The LCA Problem Revisited

Michael A.Bender & Martin Farach-Colton

Presented by: Dvir Halevi

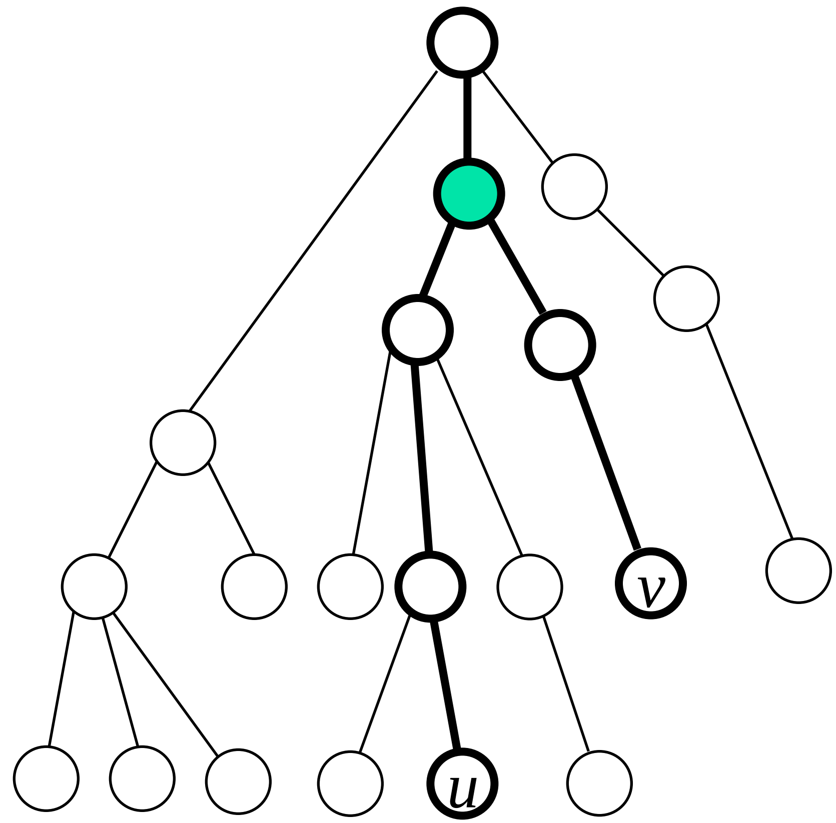
Agenda

- Definitions
- Reduction from LCA to RMQ
- Trivial algorithms for RMQ
- ST algorithm for RMQ
- A faster algorithm for a private RMQ case
- General Solution for RMQ

Definitions – Least Common Ancestor

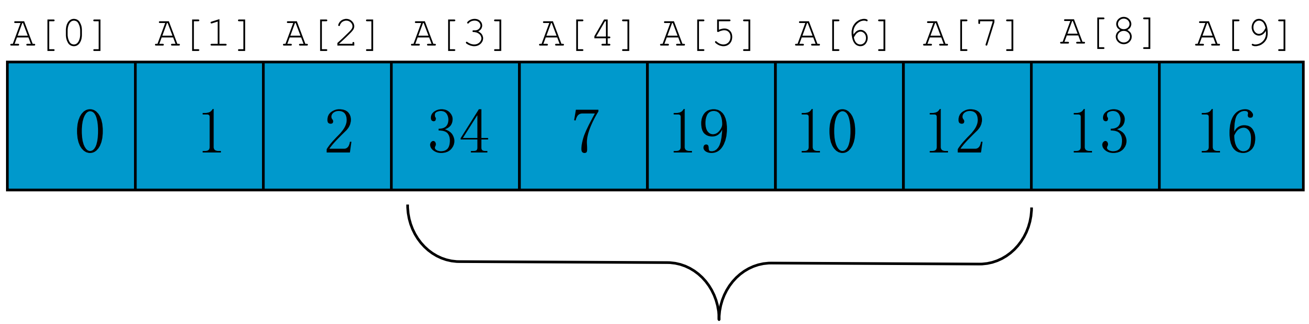
- $LCA_T(u,v)$ – given nodes u,v in T, returns the node furthest from the root that is an ancestor of both u and v.

Trivial solution: $O(n^3)$



Definitions – Range Minimum Query

- Given array A of length n.
- $RMQ_A(i,j)$ – returns the index of the smallest element in the subarray A[i..j].



RMQ(3,7) = 4

Definitions – Complexity Notation

- Suppose an algorithm has:
  - Preprocessing time – $f(n)$
  - Query time – $g(n)$
- Notation for the overall complexity of an algorithm:

$$< f(n), g(n) >$$

- Definitions
- Reduction from LCA to RMQ
- Trivial algorithms for RMQ
- ST algorithm for RMQ
- A faster algorithm for a private RMQ case
- General Solution for RMQ

Reduction from LCA to RMQ

- In order to solve LCA queries, we will reduce the problem to RMQ.

- Lemma:

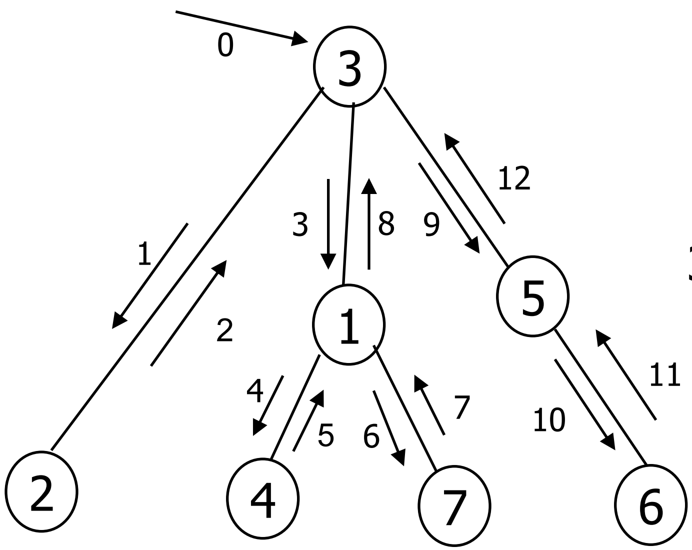
  If there is an $<f(n), g(n)>$ *solution for RMQ, then there is an* $<f(2n-1)+O(n), g(2n-1)+O(1)>$

  Solution for LCA.

Reduction - proof

- Observation:

  The LCA of nodes u and v is the shallowest node encountered between the visits to u and to v during a depth first search traversal of T.



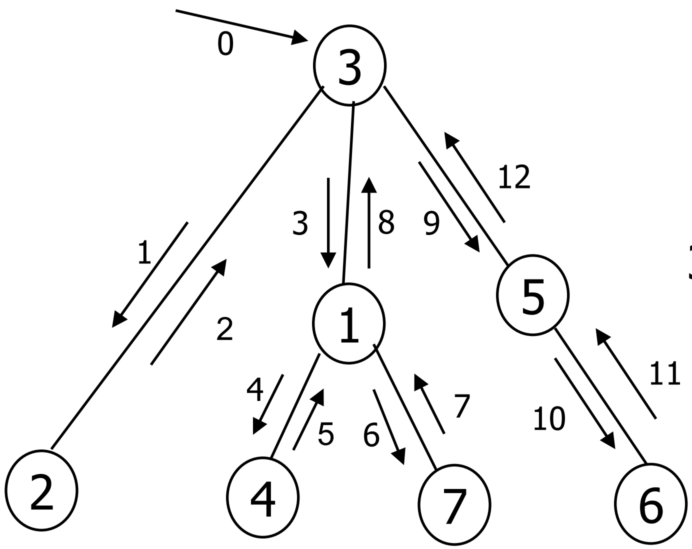
Euler tour:

Shallowest node

3 2 3 1 4 1 7 1 3 5 6 5 3

$$LCA_T(1,5) = 3$$

Reduction (cont.)



Shallowest node

Euler tour:

3 2 3 1 4 1 7 1 3 5 6 5 3

$$LCA(1, 5) = 3$$

- Remarks:
  - Euler tour size: 2n-1
  - We will use the first occurrence of i,j for the sake of concreteness (any occurrence will suffice).
  - Shallowest node must be the LCA, otherwise contradiction to a DFS run.

Reduction (cont.)

- On an input tree T, we build 3 arrays.
- Euler[1,..,2n-1] – The nodes visited in an Euler tour of T. Euler[i] is the label of the i-th node visited in the tour.
- Level[1,..2n-1] – The level of the nodes we got in the tour. Level[i] is the level of node Euler[i].

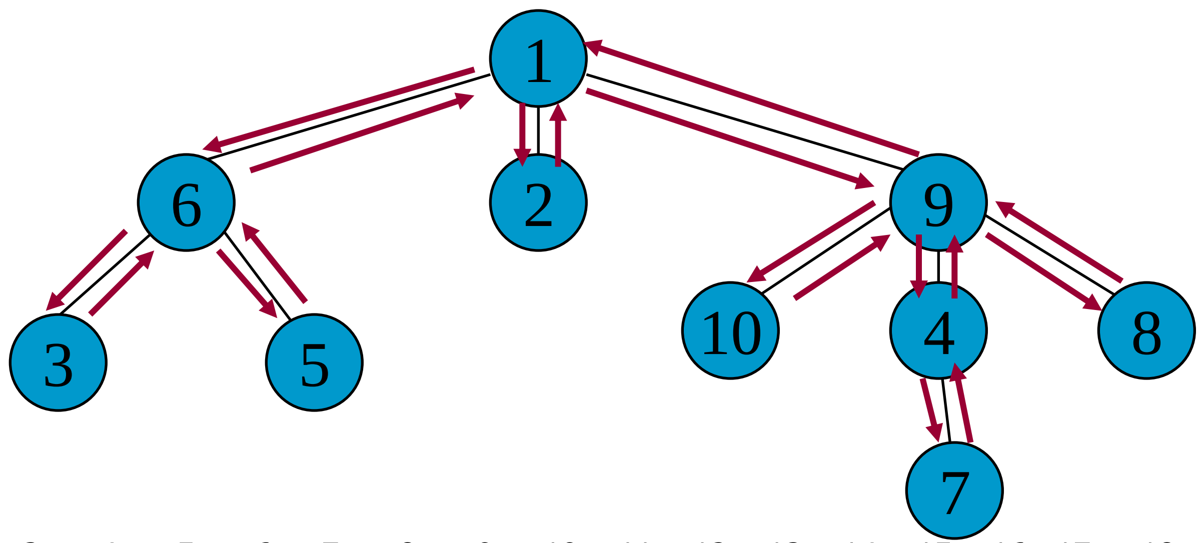
  (level is defined to be the distance from the root)
- Representative[1,..n] – Representative[i] will hold the **index** of the first occurrence of node i in Euler[].

  Representative[v] = $\arg\min_i \{Euler[i] = v\}$

Mark: Euler – E, Representative – R, Level – L

Reduction (cont.)

- Example:



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

R: 1 8 3 13 5 2 14 17 10 11

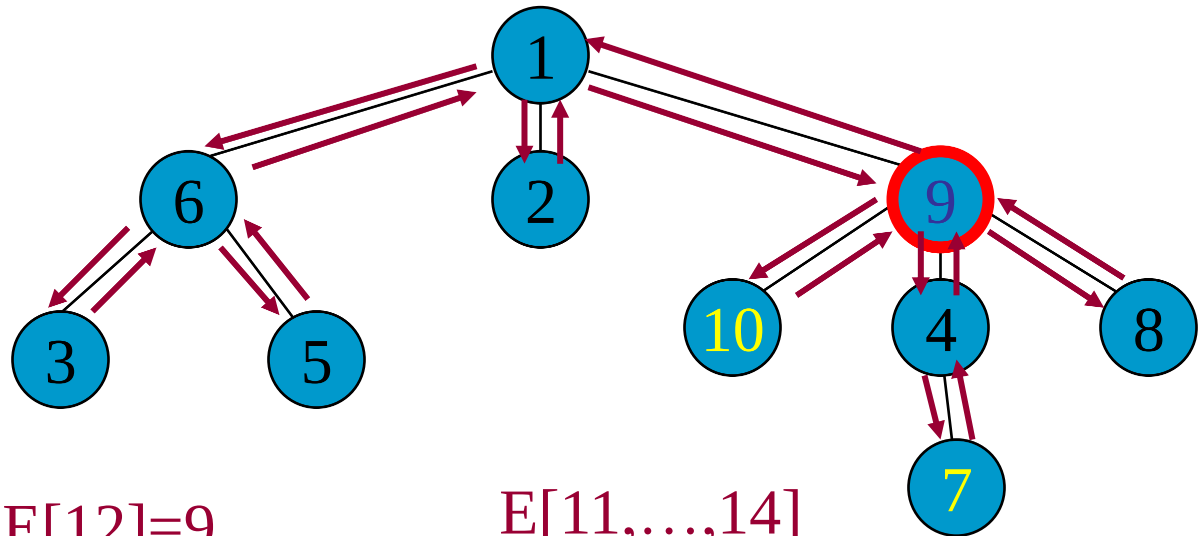
Reduction (cont.)

- To compute $LCA_T(x,y)$:
  - All nodes in the Euler tour between the first visits to x and y are E[R[x],..,R[y]] (assume R[x] < R[y])
  - The shallowest node in this subtour is at index RMQ_L(R[x],R[y]), since L[i] stores the level of the node at E[i].
  - RMQ will return the index, thus we output the node at E[RMQ_L(R[x],R[y])] as $LCA_T(x,y)$.

Reduction (cont.)

- Example:

LCA$_T$(10,7)



LCA$_T$(10,7) = E[12]=9

E[11,…,14]

E: 1 6 3 6 5 6 1 2 1 9 10 9 4 7 4 9 8 9 1

L: 0 1 2 1 2 1 0 1 0 1 2 1 2 3 2 1 2 1 0

R: 1 8 3 13 5 2 14 17 10 11

RMQ$_L$(10,7) = 12

R[7] R[10]

Reduction (cont.)

- Preprocessing Complexity:
  - L,R,E – Each is built in $O(n)$ time, during the DFS run.
  - Preprocessing L for RMQ - $f(2n-1)$
- Query Complexity:
  - RMQ query on L – $g(2n-1)$
  - Array references – $O(1)$
- Overall: $< f(2n-1)+O(n), g(2n-1)+O(1) >$
- Reduction proof is complete.
- We will only deal with RMQ solutions from this point on.

- Definitions
- Reduction from LCA to RMQ
- Trivial algorithms for RMQ
- ST algorithm for RMQ
- A faster algorithm for a private RMQ case
- General Solution for RMQ

RMQ

- Solution 1:

  Given an array A of size n, compute the RQM for every pair of indices and store in a table - $<O(n^3), O(1)>$

- Solution 2:

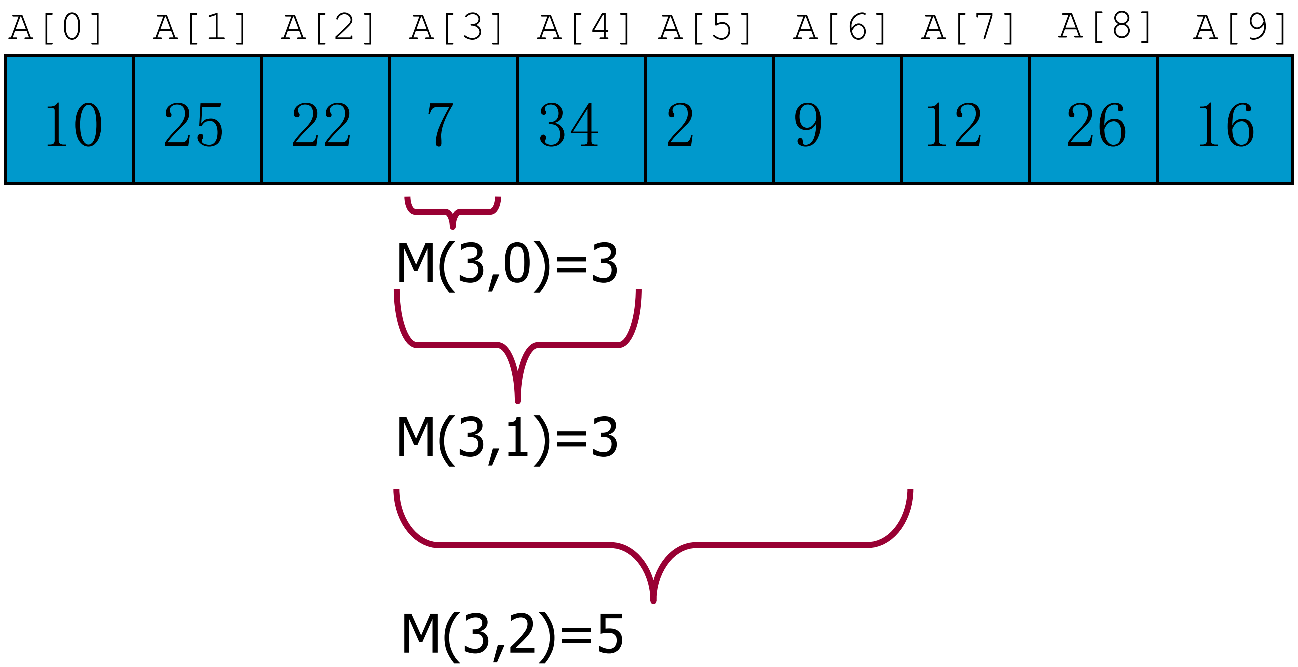
  To calculate RMQ(i,j) use the already known value of RMQ(i,j-1) .

  Complexity reduced to - $<O(n^2), O(1)>$

- Definitions
- Reduction from LCA to RMQ
- Trivial algorithms for RMQ
- ST algorithm for RMQ
- A faster algorithm for a private RMQ case
- General Solution for RMQ

ST RMQ

- Preprocess sub arrays of length 2^k
- M(i,j) = index of min value in the sub array starting at index i having length 2^j



ST RMQ

- Idea: precompute each query whose length is a power of n. For every i between 1 and n and every j between 1 and $\lfloor \log n \rfloor$ find the minimum element in the block starting at i and having length 2^j.

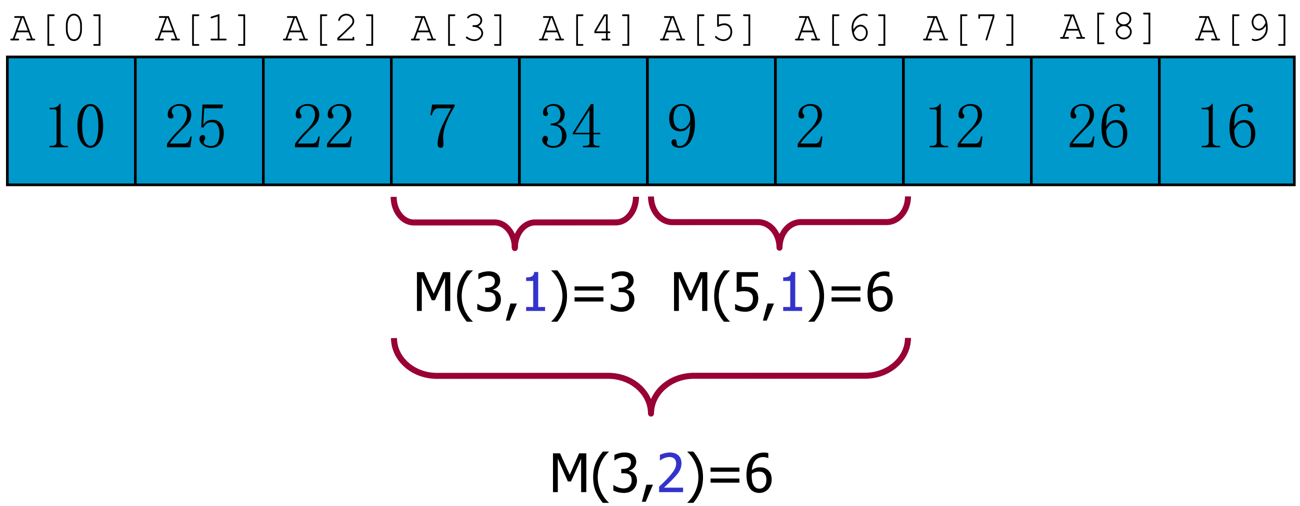
- More precisely we build table M.

$$M[i,j] = \arg\min_{k=i..i+2^j-1}\{Array[k]\}$$

- Table M therefore has size O(nlogn).

ST RMQ

- Building M – using dynamic programming we can build M in O(nlogn) time.

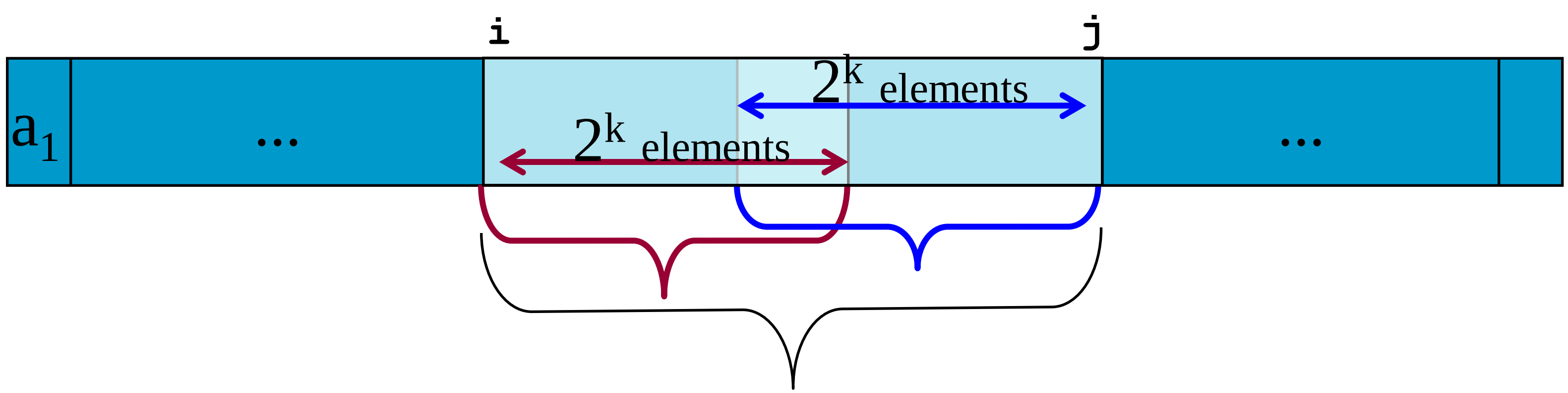


$$M[i,j]=\begin{cases} M[i,j\text{-}1] & A[M[i,j\text{-}1]] \le A[M[i+2^{j-1}-1, j-1]] \\ M[i+2^{j-1}-1, j-1] & \text{Otherwise} \end{cases}$$

ST RMQ

- Using these blocks to compute arbitrary M[i,j]
- Select two blocks that entirely cover the subrange [i..j]
- Let $k = \lfloor \log(j-i) \rfloor$ (2^k is the largest block that fits [i..j])
- Compute RMQ(i,j):

$$\mathrm{RMQ}(i,j) = \begin{cases} \mathrm{M}[i,k] & \mathrm{A}[\mathrm{M}[i,k]] \le \mathrm{A}[\mathrm{M}[j-2^k+1,k]] \\ \mathrm{M}[j-2^k+1,k] & \text{Otherwise} \end{cases}$$



ST RMQ

- Query time is O(1).
- This algorithm is known as Sparse Table(ST) algorithm for RMQ, with complexity:

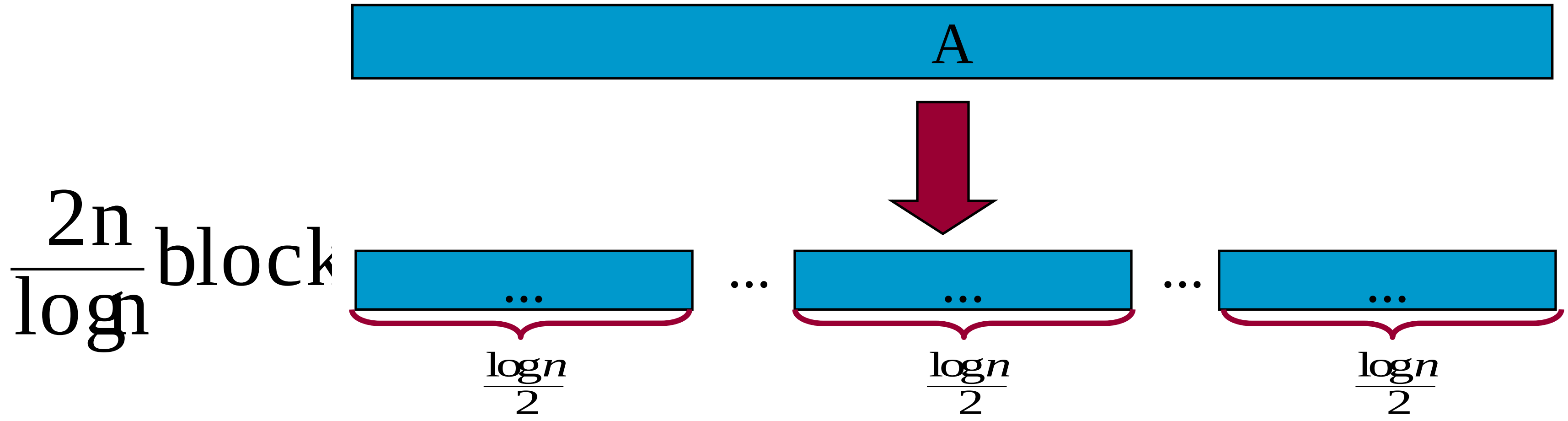
$$< O(n\log n), O(1) >$$

- Our target: get rid of the log(n) factor from the preprocessing.

- Definitions
- Reduction from LCA to RMQ
- Trivial algorithms for RMQ
- ST algorithm for RMQ
- A faster algorithm for a private RMQ case
- General Solution for RMQ

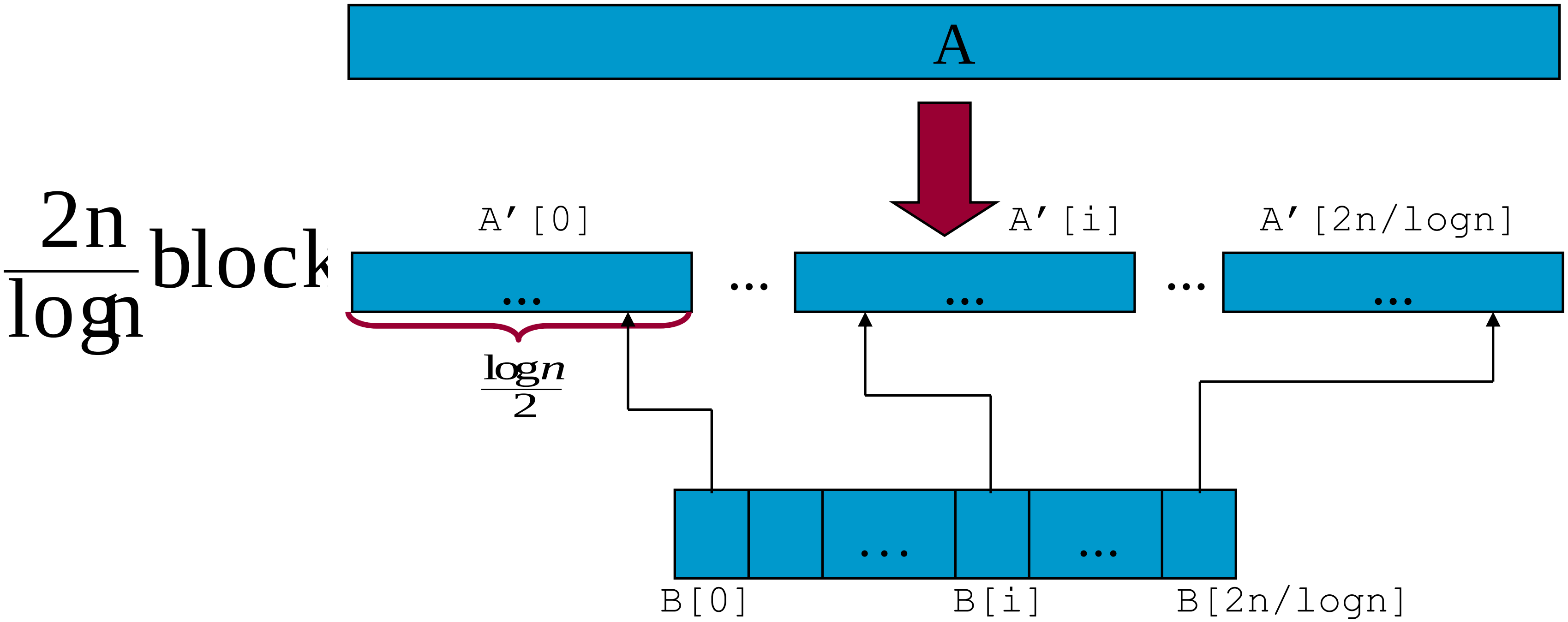
Faster RMQ

- Use a table-lookup technique to precompute answers on small subarrays, thus removing the log factor from the preprocessing.
- Partition A into $\frac{2n}{\log n}$ blocks of size $\frac{\log n}{2}$.



Faster RMQ

- A'[1,.., $\frac{2n}{\log n}$] – A'[i] is the minimum element in the i-th block of A.
- B[1,.., $\frac{2n}{\log n}$] – B'[i] is the position (index) in which value A'[i] occurs.



- Example:

n=16

A[] :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	25	22	7	34	9	2	12	26	33	24	43	5	11	19	27

10	25

22	7

34	9

…

$\frac{2n}{\log n}$ blocks = 8

A′[] :

0	1	2	
10	7	9	…

B[] :

0	1	2	
0	3	5	…

Faster RMQ

- Recall RMQ queries return the position of the minimum.
- LCA to RMQ reduction uses the position of the minimum, rather than the minimum itself.
- Use array B to keep track of where minimas in A' came from.

Faster RMQ

- Preprocess A' for RMQ using ST algorithm.
- ST's preprocessing time – O(nlogn).
- A's size – $\frac{2n}{\log n}$
- ST's preprocessing on A': $\frac{2n}{\log n}\log(\frac{2n}{\log n})=O(n)$
- ST(A') = $\langle O(n),O(1)\rangle$

Faster RMQ

- Having preprocessed A' for RMQ, how to answer RMQ(i,j) queries on A?
- i and j might be in the same block -> preprocess every block.
- i < j on different blocks, answer the query as follows:
  1. Compute minima from i to end of its block.
  2. Compute minima of all blocks in between i's and j's blocks.
  3. Compute minima from the beginning of j's block to j.
- Return the index of the minimum of these 3 values.

Faster RMQ

- i < j on different blocks, answer the query as follows:
  1. Compute minima from i to end of its block.
  2. Compute minima of all blocks in between i's and j's blocks.
  3. Compute minima from the beginning of j's block to j.

- 2 – Takes O(1) time by RMQ on A'.
- 1 & 3 – Have to answer in-block RMQ queries

- We need in-block queries whether i and j are in the same block or not.

Faster RMQ

- First Attempt: preprocess every block.

  Per block : $$\frac{\log n}{2}\log\left(\frac{\log n}{2}\right)=O(\log n\log\log n)$$

  All $\frac{2n}{\log n}$ blocks – $O(n\log\log n)$

- Second attempt: recall the LCA to RMQ reduction
- RMQ was performed on array L.
- What can we use to our advantage?

± 1 restriction

Faster RMQ

- Observation:

  Let two arrays X & Y such that $\forall i\ X[i] = Y[i] + C$

  Then $\forall i,j\ RMQ_X(i,j) = RMQ_Y(i,j)$

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	4	5	6	5	4	5	6	5	4

B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]	B[8]	B[9]
0	1	2	3	2	1	2	3	2	1

+1	+1	+1	−1	−1	+1	+1	−1	−1

$$2^{\left(\frac{\log n}{2} - 1\right)} = O(\sqrt{n})$$

- There are $O(\sqrt{n})$ normalized blocks.

Faster RMQ

- Preprocess:
  - Create $O(\sqrt{n})$ tables of size $O(\log^2 n)$ to answer all in block queries. Overall $O\left(\sqrt{n}\log^2 n\right)=O(n)$.
  - For each block in A compute which normalized block table it should use – $O(n)$
  - Preprocess A' using ST - $O(n)$
- Query:
  - Query on A' – $O(1)$
  - Query on in-blocks – $O(1)$
- Overall RMQ complexity - $\boxed{\langle O(n), O(1)\rangle}$

- Definitions
- Reduction from LCA to RMQ
- Trivial algorithms for RMQ
- ST algorithm for RMQ
- A faster algorithm for a private RMQ case
- General Solution for RMQ

General O(n) RMQ

- Reduction from RMQ to LCA
- General RMQ is solved by reducing RMQ to LCA, then reducing LCA to ± 1 RMQ.

- Lemma:
  If there is a $\langle O(n), O(1) \rangle$ solution for LCA, then there is a $\langle O(n), O(1) \rangle$ solution to RMQ.

- Proof: build a Cartesian tree of the array, activate LCA on it.

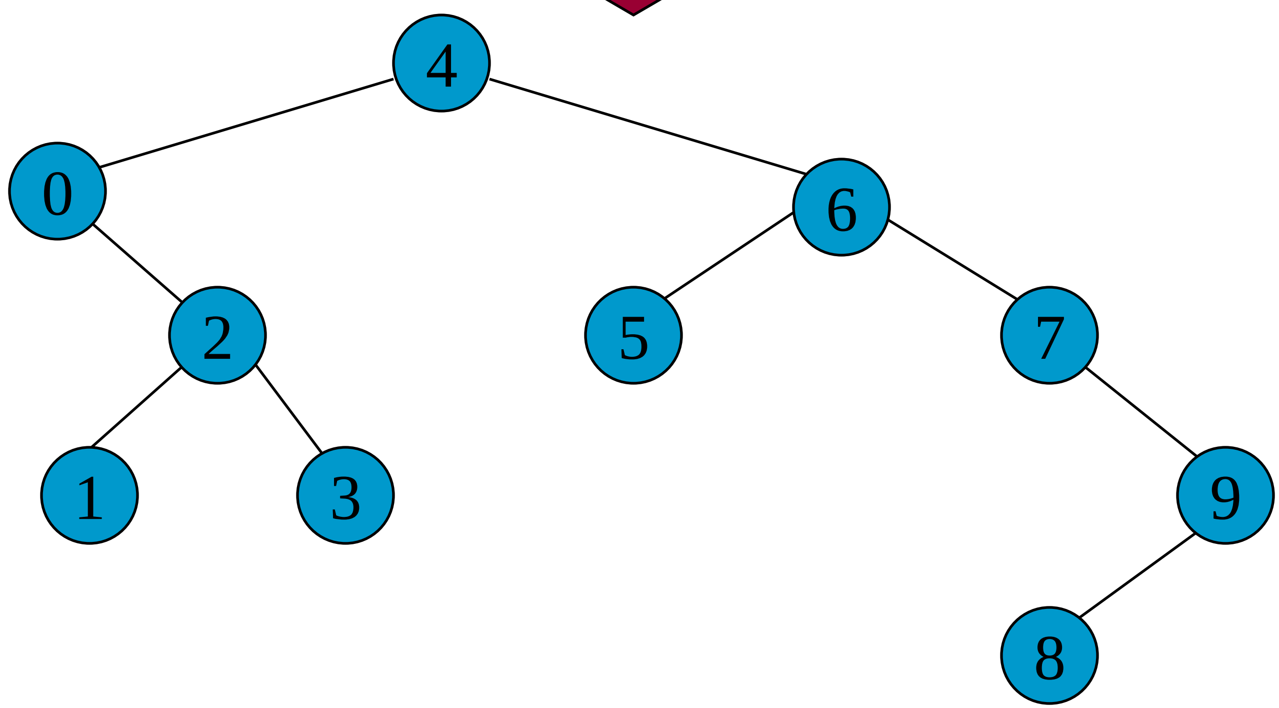
General O(n) RMQ

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
10	25	22	34	7	19	9	12	26	16

- Cartesian tree of an array A:
  - Root – minimum element of the array. Root node is labeled with the position of the minimum.
  - Root's left & right children: the recursively constructed Cartesian tress of the left & right subarrays, respectively.

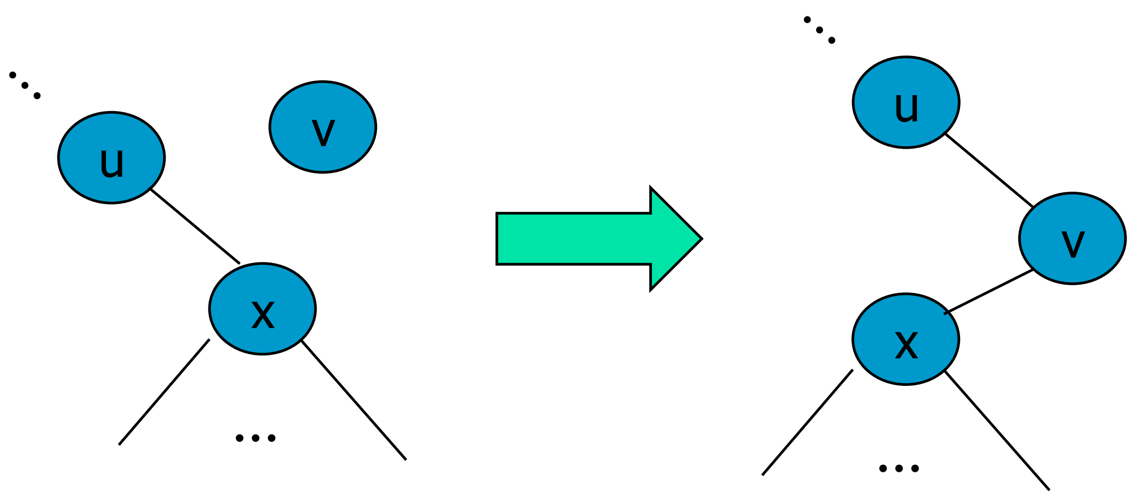
General O(n) RMQ

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
10	25	22	34	7	19	9	12	26	16



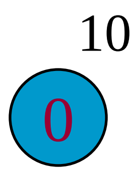
Build Cartesian tree in O(n)

- Move from left to right in the array
- Suppose C_i is the Cartesian tree of A[1,..,i]
- Node i+1 (v) has to belong in the rightmost path of C_i
- Climb the rightmost path, find the first node (u) smaller than v
- Make v the right son of u, and previous right subtree of u left son of v.



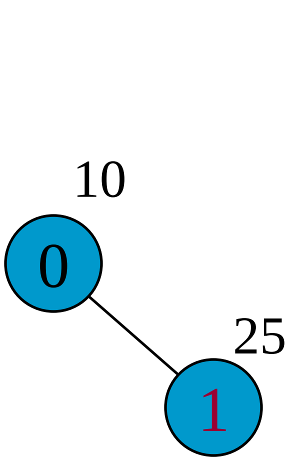
Build Cartesian tree in O(n)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
10	25	22	34	7	19	9	12	26	16



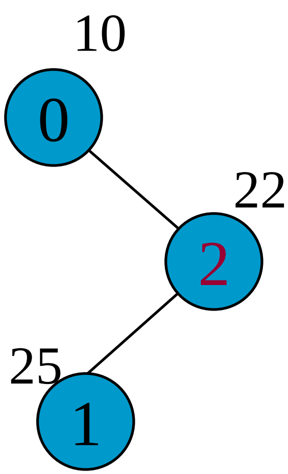
Build Cartesian tree in O(n)

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
10	25	22	34	7	19	9	12	26	16



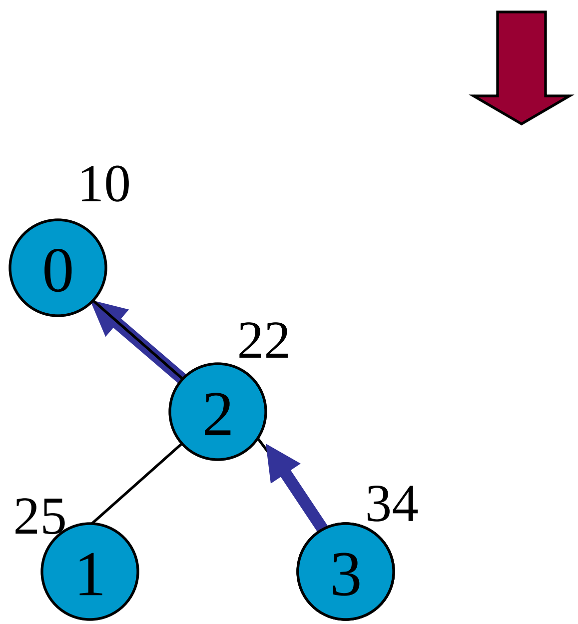
Build Cartesian tree in O(n)

A[0]	A[1]	A[**2**]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
10	25	22	34	7	19	9	12	26	16

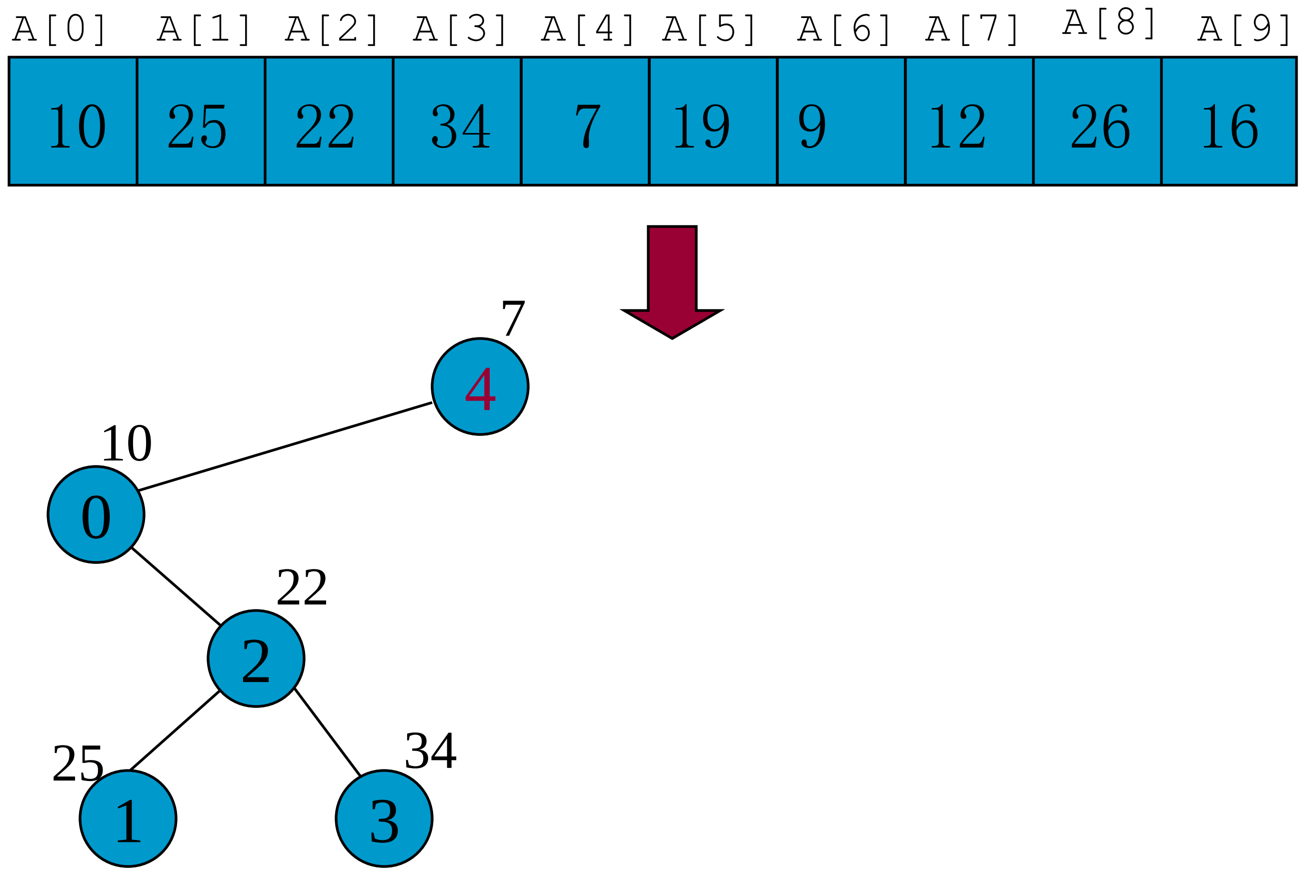


Build Cartesian tree in O(n)

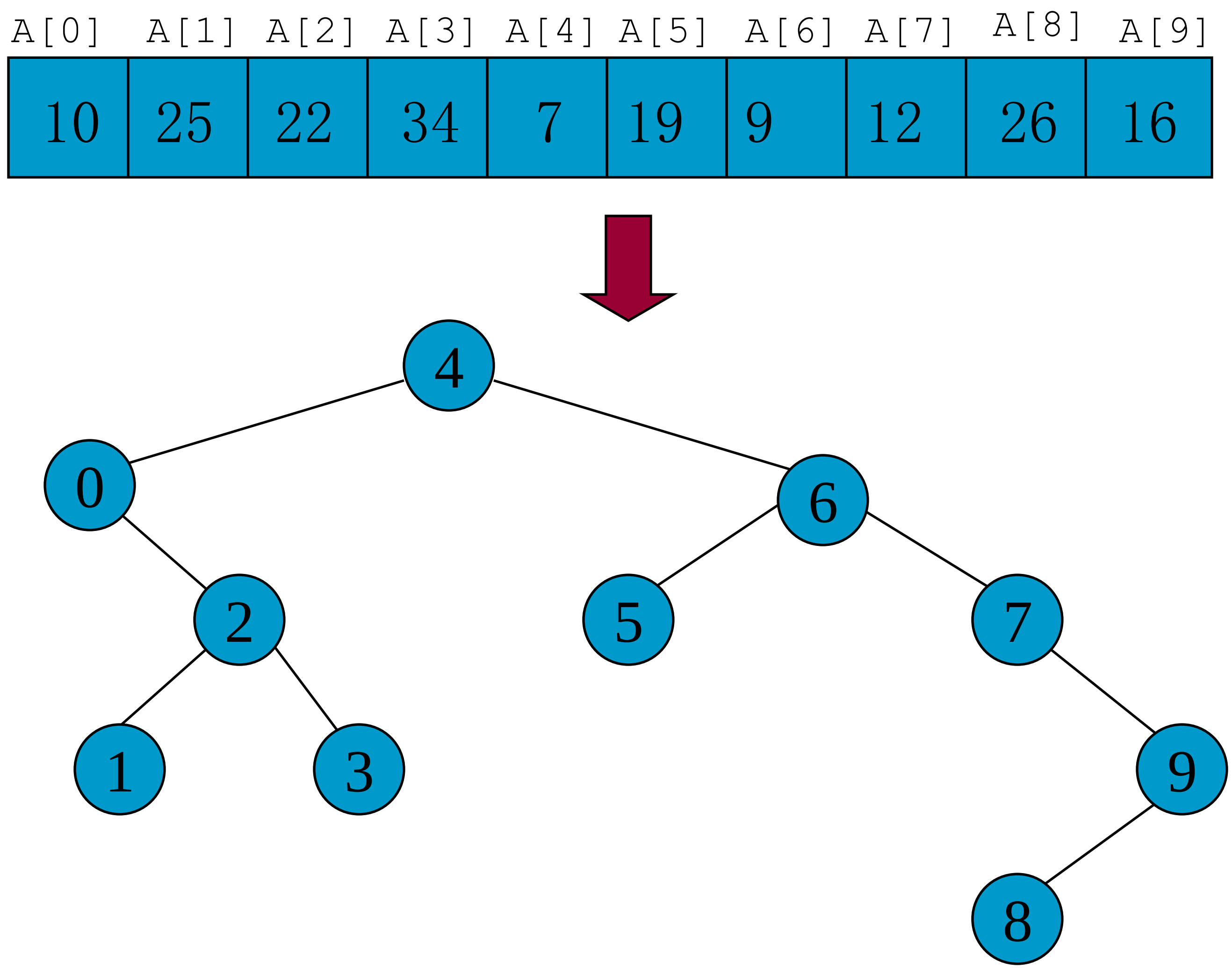
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
10	25	22	34	7	19	9	12	26	16



Build Cartesian tree in O(n)



Build Cartesian tree in O(n)



General O(n) RMQ

- How to answer RMQ queries on A?
- Build Cartesian tree C of array A.
- $RMQ_A(i,j) = LCA_C(i,j)$

- Proof:
  - let $k = LCA_C(i,j)$.
  - In the recursive description of a Cartesian tree k is the first element to split i and j.
  - k is between i,j since it splits them and is minimal because it is the first element to do so.

General O(n) RMQ

- Build Complexity:

- Every node enters the rightmost path once. Once it leaves, will never return.

- O(n).

General O(n) RMQ