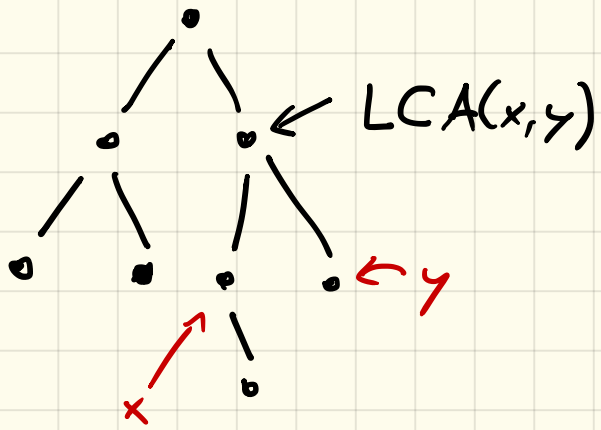
27.2.17 L2

Themes: - Static trees/arrays
- preprocessing

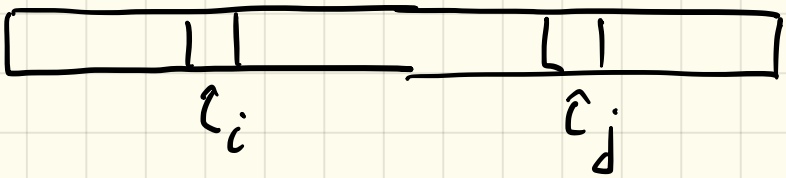


LCA(x,y) = lowest common anchestor of x,y

LA(x,k) = parent$^{(k)}$(x)

Goal Queries in $O(1)$

(with same preprocessing)



$RMQ(i,j) = \operatorname{argmin}\{x_i, x_{i+1}, \dots, x_j\}$

↕ range minimum query

$x_i \circ \cdots \circ x_j$
$\circ$ assoz.

can be done in $O(\log n)$ with range tree

Goal:

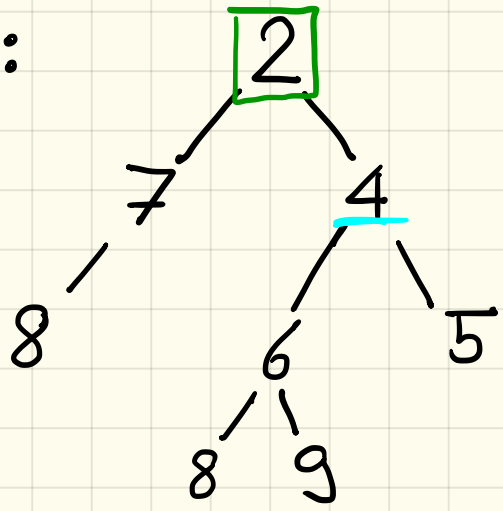
Preparation: $O(n)$ time/space ← RAM model!

Query: $O(1)$

1) Reduction RMQ → LCA

Consider $A = 8\ 7\ 2\ 8\ 6\ 9\ 4\ 5$

Build Cartesian tree T:



Observe $LCA(i,j)$ in tree $=$ $RMQ(i,j)$ in A

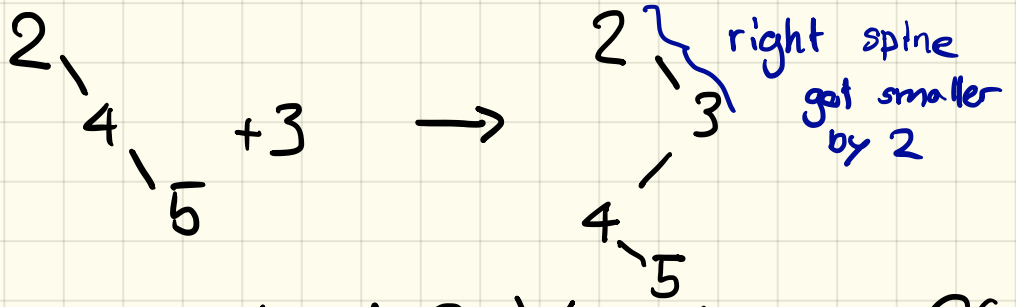
How to build the tree in $O(n)$? ($O(n^2)$ is easy)

Idea: Go through A from left to right and keep track of the right spine of T

Insert next element:

1) next element > previous ✓ $O(1)$

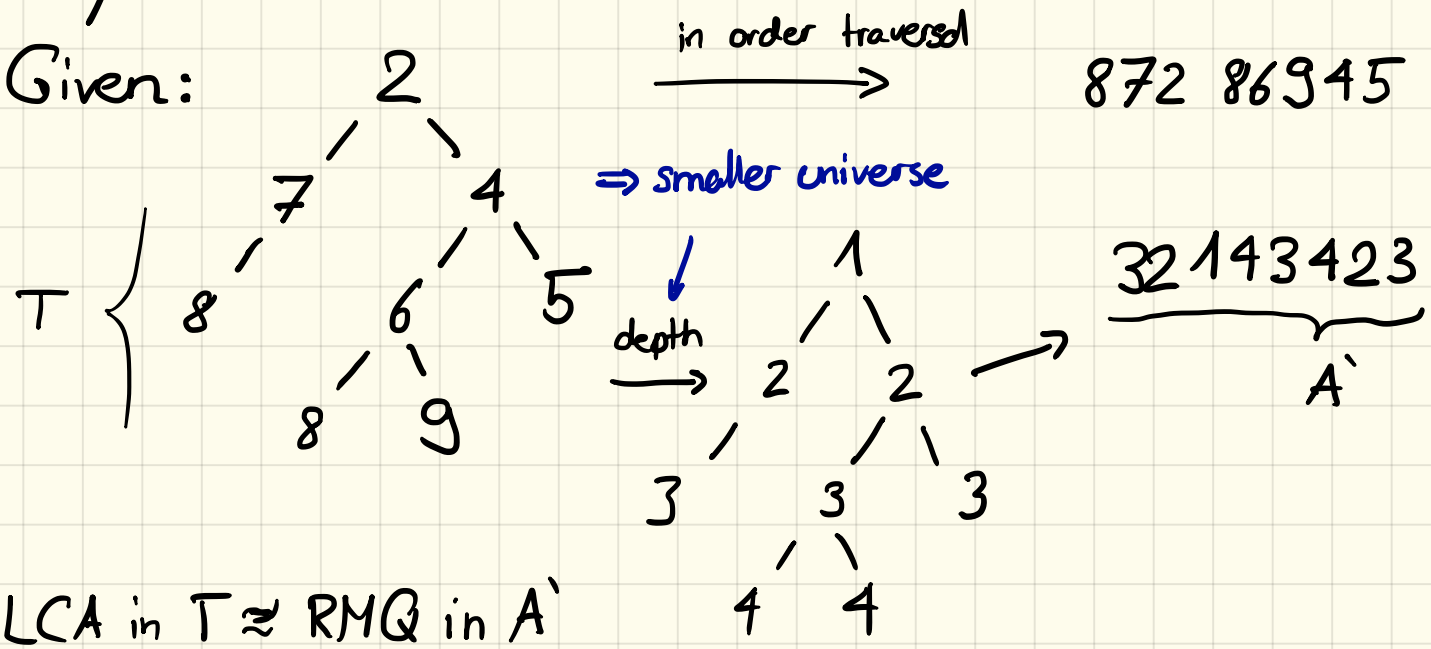
2) next element < previous



$\rightsquigarrow$ amortized $O(1)$/insert $\Rightarrow$ $O(n)$

2) LCA → RMQ

Given:

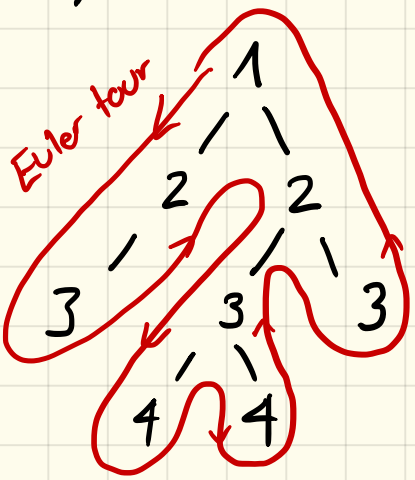


LCA in T ≈ RMQ in A`

So RMQ → LCA → RMQ

↕ U large ↑ U = [n]

1) tree → ±1 RMQ



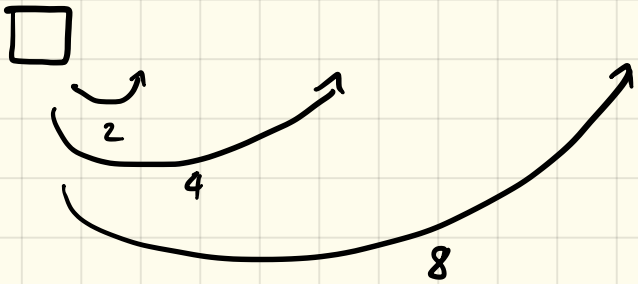
↕ i.e. $|A[i] - A[i+1]| = 1$

results in

A = 1 2 3 2 1 2 3 4 3 4 3 2 3 2 1

A contains all information about the tree and is $O(n)$.

2) RMQ in $O(1)$ time, but $O(n \log n)$ prep.



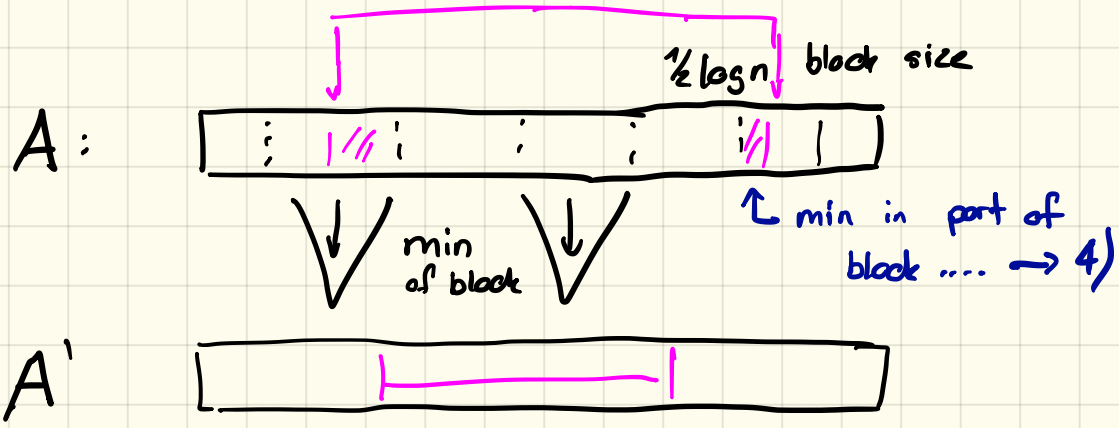
store argmin for ranges of size 2, 4, 8, ...

This works as min is idempotent.

So far: Good query time, but slow prep. time.

3) Indirection: For $n' = O(n/\log n)$ we would be done.

Solution:



4) block: [+1 -1 +1 -1] ↑ intial value

$2^{\frac{1}{2}\log n} = \sqrt{n}$ different blocks

$(\frac{1}{2}\log n)^2$ queries in a block

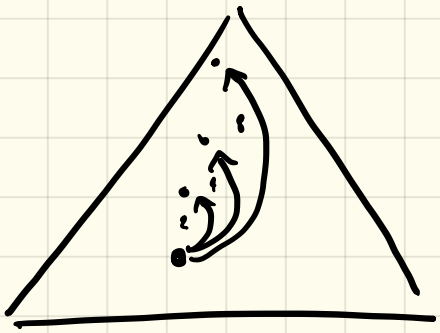
$\log\log n$ Answers

$\Rightarrow$ We need $O(\sqrt{n}\log^2 n \log\log n) = O(1)$ space to store this information.

$\Rightarrow$ So queries in a block $O(1)$

Level - Anchestry LA(x, k)

1) jump pointers: store all 2^i - parents

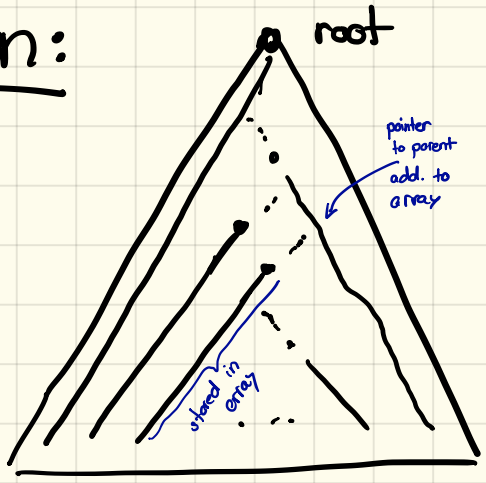


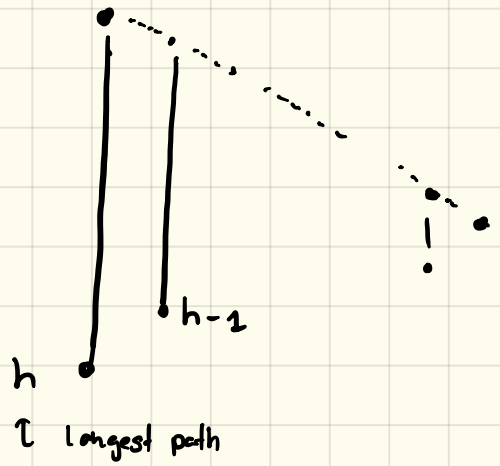
$O(n \log n)$ prep.

$O(\log n)$ query

2) long path decompasition:

store longest root leaf path in array.
↝ repeat recursively





You might need to follow $\Theta(\sqrt{n})$ path.

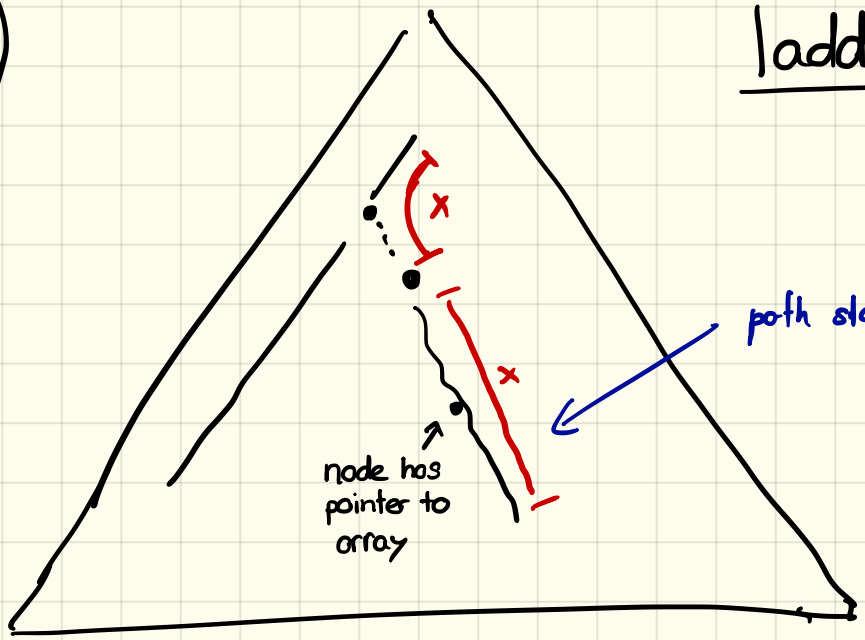
Worst case example

$O(n)$ space

$\Theta(\sqrt{n})$ query time

3)

ladder decomposition.



Idea: Extend paths, path of length x goes x up

Space: $O(n)$

node height h → path length $\geq h$

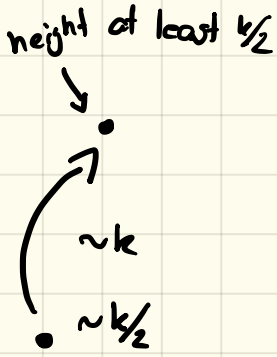
→ jump to $\geq 2h$

→ $O(\log n)$ jumps (query time)

4) Trick: 3 + 1

If you want to jump k-levels up:

Follow pointer $k/2 \leq 2^{\lfloor \lg k \rfloor} \leq k$



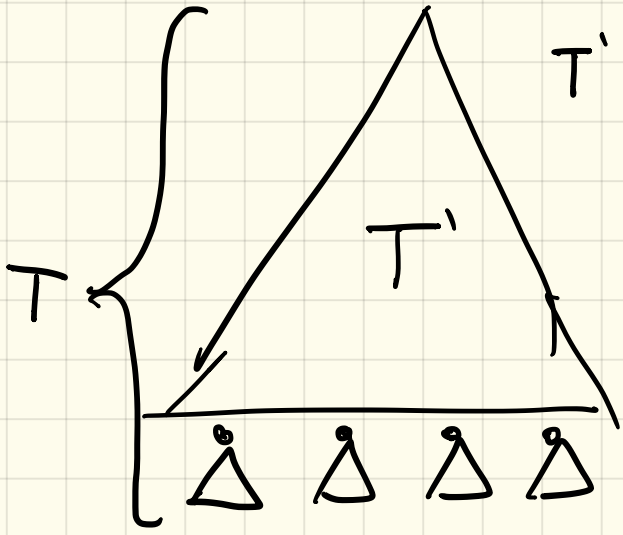
$\Rightarrow$ single ladder enough

$O(n \log n)$ preprocessing

$O(1)$ query time

5)

Problem:



T' has $O(n/\log n)$ leaves

$\leftarrow$ size $\leq \frac{1}{4} \log n$ (# nodes)

Idea: Jump pointers from leaves only

Nodes store pointer to a leaf below and the distance to it.

$$LA(x,k) = LA(v, k+d)$$

$O(n + L \cdot \log n)$ space

(n: ladder, L: # leaves)

$O(1)$ query times

→ Use this for T'

6) Preprocess queries for trees of $\leq \frac{1}{4} \log n$ nodes.

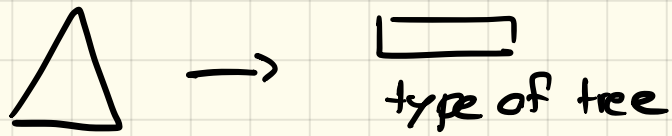
trees $\leq 2^{2 \frac{1}{4} \log n} = \sqrt{n}$

queries $(\frac{1}{4} \lg n)^2$

answer $\log \log n$

→ $O(n)$

Store nodes of small trees in array



$\Rightarrow$ $O(n)$ space

$O(1)$ query