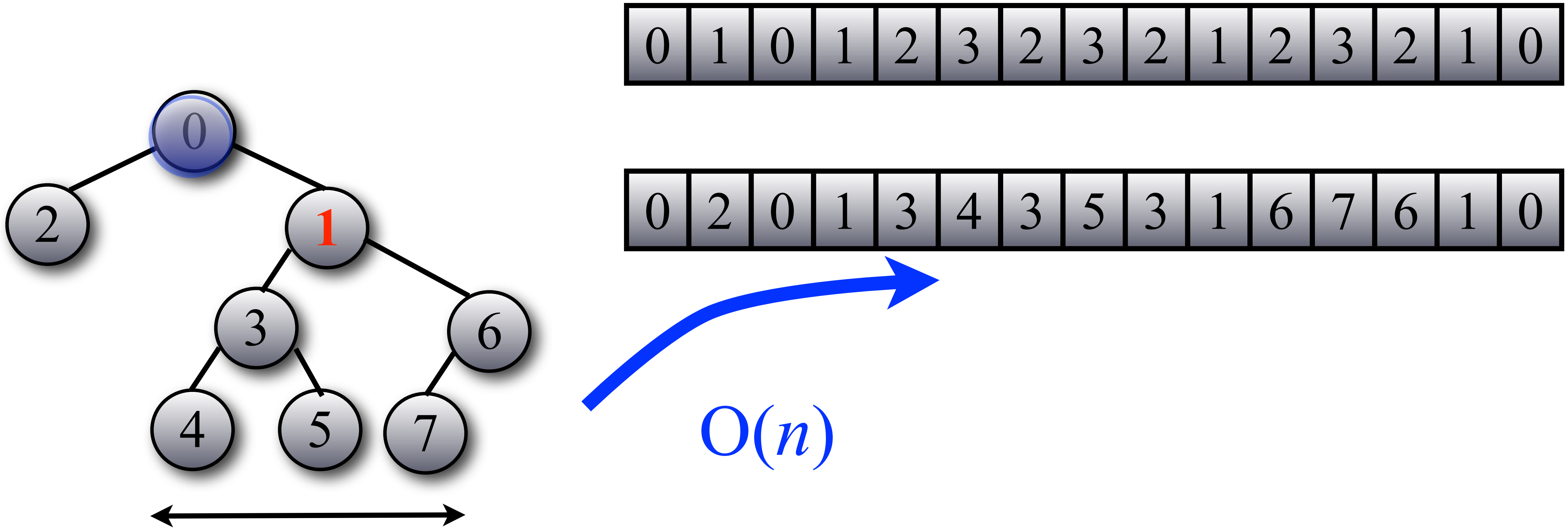
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ
& Cartesian Trees



LCA: O(n) prep. O(1) query

[Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

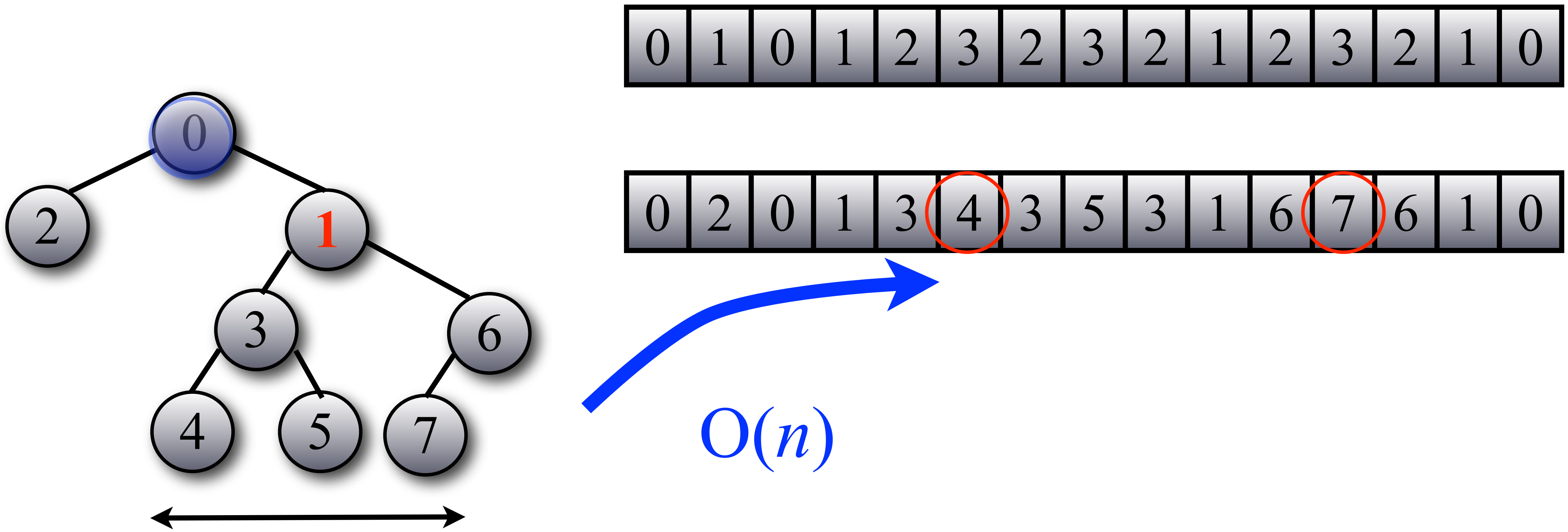
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ
& Cartesian Trees



LCA: O(n) prep. O(1) query

[Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

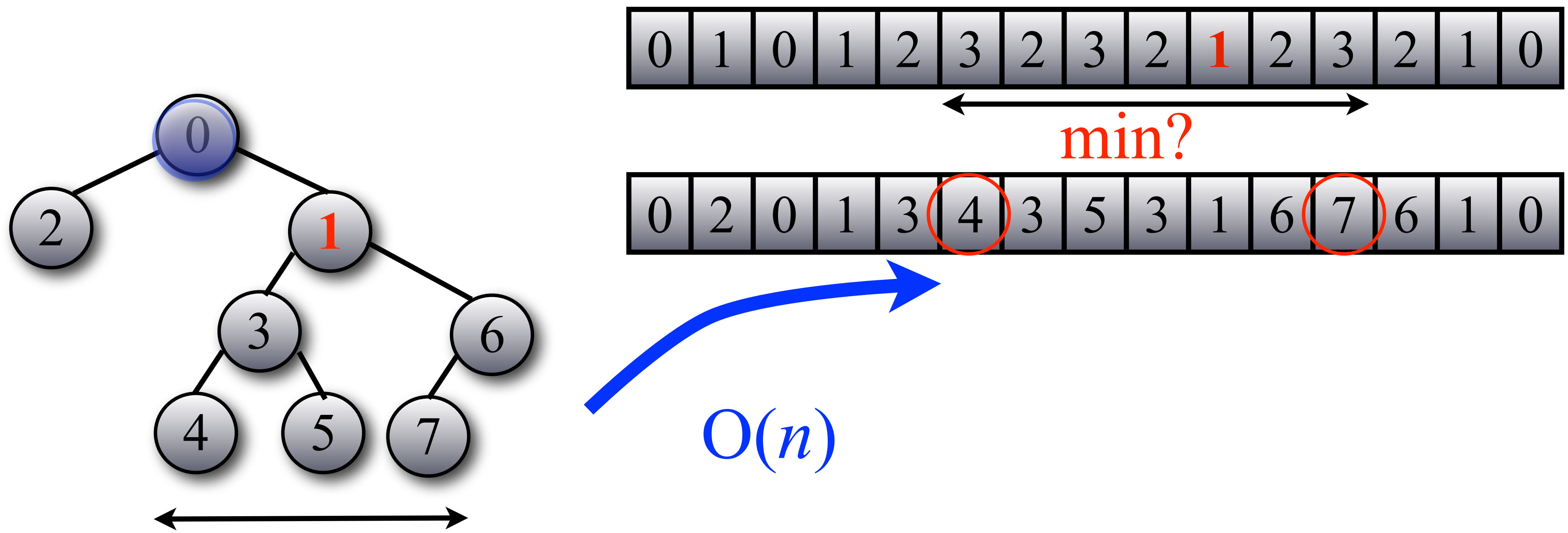
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ
& Cartesian Trees



LCA: O(n) prep. O(1) query

[Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

RMQ

- Warmup: $O(n \log n)$ prep. $O(1)$ query:

2	0	4	3	5	4	7	0	5	6	1	4	8	6	7	3	4	2	5	4

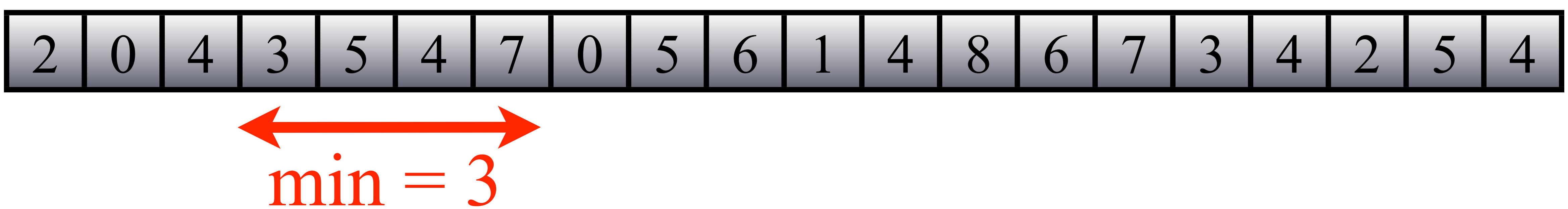
RMQ

- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval I s.t $|I|$ is a power of two

2	0	4	3	5	4	7	0	5	6	1	4	8	6	7	3	4	2	5	4

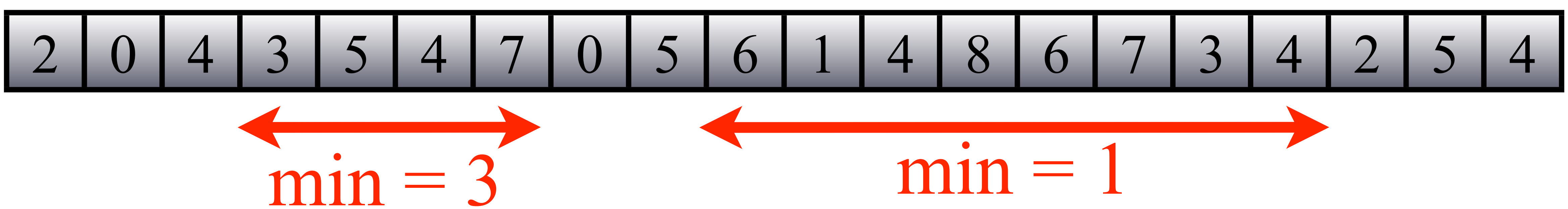
RMQ

- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval I s.t $|I|$ is a power of two



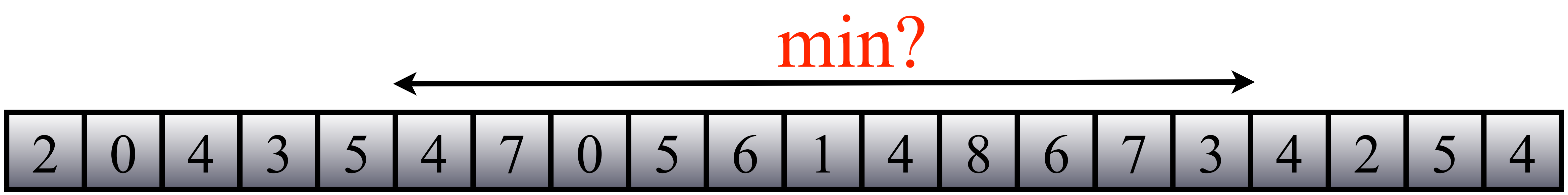
RMQ

- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval I s.t $|I|$ is a power of two



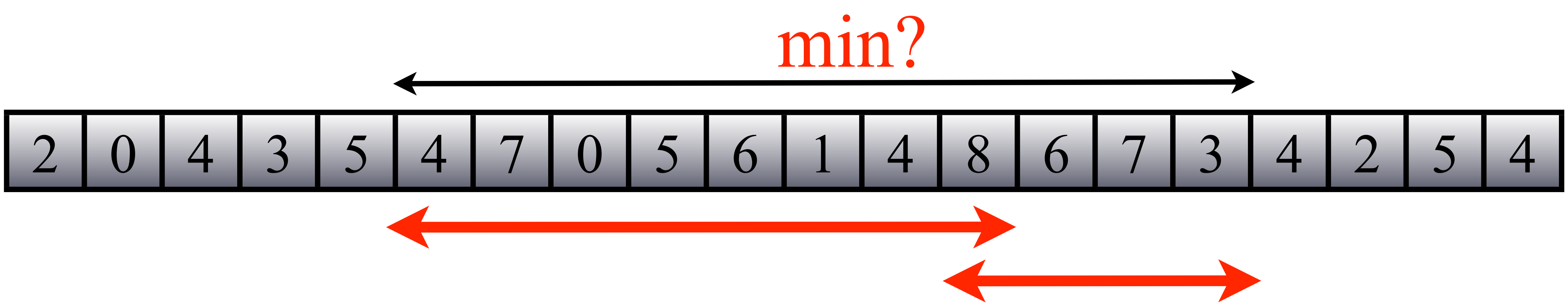
RMQ

- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval I s.t $|I|$ is a power of two
  - Query is composed of two overlapping intervals



RMQ

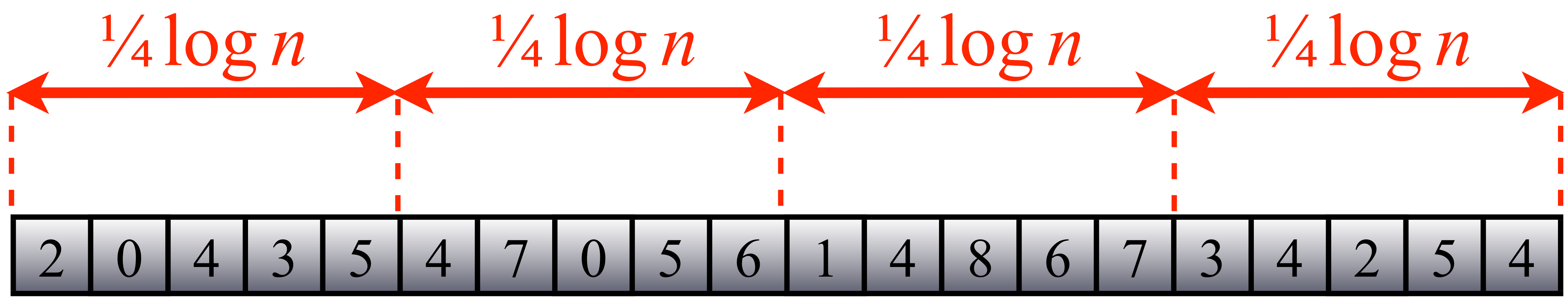
- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval I s.t $|I|$ is a power of two
  - Query is composed of two overlapping intervals



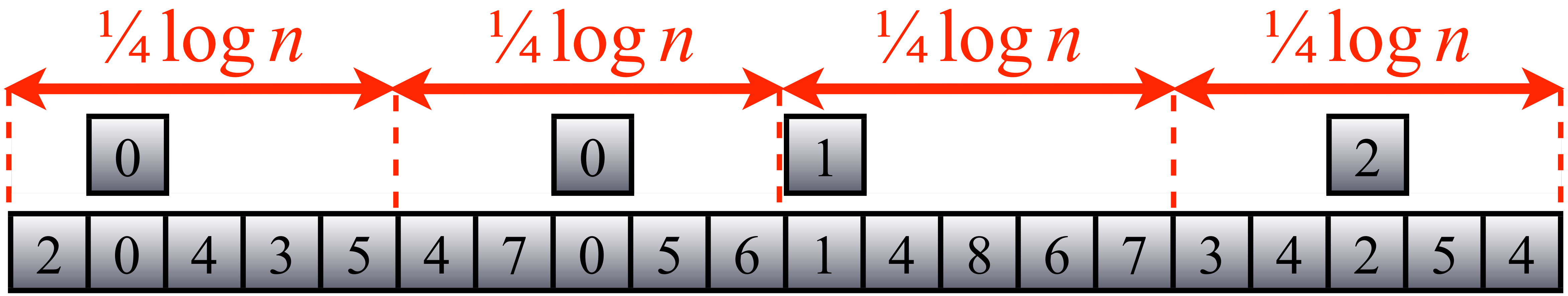
RMQ

2	0	4	3	5	4	7	0	5	6	1	4	8	6	7	3	4	2	5	4

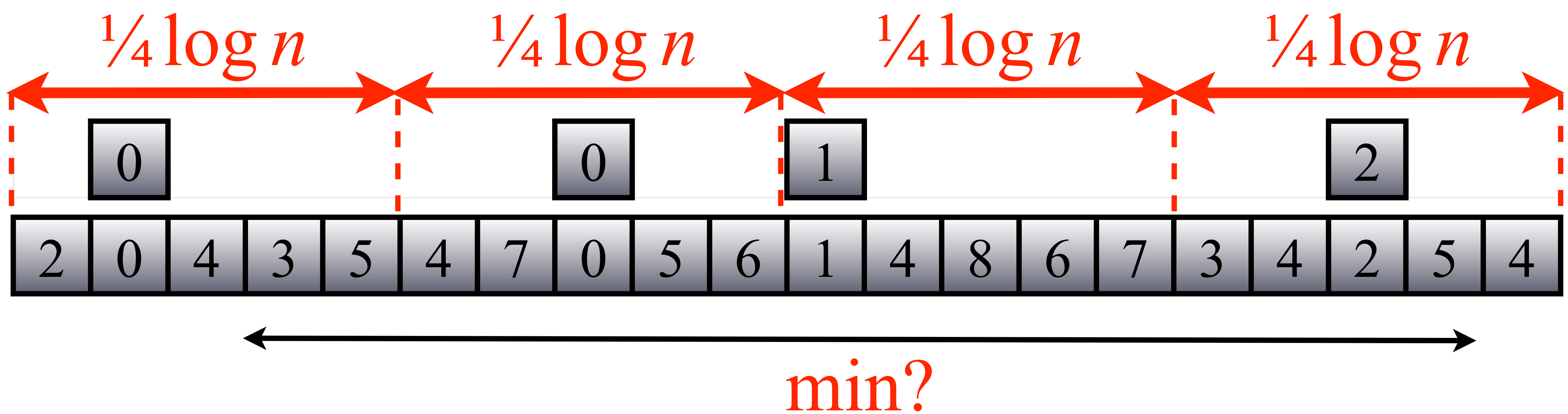
RMQ



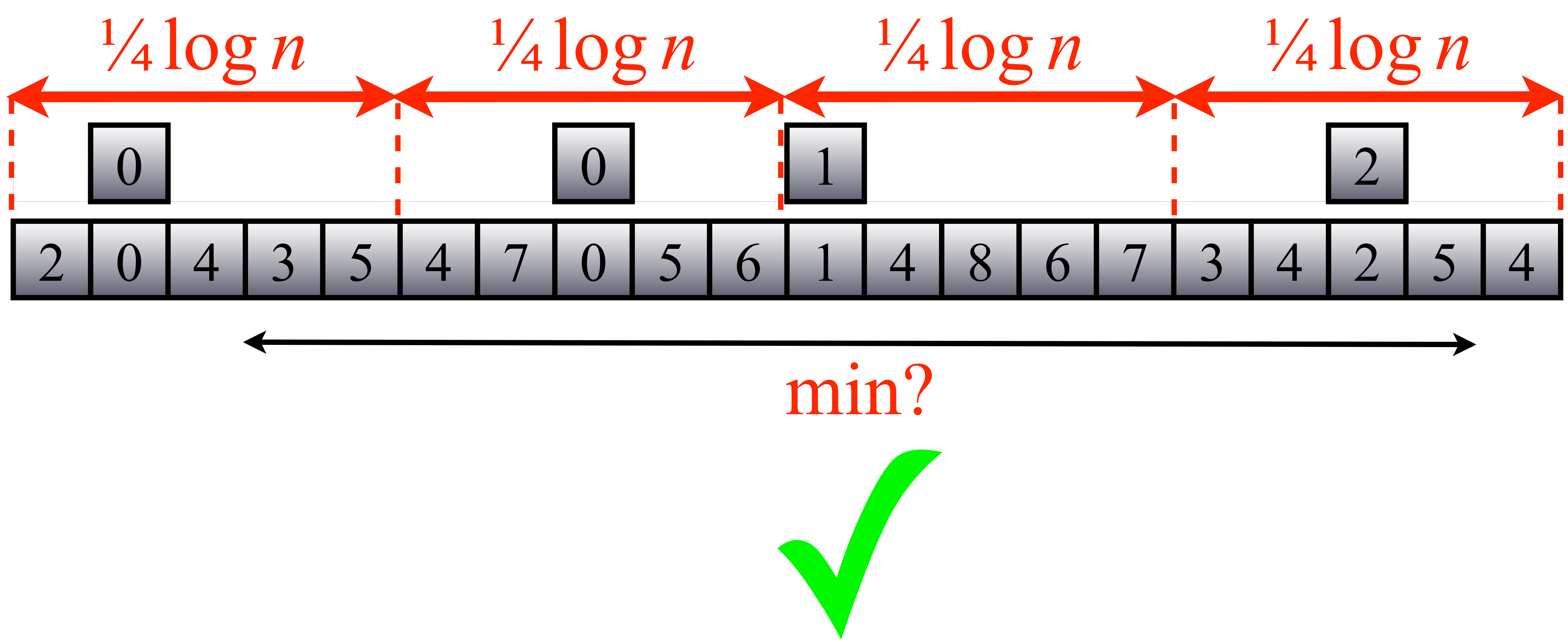
RMQ



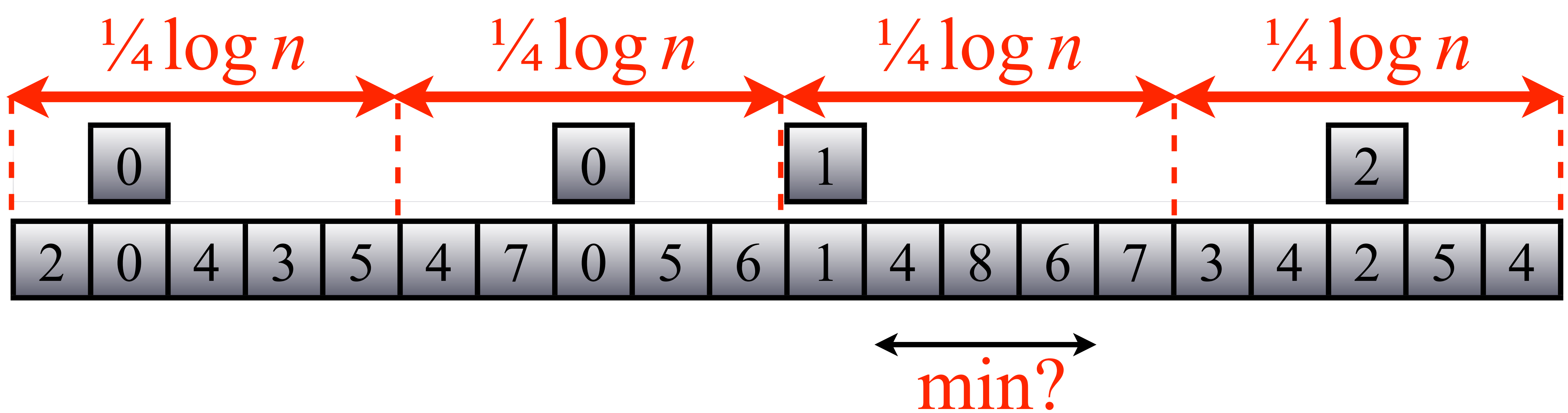
RMQ



RMQ

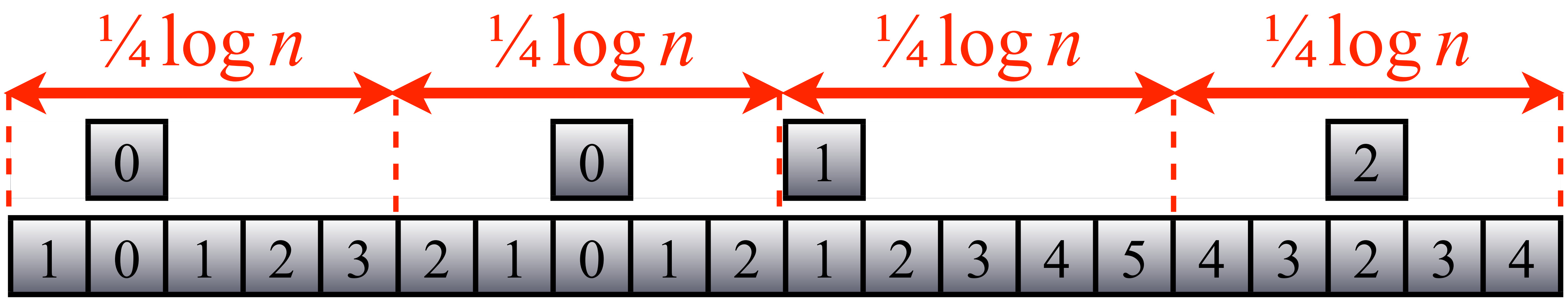


RMQ



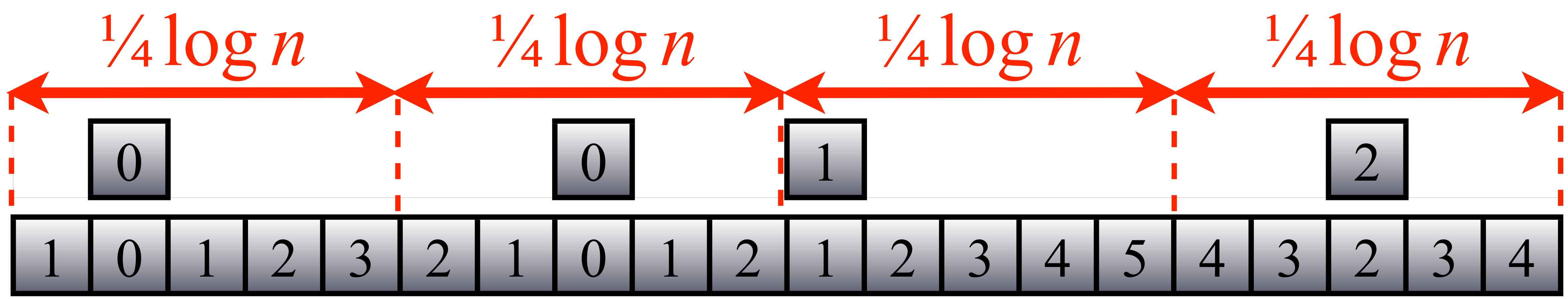
RMQ ±1

- RMQ ⇒ LCA ⇒ RMQ ±1



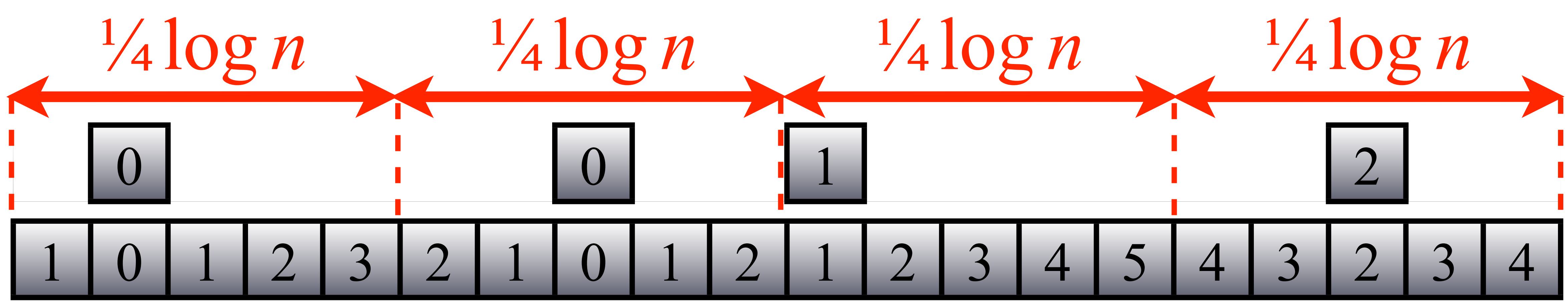
RMQ ±1

- RMQ ⟹ LCA ⟹ RMQ ±1
- # *different* Blocks = # different ±1 vectors = $2^{\frac{1}{4}\log n} = n^{\frac{1}{4}}$
  - Lookup table



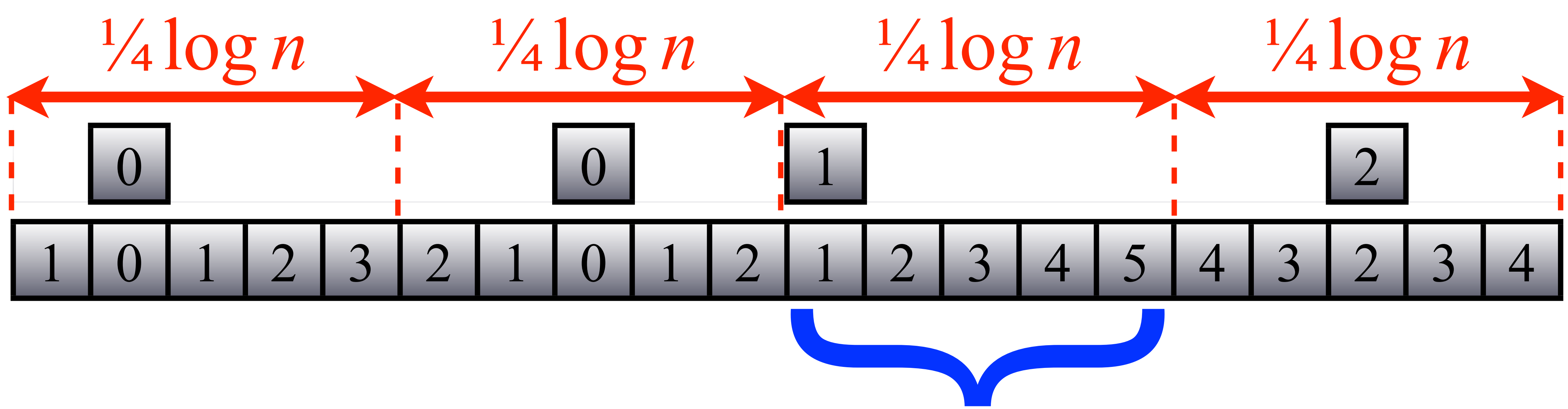
Recursive Solution

- Use the "Warmup" solution on each block



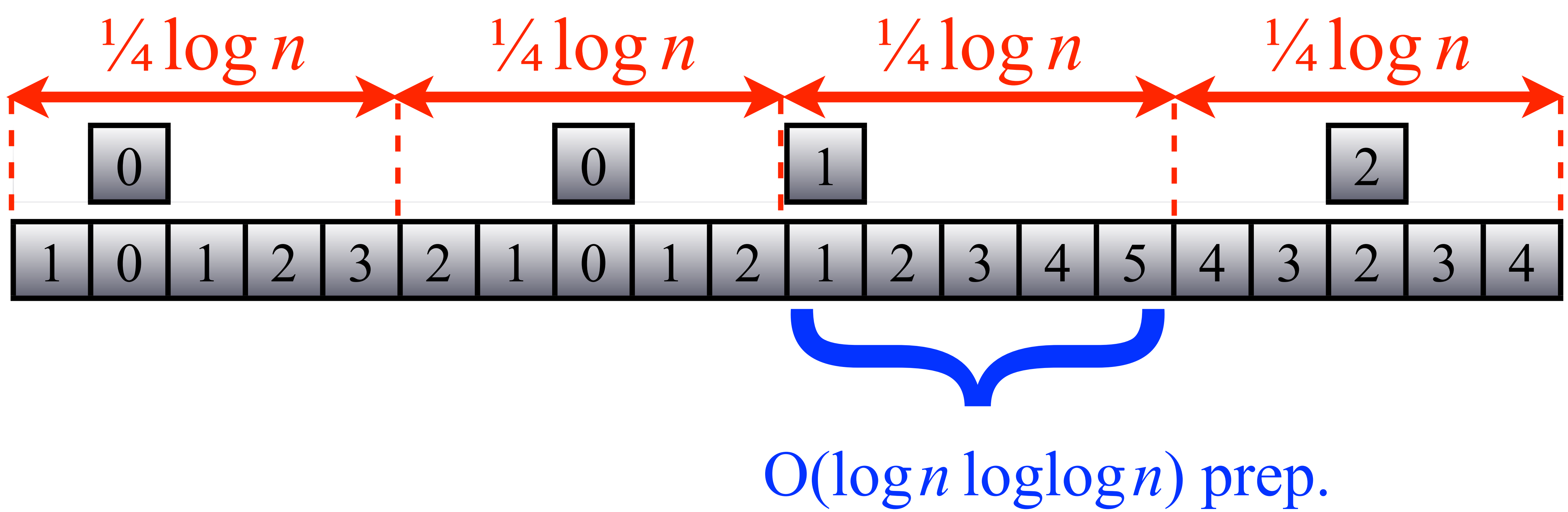
Recursive Solution

- Use the "Warmup" solution on each block



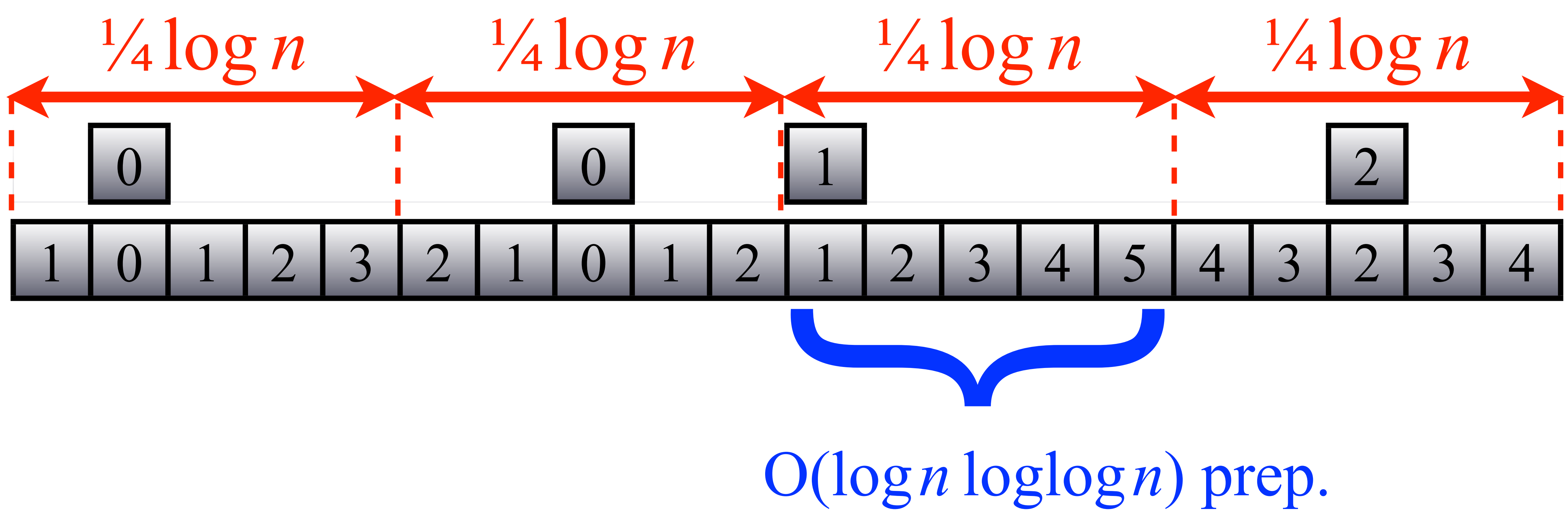
Recursive Solution

- Use the "Warmup" solution on each block



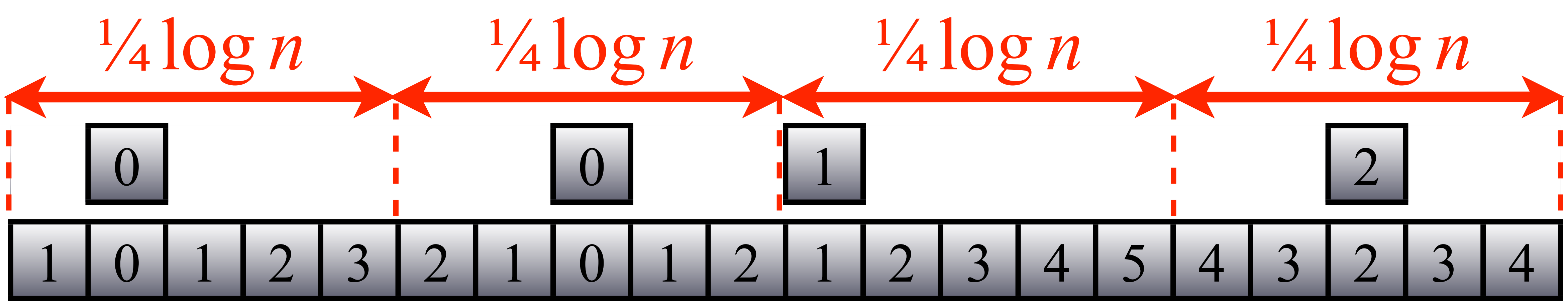
Recursive Solution

- Use the "Warmup" solution on each block
  - O(n loglog n) prep. O(1) query



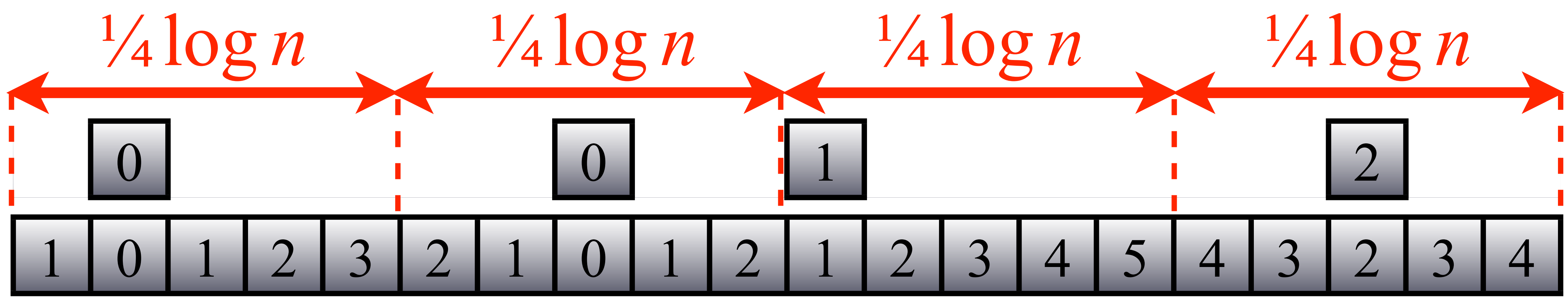
Recursive Solution

- Use the "Warmup" solution on each block
  - $O(n \log\log n)$ prep. $O(1)$ query
  - $O(n \log\log\log n)$ prep. $O(1)$ query



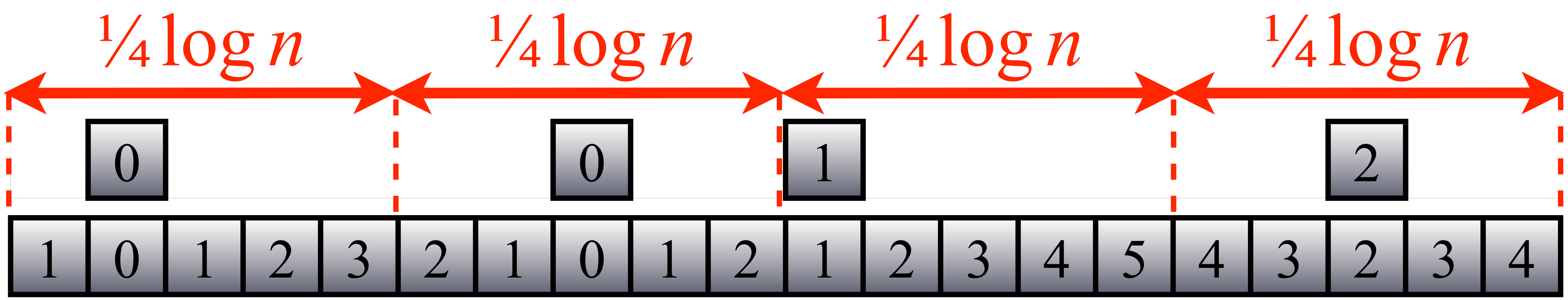
Recursive Solution

- Use the "Warmup" solution on each block
  - $O(n \log\log n)$ prep. $O(1)$ query
  - $O(n \log\log\log n)$ prep. $O(1)$ query
  - $O(n\,\alpha_k(n))$ prep. $O(k)$ query [Alon&Schieber 1987, Chazelle&Rosenberg 1989]



Recursive Solution

- Use the "Warmup" solution on each block
  - O(n loglog n) prep. O(1) query
  - O(n logloglog n) prep. O(1) query
  - O($n\,\alpha_k(n)$) prep. O(k) query [Alon&Schieber 1987, Chazelle&Rosenberg 1989]



- Why?
    - ~~MIN~~ → any semiring operation
    - RMQ generalizations
    - Parallel Computing

Parallel RMQ

[Berkman and Vishkin 1993]

1	0	1	2	3	2	1	0	1	2	1	2	3	4	5	4	3	2	3	4

Parallel RMQ
[Berkman and Vishkin 1993]

- Min of n elements in O(1) time using n^2 processors [Valiant 1975]

1	0	1	2	3	2	1	0	1	2	1	2	3	4	5	4	3	2	3	4

Parallel RMQ

[Berkman and Vishkin 1993]

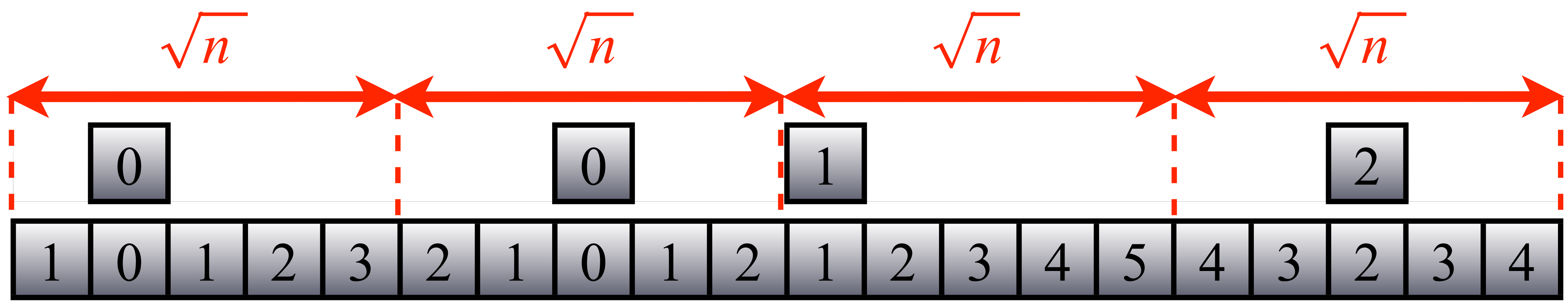
- Min of n elements in O(1) time using n^2 processors [Valiant 1975]
    - O(1) RMQ using n^4 processors

1	0	1	2	3	2	1	0	1	2	1	2	3	4	5	4	3	2	3	4

Parallel RMQ

[Berkman and Vishkin 1993]

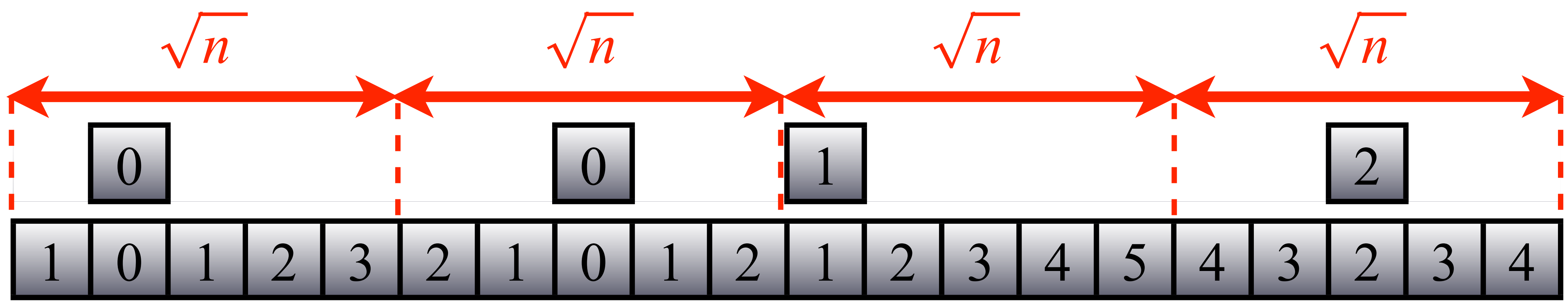
- Min of n elements in O(1) time using n^2 processors [Valiant 1975]
    - O(1) RMQ using n^4 processors
    - O(1) RMQ using $n^{2.5}$ processors



Parallel RMQ

[Berkman and Vishkin 1993]

- Min of n elements in O(1) time using n^2 processors [Valiant 1975]
  - O(1) RMQ using n^4 processors
  - O(1) RMQ using $n^{2.5}$ processors
  - O(1/ε) RMQ using $n^{1+\varepsilon}$ processors



Parallel RMQ ±1

[Berkman and Vishkin 1993]

1	0	1	2	3	2	1	0	1	2	1	2	3	4	5	4	3	2	3	4

Parallel RMQ ± 1

[Berkman and Vishkin 1993]

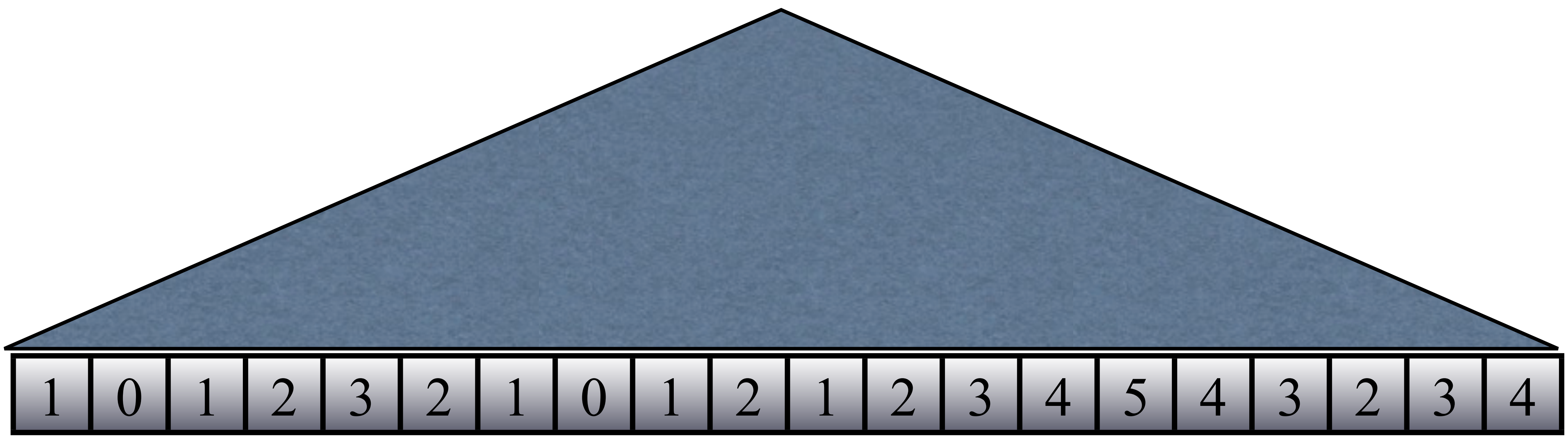
- Min of n integers each between 1 and n in O(1) time using n processors [Fich, Ragde and Wigderson 1984]

1	0	1	2	3	2	1	0	1	2	1	2	3	4	5	4	3	2	3	4

Parallel RMQ ±1

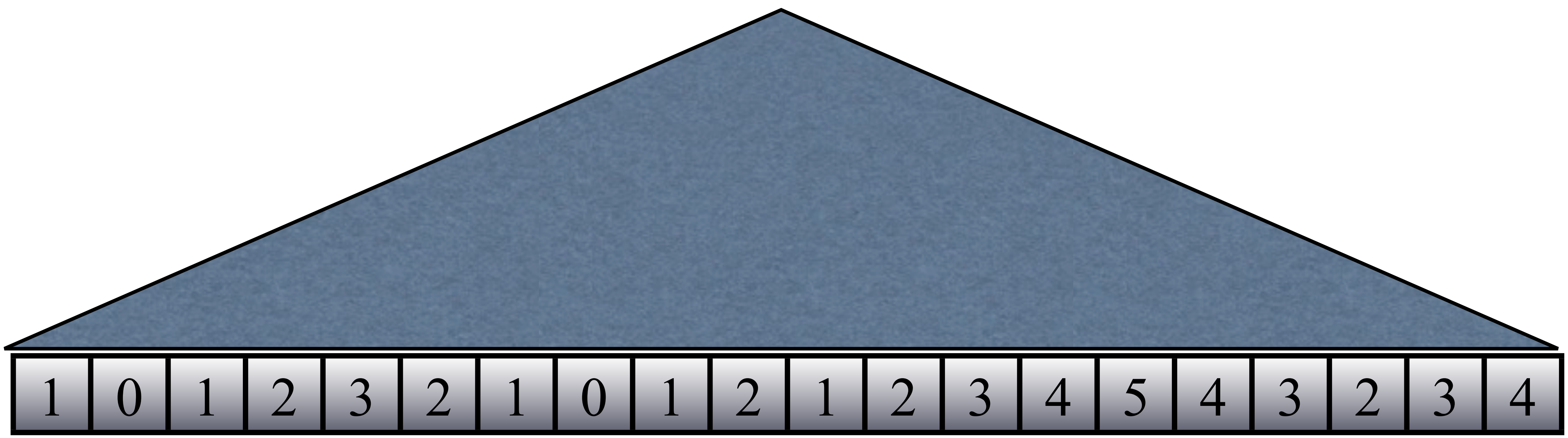
[Berkman and Vishkin 1993]

- Min of n integers <u>each between 1 and n</u> in O(1) time using n processors [Fich, Ragde and Wigderson 1984]



Parallel RMQ ±1
[Berkman and Vishkin 1993]

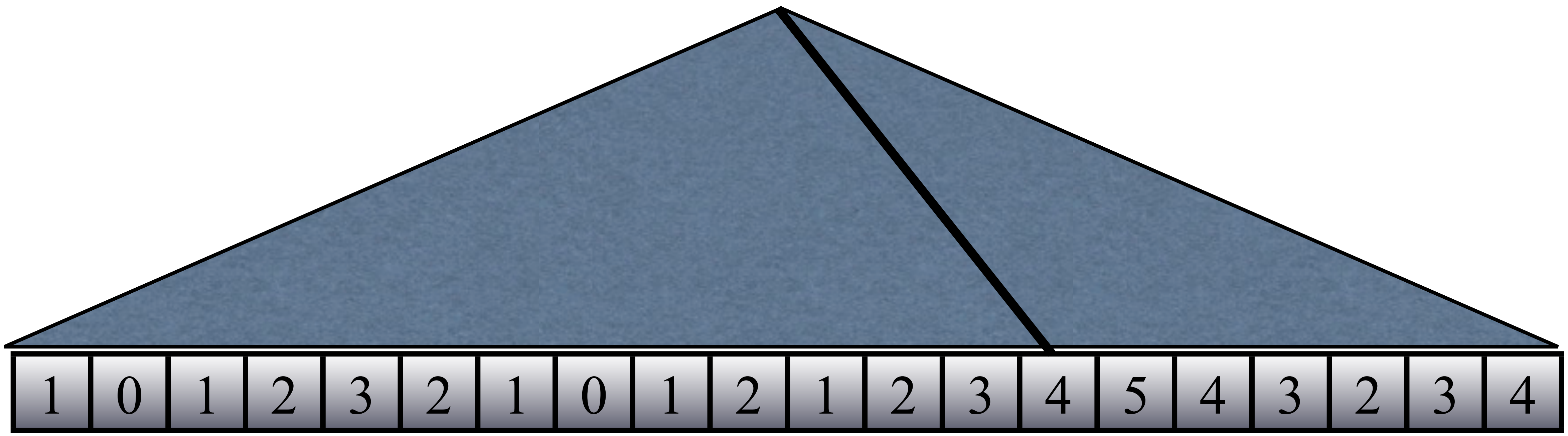
- Min of n integers <u>each between 1 and n</u> in O(1) time using n processors [Fich, Ragde and Wigderson 1984]



Parallel RMQ ±1

[Berkman and Vishkin 1993]

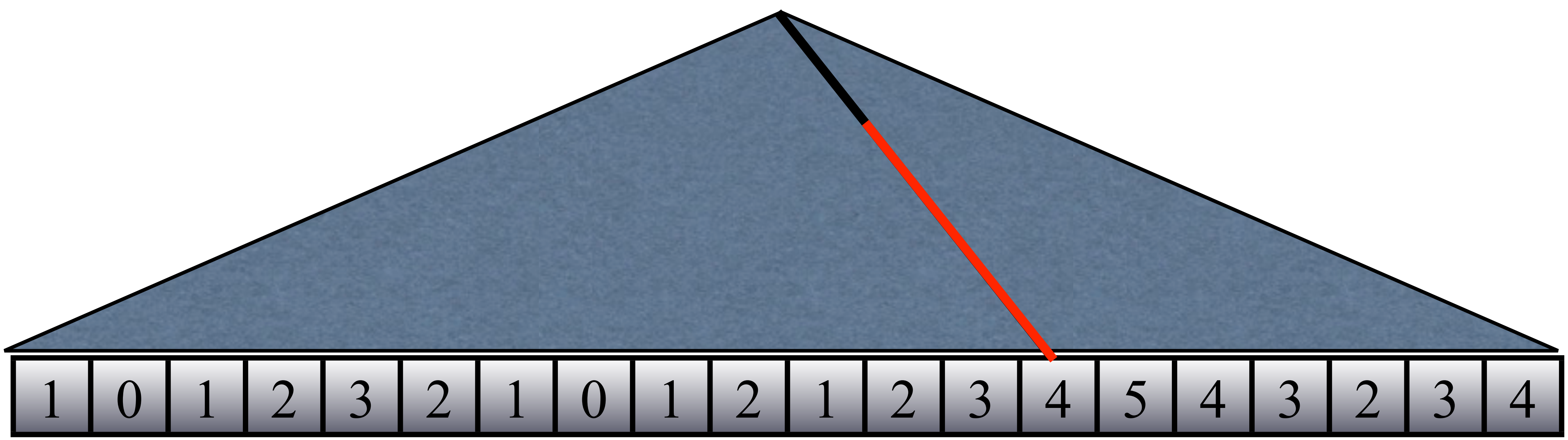
- Min of n integers <u>each between 1 and n</u> in O(1) time using n processors [Fich, Ragde and Wigderson 1984]



Parallel RMQ ±1

[Berkman and Vishkin 1993]

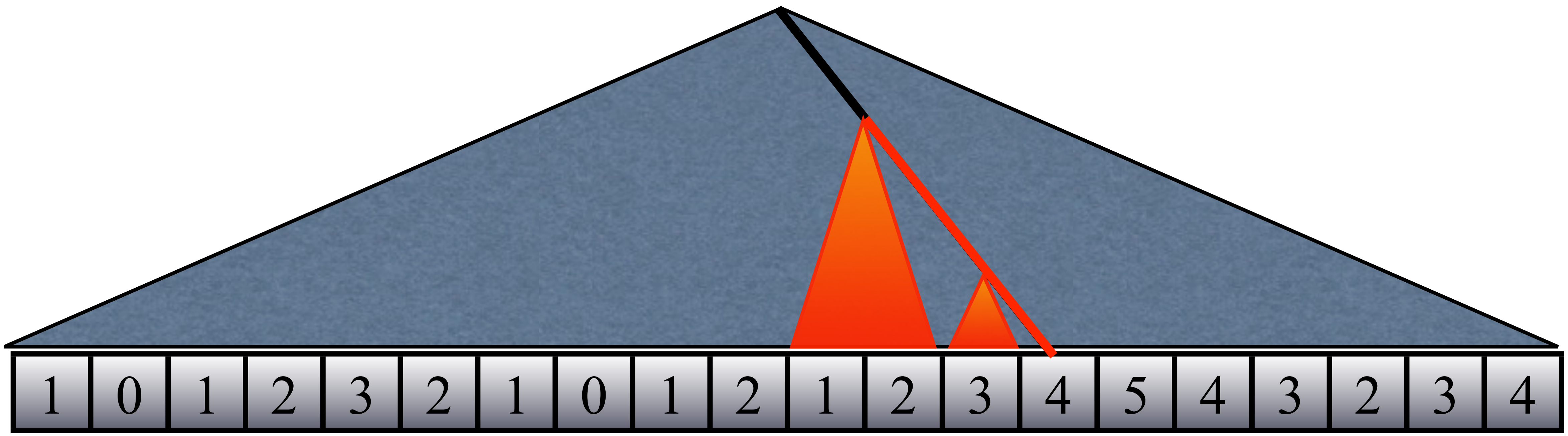
- Min of n integers <u>each between 1 and n</u> in O(1) time using n processors [Fich, Ragde and Wigderson 1984]



Parallel RMQ ± 1

[Berkman and Vishkin 1993]

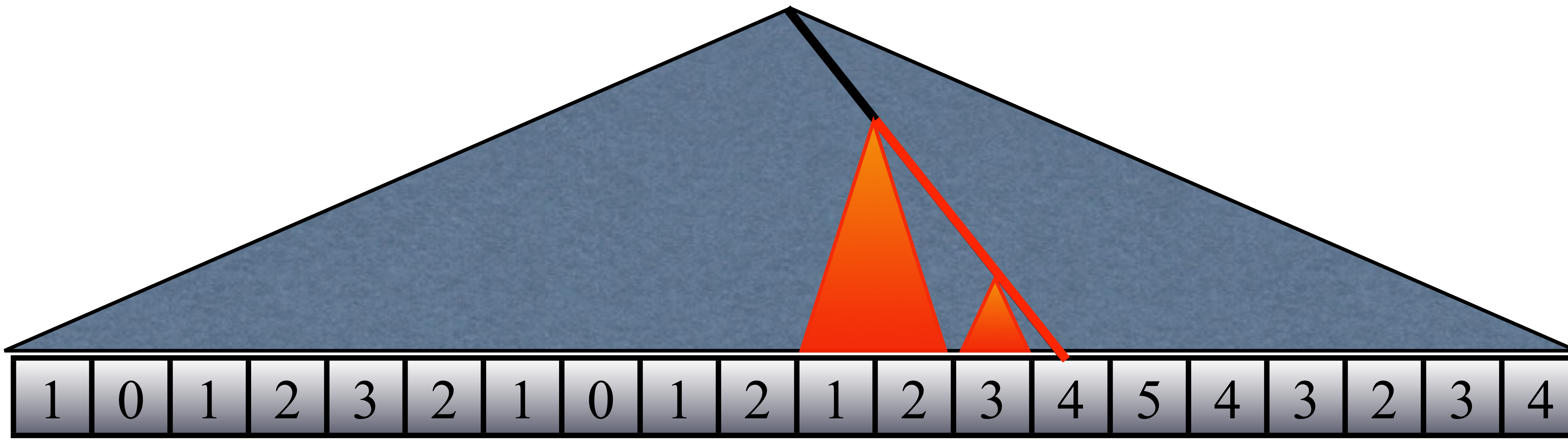
- Min of n integers <u>each between 1 and n</u> in O(1) time using n processors [Fich, Ragde and Wigderson 1984]



Parallel RMQ ±1

[Berkman and Vishkin 1993]

- Min of n integers <u>each between 1 and n</u> in O(1) time using n processors [Fich, Ragde and Wigderson 1984]

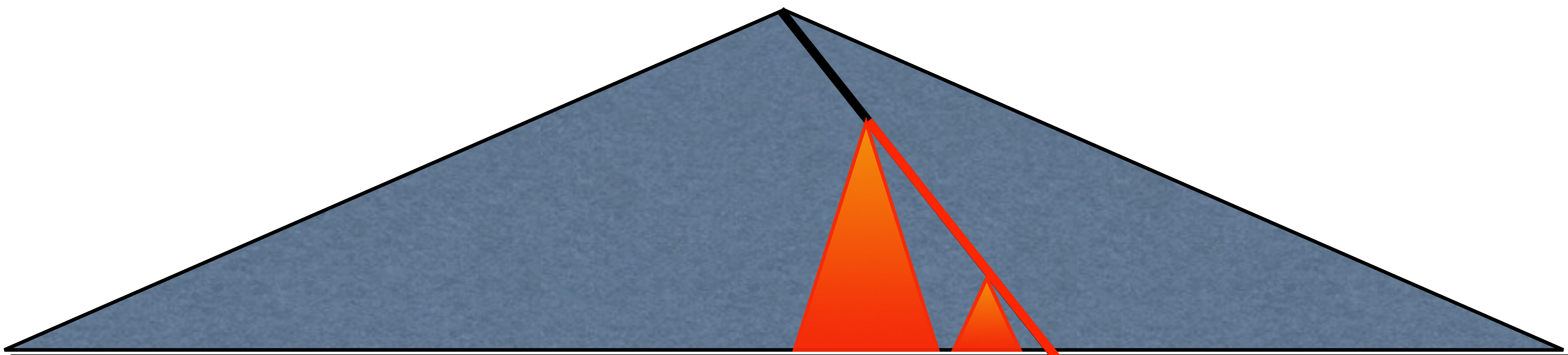


- $n \log^3 n$ processors, O(1) time, O(1) query

Parallel RMQ ±1

[Berkman and Vishkin 1993]

- Min of n integers each between 1 and n in O(1) time using n processors [Fich, Ragde and Wigderson 1984]



1	0	1	2	3	2	1	0	1	2	1	2	3	4	5	4	3	2	3	4

- $n \log^3 n$ processors, O(1) time, O(1) query
- $n\, \alpha_k(n)$ processors, O(k) time, O(k) query

Problems with RMQ ±1

- RMQ → LCA → RMQ ±1

2	0	4	3	5	4	7	0	5	6	1	4	8	6	7	3	4	2	5	4

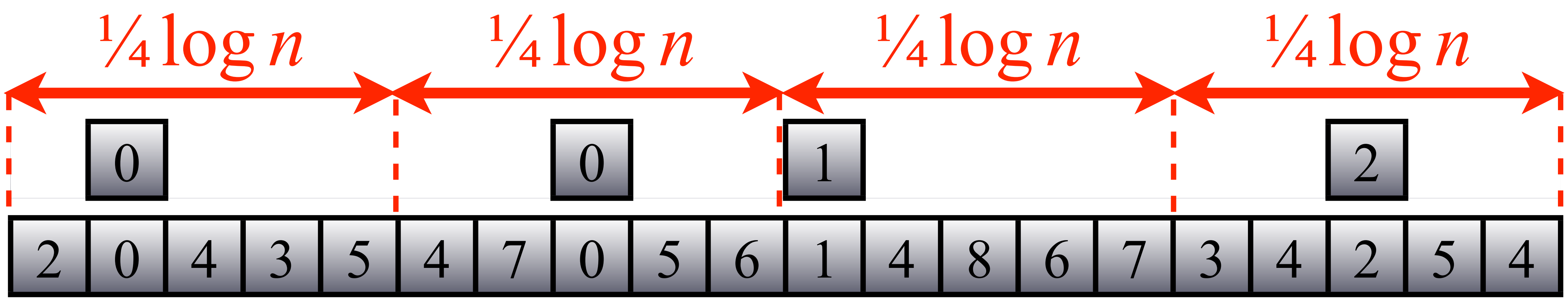
Problems with RMQ ±1

- RMQ → LCA → RMQ ±1
  - inefficient in parallel
  - inefficient in terms of cache-misses (can't be done via scans)

2	0	4	3	5	4	7	0	5	6	1	4	8	6	7	3	4	2	5	4

Problems with RMQ ±1

- RMQ → LCA → RMQ ±1
  - inefficient in parallel
  - inefficient in terms of cache-misses (can't be done via scans)

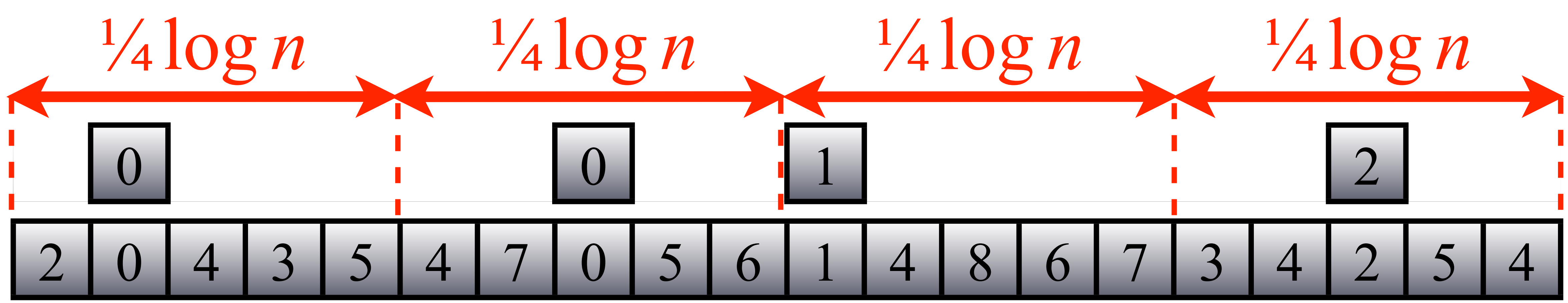


- # *different* Blocks = # different Cartesian trees = $4^{\frac{1}{4}\log n} = \sqrt{n}$

[Fischer, Heun 2006]

Problems with RMQ ±1

- RMQ → LCA → RMQ ±1
  - inefficient in parallel
  - inefficient in terms of cache-misses (can't be done via scans)



- # *different* Blocks = # different Cartesian trees = $4^{\frac{1}{4}\log n} = \sqrt{n}$

[Fischer, Heun 2006]

- Lookup table: <u>index</u>, <u>construct</u>

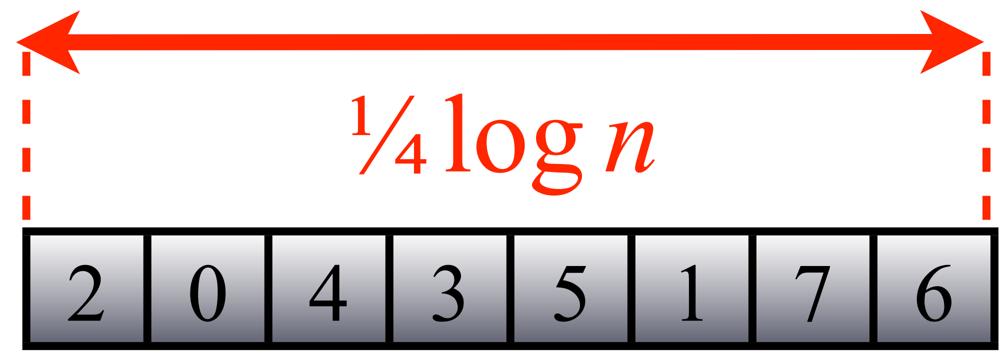
Cache-Oblivious RMQ

[Demaine, Landau and W. 2009]

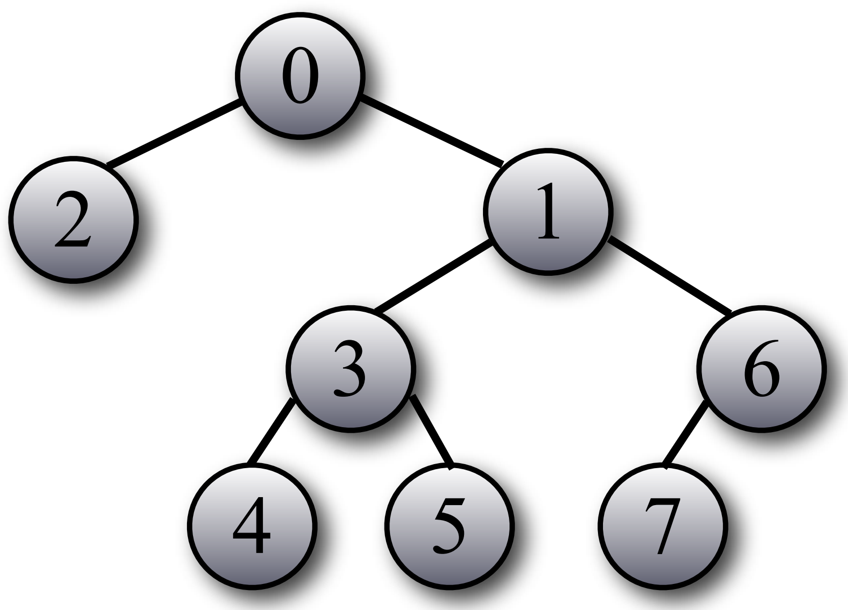
- An optimal RMQ solution that only makes sequential scans

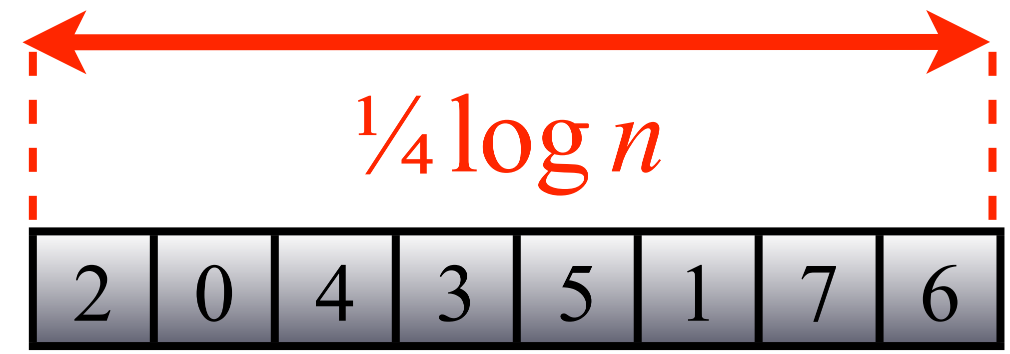
$O(n)$ prep. $O(1)$ query (serial algorithm)

A Cache-Oblivious Cartesian Tree

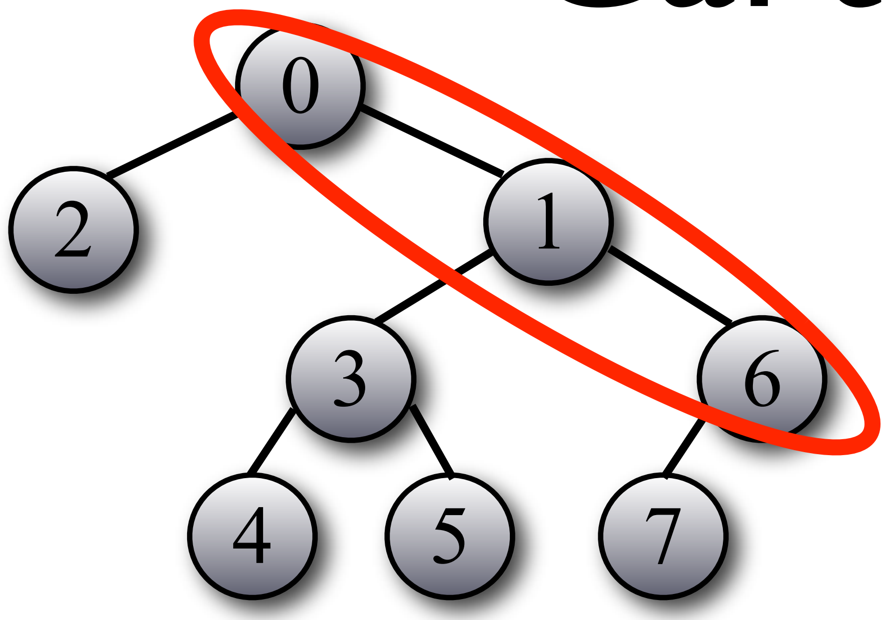


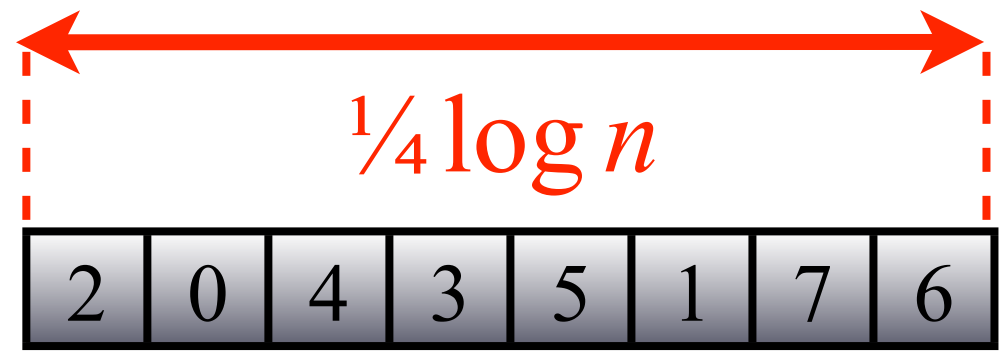
A Cache-Oblivious Cartesian Tree





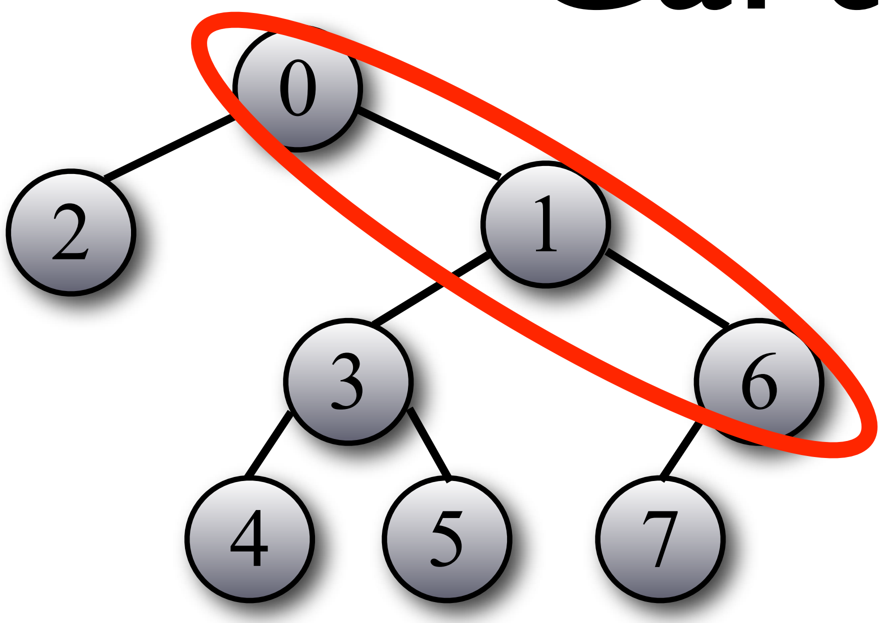
A Cache-Oblivious Cartesian Tree

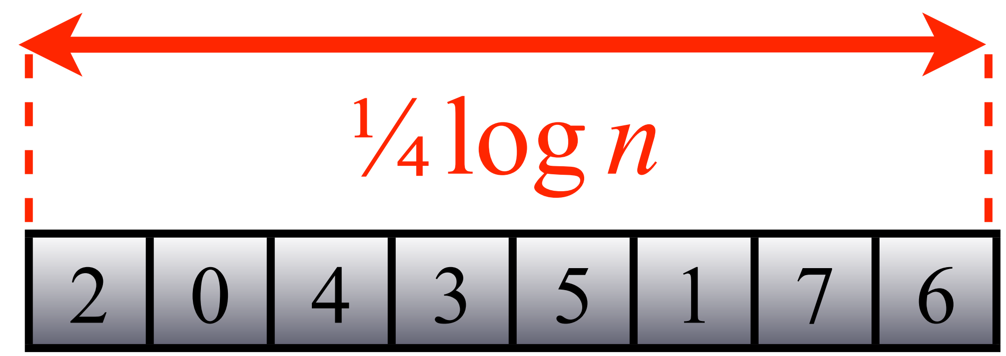




- cache-oblivious stack holds rightmost path

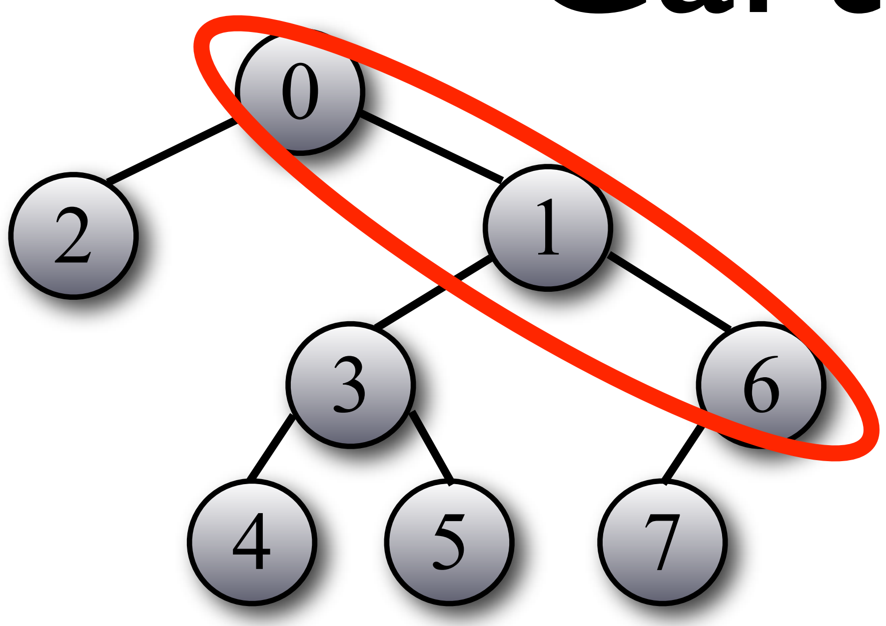
A Cache-Oblivious Cartesian Tree

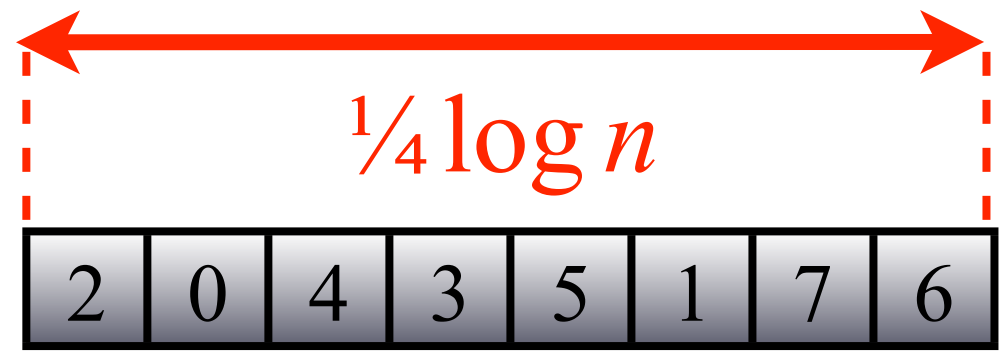




- cache-oblivious stack holds rightmost path
- when we climb (pop) i vertices, concatenate $0\underbrace{111\cdots11}_{i}$

A Cache-Oblivious Cartesian Tree

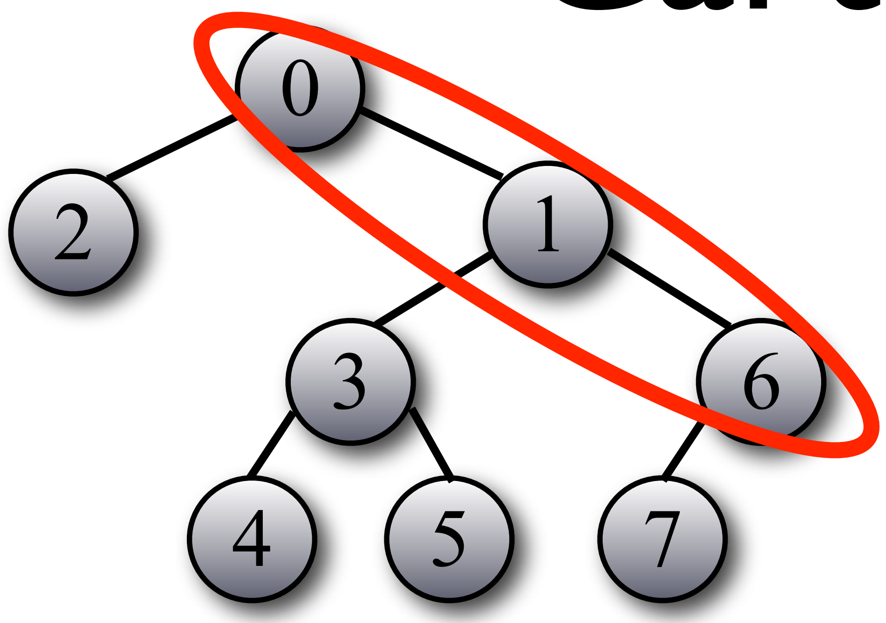


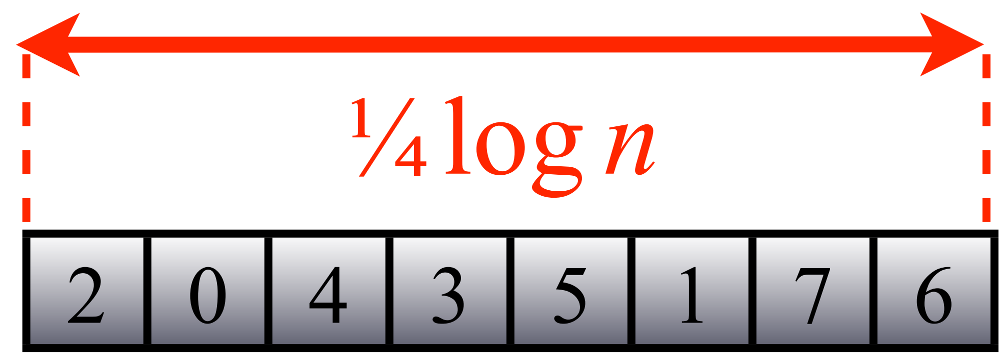


$$0\underbrace{11\cdots1}_{i_1}0\underbrace{11\cdots1}_{i_2}0\underbrace{11\cdots1}_{i_3}0\underbrace{11\cdots1}_{i_4}$$

- cache-oblivious stack holds rightmost path
- when we climb (pop) i vertices, concatenate $0\underbrace{111\cdots11}_{i}$

A Cache-Oblivious Cartesian Tree

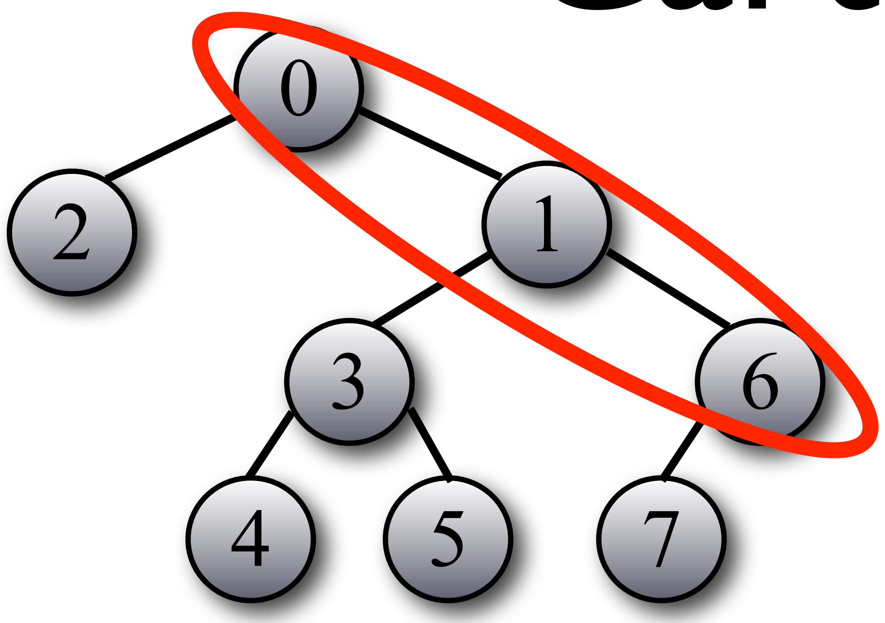


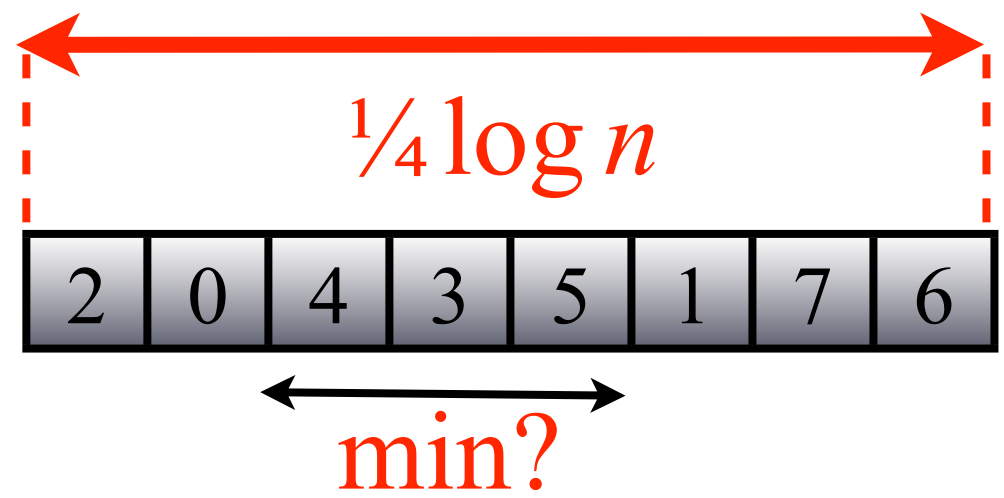


$$0\underbrace{11\cdots1}_{i_1}0\underbrace{11\cdots1}_{i_2}0\underbrace{11\cdots1}_{i_3}0\underbrace{11\cdots1}_{i_4} \in [\sqrt{n}]$$

- cache-oblivious stack holds rightmost path
- when we climb (pop) i vertices, concatenate $0\underbrace{111\cdots11}_{i}$

A Cache-Oblivious Cartesian Tree



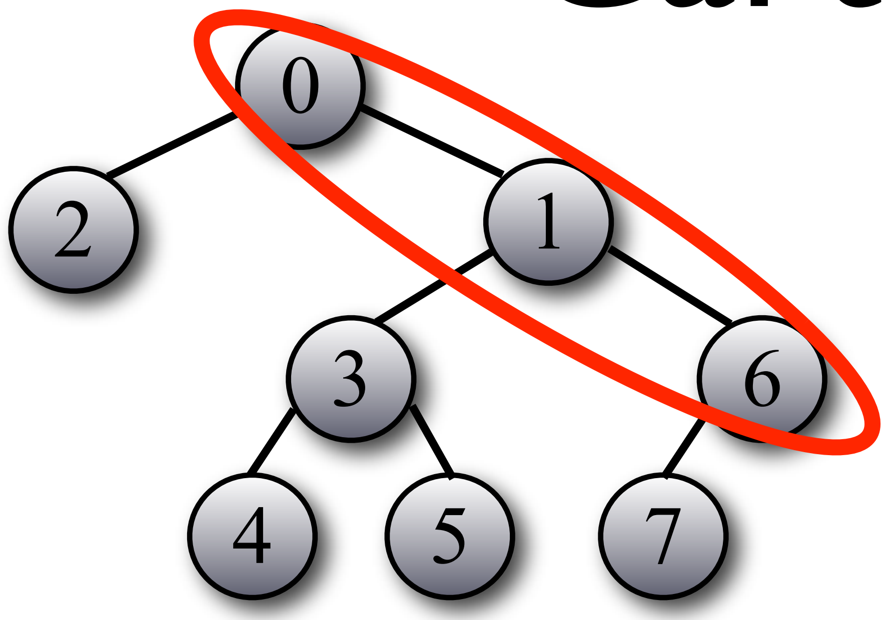


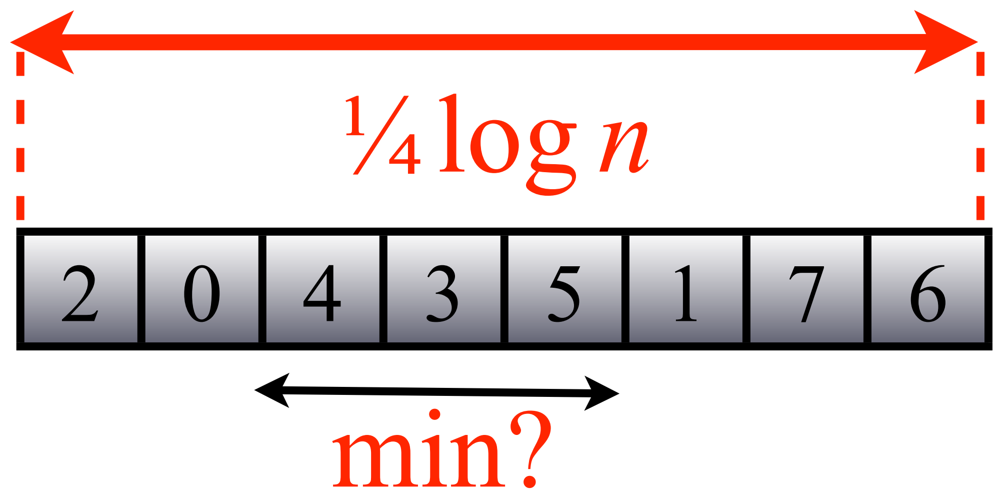
- ∀ binary string and ∀ query:

$$0\underbrace{11\cdots1}_{i_1}0\underbrace{11\cdots1}_{i_2}0\underbrace{11\cdots1}_{i_3}0\underbrace{11\cdots1}_{i_4} \in [\sqrt{n}\,]$$

- cache-oblivious stack holds rightmost path
- when we climb (pop) i vertices, concatenate $0\underbrace{111\cdots11}_{i}$

A Cache-Oblivious Cartesian Tree





- ∀ binary string and ∀ query:

$$0\underbrace{11\ldots1}_{i_1}0\underbrace{11\ldots1}_{i_2}0\underbrace{11\ldots1}_{i_3}0\underbrace{11\ldots1}_{i_4} \in [\sqrt{n}]$$

- cache-oblivious stack holds rightmost path
- when we climb (pop) i vertices, concatenate $0\underbrace{111\cdots11}_{i}$

RMQ Generalization I: A Cartesian Tree of a Tree

[Demaine, Landau and W. 2009]

RMQ Generalization I:
A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph

RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

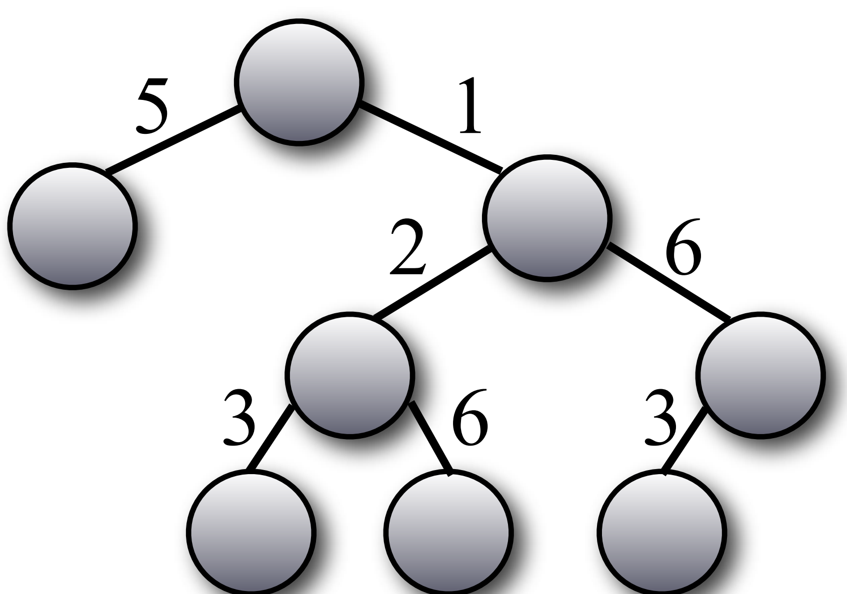
RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]

RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

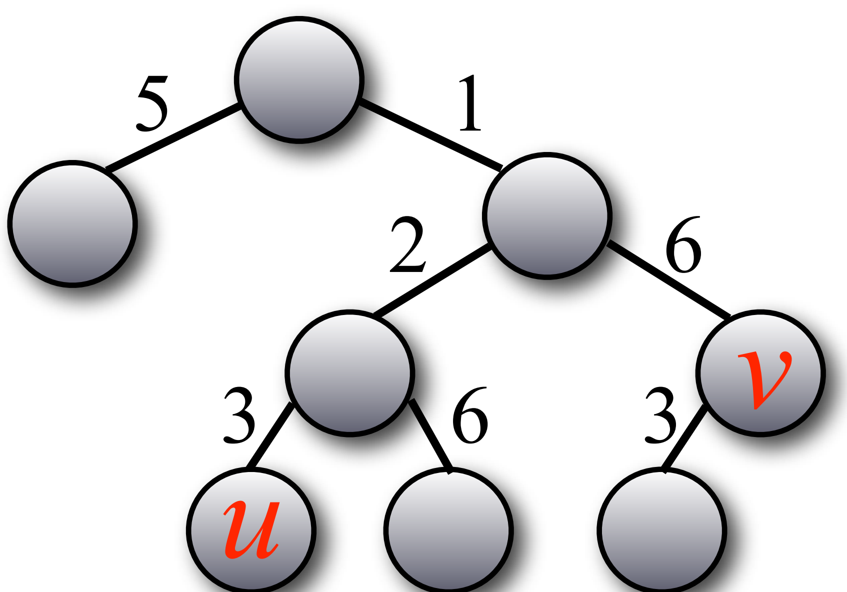
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

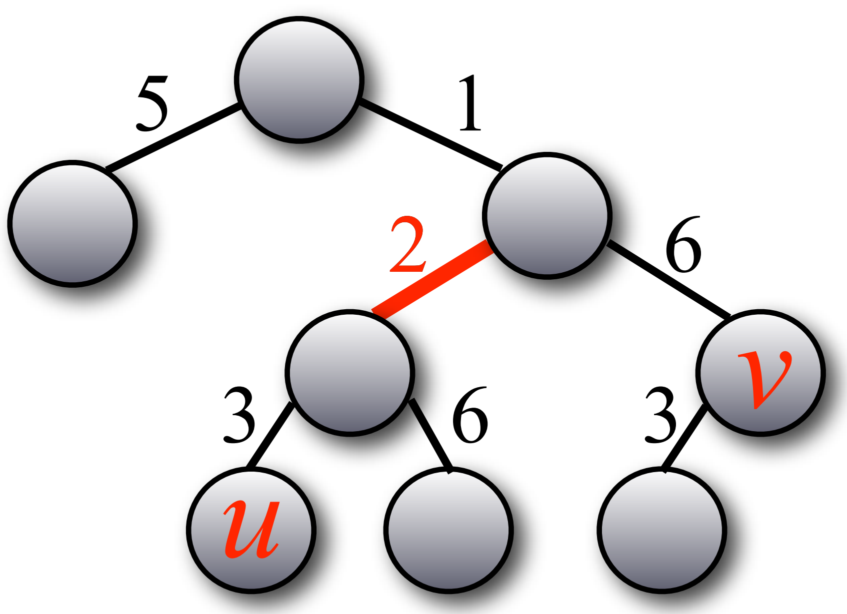
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

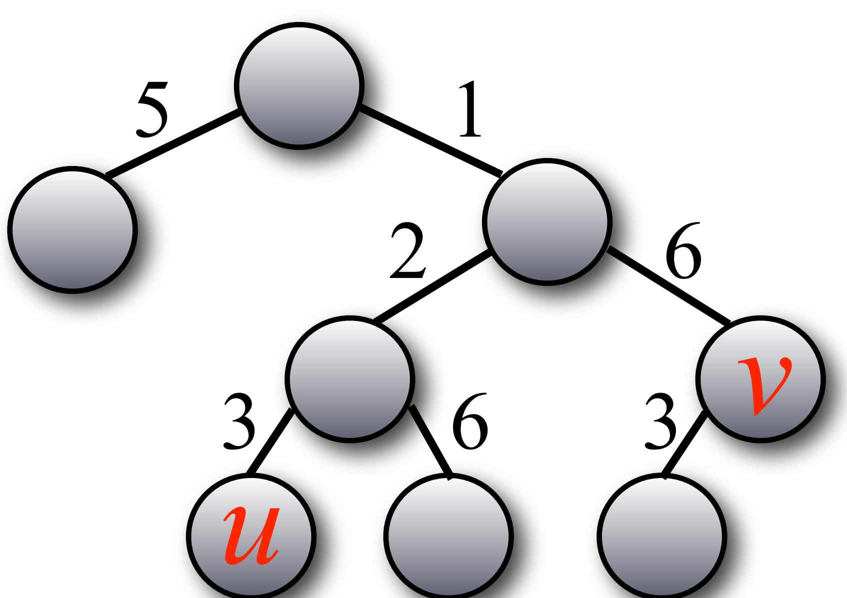
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

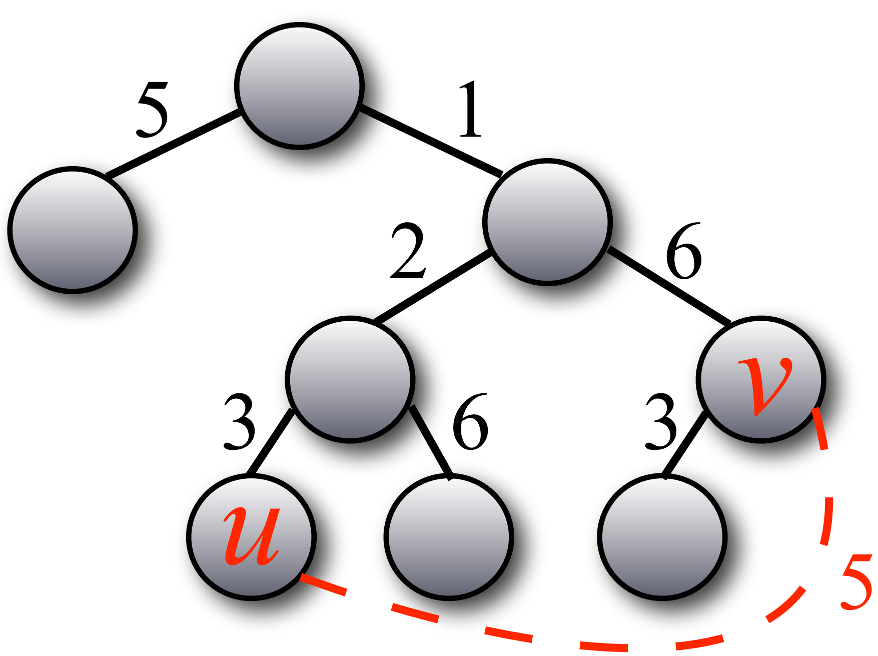
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

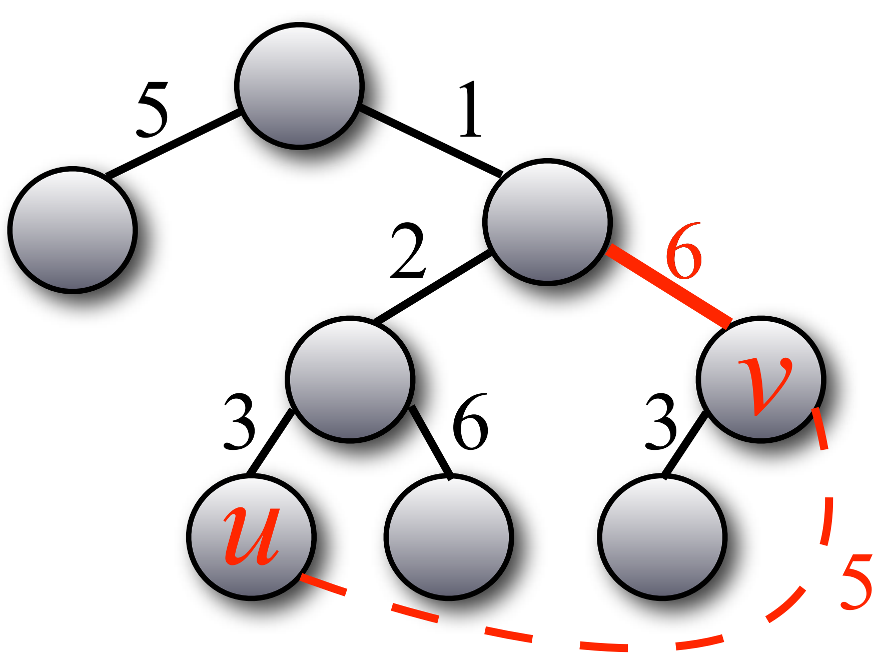
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

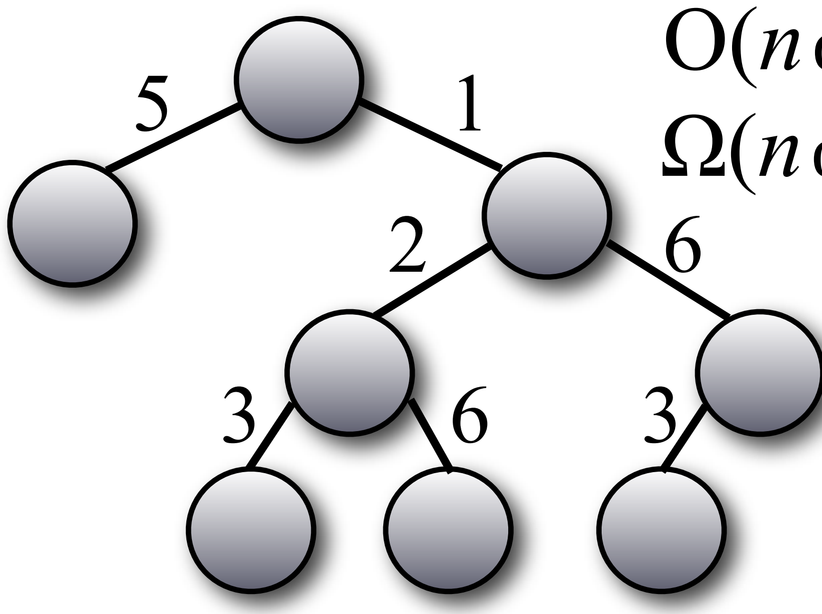
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem

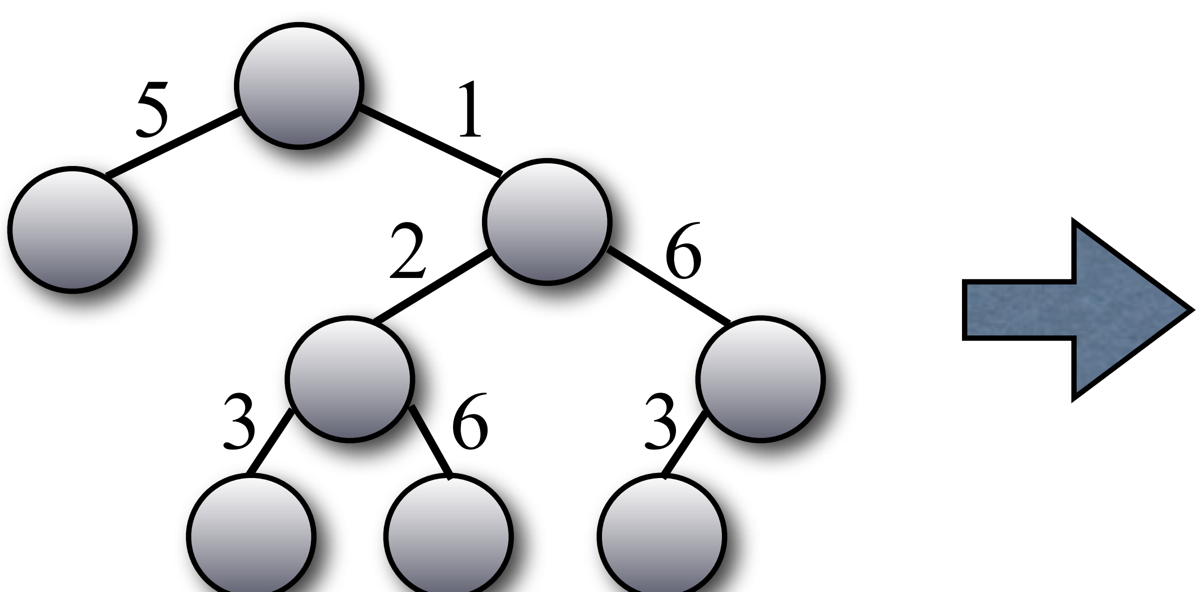


$O(n\,\alpha_k(n))$ prep. $O(k)$ query [Alon and Schieber 1987]

$\Omega(n\,\alpha_k(n))$ prep. for $O(k)$ query [Pettie 2002]

RMQ Generalization I: A Cartesian Tree of a Tree

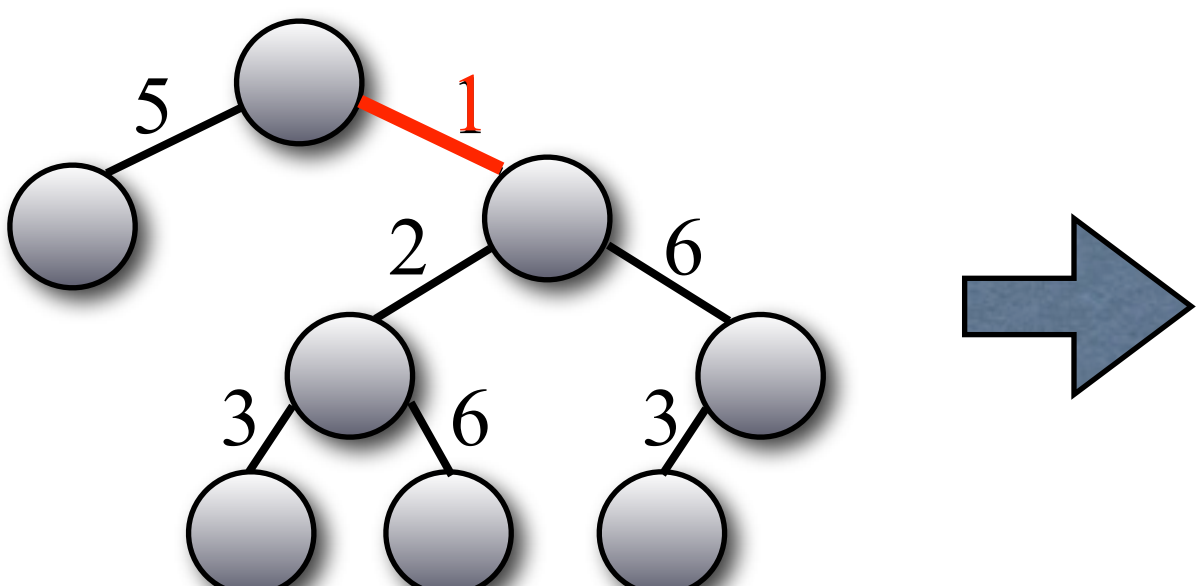
- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

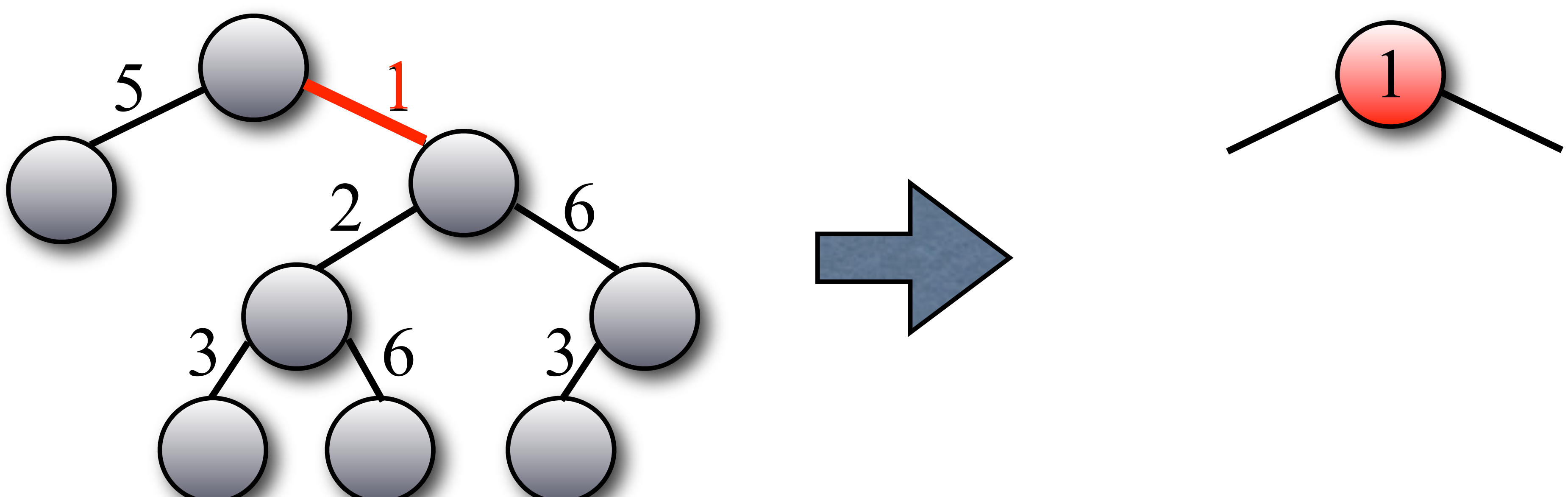
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

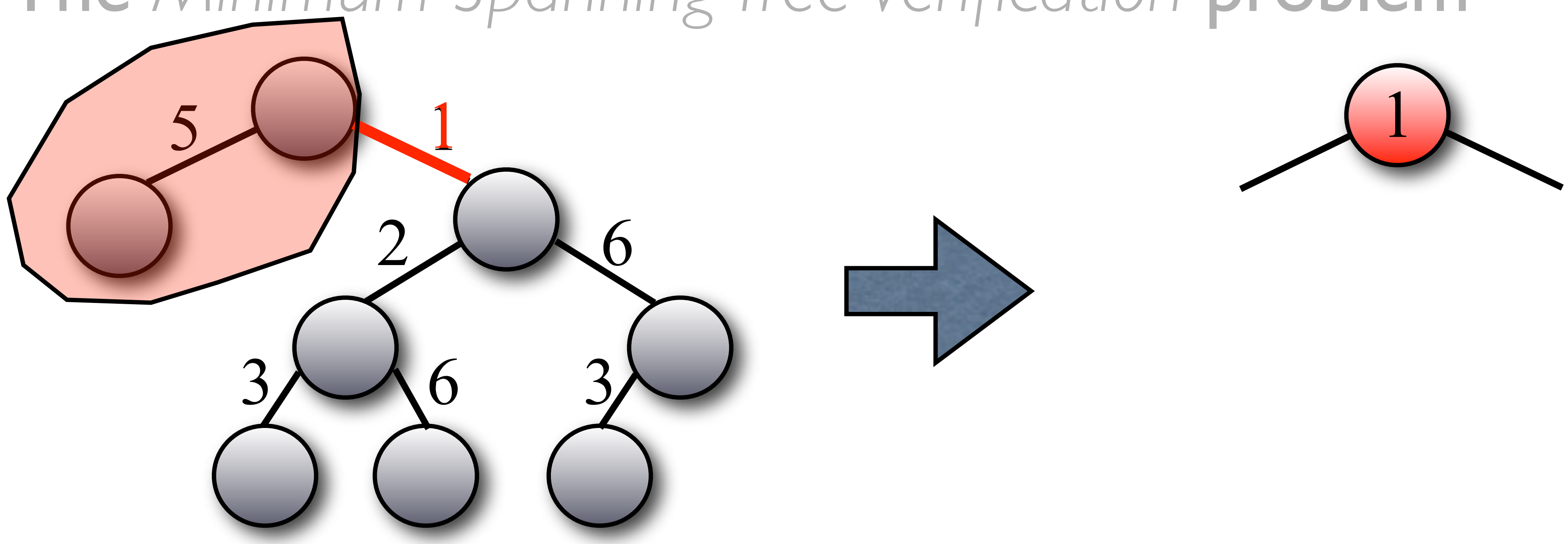
- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



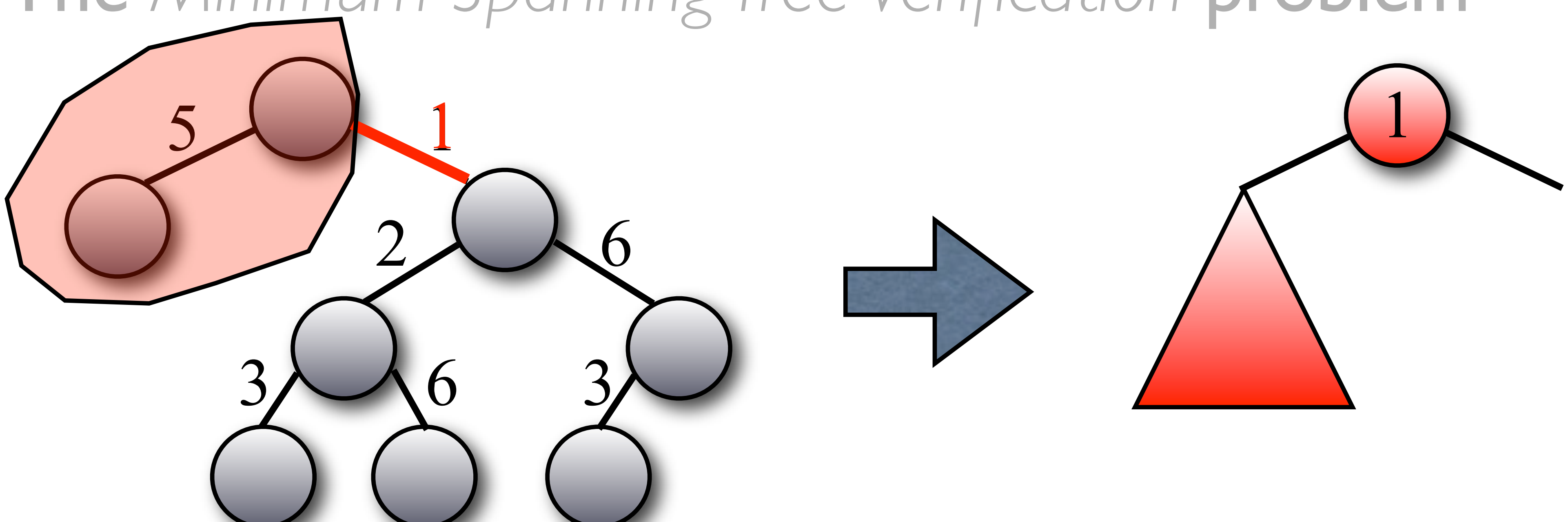
RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



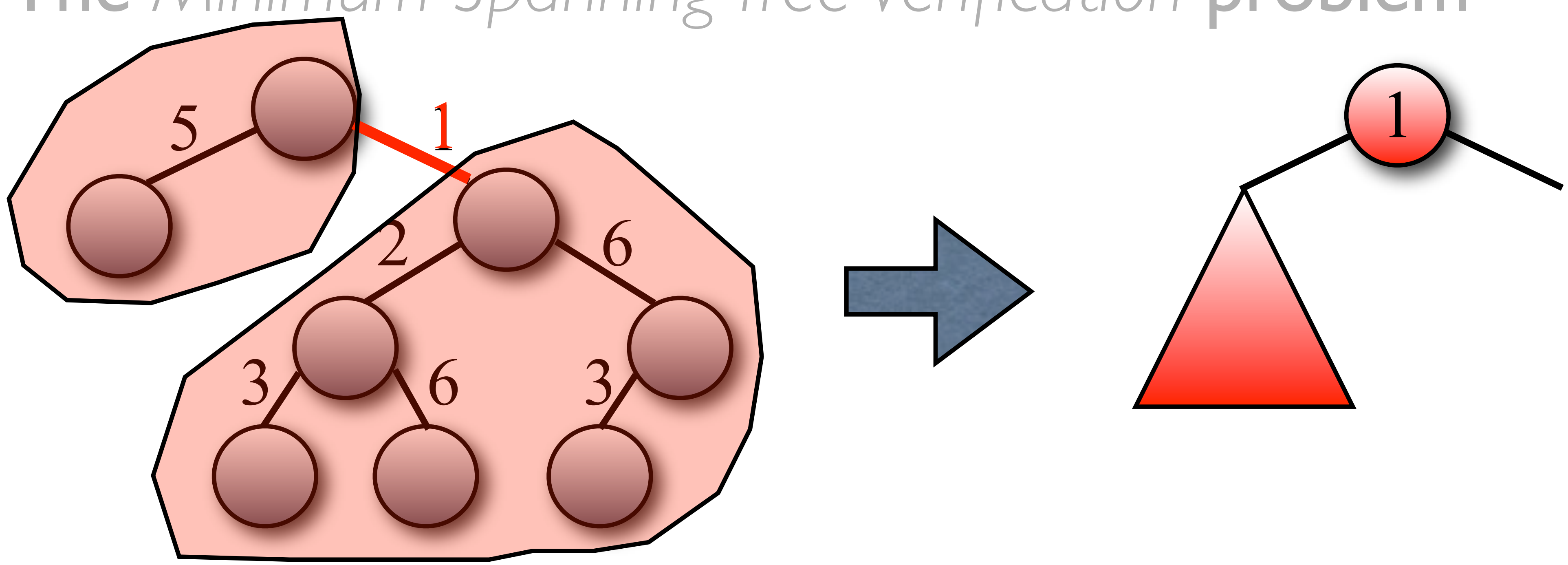
RMQ Generalization I: A Cartesian Tree of a Tree

- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

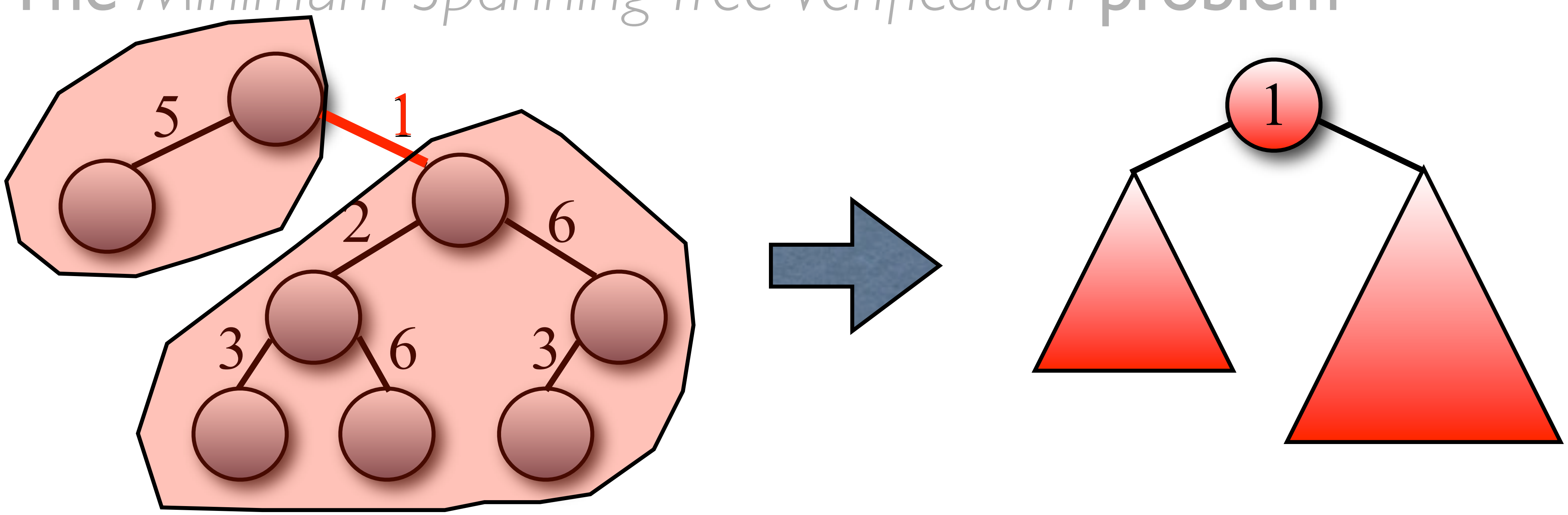
- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path
- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



RMQ Generalization I: A Cartesian Tree of a Tree

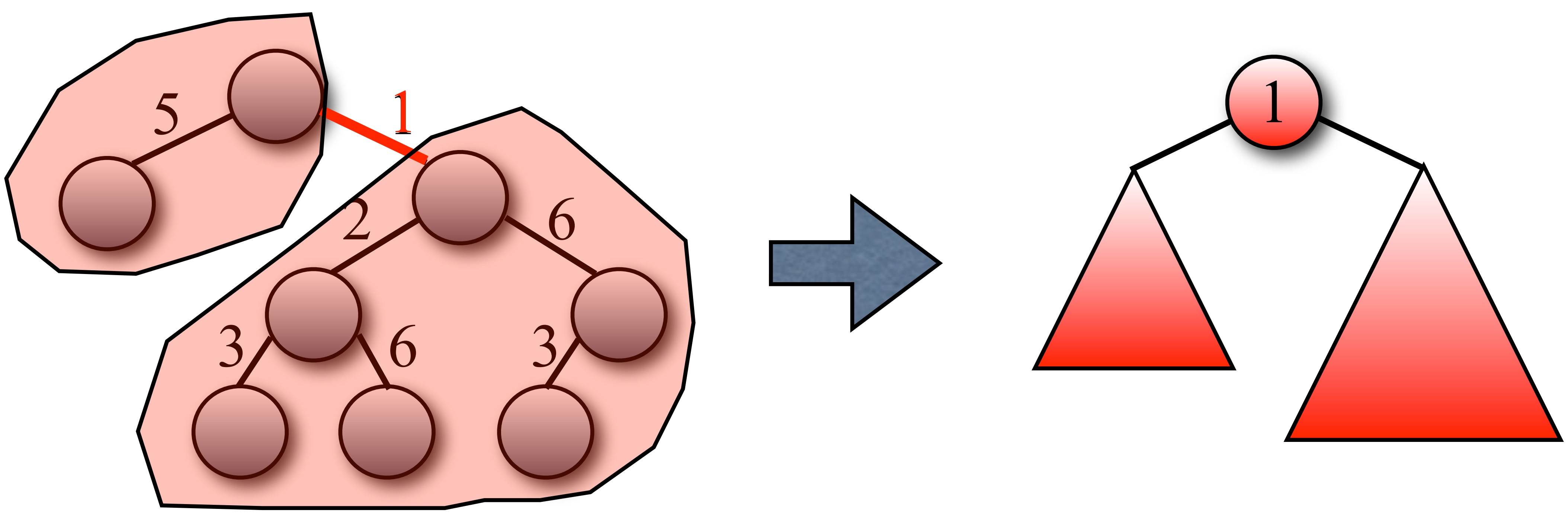
- The *Bottleneck Edge Query* problem (RMQ on graphs\trees):
  - preprocess an edge-weighted graph
  - query asks for max flow that can be routed between *u,v* along any simple path

- Undirected graphs: solve problem on the Maximum Spanning Tree [Hu 1961]
- The *Minimum Spanning Tree Verification* problem



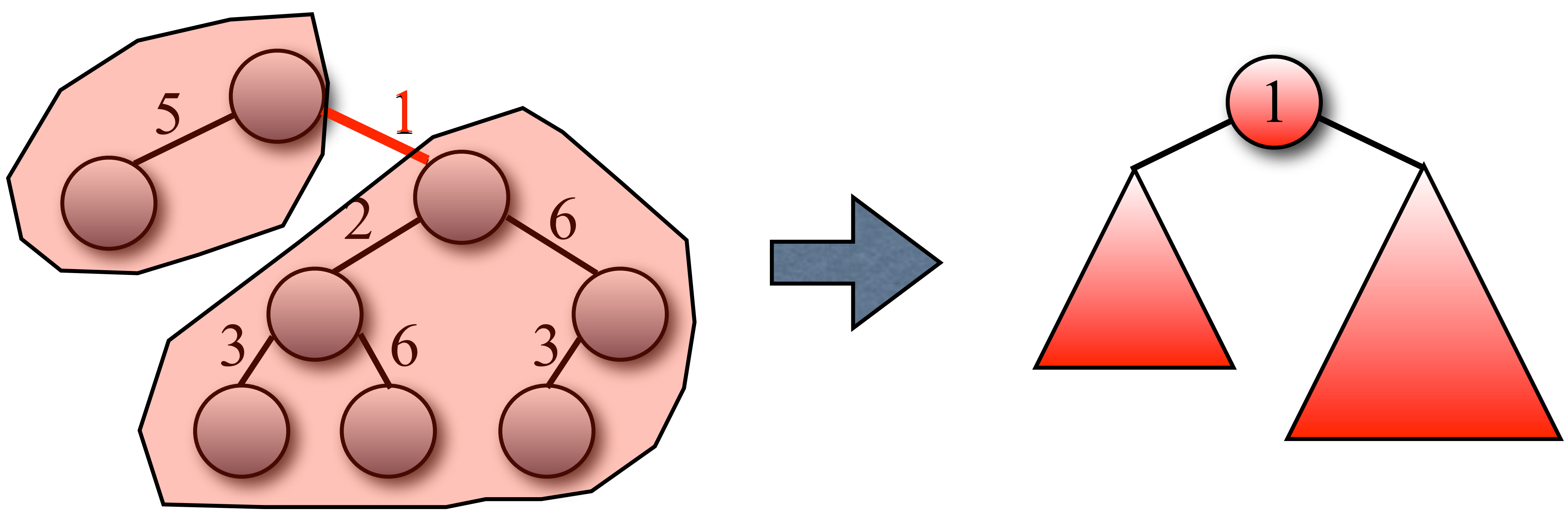
RMQ Generalization I: A Cartesian Tree of a Tree

- Construct Cartesian tree of an input tree in $O(n + \mathrm{sort(edges)})$



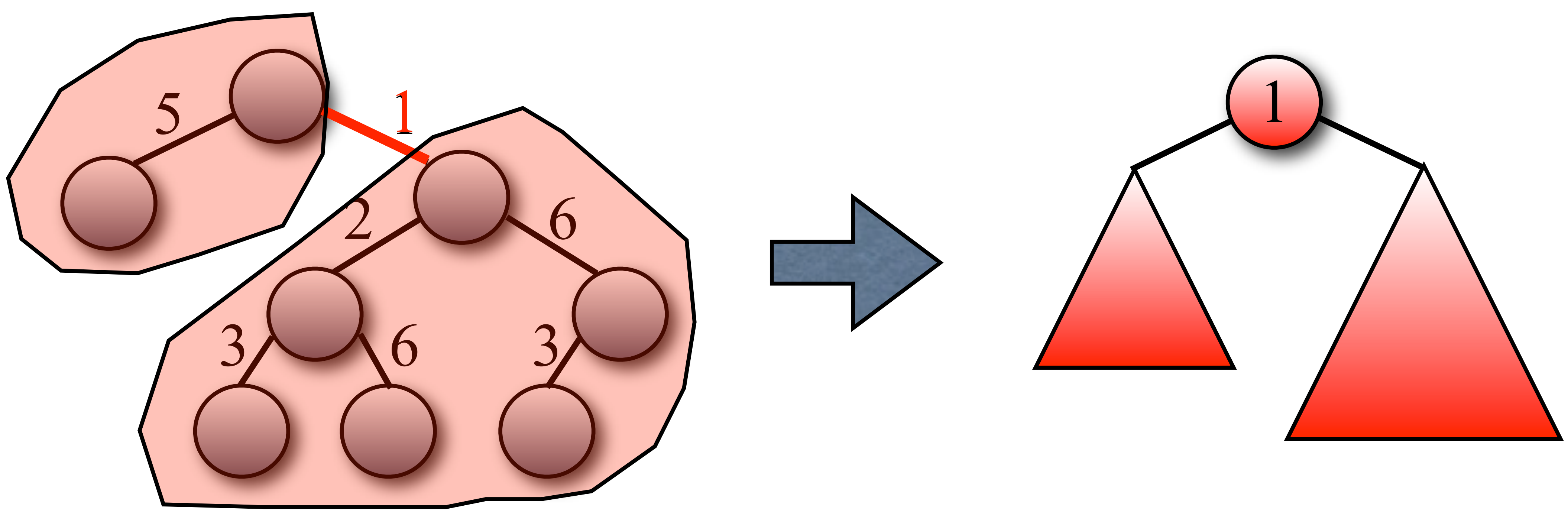
RMQ Generalization I: A Cartesian Tree of a Tree

- Construct Cartesian tree of an input tree in $O(n + \text{sort(edges)})$
- Tight lower bound



RMQ Generalization I: Compared to AlonSchieber

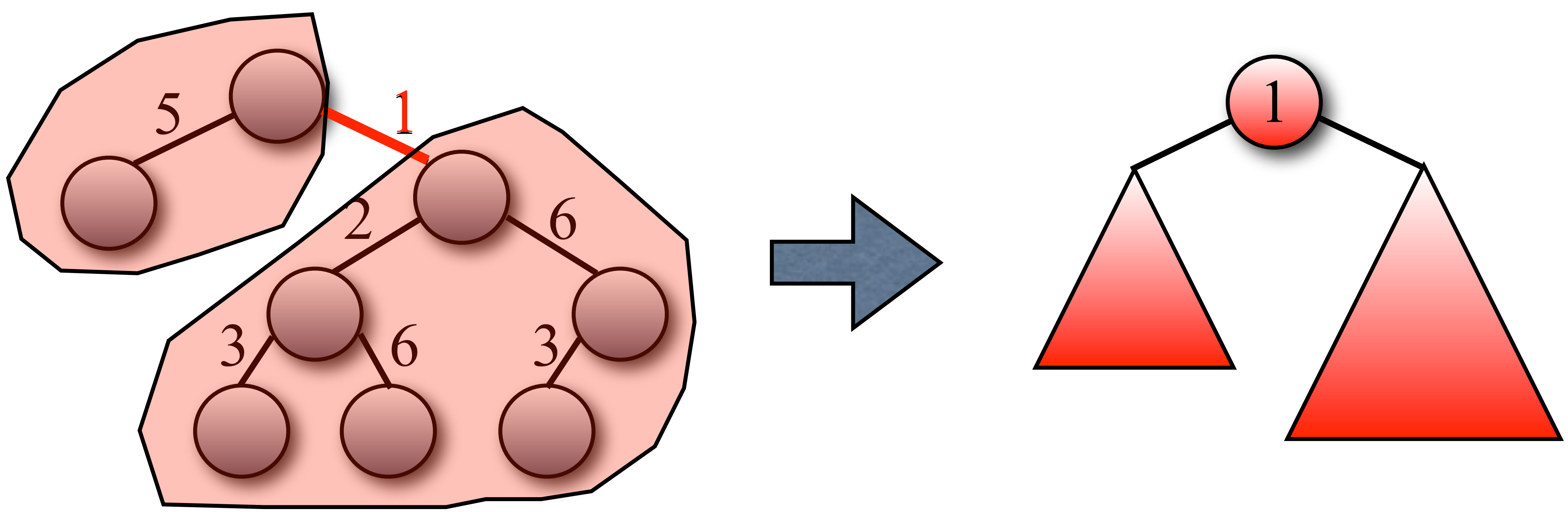
- Construct Cartesian tree of an input tree in $O(n + \text{sort(edges)})$
- Tight lower bound



RMQ Generalization I: Compared to AlonSchieber

- Construct Cartesian tree of an input tree in $O(n + \text{sort(edges)})$
- Tight lower bound

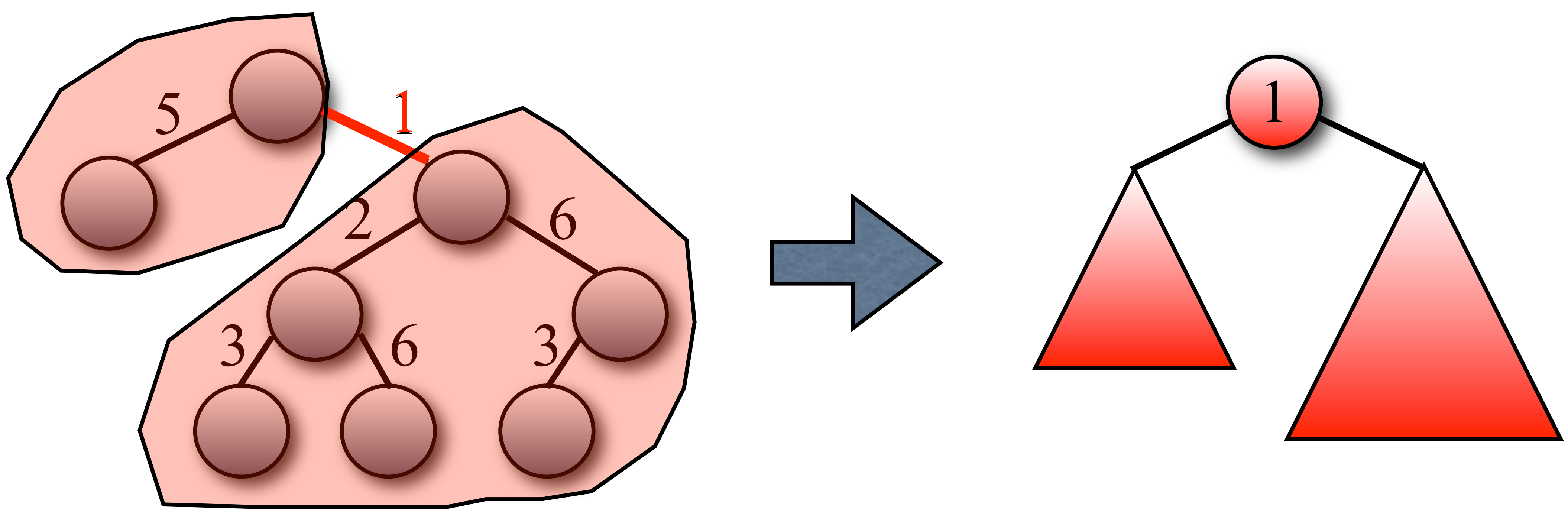
- Linear-time if edge-weights are sorted or integers



RMQ Generalization I: Compared to AlonSchieber

- Construct Cartesian tree of an input tree in $O(n + \text{sort(edges)})$
- Tight lower bound

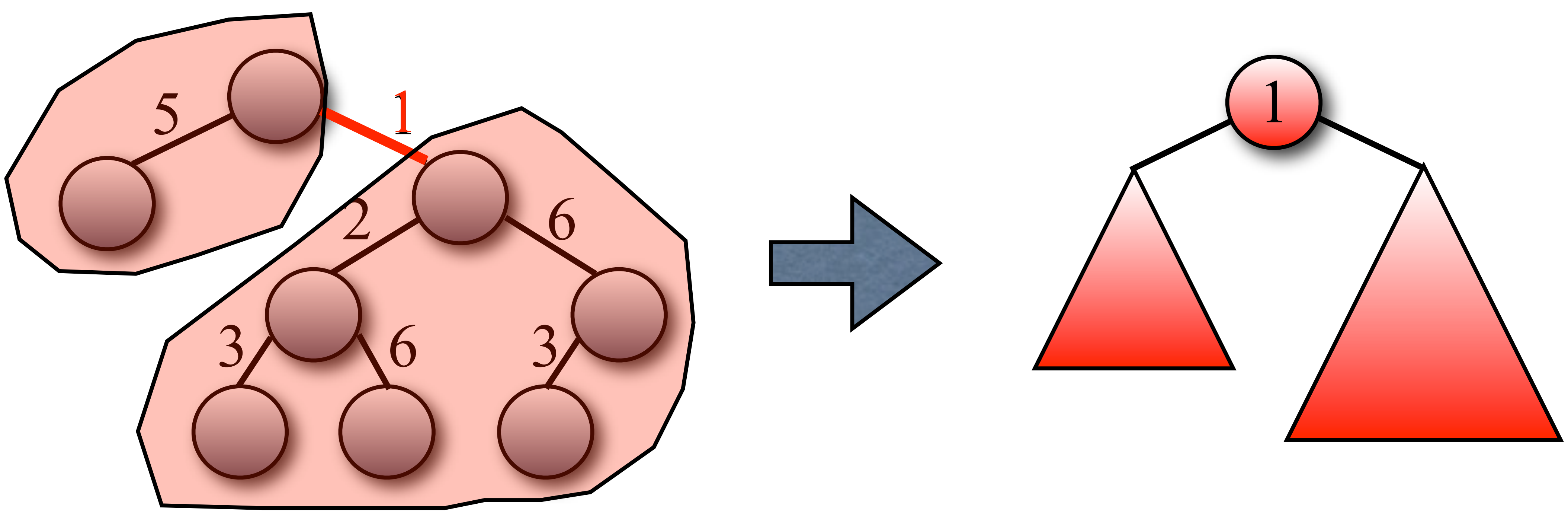
- Linear-time if edge-weights are sorted or integers
- Otherwise we get linear-space, $O(n \log^{[k]} n)$ prep. $O(k)$ query



RMQ Generalization I: Compared to AlonSchieber

- Construct Cartesian tree of an input tree in $O(n + \mathrm{sort}(\mathrm{edges}))$
- Tight lower bound

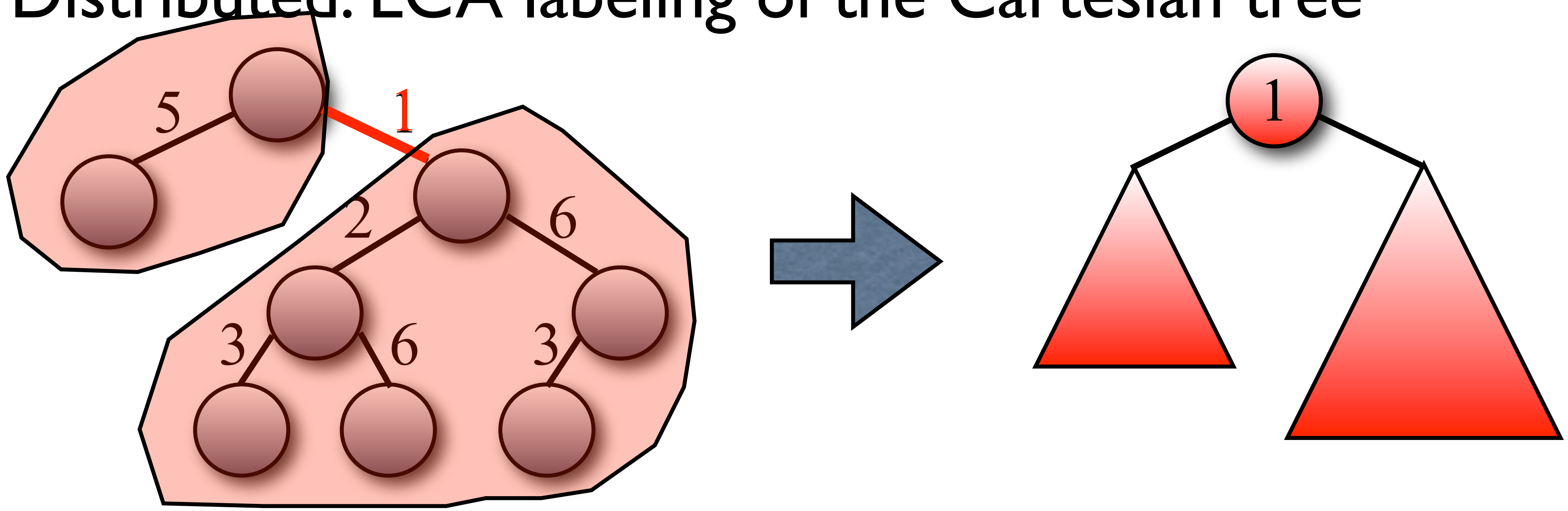
- Linear-time if edge-weights are sorted or integers
- Otherwise we get linear-space, $O(n \log^{[k]} n)$ prep. $O(k)$ query
- Maintain dynamic Cartesian tree and LCA info



RMQ Generalization I: Compared to AlonSchieber

- Construct Cartesian tree of an input tree in $O(n + \mathrm{sort(edges)})$
- Tight lower bound

- Linear-time if edge-weights are sorted or integers
- Otherwise we get linear-space, $O(n \log^{[k]} n)$ prep. $O(k)$ query
- Maintain dynamic Cartesian tree and LCA info
- Distributed: LCA labeling of the Cartesian tree



RMQ Generalization II: 2D Cartesian Tree

[Demaine, Landau and W. 2009]

RMQ Generalization II: 2D Cartesian Tree

- The *2D-RMQ* problem:

RMQ Generalization II: 2D Cartesian Tree

- The *2D-RMQ* problem:

2	0	4	3	1	9	3	3
7	3	4	3	5	0	7	6
2	0	3	8	5	6	4	4
4	7	4	3	5	8	8	6
2	8	1	8	5	1	7	6
2	0	4	3	5	5	7	6

RMQ Generalization II: 2D Cartesian Tree

- The *2D-RMQ* problem:

2	0	4	3	1	9	3	3
7	3	4	3	5	0	7	6
2	0	3	8	5	6	4	4
4	7	4	3	5	8	8	6
2	8	1	8	5	1	7	6
2	0	4	3	5	5	7	6

RMQ Generalization II: 2D Cartesian Tree

- The *2D-RMQ* problem:

2	0	4	3	1	9	3	3
7	3	4	3	5	0	7	6
2	0	3	8	5	6	4	4
4	7	4	3	5	8	8	6
2	8	1	8	5	1	7	6
2	0	4	3	5	5	7	6

- $O(n^2 \log n)$ prep. $O(\log n)$ query [Gabow, Bentley, Tarjan 1984]
- $O(n^2 \alpha_k(n)^2)$ prep. $O(k)$ query [Chazelle, Rosenberg 1989]
- $O(n^2 \log^{[k]} n)$ prep, $O(n^2)$ space, $O(k)$ query [Amir, Fischer, Lewenstein 2007]

RMQ Generalization II: 2D Cartesian Tree

- The *2D-RMQ* problem:

No 2D Cartesian tree:
different 2D-RMQ matrices ≈ $n^2!$

2	0	4	3	1	9	3	3
7	3	4	3	5	0	7	6
2	0	3	8	5	6	4	4
4	7	4	3	5	8	8	6
2	8	1	8	5	1	7	6
2	0	4	3	5	5	7	6

- $O(n^2 \log n)$ prep. $O(\log n)$ query [Gabow, Bentley,Tarjan 1984]
- $O(n^2 \alpha_k(n)^2)$ prep. $O(k)$ query [Chazelle, Rosenberg 1989]
- $O(n^2 \log^{[k]} n)$ prep, $O(n^2)$ space, $O(k)$ query [Amir, Fischer, Lewenstein 2007]

RMQ Generalization II: 2D Cartesian Tree

- The *2D-RMQ* problem:

No 2D Cartesian tree:

\# *different* 2D-RMQ matrices ≈ $n^2!$

2	0	4	3	1	9	3	3
7	3	4	3	5	0	7	6
2	0	3	8	5	6	4	4
4	7	4	3	5	8	8	6
2	8	1	8	5	1	7	6
2	0	4	3	5	5	7	6

- $O(n^2 \log n)$ prep. $O(\log n)$ query [Gabow, Bentley,Tarjan 1984]
- $O(n^2 \alpha_k(n)^2)$ prep. $O(k)$ query [Chazelle, Rosenberg 1989]
- $O(n^2 \log^{[k]} n)$ prep, $O(n^2)$ space, $O(k)$ query [Amir, Fischer, Lewenstein 2007]
- $O(n^2)$ prep. $O(1)$ query [Yuan, Atallah 2010]

Thank You!