Advanced Algorithms – COMS31900

Range Minimum Queries

Raphaël Clifford

Slides by Benjamin Sach

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

n

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

A (length n):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54

After preprocessing, a **range minimum query** is given by $\mathrm{RMQ}(i, j)$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

n

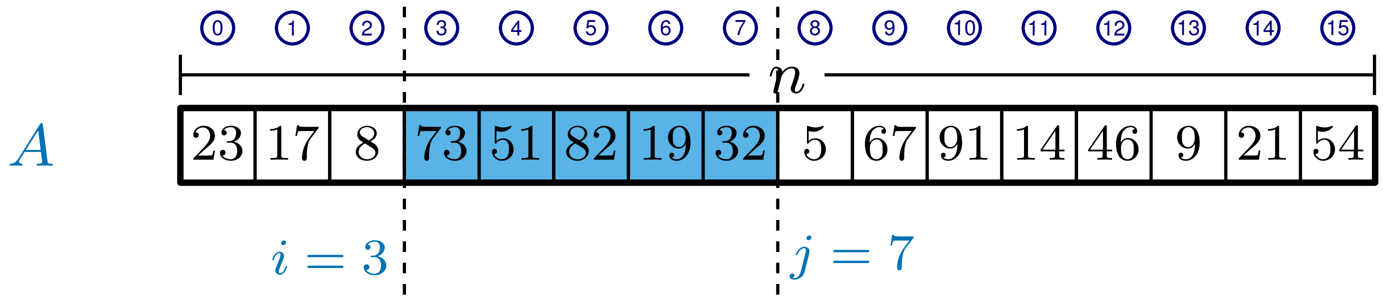
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54

After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

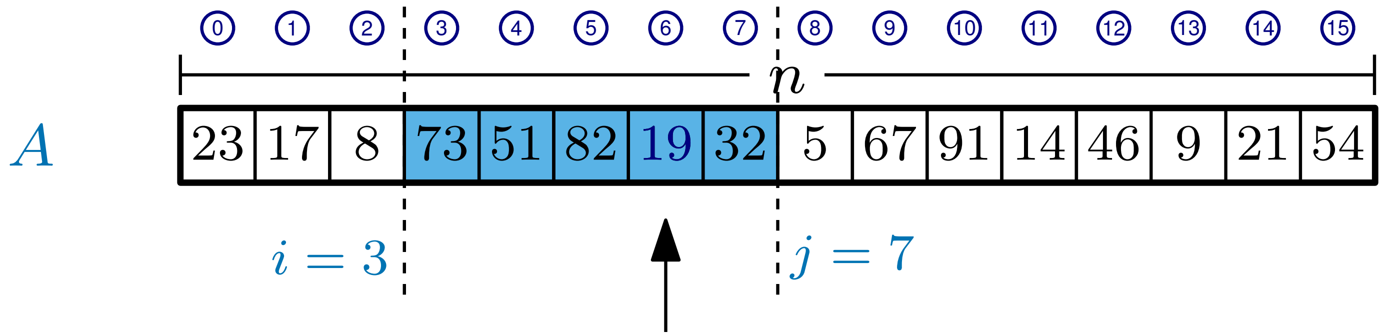


After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

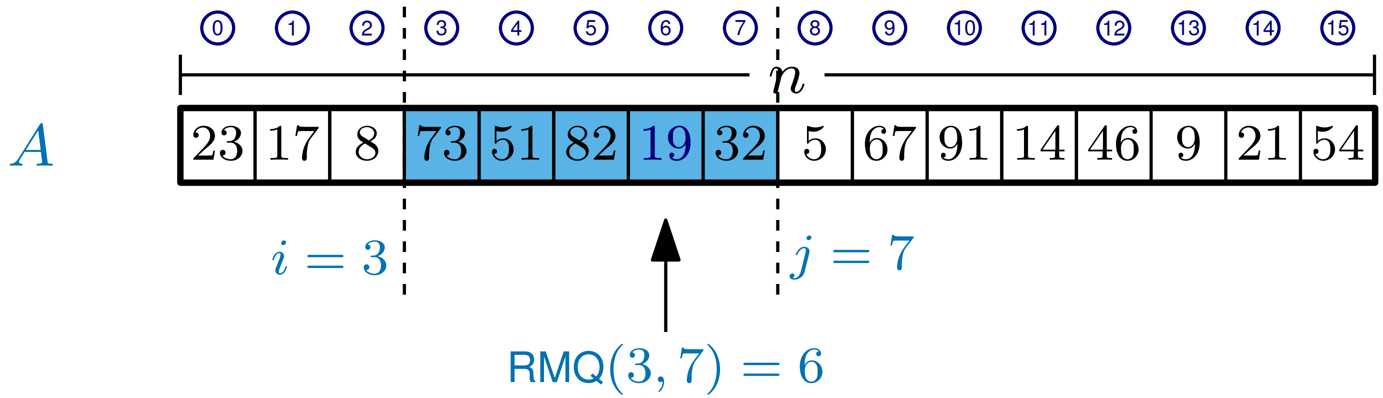


After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...

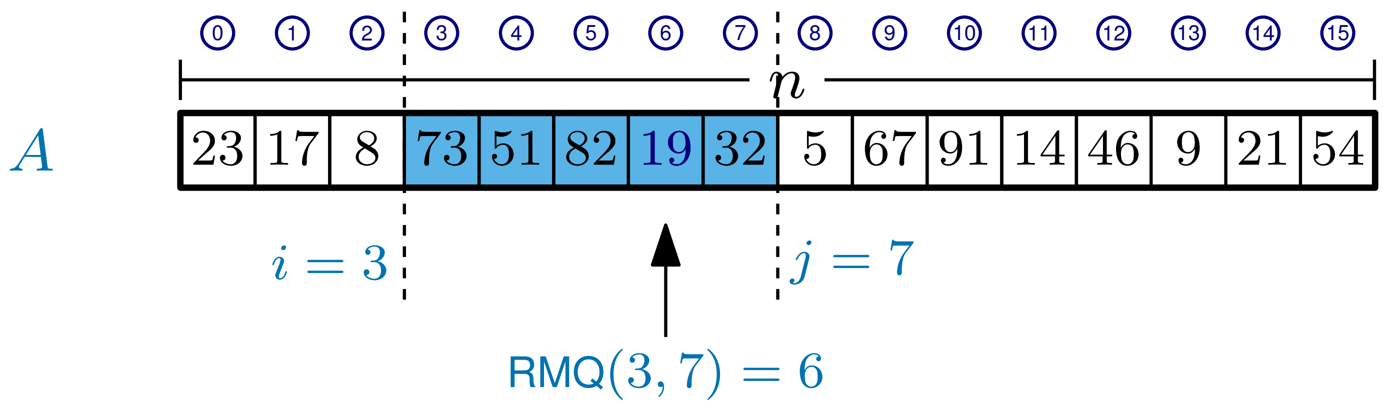


After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...



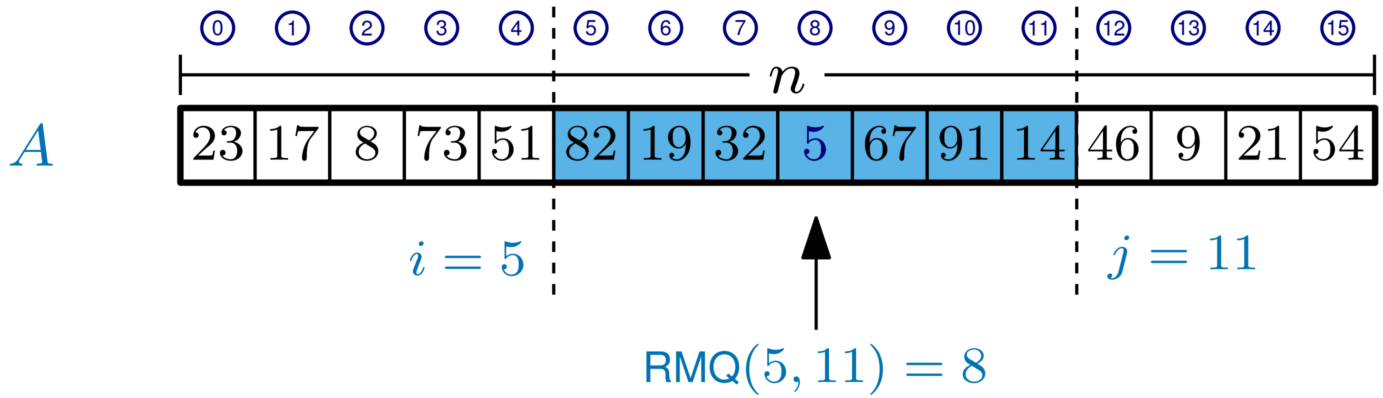
After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

e.g. RMQ$(3, 7) = 6$, which is the location of the smallest element in $A[3, 7]$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

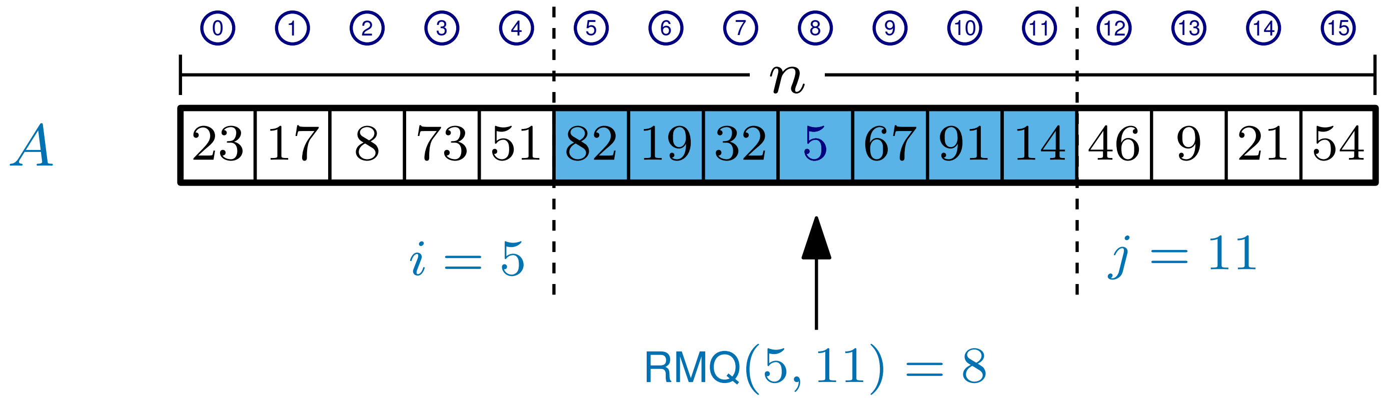
the output is the location of the smallest element in $A[i, j]$

e.g. RMQ$(3, 7) = 6$, which is the location of the smallest element in $A[3, 7]$

e.g. RMQ$(5, 11) = 8$, which is the location of the smallest element in $A[5, 11]$

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

e.g. RMQ$(3, 7) = 6$, which is the location of the smallest element in $A[3, 7]$

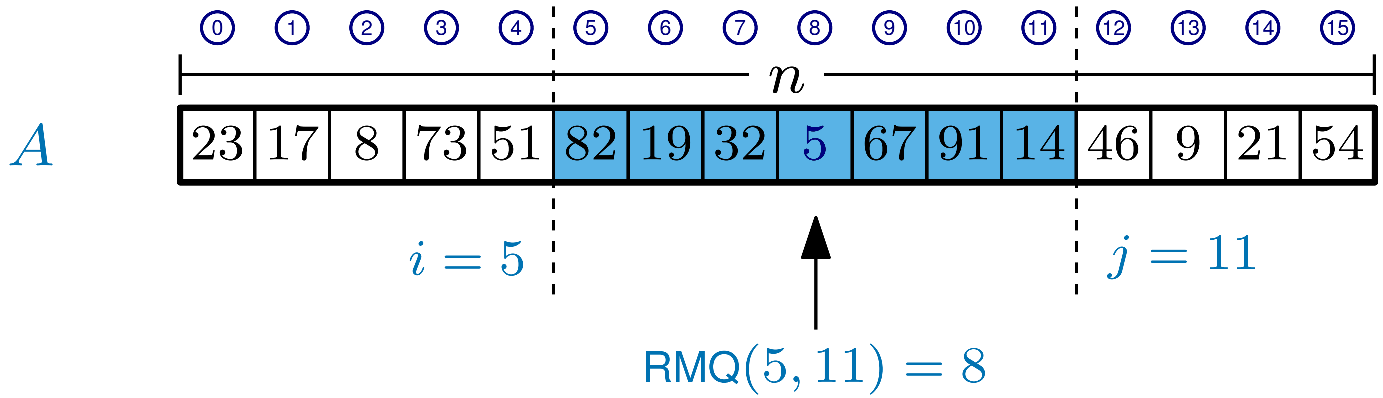
e.g. RMQ$(5, 11) = 8$, which is the location of the smallest element in $A[5, 11]$

- We will discuss several algorithms which give trade-offs between

  space used, prep. time and query time

Range minimum query

Preprocess an integer array A (length n) to answer range minimum queries. . .



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

e.g. RMQ$(3, 7) = 6$, which is the location of the smallest element in $A[3, 7]$

e.g. RMQ$(5, 11) = 8$, which is the location of the smallest element in $A[5, 11]$

- We will discuss several algorithms which give trade-offs between

  space used, prep. time and query time

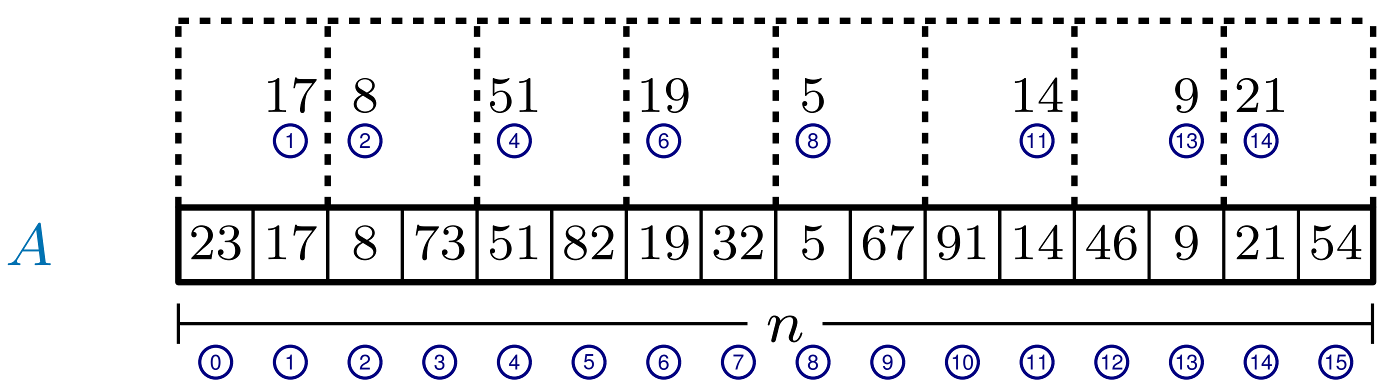
- Ideally we would like $O(n)$ space, $O(n)$ prep. time and $O(1)$ query time

Block decomposition

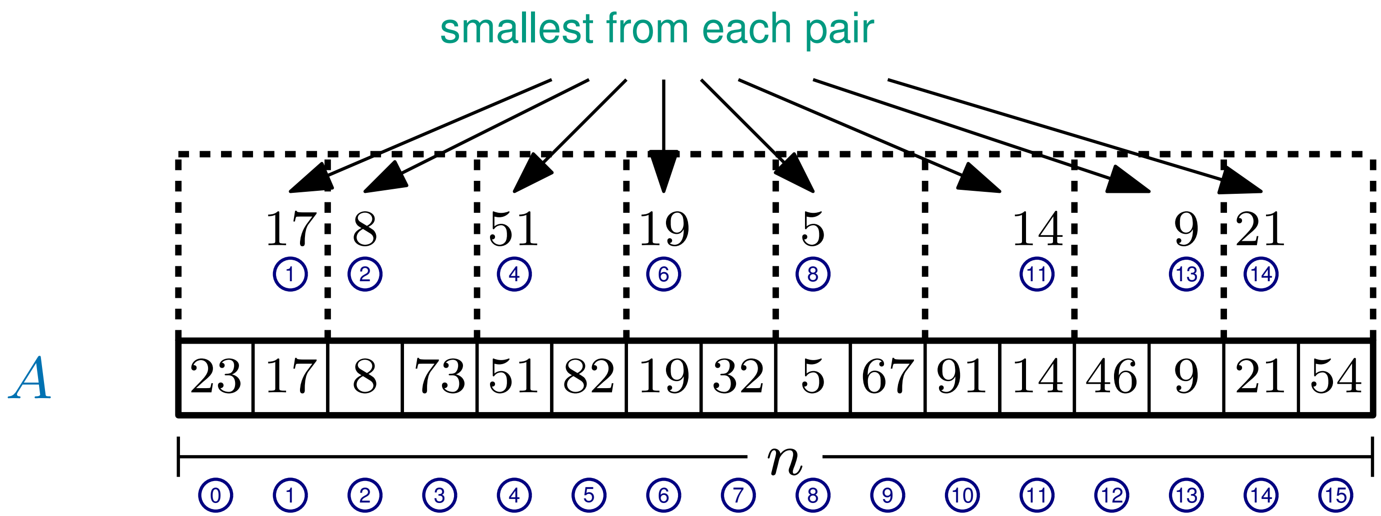
A	23	17	8	73	51	82	19	32	5	67	91	14	46	9	21	54
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n

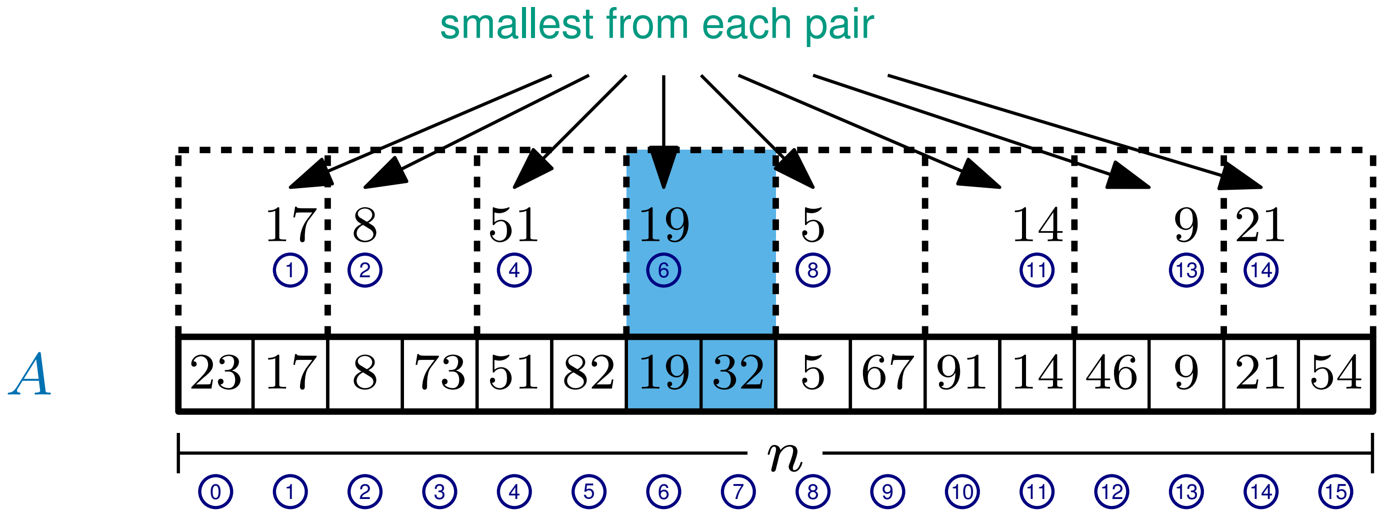
Block decomposition



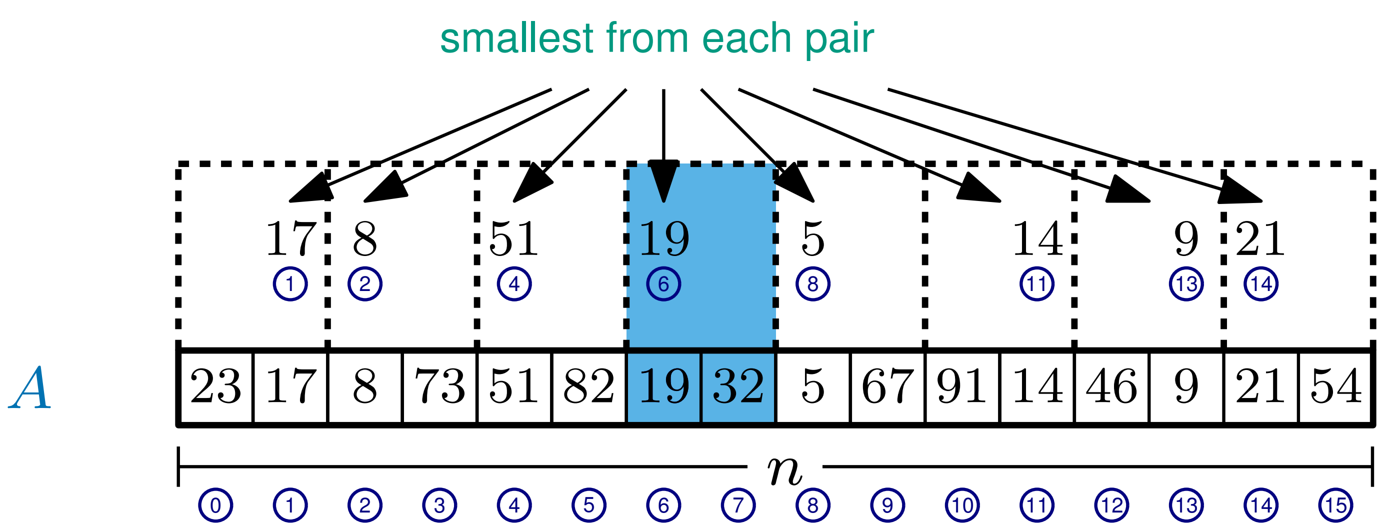
Block decomposition



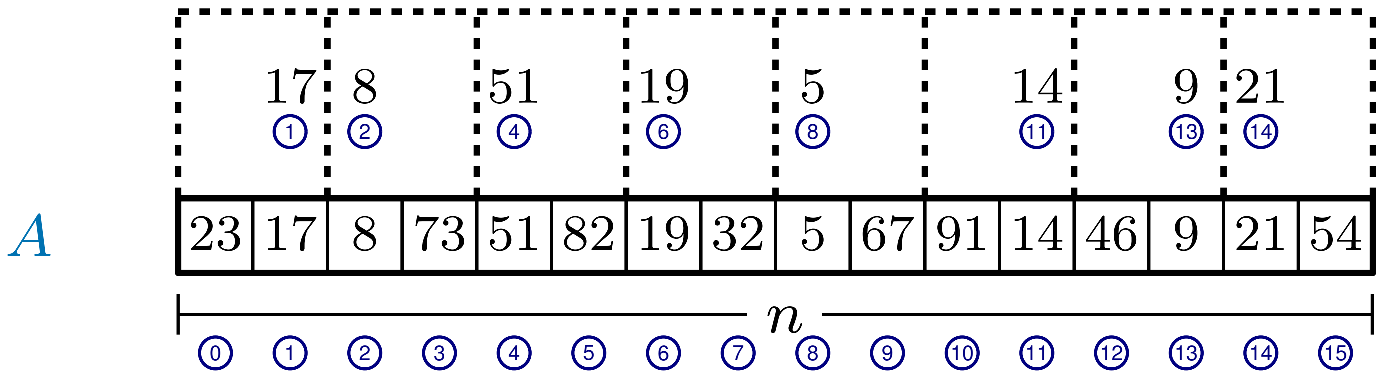
Block decomposition



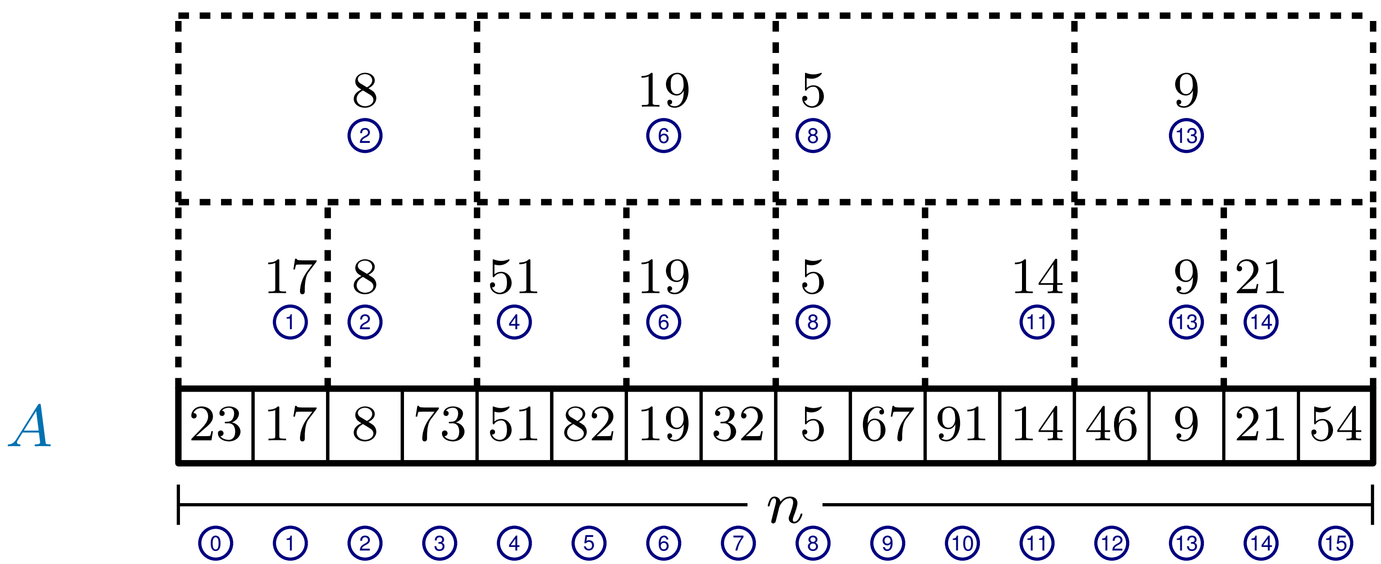
Block decomposition



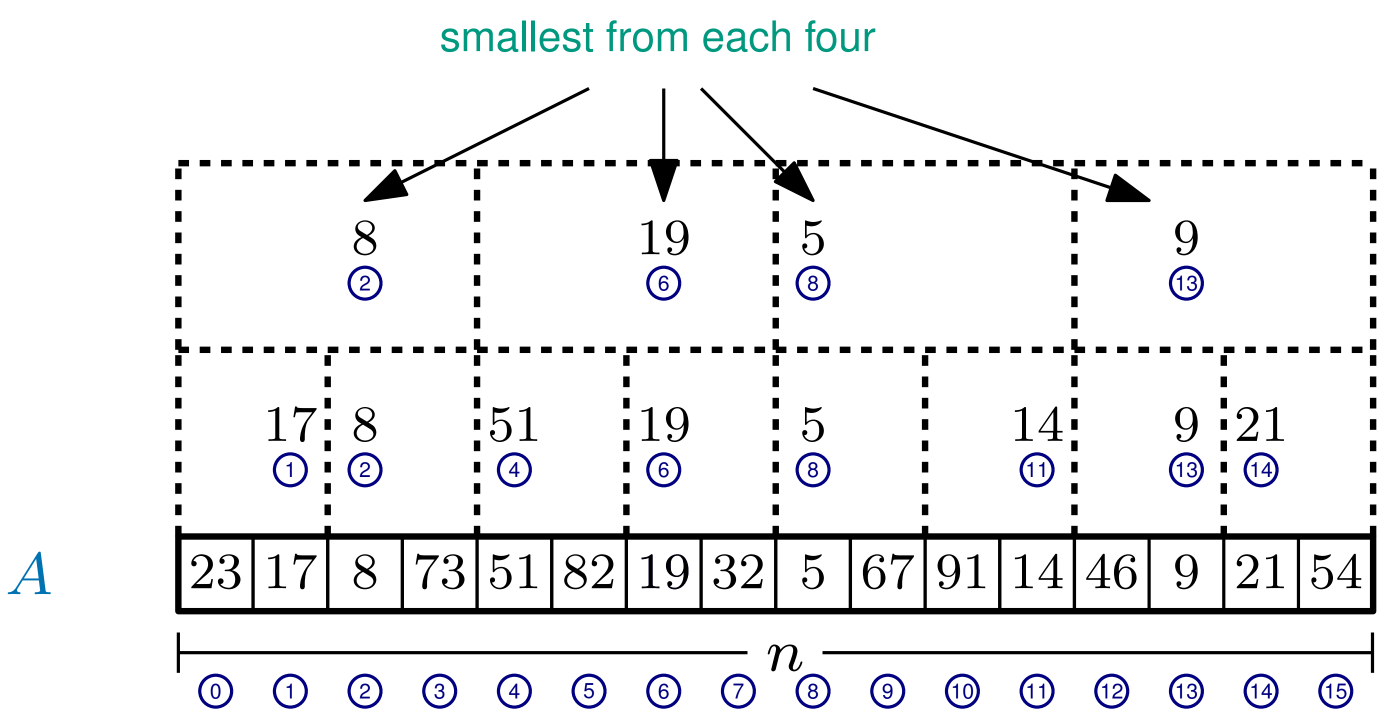
Block decomposition



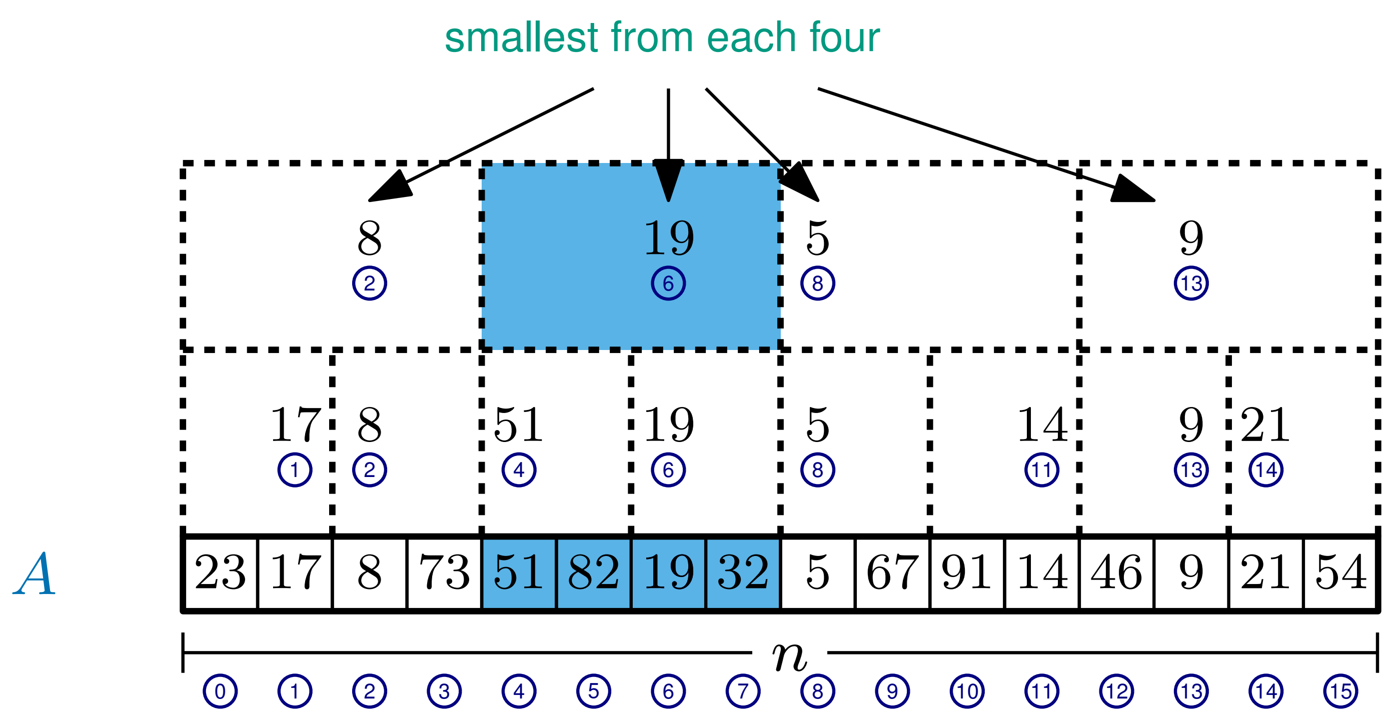
Block decomposition



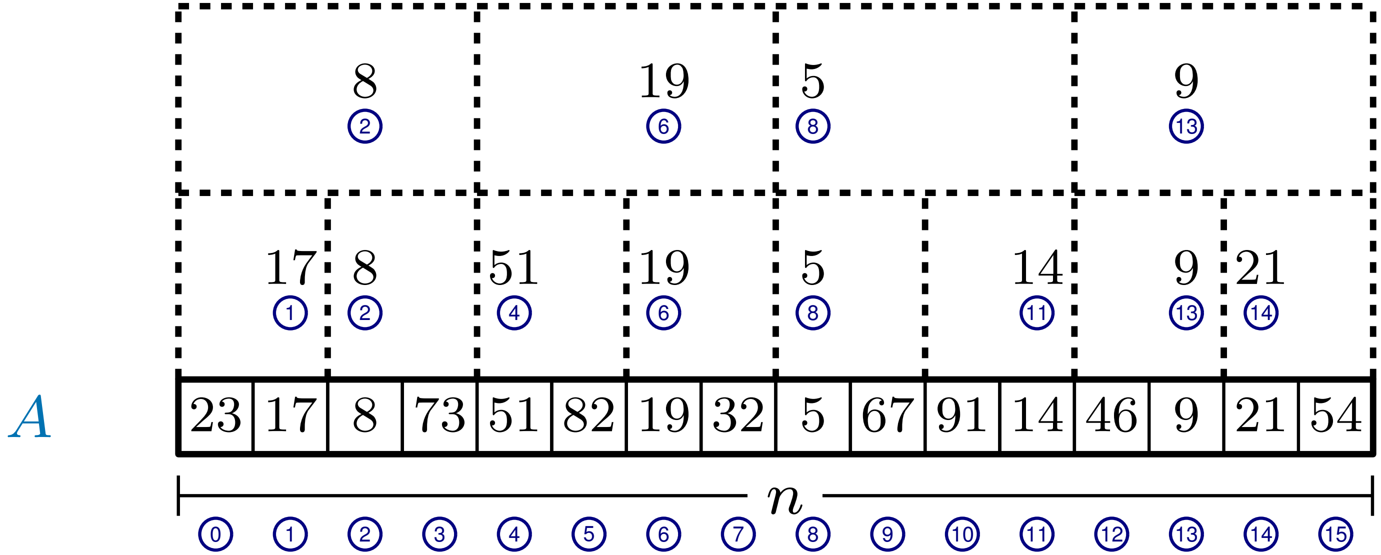
Block decomposition



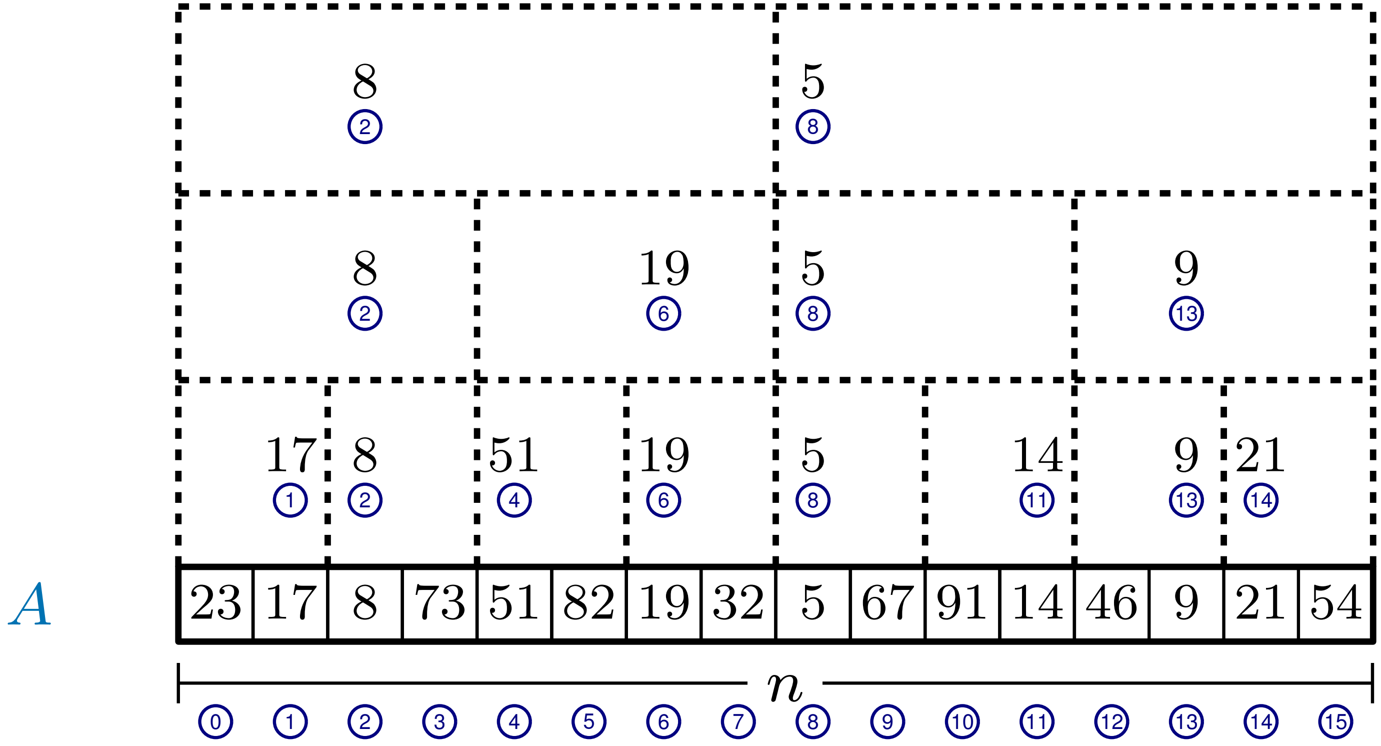
Block decomposition



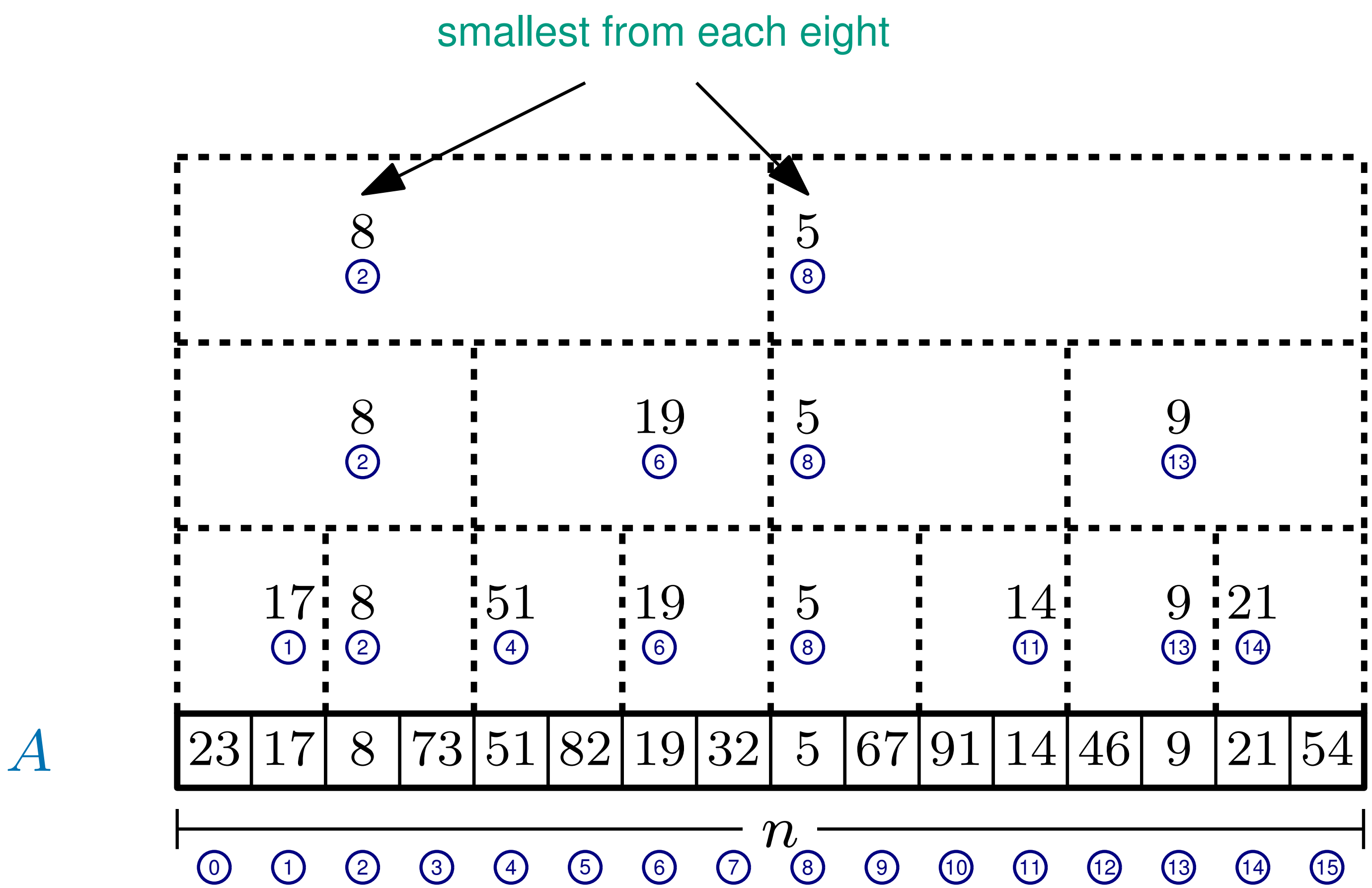
Block decomposition



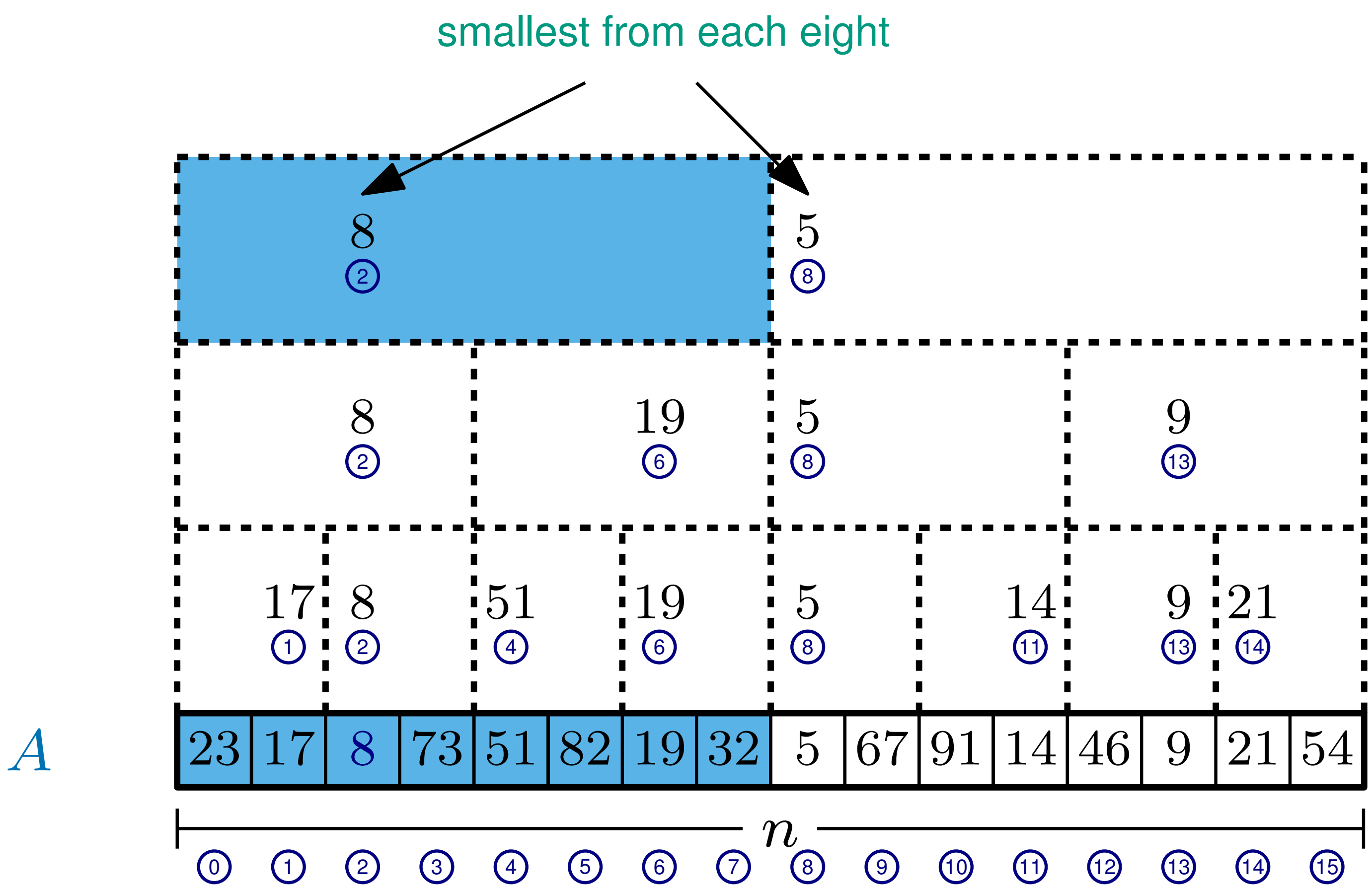
Block decomposition



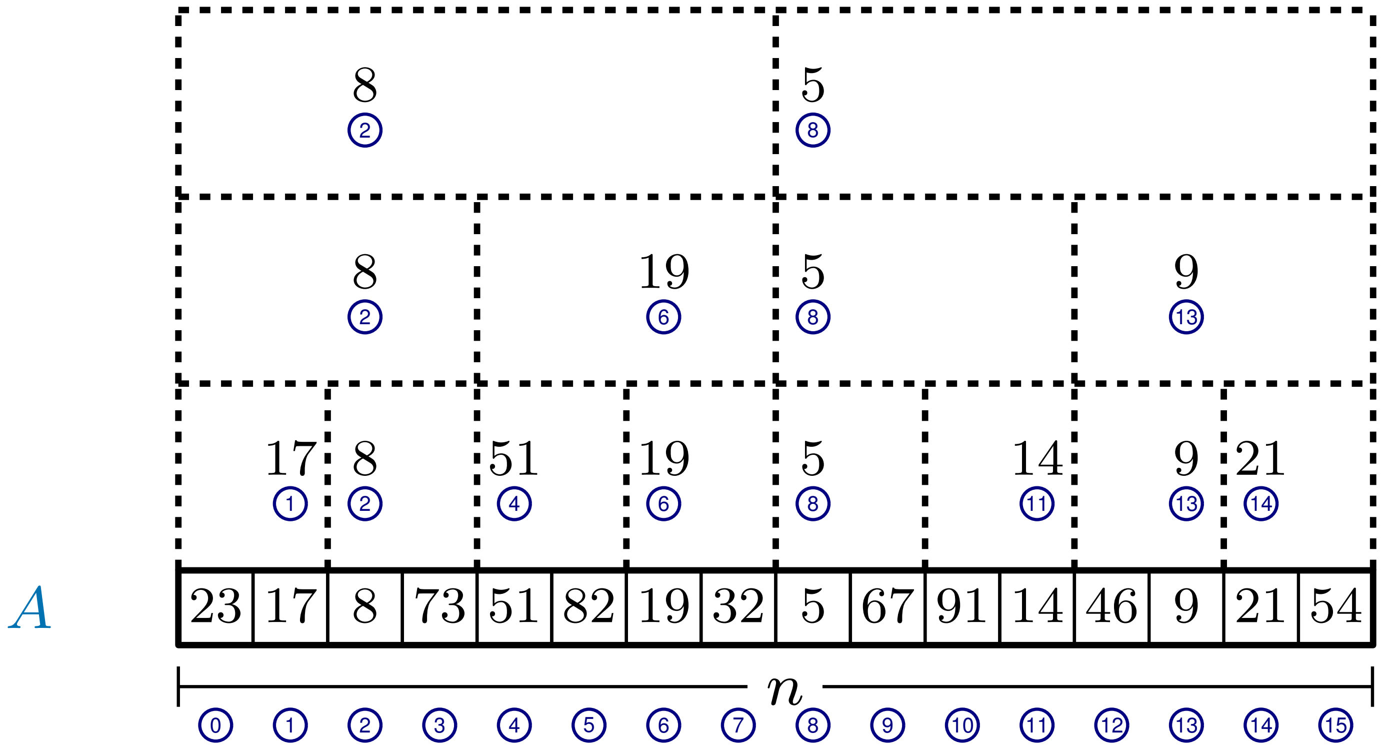
Block decomposition



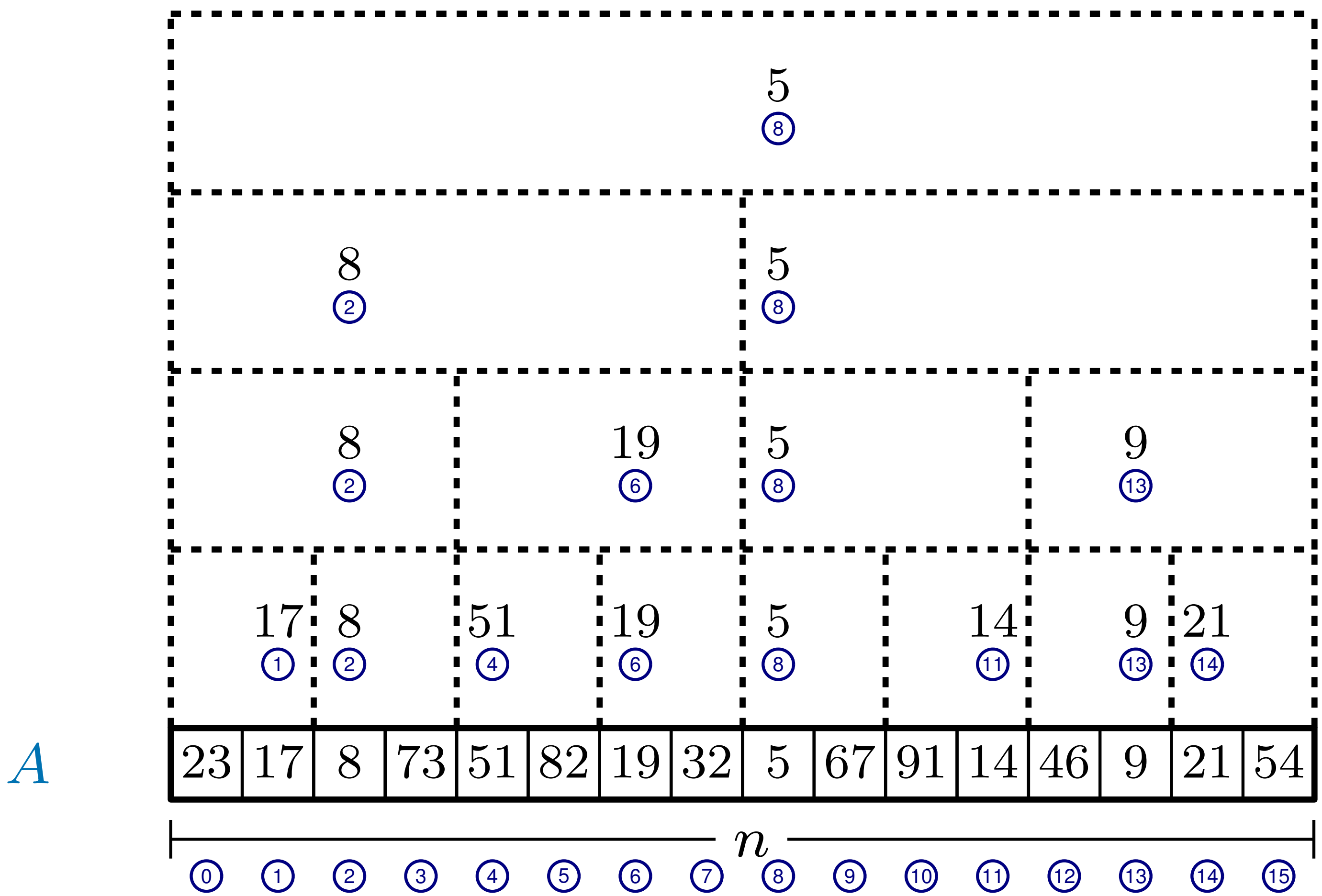
Block decomposition



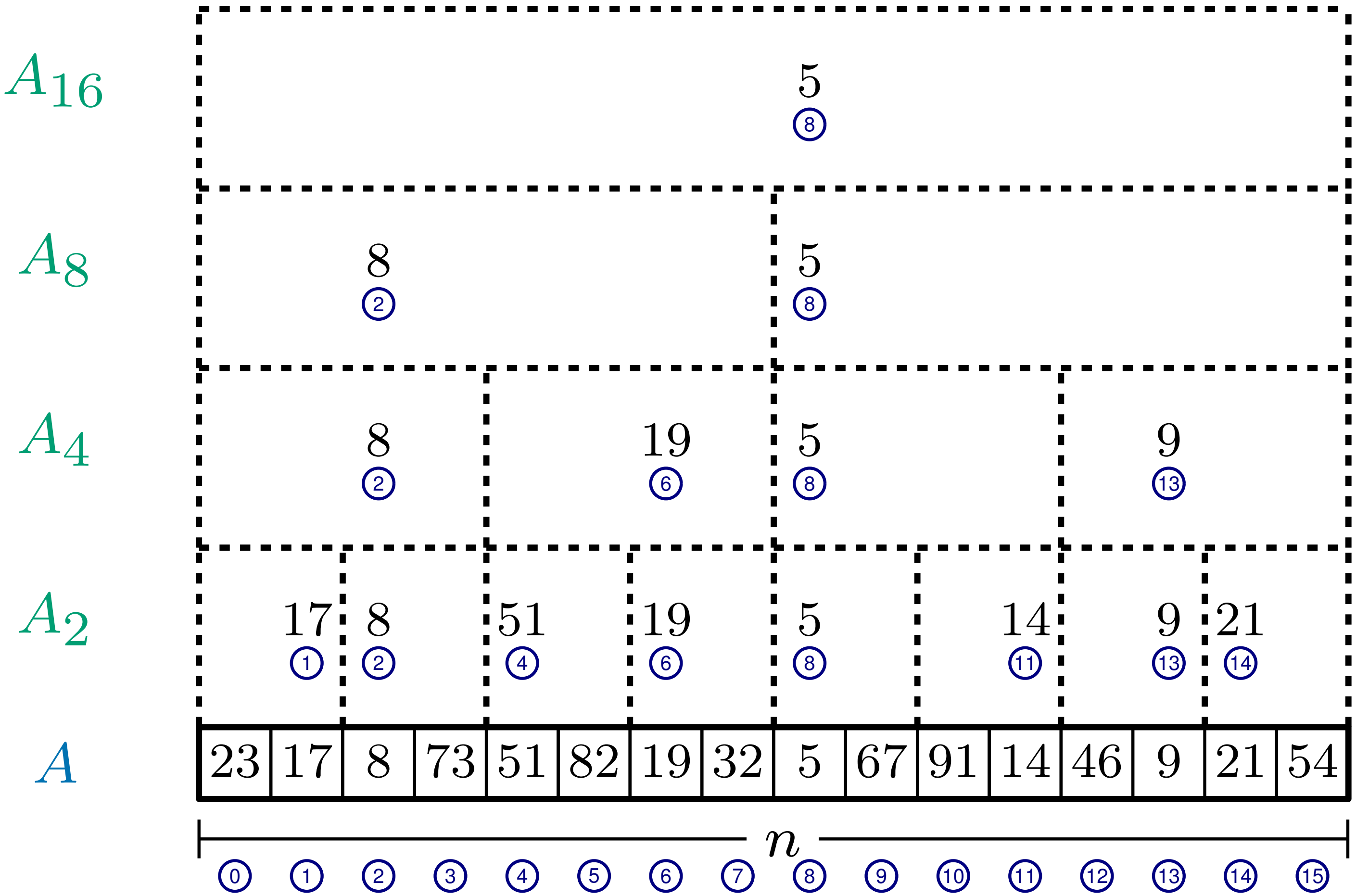
Block decomposition



Block decomposition

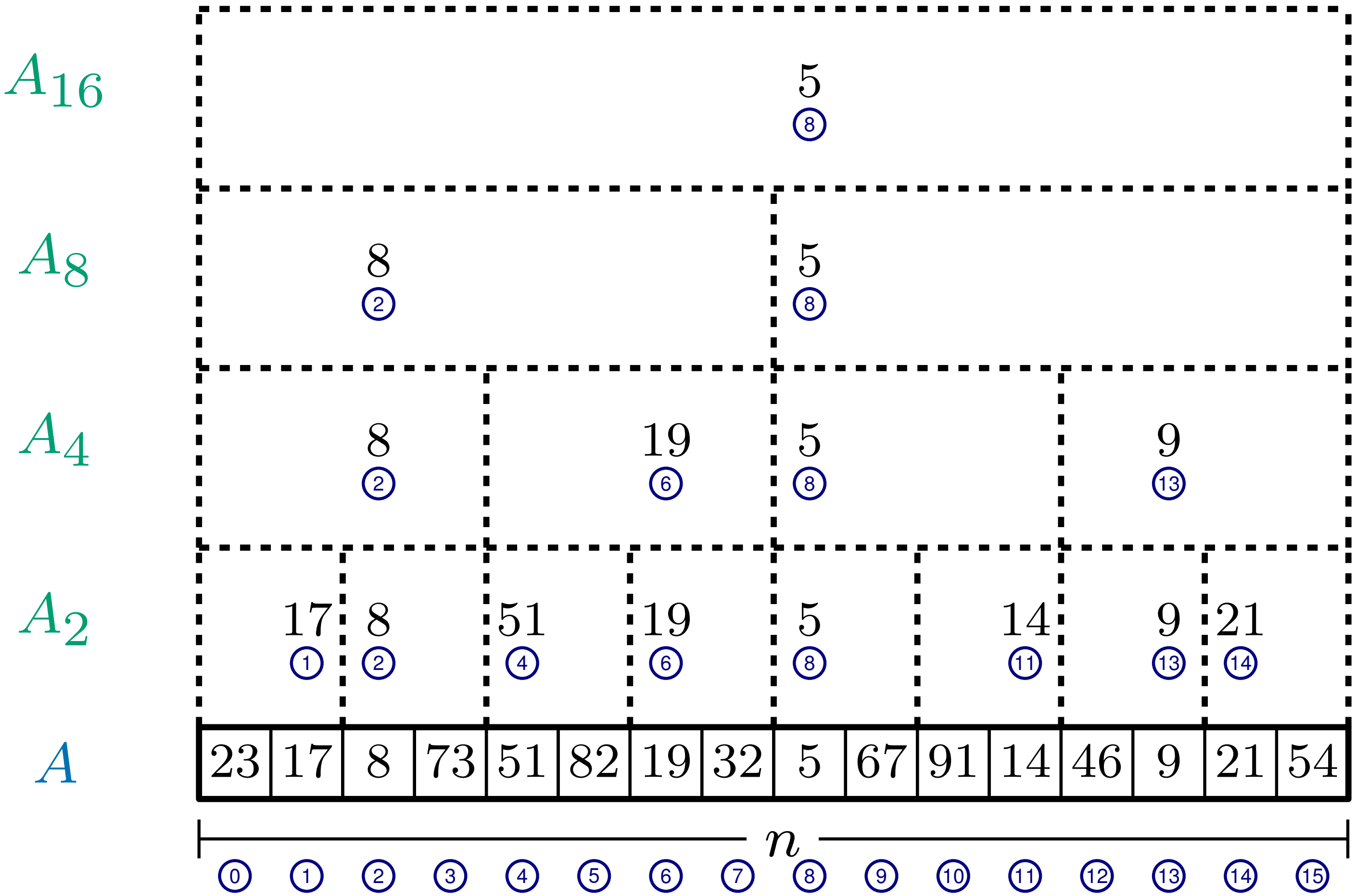


Block decomposition



Block decomposition

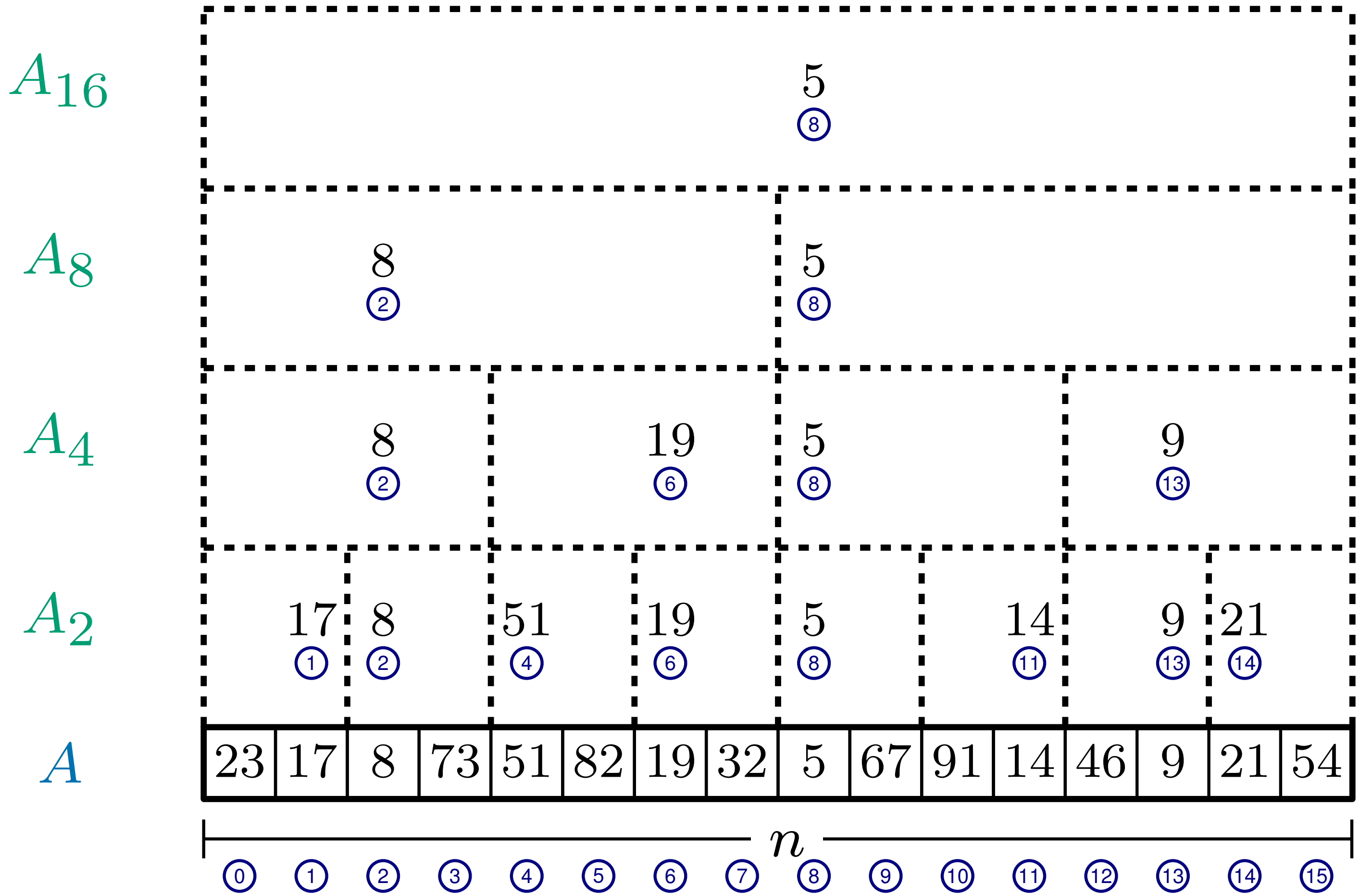
A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.



Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

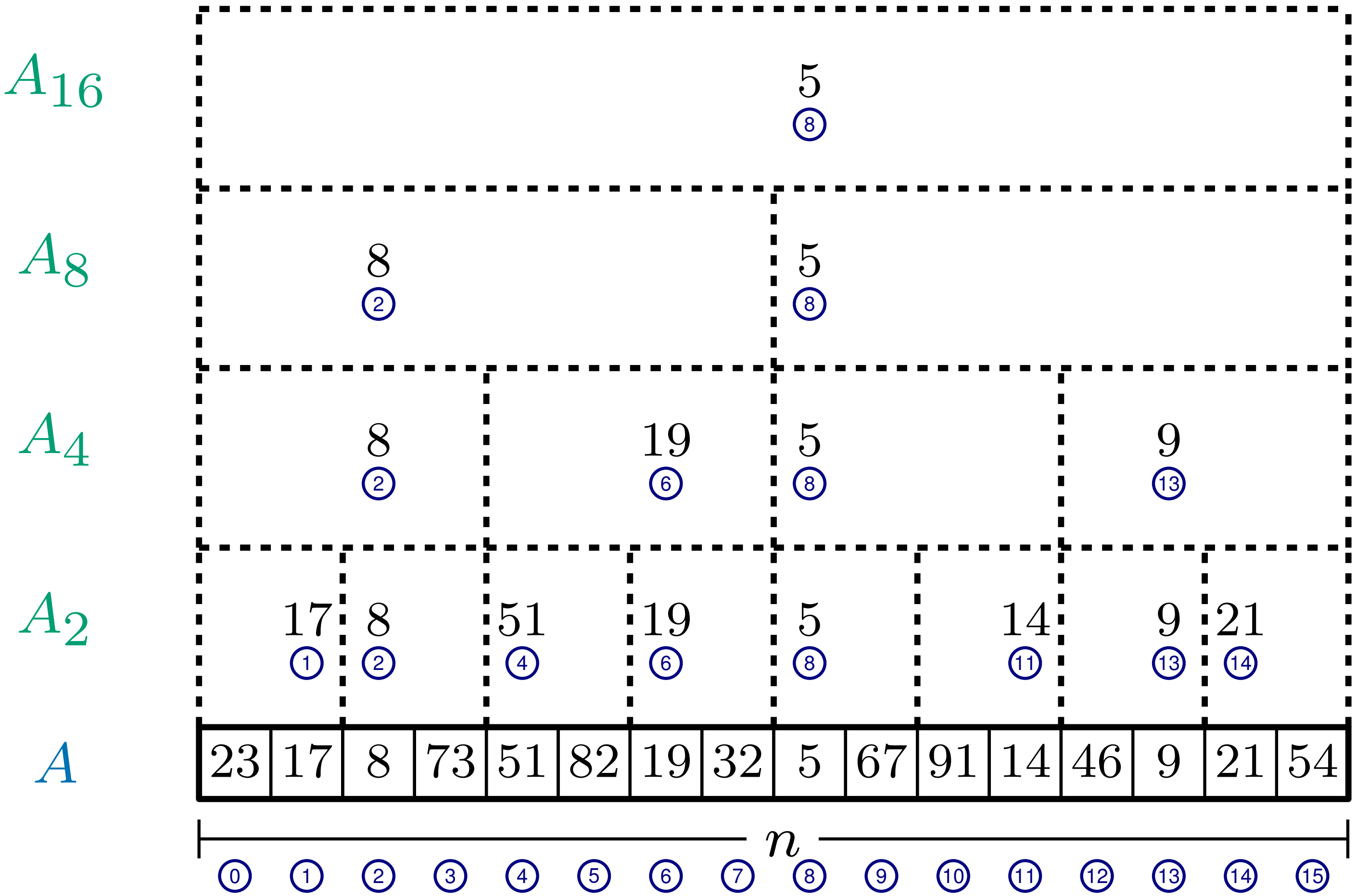


Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this?

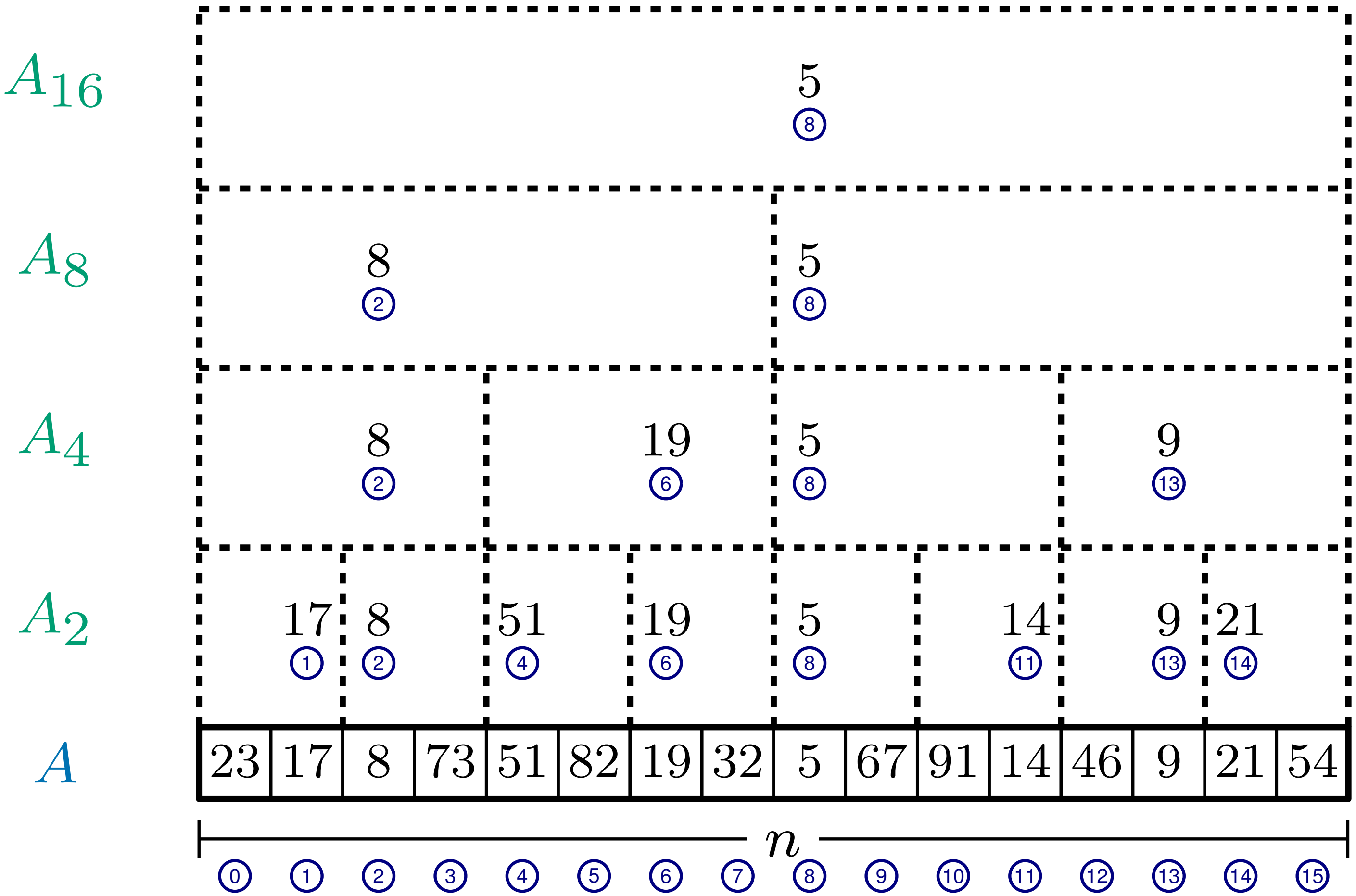


Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total

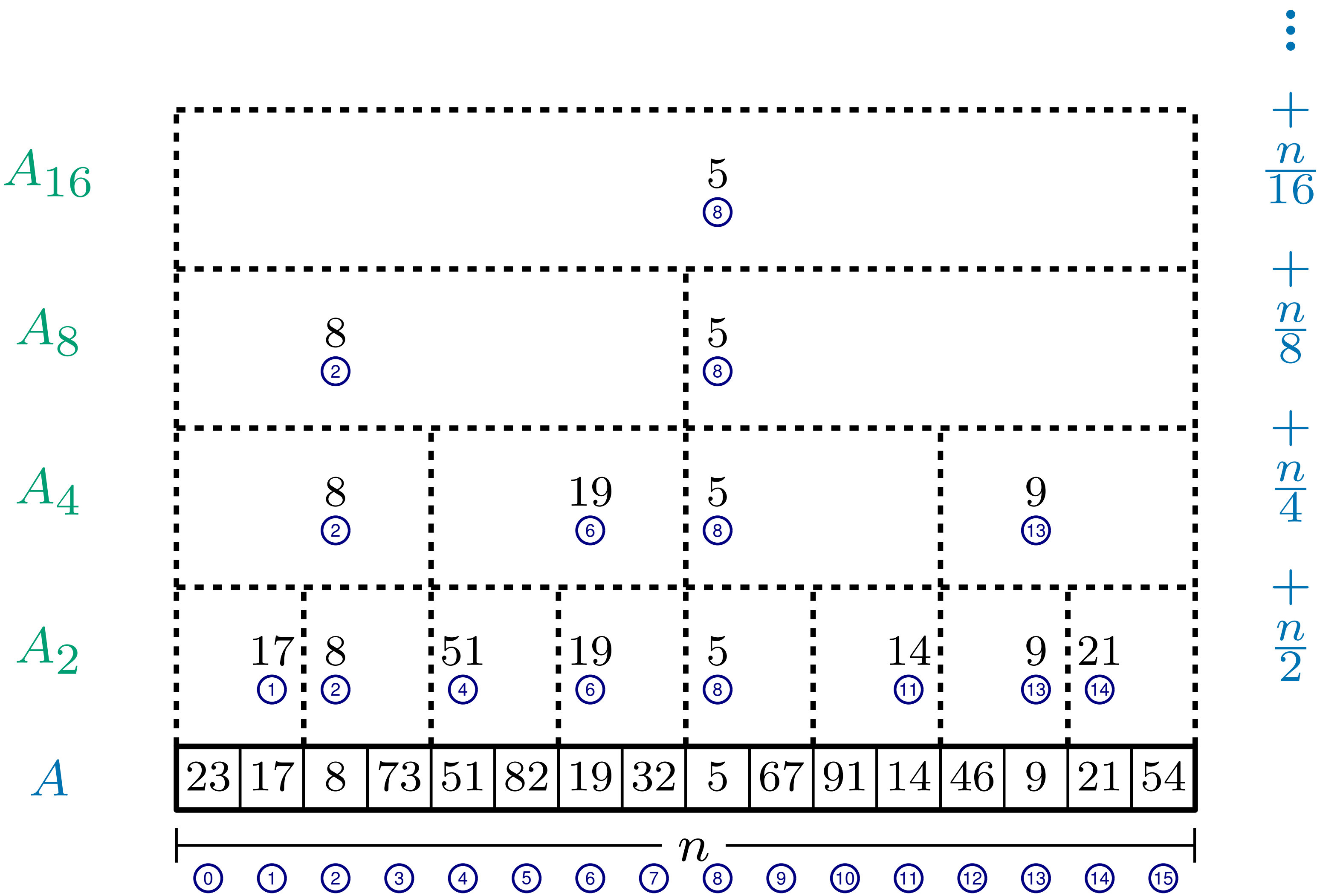


Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total



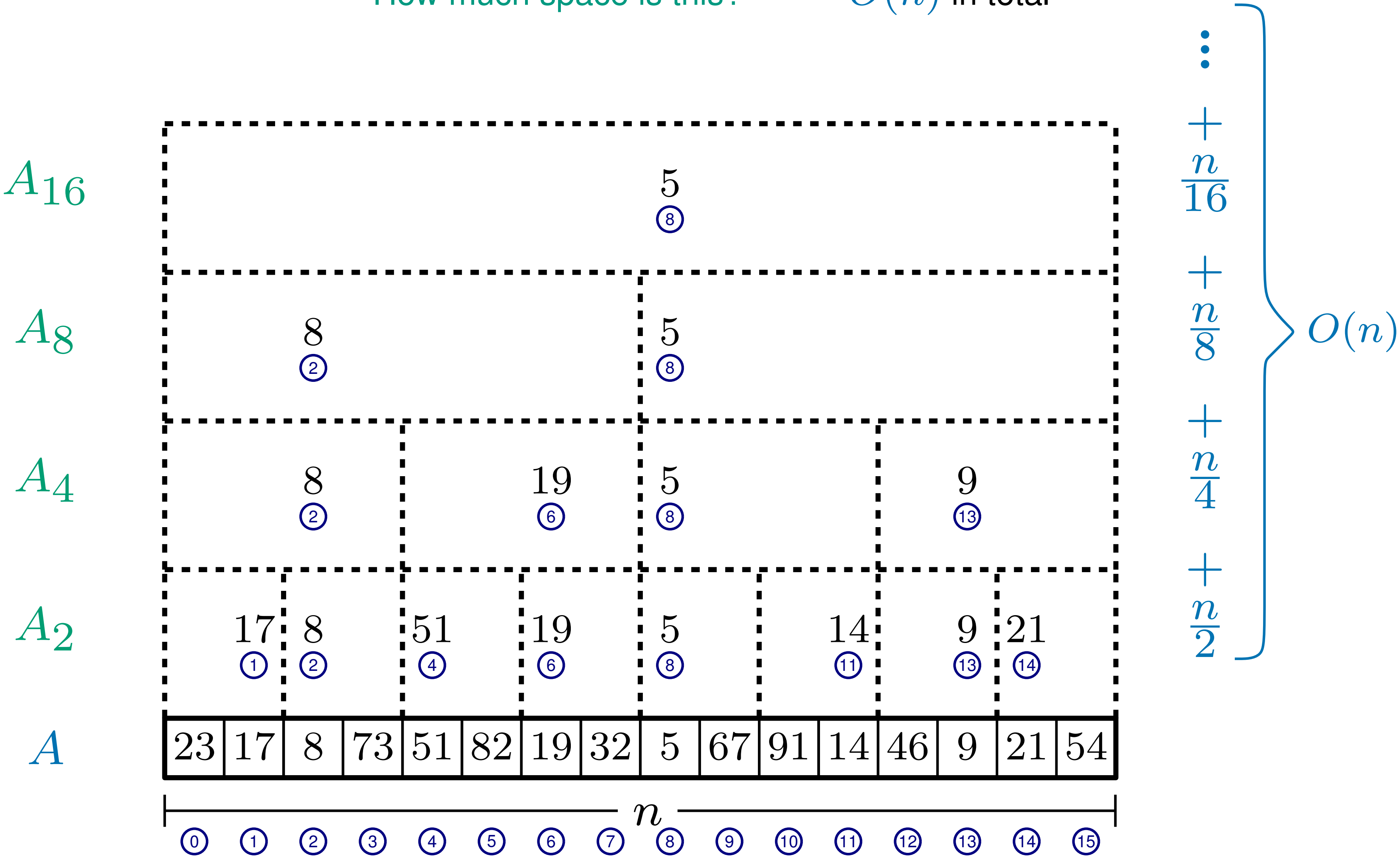
Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$

where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total



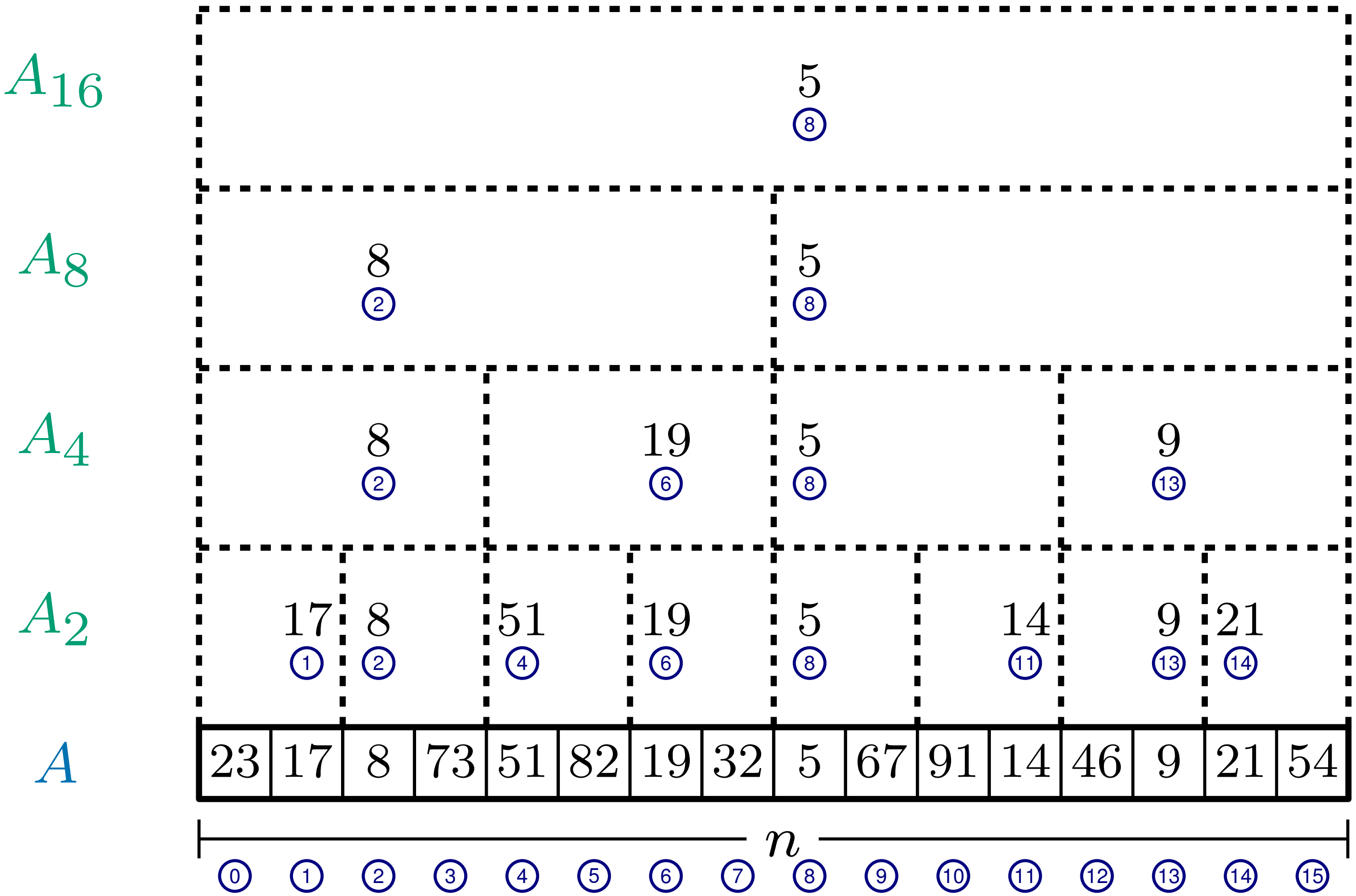
Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$

where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total



Block decomposition

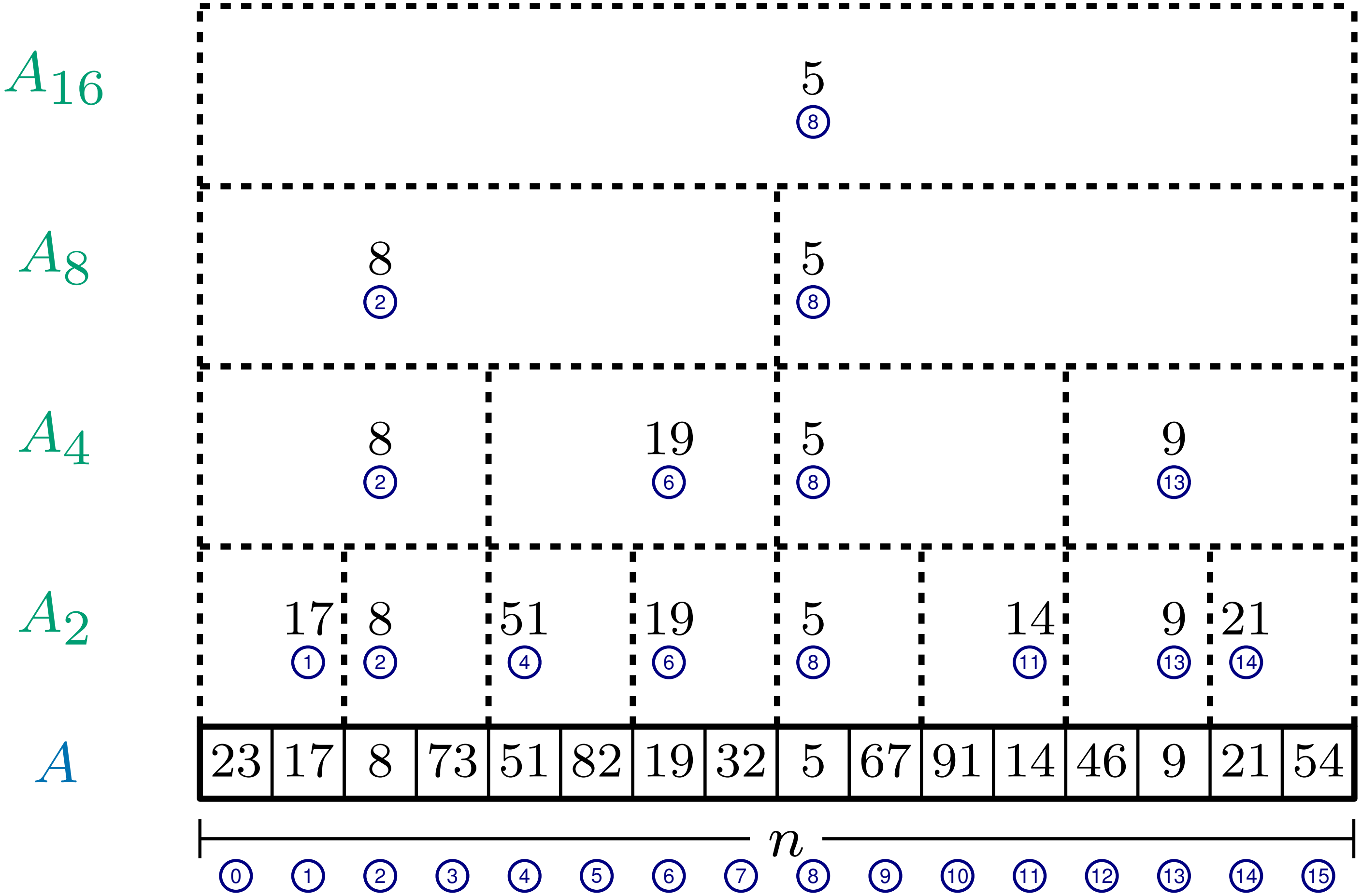
A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$

where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total

How quickly can we build them?



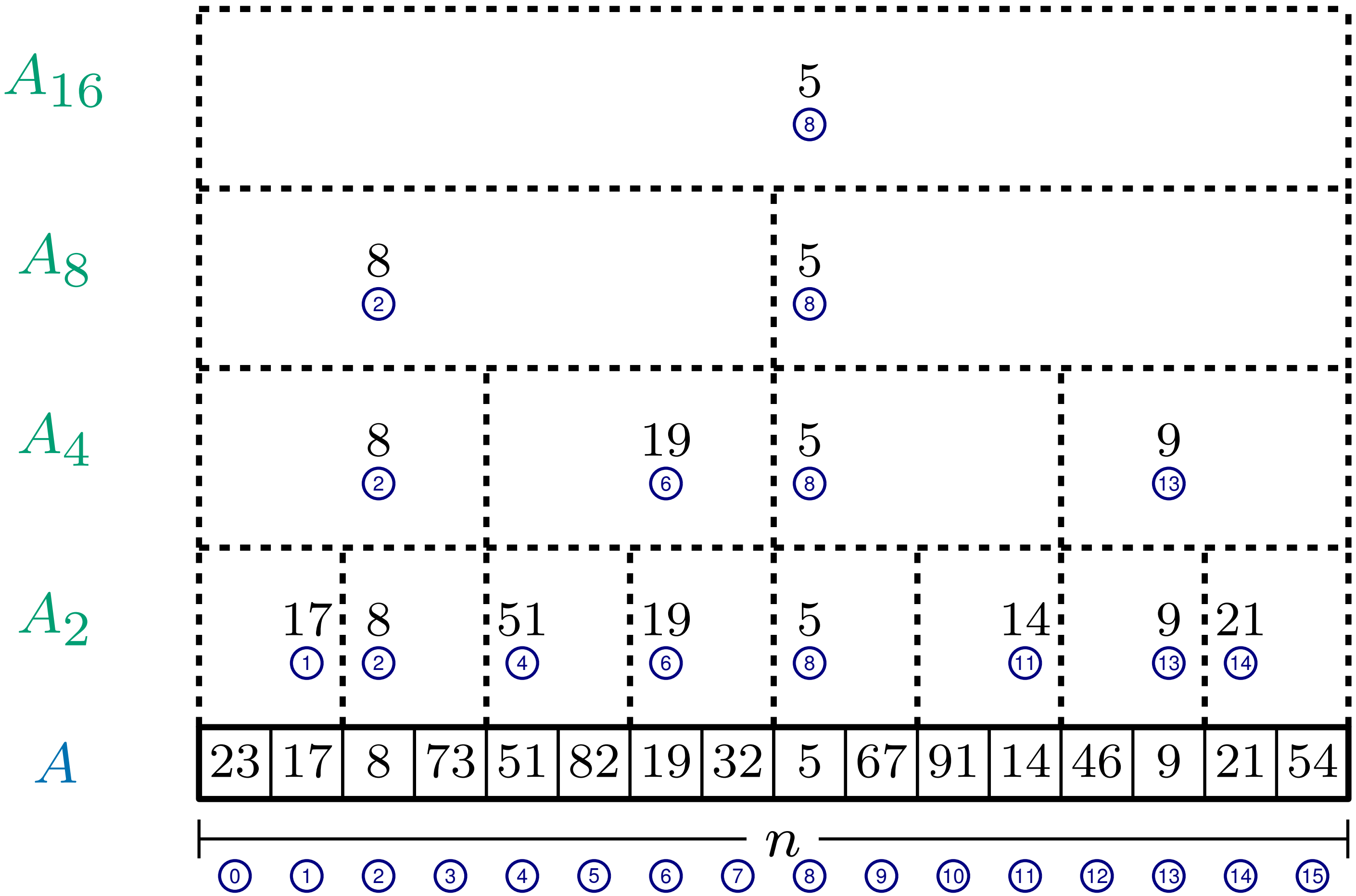
Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total

How quickly can we build them?



construct the A_k arrays bottom-up

Block decomposition

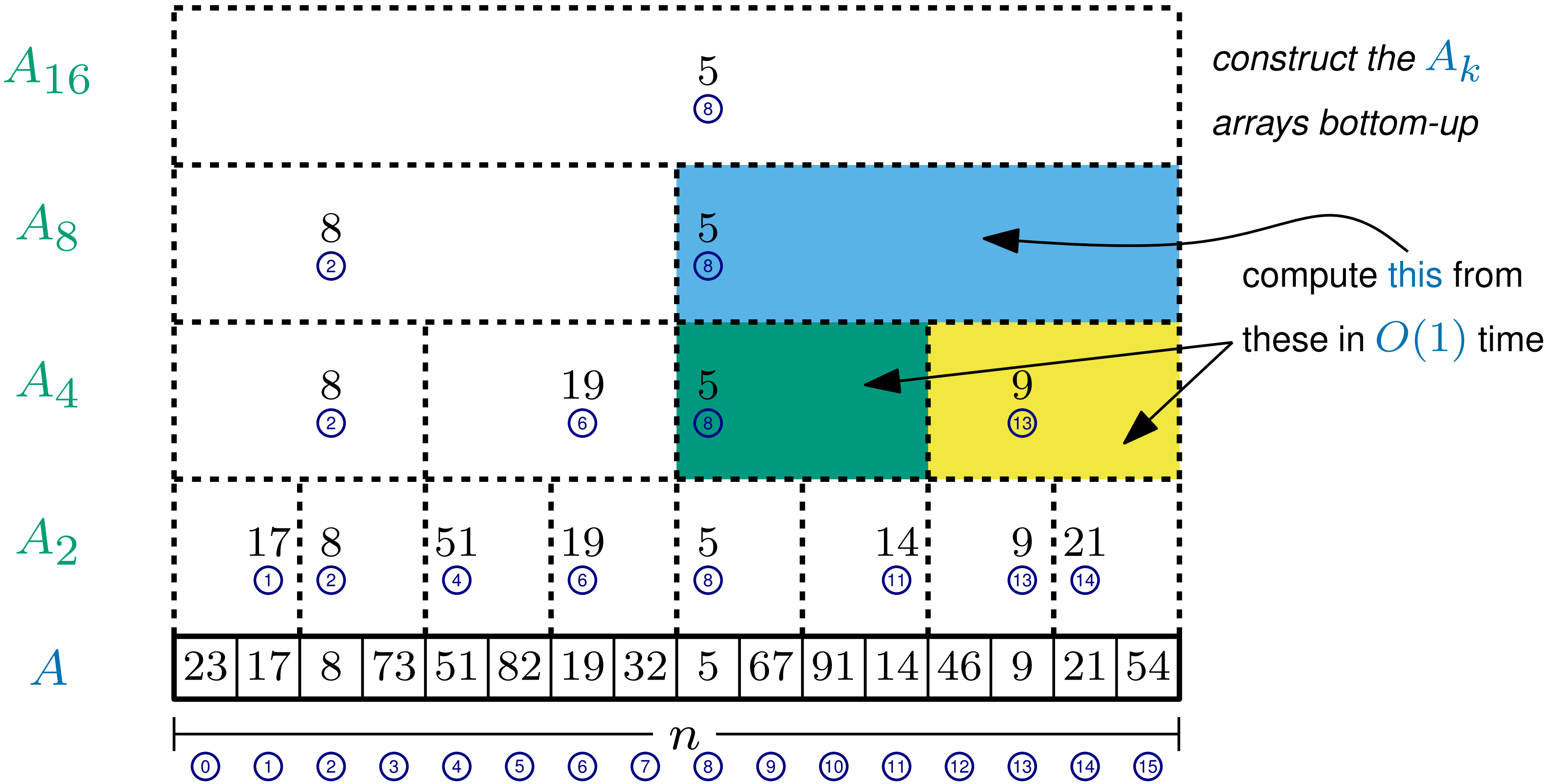
A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$

where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total

How quickly can we build them?



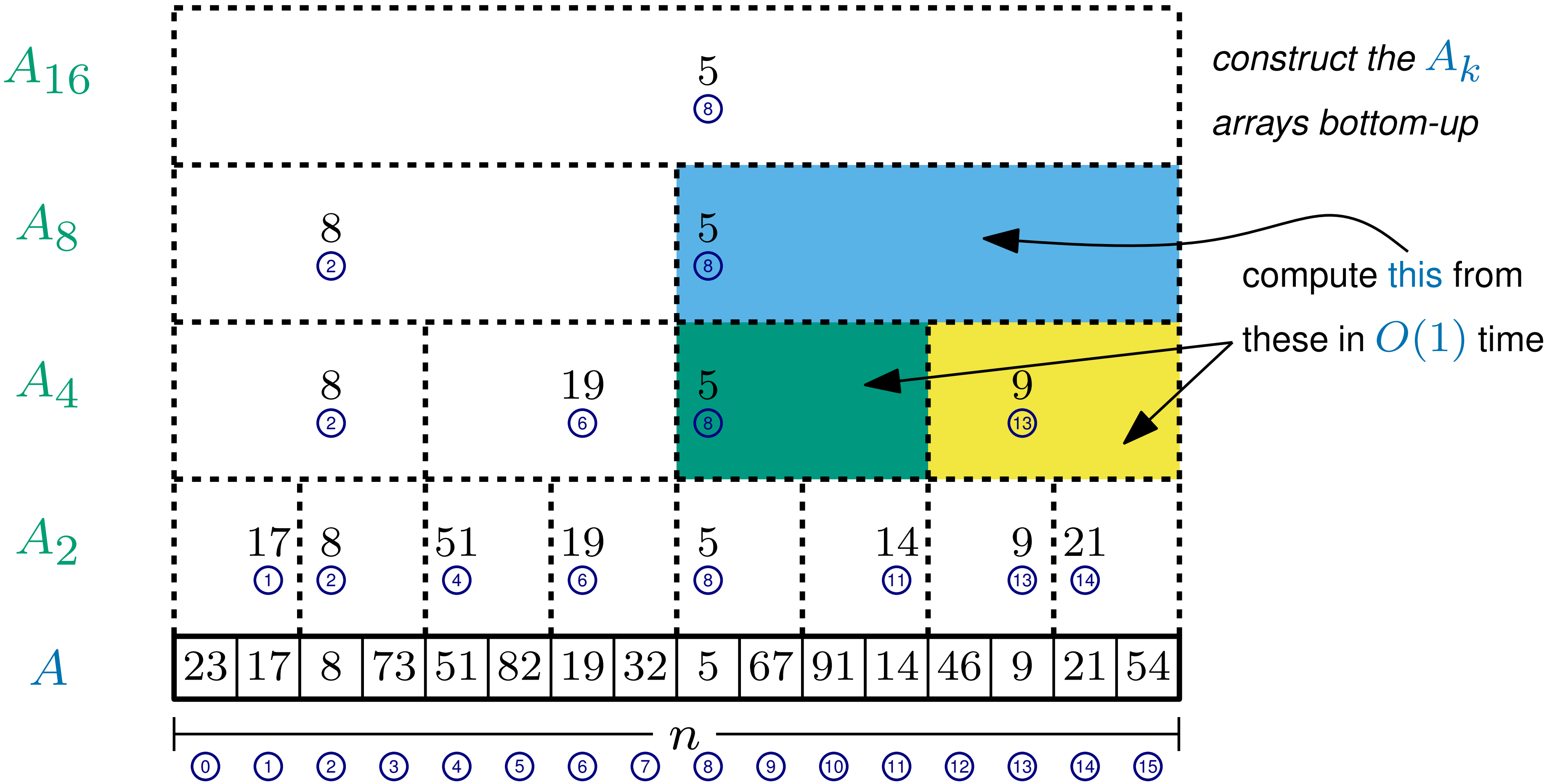
Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total

How quickly can we build them? $O(n)$ preprocessing time



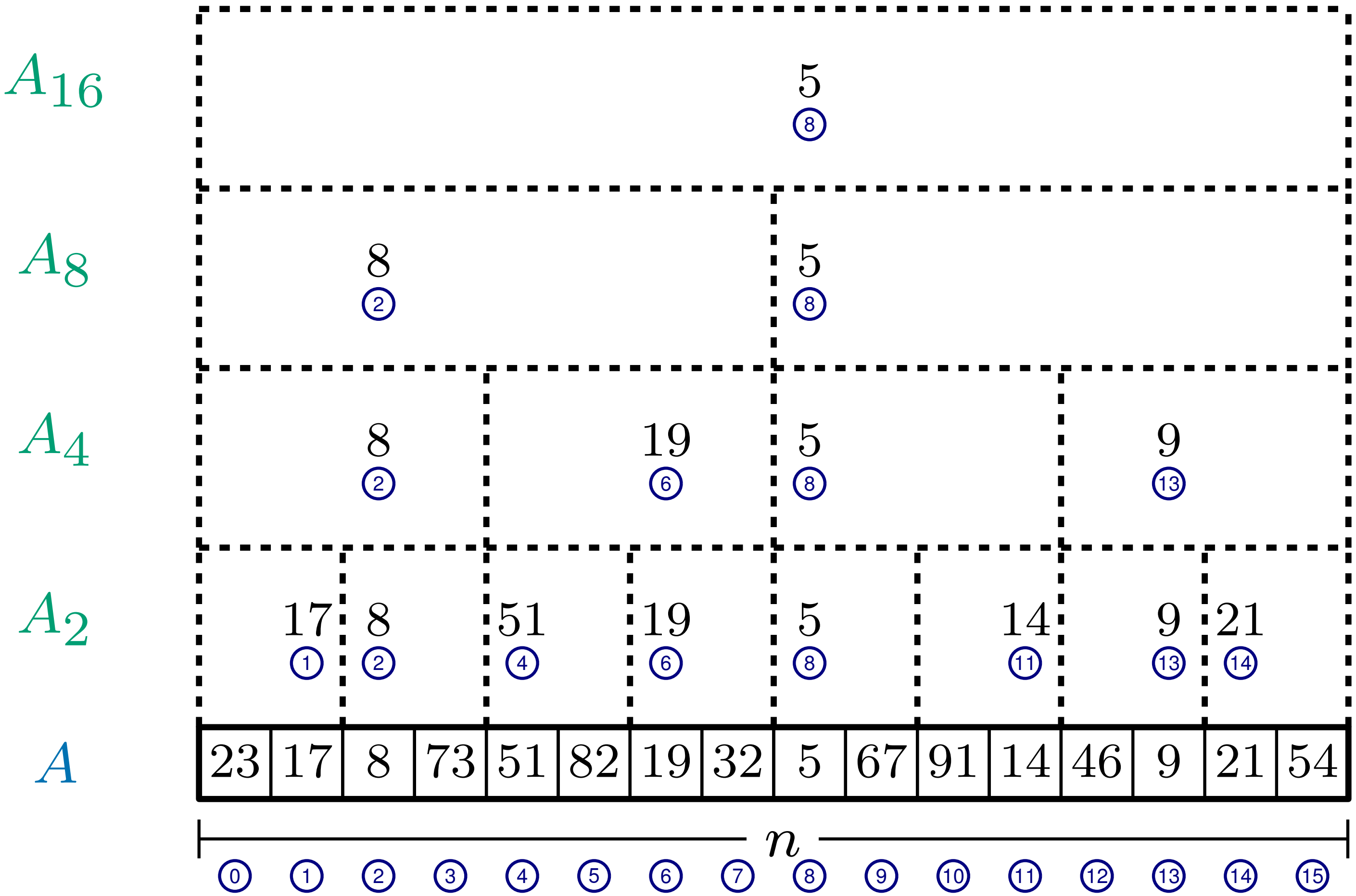
Block decomposition

A_k is an array of length $\frac{n}{k}$ so that for all i: $A_k[i] = (x, v)$
where v is the minimum in $A[ik, (i+1)k]$ and x is its location in A.

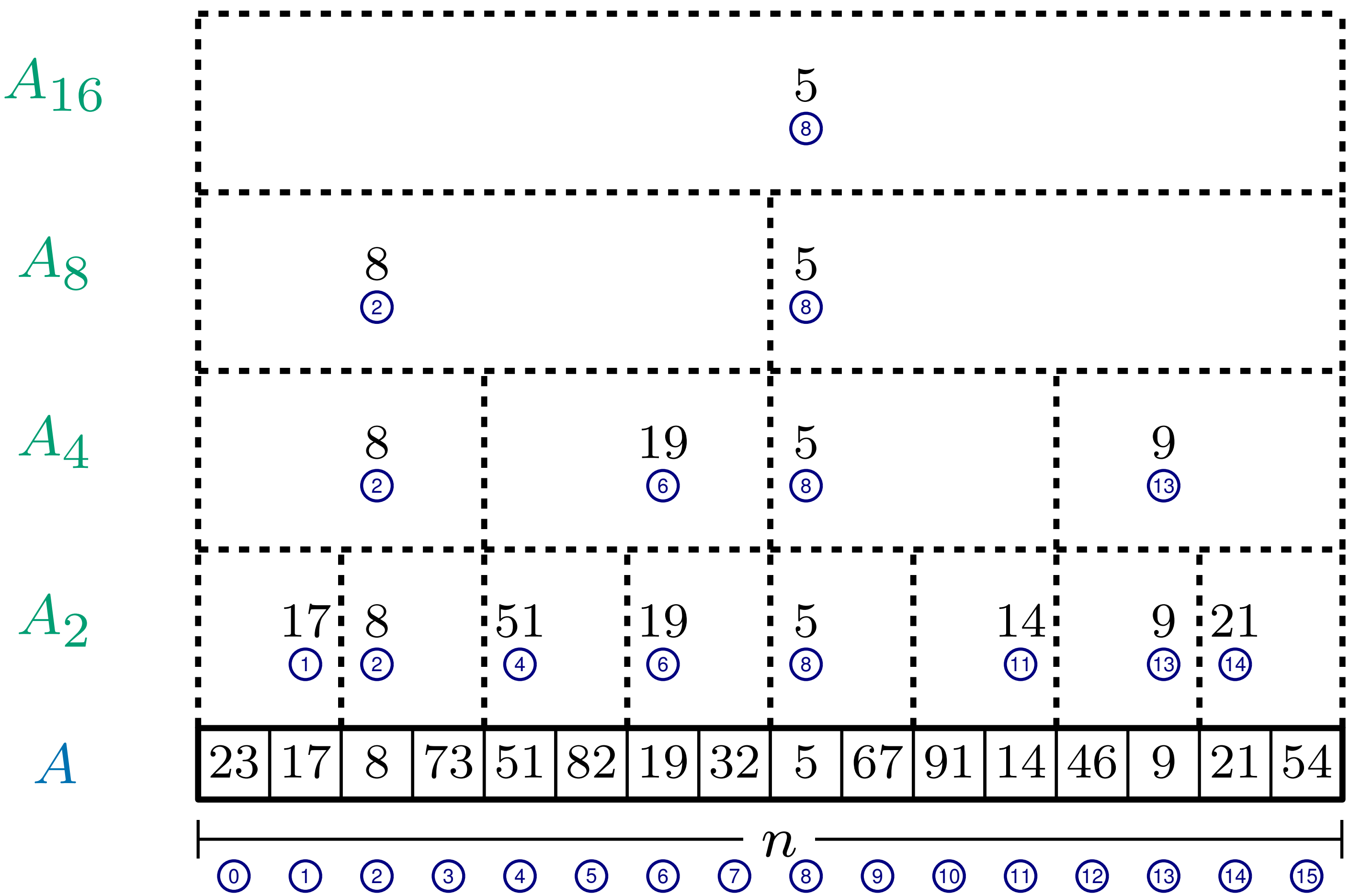
We store A_k for all $k = 1, 2, 4, 8 \ldots \leqslant n$

How much space is this? $O(n)$ in total

How quickly can we build them? $O(n)$ preprocessing time

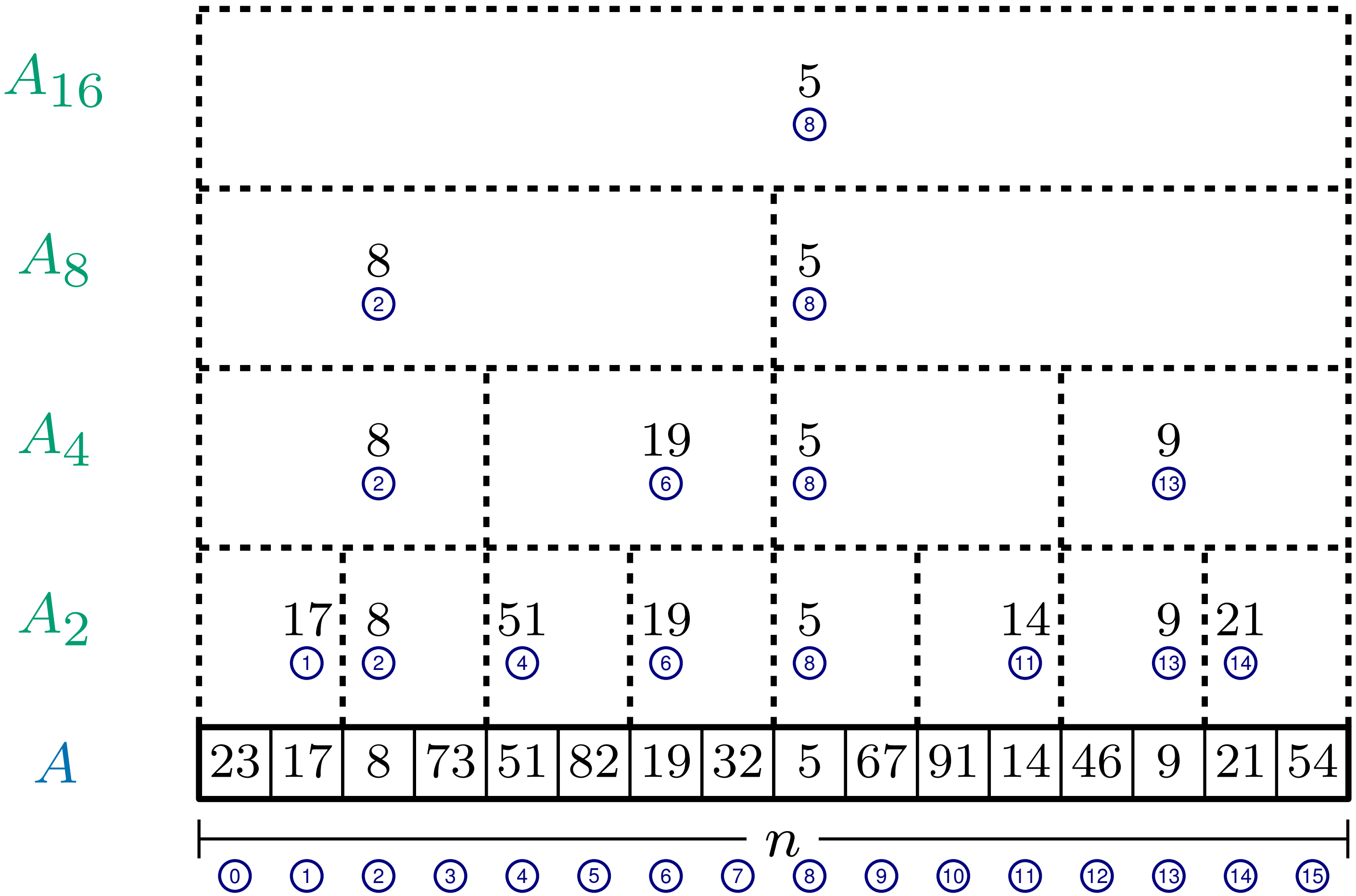


Block decomposition



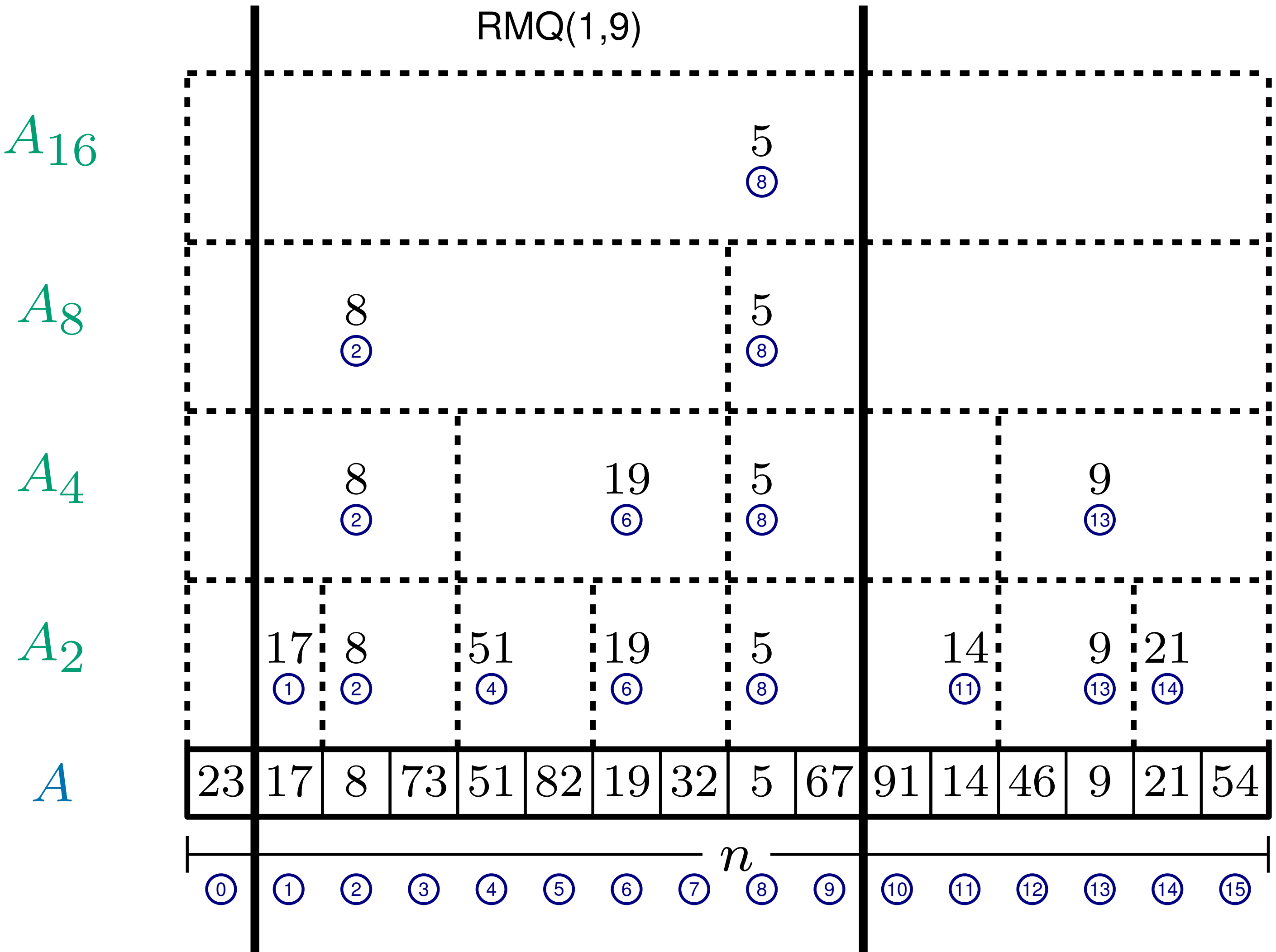
Block decomposition

How do we find RMQ(i,j)?



Block decomposition

How do we find RMQ(i,j)?

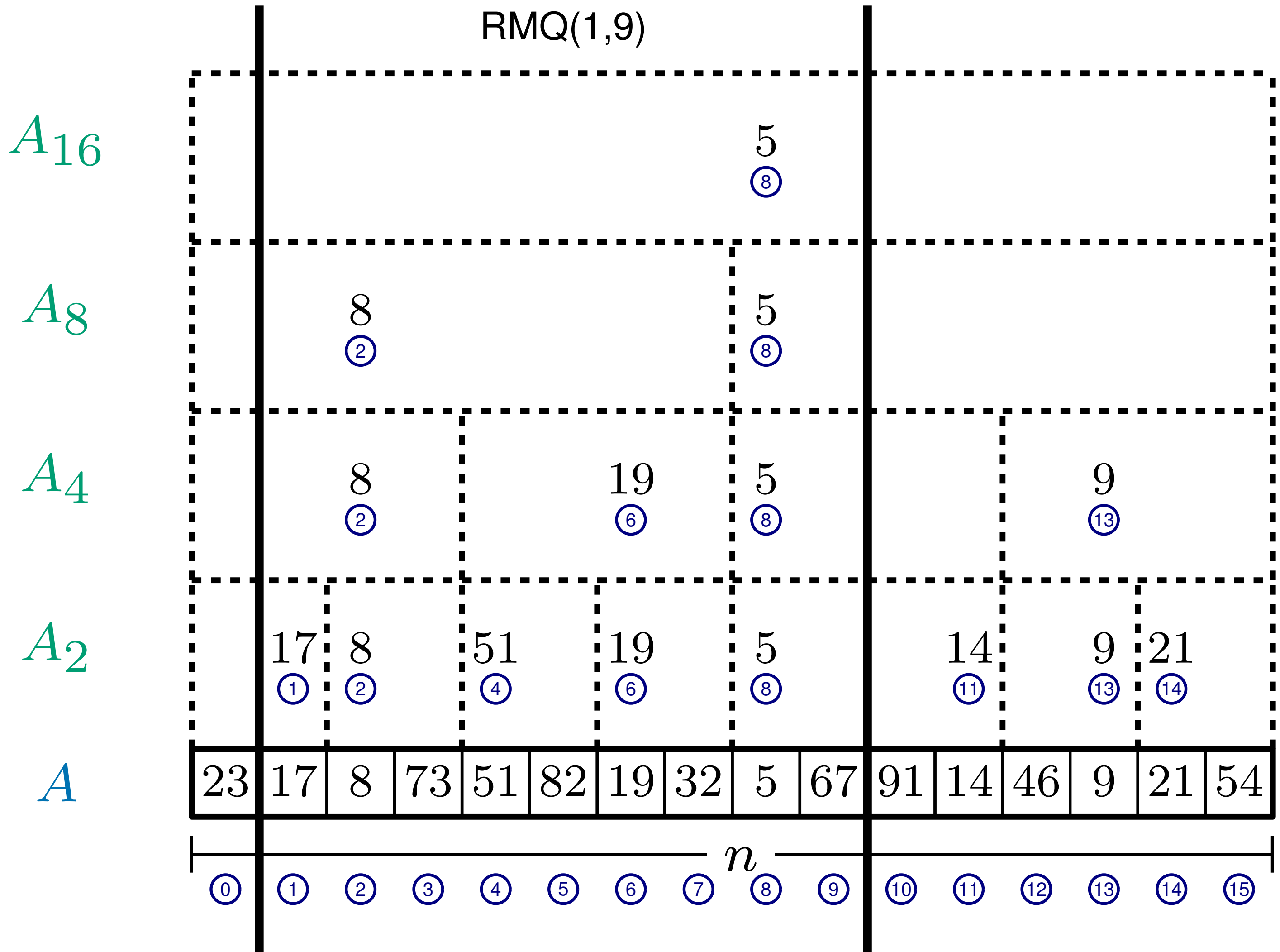


Block decomposition

How do we find RMQ(i,j)?

Find the largest *block* which

is completely contained within the query interval

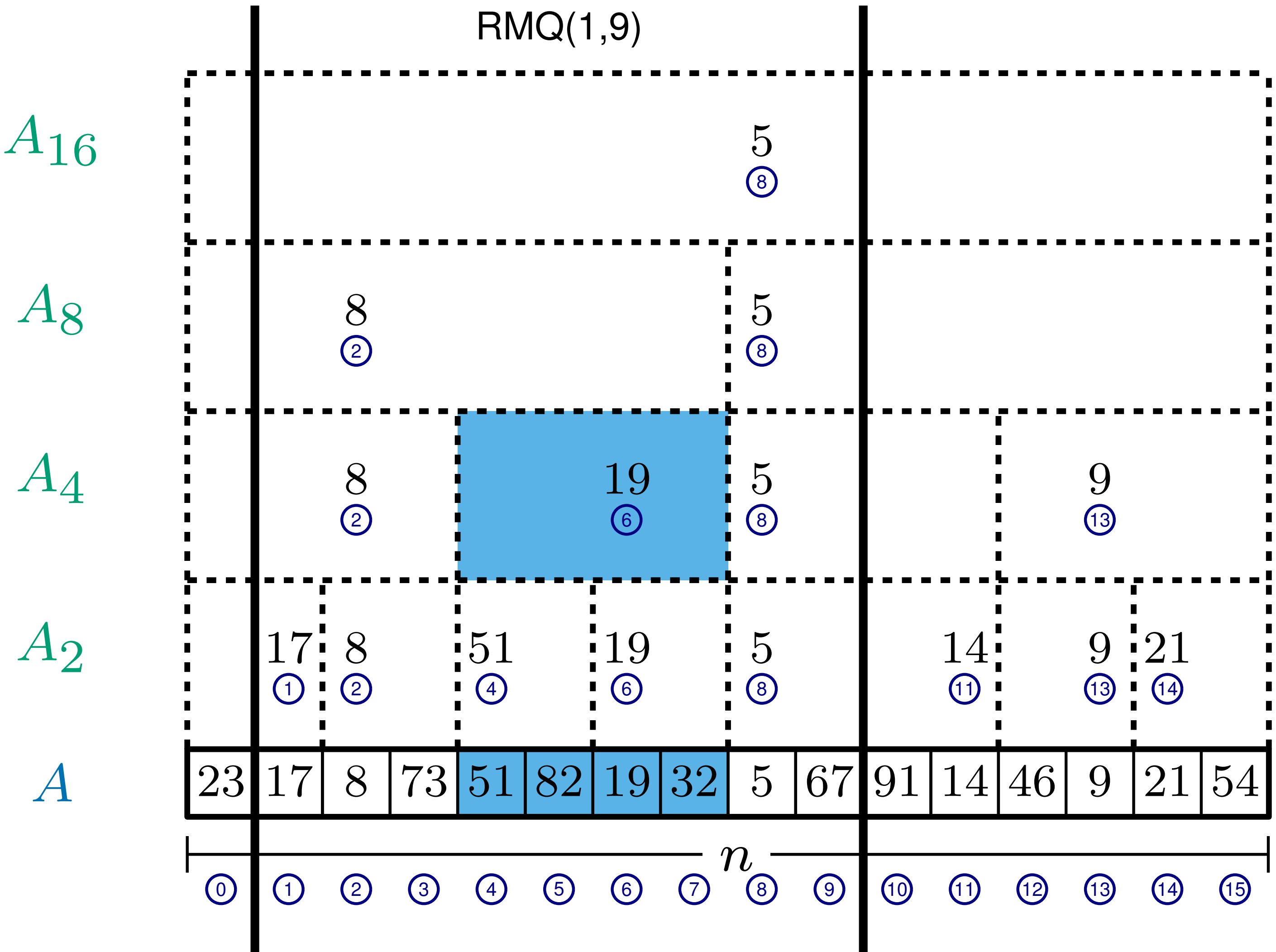


Block decomposition

How do we find RMQ(i,j)?

Find the largest *block* which

is completely contained within the query interval



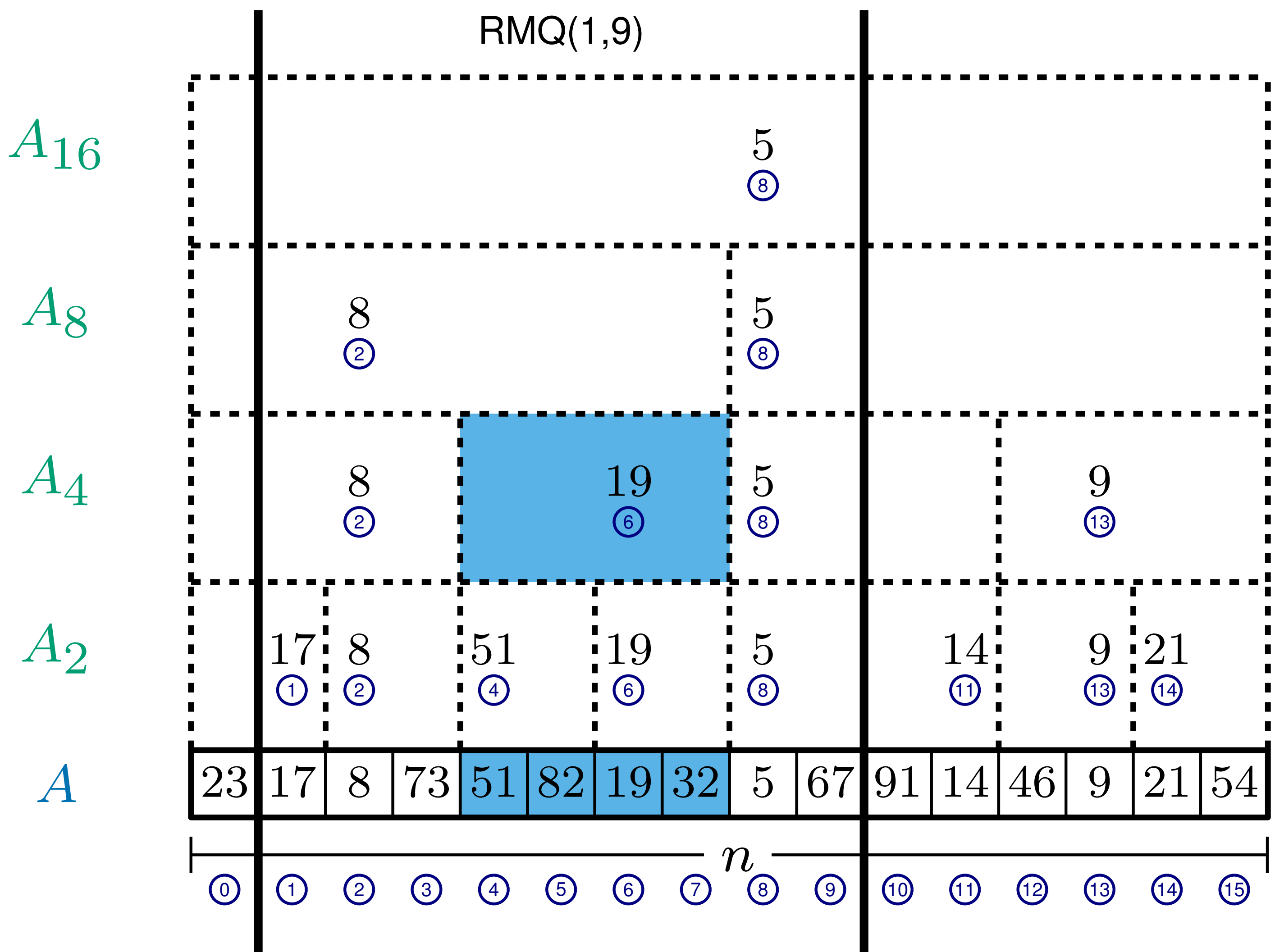
Block decomposition

How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before



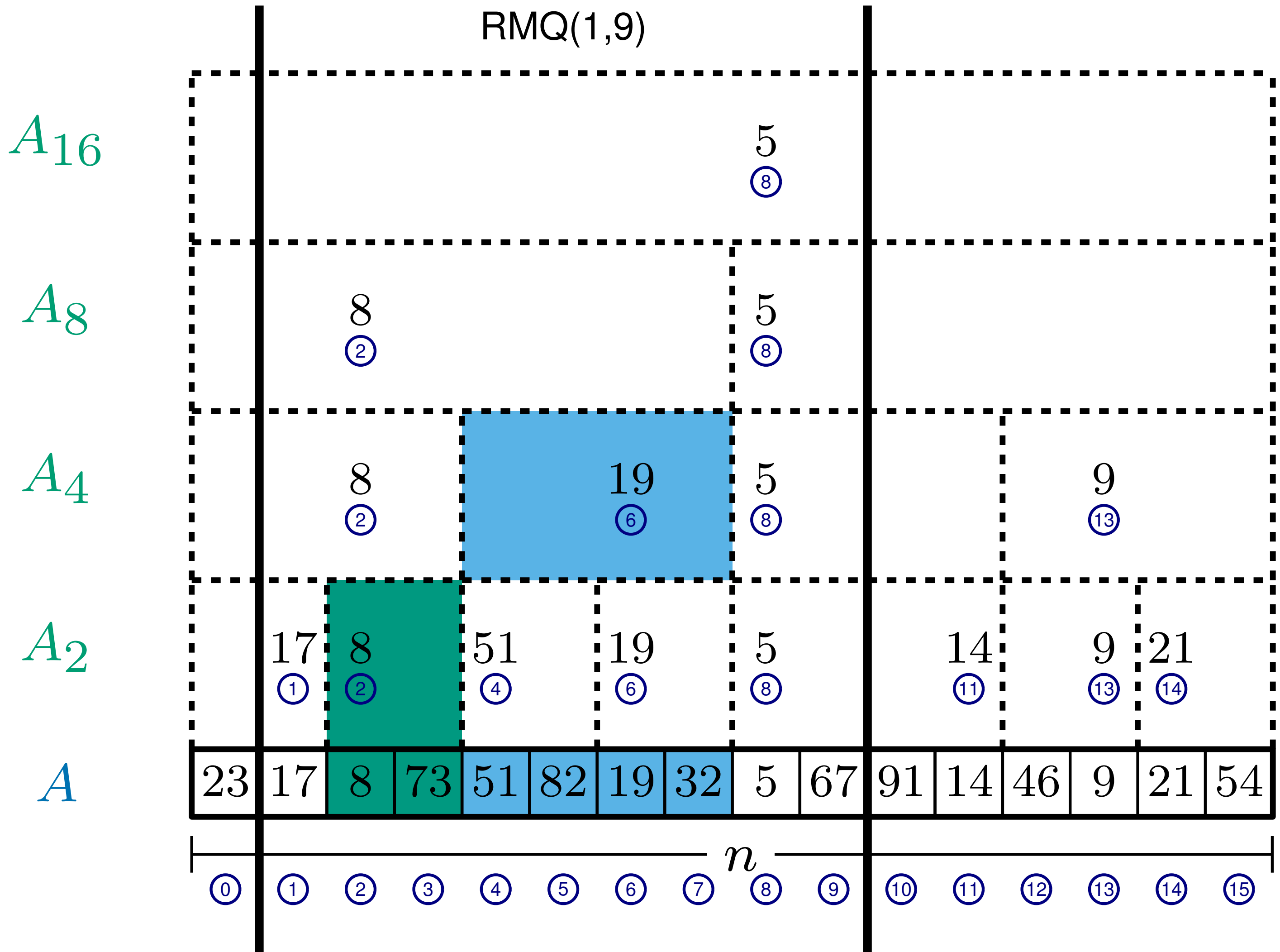
Block decomposition

How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before



Block decomposition

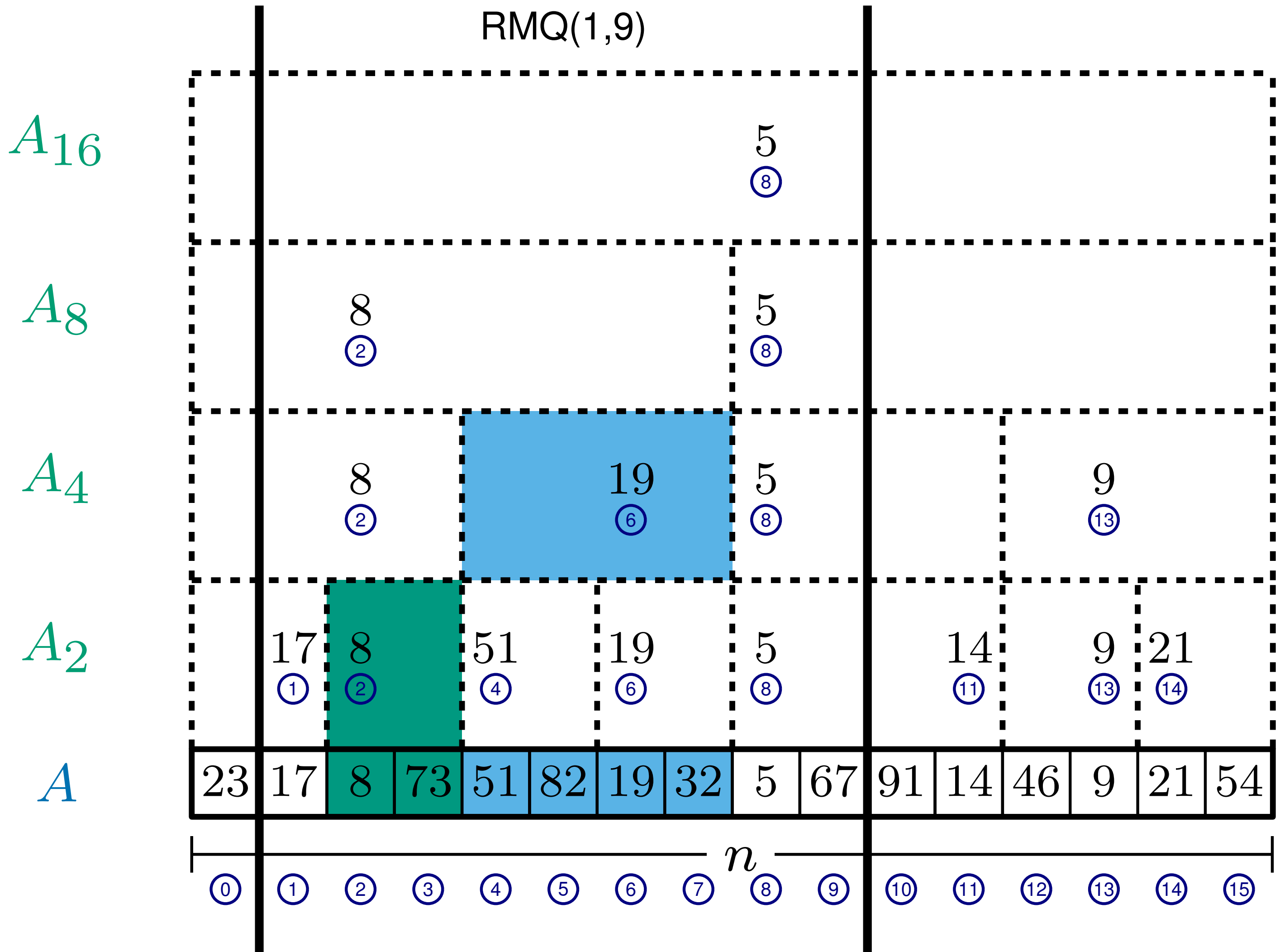
How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

(break ties arbitrarily)



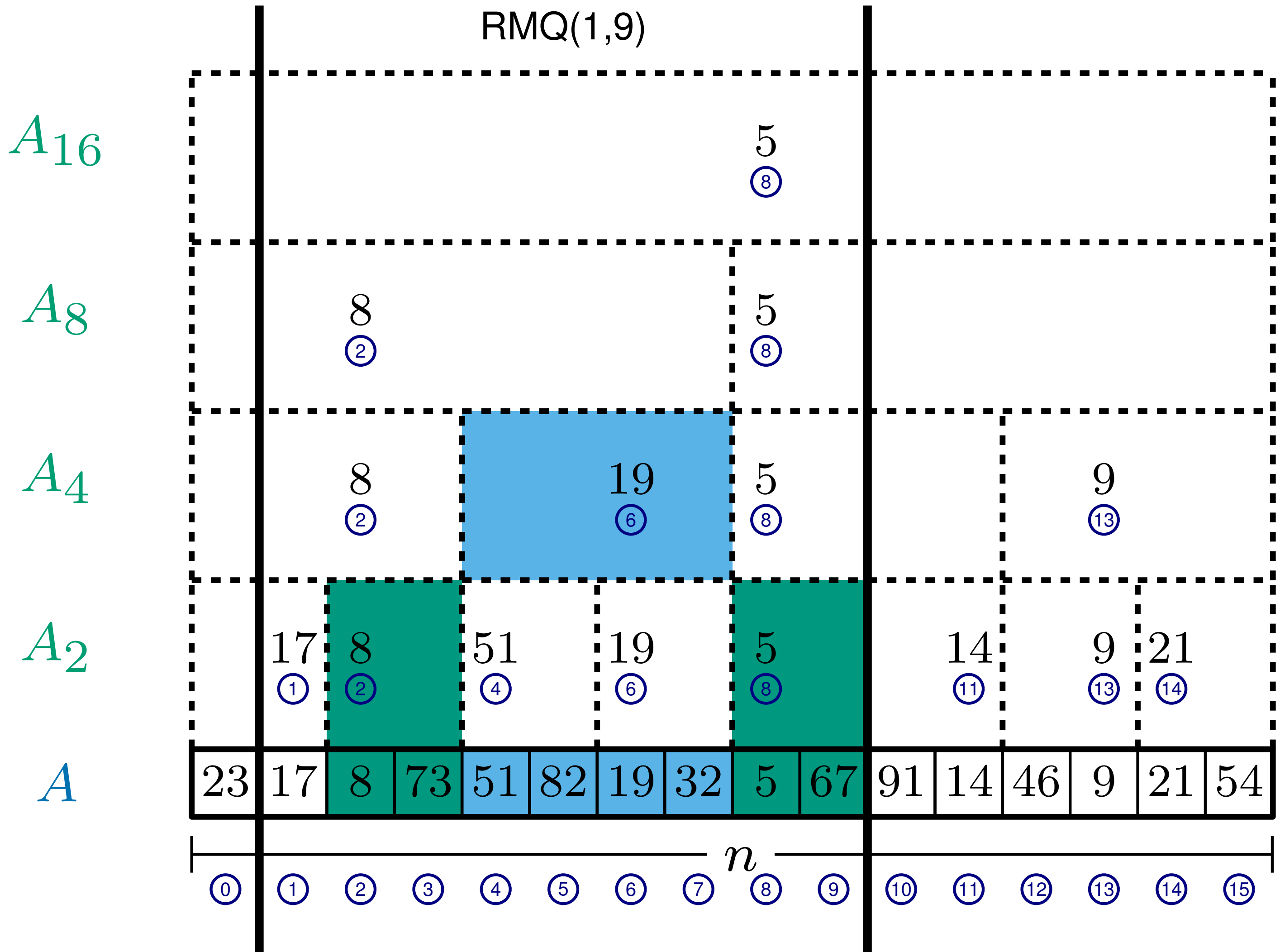
Block decomposition

How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before



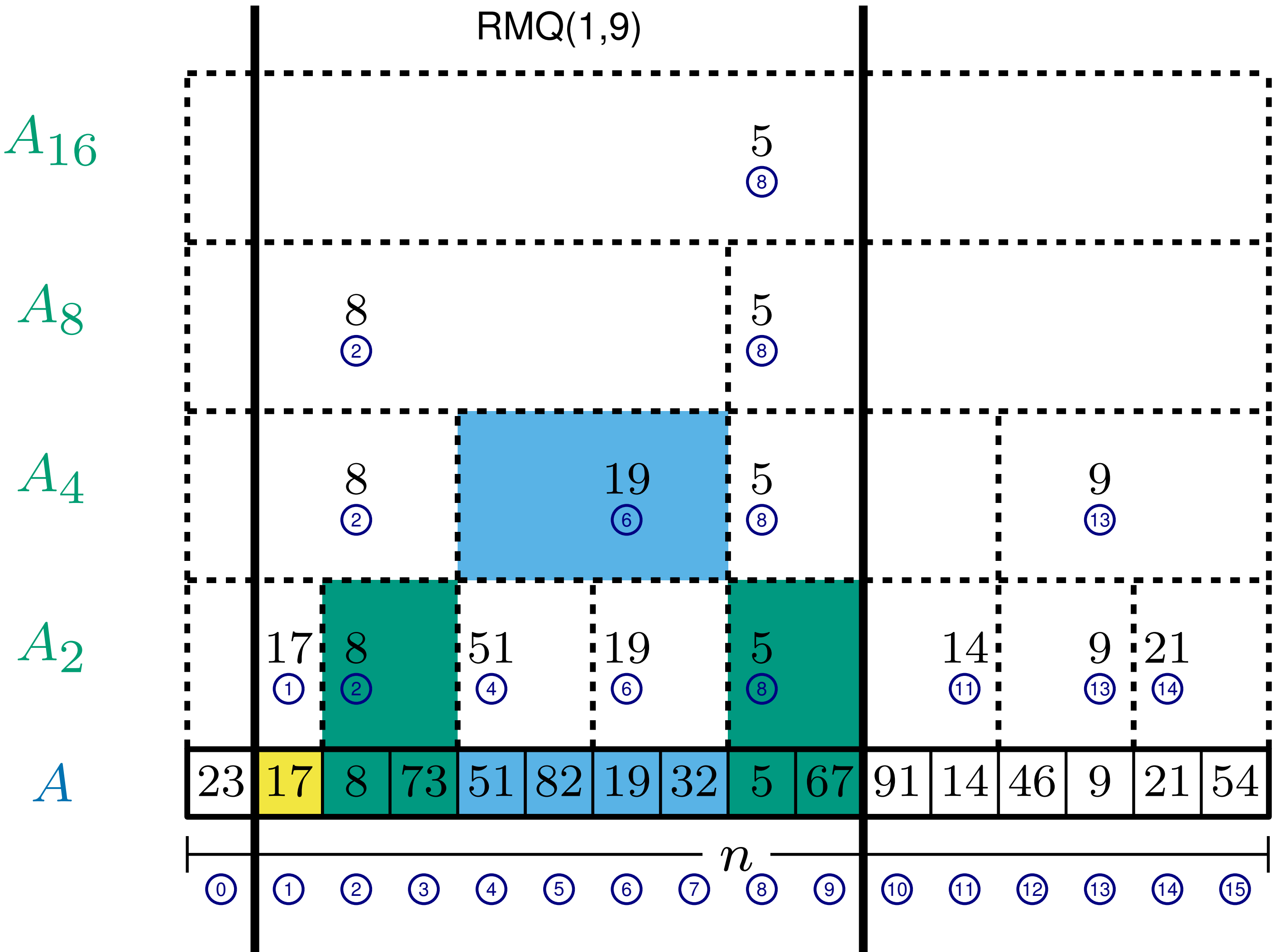
Block decomposition

How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before



Block decomposition

How do we find RMQ(i,j)?

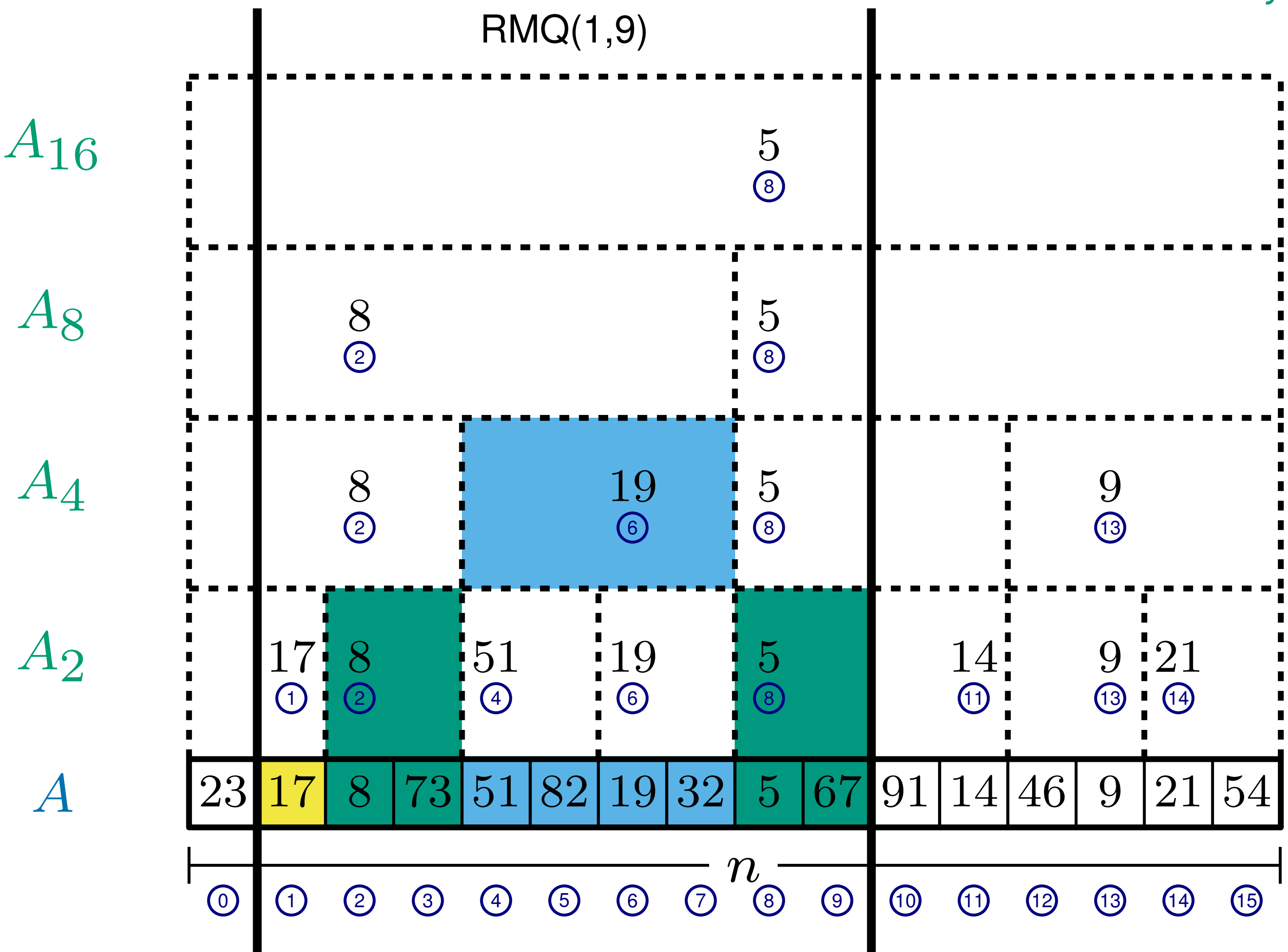
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



Block decomposition

How do we find RMQ(i,j)?

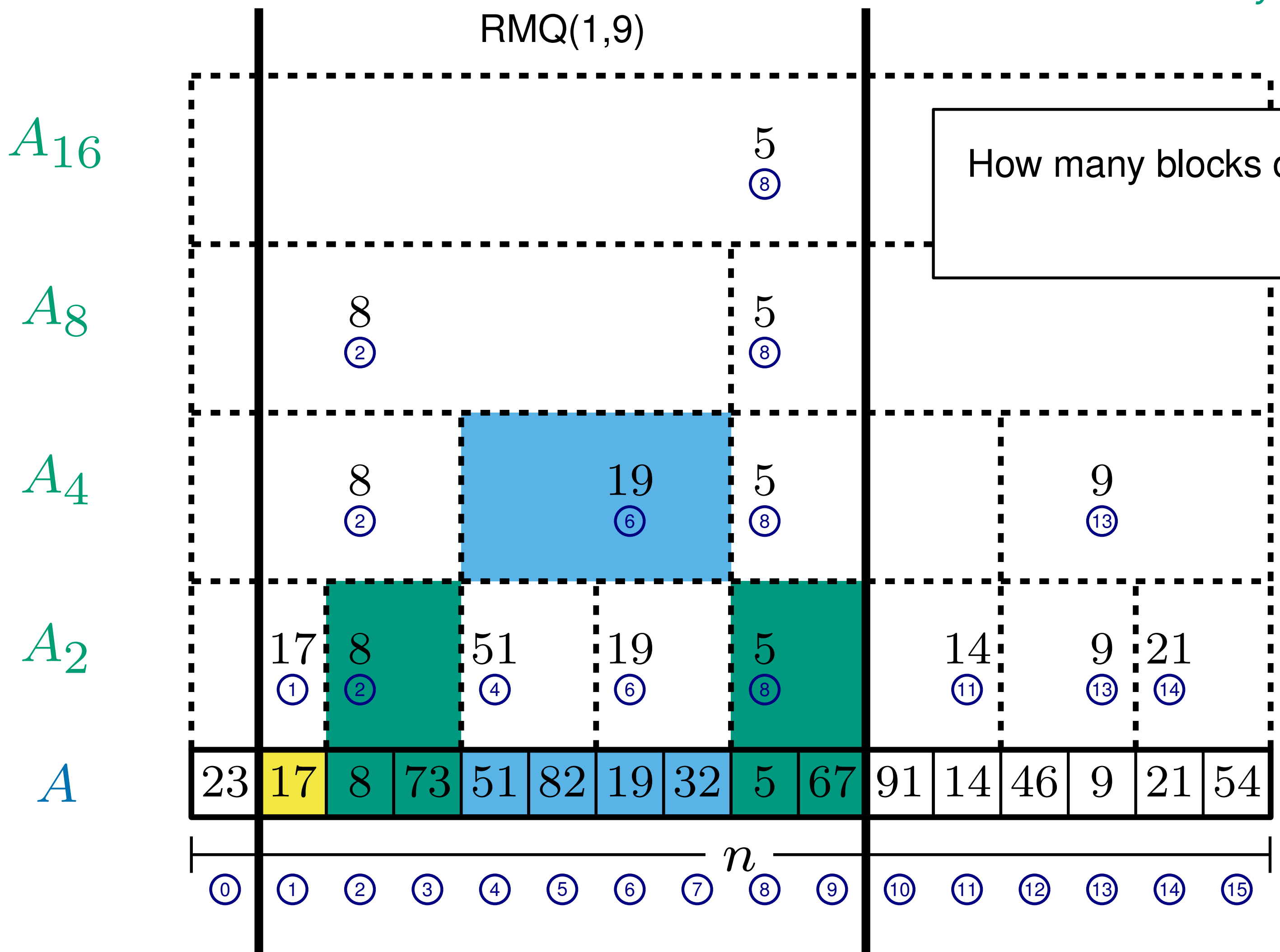
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

Block decomposition

How do we find RMQ(i,j)?

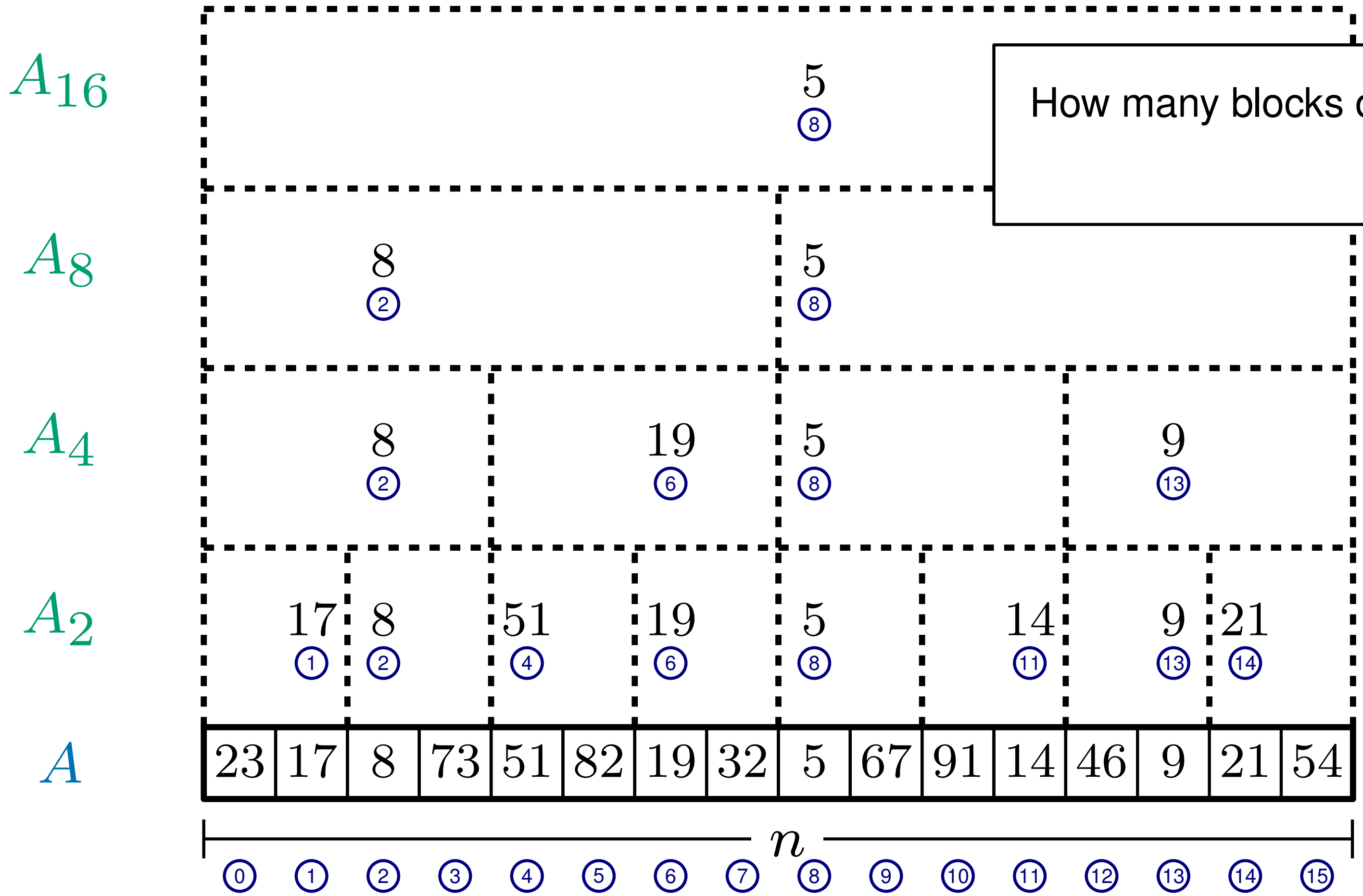
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

Block decomposition

How do we find RMQ(i,j)?

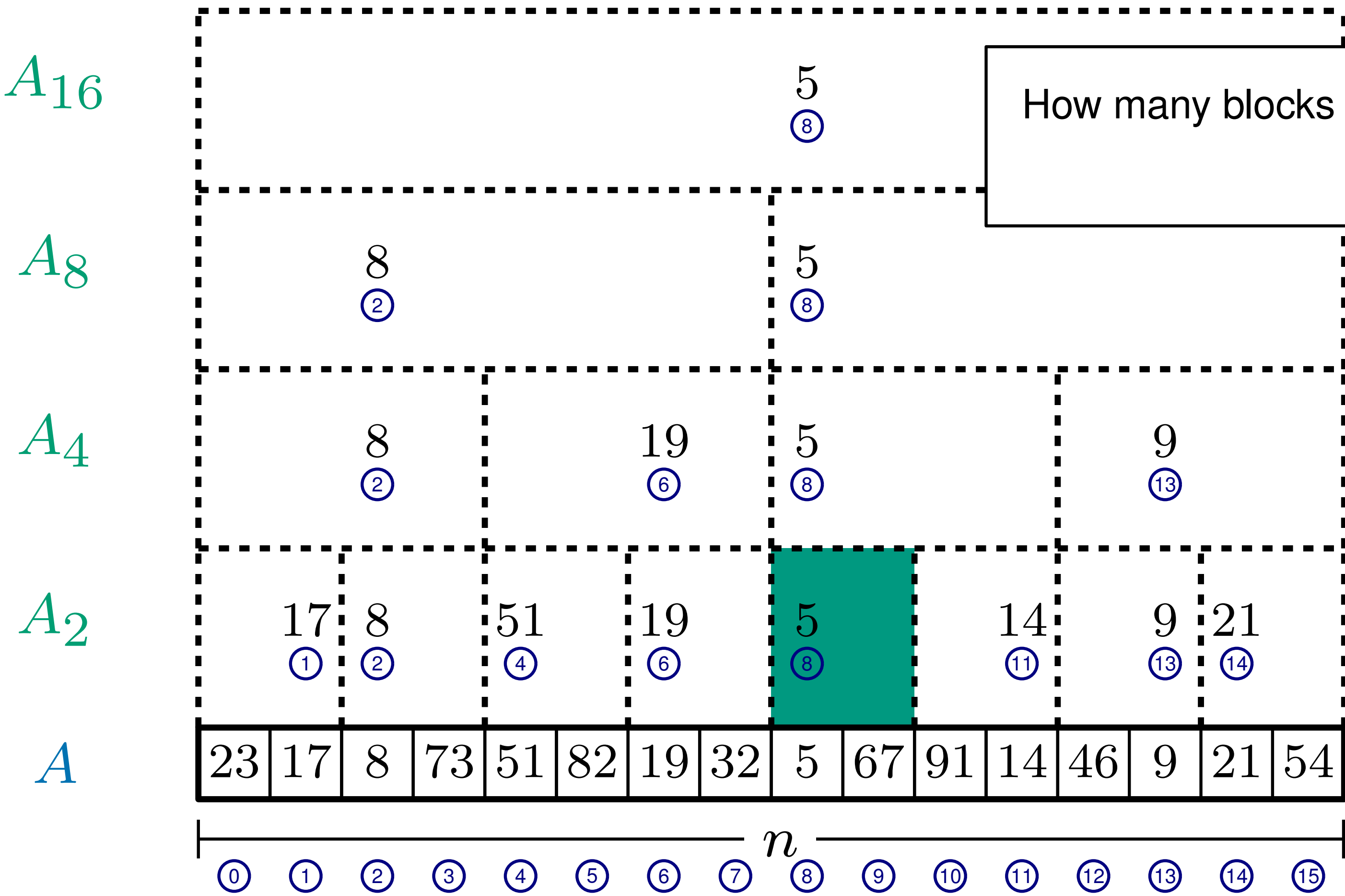
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

Block decomposition

How do we find RMQ(i,j)?

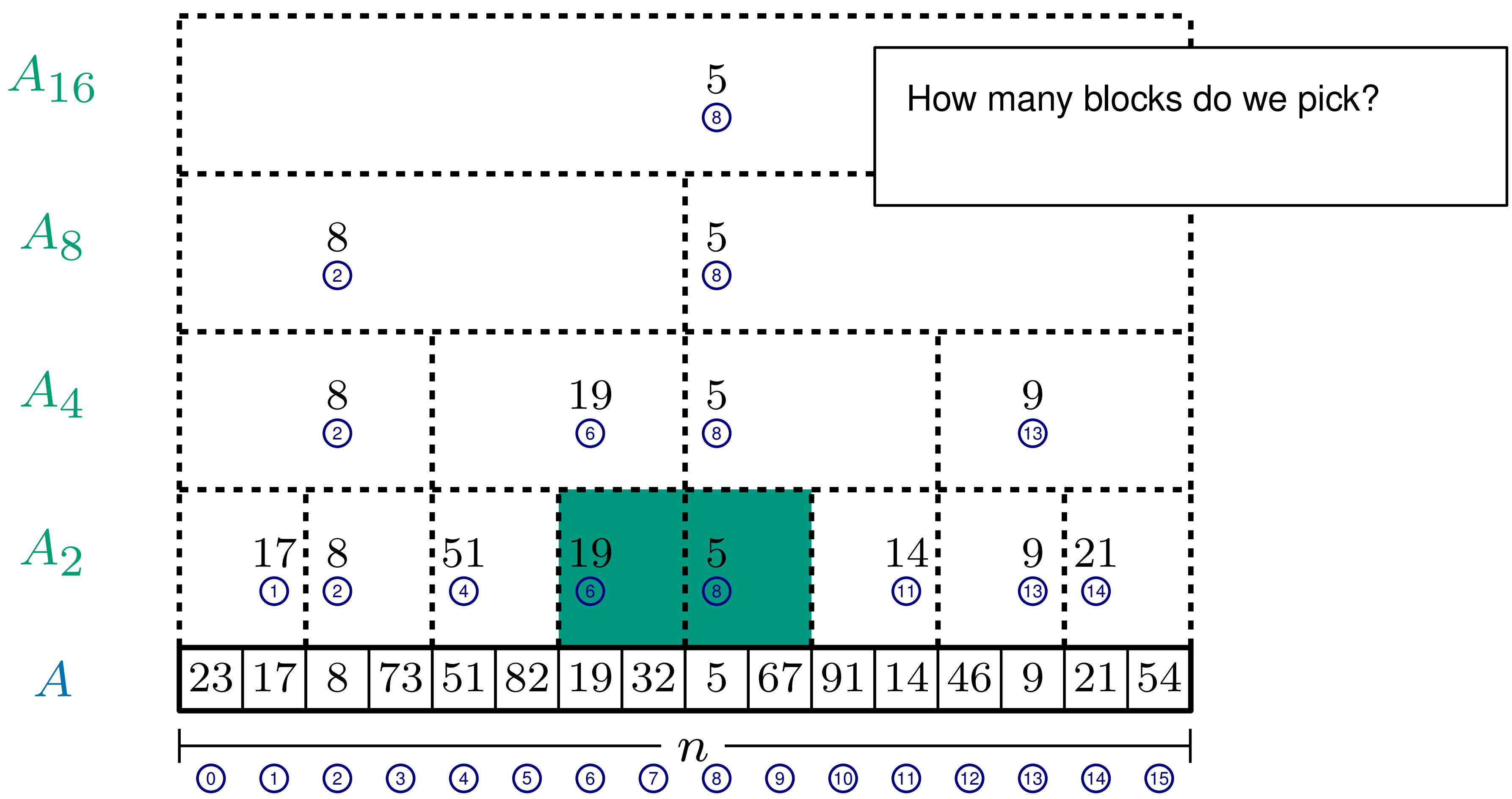
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



Block decomposition

How do we find RMQ(i,j)?

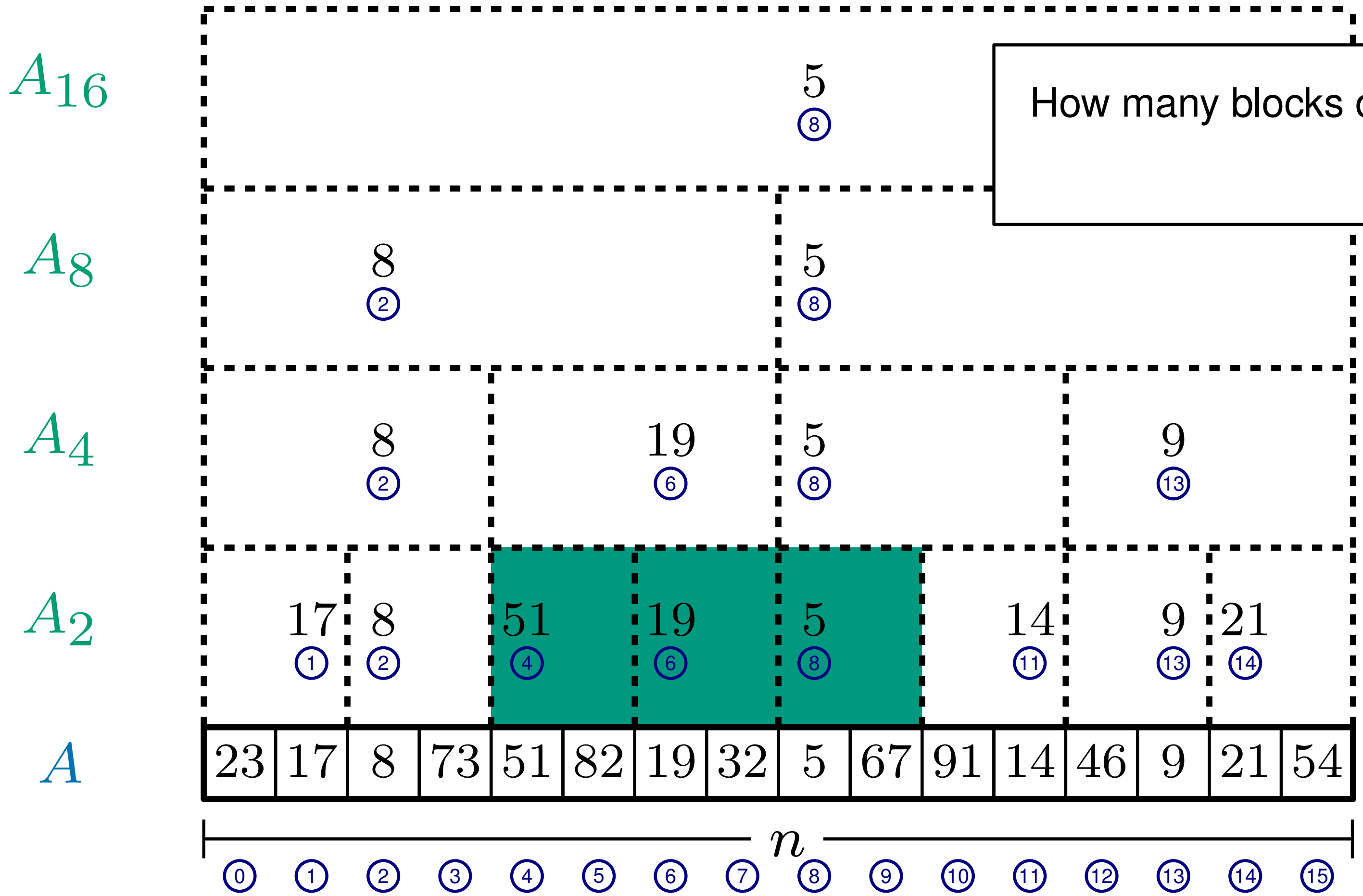
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

Block decomposition

How do we find RMQ(i,j)?

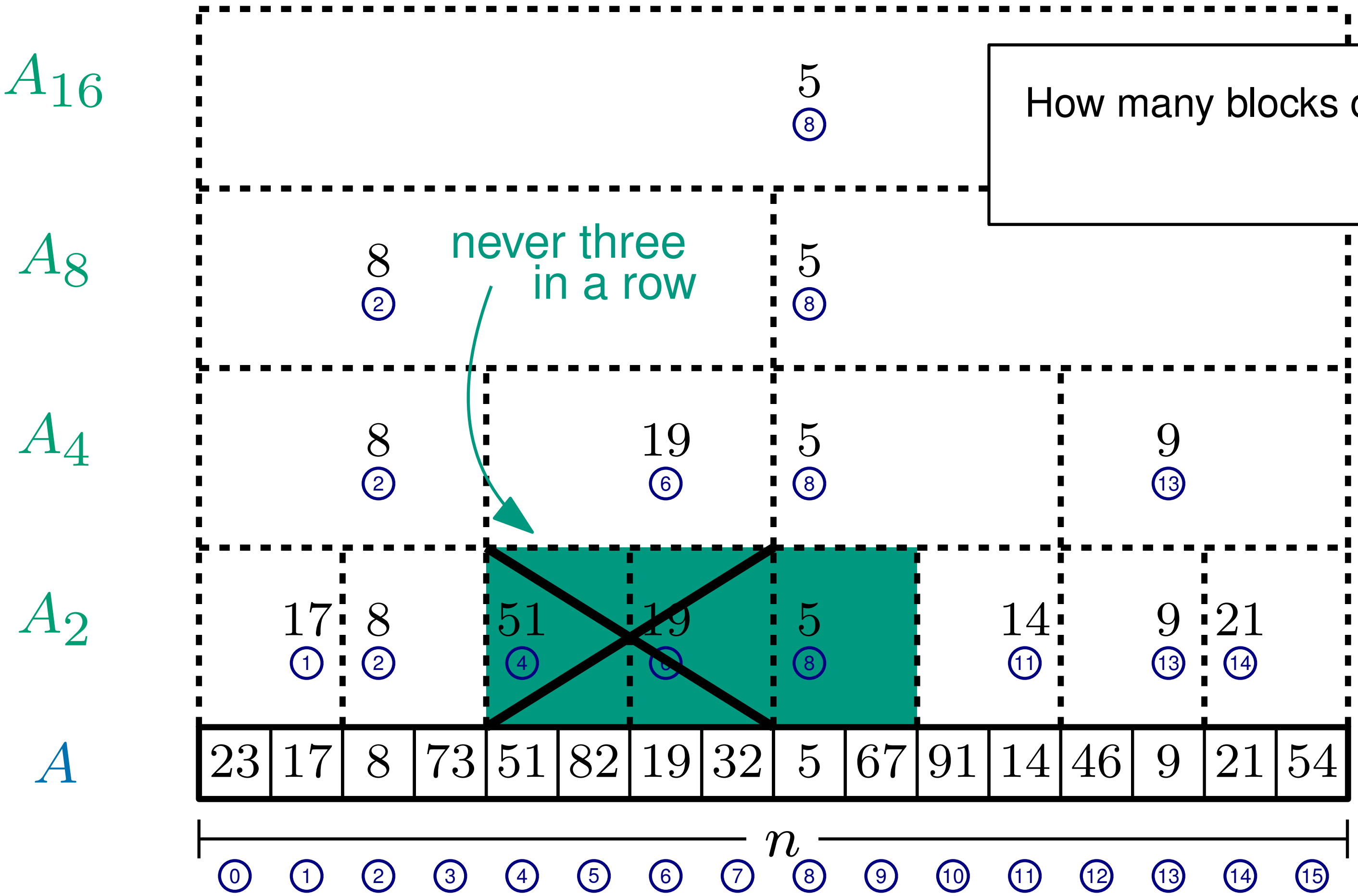
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

Block decomposition

How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

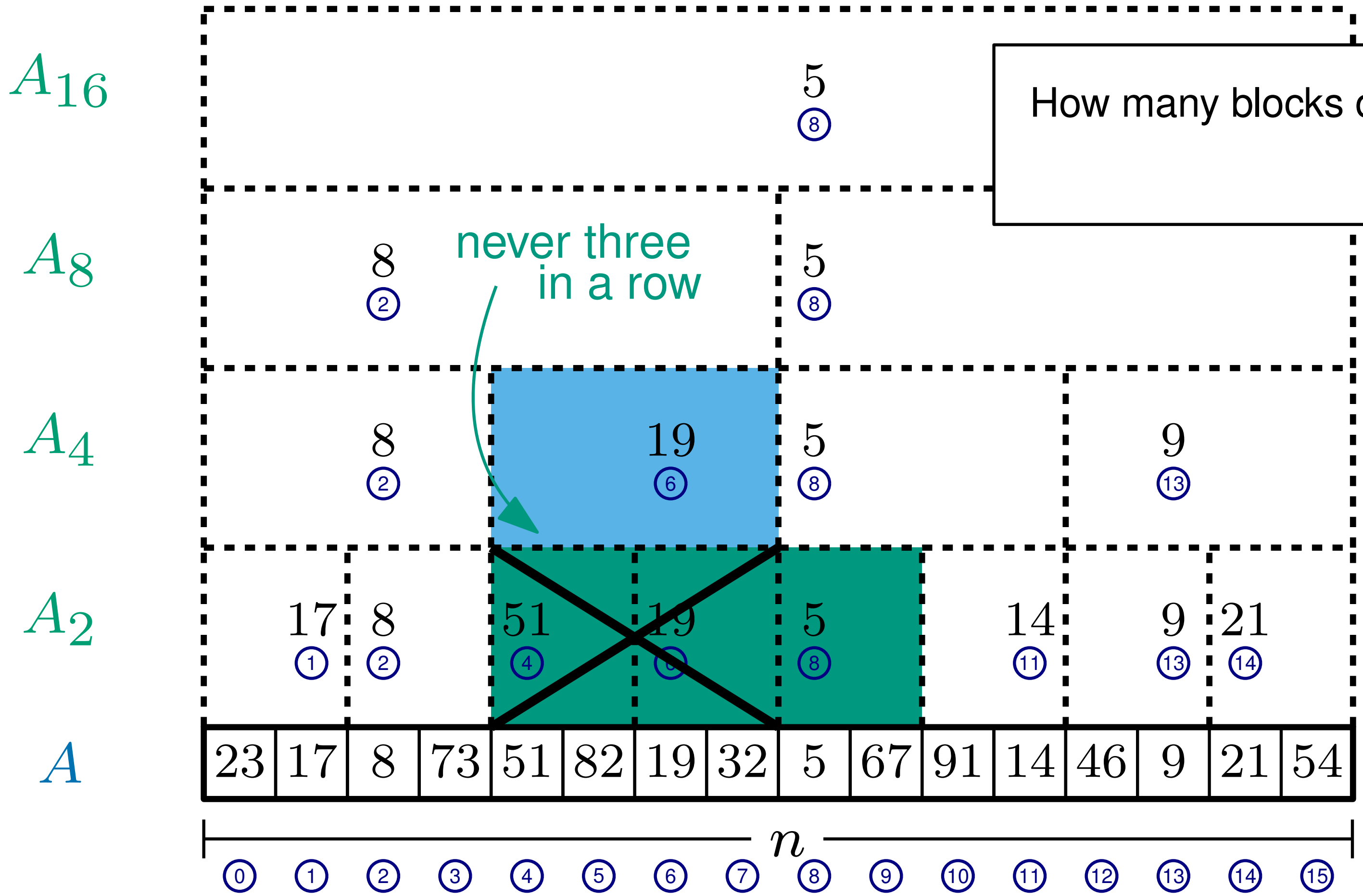
is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query

How many blocks do we pick?



Block decomposition

How do we find RMQ(i,j)?

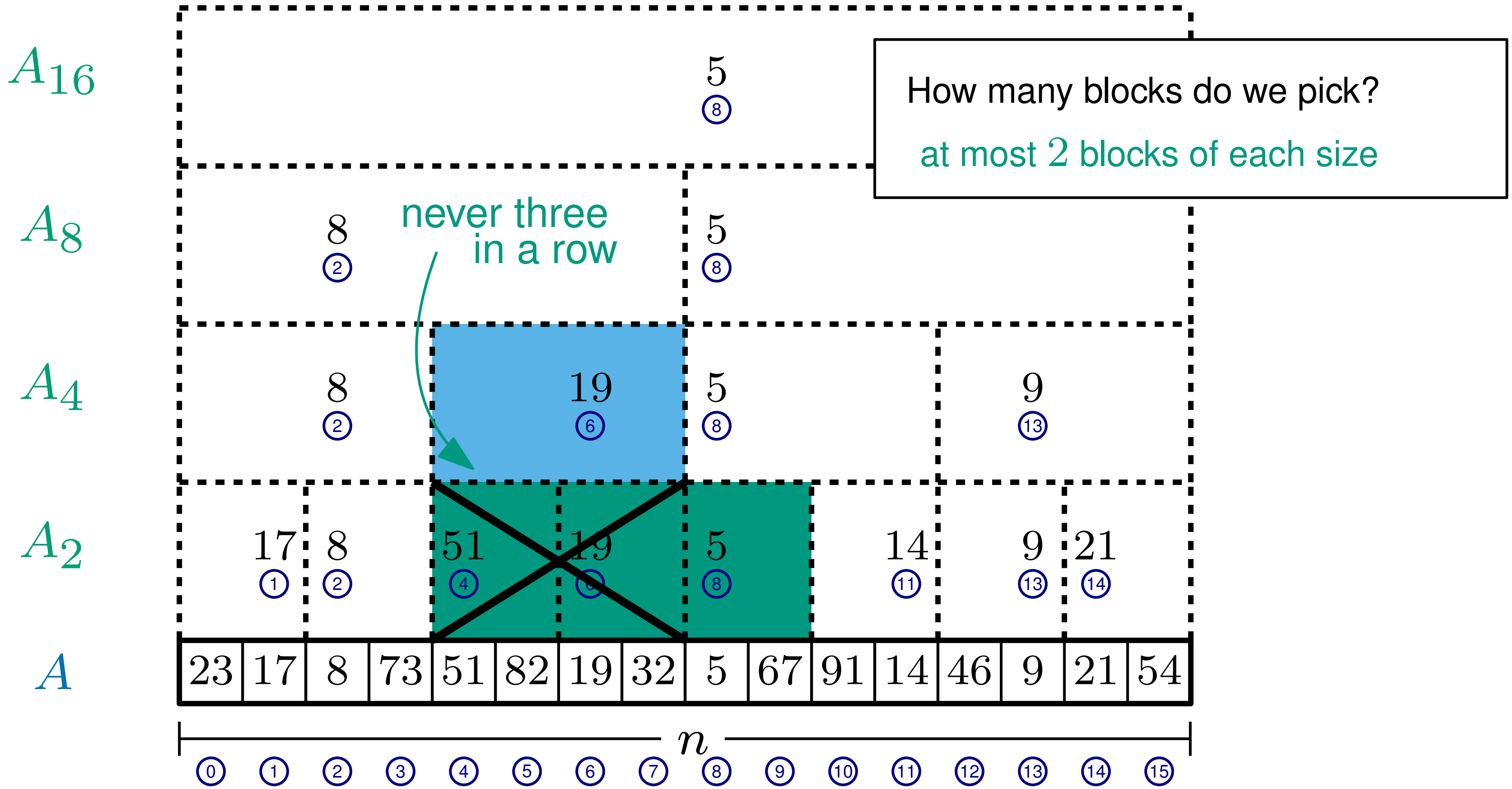
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



Block decomposition

How do we find RMQ(i,j)?

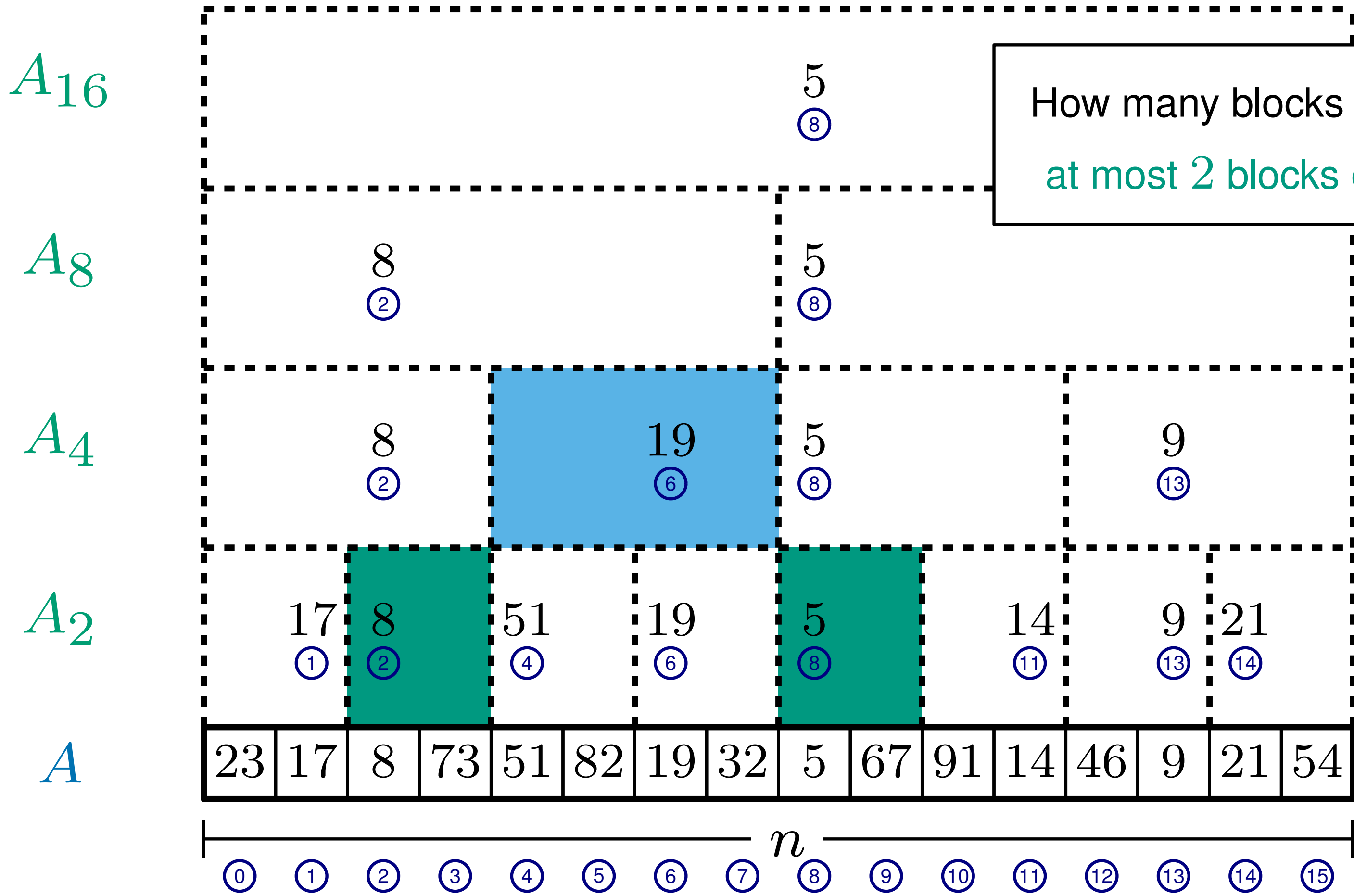
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

Block decomposition

How do we find RMQ(i,j)?

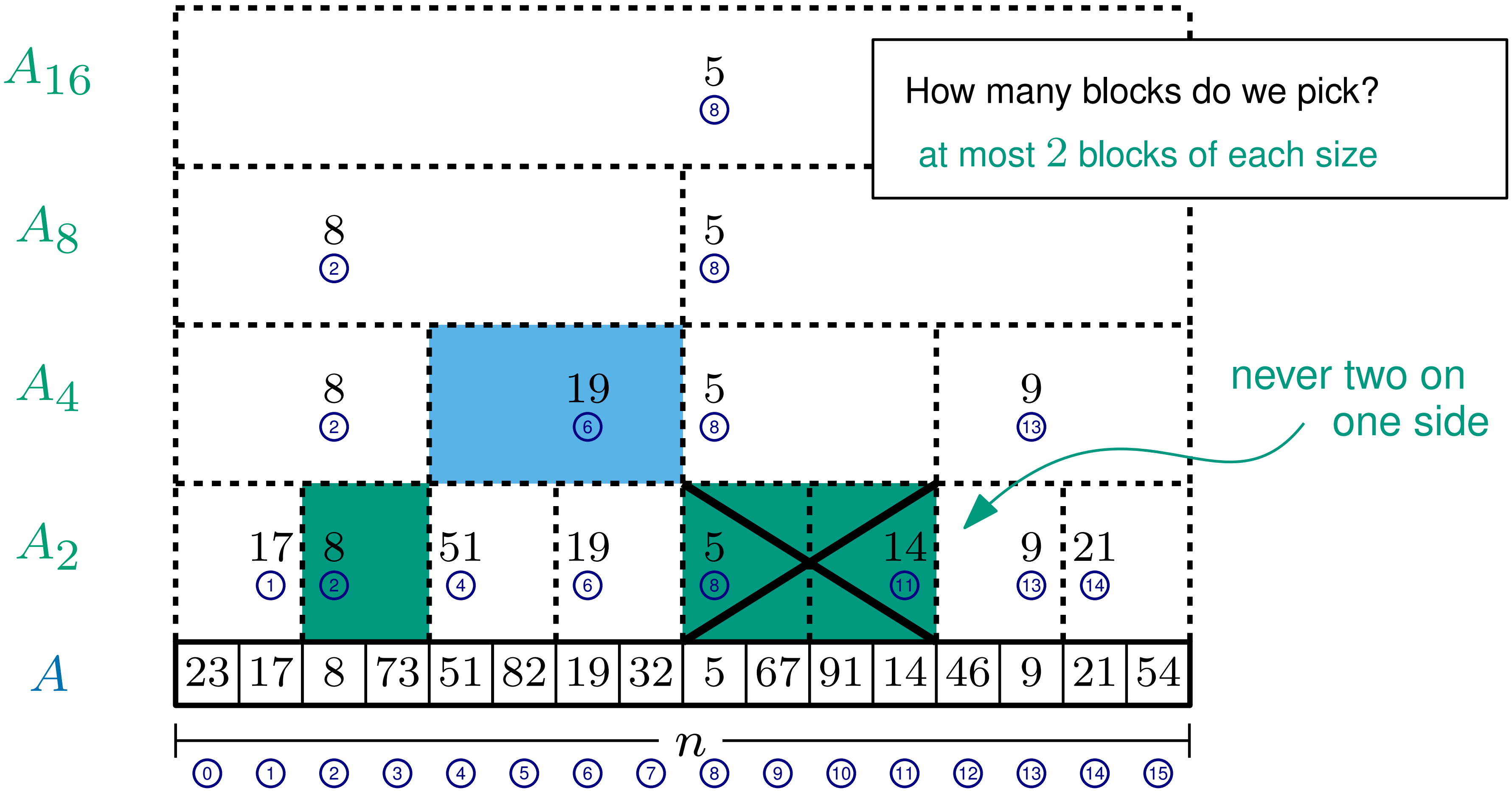
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

Block decomposition

How do we find RMQ(i,j)?

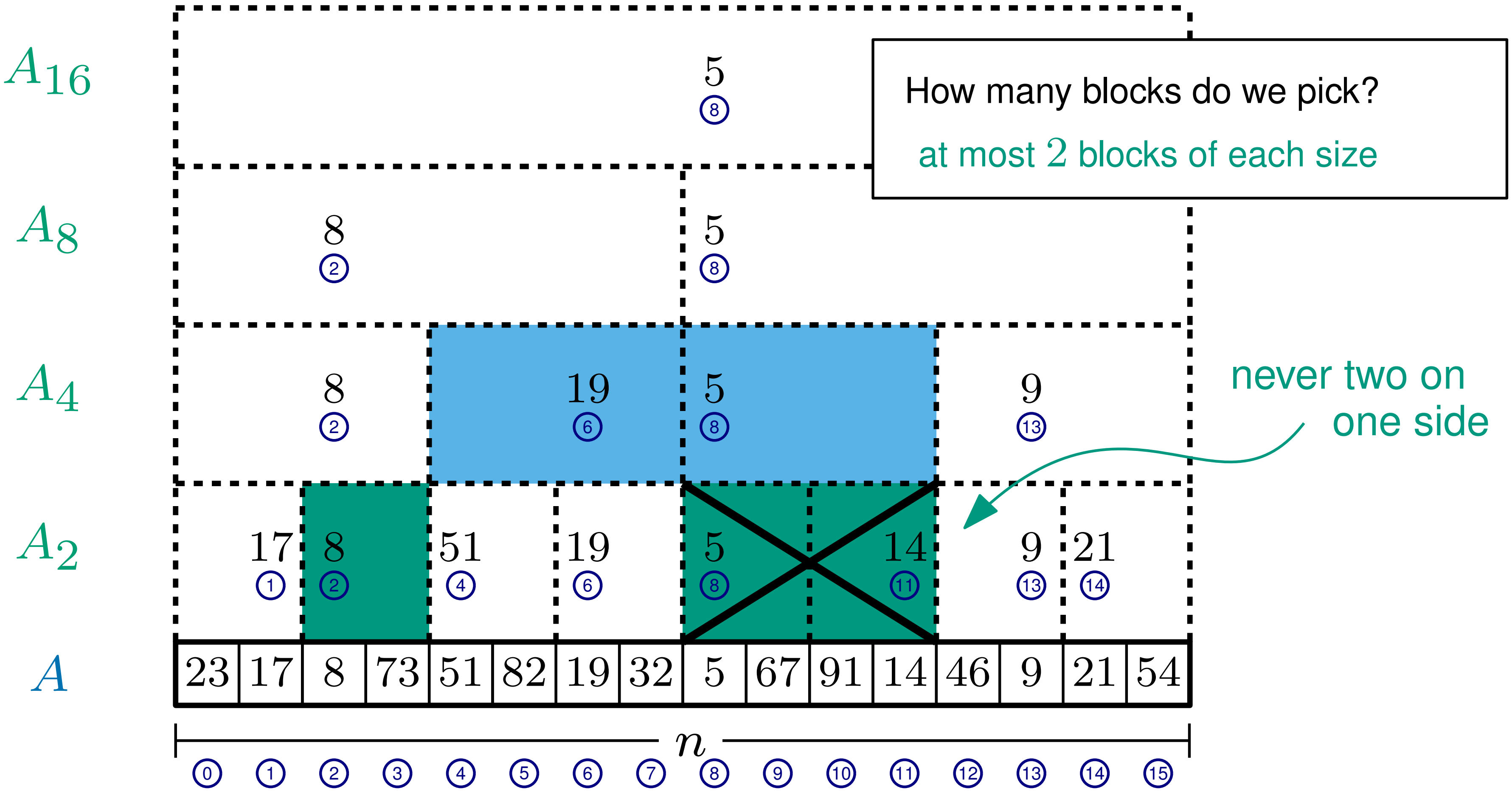
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



Block decomposition

How do we find RMQ(i,j)?

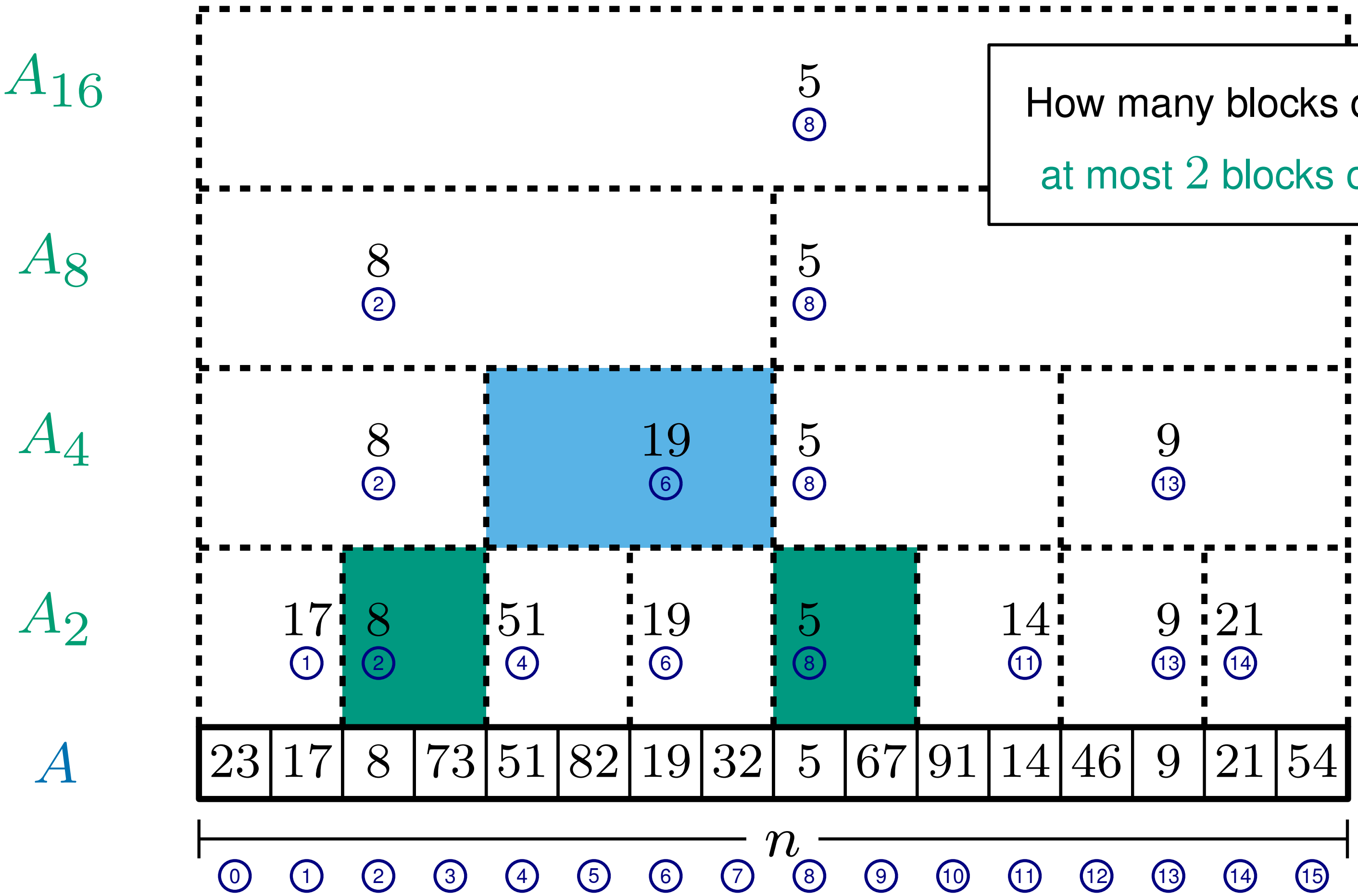
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

Block decomposition

How do we find RMQ(i,j)?

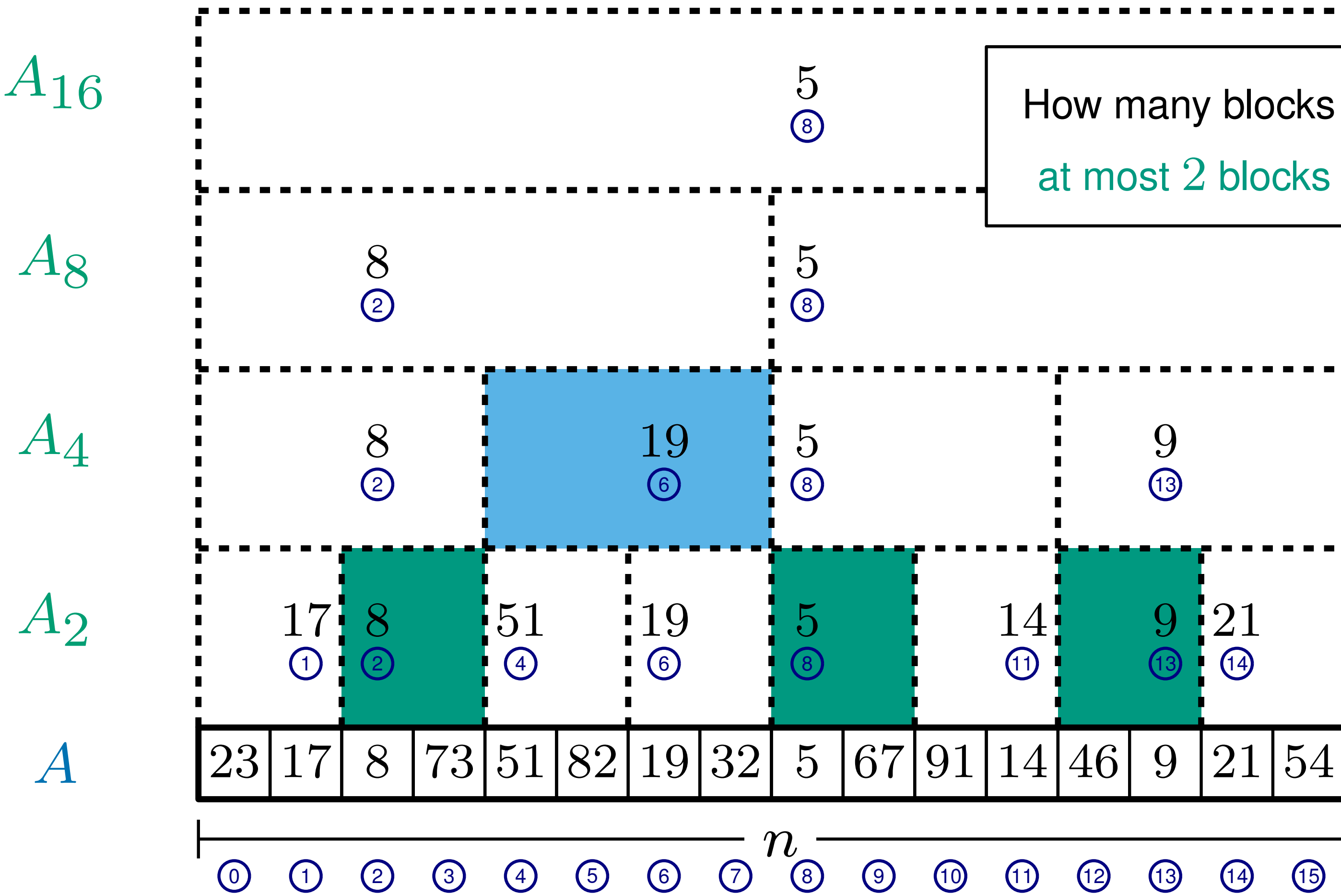
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

Block decomposition

How do we find RMQ(i,j)?

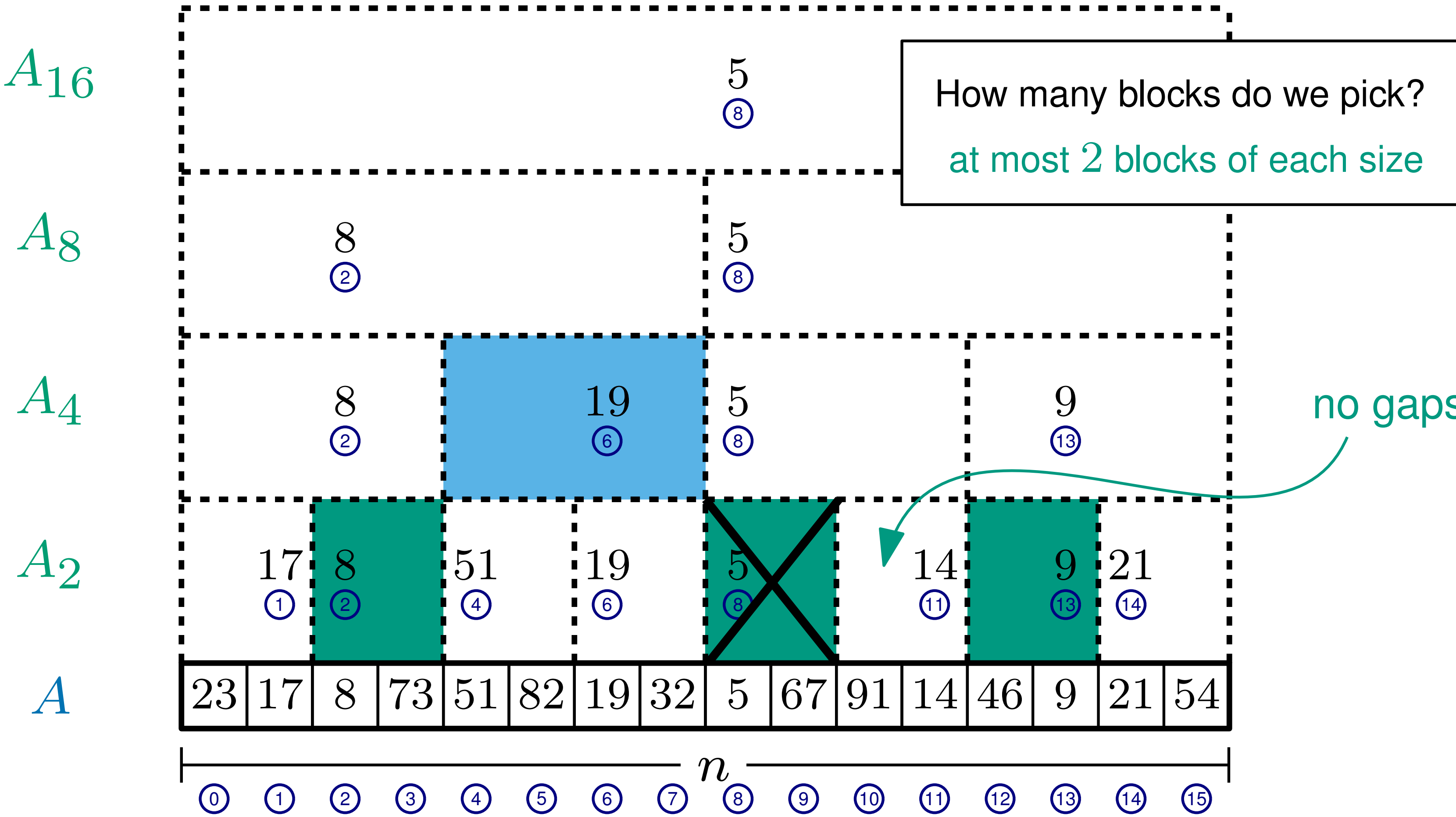
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

Block decomposition

How do we find RMQ(i,j)?

Repeat: Find the largest *block* which

is completely contained within the query interval

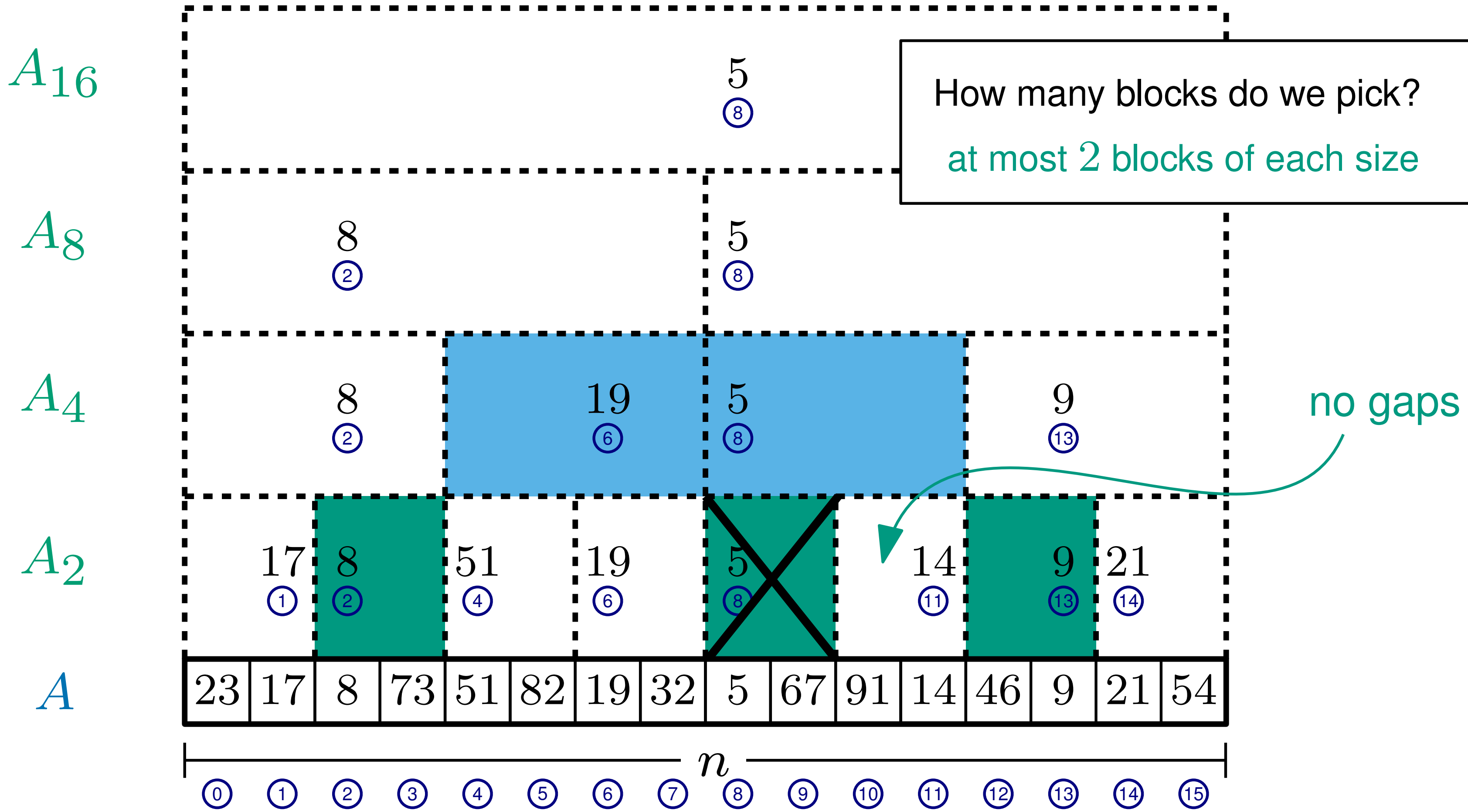
but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query

How many blocks do we pick?

at most 2 blocks of each size



Block decomposition

How do we find RMQ(i,j)?

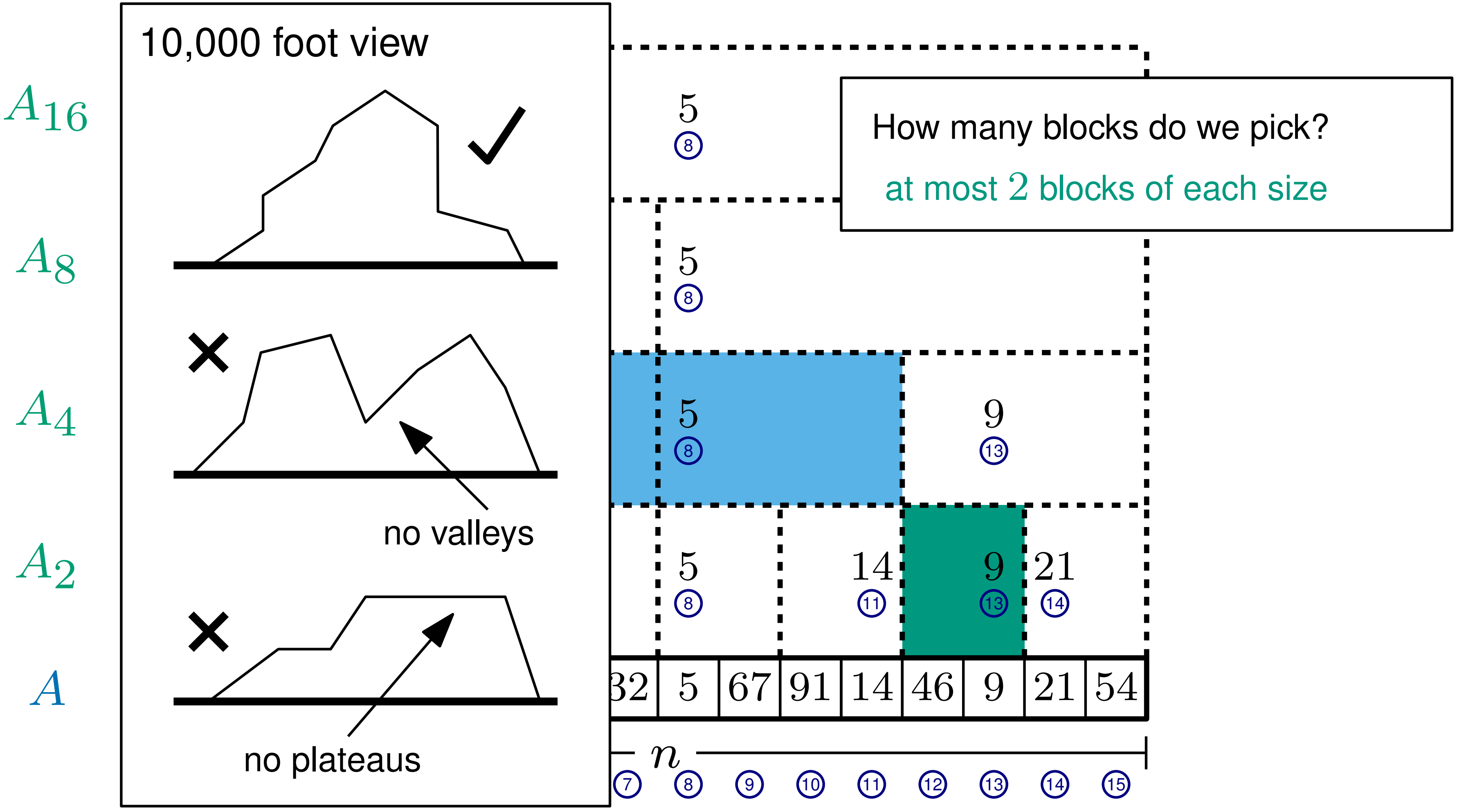
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



Block decomposition

How do we find RMQ(i,j)?

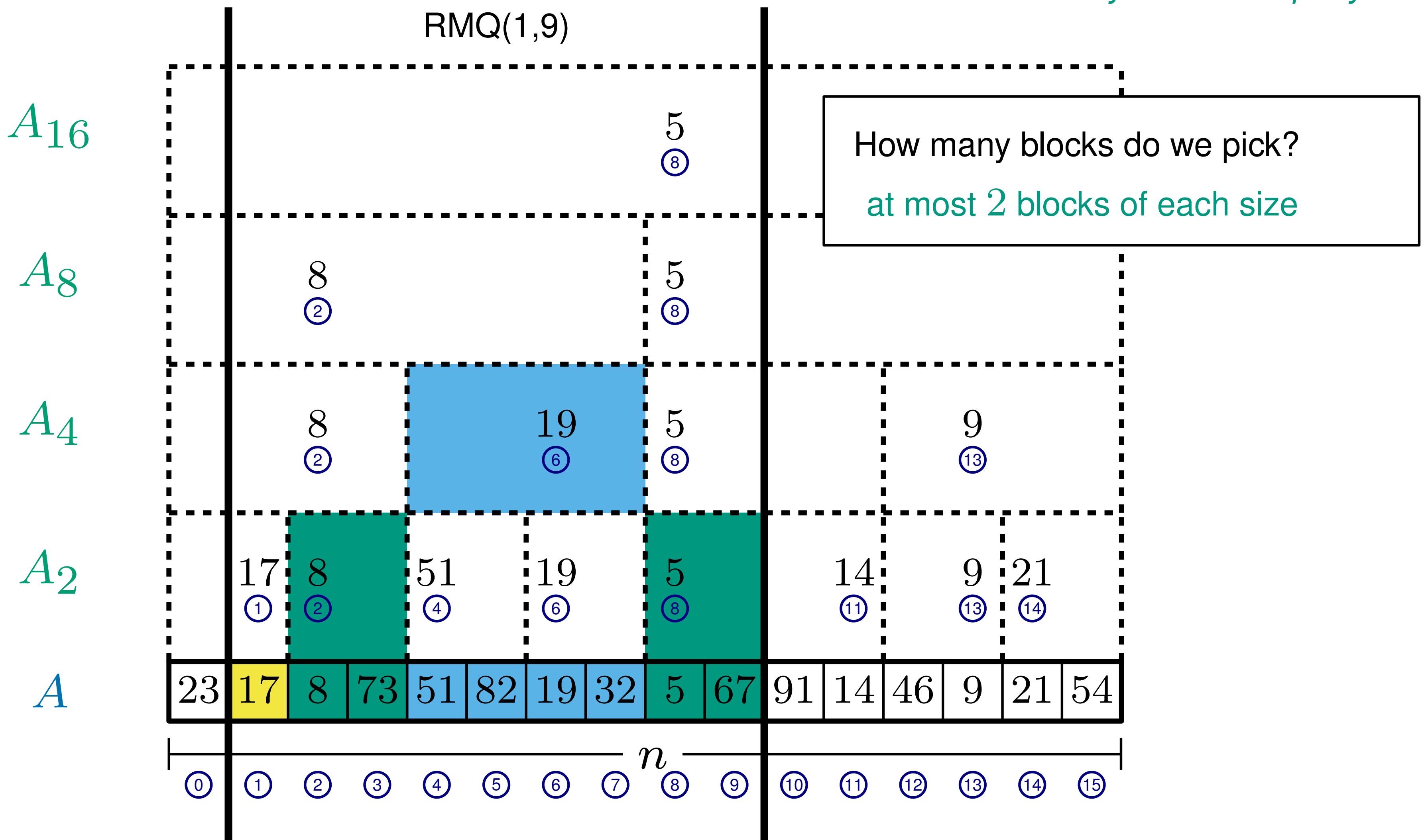
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



Block decomposition

How do we find RMQ(i,j)?

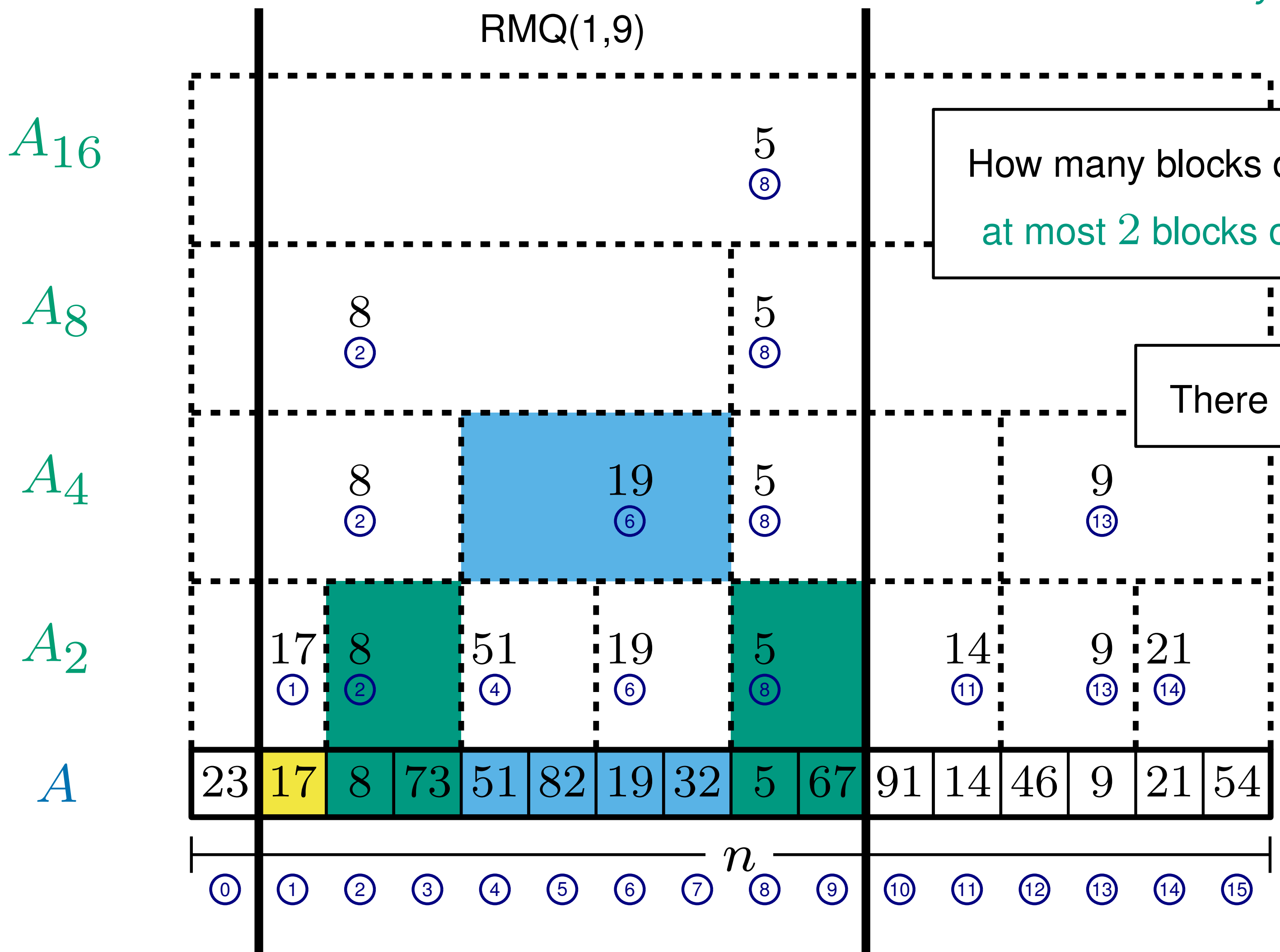
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

There are $O(\log n)$ sizes

Block decomposition

How do we find RMQ(i,j)?

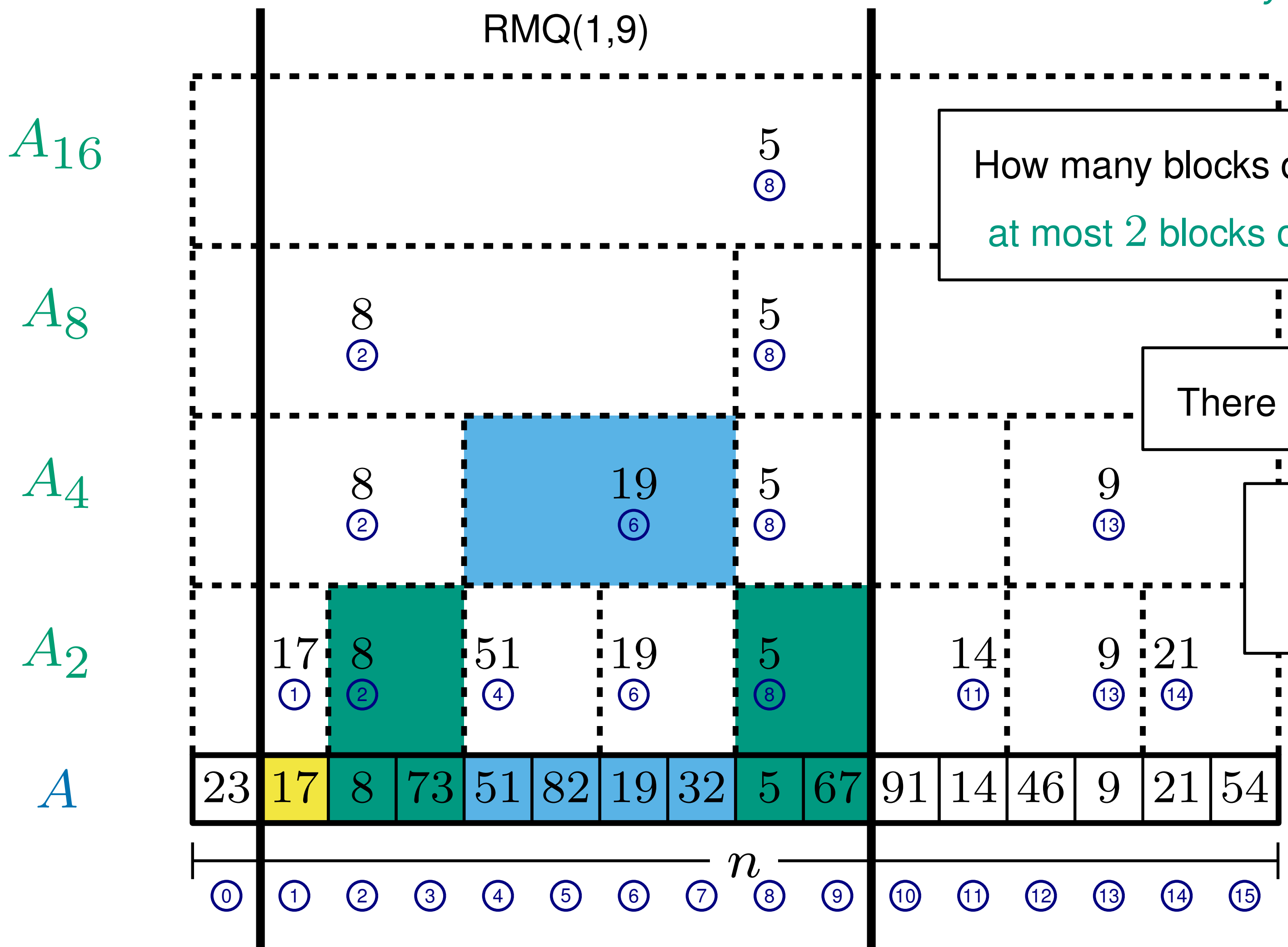
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

There are $O(\log n)$ sizes

Picking the blocks from A_k takes $O(1)$ time

Block decomposition

How do we find RMQ(i,j)?

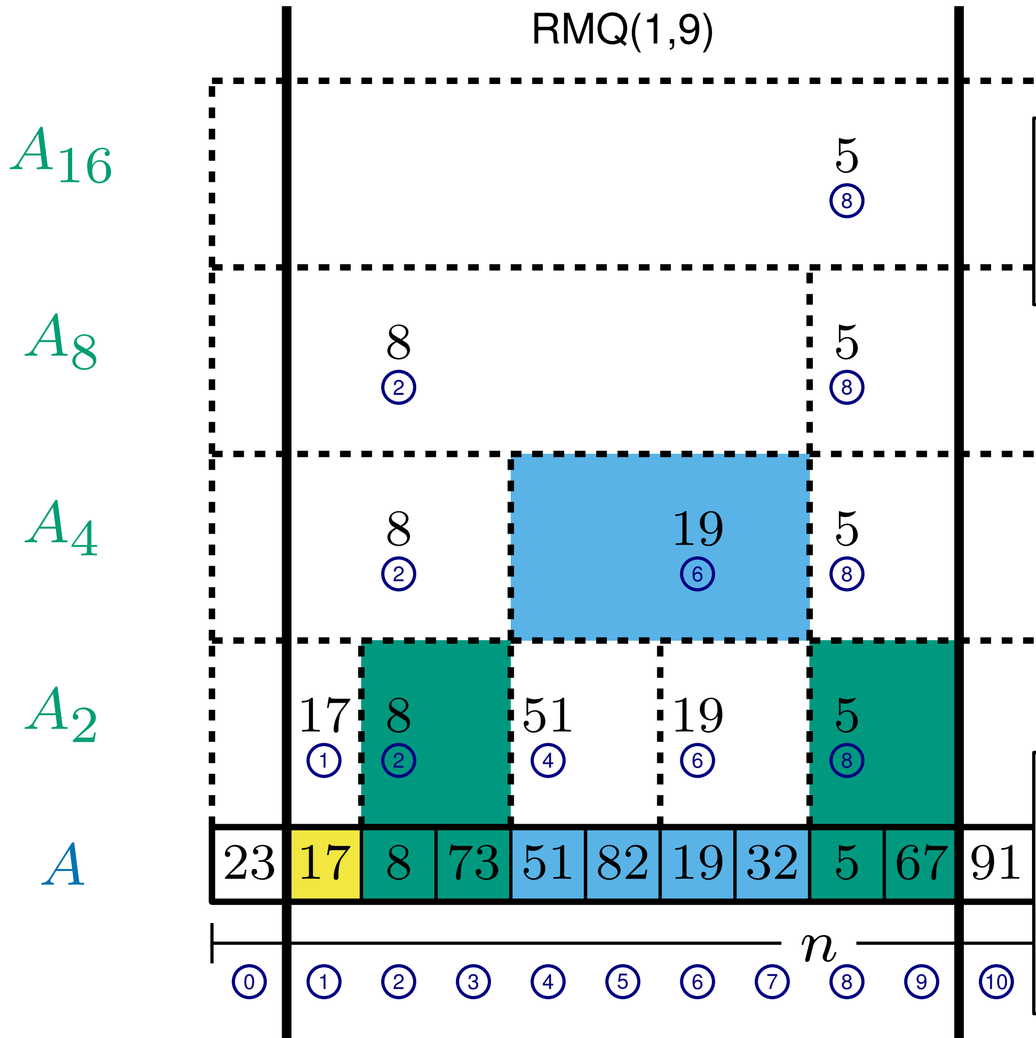
Repeat: Find the largest *block* which

is completely contained within the query interval

but doesn't overlap a block you chose before

The minimum is the smallest in all these blocks

because they cover the query



How many blocks do we pick?

at most 2 blocks of each size

There are $O(\log n)$ sizes

Picking the blocks from A_k takes $O(1)$ time

So we have . . . $O(n)$ space,

$O(n)$ prep time

$O(\log n)$ query time

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$

A

The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

More space, faster queries

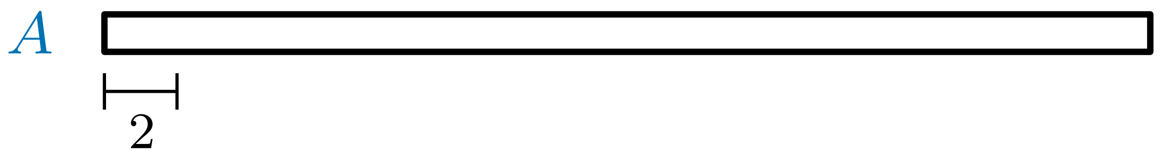
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \dots$

A

The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

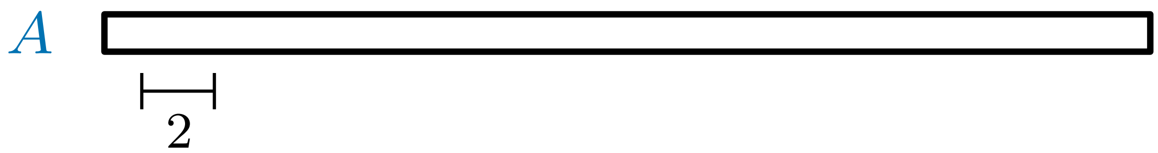
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

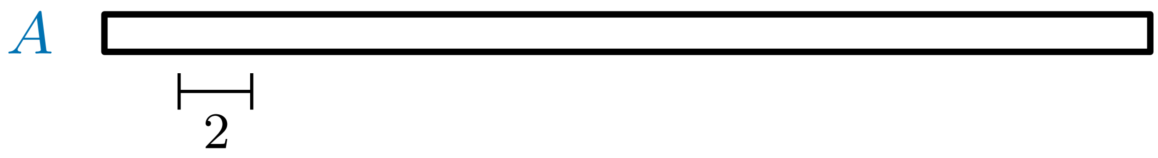
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

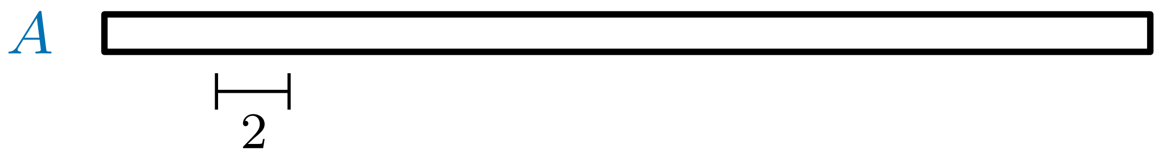
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

More space, faster queries

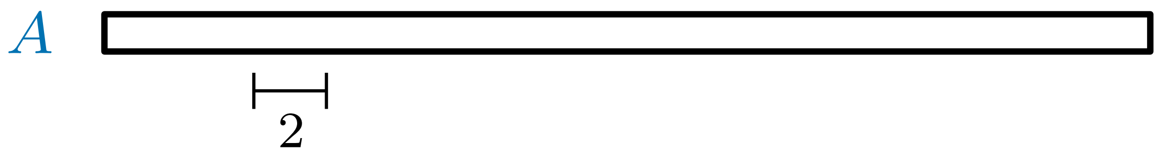
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \dots$



The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

More space, faster queries

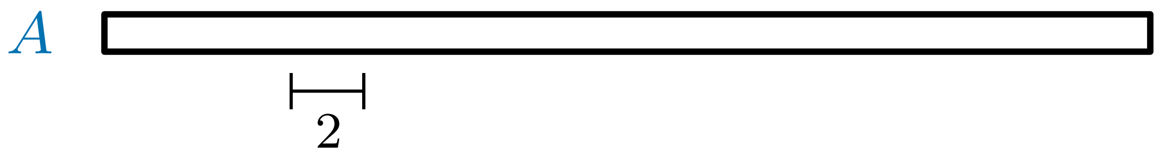
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

More space, faster queries

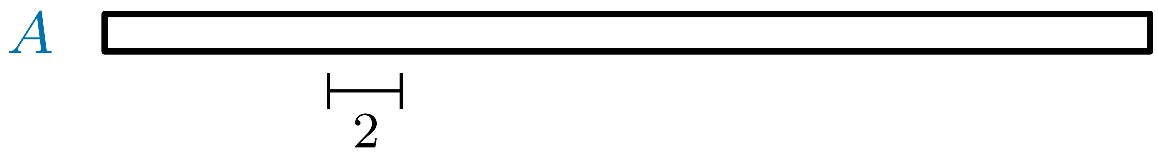
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

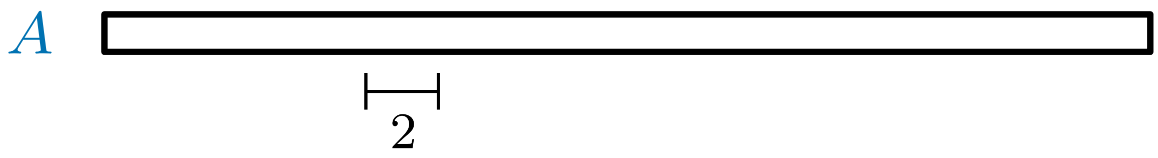
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

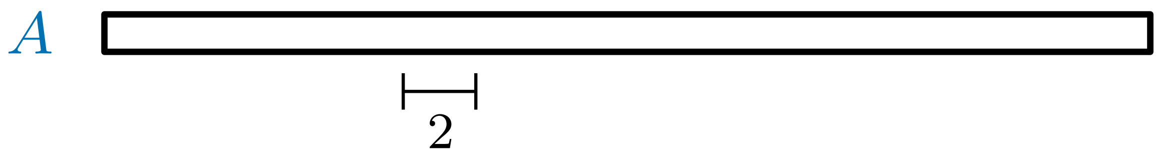
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

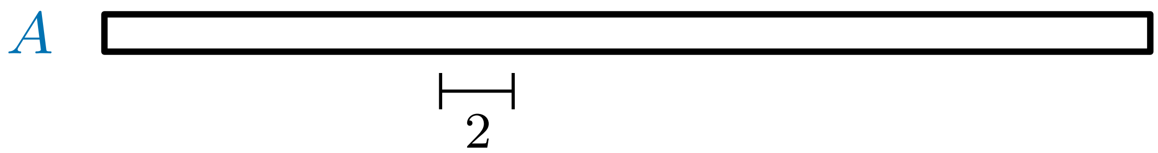
Key Idea precompute the answers for every interval of length $2, 4, 8, 16\ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

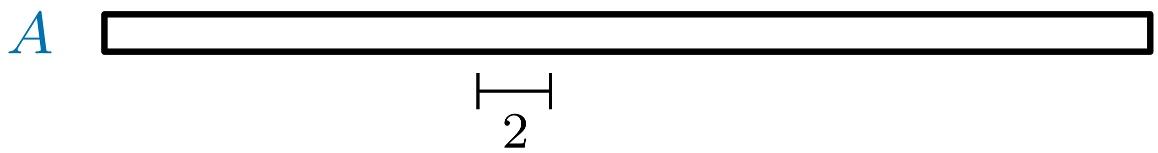
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

More space, faster queries

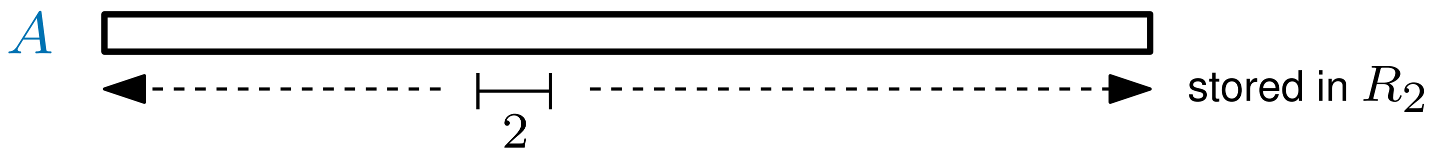
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

More space, faster queries

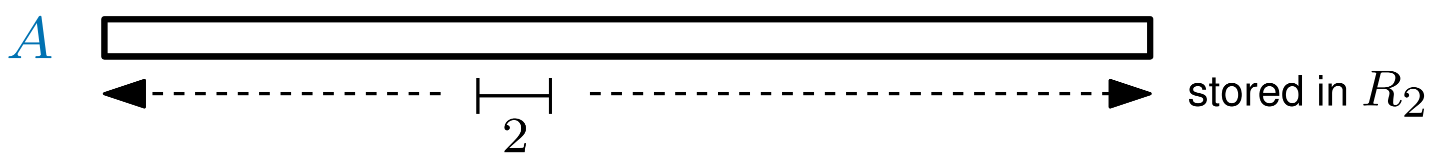
Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$

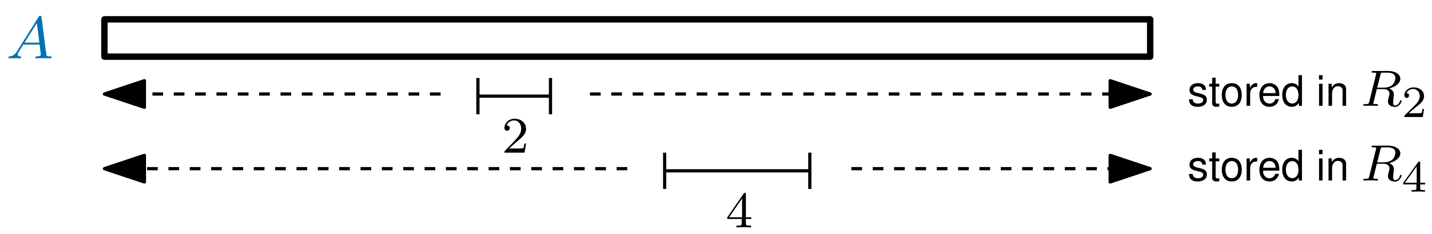


The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$

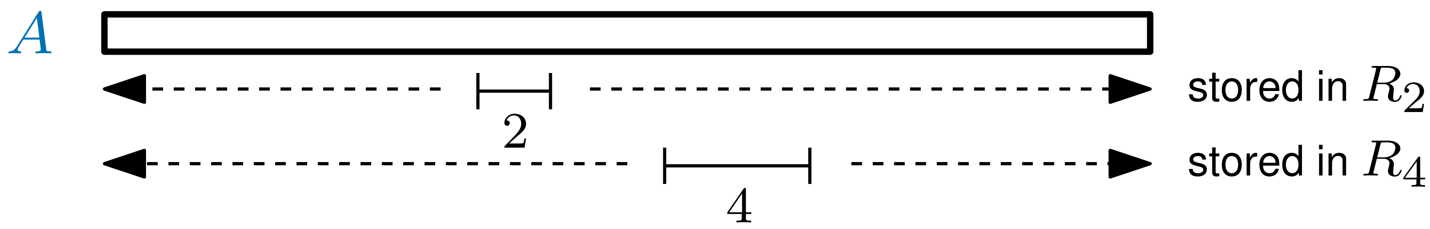


The array R_2 stores $\mathsf{RMQ}(i, i+1)$ for all i

R_4 stores $\mathsf{RMQ}(i, i+3)$ for all i

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



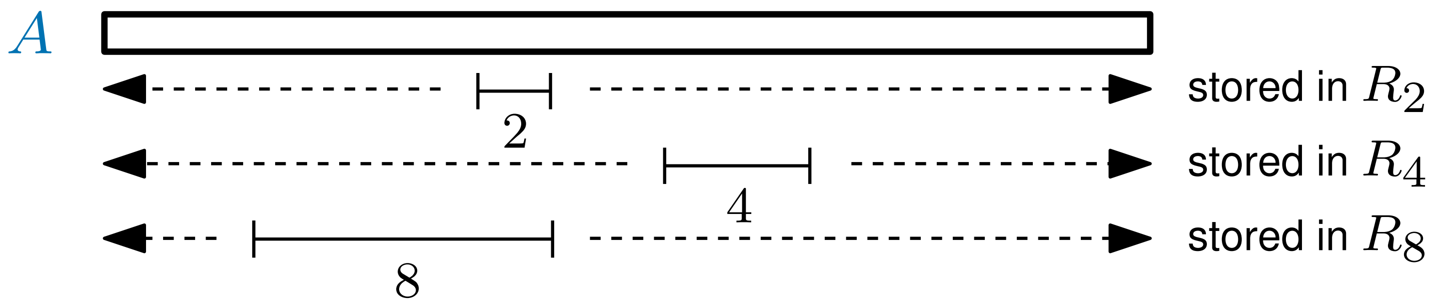
The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



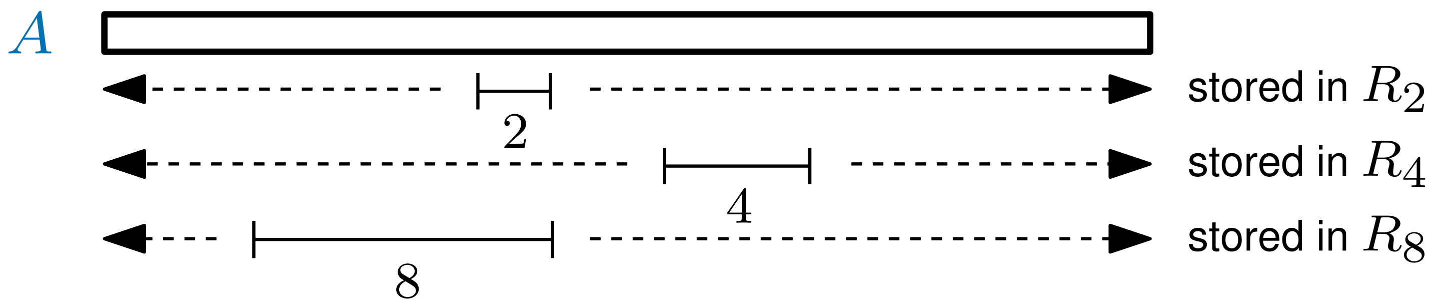
The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

R_4 stores $\text{RMQ}(i, i+3)$ for all i

R_8 stores $\text{RMQ}(i, i+7)$ for all i

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

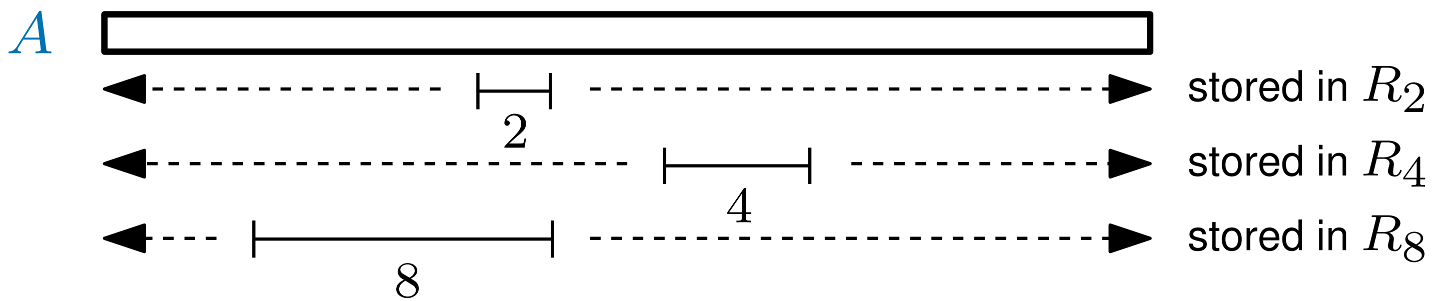
R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

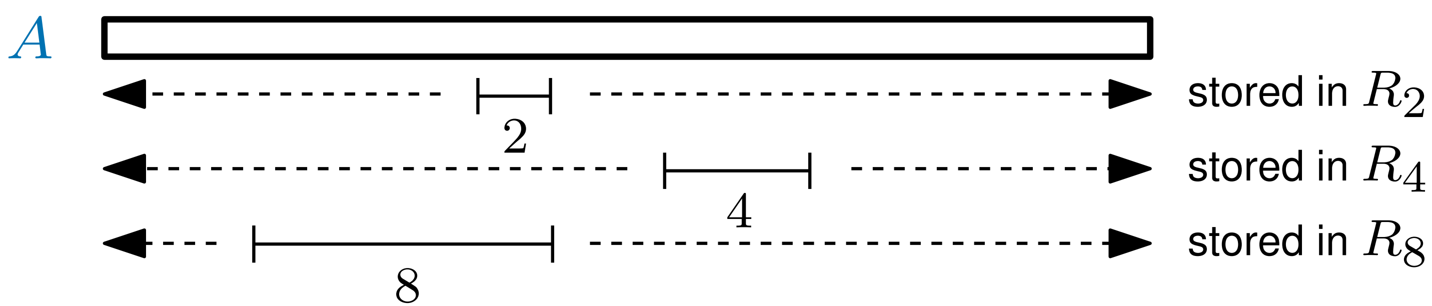
R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16\ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

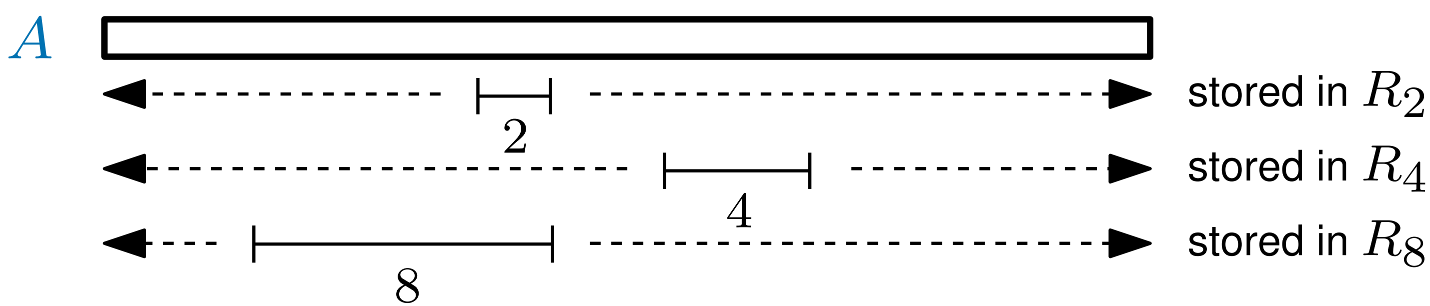
R_k stores RMQ$(i, i+k-1)$ for all i

We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

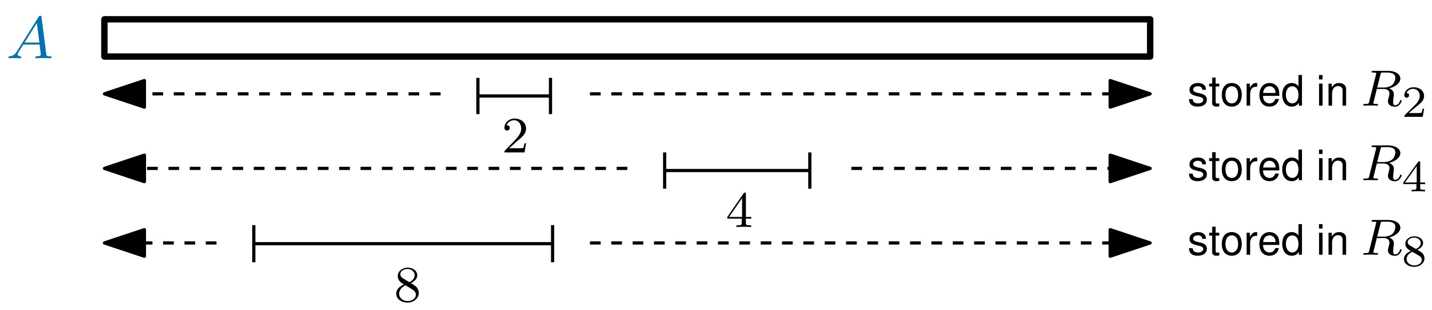
We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

We build R_2 from A in $O(n)$ time

R_k stores RMQ$(i, i+k-1)$ for all i

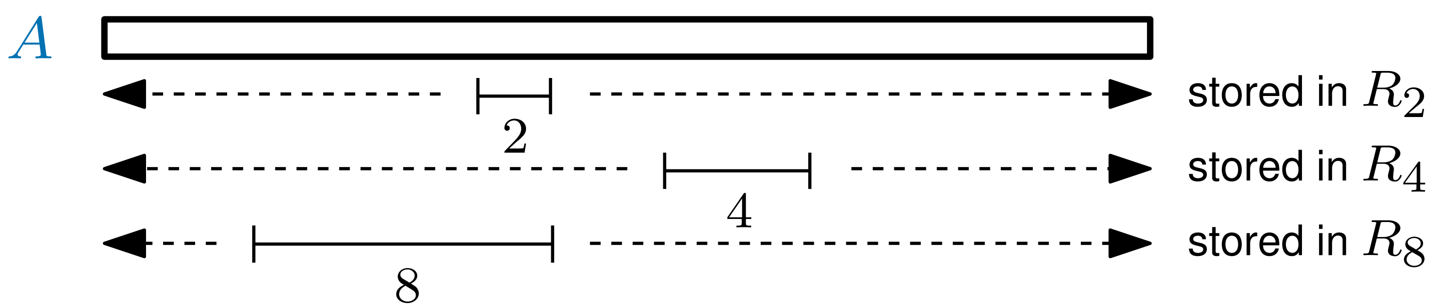
We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

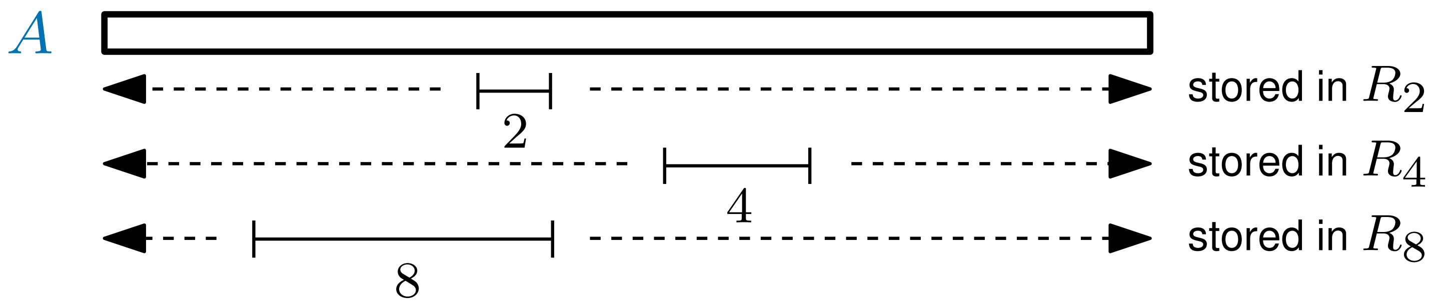
We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

R_4 stores $\text{RMQ}(i, i+3)$ for all i

R_8 stores $\text{RMQ}(i, i+7)$ for all i

R_k stores $\text{RMQ}(i, i+k-1)$ for all i

We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

how?

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$

A

The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

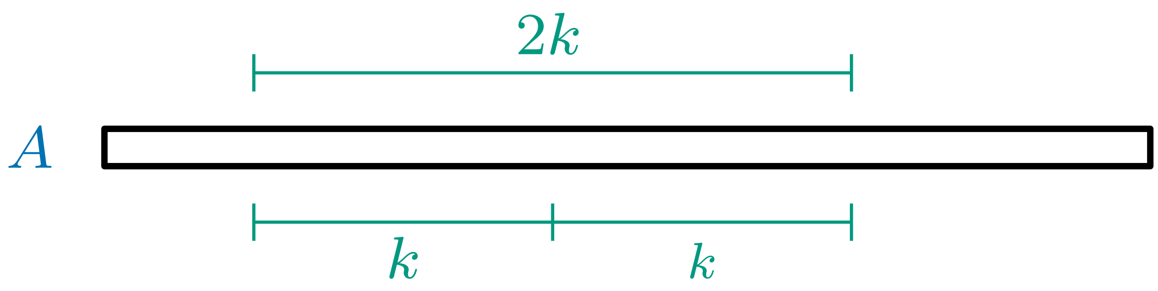
We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

how?

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16\ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ total space

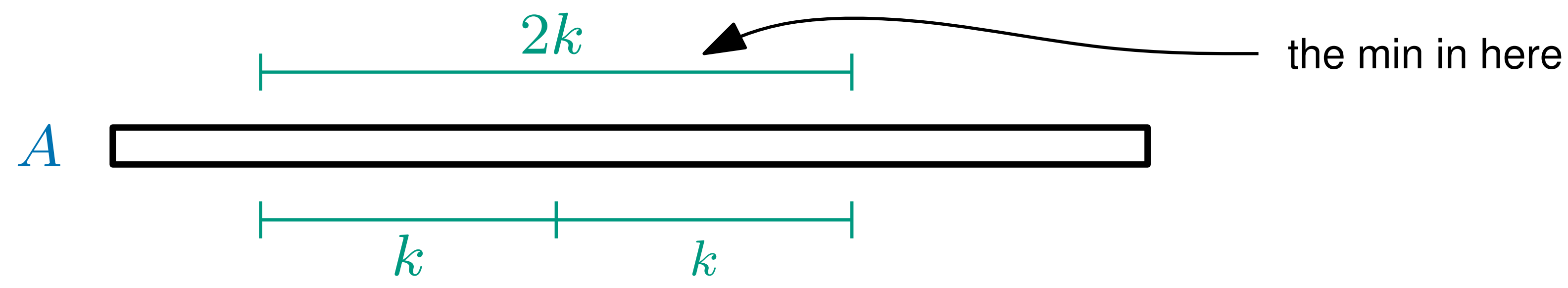
We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

how?

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



We build R_2 from A in $O(n)$ time

The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_{2k} from R_k in $O(n)$ time

how?

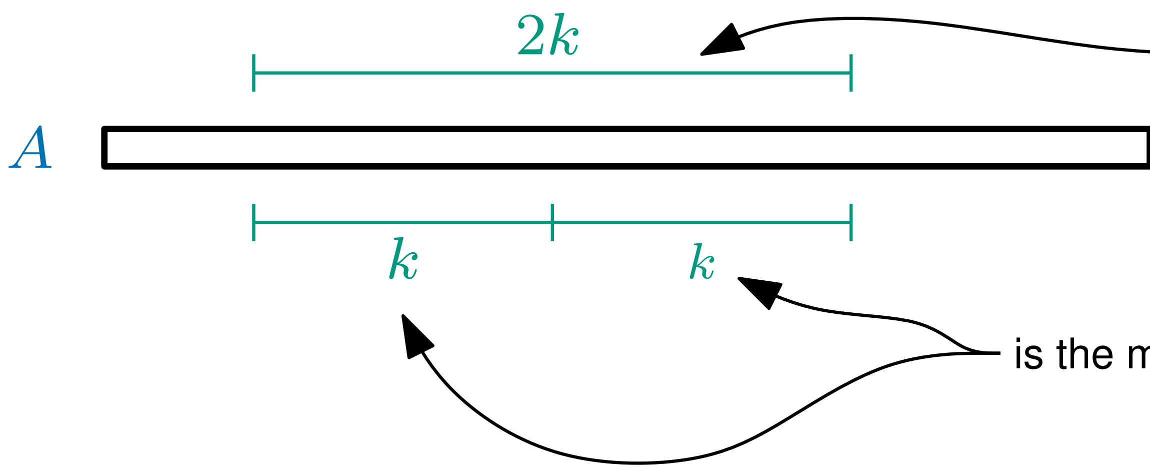
We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



the min in here

is the min of these two mins
(which we have in R_k)

The array R_2 stores $\text{RMQ}(i, i+1)$ for all i

R_4 stores $\text{RMQ}(i, i+3)$ for all i

R_8 stores $\text{RMQ}(i, i+7)$ for all i

R_k stores $\text{RMQ}(i, i+k-1)$ for all i

We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

how?

We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ total space

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$

A $\boxed{}$

The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

how?

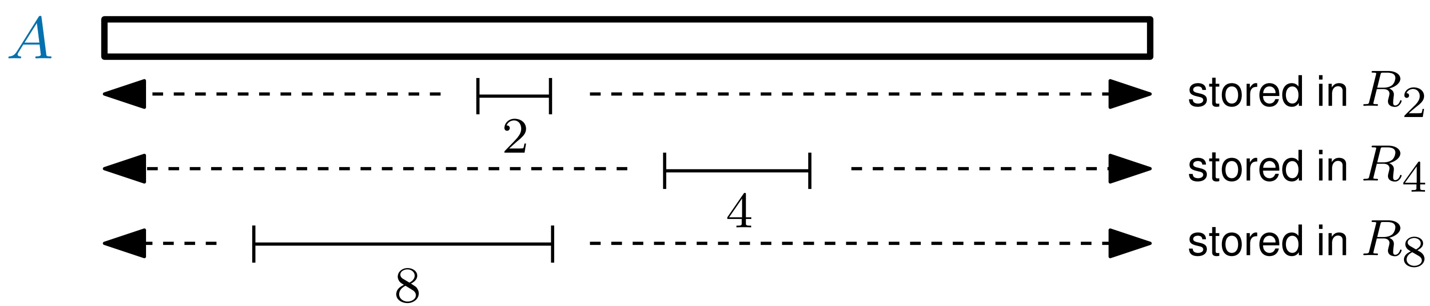
We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \ldots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_2 from A in $O(n)$ time

We build R_{2k} from R_k in $O(n)$ time

how?

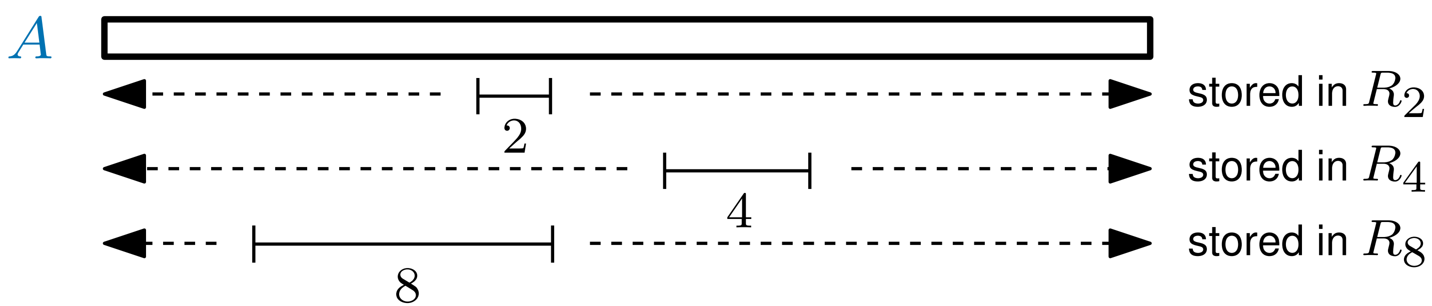
We build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

each of the $O(\log n)$ arrays uses $O(n)$ space

so $O(n \log n)$ *total space*

More space, faster queries

Key Idea precompute the answers for every interval of length $2, 4, 8, 16 \dots$



The array R_2 stores RMQ$(i, i+1)$ for all i

R_4 stores RMQ$(i, i+3)$ for all i

R_8 stores RMQ$(i, i+7)$ for all i

R_k stores RMQ$(i, i+k-1)$ for all i

We build R_2 from A in $O(n)$ time

We build R_k for $k = 2, 4, 8, 16 \dots \leqslant n$

We build R_{2k} from R_k in $O(n)$ time

how?

each of the $O(\log n)$ arrays uses $O(n)$ space

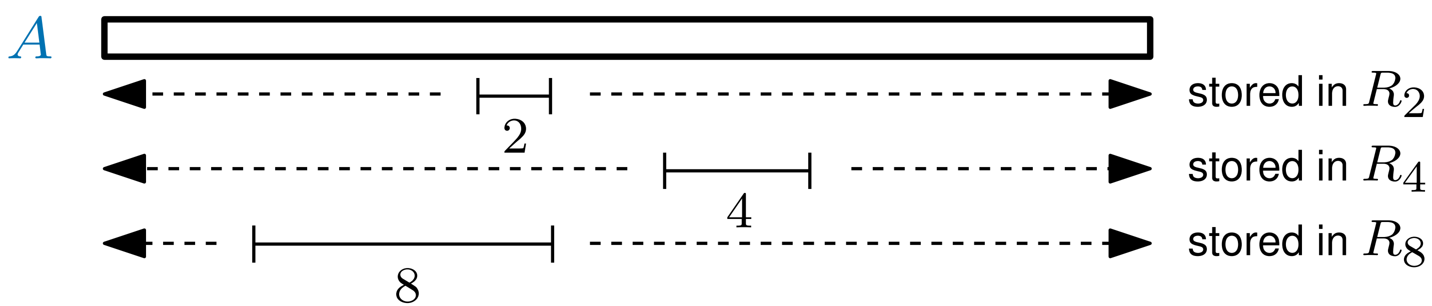
so $O(n \log n)$ total space

This takes $O(n \log n)$ prep time

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



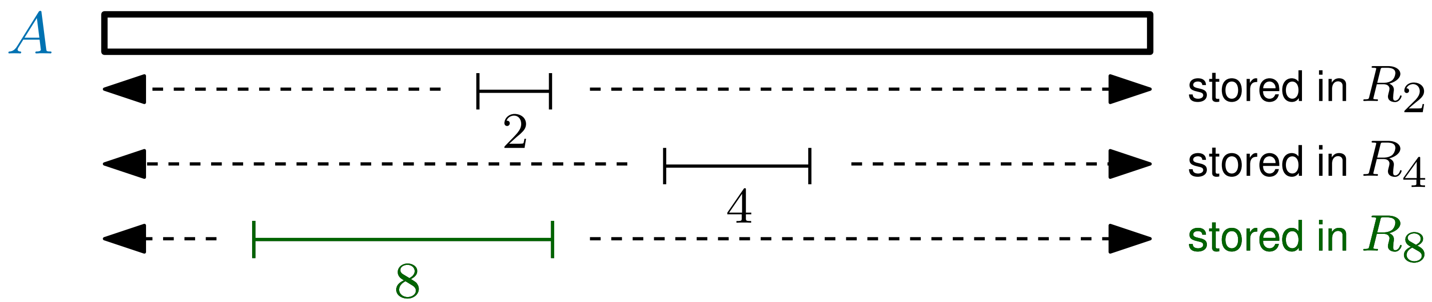
How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



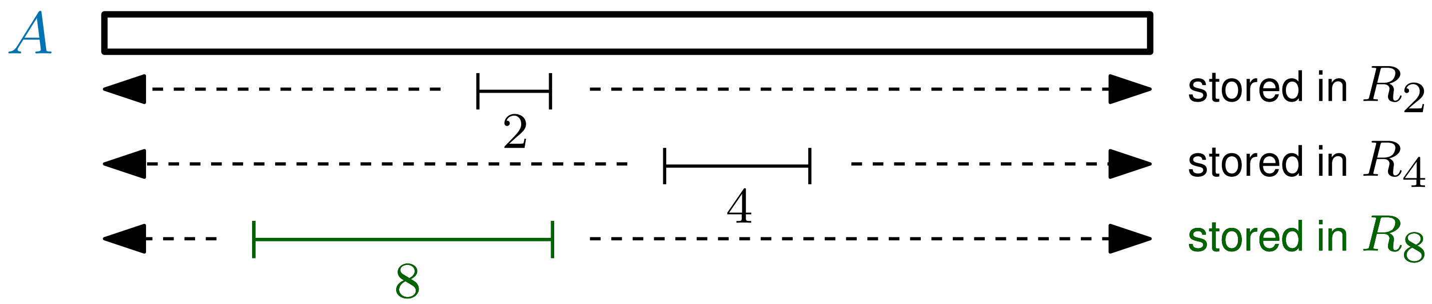
How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

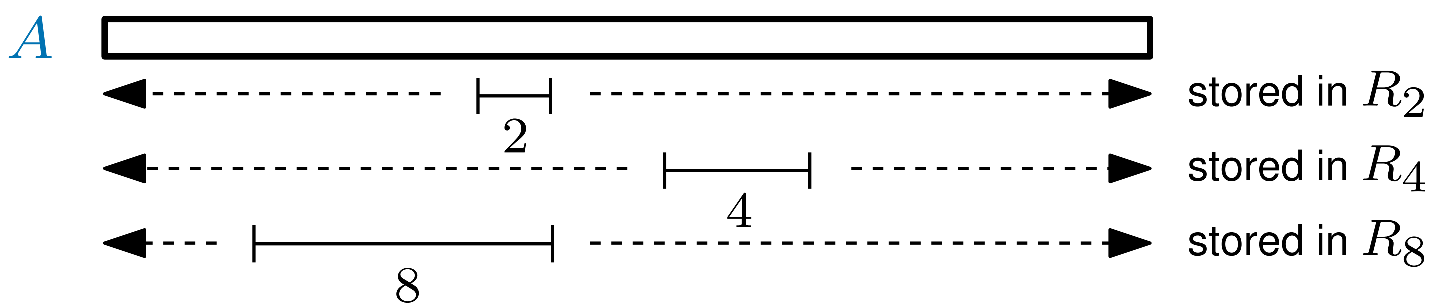
If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

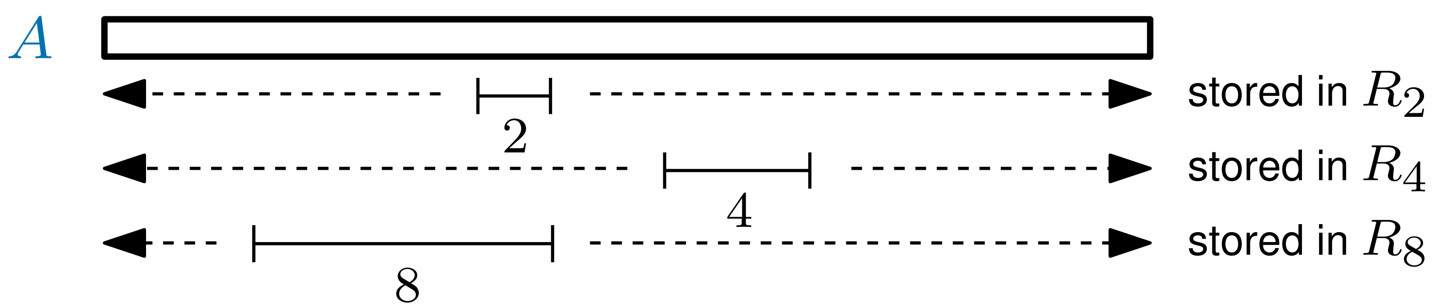
these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

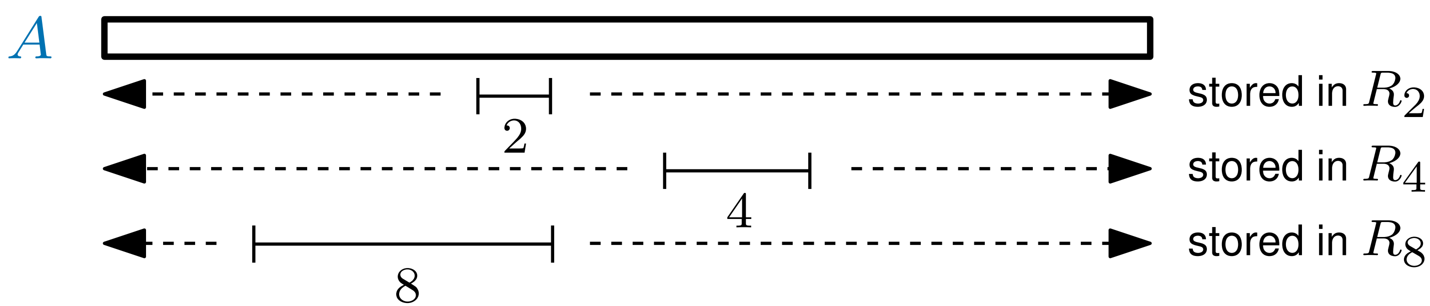
these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

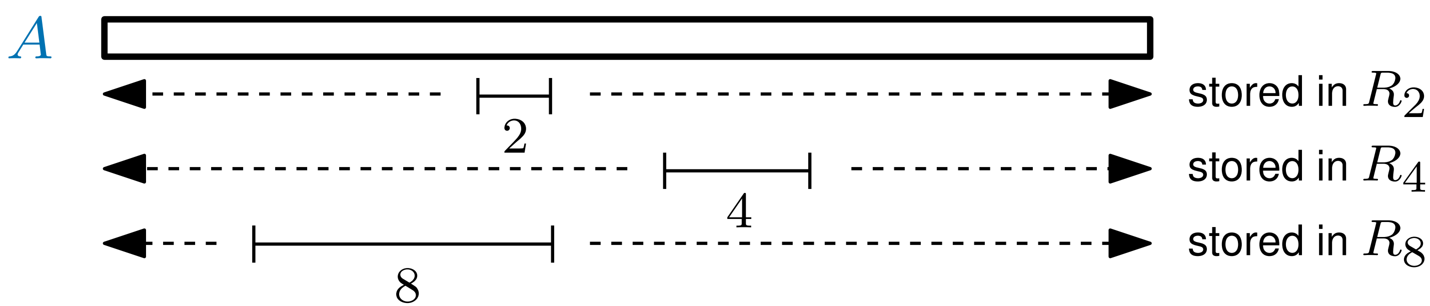
Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

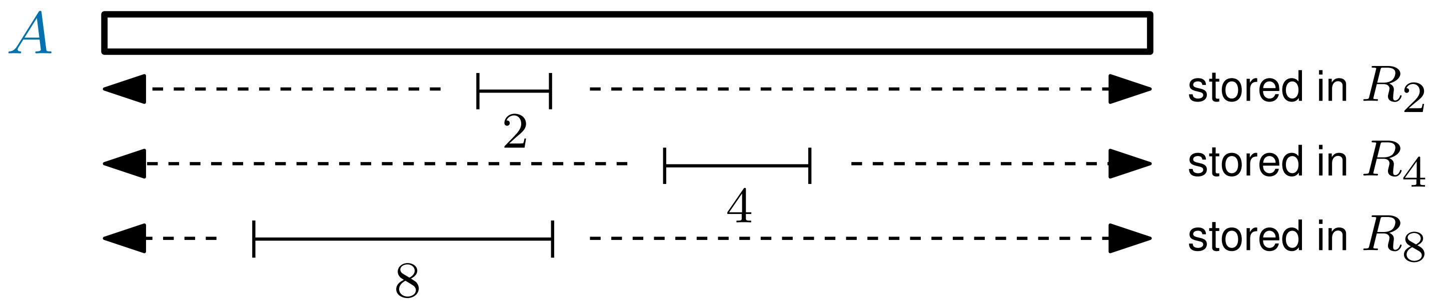
Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

(these two queries take $O(1)$ time)

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

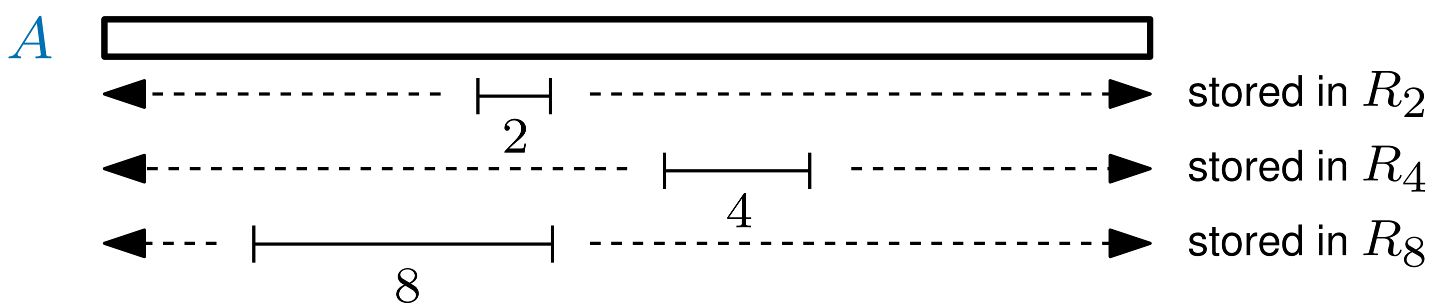
Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

(these two queries take $O(1)$ time)

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

A

How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$

How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

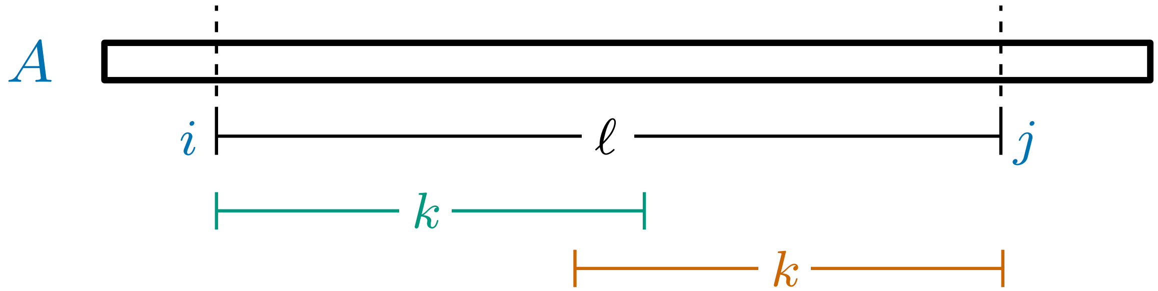
(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

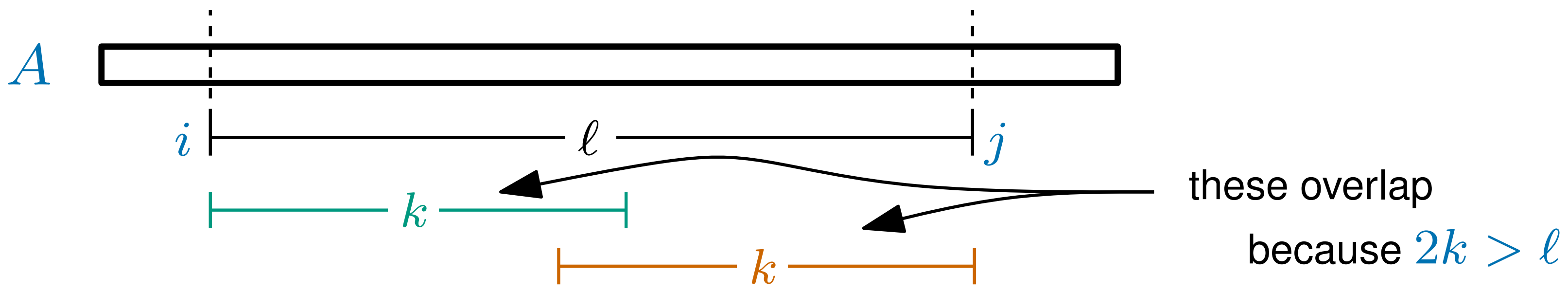
(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

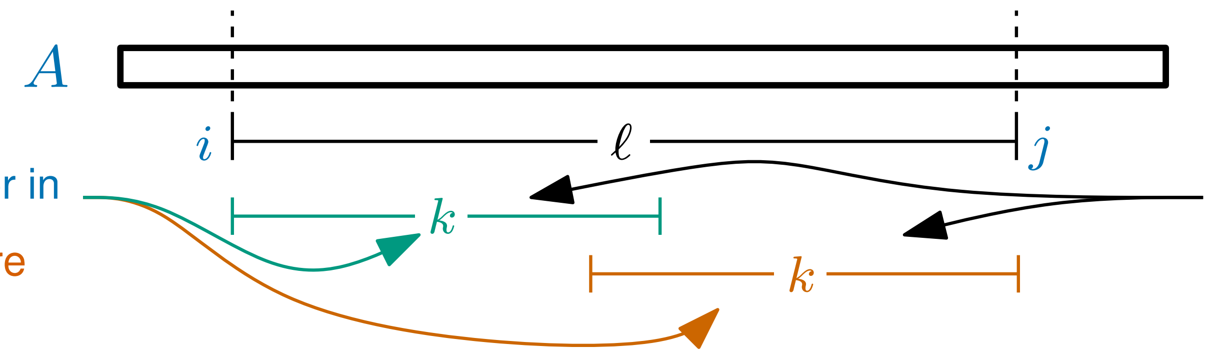
(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?

More space, faster queries

R_k stores RMQ$(i, i+k-1)$ for all i,

we build R_k for $k = 2, 4, 8, 16 \ldots \leqslant n$



the min is either in here or in here

these overlap because $2k > \ell$

How do we compute RMQ(i, j)?

If the interval length, $\ell = (j - i + 1)$, is a power-of-two - just look up the answer

these queries take $O(1)$ time

Otherwise, find the $k = 2, 4, 8, 16 \ldots$ such that $k \leqslant \ell < 2k$

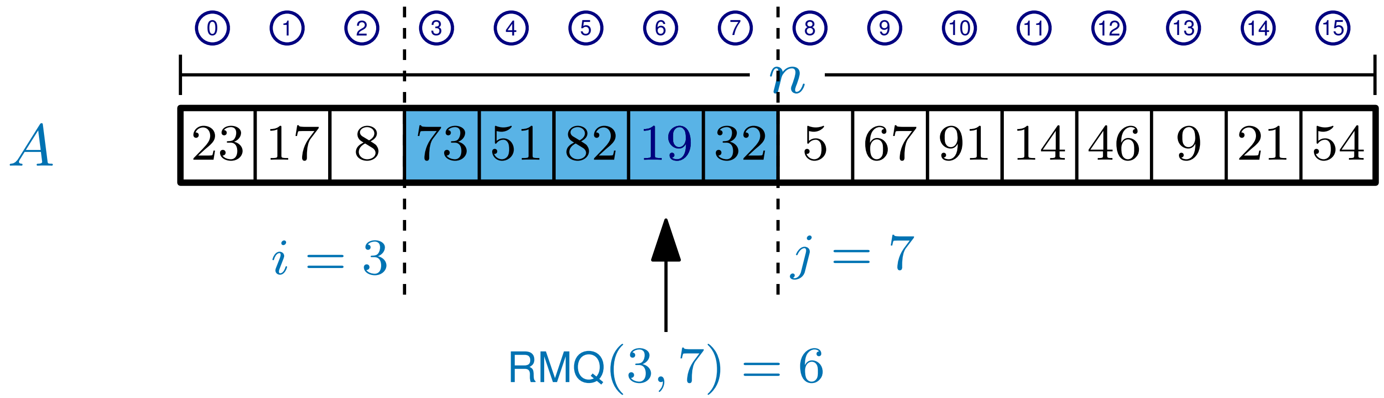
Compute the minimum of RMQ$(i, i+k-1)$ and RMQ$(j-k+1, j)$

(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?

Range minimum query (intermediate) summary

Preprocess an integer array A (length n) to answer range minimum queries...

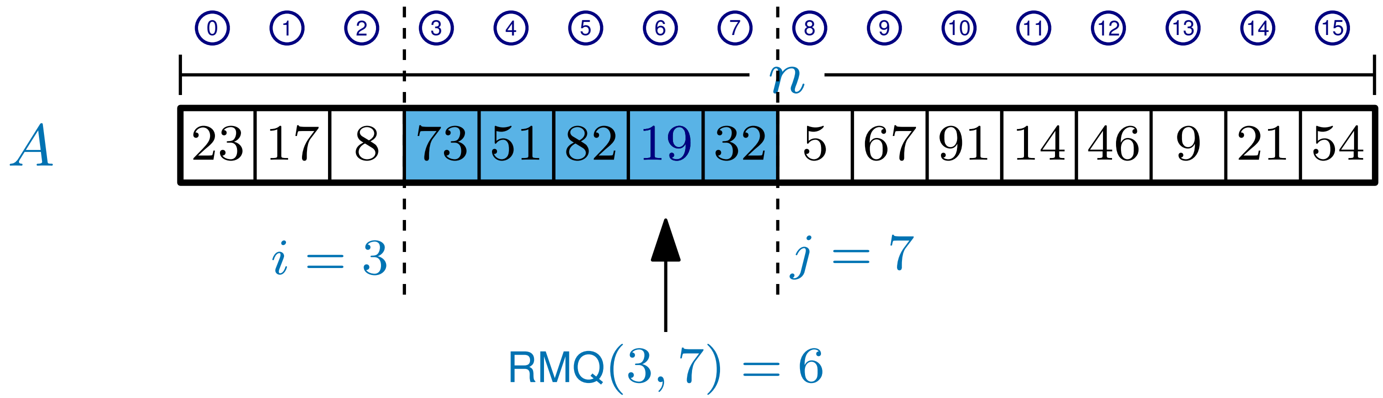


After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Range minimum query (intermediate) summary

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Solution 2

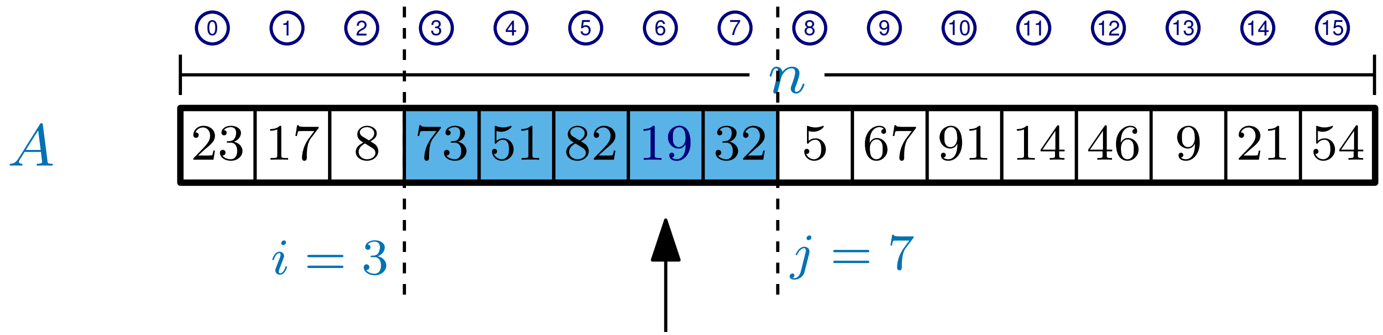
$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Range minimum query (intermediate) summary

Preprocess an integer array A (length n) to answer range minimum queries...



RMQ$(3, 7) = 6$

After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Can we do better?

Solution 2

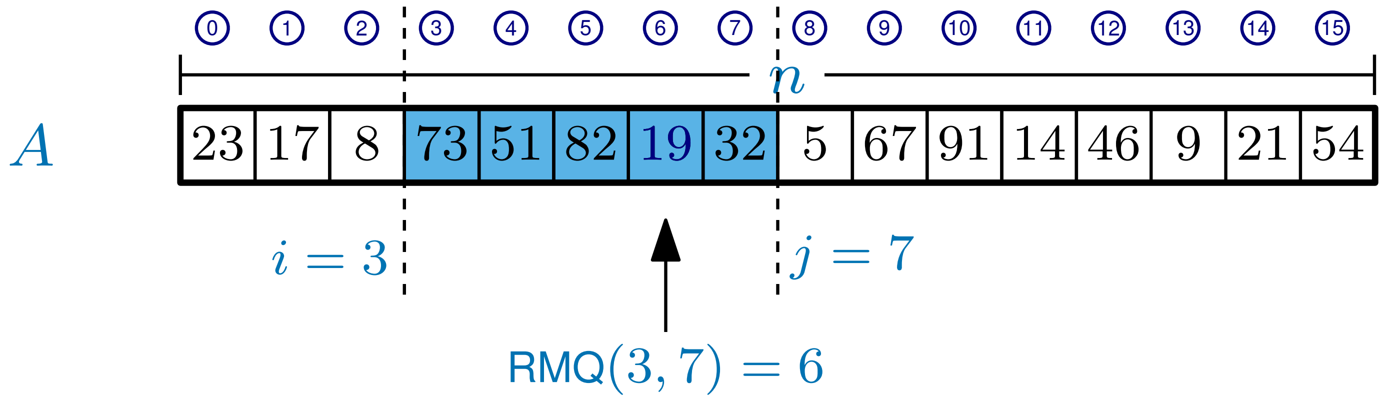
$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Range minimum query (intermediate) summary

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Can we do better?

(yes)

Solution 2

$O(n \log n)$ space

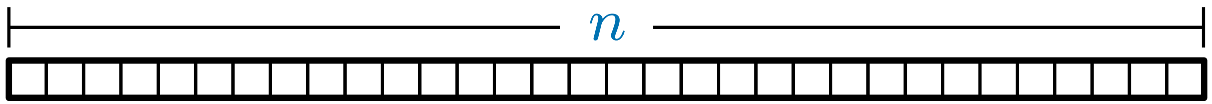
$O(n \log n)$ prep time

$O(1)$ query time

Low-resolution RMQ

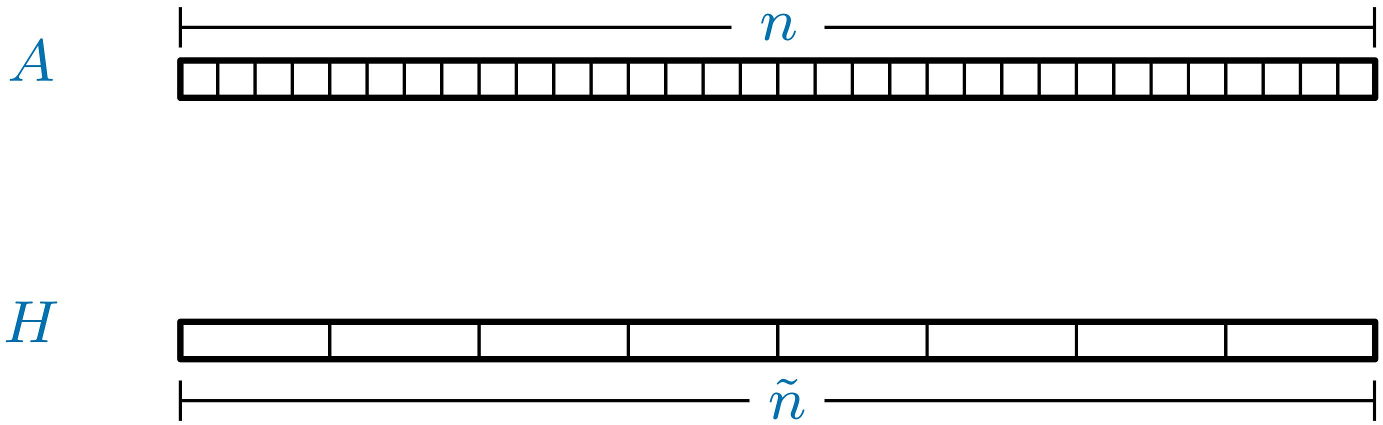
Key Idea replace A with a smaller, *'low resolution'* array H

A



Low-resolution RMQ

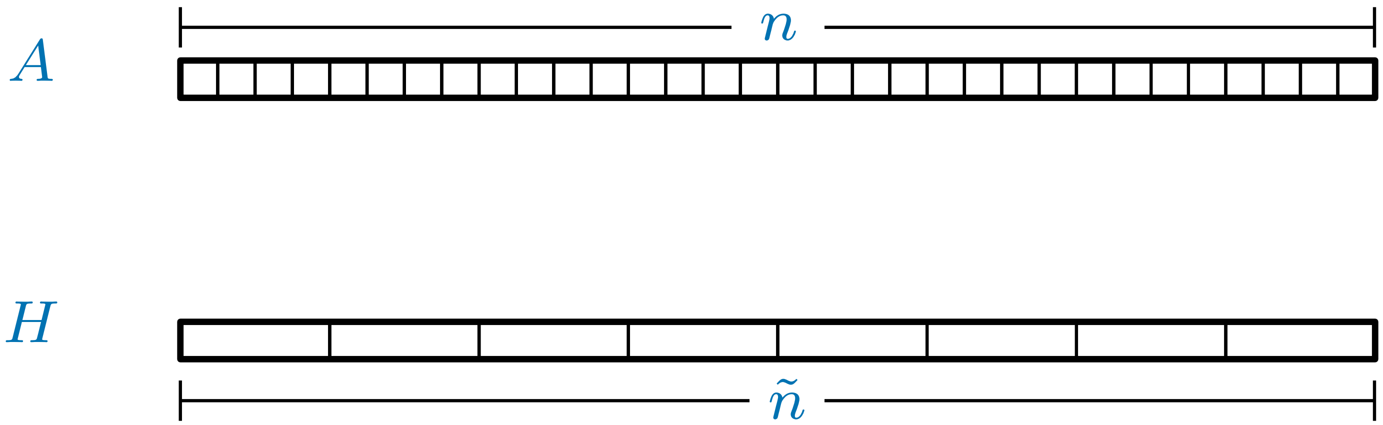
Key Idea replace A with a smaller, *'low resolution'* array H



Low-resolution RMQ

Key Idea replace A with a smaller, *'low resolution'* array H

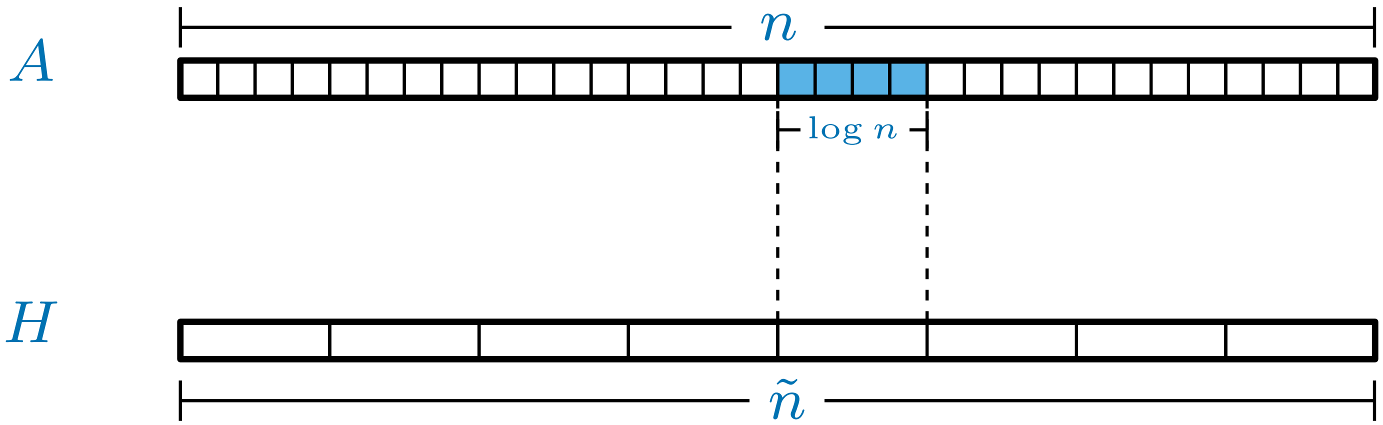
$$\boxed{\tilde{n} = \frac{n}{\log n}}$$



Low-resolution RMQ

Key Idea replace A with a smaller, *'low resolution'* array H

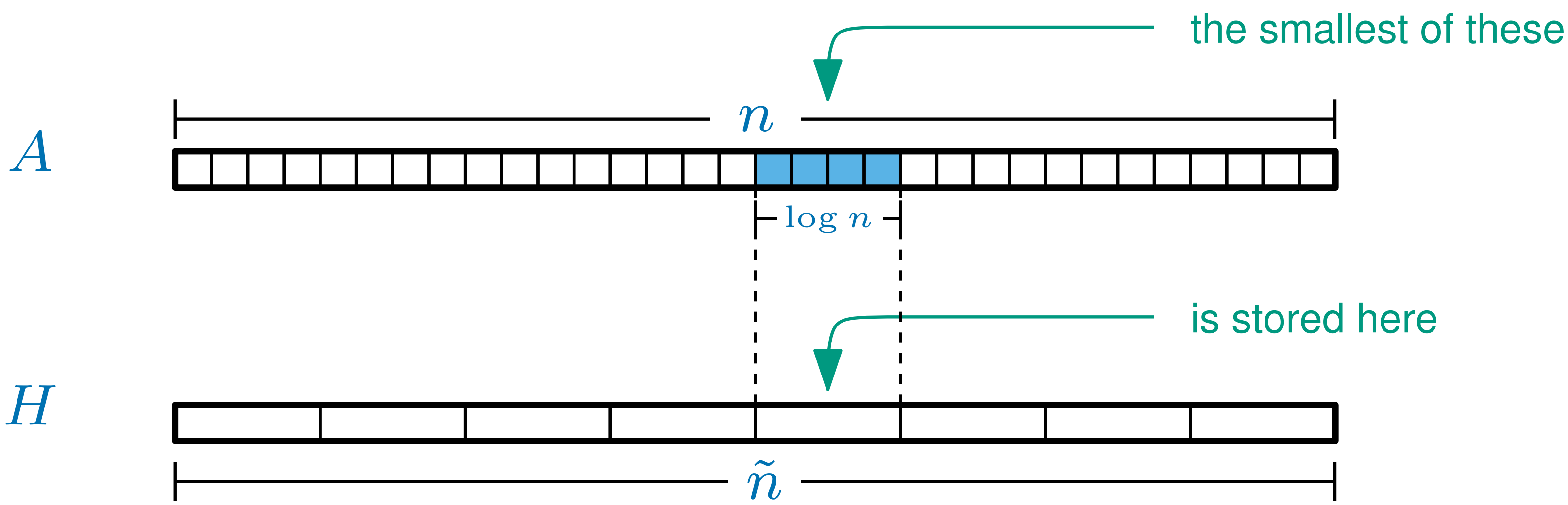
$$\tilde{n} = \frac{n}{\log n}$$



Low-resolution RMQ

Key Idea replace A with a smaller, *'low resolution'* array H

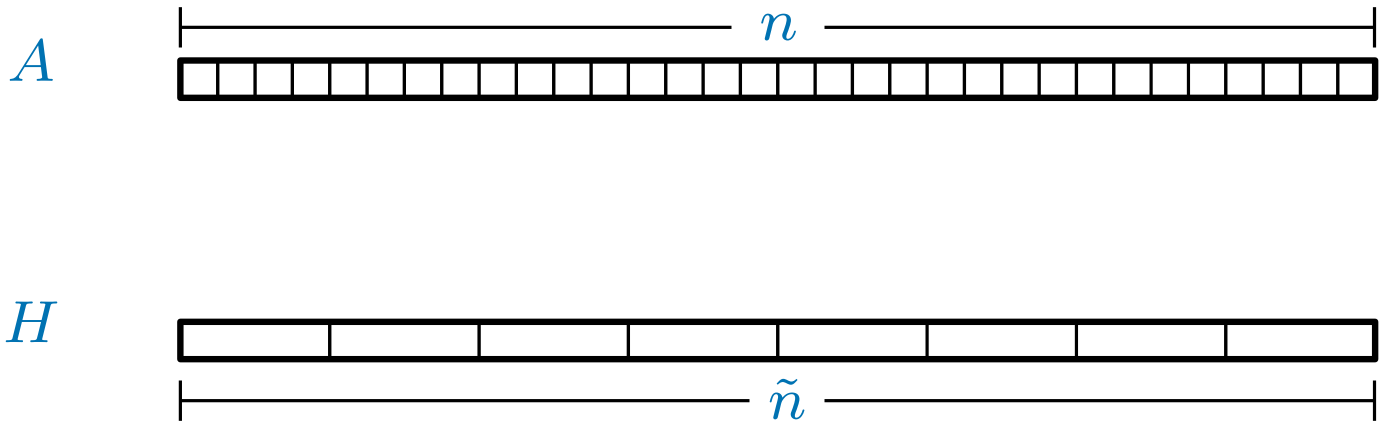
$$\tilde{n} = \frac{n}{\log n}$$



Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

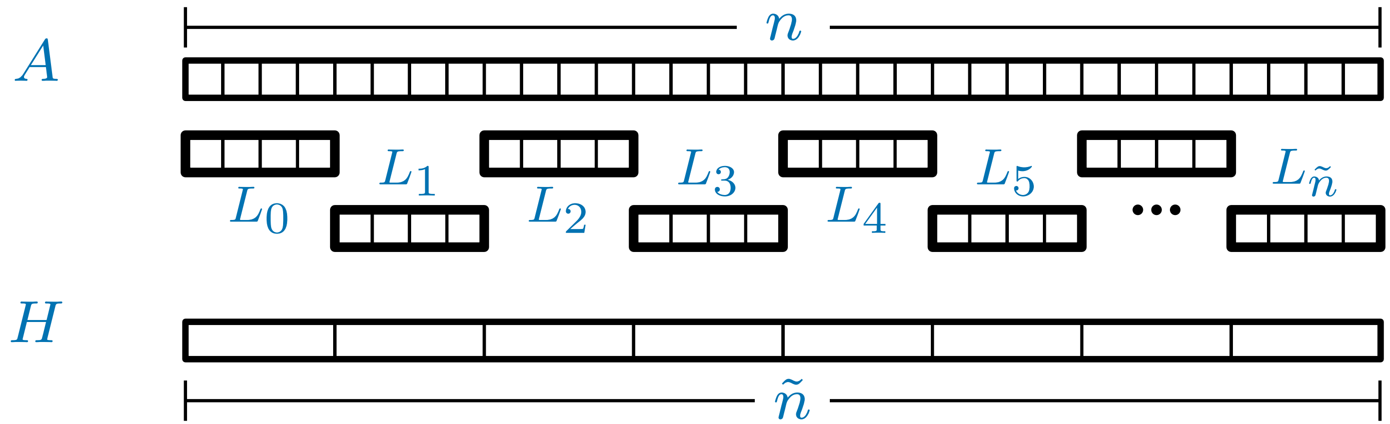


Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*

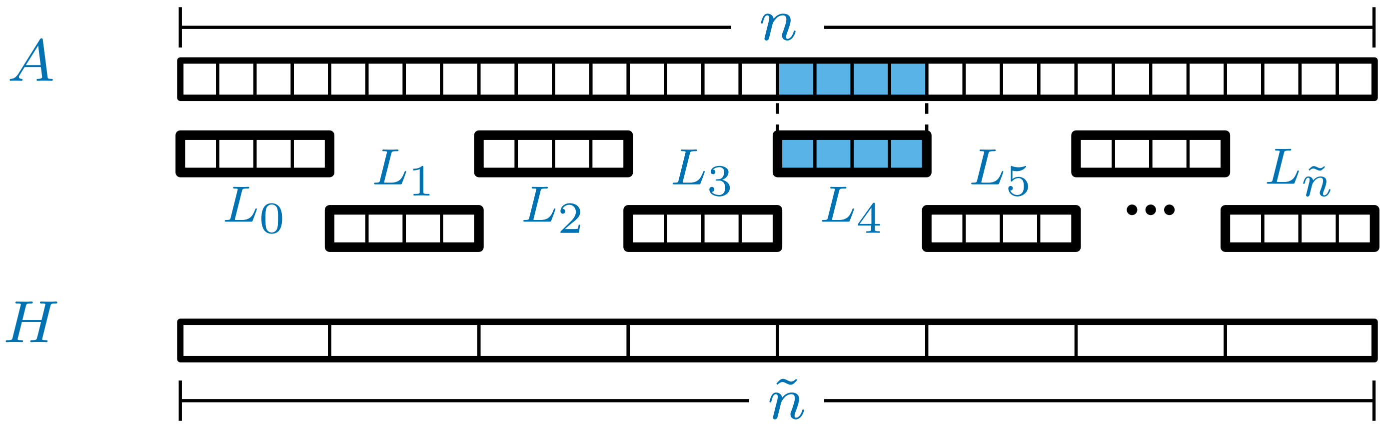


Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*

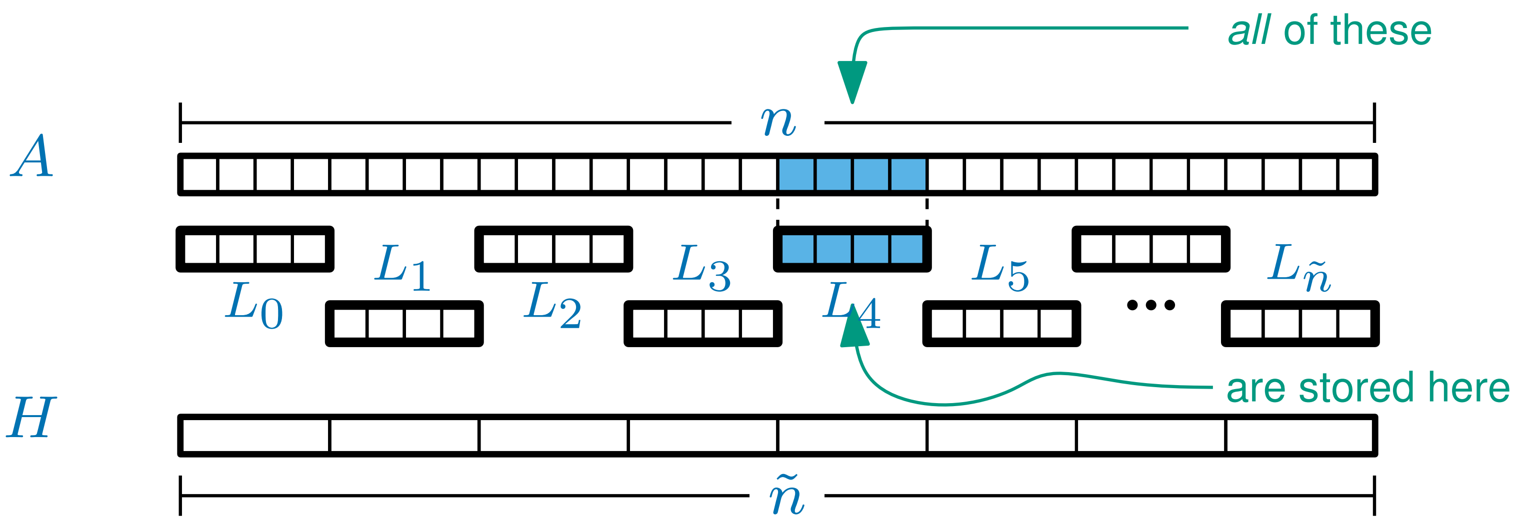


Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*

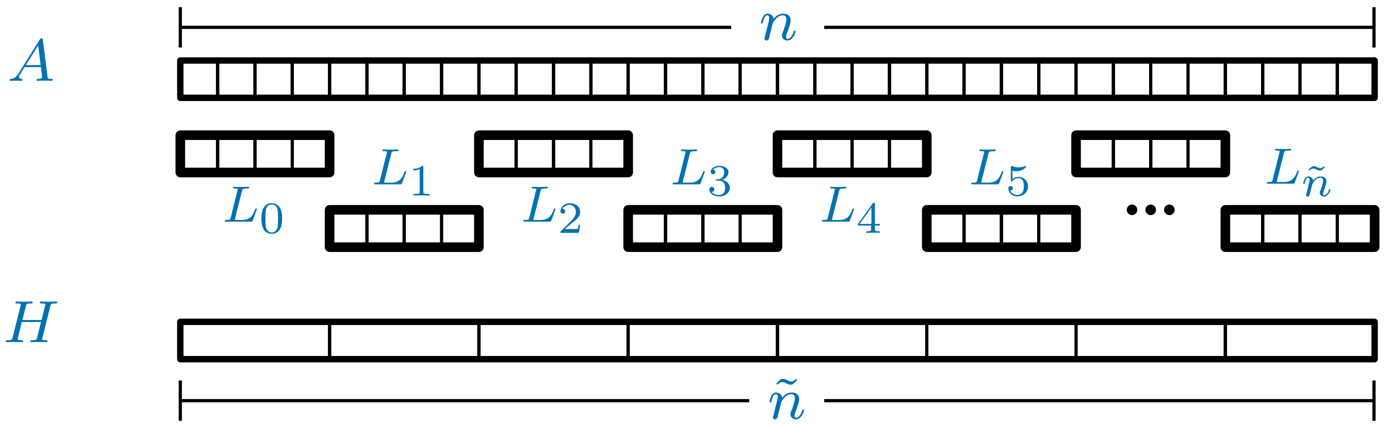


Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*

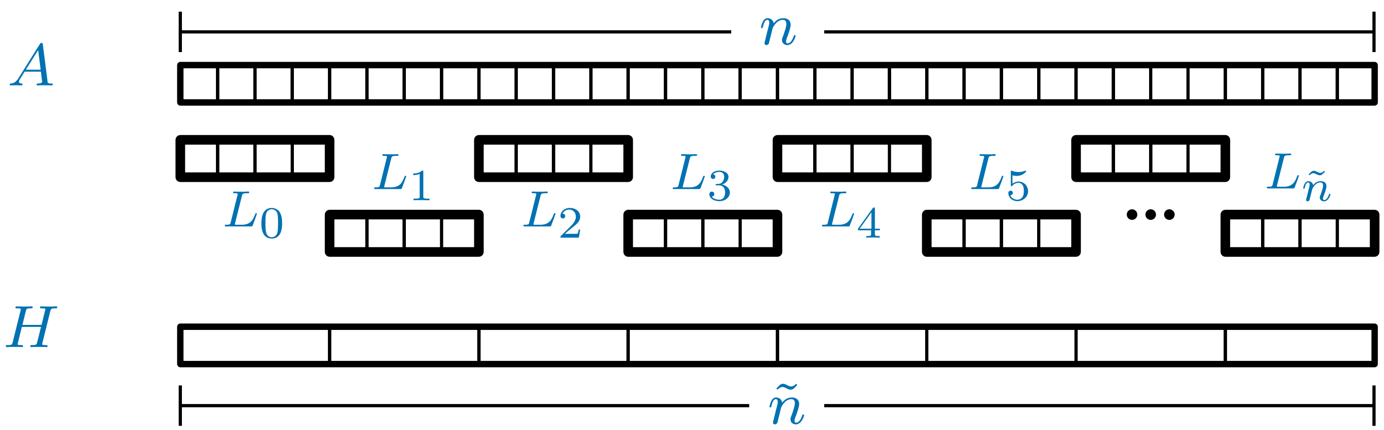


Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



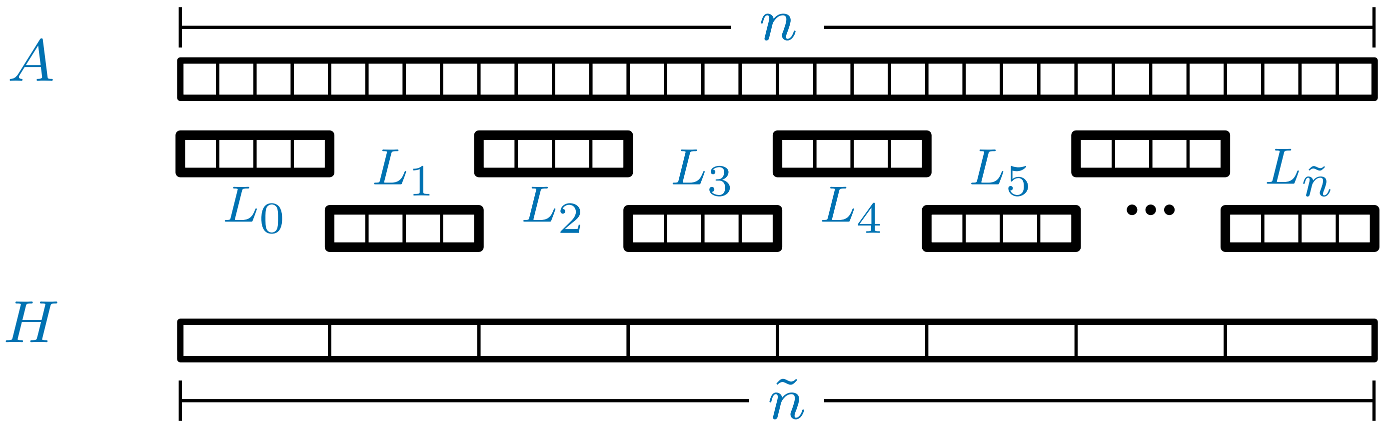
Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs. . .

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

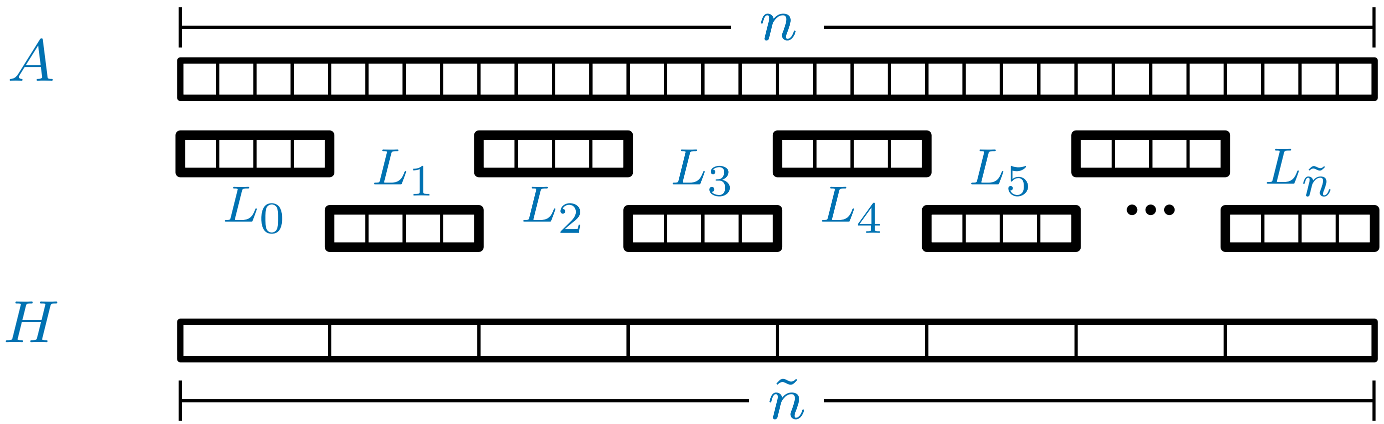
using **Solution 2**

Low-resolution RMQ

$$\boxed{\tilde{n} = \frac{n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs. . .

using **Solution 2**

Recall. . .

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

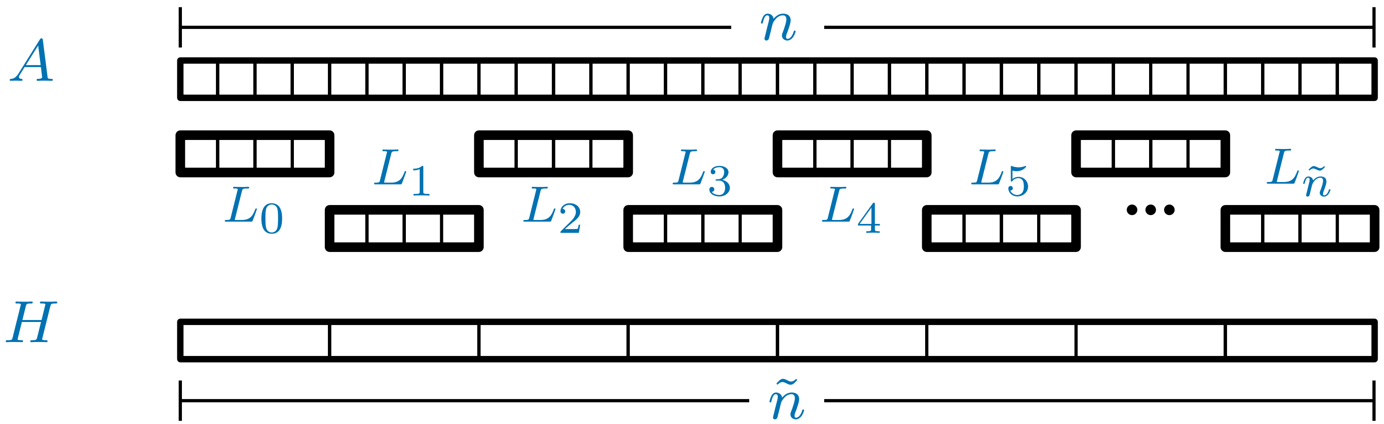
$O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2**

Recall...

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

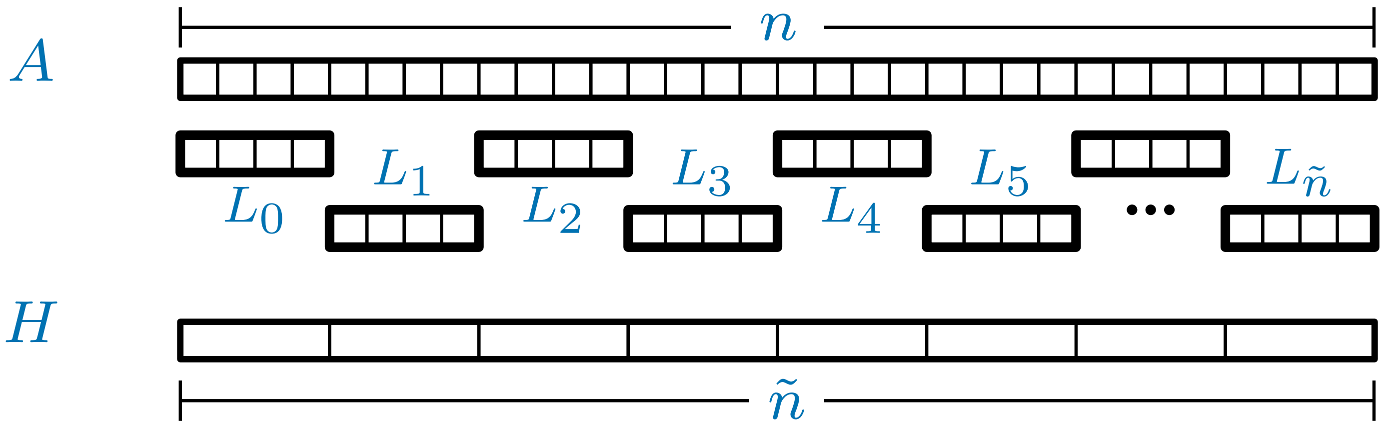
$O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2**

Recall...

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Solution 2 on H

$O(\tilde{n} \log \tilde{n})$ space

$O(\tilde{n} \log \tilde{n})$ prep time

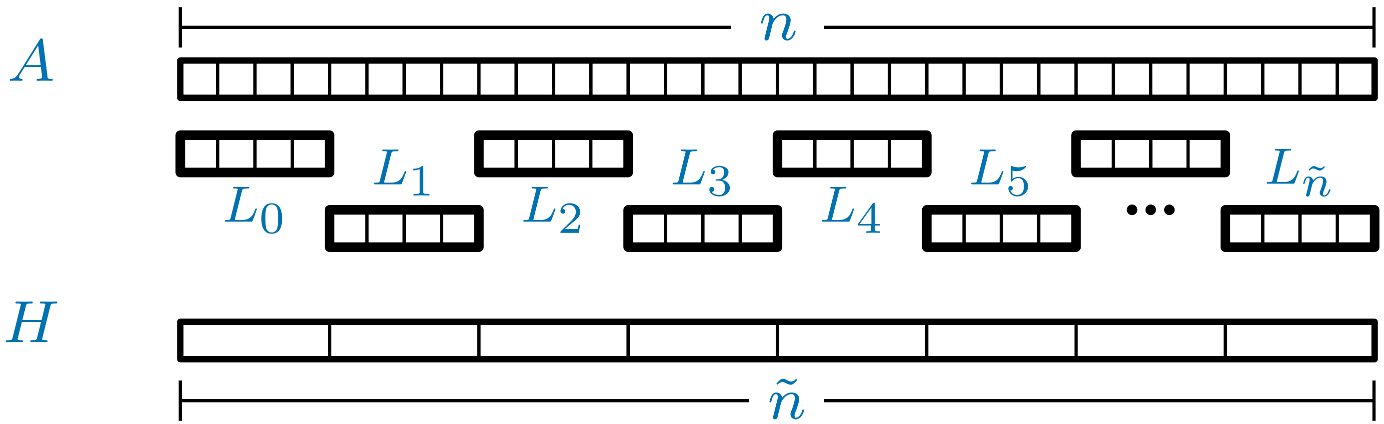
$O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2**

Recall...

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Solution 2 on H

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

$O(\tilde{n} \log \tilde{n})$ prep time

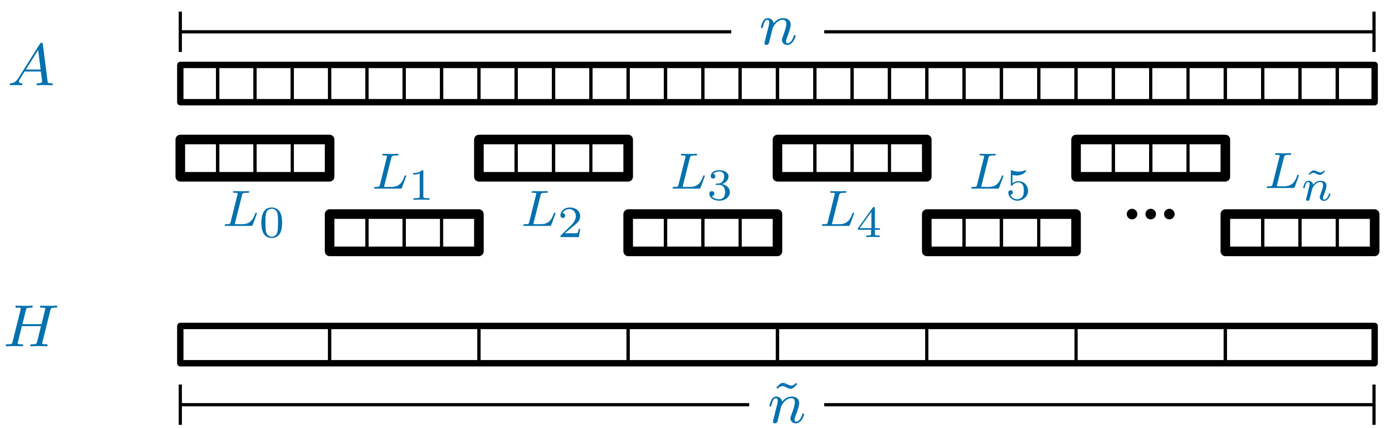
$O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2**

Recall...

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Solution 2 on H

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

$O(\tilde{n} \log \tilde{n})$ prep time

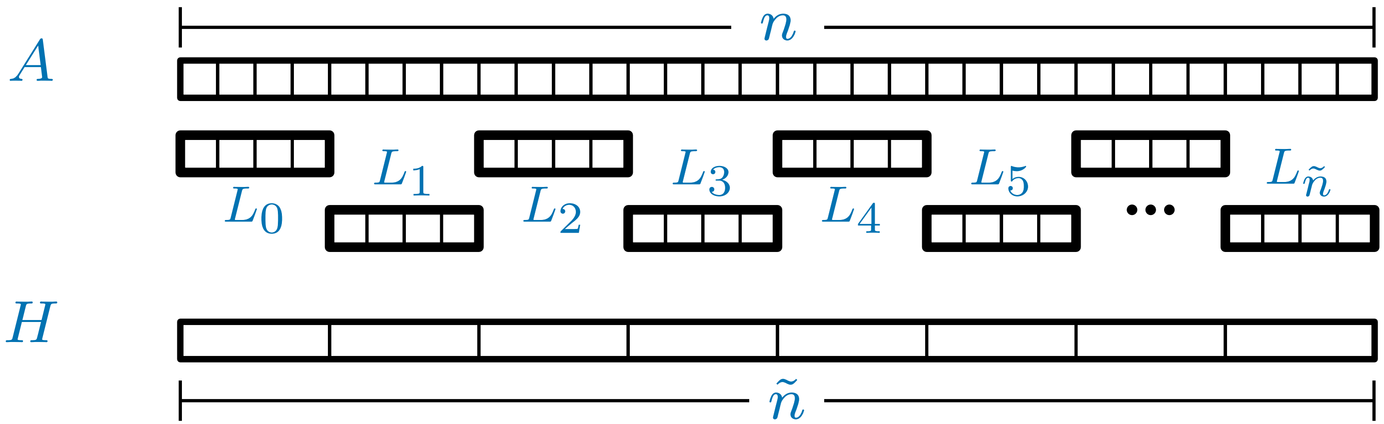
$O(1)$ query time

Low-resolution RMQ

$$\boxed{\tilde{n} = \frac{n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2**

Recall...

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Solution 2 on H

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

$O(\tilde{n} \log \tilde{n})$ prep time $= O(n)$

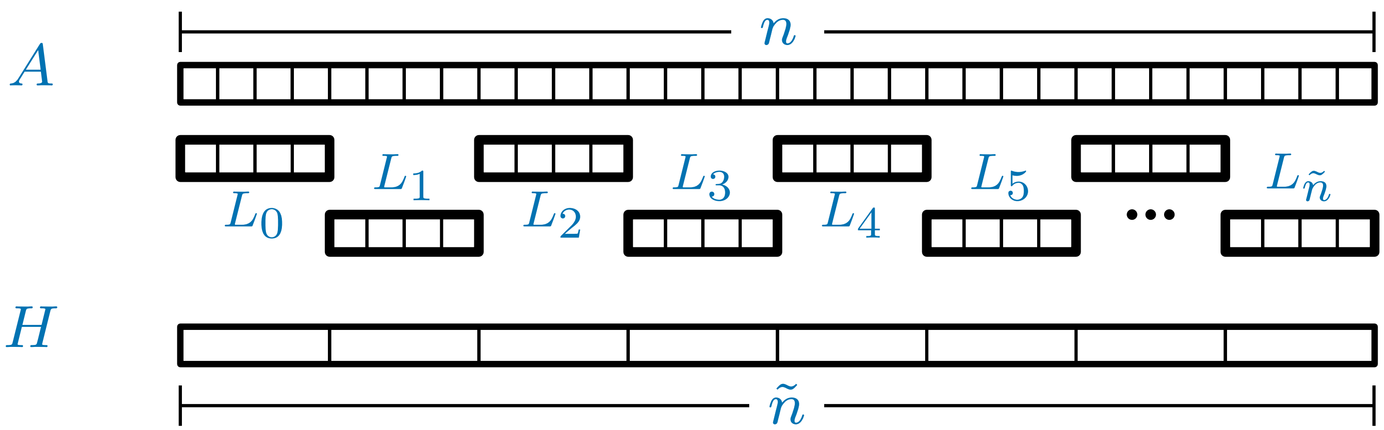
$O(1)$ query time

Low-resolution RMQ

$$\boxed{\tilde{n} = \frac{n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2** in $O(n)$ space/prep time

Recall...

Solution 2 on A

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Solution 2 on H

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

$O(\tilde{n} \log \tilde{n})$ prep time $= O(n)$

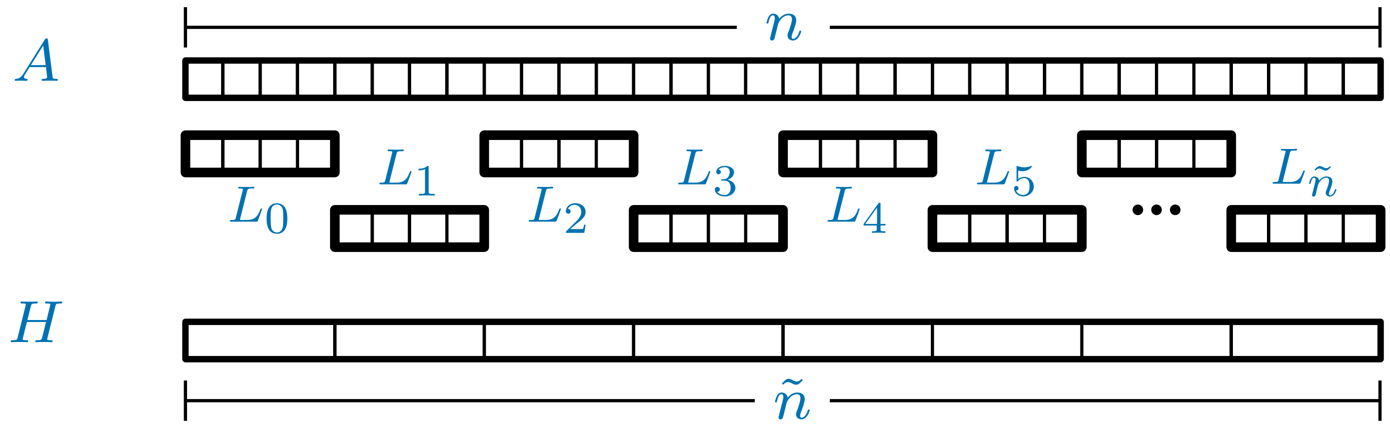
$O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

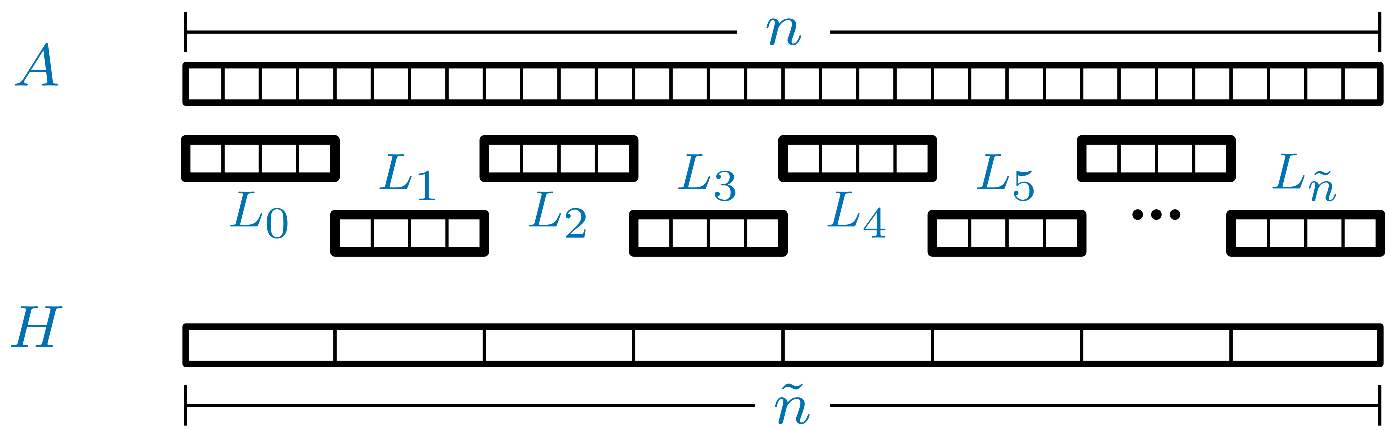
using **Solution 2** in $O(n)$ space/prep time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs...

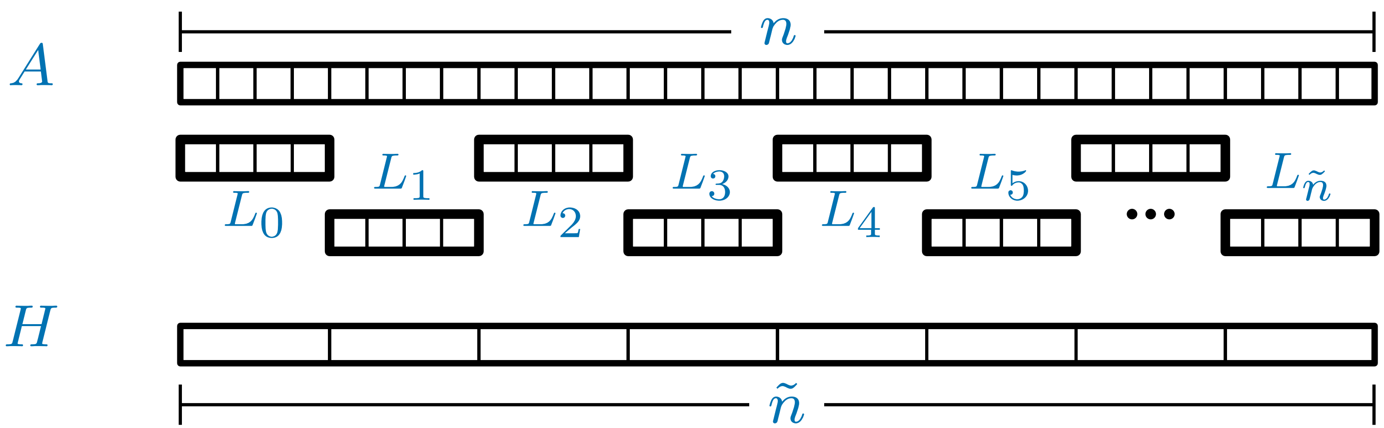
using **Solution 2**

Low-resolution RMQ

$$\boxed{\tilde{n} = \frac{n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs. . .

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs. . .

using **Solution 2**

Solution 2 on L_i

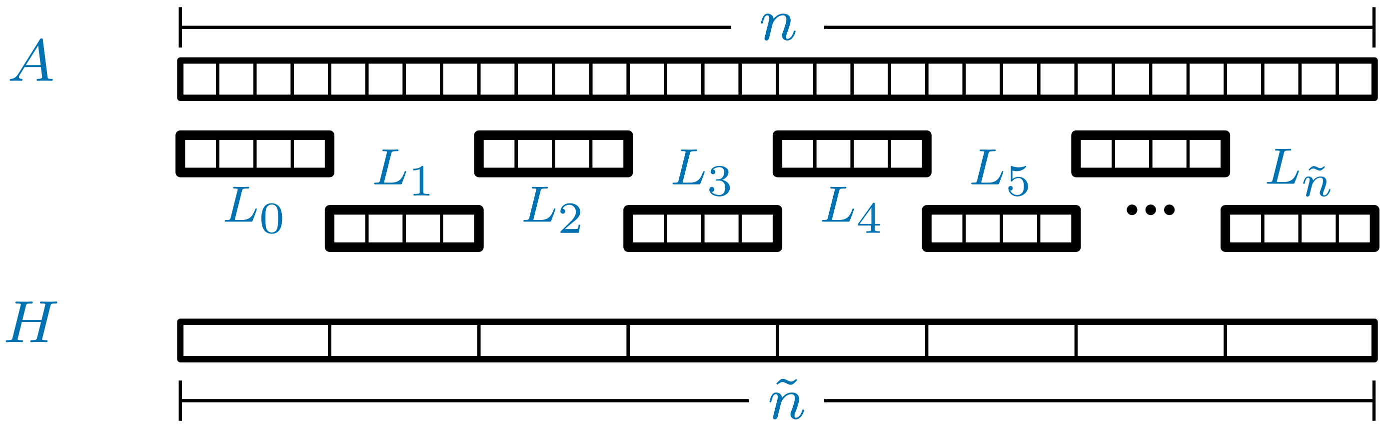
$O((\log n) \log\log n))$ space/prep time $\quad$ $O(1)$ query time

Low-resolution RMQ

$\tilde{n} = \frac{n}{\log n}$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs. . .

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs. . .

using **Solution 2** in $O(\log n \log \log n)$ space/prep time

Solution 2 on L_i

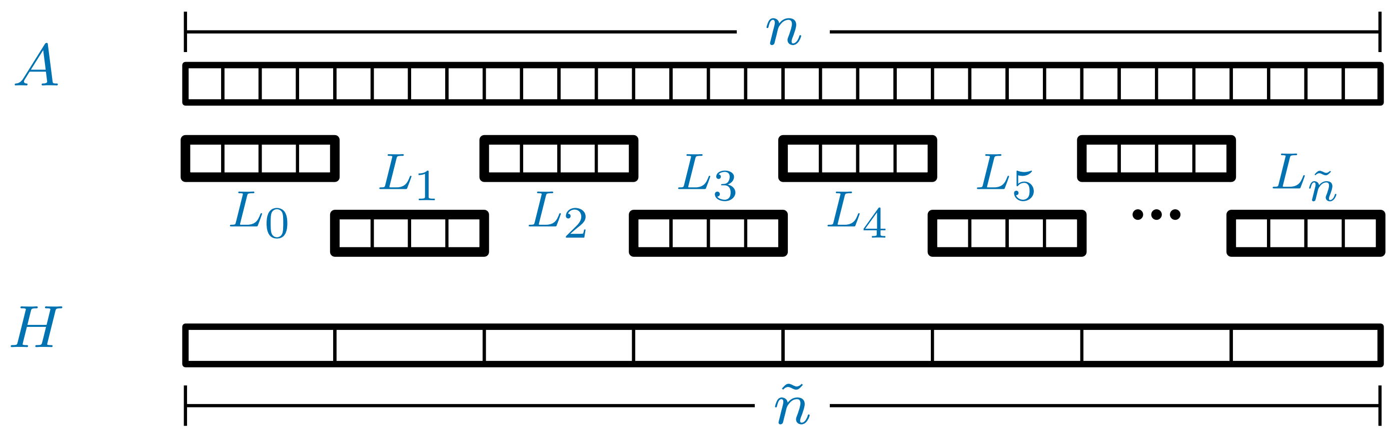
$O((\log n) \log \log n))$ space/prep time $\quad$ $O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs. . .

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs. . .

using **Solution 2** in $O(\log n \log \log n)$ space/prep time

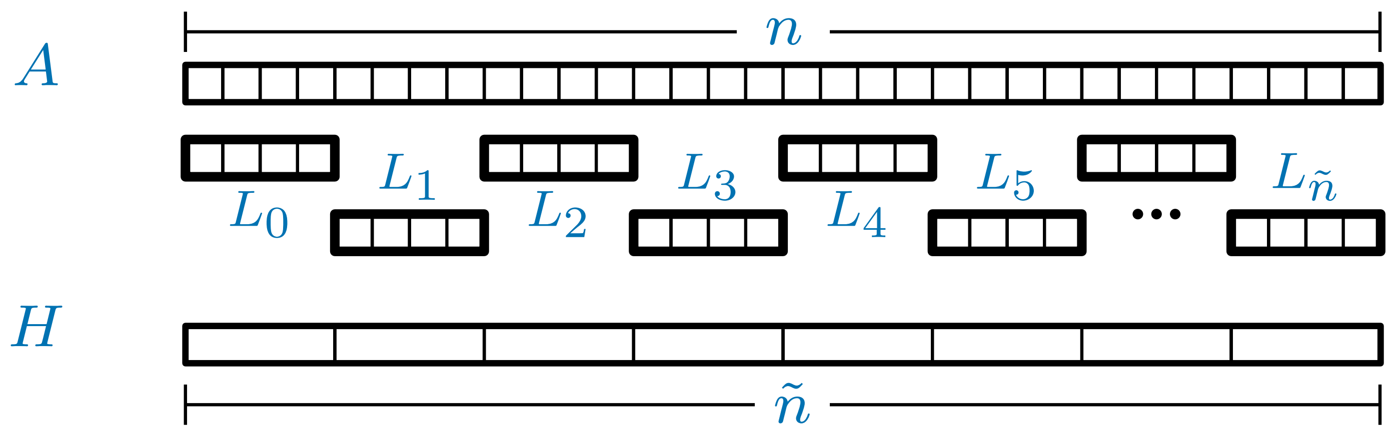
Total space = $O(n) + O(\tilde{n} \log n \log \log n)$

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs...

using **Solution 2** in $O(\log n \log \log n)$ space/prep time

Total space = $O(n) + O(\tilde{n} \log n \log \log n)$

space for RMQ structure for H

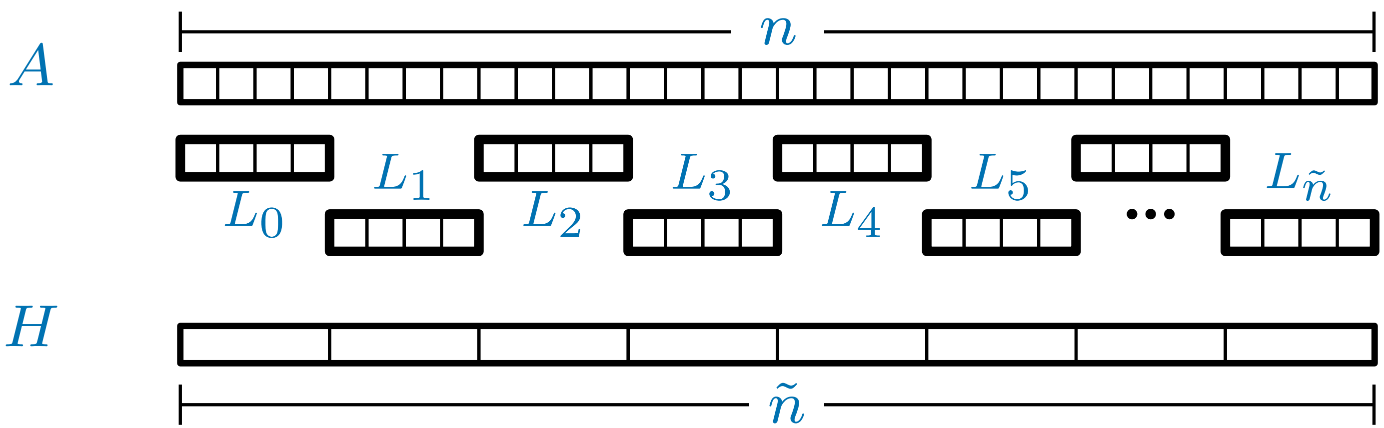
space for RMQ structures for all the L_i arrays

Low-resolution RMQ

$$\boxed{\tilde{n} = \frac{n}{\log n}}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs. . .

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs. . .

using **Solution 2** in $O(\log n \log \log n)$ space/prep time

Total space = $O(n) + O(\tilde{n} \log n \log \log n) = O(n \log \log n)$

space for RMQ structure for H

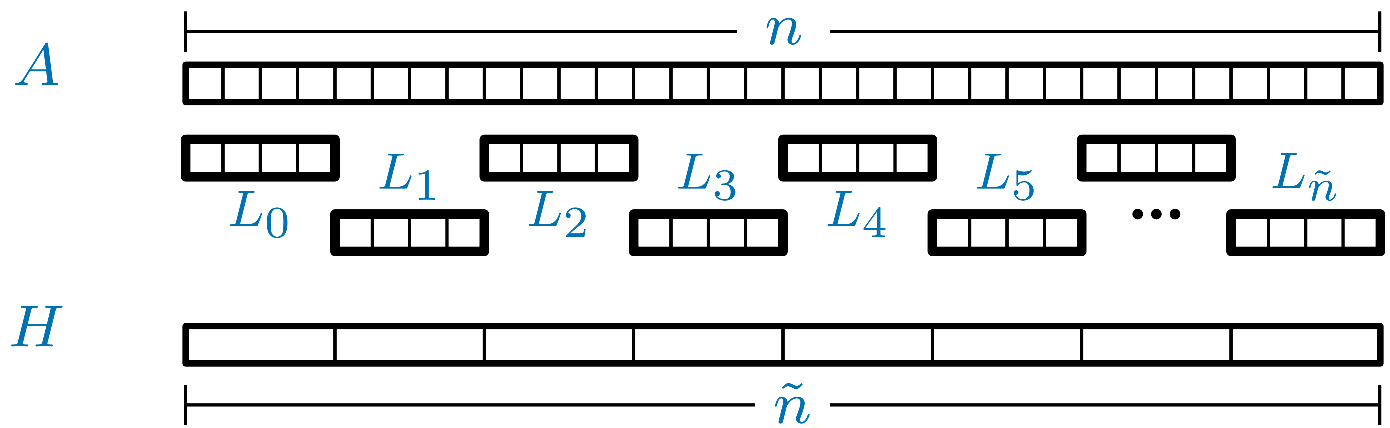
space for RMQ structures for all the L_i arrays

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs...

using **Solution 2** in $O(\log n \log\log n)$ space/prep time

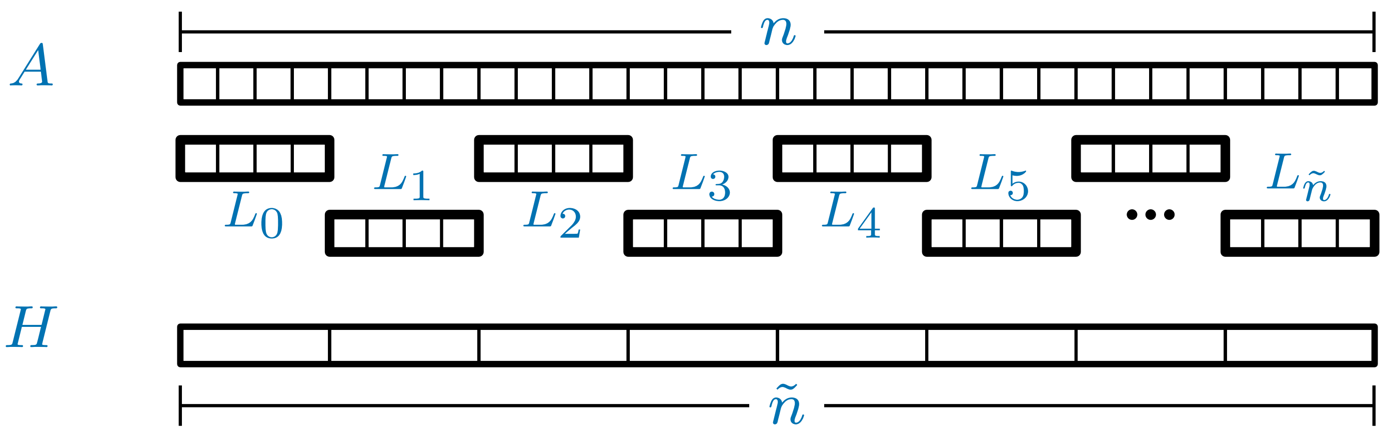
Total space = $O(n) + O(\tilde{n} \log n \log\log n) = O(n \log\log n)$

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



Preprocess the array H (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs...

using **Solution 2** in $O(n)$ space/prep time

Preprocess each array L_i (which has length $\log n$) to answer RMQs...

using **Solution 2** in $O(\log n \log \log n)$ space/prep time

Total space = $O(n) + O(\tilde{n} \log n \log \log n) \ = O(n \log \log n)$

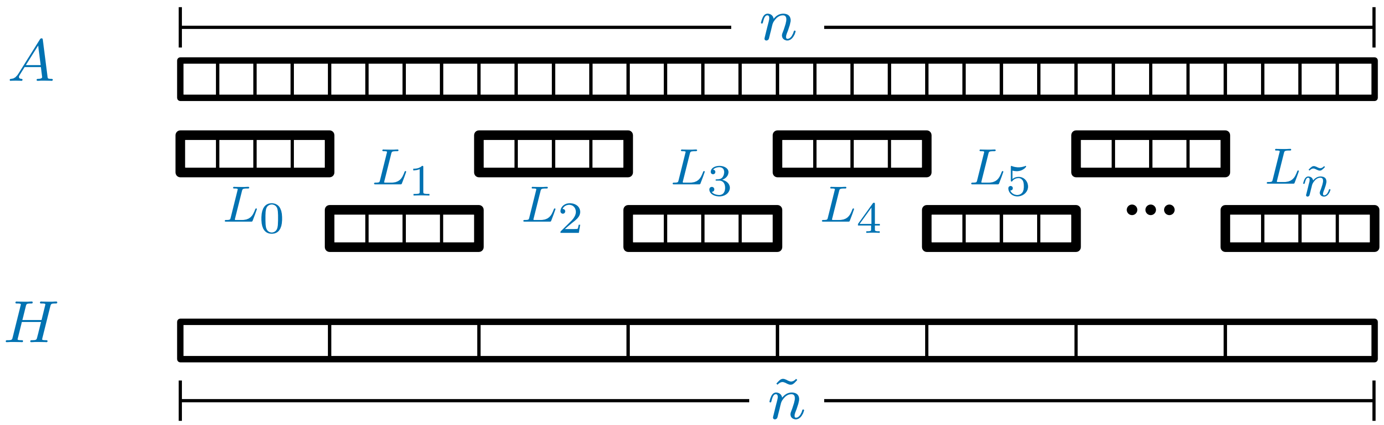
Total prep. time = $O(n \log \log n)$

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*

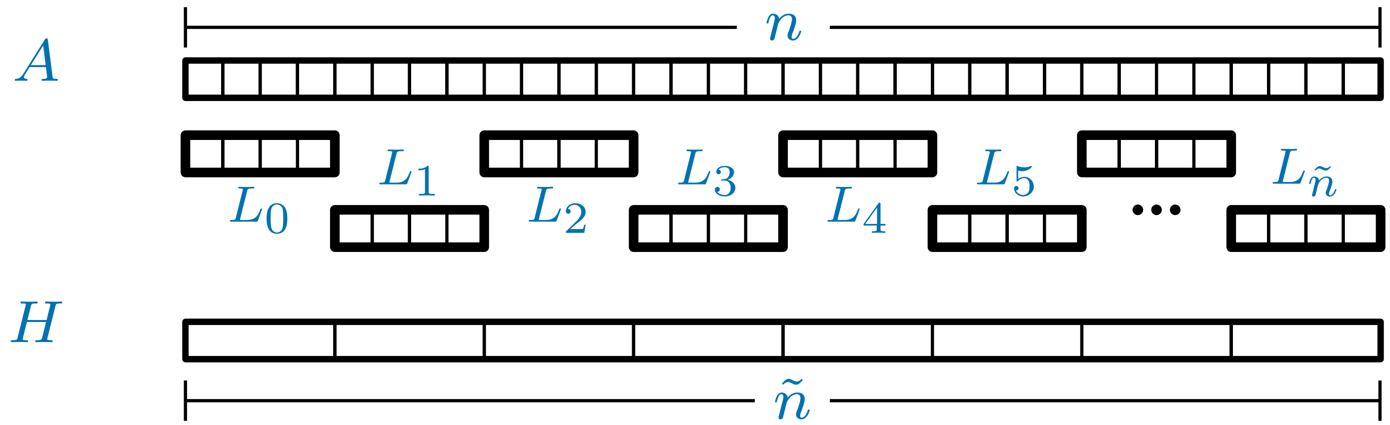


Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



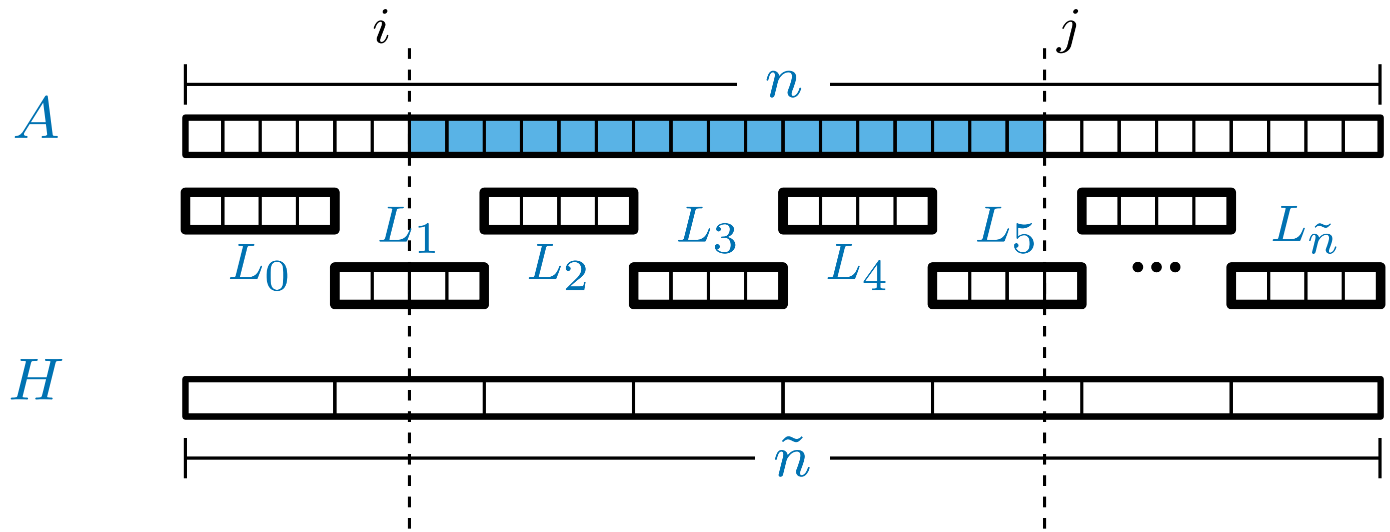
How do we answer a query in A?

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



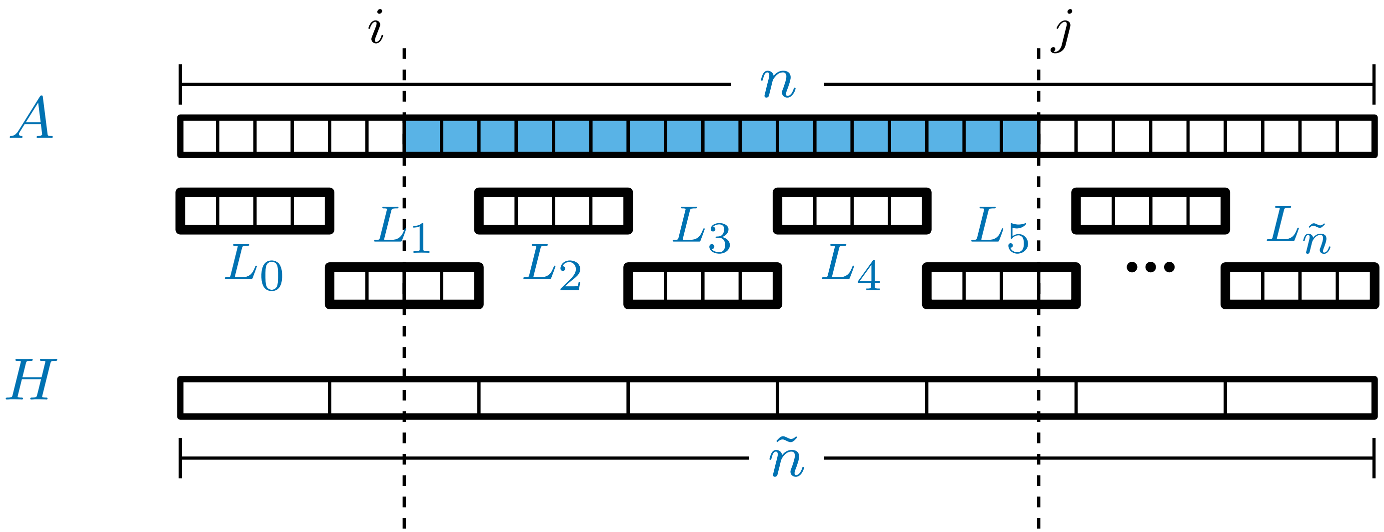
How do we answer a query in A?

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



How do we answer a query in A?

Do at most one query in H...

and one query in at most two different L_i

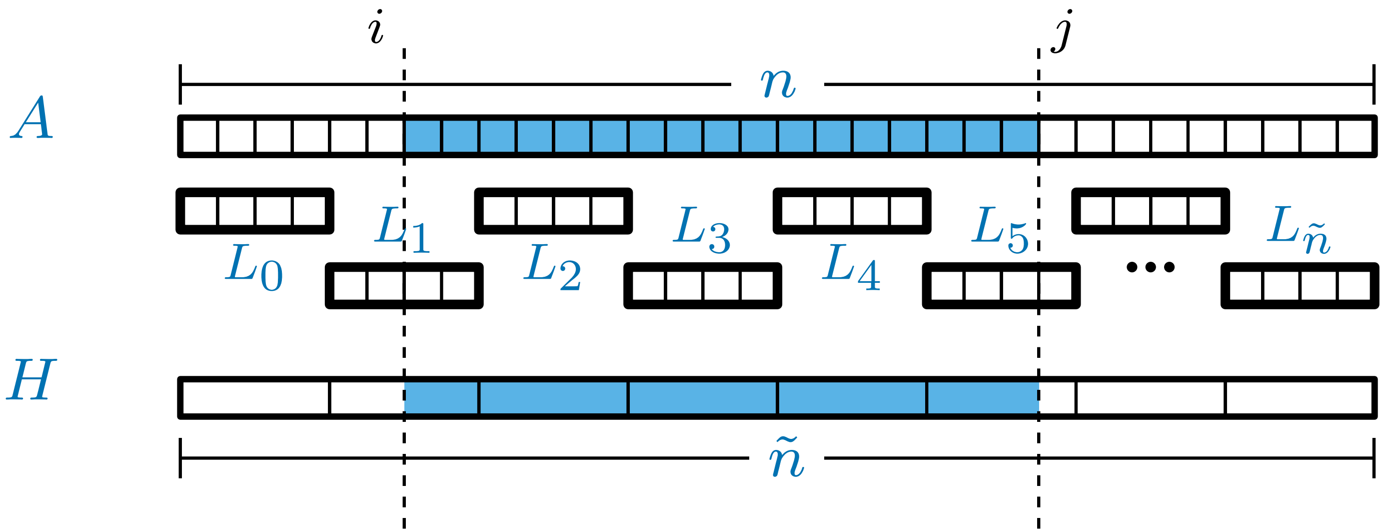
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i

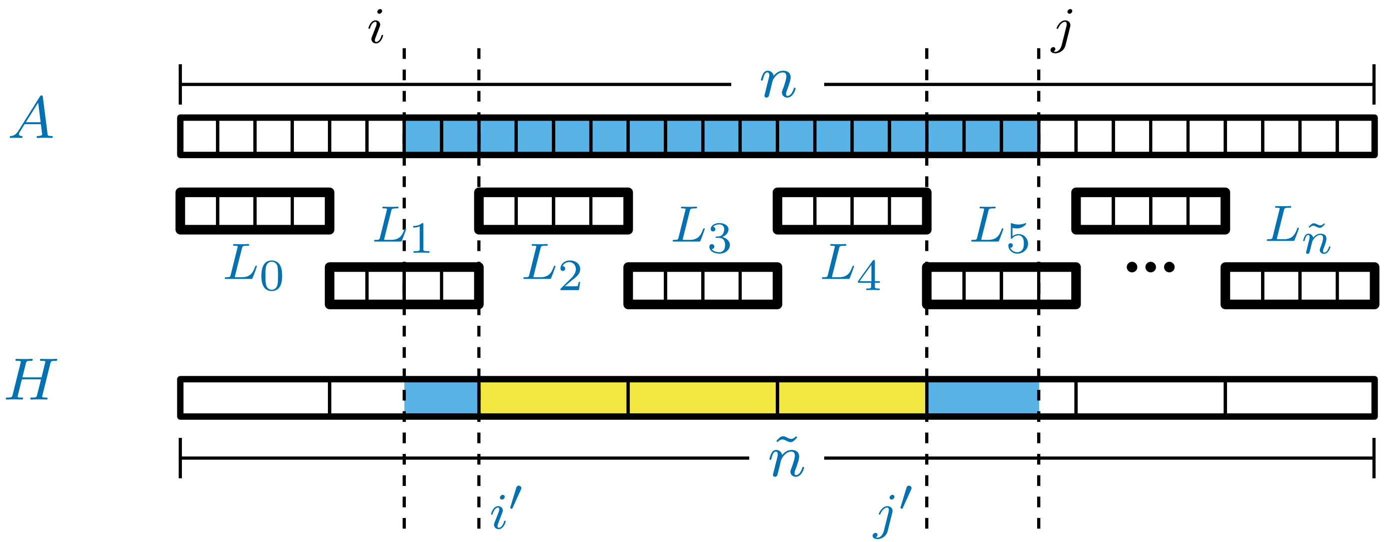
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i

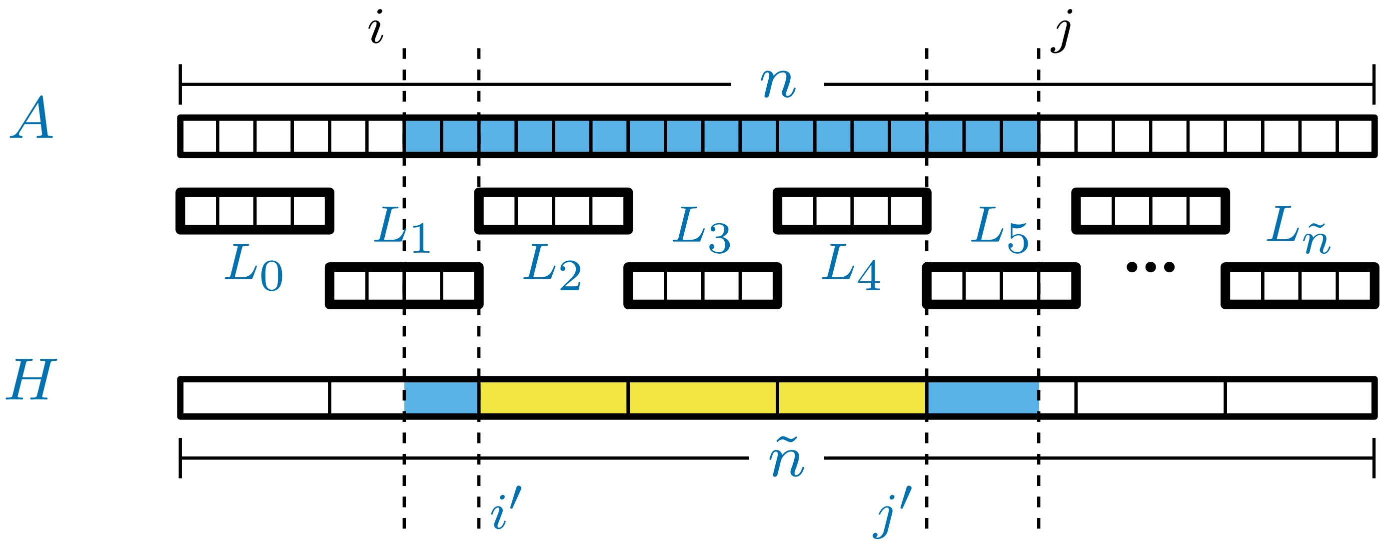
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in $H\ldots$

and one query in at most two different L_i

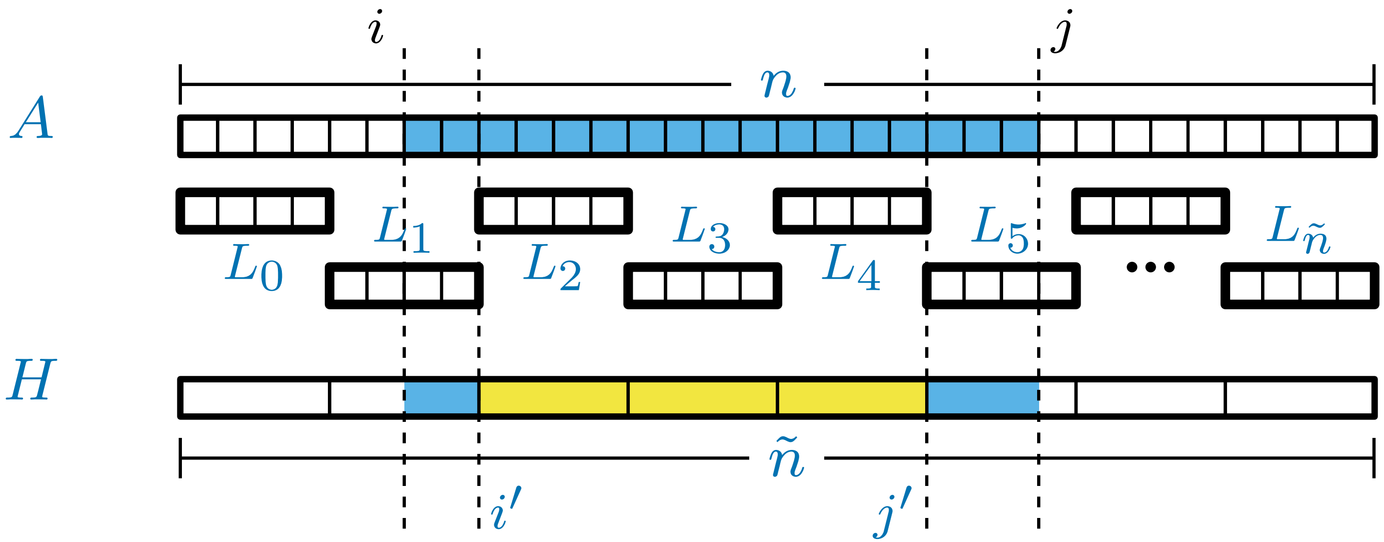
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



How do we answer a query in A?

$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

indices into H

Do at most one query in H...

and one query in at most two different L_i

then take the smallest

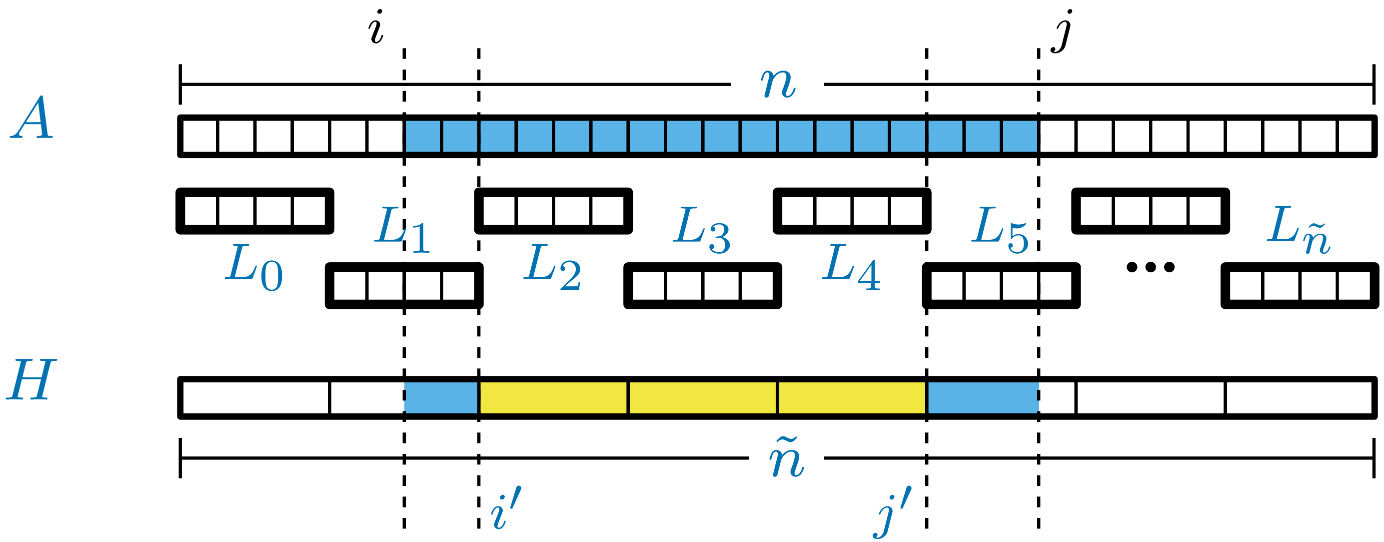
University of BRISTOL

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in H...

and one query in at most two different L_i

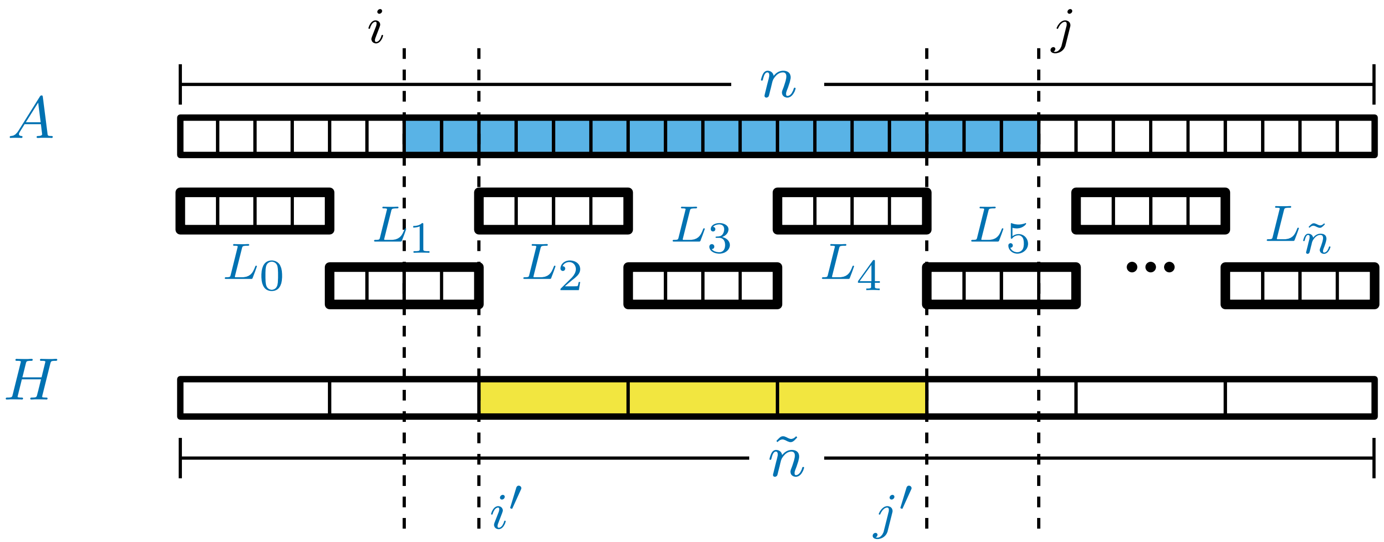
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i

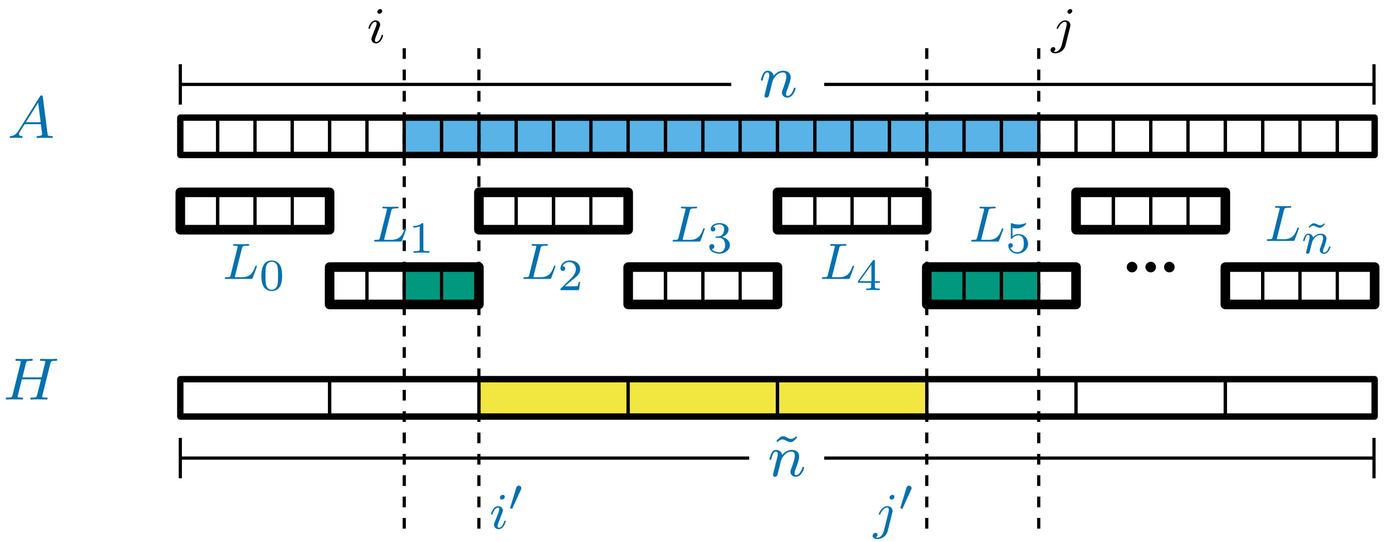
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in H...

and one query in at most two different L_i

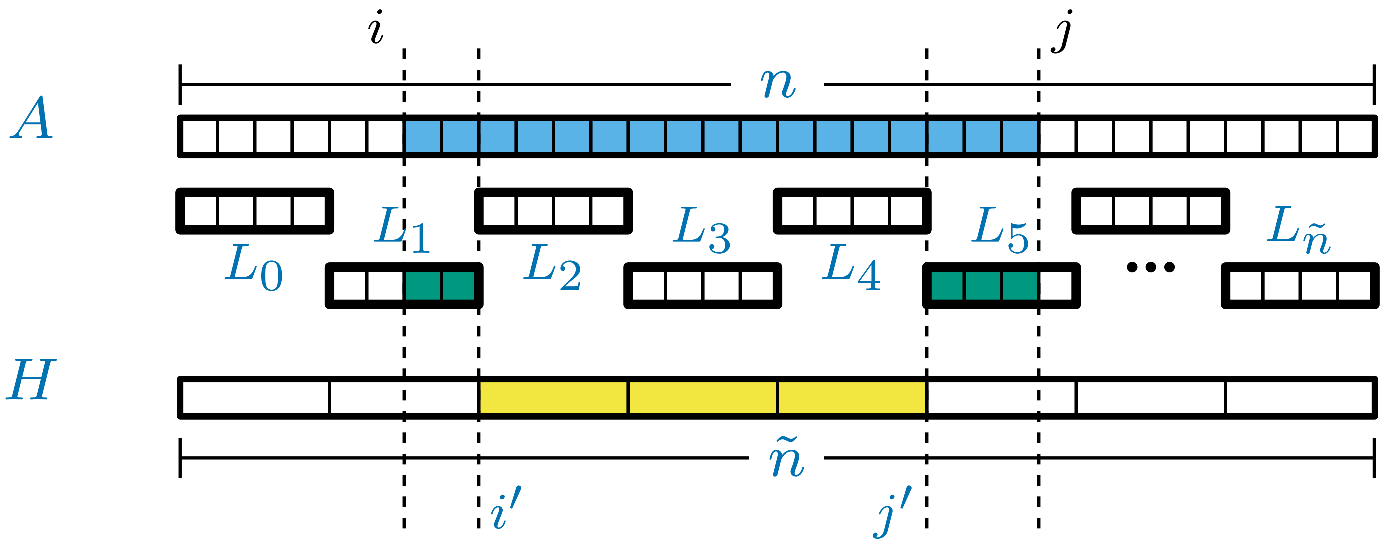
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in H...

and one query in at most two different L_i (here we query L_1 and L_5)

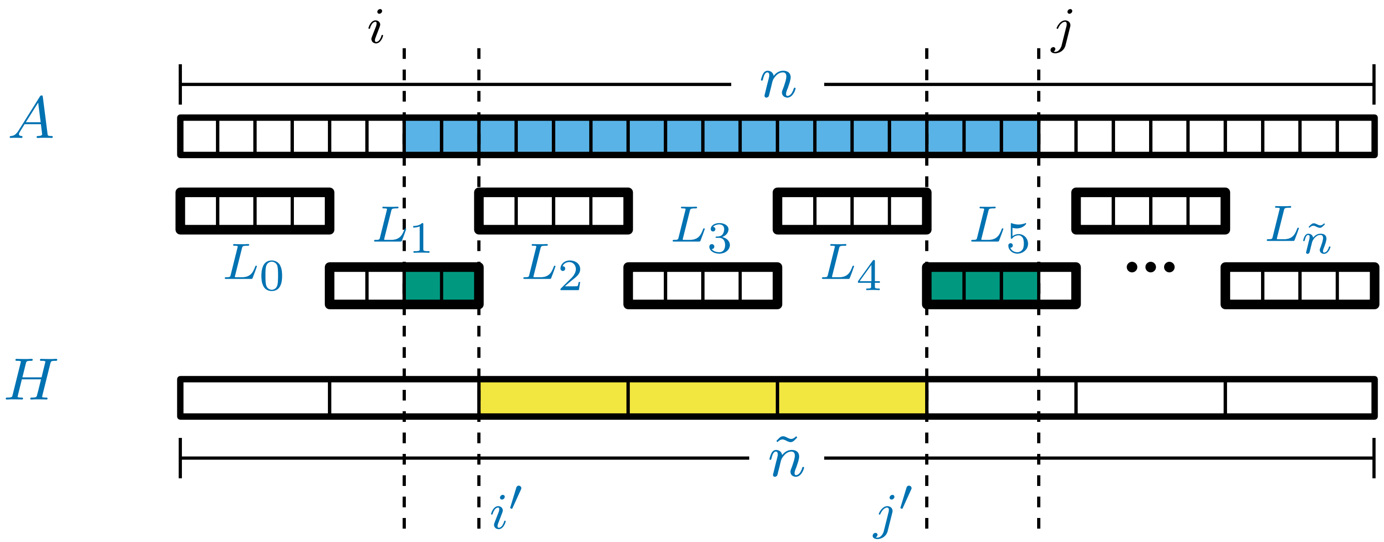
then take the smallest

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i (here we query L_1 and L_5)

then take the smallest

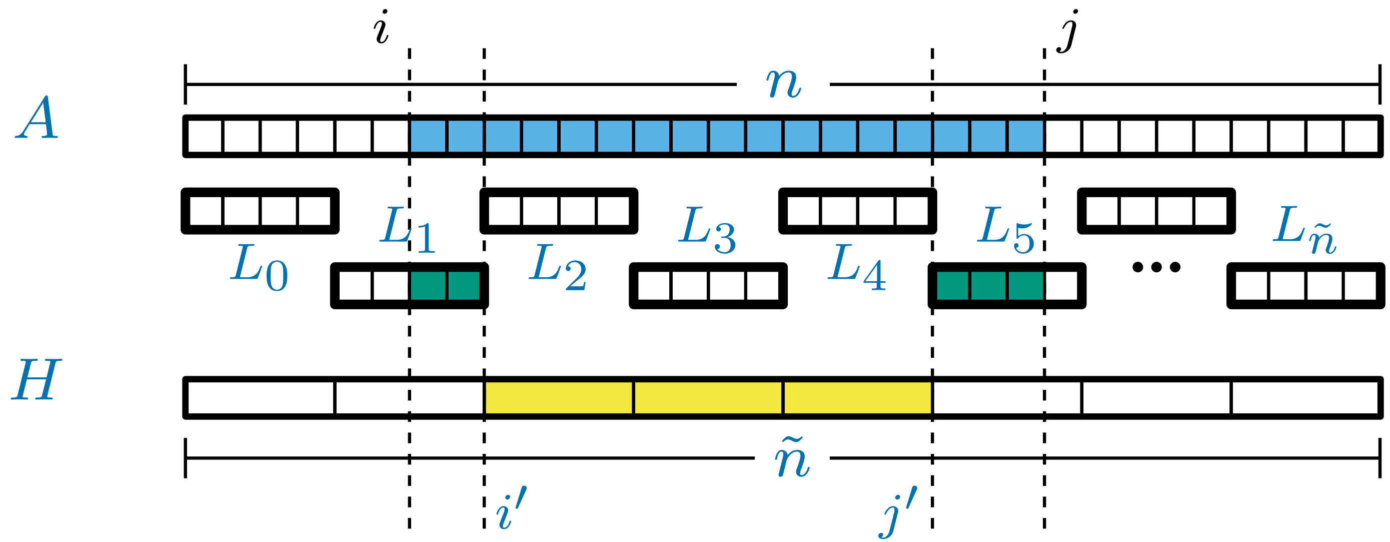
This takes $O(1)$ *total query time*

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



How do we answer a query in A?

$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

Do at most one query in H...

and one query in at most two different L_i (here we query L_1 and L_5)

then take the smallest

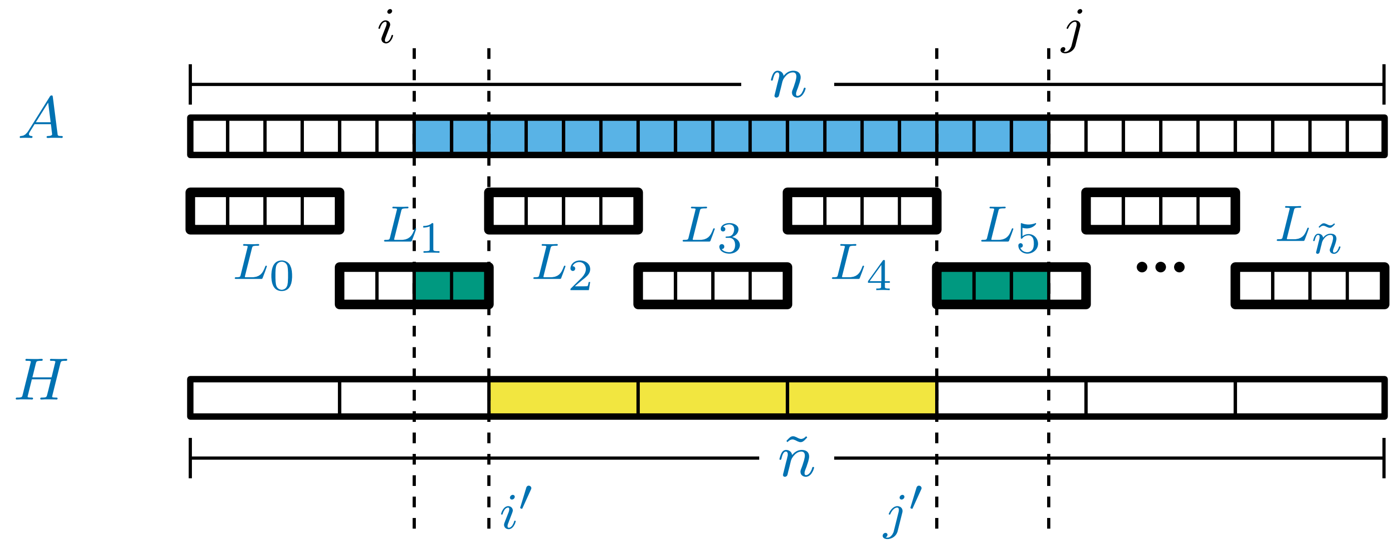
This takes $O(1)$ total query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i (here we query L_1 and L_5)

then take the smallest

This takes $O(1)$ total query time

Solution 3

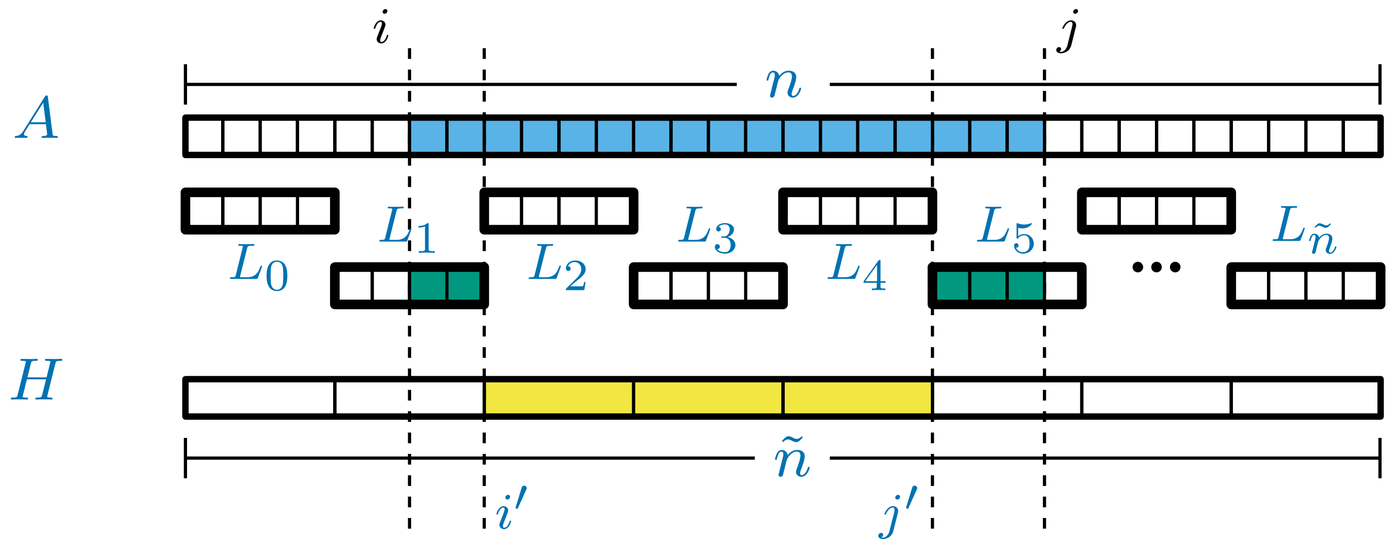
$O(n \log \log n)$ space $O(n \log \log n)$ prep time $O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i (here we query L_1 and L_5)

then take the smallest

This takes $O(1)$ total query time

Solution 4

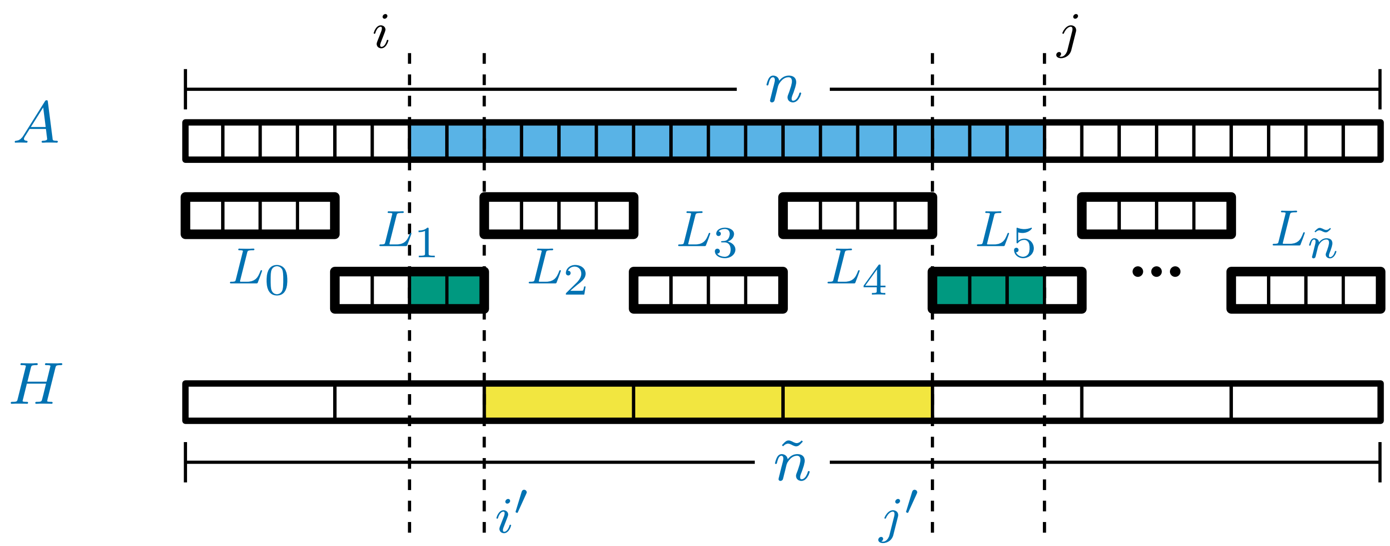
$O(n \log \log \log n)$ space $\quad O(n \log \log \log n)$ prep time $\quad O(1)$ query time

Low-resolution RMQ

$$\tilde{n} = \frac{n}{\log n}$$

Key Idea replace A with a smaller, *'low resolution'* array H

and many small arrays $L_0, L_1, L_2 \ldots$ *'for the details'*



$$i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor$$

How do we answer a query in A?

Do at most one query in $H \ldots$

and one query in at most two different L_i (here we query L_1 and L_5)

then take the smallest

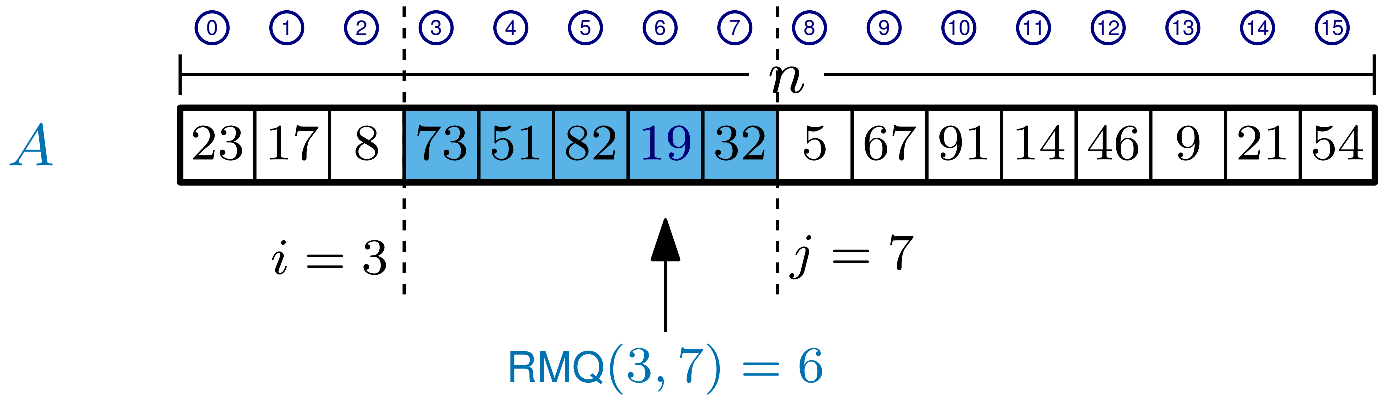
This takes $O(1)$ total query time

Solution 4 *how?*

$O(n \log \log \log n)$ space $O(n \log \log \log n)$ prep time $O(1)$ query time

Range minimum query summary

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Solution 3

$O(n \log \log n)$ space

$O(n \log \log n)$ prep time

$O(1)$ query time

Solution 2

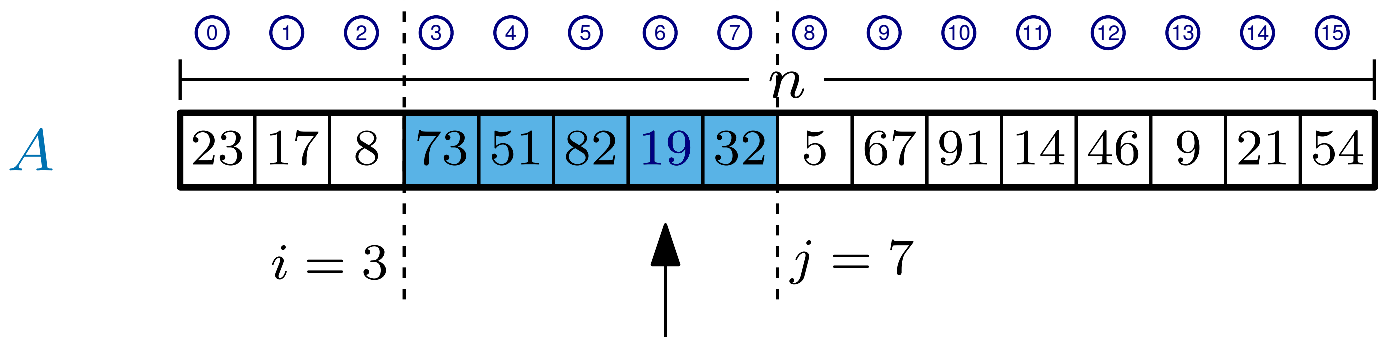
$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Range minimum query summary

Preprocess an integer array A (length n) to answer range minimum queries...



$\text{RMQ}(3, 7) = 6$

After preprocessing, a **range minimum query** is given by $\text{RMQ}(i, j)$

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Solution 3

$O(n \log \log n)$ space

$O(n \log \log n)$ prep time

$O(1)$ query time

Solution 2

$O(n \log n)$ space

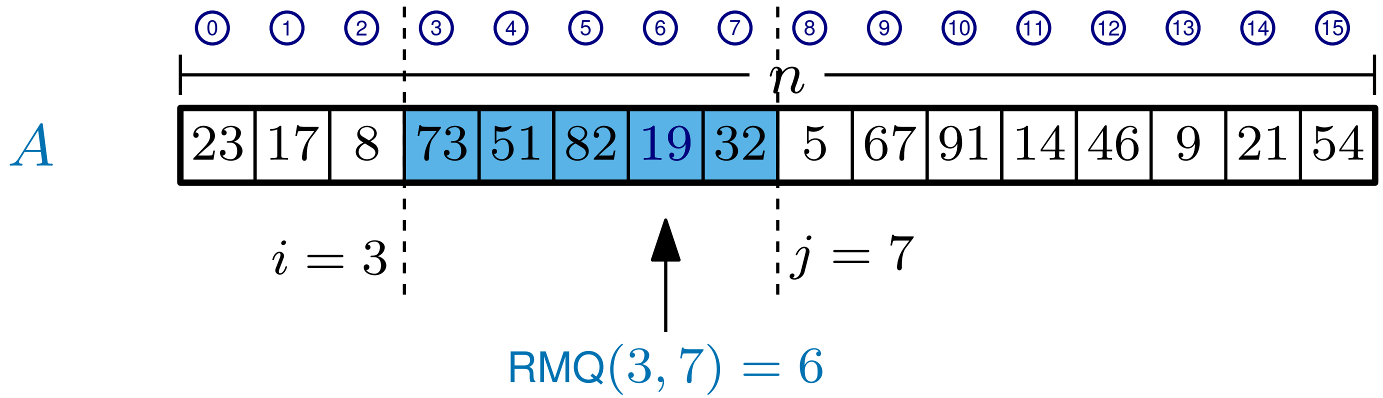
$O(n \log n)$ prep time

$O(1)$ query time

Can we do $O(n)$ space and $O(1)$ query time?

Range minimum query summary

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Solution 3

$O(n \log \log n)$ space

$O(n \log \log n)$ prep time

$O(1)$ query time

Solution 2

$O(n \log n)$ space

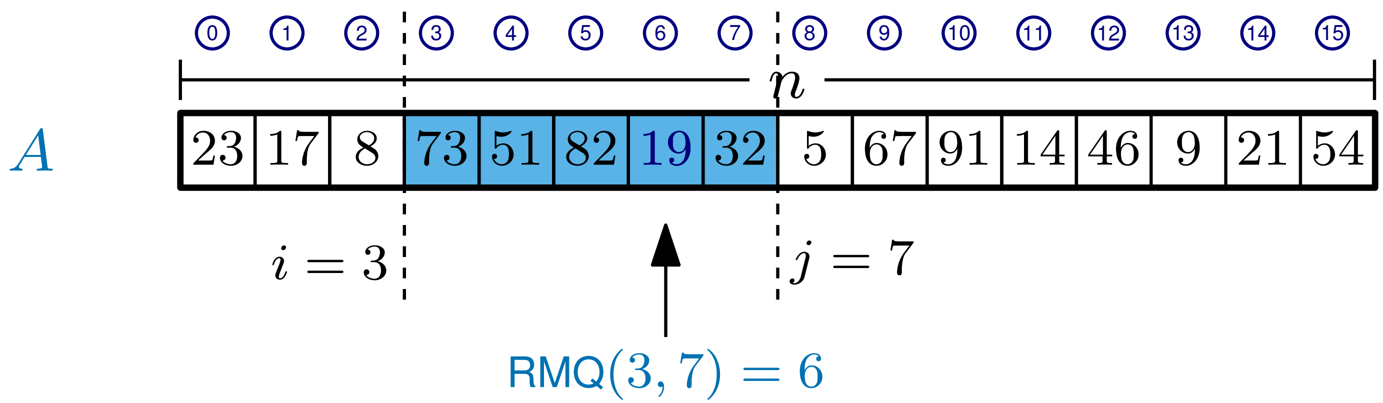
$O(n \log n)$ prep time

$O(1)$ query time

Can we do $O(n)$ space and $O(1)$ query time? *Yes...*

Range minimum query summary

Preprocess an integer array A (length n) to answer range minimum queries...



After preprocessing, a **range minimum query** is given by RMQ(i, j)

the output is the location of the smallest element in $A[i, j]$

Solution 1

$O(n)$ space

$O(n)$ prep time

$O(\log n)$ query time

Solution 3

$O(n \log \log n)$ space

$O(n \log \log n)$ prep time

$O(1)$ query time

Solution 2

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

Can we do $O(n)$ space and $O(1)$ query time? *Yes...* *but not until next lecture*