Building Cartesian trees in linear time

Given an array $A = (A_0, ..., A_{N-1})$

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 $\text{null}$ to the left, to the right of the array

2. Turn array into a linked list

3. As long as there is a local maximum $x$ in list,
   \[ \text{link}(x) \]

\[ \cdots \rightarrow A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow \cdots \]
\[ x \quad \text{is local max}. \quad x > M \quad x > 2 \]

\[ \text{link}(x): \text{make } x \text{ child of } \max(M, 2) \]
\[ \text{if } y > 2, \text{ make } x \text{ right child of } y \]
\[ \cdots \rightarrow A_0 \rightarrow x \rightarrow \cdots \]
\[ \text{if } z > y, \text{ make } x \text{ left child of } z \]
\[ \cdots \rightarrow m \rightarrow z \rightarrow \cdots \]
\[ x \text{ dropped out of list} \]

Operation takes $O(1)$ time

- after $N-1$ link operations
After min link operations:

- Min heap:
  - $\infty \rightarrow 3 \rightarrow 8 \rightarrow 5 \rightarrow \ldots \rightarrow 4 \rightarrow \infty$

- Start at leftmost item $i$.
- While $i$ not at the end:
  - If local min at $i$, then link $(i)$.
  - Else, $i \leftarrow i.next$.
- Left if $i$ no local min.
- Every iteration makes progress:
  - Either cut out one node (reduces list size by 1).
  - Or move $i$ to the right.

Example:

```
5 1 3 7 4
```

- Local min at 1:

```
\infty \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow \infty
```

Result:

```
\infty \rightarrow 2 \rightarrow 3 \rightarrow 5 6
```

Running time $O(n)$:
- $\text{link}(x) \rightarrow O(1)$ time.
- How to find local min?
Correctness

1. binary tree \( \xrightarrow{?} \) criteria tree
2. search tree
3. min-heap

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

1. \( \text{local max} \rightarrow 2 \rightarrow x \rightarrow y \rightarrow y_? \)

\( x \) becomes left child of \( y \)

\( \rightarrow 2 \rightarrow y \rightarrow \ldots \)

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.