Range minimum query

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

$$A = \begin{bmatrix} 23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54 \end{bmatrix}$$
Range minimum query

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

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

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

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

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 $\text{RMQ}(i, j)$

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 $\text{RMQ}(i, j)$

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 $\text{RMQ}(i, j)$

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 $\text{RMQ}(i, j)$

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

e.g. $\text{RMQ}(3, 7) = 6$, 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 $\text{RMQ}(i, j)$

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

e.g. $\text{RMQ}(3, 7) = 6$, which is the location of the smallest element in $A[3, 7]$
e.g. $\text{RMQ}(5, 11) = 8$, 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...

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>8</td>
<td>73</td>
<td>51</td>
<td>82</td>
<td>19</td>
<td>32</td>
<td>5</td>
<td>67</td>
<td>91</td>
<td>14</td>
<td>46</td>
<td>9</td>
<td>21</td>
<td>54</td>
</tr>
</tbody>
</table>

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

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

e.g. $\text{RMQ}(3, 7) = 6$, which is the location of the smallest element in $A[3, 7]$
e.g. $\text{RMQ}(5, 11) = 8$, 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...


\[
\begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54 \\
\end{array}
\]

\(i = 5\) \hspace{1cm} \(j = 11\)

\[\text{RMQ}(5, 11) = 8\]

After preprocessing, a range minimum query is given by $\text{RMQ}(i, j)$
the output is the location of the smallest element in $A[i, j]$

\[\text{e.g. } \text{RMQ}(3, 7) = 6\text{, which is the location of the smallest element in }A[3, 7]\]
\[\text{e.g. } \text{RMQ}(5, 11) = 8\text{, which is the location of the smallest element in }A[5, 11]\]

- 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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>23</th>
<th>17</th>
<th>8</th>
<th>73</th>
<th>51</th>
<th>82</th>
<th>19</th>
<th>32</th>
<th>5</th>
<th>67</th>
<th>91</th>
<th>14</th>
<th>46</th>
<th>9</th>
<th>21</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
Block decomposition

\[
A = \begin{bmatrix}
17 & 8 & 51 & 19 & 5 & 14 & 9 & 21 & 54 \\
23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54
\end{bmatrix}
\]
Block decomposition

<table>
<thead>
<tr>
<th>A</th>
<th>23</th>
<th>17</th>
<th>8</th>
<th>73</th>
<th>51</th>
<th>82</th>
<th>19</th>
<th>32</th>
<th>5</th>
<th>67</th>
<th>91</th>
<th>14</th>
<th>46</th>
<th>9</th>
<th>21</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

smallest from each pair
Block decomposition

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>17</td>
<td>8</td>
<td>51</td>
<td>82</td>
<td>19</td>
<td>32</td>
<td>5</td>
<td>67</td>
<td>91</td>
<td>14</td>
<td>46</td>
<td>9</td>
<td>21</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

smallest from each pair
Block decomposition

The table above shows a matrix $A = (a_{ij})$, where $a_{ij}$ are the elements of the matrix. The process of block decomposition involves selecting the smallest element from each pair of elements in the matrix. For example, in the matrix $A$, the smallest elements from each pair are highlighted in blue. The process is as follows:

1. Select the smallest element from each pair of elements in the matrix.
2. Highlight these elements with a color, such as blue.
3. The highlighted elements represent the smallest from each pair.

The resulting matrix with highlighted elements is shown in the diagram.
Block decomposition

$$A = \begin{bmatrix}
17 & 8 & 51 & 19 & 5 & 14 & 9 & 21 & 54 \\
23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54
\end{bmatrix}$$
Block decomposition
Block decomposition

smallest from each four

\[ A = \begin{bmatrix}
8 & 17 & 19 & 9 \\
8 & 51 & 19 & 14 \\
73 & 82 & 5 & 46 \\
19 & 32 & 67 & 9 \\
3 & 51 & 21 & 21 \\
8 & 19 & 5 & 14 \\
73 & 82 & 67 & 9 \\
19 & 32 & 9 & 21 \\
3 & 51 & 14 & 54 \\
5 & 5 & 46 & 15
\end{bmatrix} \]
Block decomposition

smallest from each four

A

23 17 8 73 51 82 19 32 5 67 91 14 46 9 21 54

n

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Block decomposition

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>17</td>
<td>19</td>
<td>5</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>8</td>
<td>73</td>
<td>51</td>
<td>82</td>
<td>19</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>67</td>
<td>91</td>
<td>14</td>
<td>46</td>
<td>9</td>
<td>21</td>
<td>54</td>
</tr>
</tbody>
</table>

$A$
Block decomposition
Block decomposition

smallest from each eight

\[
\begin{array}{cccccccccc}
23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15
\end{array}
\]
Block decomposition

smallest from each eight

\[
\begin{array}{cccccccccccccccc}
23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54 \\
\end{array}
\]
Block decomposition

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>23</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>51</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>82</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>32</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>67</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>91</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>46</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>29</td>
</tr>
<tr>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>54</td>
<td>33</td>
</tr>
</tbody>
</table>

$n$ represents the number of columns.
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 $A$. 

<table>
<thead>
<tr>
<th>A</th>
<th>23</th>
<th>17</th>
<th>8</th>
<th>73</th>
<th>51</th>
<th>82</th>
<th>19</th>
<th>32</th>
<th>5</th>
<th>67</th>
<th>91</th>
<th>14</th>
<th>46</th>
<th>9</th>
<th>21</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
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 \(k = 1, 2, 4, 8 \ldots \leq n\)
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 \( k = 1, 2, 4, 8 \ldots \leq n \)

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 $k = 1, 2, 4, 8 \ldots \leq n$

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 $A[i_k, (i + 1)_k]$ and $x$ is its location in $A$.

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

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

\[ + \quad \frac{n}{16} \]
\[ + \quad \frac{n}{8} \]
\[ + \quad \frac{n}{4} \]
\[ + \quad \frac{n}{2} \]

\[ n \]

\[ n \]

$A_{16}$

$A_8$

$A_4$

$A_2$

$A$
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 $k = 1, 2, 4, 8 \ldots \leq n$

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

<table>
<thead>
<tr>
<th>$A_16$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5 8</td>
</tr>
</tbody>
</table>

| $A_8$  | 8 2  | 5 8  |

| $A_4$  | 8 2  | 19 6  | 5 8  | 9 13 |

| $A_2$  | 17 2  | 51 4  | 19 6  | 5 8  | 14 11  | 9 13  | 21 14 |

| $A$    | 23 17  | 8 73  | 51 82  | 19 32  | 5 67  | 91 14  | 46 9  | 21 54  |

$n$
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 $k = 1, 2, 4, 8 \ldots \leq n$

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 $k = 1, 2, 4, 8 \ldots \leq n$

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

How quickly can we build them?

\begin{verbatim}

A_{16}  
\hline
  \hline
  \hline

A_8  
\hline
  \hline
  \hline

A_4  
\hline
  \hline
  \hline

A_2  
\hline
  \hline
  \hline

A  
\hline
\end{verbatim}

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>8</td>
<td>73</td>
<td>51</td>
<td>82</td>
<td>19</td>
<td>32</td>
<td>5</td>
<td>67</td>
<td>91</td>
<td>14</td>
</tr>
</tbody>
</table>

$n$

\end{verbatim}
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 $k = 1, 2, 4, 8 \ldots \leq n$

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

How quickly can we build them?

<table>
<thead>
<tr>
<th>$A_{16}$</th>
<th></th>
<th></th>
<th>5</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{8}$</td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{4}$</td>
<td>19</td>
<td>6</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_{2}$</td>
<td>17</td>
<td>2</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>6</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>13</td>
<td>9</td>
<td>21</td>
</tr>
</tbody>
</table>

Construct the $A_k$ arrays bottom-up

<table>
<thead>
<tr>
<th>$A$</th>
<th>23</th>
<th>17</th>
<th>8</th>
<th>73</th>
<th>51</th>
<th>82</th>
<th>19</th>
<th>32</th>
<th>5</th>
<th>67</th>
<th>91</th>
<th>14</th>
<th>46</th>
<th>9</th>
<th>21</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

$n$
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 $k = 1, 2, 4, 8 \ldots \leq n$

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

How quickly can we build them?

construct the $A_k$
arrays bottom-up

compute this from
these in $O(1)$ 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 $k = 1, 2, 4, 8 \ldots \leq n$

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

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

$A_{16}$

$A_8$

$A_4$

$A_2$

$A$

construct the $A_k$
arrays bottom-up

compute this from
these in $O(1)$ time

\begin{array}{cccccc}
23 & 17 & 8 & 73 & 51 & 82 \\
19 & 32 & 5 & 67 & 91 & 14 \\
14 & 46 & 9 & 21 & 54 \\
\end{array}
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 $k = 1, 2, 4, 8 \ldots \leq n$

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

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

$A_16$

\[
\begin{array}{cccc}
\hline
5 \\
\hline
\end{array}
\]

$A_8$

\[
\begin{array}{cccc}
8 & 5 \\
\hline
2 & 8 \\
\hline
\end{array}
\]

$A_4$

\[
\begin{array}{cccc}
8 & 19 & 5 \\
\hline
2 & 6 & 8 \\
\hline
\end{array}
\]

$A_2$

\[
\begin{array}{cccc}
17 & 8 & 51 & 19 & 14 & 9 & 21 \\
\hline
1 & 2 & 4 & 6 & 11 & 13 & 14 \\
\hline
\end{array}
\]

$A$

\[
\begin{array}{cccccccccccc}
23 & 17 & 8 & 73 & 51 & 82 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54 \\
\hline
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
\end{array}
\]
Block decomposition

\[
\begin{array}{cccc}
A_{16} & & & 5 \\
A_8 & 8 & 5 & \\
A_4 & 8 & 19 & 5 & 9 \\
A_2 & 17 & 8 & 51 & 19 & 5 & 14 & 9 & 21 \\
A & 23 & 17 & 8 & 73 & 19 & 32 & 5 & 67 & 91 & 14 & 46 & 9 & 21 & 54 \\
\end{array}
\]
How do we find $\text{RMQ}(i,j)$?
How do we find $\text{RMQ}(i,j)$?
How do we find $\text{RMQ}(i,j)$?

Find the largest block which is completely contained within the query interval.

Block decomposition

<table>
<thead>
<tr>
<th>A16</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A8</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A4</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>23</th>
<th>17</th>
<th>8</th>
<th>73</th>
<th>51</th>
<th>82</th>
<th>19</th>
<th>32</th>
<th>5</th>
<th>67</th>
<th>91</th>
<th>14</th>
<th>46</th>
<th>9</th>
<th>21</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

$n$
How do we find $\text{RMQ}(i,j)$?

Find the largest block which is completely contained within the query interval $\text{RMQ}(1,9)$.
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*
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*
How do we find $\text{RMQ}(i,j)$?

**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)*
How do we find $\text{RMQ}(i,j)$?

**Repeat:** Find the largest block which is completely contained within the query interval but doesn’t overlap a block you chose before.
How do we find $\text{RMQ}(i,j)$?

**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 $\text{RMQ}(i, j)$?

**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 $\text{RMQ}(i,j)$?

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

<table>
<thead>
<tr>
<th>$A_{16}$</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{8}$</td>
<td>$8$</td>
<td>$5$</td>
<td>$5$</td>
<td>$9$</td>
</tr>
<tr>
<td>$A_{4}$</td>
<td>$8$</td>
<td>$19$</td>
<td>$5$</td>
<td>$9$</td>
</tr>
<tr>
<td>$A_{2}$</td>
<td>$17$</td>
<td>$51$</td>
<td>$19$</td>
<td>$14$</td>
</tr>
<tr>
<td>$A$</td>
<td>$23$</td>
<td>$17$</td>
<td>$8$</td>
<td>$32$</td>
</tr>
</tbody>
</table>

How many blocks do we pick?
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.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{16}$</td>
<td>$A_{8}$</td>
<td>$A_{4}$</td>
<td>$A_{2}$</td>
<td>$A$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>19</td>
<td>9</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>6</td>
<td>21</td>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>19</td>
<td>46</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>67</td>
<td>14</td>
<td>13</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>11</td>
<td>9</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>14</td>
<td>13</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>11</td>
<td>9</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>51</td>
<td>82</td>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many blocks do we pick? 19
How do we find $\text{RMQ}(i,j)$?

**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 $\text{RMQ}(i,j)$?

**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*

---

<table>
<thead>
<tr>
<th></th>
<th>$A_{16}$</th>
<th></th>
<th>$A_{14}$</th>
<th>$A_{8}$</th>
<th>$A_{4}$</th>
<th>$A_{2}$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>13</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>21</td>
<td>21</td>
<td>13</td>
<td>14</td>
<td>46</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>21</td>
<td>54</td>
<td>14</td>
<td>14</td>
<td>21</td>
<td>15</td>
</tr>
</tbody>
</table>

How many blocks do we pick?
**Block decomposition**

How do we find $\text{RMQ}(i,j)$?

**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*

<table>
<thead>
<tr>
<th></th>
<th>$A_{16}$</th>
<th></th>
<th></th>
<th>$5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{8}$</td>
<td>8</td>
<td>2</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>$A_{4}$</td>
<td>8</td>
<td>2</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>$A_{2}$</td>
<td>17</td>
<td>8</td>
<td>51</td>
<td>19</td>
</tr>
<tr>
<td>$A$</td>
<td>23</td>
<td>17</td>
<td>8</td>
<td>73</td>
</tr>
</tbody>
</table>

How many blocks do we pick?
How do we find \( \text{RMQ}(i,j) \)?

**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?

never three in a row
How do we find \( \text{RMQ}(i,j) \)?

**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.

<table>
<thead>
<tr>
<th>Block</th>
<th>( A_{16} )</th>
<th>( A_{8} )</th>
<th>( A_{4} )</th>
<th>( A_{2} )</th>
<th>( A )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 ( \overset{2}{\circ} )</td>
<td>19 ( \overset{6}{\circ} )</td>
<td>51 ( \overset{4}{\circ} )</td>
<td>17 ( \overset{1}{\circ} )</td>
<td>23 ( \overset{0}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>8 ( \overset{2}{\circ} )</td>
<td>19 ( \overset{6}{\circ} )</td>
<td>51 ( \overset{4}{\circ} )</td>
<td>51 ( \overset{4}{\circ} )</td>
<td>17 ( \overset{1}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>8 ( \overset{2}{\circ} )</td>
<td>51 ( \overset{4}{\circ} )</td>
<td>19 ( \overset{6}{\circ} )</td>
<td>19 ( \overset{6}{\circ} )</td>
<td>8 ( \overset{2}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>8 ( \overset{2}{\circ} )</td>
<td>19 ( \overset{6}{\circ} )</td>
<td>51 ( \overset{4}{\circ} )</td>
<td>19 ( \overset{6}{\circ} )</td>
<td>17 ( \overset{1}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>5 ( \overset{8}{\circ} )</td>
<td>5 ( \overset{8}{\circ} )</td>
<td>5 ( \overset{8}{\circ} )</td>
<td>14 ( \overset{13}{\circ} )</td>
<td>14 ( \overset{13}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>9 ( \overset{13}{\circ} )</td>
<td>9 ( \overset{13}{\circ} )</td>
<td>14 ( \overset{13}{\circ} )</td>
<td>9 ( \overset{13}{\circ} )</td>
<td>9 ( \overset{13}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>21 ( \overset{14}{\circ} )</td>
<td>21 ( \overset{14}{\circ} )</td>
<td>21 ( \overset{14}{\circ} )</td>
<td>21 ( \overset{14}{\circ} )</td>
<td>54 ( \overset{15}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>11 ( \overset{12}{\circ} )</td>
<td>11 ( \overset{12}{\circ} )</td>
<td>11 ( \overset{12}{\circ} )</td>
<td>11 ( \overset{12}{\circ} )</td>
<td>11 ( \overset{12}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>13 ( \overset{14}{\circ} )</td>
<td>13 ( \overset{14}{\circ} )</td>
<td>13 ( \overset{14}{\circ} )</td>
<td>13 ( \overset{14}{\circ} )</td>
<td>13 ( \overset{14}{\circ} )</td>
</tr>
<tr>
<td></td>
<td>15 ( \overset{15}{\circ} )</td>
<td>15 ( \overset{15}{\circ} )</td>
<td>15 ( \overset{15}{\circ} )</td>
<td>15 ( \overset{15}{\circ} )</td>
<td>15 ( \overset{15}{\circ} )</td>
</tr>
</tbody>
</table>

How many blocks do we pick?

Never three in a row.

\( n \)
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.

```
23 17 8 73 51 82 19 32 5 67 91 14 46 9 21 54
```

How many blocks do we pick? at most 2 blocks of each size.
Block decomposition

How do we find \(\text{RMQ}(i,j)\)?

- **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*

|   | \(A_{16}\) |   | \(A_{8}\) |   | \(A_{4}\) |   | \(A_{2}\) |   | \(A\) |
|---|---|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |   |   |
| \(A_{16}\) |   |   |   |   |   |   |   |   |
| \(A_{8}\) |   |   |   |   |   |   |   |   |
| \(A_{4}\) |   |   |   |   |   |   |   |   |
| \(A_{2}\) |   |   |   |   |   |   |   |   |
| \(A\) |   |   |   |   |   |   |   |   |
| \(n\) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

How many blocks do we pick?

at most 2 blocks of each size
How do we find \( \text{RMQ}(i,j) \)?

**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

never two on one side
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*

---

**Diagram:**

- **$A_{16}$**
- **$A_{8}$**
- **$A_{4}$**
- **$A_{2}$**
- **$A$**

How many blocks do we pick?

- at most 2 blocks of each size

never two on one side
Block decomposition

How do we find $\text{RMQ}(i,j)$?

**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*

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_4$</th>
<th>$A_8$</th>
<th>$A_{16}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>17</td>
<td>8</td>
<td>73</td>
<td>51</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>51</td>
<td>82</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>19</td>
<td>5</td>
<td>67</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>51</td>
<td>82</td>
<td>32</td>
</tr>
</tbody>
</table>

How many blocks do we pick? at most 2 blocks of each size
How do we find $\text{RMQ}(i, j)$?

**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.

---

**Diagram:**

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

**Note:** How many blocks do we pick? at most 2 blocks of each size.
How do we find \( \text{RMQ}(i,j) \)?

**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*

---

<table>
<thead>
<tr>
<th>Block</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_{16} )</td>
<td>5</td>
</tr>
<tr>
<td>( A_8 )</td>
<td>8, 5</td>
</tr>
<tr>
<td>( A_4 )</td>
<td>8, 19, 5</td>
</tr>
<tr>
<td>( A_2 )</td>
<td>17, 8, 51, 19, 5</td>
</tr>
<tr>
<td>( A )</td>
<td>23, 17, 8, 73, 51, 82, 19, 32, 5, 67, 91, 14, 46, 9, 21, 54</td>
</tr>
</tbody>
</table>

How many blocks do we pick?

at most 2 blocks of each size

no gaps
Block decomposition

How do we find \( \text{RMQ}(i,j) \)?

**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

no gaps
How do we find $\text{RMQ}(i,j)$?

**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*

**10,000 foot view**

- $A_{16}$
- $A_8$
- $A_4$
- $A_2$
- $A$

**How many blocks do we pick?**

*at most 2 blocks of each size*
How do we find $\text{RMQ}(i,j)$?

**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 \( \text{RMQ}(i,j) \)?

**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
How do we find \( \text{RMQ}(i,j) \)?

**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*

---

<table>
<thead>
<tr>
<th>( A_{16} )</th>
<th>( A_8 )</th>
<th>( A_4 )</th>
<th>( A_2 )</th>
<th>( A )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 23 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 17 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 8 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 73 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 51 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 82 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 19 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 32 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 5 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 67 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( 91 )</td>
</tr>
</tbody>
</table>

**RMQ(1,9)**

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 $2, 4, 8, 16 \ldots$

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, ...
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 \(2, 4, 8, 16 \ldots\)

![Diagram of array 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 $2, 4, 8, 16 \ldots$.

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 $\text{RMQ}(i, i + 1)$ 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 $i$
**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 $i$. 

![Diagram of array A with interval of length 2]
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 $i$. 

![Diagram](image-url)
More space, faster queries

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

![Diagram of array A with intervals and RMQ stores in R2]

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 $i$

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

**Key Idea** precompute the answers for every interval of length 2, 4, 8, 16, ...  

The array $R_2$ stores $\text{RMQ}(i, i+1)$ for all $i$  
$R_4$ stores $\text{RMQ}(i, i+3)$ 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 $\text{RMQ}(i, i + 1)$ for all $i$
- $R_4$ stores $\text{RMQ}(i, i + 3)$ for all $i$
- $R_8$ stores $\text{RMQ}(i, i + 7)$ for all $i$
**Key Idea** precompute the answers for every interval of length 2, 4, 8, 16, ...
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 $i$
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 $k = 2, 4, 8, 16 \ldots \leq n$
More space, faster queries

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

The array $R_2$ stores $\text{RMQ}(i, i + 1)$ 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 $k = 2, 4, 8, 16 \ldots \leq n$

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 . . .

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

- $R_2$ stores $RMQ(i, i + 3)$ for all $i$
- $R_4$ stores $RMQ(i, i + 7)$ for all $i$
- $R_k$ stores $RMQ(i, i + k - 1)$ for all $i$

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

- 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 $R_2$ from $A$ in $O(n)$ time

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

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

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

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

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 $k = 2, 4, 8, 16 \ldots \leq n$

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 . . .

The array $R_k$ stores $\text{RMQ}(i, i + 1)$ for all $i$
- $R_2$ stores $\text{RMQ}(i, i + 3)$ for all $i$
- $R_4$ 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 $k = 2, 4, 8, 16 \ldots \leq n$

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

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

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

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

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

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

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

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 $k = 2, 4, 8, 16 \ldots \leq n$

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 $2, 4, 8, 16 \ldots$

$A$

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

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

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
```

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
</table>

stored in $R_2$

stored in $R_4$

stored in $R_8$

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 $k = 2, 4, 8, 16 \ldots \leq n$

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

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

how?
**More space, faster queries**

**Key Idea** precompute the answers for every interval of length 2, 4, 8, 16 . . .

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 $k = 2, 4, 8, 16 \ldots \leq n$

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

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

How do we compute $\text{RMQ}(i, j)$?

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

we build \(R_k\) for \(k = 2, 4, 8, 16 \ldots \leq n\)

\[
\begin{array}{c}
A \\
\hline
1 \quad 2 \\
3 \quad 4 \\
5 \quad 8 \\
\end{array}
\]

stored in \(R_2\)

stored in \(R_4\)

stored in \(R_8\)

How do we compute \(\text{RMQ}(i, j)\)?

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

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

How do we compute $\text{RMQ}(i, j)$?

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 $\text{RMQ}(i, i + k - 1)$ for all $i$,
we build $R_k$ for $k = 2, 4, 8, 16 \ldots \leq n$

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$
More space, faster queries

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

we build \(R_k\) for \(k = 2, 4, 8, 16 \ldots \leq n\)

How do we compute \(\text{RMQ}(i, j)\)?

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 \(k \leq \ell < 2k\)
More space, faster queries

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

we build \( R_k \) for \( k = 2, 4, 8, 16 \ldots \leq n \)

How do we compute \( \text{RMQ}(i, j) \)?

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 \( k \leq \ell < 2k \)

Compute the minimum of \( \text{RMQ}(i, i + k - 1) \) and \( \text{RMQ}(j - k + 1, j) \)
More space, faster queries

$R_k$ stores $\text{RMQ}(i, i + k - 1)$ for all $i$,
we build $R_k$ for $k = 2, 4, 8, 16 \ldots \leq n$

\[
\begin{array}{c}
A \\
\text{stored in } R_2 \\
\text{stored in } R_4 \\
\text{stored in } R_8
\end{array}
\]

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

Compute the minimum of $\text{RMQ}(i, i + k - 1)$ and $\text{RMQ}(j - k + 1, j)$
(these two queries take $O(1)$ time)
More space, faster queries

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

we build \( R_k \) for \( k = 2, 4, 8, 16 \ldots \leq n \)

How do we compute \( \text{RMQ}(i, j) \)?

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 \( k \leq \ell < 2k \)

Compute the minimum of \( \text{RMQ}(i, i + k - 1) \) and \( \text{RMQ}(j - k + 1, j) \)

(these two queries take \( O(1) \) time)
More space, faster queries

$R_k$ stores $\text{RMQ}(i, i + k - 1)$ for all $i$,
we build $R_k$ for $k = 2, 4, 8, 16 \ldots \leq n$

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

Compute the minimum of $\text{RMQ}(i, i + k - 1)$ and $\text{RMQ}(j - k + 1, j)$
(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?
More space, faster queries

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

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

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

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

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

This takes $O(1)$ time but why does it work?
More space, faster queries

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

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

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

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

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

This takes $O(1)$ time but why does it work?
More space, faster queries

$R_k$ stores $\text{RMQ}(i, i + k - 1)$ for all $i$,
we build $R_k$ for $k = 2, 4, 8, 16 \ldots \leq n$

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

Compute the minimum of $\text{RMQ}(i, i + k - 1)$ and $\text{RMQ}(j - k + 1, j)$
(these two queries take $O(1)$ time)

This takes $O(1)$ time but why does it work?
More space, faster queries

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

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

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

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

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

This takes $O(1)$ time but why does it work?
More space, faster queries

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

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

How do we compute $\text{RMQ}(i, j)$?

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 $k \leq \ell < 2k$

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

(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 $\text{RMQ}(i, j)$
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 $\text{RMQ}(i, j)$

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 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?

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 $\text{RMQ}(i, j)$

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

Can we do better?

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 $H$
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 $H$

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

**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 \( H \)

and many small arrays \( L_0, L_1, L_2 \ldots \) ‘for the details’

\[
\tilde{n} = \frac{n}{\log n}
\]
Low-resolution RMQ

**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’

\[ \tilde{n} = \frac{n}{\log n} \]
Low-resolution RMQ

**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'*

\[ \tilde{n} = \frac{n}{\log n} \]
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’

\[ \tilde{n} = \frac{n}{\log n} \]
Low-resolution RMQ

**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’

\[ \tilde{n} = \frac{n}{\log n} \]

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

**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’

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

Preprocess the array $H$ (which has length $\tilde{n} = \frac{n}{\log n}$) to answer RMQs... using **Solution 2**
Low-resolution RMQ

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

**Recall...**

**Solution 2 on \( A \)**

- \( O(n \log n) \) space
- \( O(n \log n) \) prep time
- \( O(1) \) query time
Low-resolution RMQ

**Key Idea** replace \( A \) with a smaller, ‘low resolution’ array \( H \)
and many small arrays \( L_0, L_1, L_2 \ldots \) ‘for the details’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

**Recall**...

**Solution 2 on** \( A \)

- \( O(n \log n) \) space
- \( O(n \log n) \) prep time
- \( O(1) \) query time
Low-resolution RMQ

**Key Idea** replace $A$ with a smaller, ‘low resolution’ array $H$
and many small arrays $L_0, L_1, L_2$ . . . ‘for the details’

![Diagram of A and H arrays]

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

*Recall...*

<table>
<thead>
<tr>
<th>Solution 2 on $A$</th>
<th>Solution 2 on $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log n)$ space</td>
<td>$O(\tilde{n} \log \tilde{n})$ space</td>
</tr>
<tr>
<td>$O(n \log n)$ prep time</td>
<td>$O(\tilde{n} \log \tilde{n})$ prep time</td>
</tr>
<tr>
<td>$O(1)$ query time</td>
<td>$O(1)$ query time</td>
</tr>
</tbody>
</table>
Low-resolution RMQ

**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…

*Recall…*

<table>
<thead>
<tr>
<th>Solution 2 on $A$</th>
<th>Solution 2 on $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log n)$ space</td>
<td>$O(\tilde{n} \log \tilde{n})$ space = $O \left( \left( \frac{n}{\log n} \right) \log \left( \frac{n}{\log n} \right) \right)$</td>
</tr>
<tr>
<td>$O(n \log n)$ prep time</td>
<td>$O(\tilde{n} \log \tilde{n})$ prep time</td>
</tr>
<tr>
<td>$O(1)$ query time</td>
<td>$O(1)$ query time</td>
</tr>
</tbody>
</table>
Low-resolution RMQ

**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...

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

**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...

Recall...

<table>
<thead>
<tr>
<th>Solution 2 on $A$</th>
<th>Solution 2 on $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log n)$ space</td>
<td>$O(\tilde{n} \log \tilde{n})$ space $= O\left(\left(\frac{n}{\log n}\right) \log \left(\frac{n}{\log n}\right)\right) = O(n)$</td>
</tr>
<tr>
<td>$O(n \log n)$ prep time</td>
<td>$O(\tilde{n} \log \tilde{n})$ prep time $= O(n)$</td>
</tr>
<tr>
<td>$O(1)$ query time</td>
<td>$O(1)$ query time</td>
</tr>
</tbody>
</table>
Low-resolution RMQ

**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...

Recall...

<table>
<thead>
<tr>
<th>Solution 2 on $A$</th>
<th>Solution 2 on $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log n)$ space</td>
<td>$O(\tilde{n} \log \tilde{n})$ space $= O\left(\left(\frac{n}{\log n}\right) \log \left(\frac{n}{\log n}\right)\right) = O(n)$</td>
</tr>
<tr>
<td>$O(n \log n)$ prep time</td>
<td>$O(\tilde{n} \log \tilde{n})$ prep time $= O(n)$</td>
</tr>
<tr>
<td>$O(1)$ query time</td>
<td>$O(1)$ query time</td>
</tr>
</tbody>
</table>
Low-resolution RMQ

**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

**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*

\[ \tilde{n} = \frac{n}{\log n} \]

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**

**Key Idea** replace $A$ with a smaller, ‘low resolution’ array $H$

and many small arrays $L_0, L_1, L_2$ . . . ‘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** $L_i$

$O\left((\log n \log \log n)\right)$ space/prep time $O(1)$ query time
Low-resolution RMQ

**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’

\[ \tilde{n} = \frac{n}{\log n} \]

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

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

$$\text{Total space} = O(n) + O(\tilde{n} \log n \log \log n)$$
Low-resolution RMQ

**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’

![Diagram showing arrays $A$ and $H$]

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

**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

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’

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

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**

**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

**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’

$$\tilde{n} = \frac{n}{\log n}$$
Low-resolution RMQ

**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 $A$?

\[ \tilde{n} = \frac{n}{\log n} \]
Low-resolution RMQ

**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’

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

How do we answer a query in $A$?
Low-resolution RMQ

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

**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’

\[ \tilde{n} = \frac{n}{\log n} \]

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

**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’

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

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**

**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

$$
\tilde{n} = \frac{n}{\log n}
$$
Low-resolution RMQ

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

**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’*

\[
\tilde{n} = \frac{n}{\log n}
\]

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

\[
i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor
\]
Low-resolution RMQ

**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'*

\[
\tilde{n} = \frac{n}{\log n}
\]

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

\[
i' = \left\lceil \frac{i}{\log n} \right\rceil \quad j' = \left\lfloor \frac{j}{\log n} \right\rfloor
\]
**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’

\[ \tilde{n} = \frac{n}{\log n} \]

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

**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’

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

**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

$\lfloor \frac{i}{\log n} \rfloor$ $\leq i' \leq \lceil \frac{i}{\log n} \rceil$ $\lfloor \frac{j}{\log n} \rfloor$ $\leq j' \leq \lceil \frac{j}{\log n} \rceil$

*This takes $O(1)$ total query time*
Low-resolution RMQ

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

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

*This takes \( O(1) \) total query time*
Key Idea replace $A$ with a smaller, ‘low resolution’ array $H$

and many small arrays $L_0$, $L_1$, $L_2$ . . . ‘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 $L_i$ (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 \hspace{1cm} $O(n \log \log n)$ prep time \hspace{1cm} $O(1)$ query time
Low-resolution RMQ

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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

This takes \( O(1) \) total query time

Solution 4

\( O(n \log \log \log n) \) space \( O(n \log \log \log n) \) prep time \( O(1) \) query time
Low-resolution RMQ

**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’

\[
\tilde{n} = \frac{n}{\log n}
\]

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

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

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
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 2**

$O(n \log n)$ space

$O(n \log n)$ prep time

$O(1)$ query time

**Solution 3**

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

$O(n \log \log n)$ 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 $\text{RMQ}(i, j)$
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 2
- $O(n \log n)$ space
- $O(n \log n)$ prep time
- $O(1)$ query time

Solution 3
- $O(n \log \log n)$ space
- $O(n \log \log n)$ 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 $\text{RMQ}(i, j)$

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 2

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

Solution 3

- $O(n \log \log n)$ space
- $O(n \log \log n)$ 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 $\text{RMQ}(i, j)$
the output is the location of the smallest element in $A[i, j]$

<table>
<thead>
<tr>
<th>Solution 1</th>
<th>Solution 2</th>
<th>Solution 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$ space</td>
<td>$O(n \log n)$ space</td>
<td>$O(n \log \log n)$ space</td>
</tr>
<tr>
<td>$O(n)$ prep time</td>
<td>$O(n \log \log n)$ prep time</td>
<td>$O(n \log \log n)$ prep time</td>
</tr>
<tr>
<td>$O(\log n)$ query time</td>
<td>$O(1)$ query time</td>
<td>$O(1)$ query time</td>
</tr>
</tbody>
</table>

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