9 Range-Minimum Queries

27 April 2020

Sebastian Wild

version `2020-04-27 15:58`

Outline

9 Range-Minimum Queries

9.1 Introduction
9.2 RMQ, LCP, LCE, LCA — WTF?
9.3 Sparse Tables
9.4 Cartesian Trees
9.5 “Four Russians” Table

9.1 Introduction

Range-minimum queries (RMQ)

array/numbers don't change

- **Given:** Static array $A[0..n)$ of numbers
- **Goal:** Find minimum in a range;
  A known in advance and can be preprocessed

$\mathrm{RMQ}(7, 15) = 10$

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

- **Nitpicks:**
  - Report *index* of minimum, not its value
  - Report *leftmost* position in case of ties

Clicker Question

Given the array from the slides, what is $\mathrm{RMQ}_A(1,6)$ = 1

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

pingo.upb.de/622222

Rules of the Game

- comparison-based $\rightsquigarrow$ values don't matter, only relative order
- Two main quantities of interest:
  1. **Preprocessing time**: Running time $P(n)$ of the preprocessing step
  2. **Query time**: Running time $Q(n)$ of one query (using precomputed data)
- Write "$\langle P(n), Q(n) \rangle$ time solution" for short

prep- query

(also: space usage $\le P(n)$)

Clicker Question

What do you think, what running times can we achieve? For a $\langle P(n), Q(n) \rangle$ time solution, enter “`<P(n),Q(n)>`”.

`pingo.upb.de/622222`

9.2 RMQ, LCP, LCE, LCA — WTF?

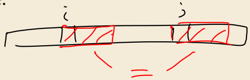
Recall Unit 6

Application 4: Longest Common Extensions

- We implicitly used a special case of a more general, versatile idea:

  The ***longest common extension (LCE)*** data structure:

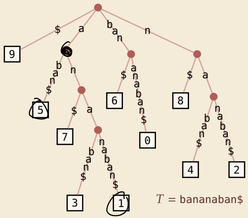
  - **Given:** String $T[0..n-1]$
  - **Goal:** Answer LCE queries, i. e.,
    given positions i, j in T,
    how far can we read the same text from there?
    formally: $\mathrm{LCE}(i,j) = \max\{\ell : T[i..i+\ell) = T[j..j+\ell)\}$



$\rightsquigarrow$ use suffix tree of T!

longest common prefix of ith and jth suffix

- In $\mathcal{T}$: $\mathrm{LCE}(i,j) \;=\; \mathrm{LCP}(T_i, T_j)$ $\rightsquigarrow$ same thing, different name!
  $=$ string depth of
  lowest common ancester (LCA) of
  leaves $\boxed{i}$ and $\boxed{j}$



- in short: $\boxed{\mathrm{LCE}(i,j) = \mathrm{LCP}(T_i, T_j) = \mathrm{stringDepth}\big(\mathrm{LCA}(\boxed{i}, \boxed{j})\big)}$

18

Recall Unit 6

Efficient LCA

How to find lowest common ancestors?

- Could walk up the tree to find LCA $\rightsquigarrow$ $\Theta(n)$ worst case
- Could store all LCAs in big table $\rightsquigarrow$ $\Theta(n^2)$ space and preprocessing

Amazing result: Can compute data structure in $\Theta(n)$ time and space that finds any LCA is **constant(!) time**.

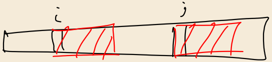
- a bit tricky to understand
- but a theoretical breakthrough
- and useful in practice

and suffix tree construction inside . . .

$\rightsquigarrow$ for now, use $O(1)$ LCA as black box.

$\rightsquigarrow$ After linear preprocessing (time & space), we can find LCEs in $O(1)$ time.

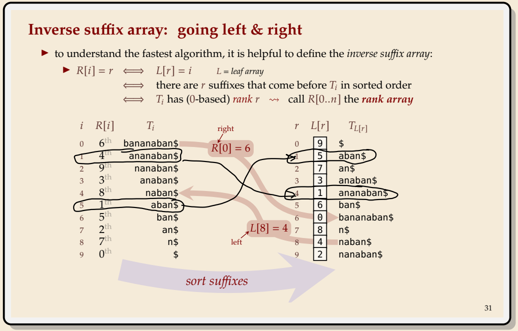
Finally: Longest common extensions

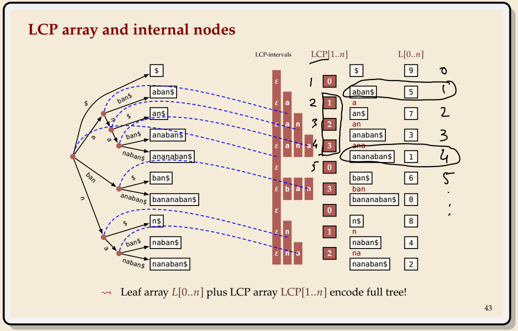


- In Unit 6: Left question open how to compute LCA in suffix trees
- But: Enhanced Suffix Array makes life easier!

$$\boxed{\mathrm{LCE}(i,j) \;=\; \mathrm{RMQ}_{\mathrm{LCP}}(R[i]+1, R[j])}$$

LCP[↓] = LCP[2] = 1 $\mathrm{RMQ}_{LCP}(2,4) = 2$





RMQ Implications for LCE

- Recall: Can compute (inverse) suffix array and LCP array in $O(n)$ time

$\rightsquigarrow$ A $\langle P(n), Q(n) \rangle$ time RMQ data structure implies a $\langle P(n), Q(n) \rangle$ time solution for longest-common extensions

9.3 Sparse Tables

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

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

- Two easy solutions show extreme ends of scale:

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

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

- Two easy solutions show extreme ends of scale:

1. **Scan on demand**
   - no preprocessing at all
   - answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

   $\rightsquigarrow \langle O(1), O(n) \rangle$

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

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

- Two easy solutions show extreme ends of scale:

1. **Scan on demand**

- no preprocessing at all
- answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

$\rightsquigarrow$ $\langle O(1), O(n) \rangle$

2. **Precompute all**

- Precompute all answers in a big 2D array $M[0..n)[0..n)$
- queries simple: $\mathrm{RMQ}(i, j) = M[i][j]$

$\rightsquigarrow$ $\langle O(n^3), O(1) \rangle$

$0 \le i < n$

$i \le j < n$

$\Theta(n^2)$ entries

Trivial Solutions

$\mathrm{RMQ}(7, 15) = 10$

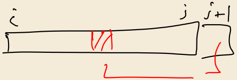
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	4	7	10	5	6	3	11	2	2	3	6	10	9	13	4	6	16	10

- Two easy solutions show extreme ends of scale:

1. **Scan on demand**
   - no preprocessing at all
   - answer $\mathrm{RMQ}(i, j)$ by scanning through $A[i..j]$, keeping track of min

   $\rightsquigarrow \langle O(1), O(n) \rangle$

2. **Precompute all**



   - Precompute all answers in a big 2D array $M[0..n)[0..n)$
   - queries simple: $\mathrm{RMQ}(i, j) = M[i][j]$

   $\rightsquigarrow \langle O(n^3), O(1) \rangle$

   - Preprocessing can reuse partial results $\rightsquigarrow \langle \underline{O(n^2)}, O(1) \rangle$

Sparse Table

- **Idea:** Like "precompute-all", but keep only some entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k. $0 \le i < n$

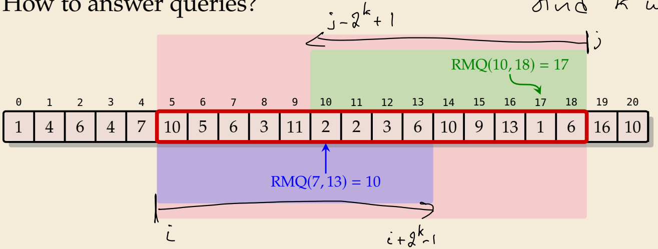
  $\rightsquigarrow \le n \cdot \underline{\lg n}$ entries ↪ store $M[i][k]$

Sparse Table

- **Idea:** Like "precompute-all", but keep only some entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k.
  $\rightsquigarrow \ \le n \cdot \lg n$ entries

$\ell = j - i + 1$: Can always find k with: $\frac{\ell}{2} \le 2^k \le \ell$

- How to answer queries?



$\rightsquigarrow$ RMQ(i,j)

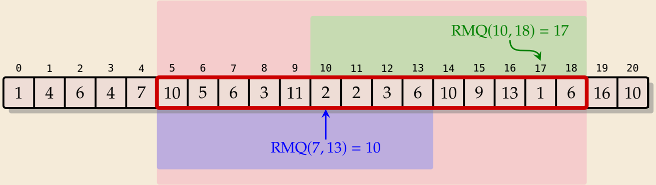
$= \arg\min \{ A[rmq], A[rmq] \}$

$rmq = RMQ(i, i+2^k-1)$

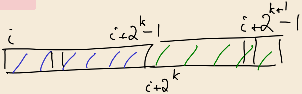
$rmq = RMQ(j-2^k+1, j)$

Sparse Table

- **Idea:** Like "precompute-all", but keep only some entries
- store $M[i][j]$ iff $\ell = j - i + 1$ is 2^k.
  $\rightsquigarrow \le n \cdot \lg n$ entries
- How to answer queries?



- Preprocessing can be done in $O(n \log n)$ times



$\rightsquigarrow \langle O(n \log n), O(1) \rangle$ time solution!

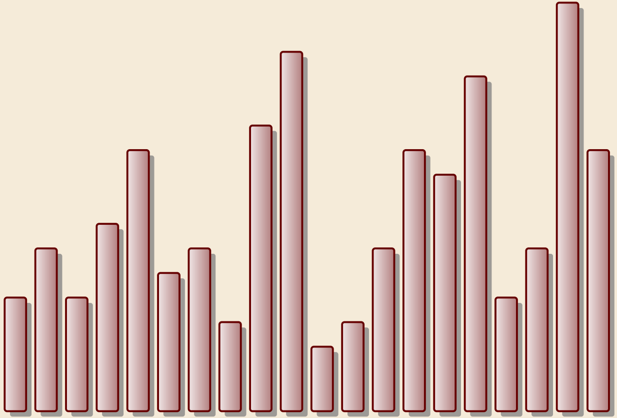
eventually $\langle O(n), O(1) \rangle$

9.4 Cartesian Trees

Range-maximum queries

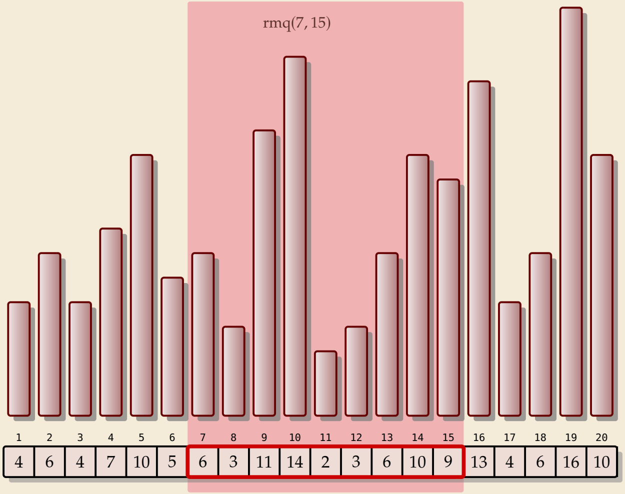
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

Range-maximum queries



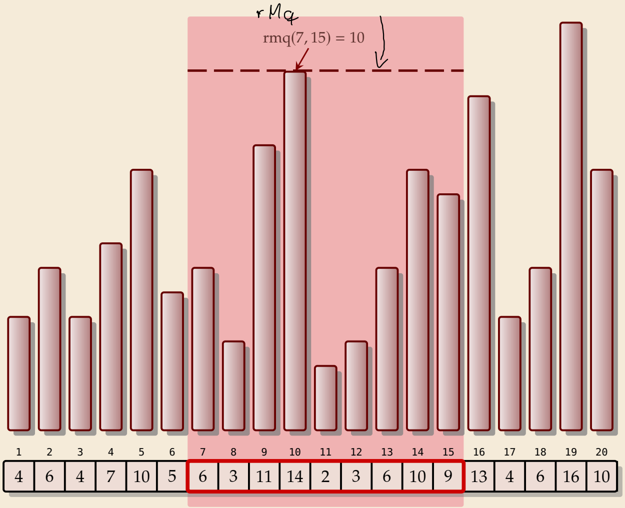
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	6	4	7	10	5	6	3	11	14	2	3	6	10	9	13	4	6	16	10

Range-maximum queries



- **R**ange-**m**ax **q**ueries on array A:
  $$\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$$
  $$= \textit{index} \text{ of max}$$

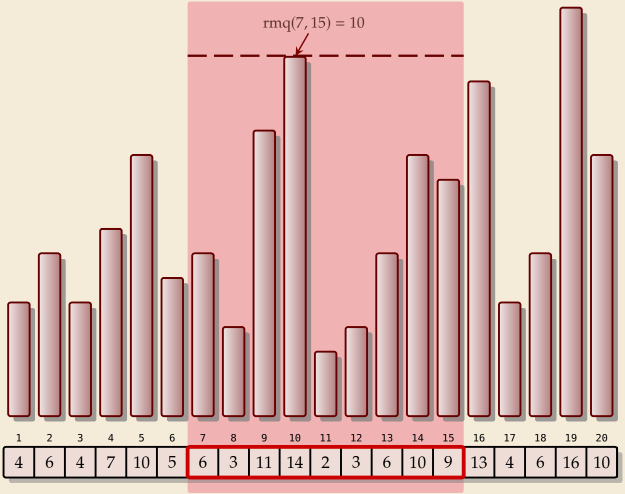
Range-maximum queries



- **<u>R</u>ange-<u>m</u>ax <u>q</u>ueries** on array A:

$$\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$$
$$= \textit{index} \text{ of max}$$

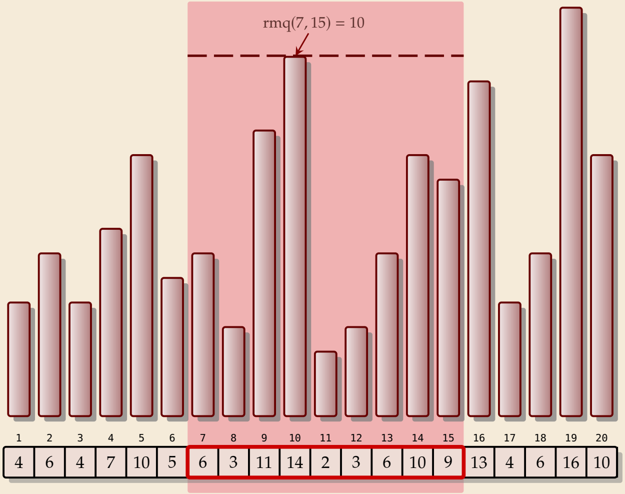
Range-maximum queries



- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i,j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max

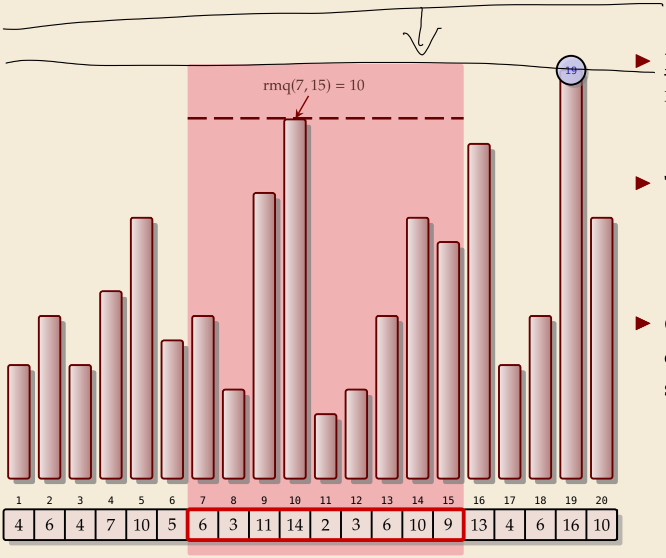
- **Task:** Preprocess A,
  then answer RMQs fast

Range-maximum queries



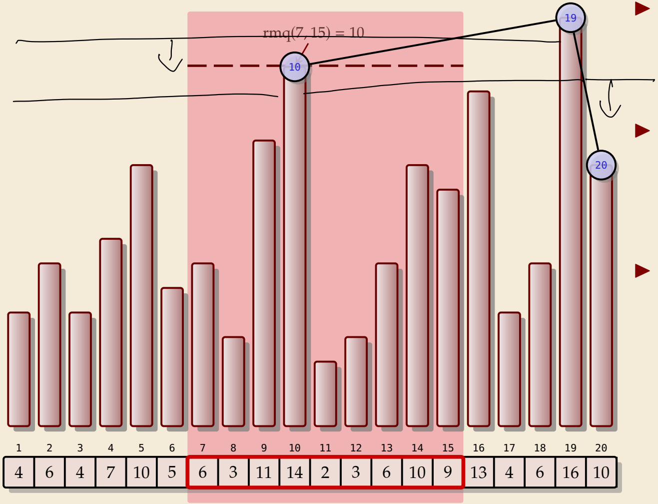
- **Range-max queries** on array A:
  $$\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$$
  $$= \textit{index} \text{ of max}$$
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!

Range-maximum queries



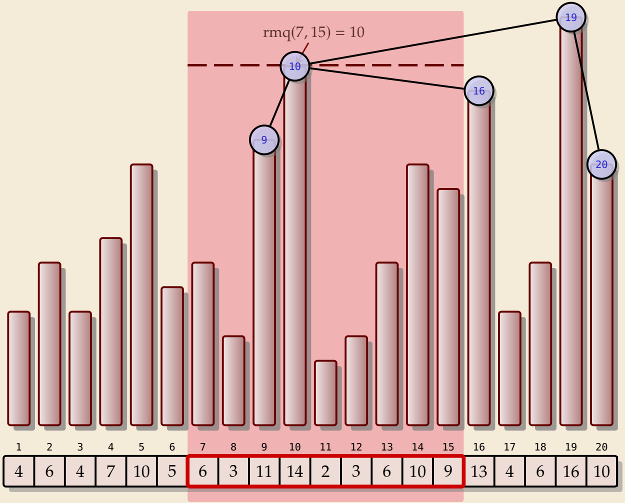
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

Range-maximum queries



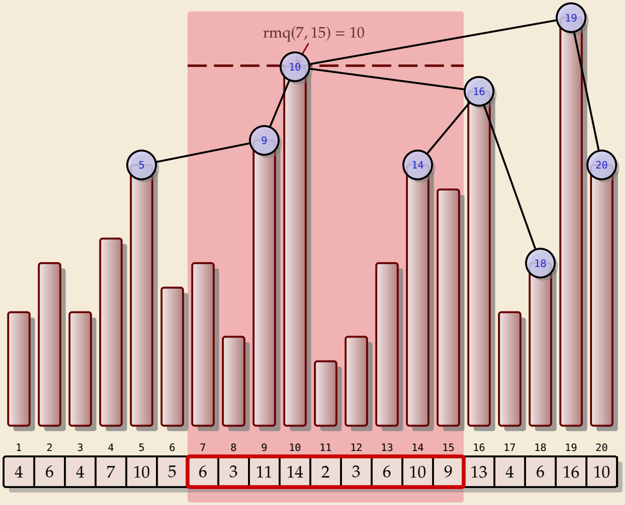
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

Range-maximum queries



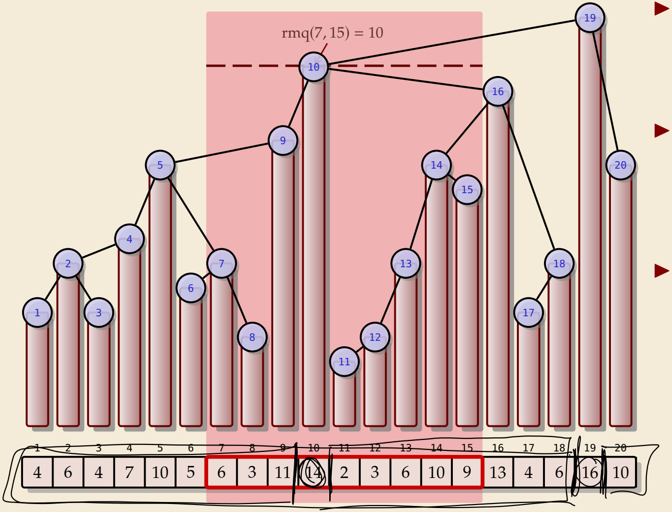
- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

Range-maximum queries



- **Range-max queries** on array A:
  $\mathrm{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A,
  then answer RMQs fast
  ideally constant time!
- **Cartesian tree:** (cf. *treap*)
  construct binary tree by
  sweeping line down

Range-maximum queries



- **R**ange-**m**ax **q**ueries on array A:

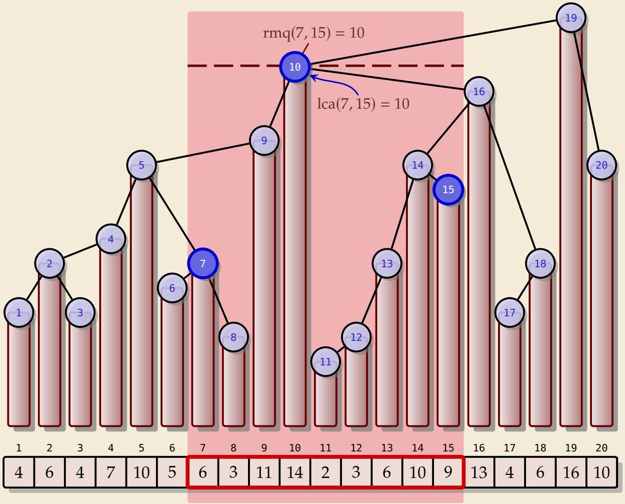
  $\text{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$

  $= \textit{index}$ of max

- **Task:** Preprocess A, then answer RMQs fast ideally constant time!

- **Cartesian tree:** (cf. *treap*) construct binary tree by sweeping line down

Range-maximum queries



- **Range-max queries** on array A:
  $\text{rmq}_A(i, j) = \arg\max_{i \le k \le j} A[k]$
  $= \textit{index}$ of max
- **Task:** Preprocess A, then answer RMQs fast ideally constant time!
- **Cartesian tree:** (cf. *treap*) construct binary tree by sweeping line down
- $\text{rmq}(i, j) =$ **lowest common ancestor** (LCA)