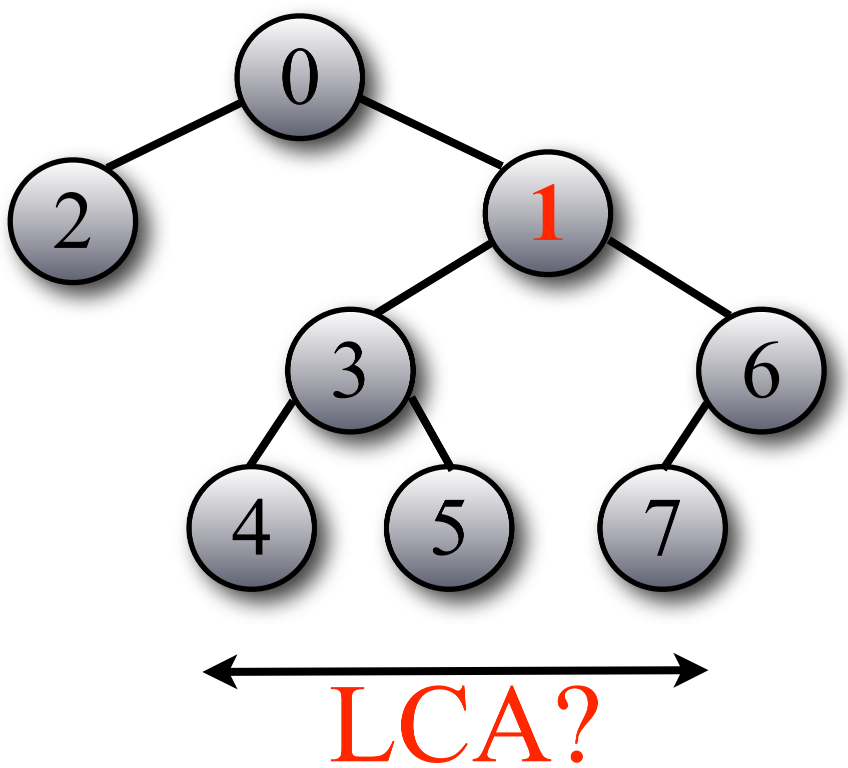
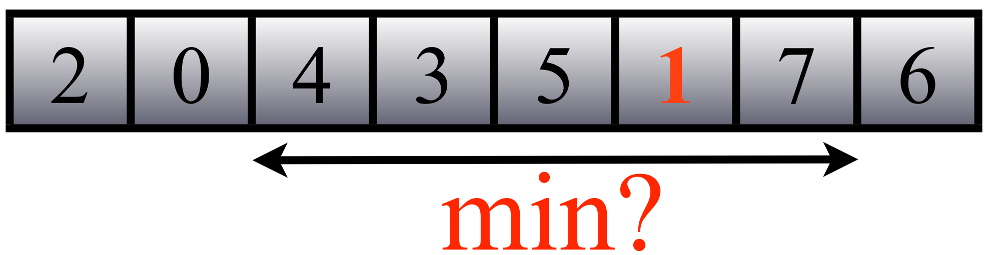
# On Cartesian Trees, Lowest Common Ancestors, and Range Minimum Queries

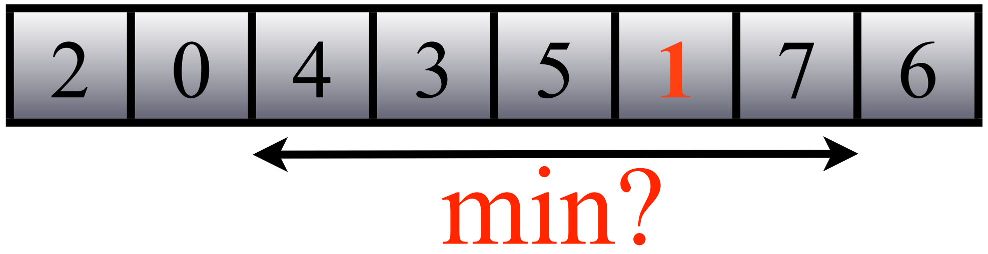


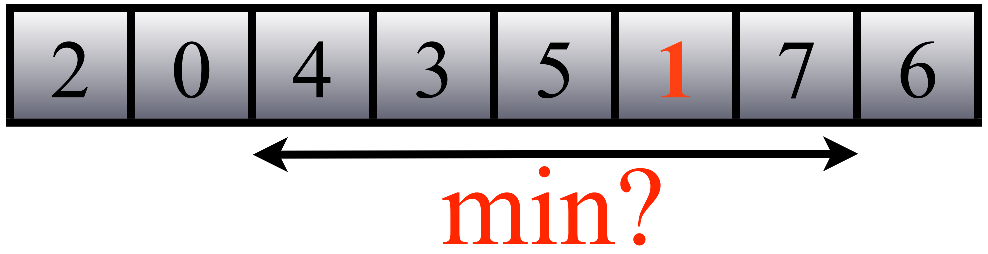
# RMQ

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 4 | 3 | 5 | 1 | 7 | 6 |

# RMQ



# RMQ



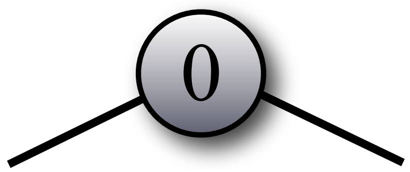

- Applications:
  - String Processing & Computational Biology (Suffix array\tree)
  - Search Engines and Document Retrieval
  - Equivalence to LCA
  - Database Queries

⋮

# RMQ
# & Cartesian Trees

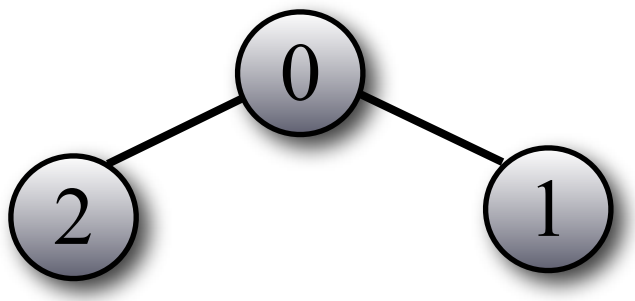
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 4 | 3 | 5 | 1 | 7 | 6 |

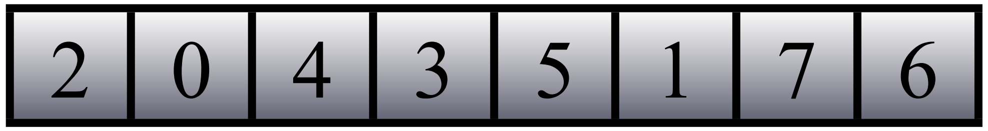
# RMQ
# & Cartesian Trees



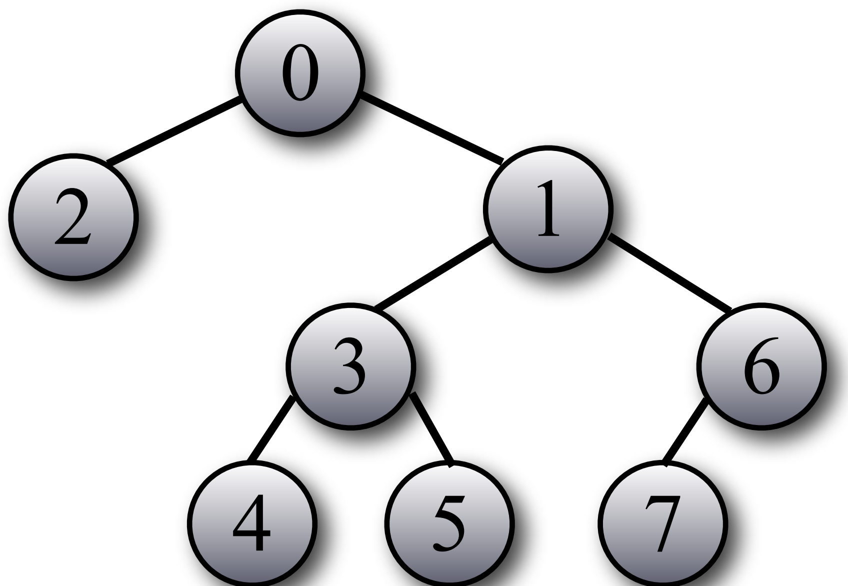

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 4 | 3 | 5 | 1 | 7 | 6 |

# RMQ
# & Cartesian Trees



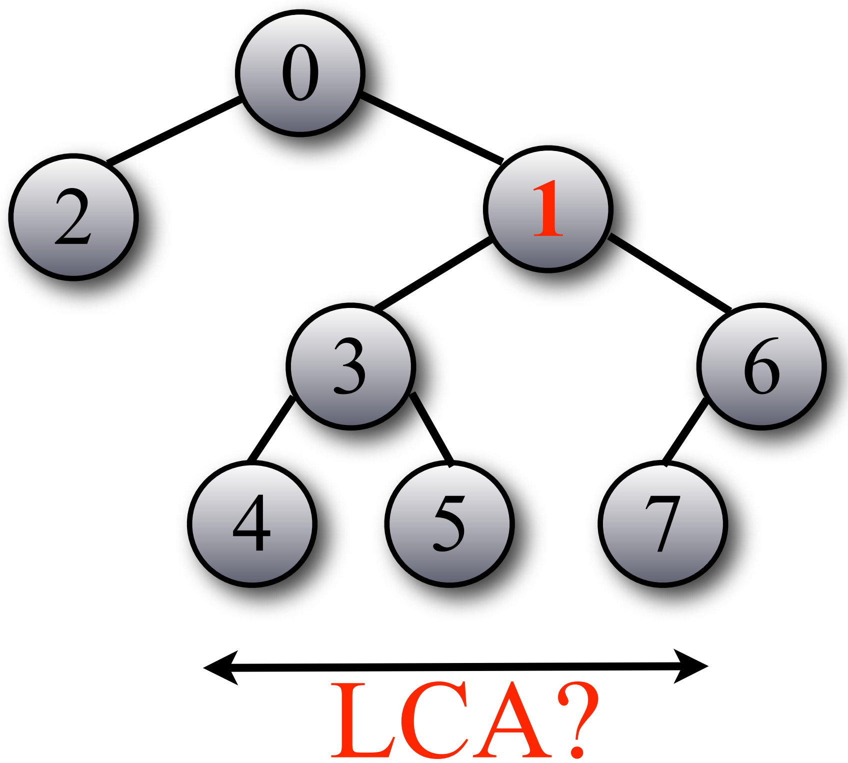



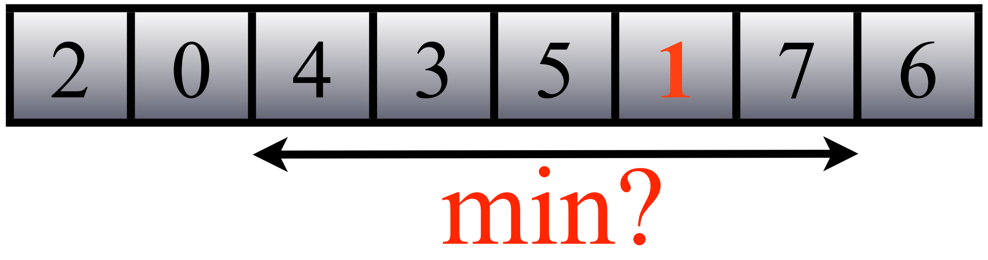
# RMQ
# & Cartesian Trees



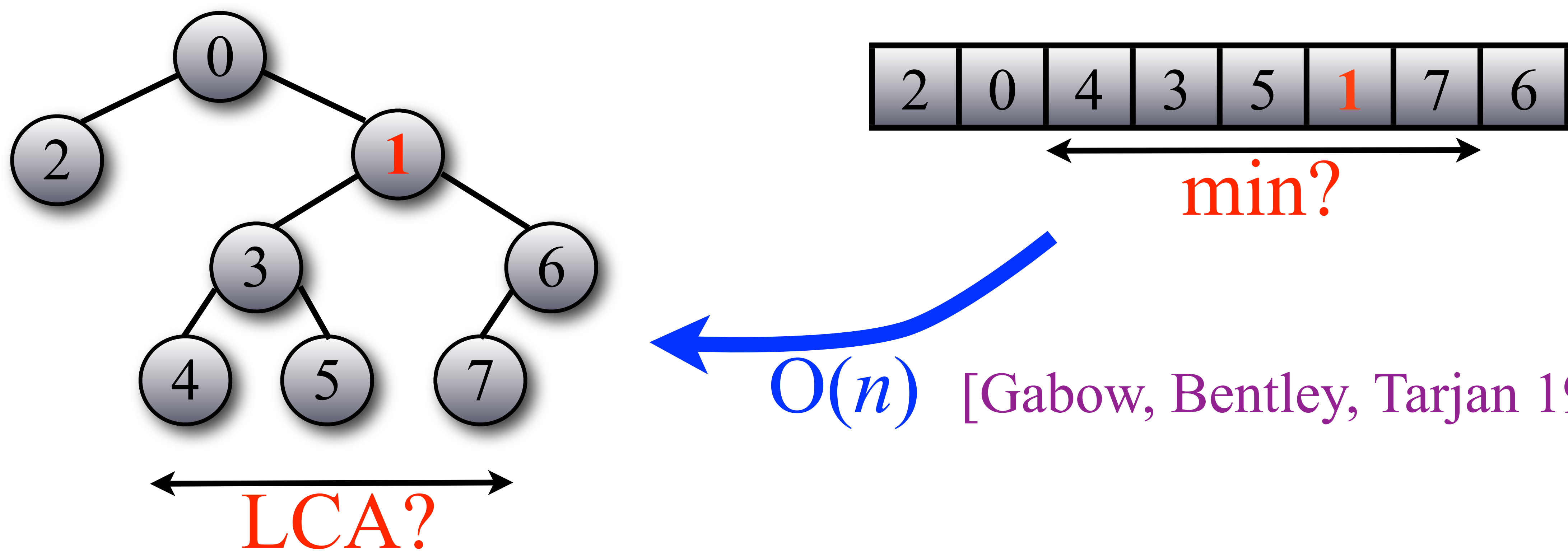

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 0 | 4 | 3 | 5 | 1 | 7 | 6 |

# RMQ
# & Cartesian Trees



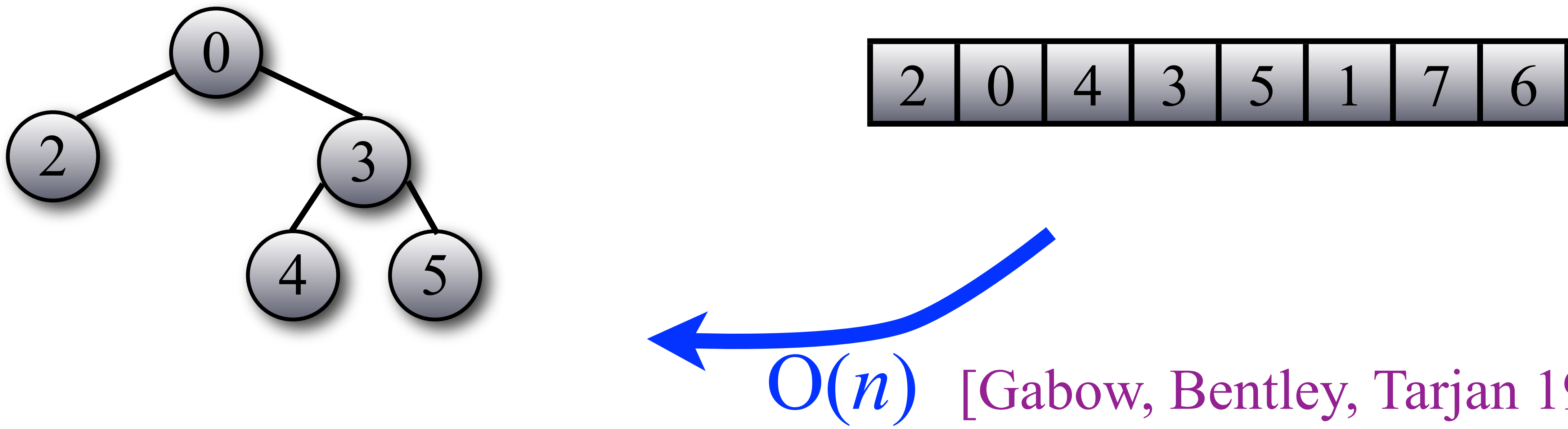



# RMQ & Cartesian Trees



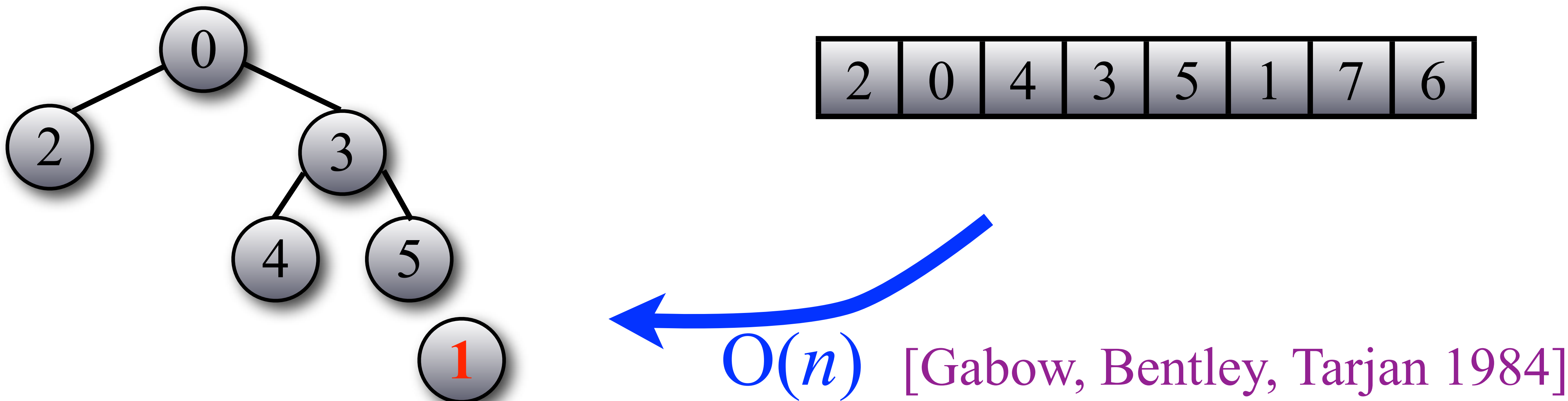

[Gabow, Bentley, Tarjan 1984]

# RMQ
# & Cartesian Trees

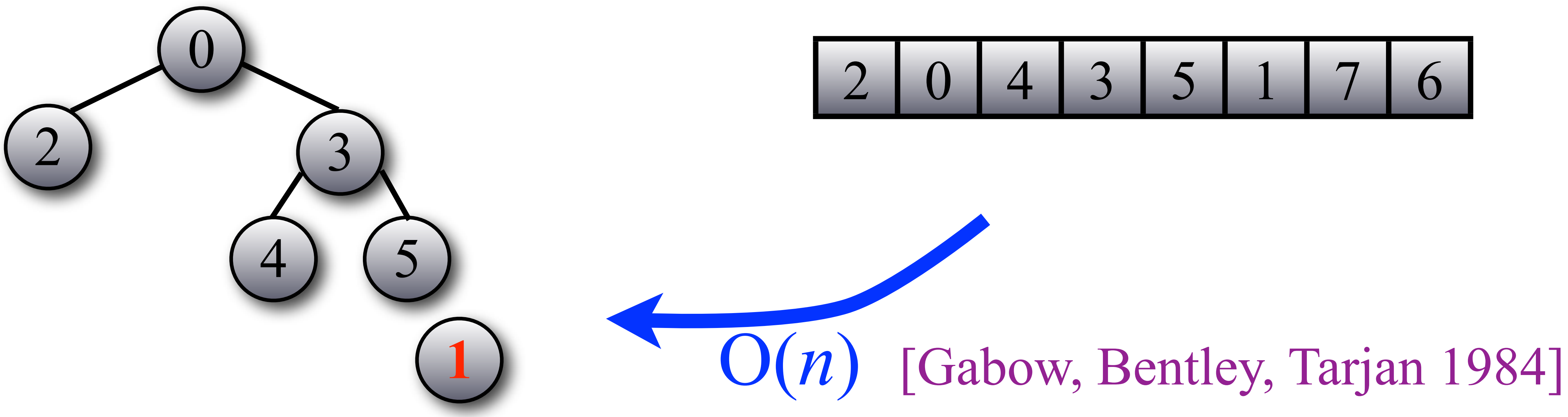



[Gabow, Bentley, Tarjan 1984]

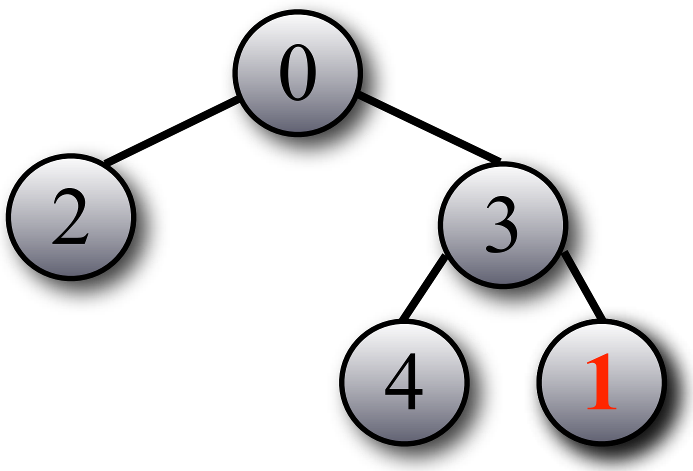
# RMQ
# & Cartesian Trees

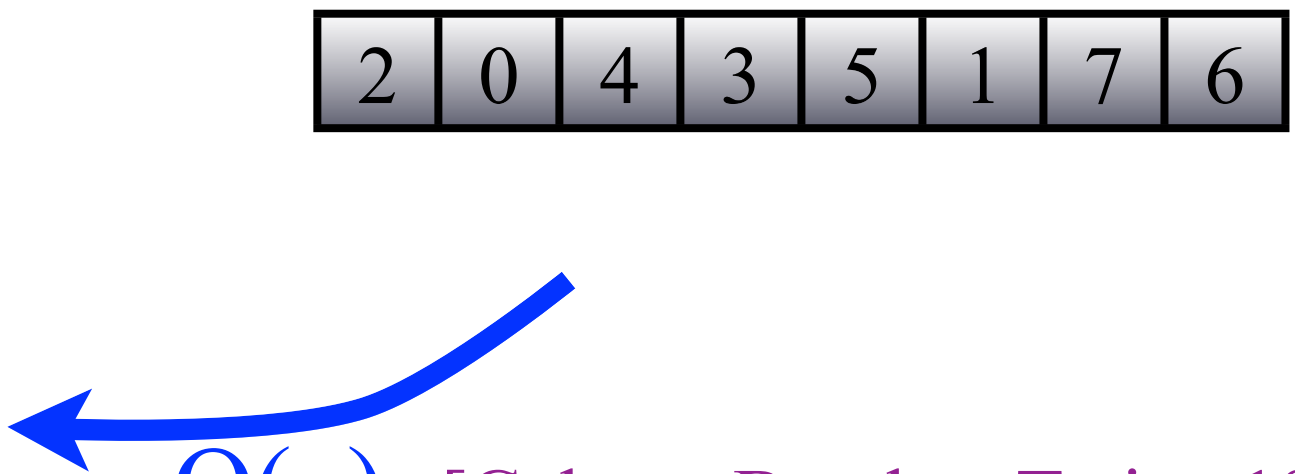


# RMQ
# & Cartesian Trees



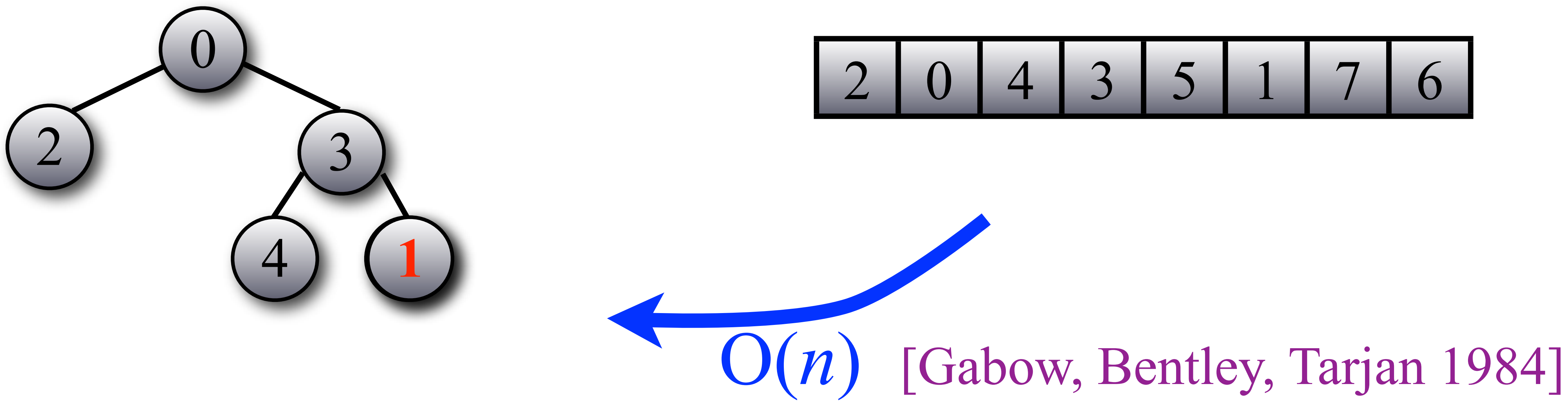
# RMQ
# & Cartesian Trees



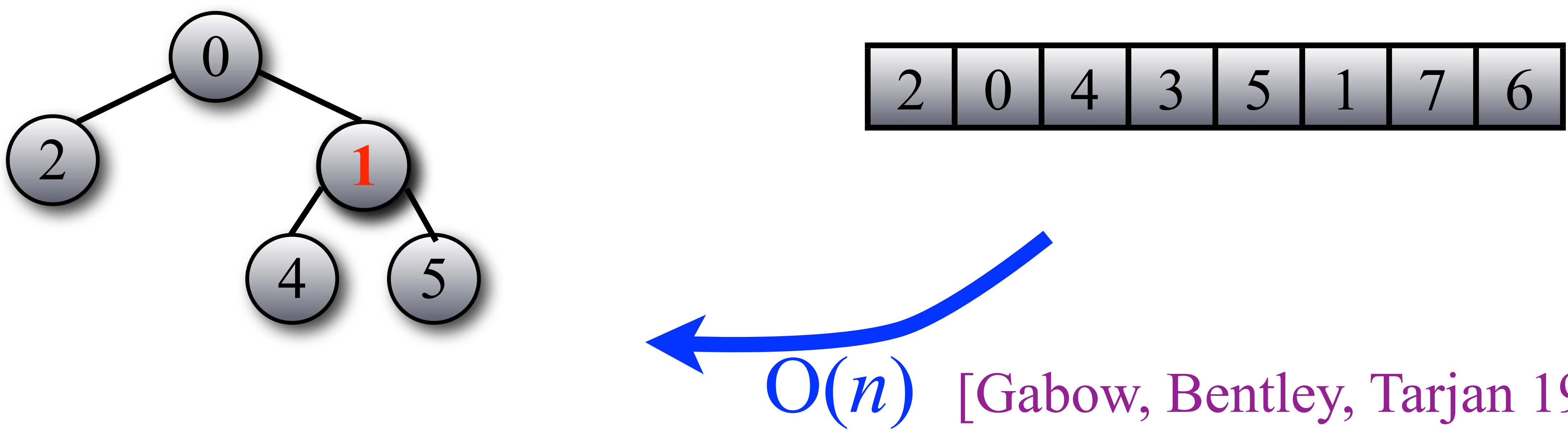




O($n$) [Gabow, Bentley, Tarjan 1984]

# RMQ
# & Cartesian Trees

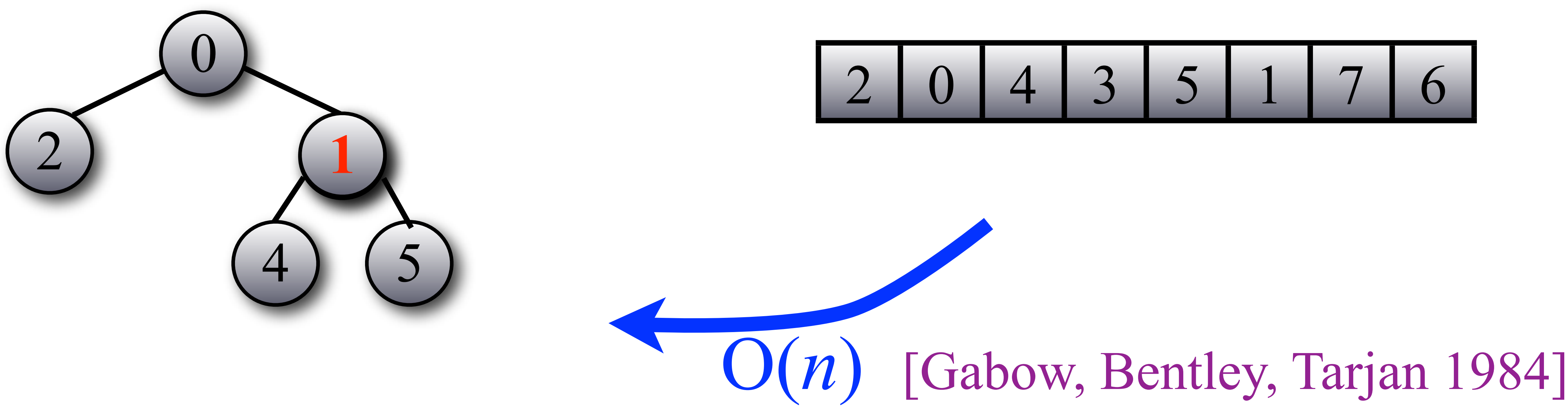


# RMQ
# & Cartesian Trees

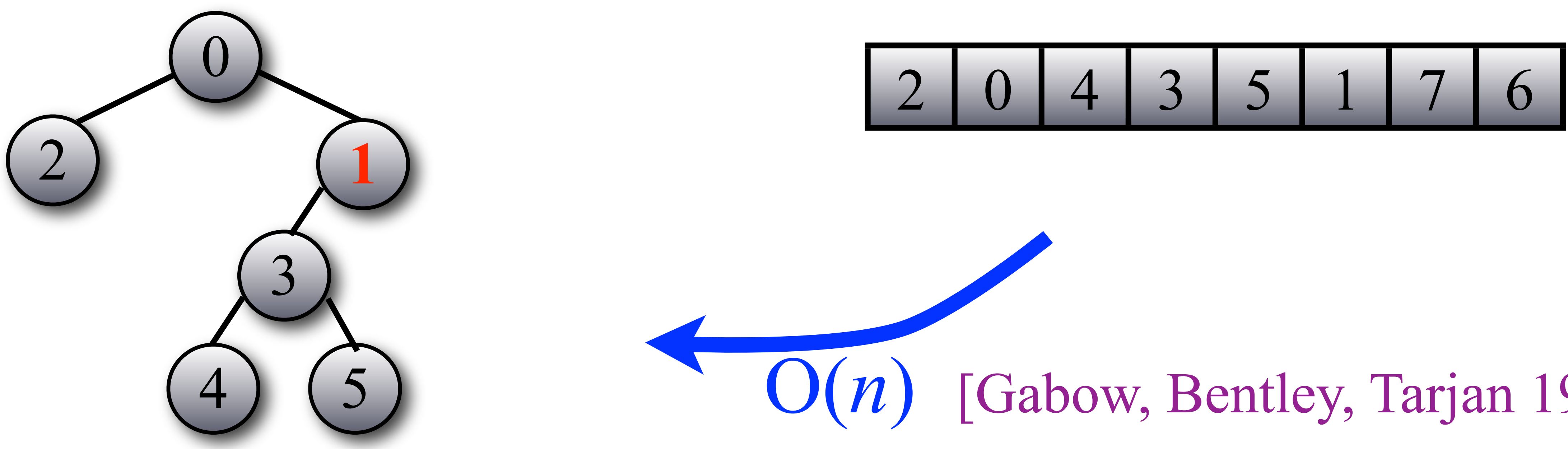



[Gabow, Bentley, Tarjan 1984]

# RMQ
# & Cartesian Trees

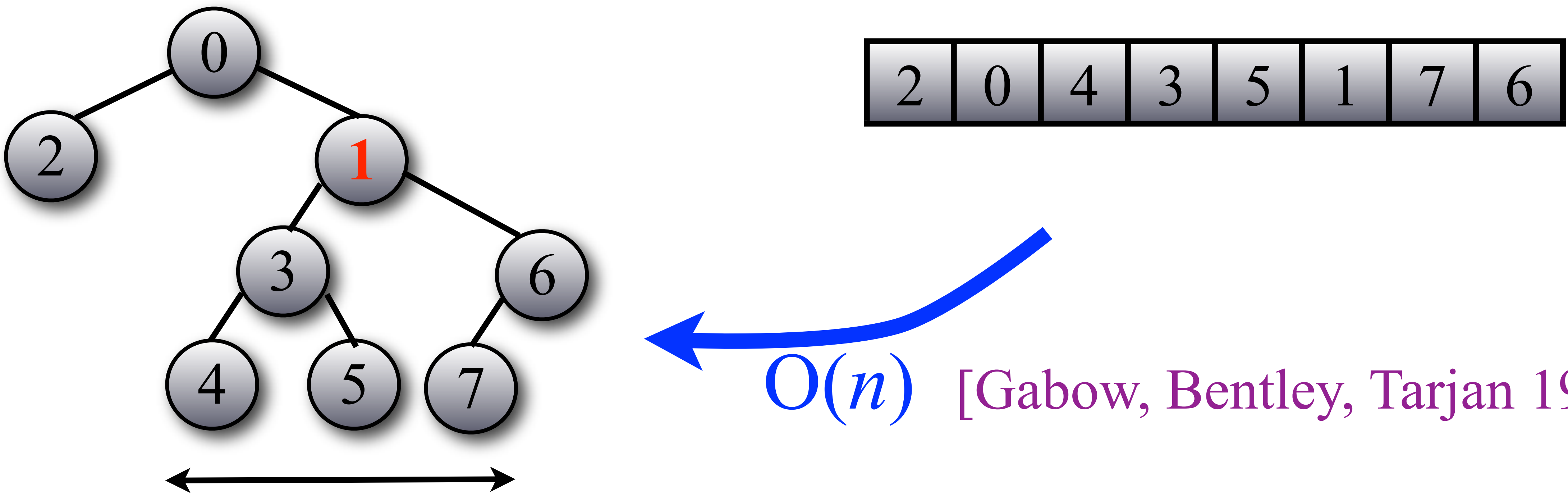


# RMQ
# & Cartesian Trees




[Gabow, Bentley, Tarjan 1984]

# RMQ & Cartesian Trees

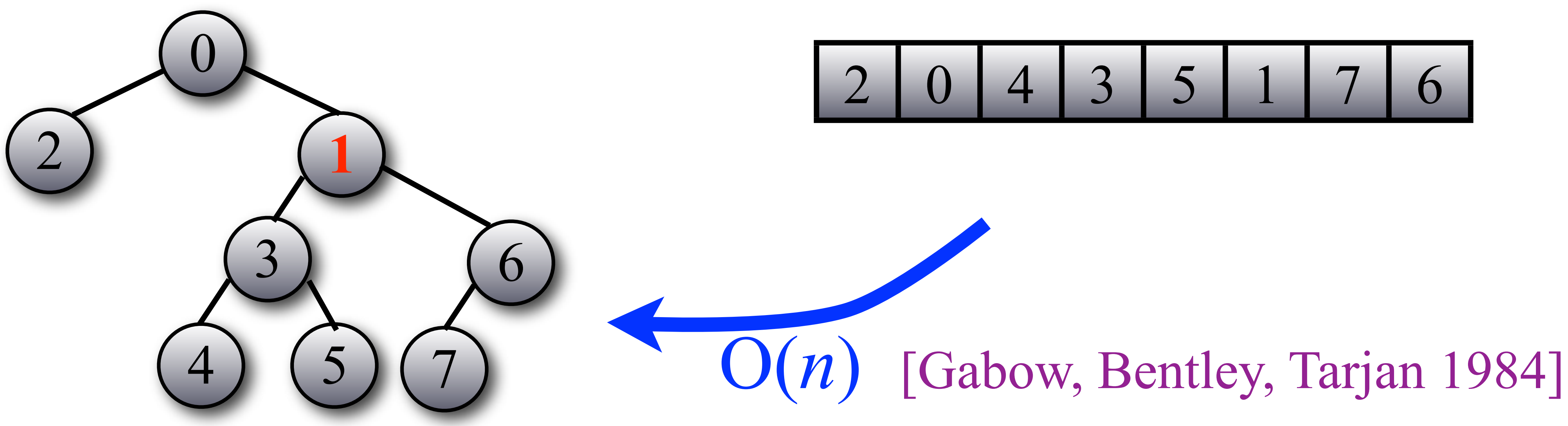



$O(n)$ [Gabow, Bentley, Tarjan 1984]

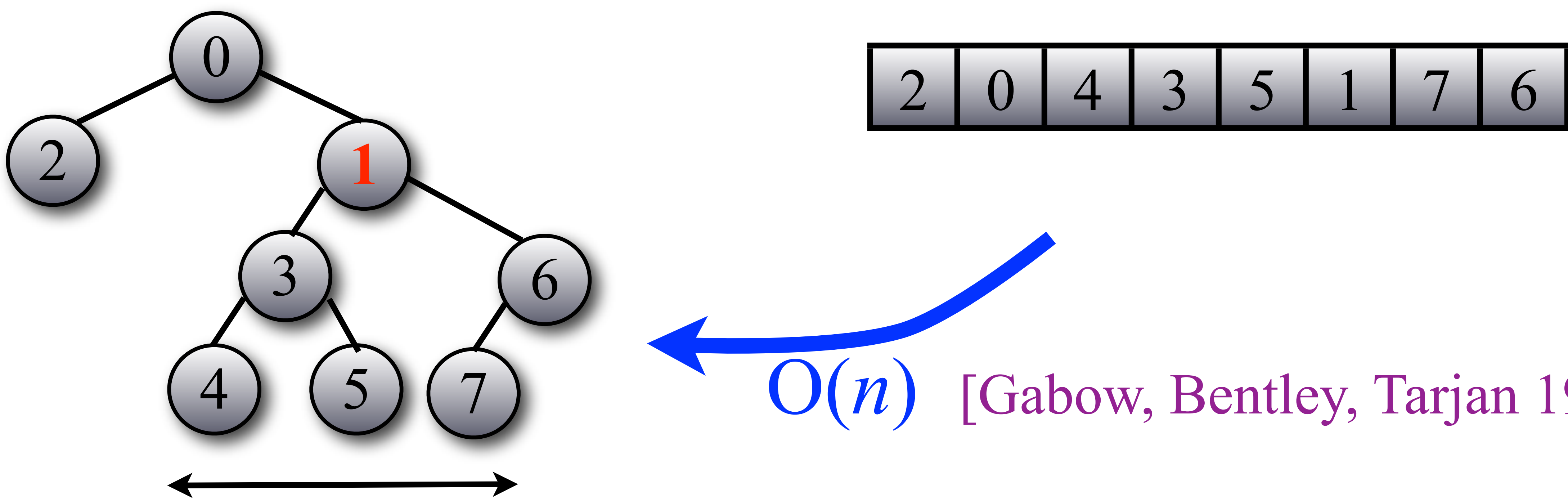
LCA: $O(n)$ prep. $O(1)$ query [Harel, Tarjan 1984]

Easy on complete binary tree MSB(u XOR v)

# RMQ
# & Cartesian Trees



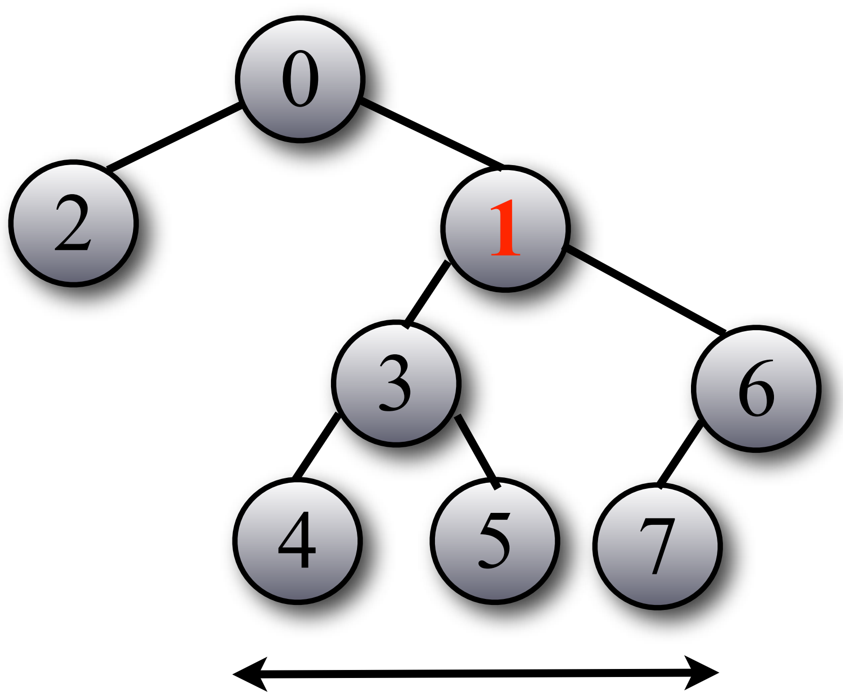
# RMQ
# & Cartesian Trees

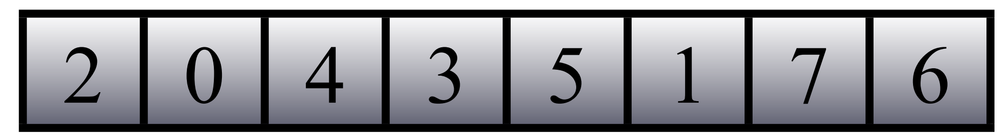



O($n$) [Gabow, Bentley, Tarjan 1984]

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]

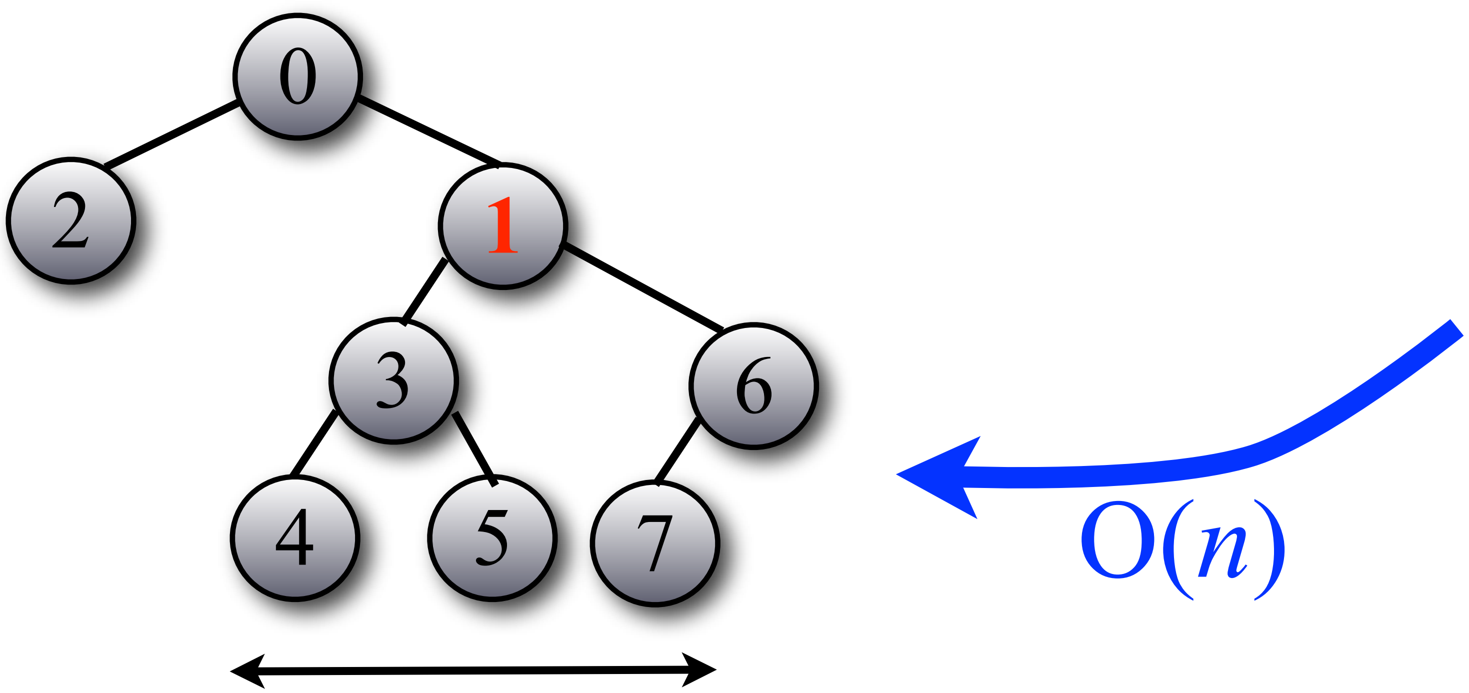
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

# RMQ
# & Cartesian Trees




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

[Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

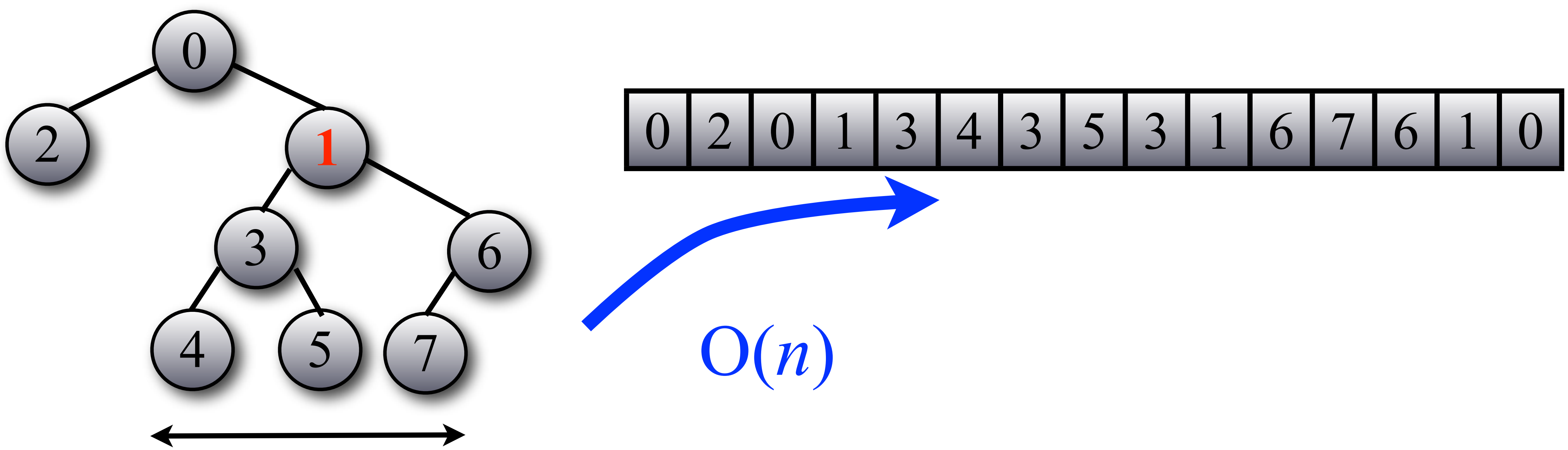
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

# RMQ
# & Cartesian Trees




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

[Schieber, Vishkin 1988]

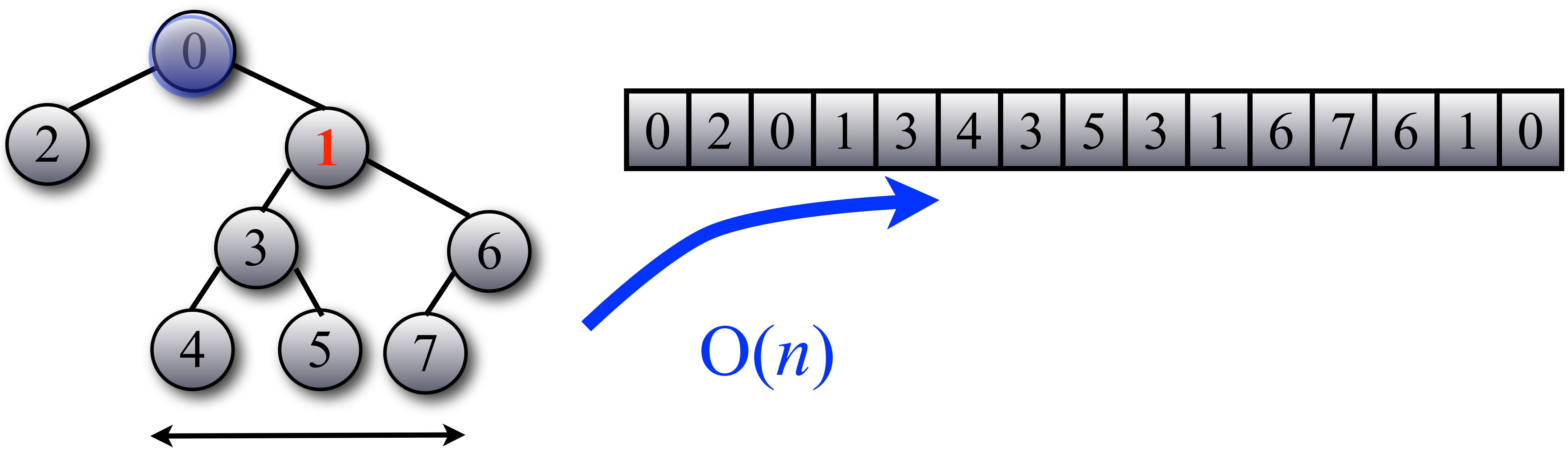
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

# RMQ
# & Cartesian Trees




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

[Harel, Tarjan 1984]

[Schieber, Vishkin 1988]

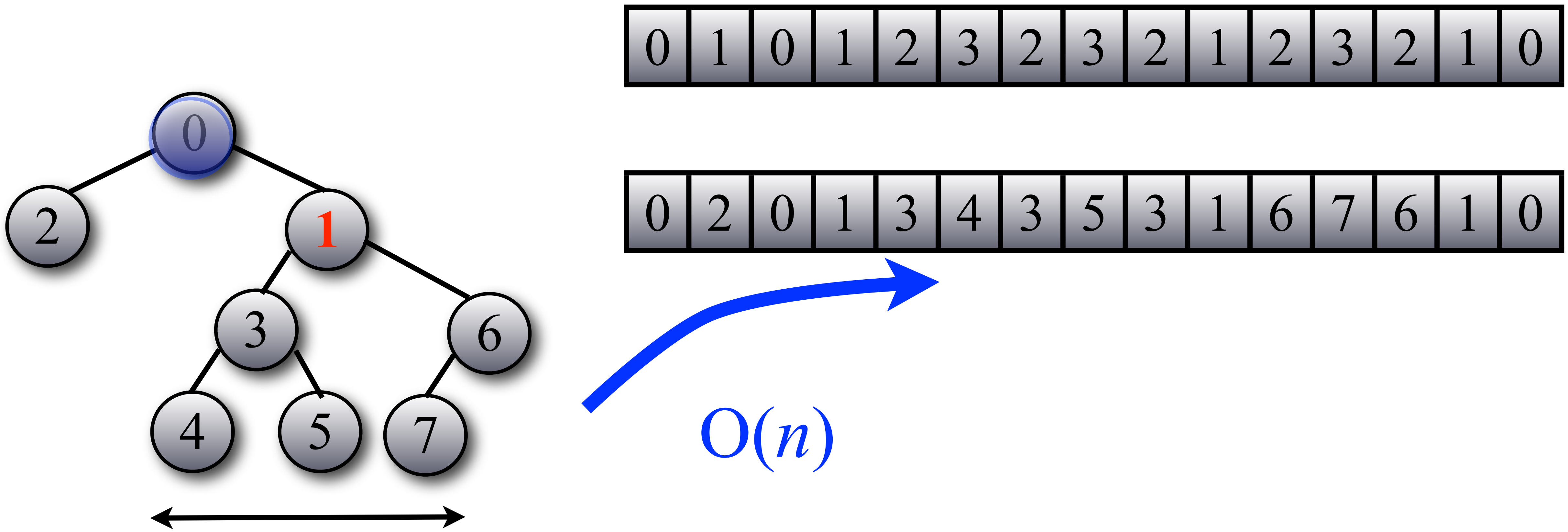
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

# RMQ
# & Cartesian Trees




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

[Schieber, Vishkin 1988]

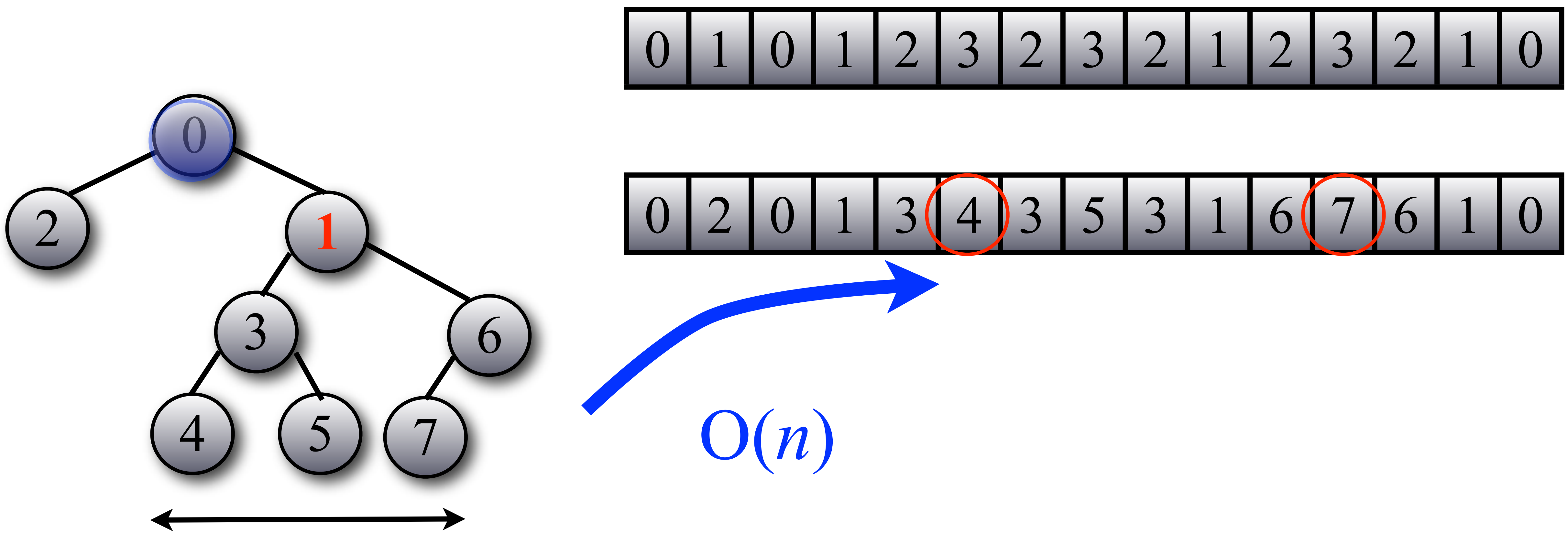
[Berkman, Vishkin 1993]

[Bender *et al.* 2005]

⋮

[Fischer, Heun 2006]

# RMQ
# & Cartesian Trees




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

[Schieber, Vishkin 1988]

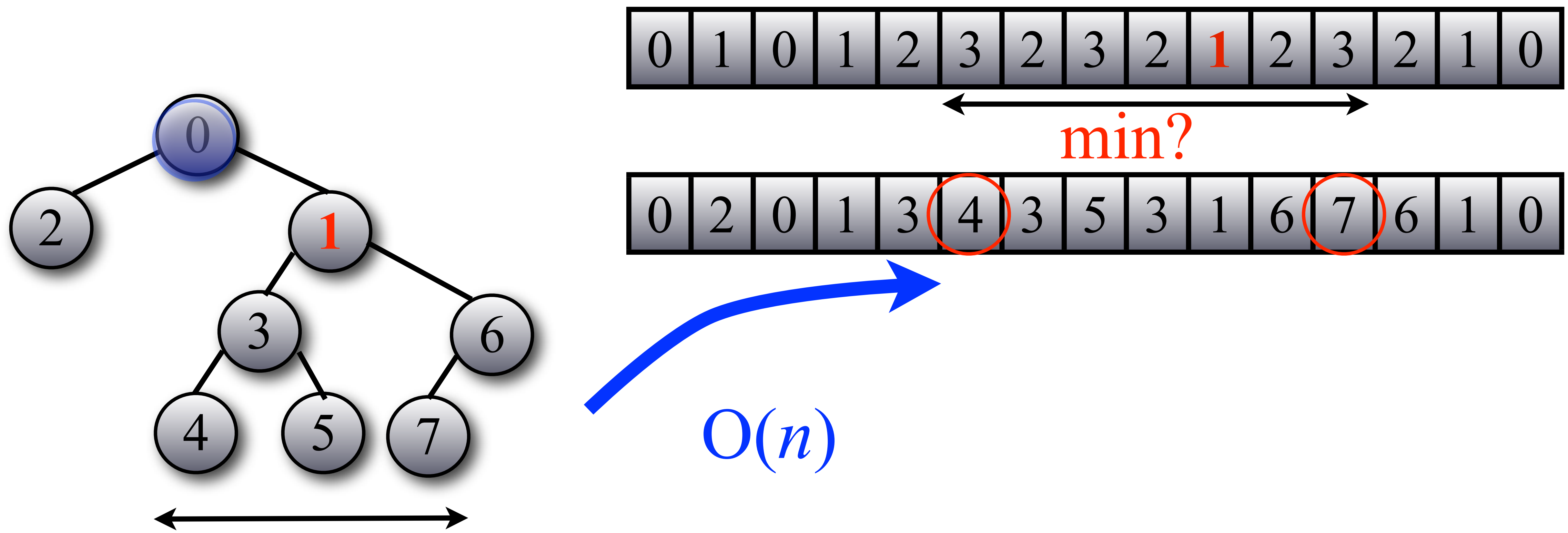
[Berkman, Vishkin 1993]

[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 |

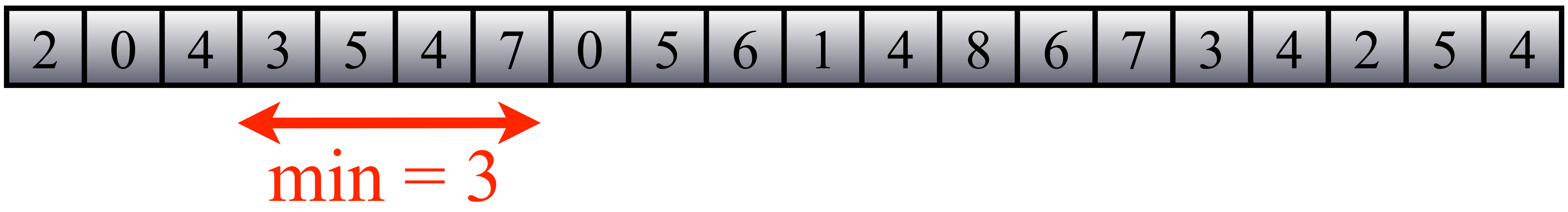
# RMQ

- 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 |

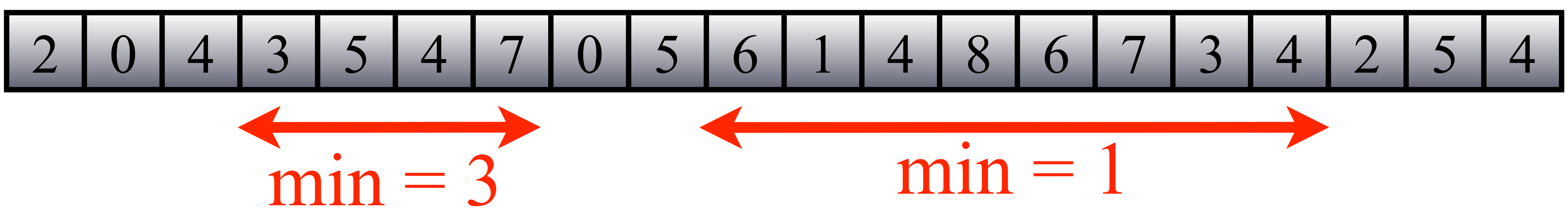
# RMQ

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



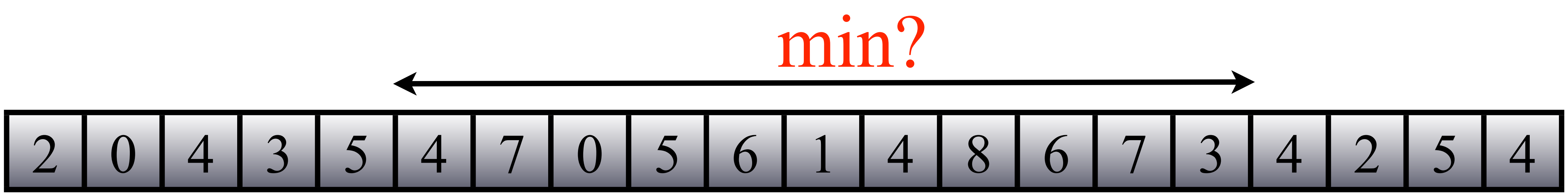
# RMQ

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



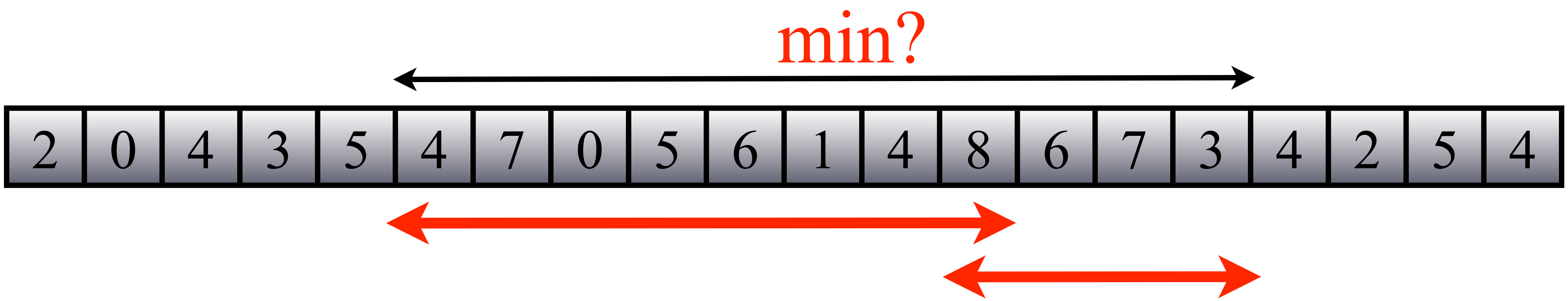
# RMQ

- 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

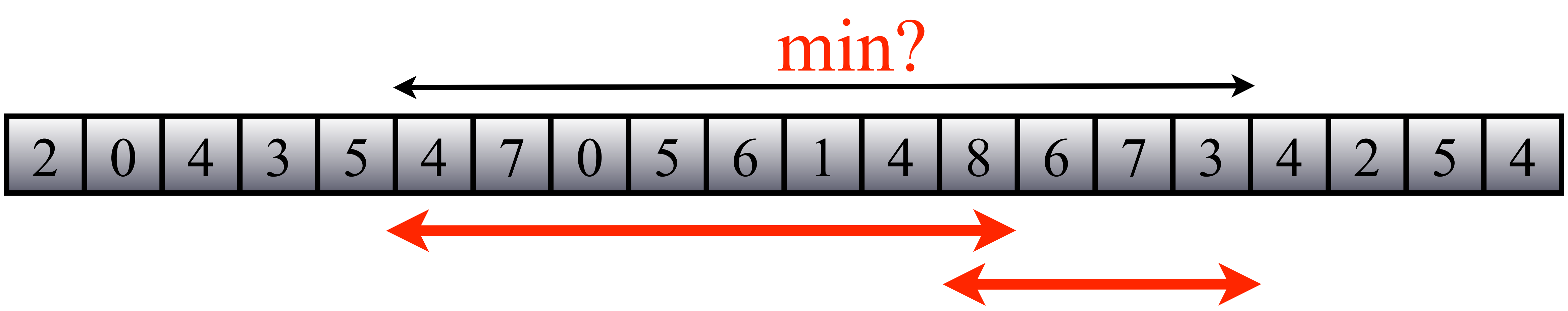
- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval $I$ s.t $|I|$ is a power of two
  - Query is composed of two overlapping intervals



# RMQ

How?

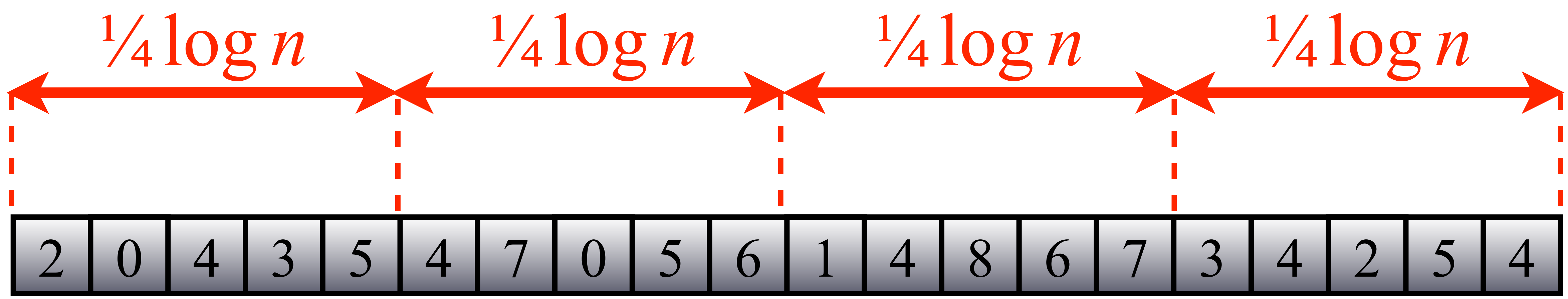
- Warmup: $O(n \log n)$ prep. $O(1)$ query:
  - Compute min of every interval $I$ s.t $|I|$ is a power of two
  - Query is composed of two overlapping intervals



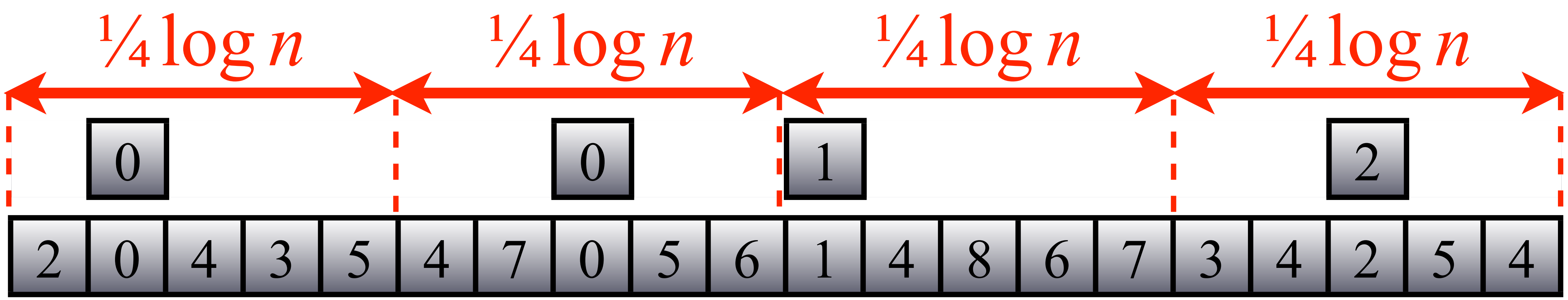
# RMQ

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

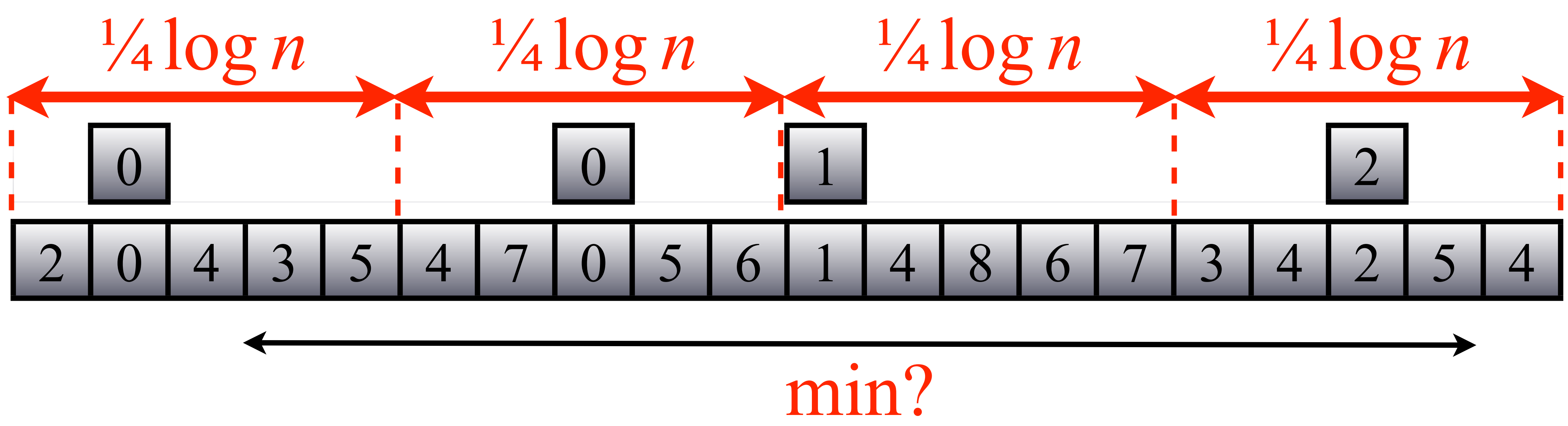
# RMQ



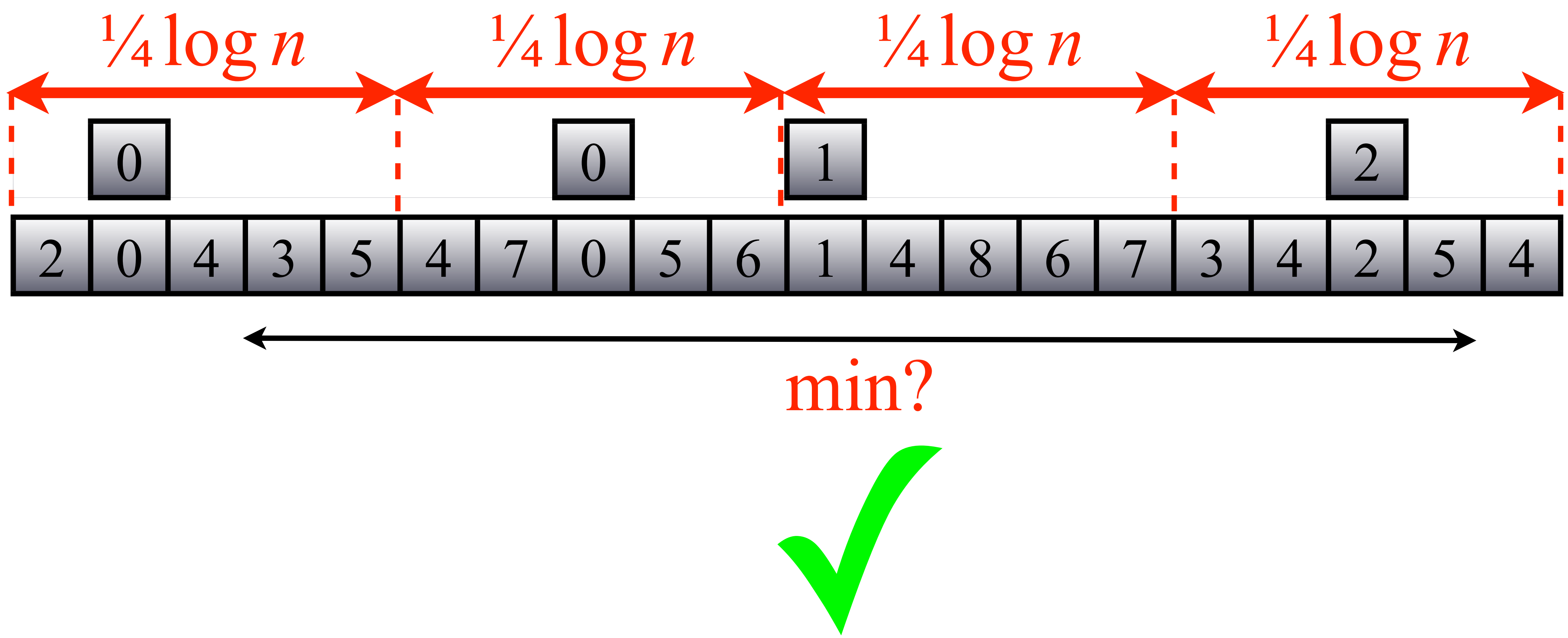
# RMQ



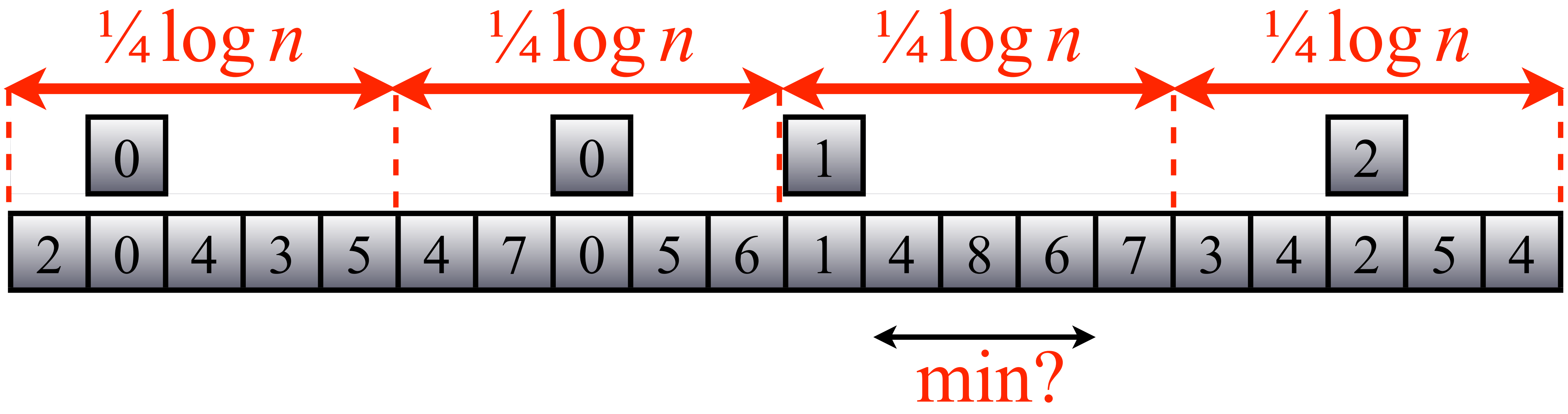
# RMQ



# RMQ

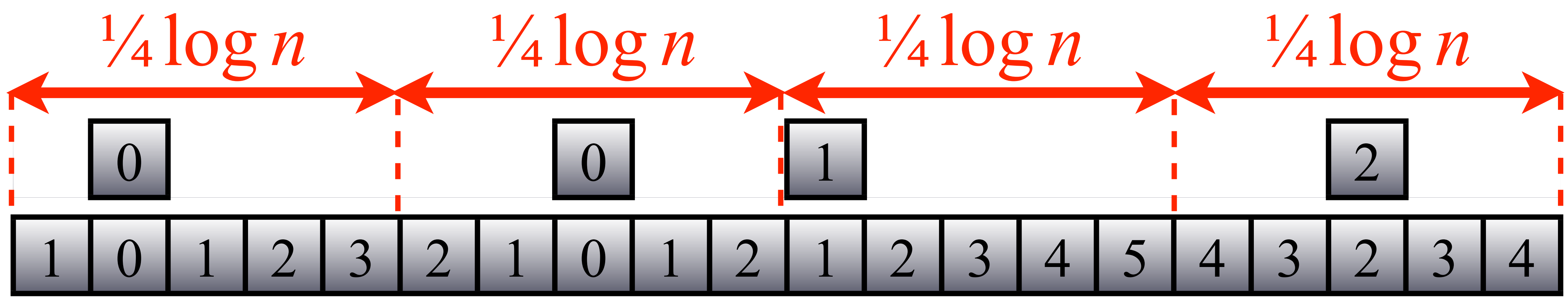


# RMQ



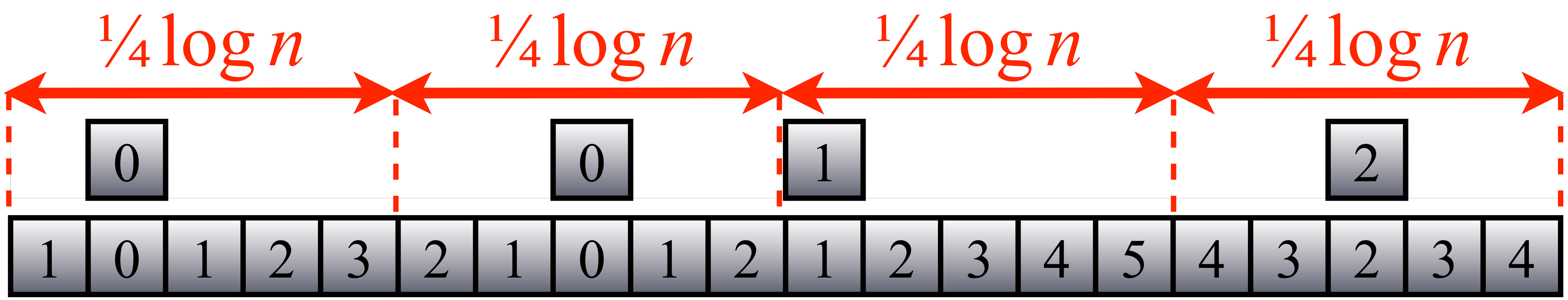
# RMQ ±1

- RMQ ➡ LCA ➡ RMQ ±1



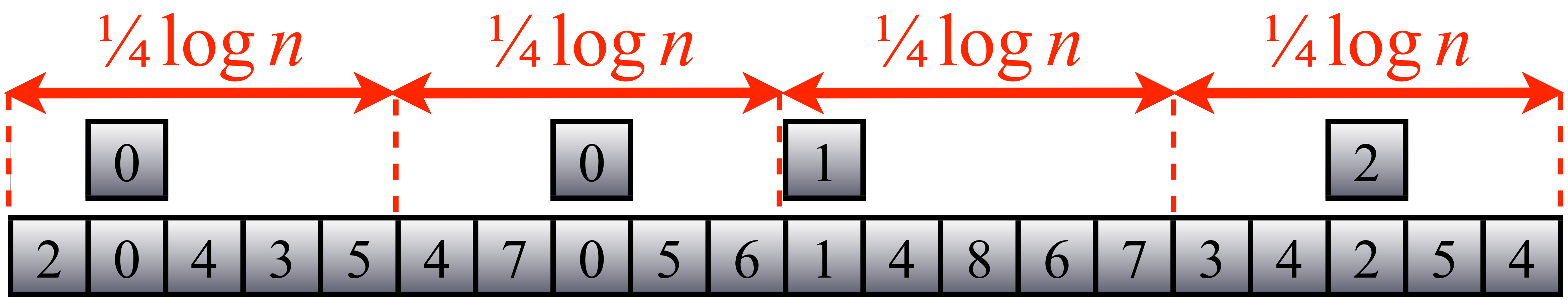
# RMQ ±1

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



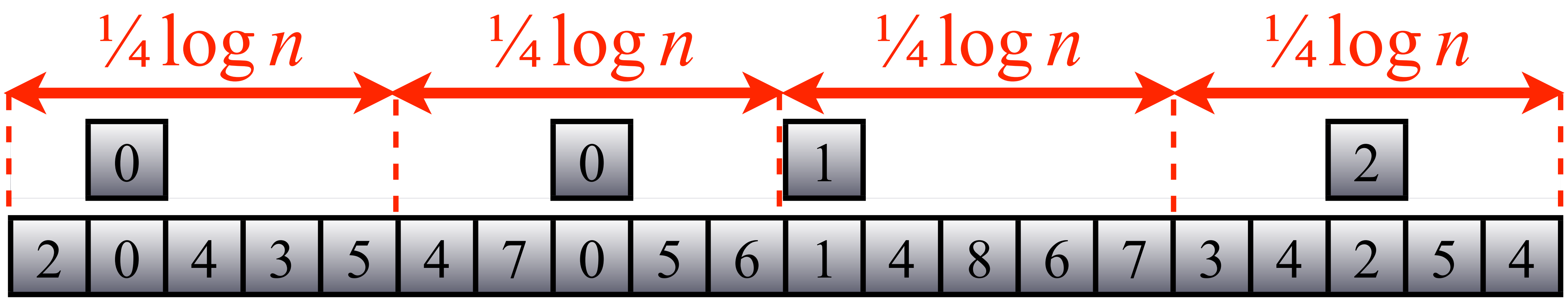
# Problems with RMQ ±1

- RMQ ➡ LCA ➡ RMQ ±1



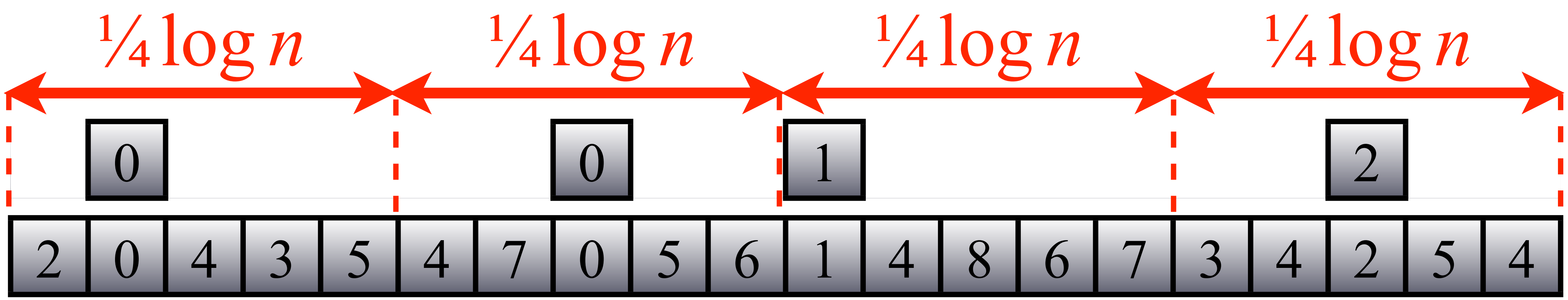
# Problems with RMQ ±1

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



# Problems with RMQ ±1

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

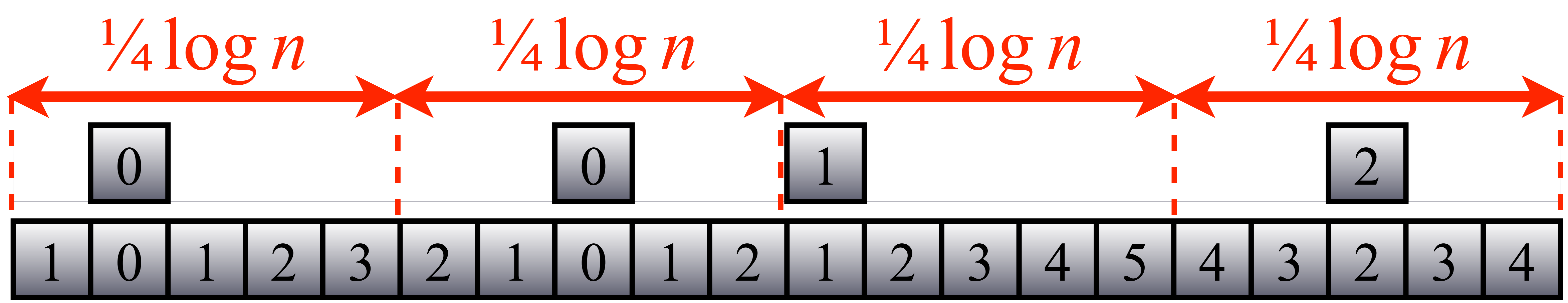



- # *different* Blocks = # different Cartesian trees = $4^{\frac{1}{4}\log n} = \sqrt{n}$

[Fischer, Heun 2006]

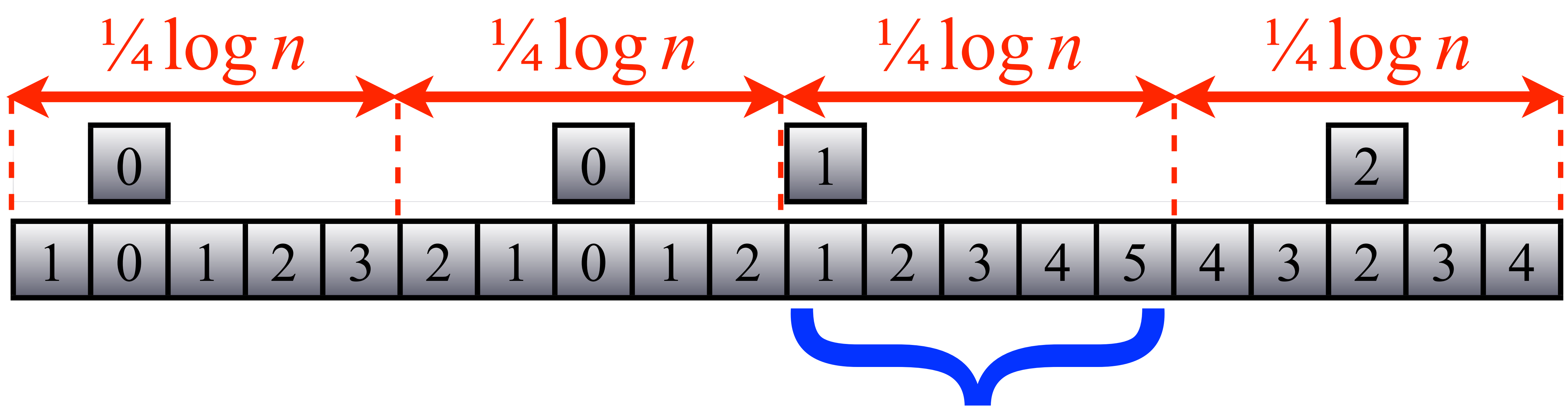
# Recursive Solution

- Use the "Warmup" solution on each block



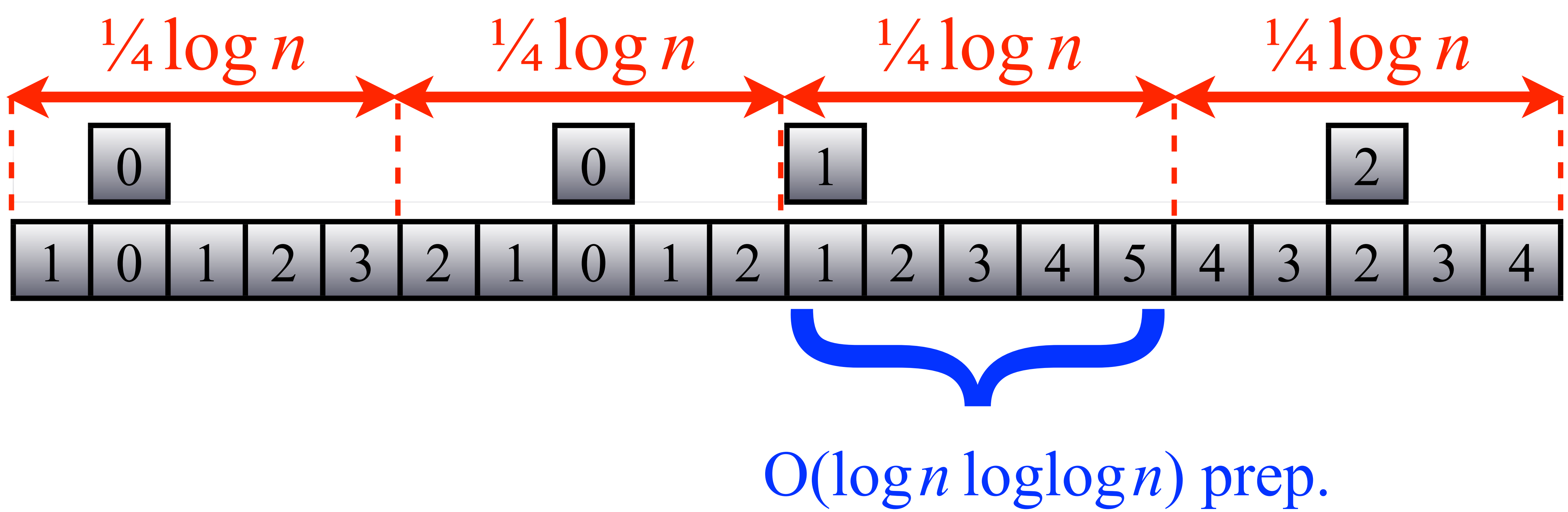
# Recursive Solution

- Use the “Warmup” solution on each block



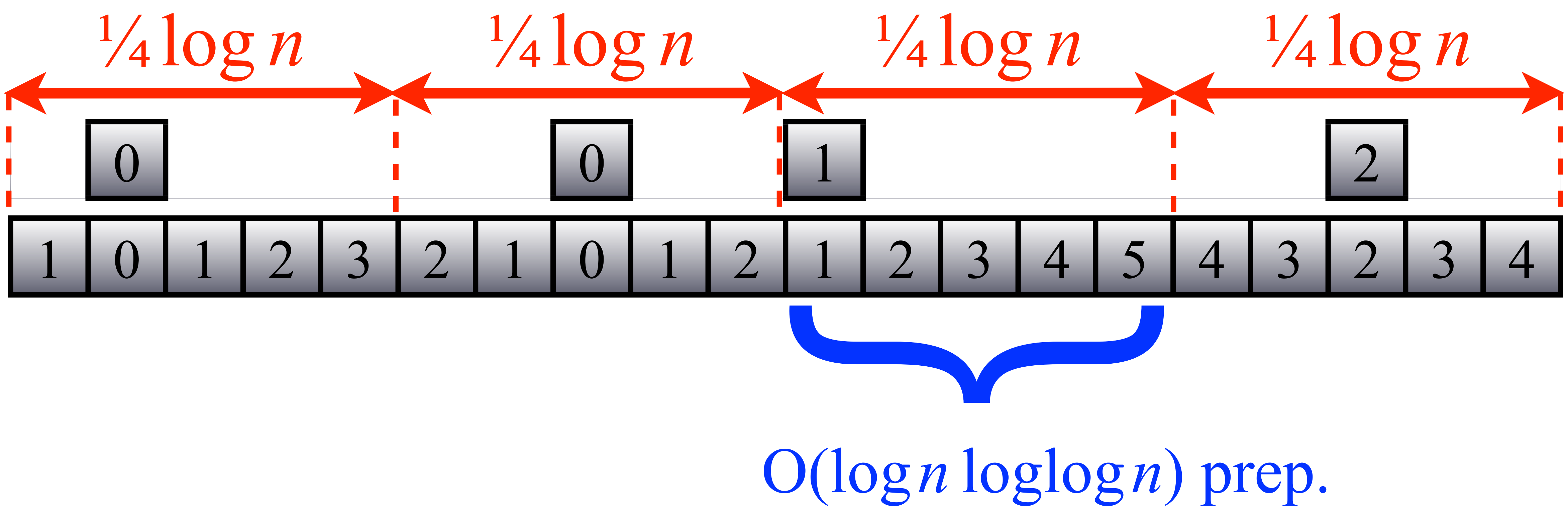
# Recursive Solution

- Use the "Warmup" solution on each block



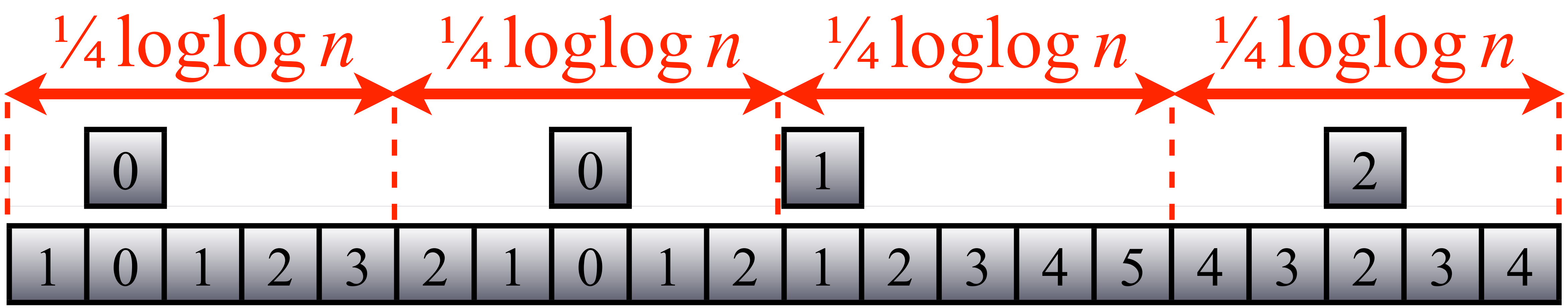
# Recursive Solution

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



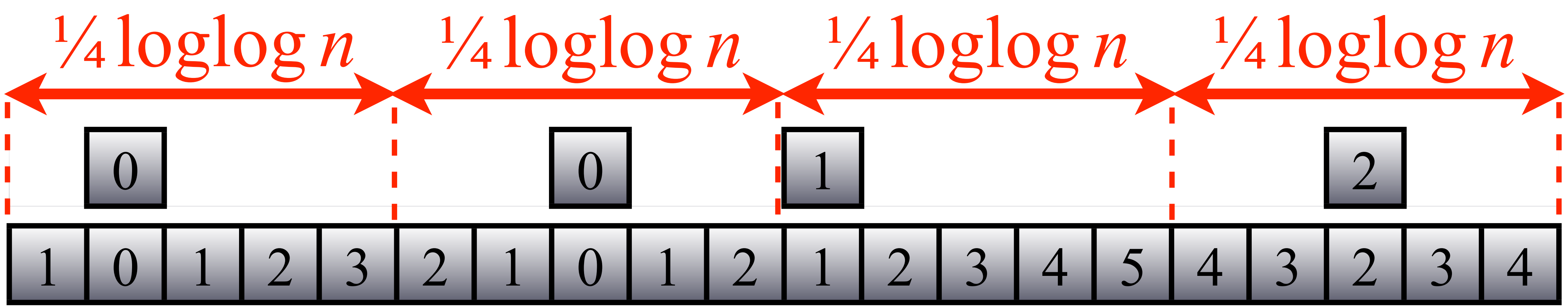
# Recursive Solution

- Use the "Warmup" solution on each block
  - $n + n \log\log n$ prep. O(1) query
  - $2n + n \log\log\log\log n$ prep. O(1) query



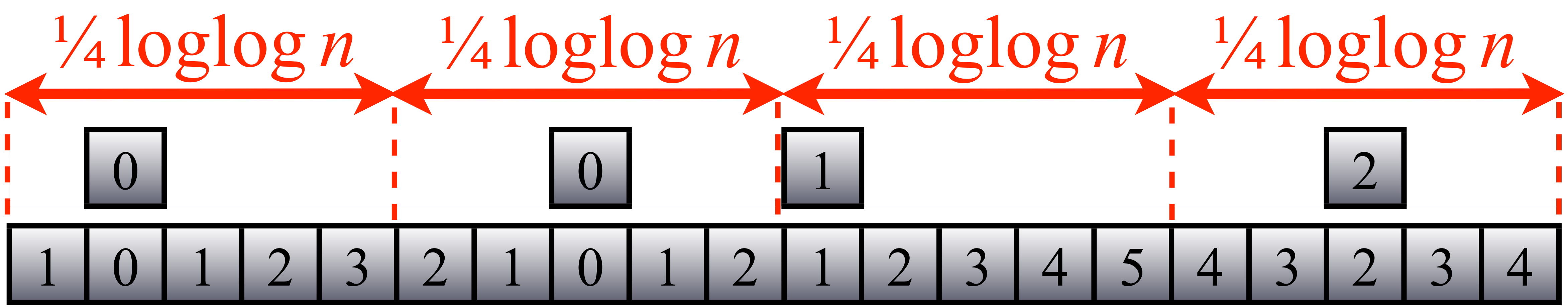
# Recursive Solution

- Use the "Warmup" solution on each block
  - $n + n \log\log n$ prep. O(1) query
  - $2n + n \log\log\log\log n$ prep. O(1) query
  - $n \log^* n$ prep. O(1) query (find in O(1) time which recursive level splits query)



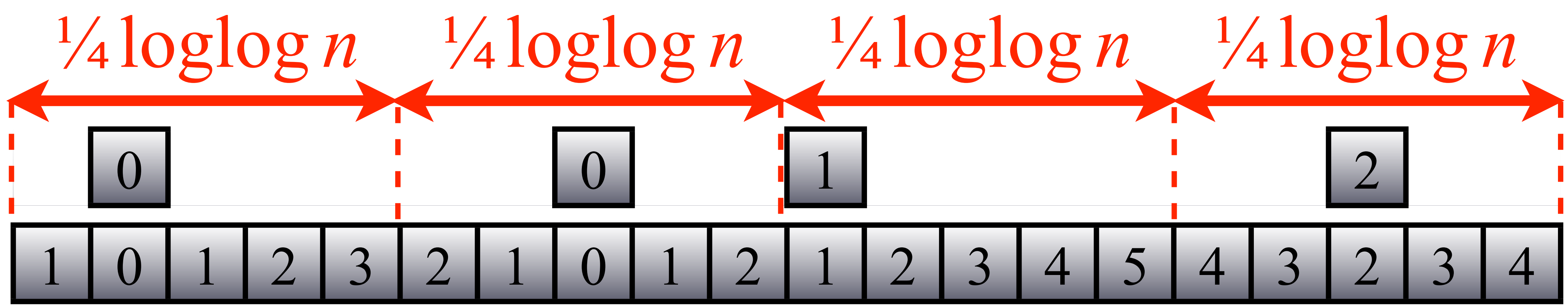
# Recursive Solution

- Use the "Warmup" solution on each block
  - $n + n \log\log n$ prep. O(1) query
  - $2n + n \log\log\log\log n$ prep. O(1) query
  - $n \log^* n$ prep. O(1) query (find in O(1) time which recursive level splits 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
  - $n + n \log\log n$ prep. O(1) query
  - $2n + n \log\log\log\log n$ prep. O(1) query
  - $n \log^* n$ prep. O(1) query (find in O(1) time which recursive level splits query)
  - $O(n\,\alpha_k(n))$ prep. O(k) query [Alon&Schieber 1987, Chazelle&Rosenberg 1989]




- Why?
  - ~~MIN~~ → any semiring operation (SUM is easy)
  - 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]

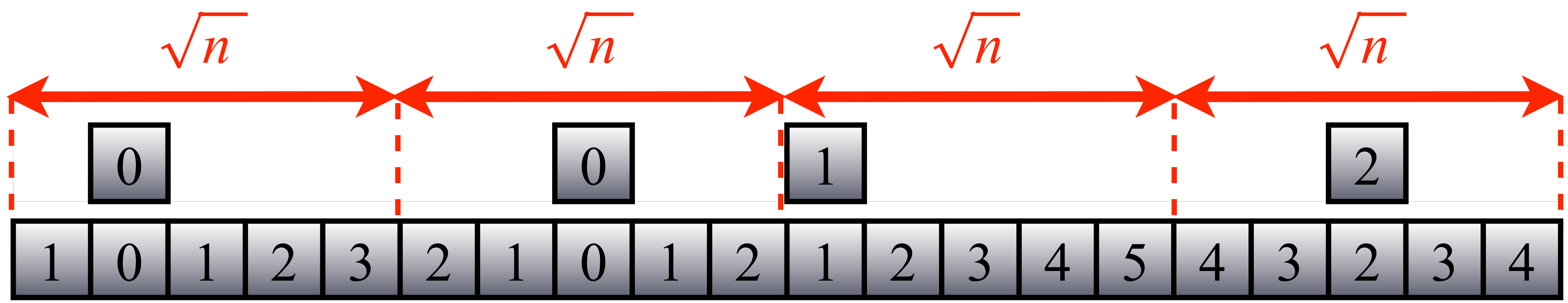
- Min of $n$ elements in O(1) time using $n^2$ processors [Valiant 1975]
  - O(1) RMQ using $n^4$ processors

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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]

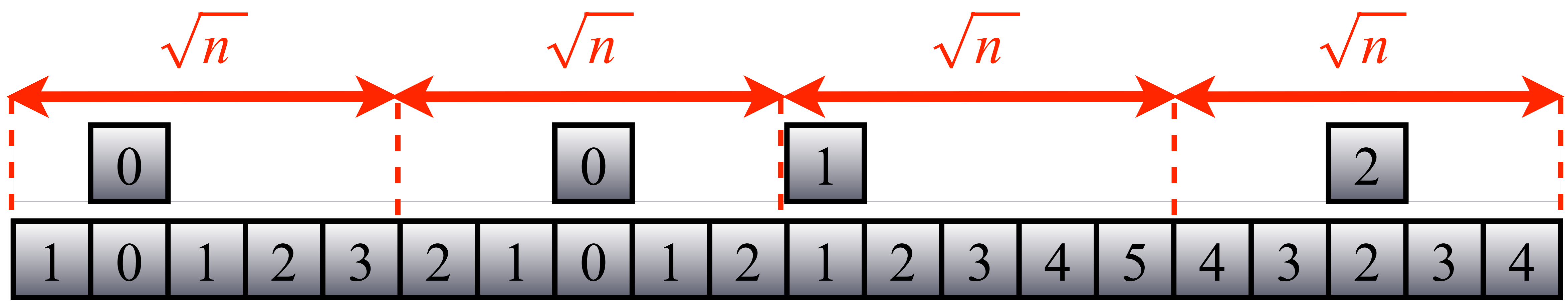
- Min of $n$ elements in O(1) time using $n^2$ processors [Valiant 1975]
  - O(1) RMQ using $n^4$ processors
  - O(1) RMQ using $n^{2.5}$ processors



# 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

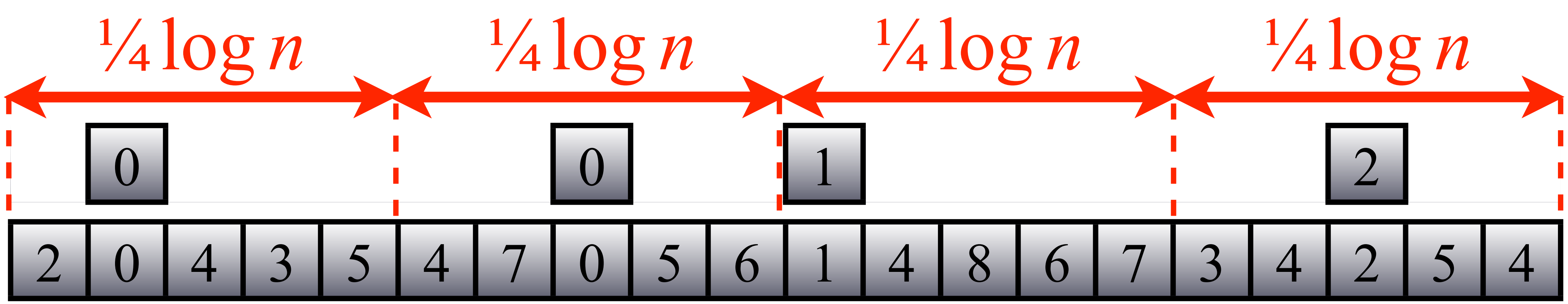


# Cache-Oblivious RMQ

[Demaine, Landau and W. 2009]

- An optimal RMQ solution that only makes sequential scans

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

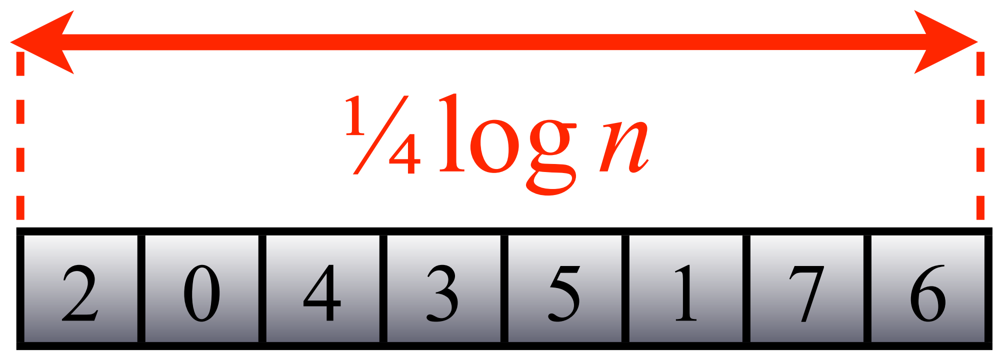



- # *different* Blocks = # different Cartesian trees = $4^{\frac{1}{4}\log n} = \sqrt{n}$

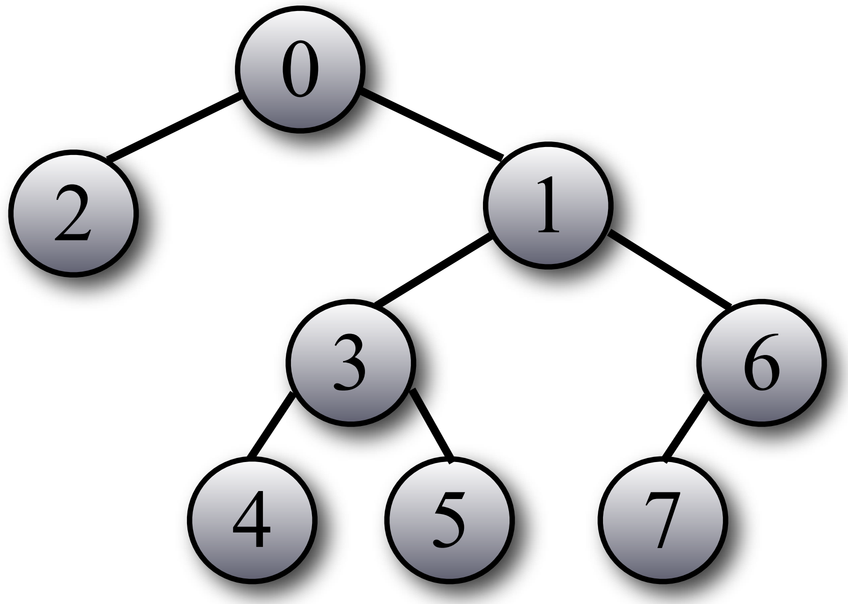
[Fischer, Heun 2006]

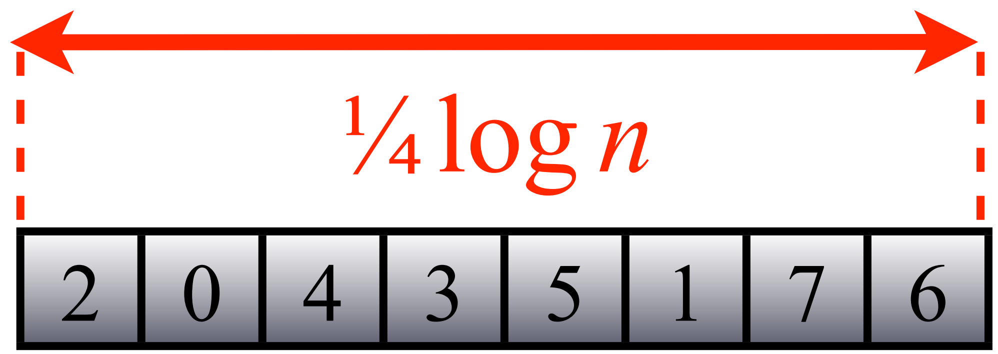
- Lookup table: index, construct

# A Cache-Oblivious Cartesian Tree

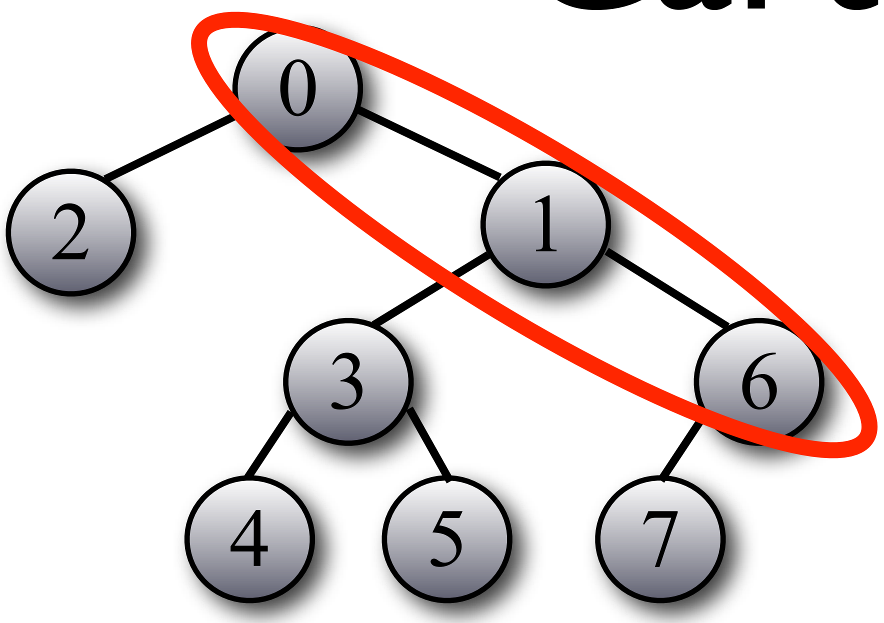


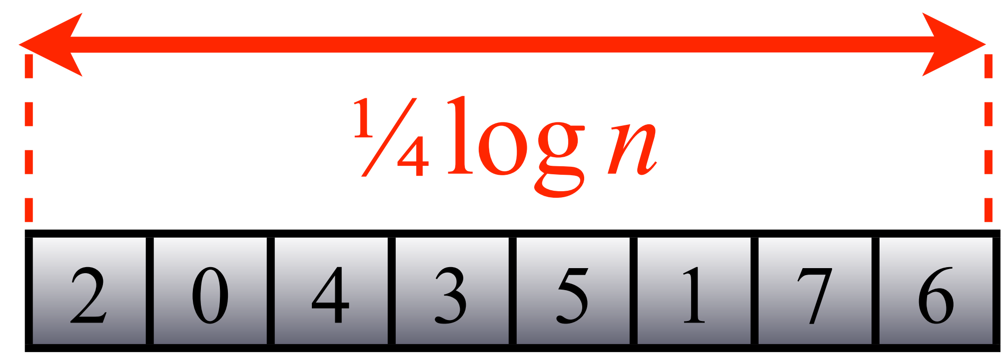
# A Cache-Oblivious Cartesian Tree



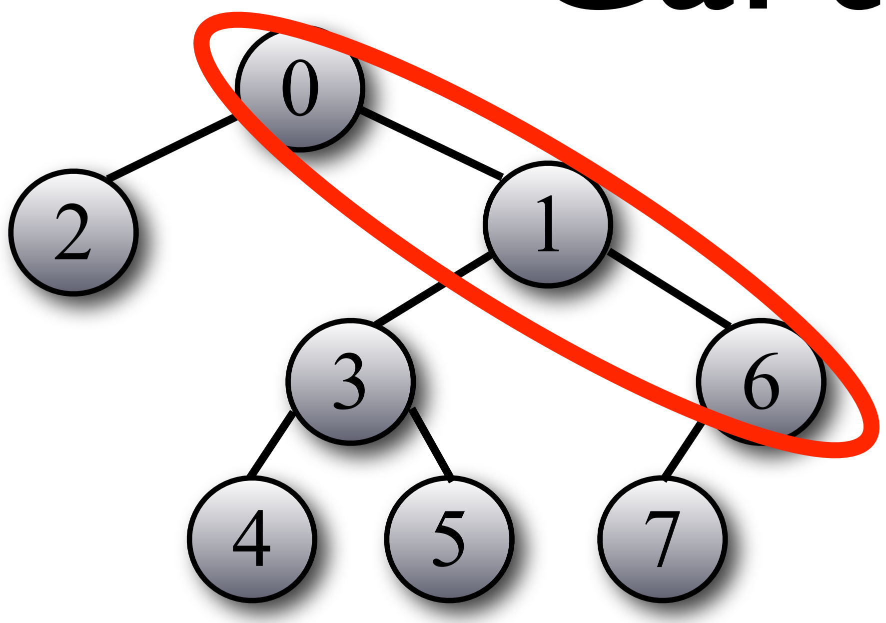
# A Cache-Oblivious Cartesian Tree

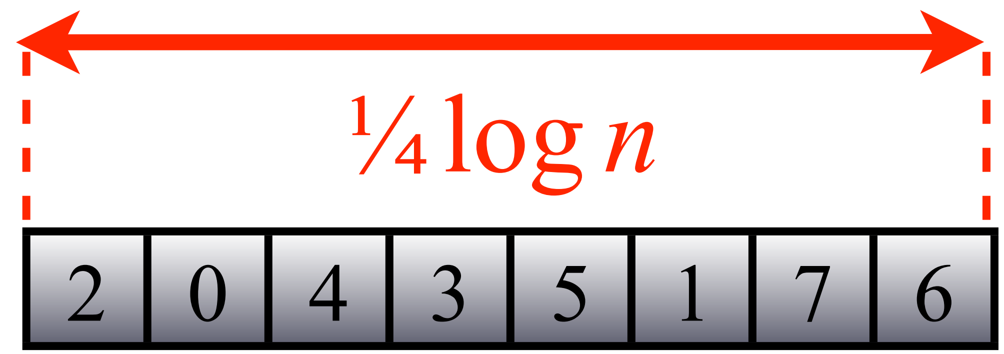



- cache-oblivious stack holds rightmost path

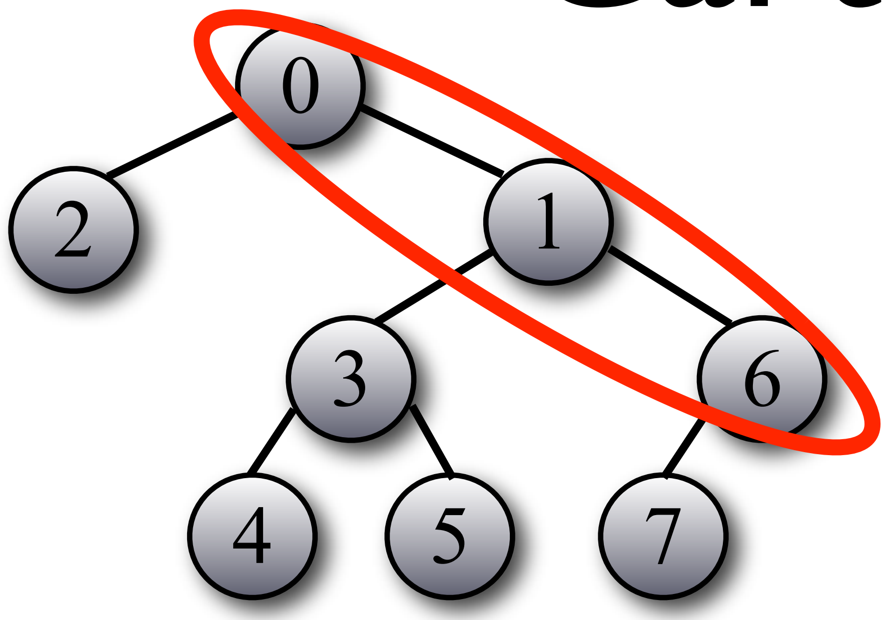
# A Cache-Oblivious Cartesian Tree

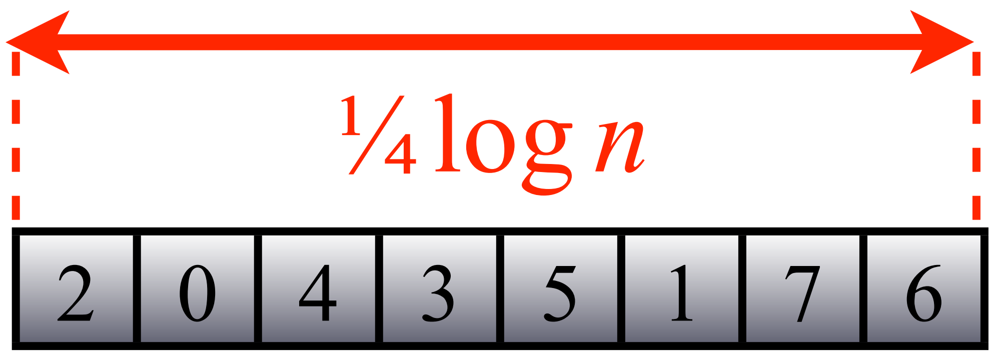



- cache-oblivious stack holds rightmost path
- when we climb (pop) $i$ vertices, concatenate $0\underbrace{111\cdots11}_{i}$

# A Cache-Oblivious Cartesian Tree

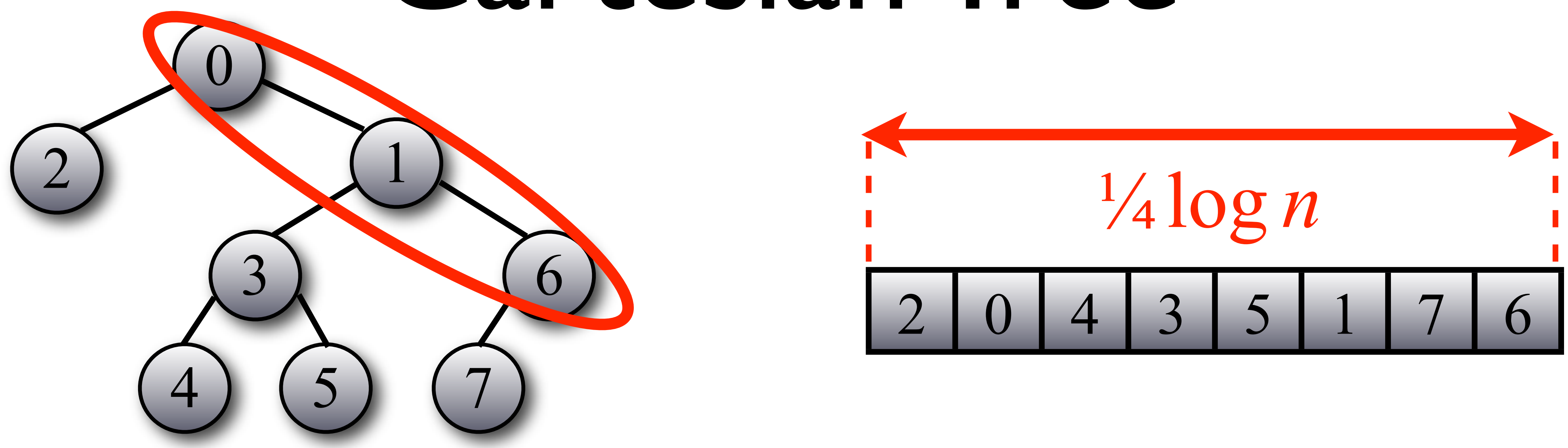



$$0\underbrace{11\cdots1}_{i_1}0\underbrace{11\cdots1}_{i_2}0\underbrace{11\cdots1}_{i_3}0\underbrace{11\cdots1}_{i_4}$$

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

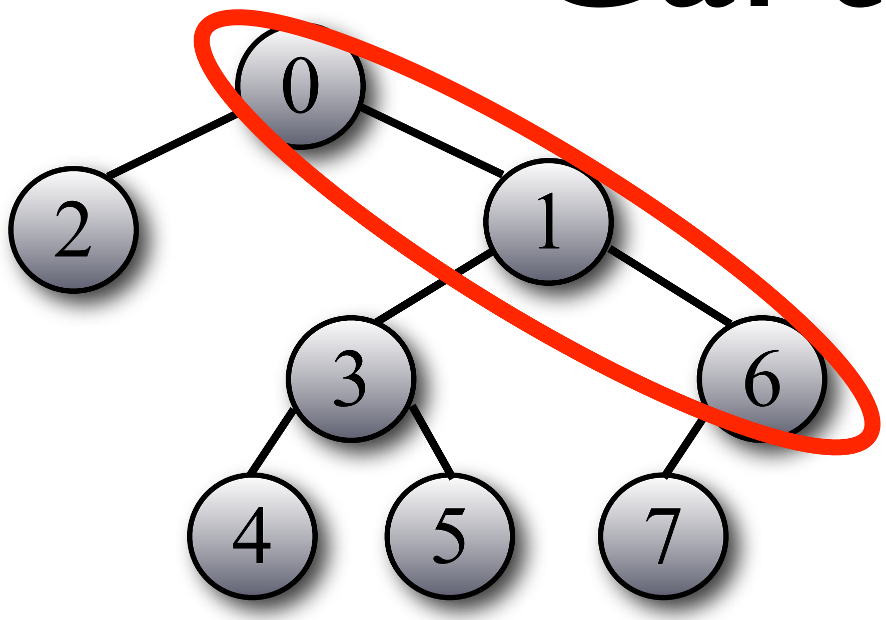
# A Cache-Oblivious Cartesian Tree

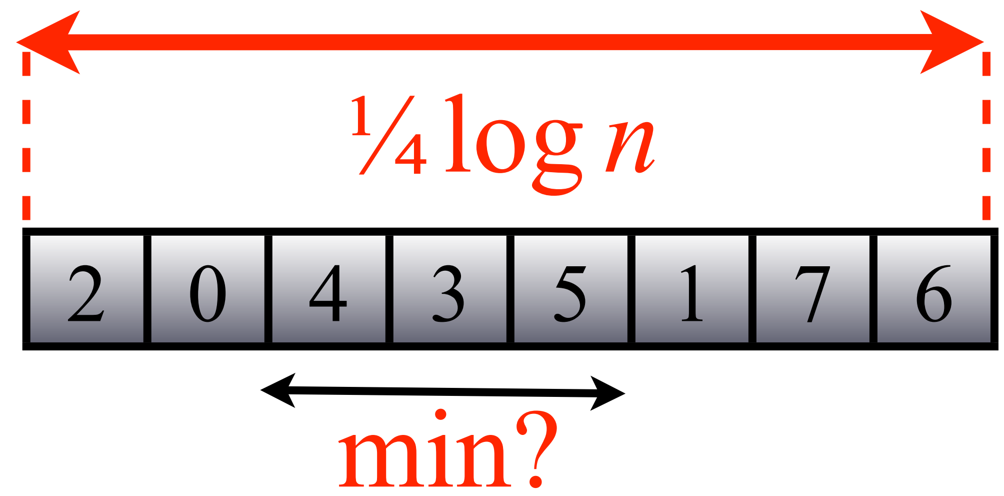



$$0\underbrace{11\cdots1}_{i_1}0\underbrace{11\cdots1}_{i_2}0\underbrace{11\cdots1}_{i_3}0\underbrace{11\cdots1}_{i_4} \in [\sqrt{n}]$$

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

# A Cache-Oblivious Cartesian Tree



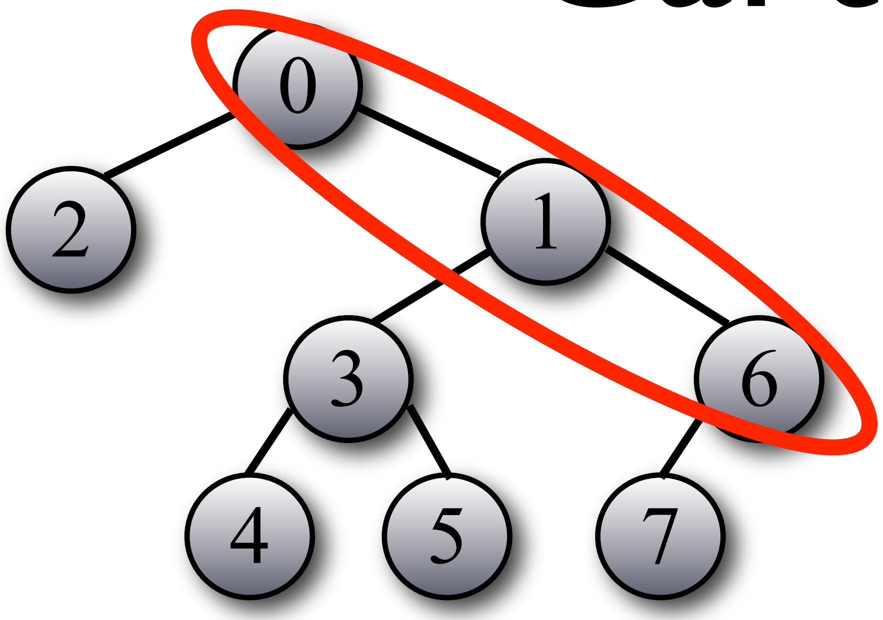



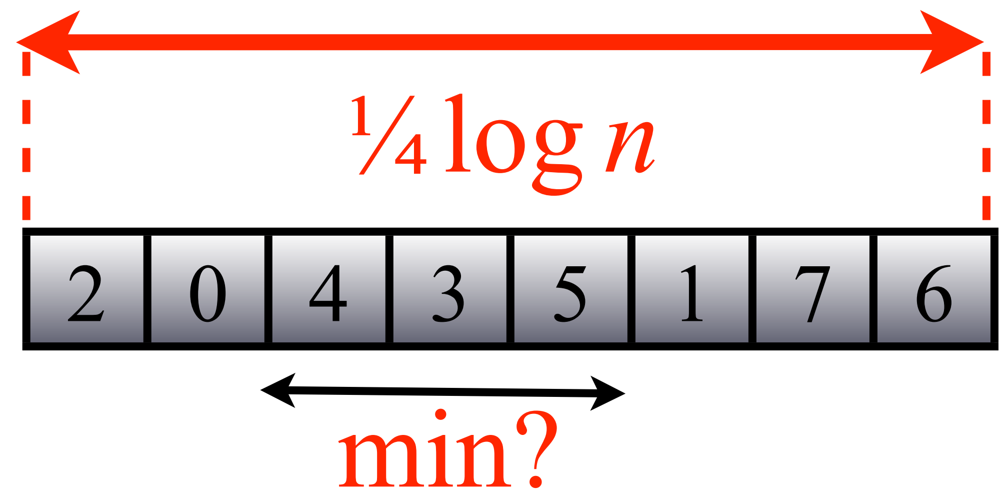

- ∀ binary string and ∀ query:

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

- 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: 2D Cartesian Tree

[Demaine, Landau and W. 2009]

# RMQ Generalization: 2D Cartesian Tree

- The *2D-RMQ* problem:

# RMQ Generalization: 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: 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: 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: 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: 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!