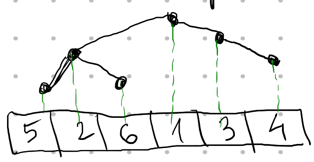
Building Cartesian trees in linear time

Given an array $A = (A[1], \dots, A[n])$

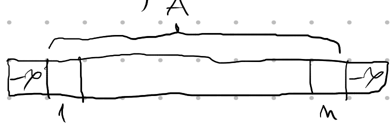
Cartesian tree of A

1. binary tree with nodes $(i, A[i])$
2. search tree according to i
3. min-heap according to $A[i]$



Algorithm

1. Add dummy $-\infty$ to the left, to the right of the array



2. Turn array into a linked list

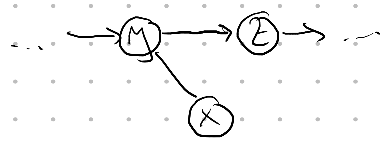
$(-\infty) \to (A[1]) \to (A[2]) \to \dots \to (A[n]) \to (-\infty)$

3. As long as there is a local maximum x in list, link(x).

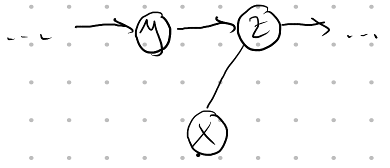
$\dots \to (y) \to (x) \to (z) \to \dots$ x is local max: $x > y$, $x > z$

link(x): make x child of $\max(y, z)$

- if $y > z$, make x right child of y



- if $z > y$, make x left child of z



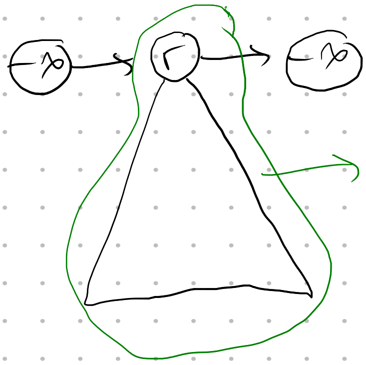
x dropped out of linked list

operation takes $O(1)$ time

after $n-1$ link operations

after $n-1$ link operations

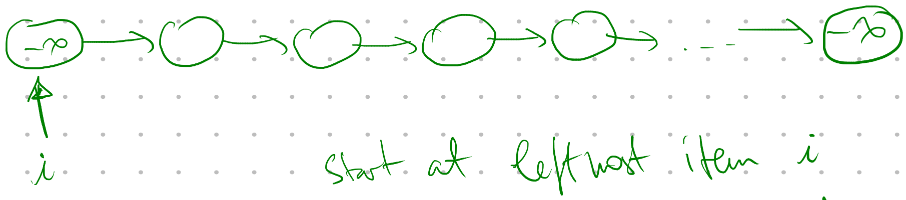
list:



Cartesian tree

running time $O(n)$

- link(x) $\rightarrow O(1)$ time
- how to find local max?

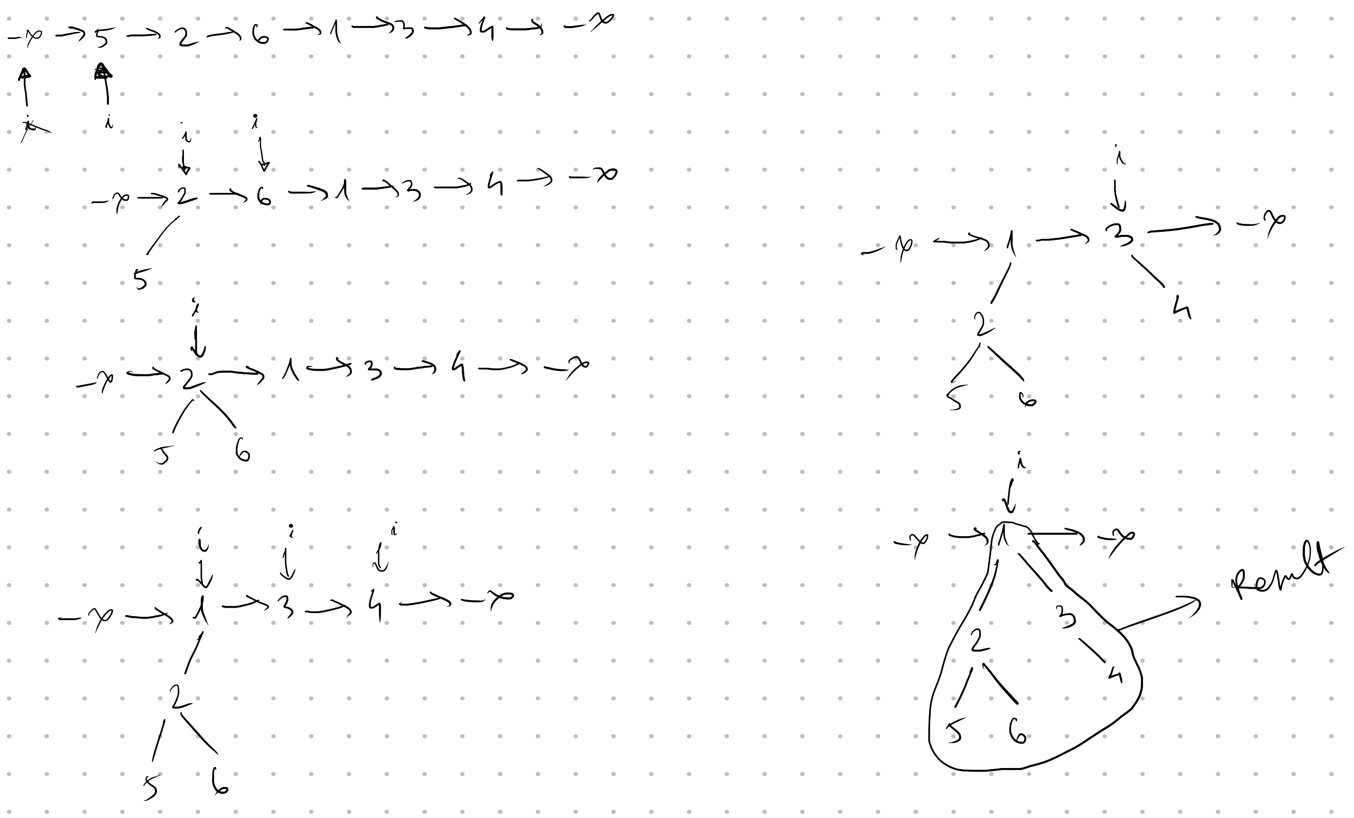


start at leftmost item i
while i not at the end
- if local max at i, then link(i)
- else $i \leftarrow i$.next

- left of i no local max
- every iteration makes progress:
  - either cut out one node (reduces list size by 1)
  - or move i to the right

example:

5	2	6	1	3	4



Correctness

1. binary tree
2. search tree
3. min-heap

$\Rightarrow$ Cartesian tree

3. link always makes a local max the child of its neighbor

1.

local max

$\to z \to x \to y \to$

$y > z$

x becomes left child of y

$\to z \to y \to \dots$

x

need to show: y will not get another left child.
This is true, because z<y, so z can never be a local max (as long as y is there),
so it will not be linked to y. When y is gone, then it cannot get left children anyway.

So a node can only get one left child. By symmetric argument, only one right child.

2.

Maintain invariant: if x is left of y in the list, then all nodes in subtree(x)
have smaller indices than all nodes in subtree(y).
This is true initially.
A single link preserves this, so always true.
From this property it follows that the trees created by link have search tree property.