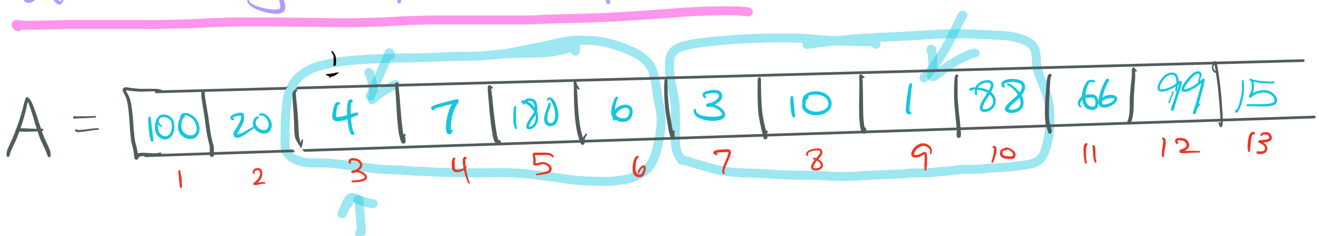
09-11 Range Minima Queries

"linearithmic" = $O(n \log n)$

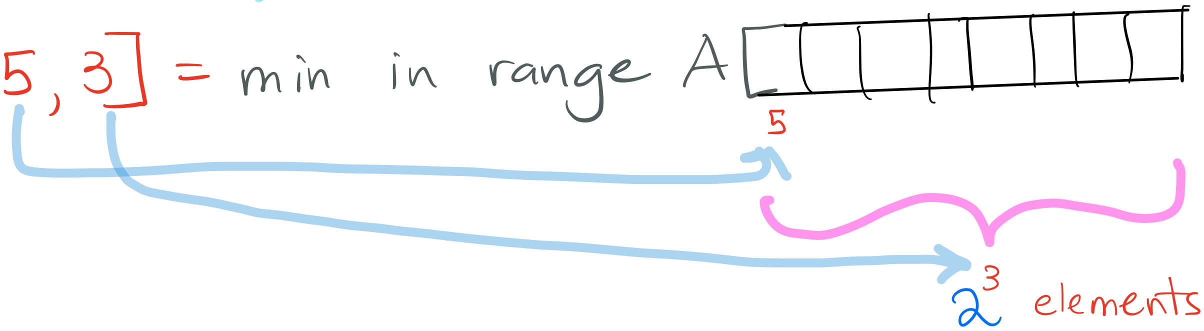
Last time, our choices were:

	Preprocessing	Query	Space
no preprocessing	$O(1)$	$O(n)$	$O(n)$
precompute everything	$O(n^2)$	$O(1)$	$O(n^2)$
segment tree	$O(n)$	$O(\log n)$	$O(n)$
Sparse Table precompute	$O(n \log n)$	$O(1)$	$O(n \log n)$
	$O(n)$	$O(\log n)$	$O(n)$

2^j-range precompute



$B[5,3]$ = min in range A



i.e. only precompute query solutions when range is exactly a power of 2, starting at each i.

$$B[i,0] = i \qquad \forall\, i \in [1..n]$$

$B[i,j]$ = index of min in $A[i \ldots i+2^j-1]$

$$B[i,j] = \begin{cases} B[i,j-1] & \text{if } A[B[i,j-1]] \leq A[B[i+2^{j-1}-1, j-1]] \\ B[i+2^{j-1}, j-1] & \text{otherwise.} \end{cases}$$

That is the pre-computing step

- takes $O(n \log n)$ time

and $O(n \log n)$ space

Query

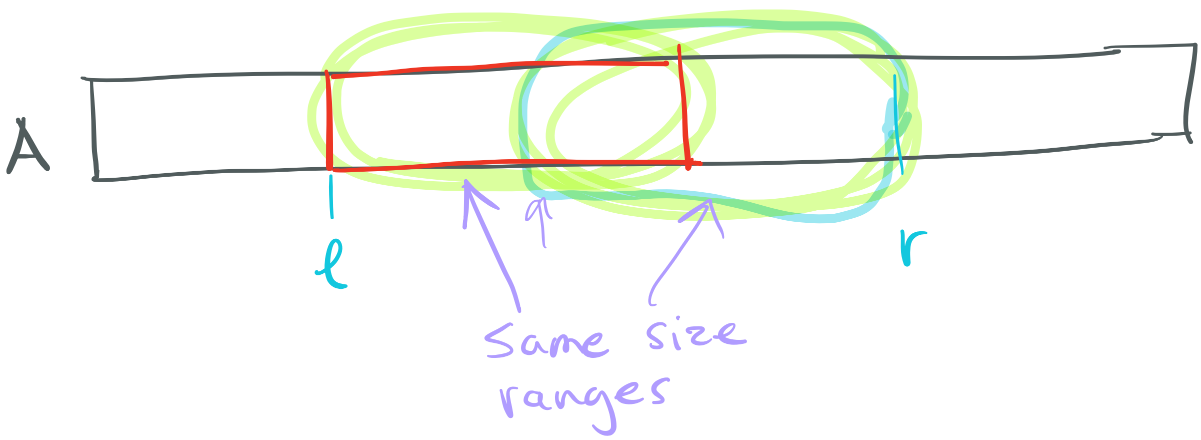
$RMQ_A(\ell, r)$

- find largest r' such that

  $r' \le r$

  $r' = \ell + 2^{j-1}$ for some int j

- look-up $B[\ell, r']$



- look-up $B[\, r - 2^j \, (+0 / +1?), r \,]$ for you to do.

- Compare the two, take the smallest

Linear time preprocessing log n queries

Linear space

Array A is conceptually divided into blocks of size $s = \frac{\log n}{4}$, ← how many? $\frac{4n}{\log n}$

$\frac{n}{(\log n/4)}$

- min for every block can be computed in time $\frac{\log n}{4} \cdot \frac{4 \cdot n}{\log n} \in O(n)$ time
- precompute and store in Look-up Table.

$RMQ_A(l, r)$

- naively search left boundary block
- naively search right boundary block

$O(s) = O(\log n)$

- Look-up each block in between ... ie naively search the Look-up Table.

$O\left(\frac{4n}{\log n}\right) = O\left(\frac{n}{\log n}\right)$

ie the block minima

Instead of naive search on the Look-up Table, use sparse table.

$\Rightarrow$ space + precompute is $O(n)$

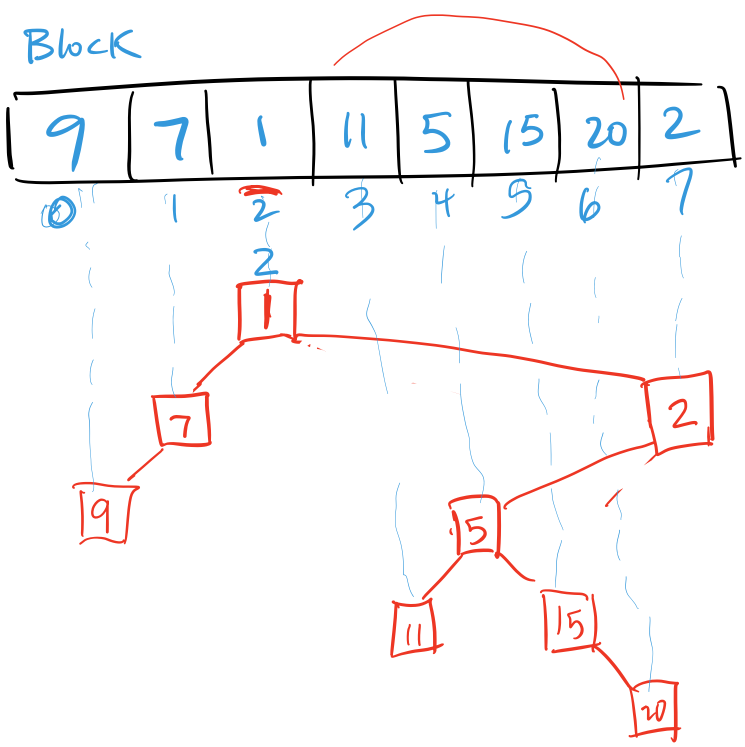
$\in O(n)$

query time $O(\log n)$. $\square$

Range Minima Queries in Constant time and Linear Space.

- use the above-given solution
  but be clever about the blocks

Construct a Cartesian Tree
for each block (of size $s = \frac{\log n}{4}$)



The solution for the whole block is always at the root.

For partial blocks, need to be able to query a range within a block/Cartesian tree and get the answer back in constant time

... and need to be able to store all the trees linear ($O(n)$) space.

Lets look at the storage first.

- The cartesian trees are of size $\frac{\log n}{4} = s$
- The number of different Cartesian trees of size s is C_s = s^{th} Catalan number

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Recurrence is $C_n = \sum_{i=1}^{n} C_{i-1} C_{n-i}$

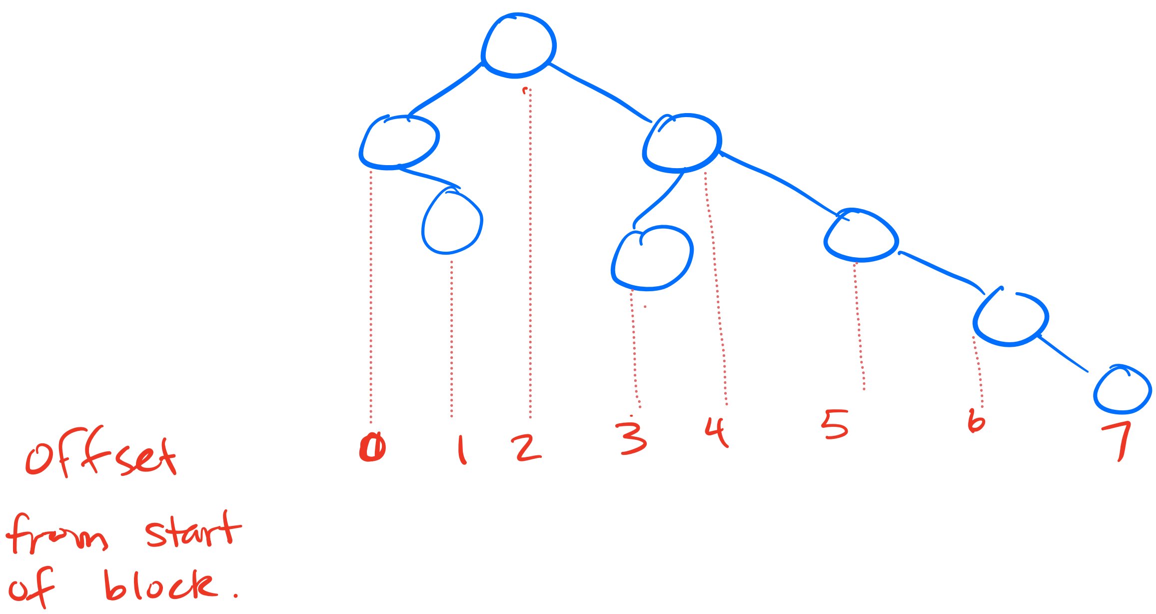
1	1	2	5	14	42	132	....
0	1	2	3	4	5	6	..

Point is, $C_n \in O(4^n)$

so $\exists$ 4^s Cartesian trees of size s

ie $4^{\frac{\log n}{4}}$ Cartesian trees of size s

Note: result depends only on shape of tree:



$\therefore$ only need to know which of the 4^s trees the block-of-interest is shaped like, then have a look-up table for each (ℓ, r) in that tree.

How many such pairs are there? ________.

$\therefore$ size of table is ________.

} on problem set.

How do you figure out what row corresponds to current block-tree?

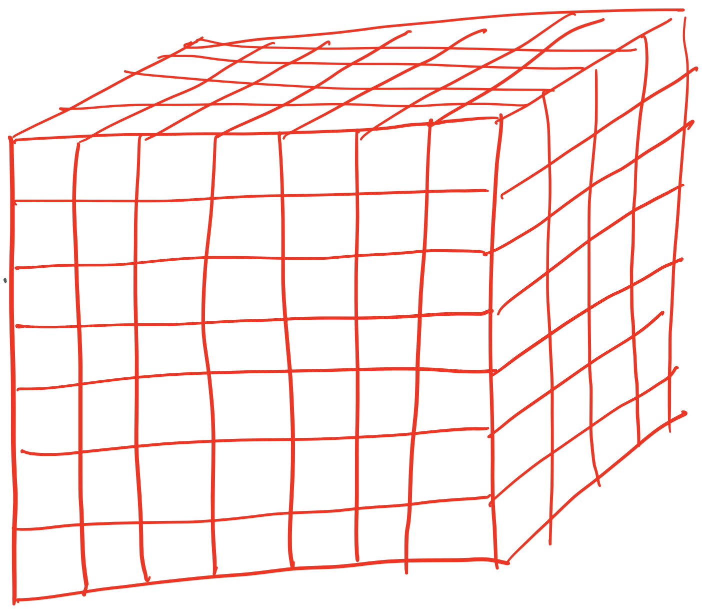
– on problem set

block tree 1

block tree 2

⋮

block tree 4^s



See en.wikipedia.org/wiki/Range_minimum_query