Welcome to CS166!

- Four handouts available up front.
  - Also available online!
- Today:
  - Why study data structures?
  - The range minimum query problem.
Why Study Data Structures?
Why Study Data Structures?

- **Explore where theory meets practice.**
  - Some of the data structures we'll cover are used extensively in practice. Many were invented about twenty miles from here!

- **Challenge your intuition for the limits of efficiency.**
  - You'd be amazed how many times we'll take a problem you're sure you know how to solve and then see how to solve it faster.

- **See the beauty of theoretical computer science.**
  - We'll cover some amazingly clever theoretical techniques in the course of this class. You'll love them.

- **Equip yourself to solve complex problems.**
  - Powerful data structures make excellent building blocks for solving seemingly difficult problems.
Course Staff

Keith Schwarz (htiek@cs.stanford.edu)

Benjamin Plaut
Mitchell Douglass
Rafa Musa
Sam Redmond

Course Staff Mailing List:
cs166-spr1718-staff@lists.stanford.edu
The Course Website

http://cs166.stanford.edu
Recommended Reading


- You'll want the third edition for this course.

- Available in the bookstore; several copies on hold at the Engineering Library.
Prerequisites

• **CS161** (Design and Analysis of Algorithms)
  • We'll assume familiarity with asymptotic notation, correctness proofs, algorithmic strategies (e.g. divide-and-conquer, dynamic programming), classical algorithms, recurrence relations, universal hashing, etc.

• **CS107** (Computer Organization and Systems)
  • We'll assume comfort working from the command-line, designing and testing nontrivial programs, and manipulating bitwise representations of data. You should have some knowledge of the memory hierarchy. You should also know how to code in both high-level and low-level languages.
Grading Policies

Midterm: **Tuesday, May 29**
7PM - 10PM
Location TBA

- 1/3 Assignments
- 1/3 Midterm
- 1/3 Final Project
Problem Sets

• The first problem set of the quarter, Problem Set 0, goes out today. It’s due next Tuesday at 2:30PM.

• This problem set is designed as a refresher on the techniques and concepts that we’ll be using over the course of this class.

• You’re welcome to work in pairs or individually. See the “Problem Set Policies” handout for more details.
Let’s Get Started!
Range Minimum Queries
The RMQ Problem

- The *Range Minimum Query problem* (*RMQ* for short) is the following:

  Given an array $A$ and two indices $i \leq j$, what is the smallest element out of $A[i], A[i + 1], ..., A[j - 1], A[j]$?
The RMQ Problem

• The *Range Minimum Query problem* (RMQ for short) is the following:

  Given an array $A$ and two indices $i \leq j$, what is the smallest element out of $A[i], A[i + 1], ..., A[j - 1], A[j]$?

• Notation: We'll denote a range minimum query in array $A$ between indices $i$ and $j$ as $\text{RMQ}_A(i, j)$.

• For simplicity, let's assume 0-indexing.
A Trivial Solution

• There's a simple $O(n)$-time algorithm for evaluating $\text{RMQ}_A(i, j)$: just iterate across the elements between $i$ and $j$, inclusive, and take the minimum!

• So... why is this problem at all algorithmically interesting?

• Suppose that the array $A$ is fixed in advance and you're told that we're going to make a number of different queries on it.

• Can we do better than the naïve algorithm?
An Observation

- In an array of length $n$, there are only $\Theta(n^2)$ possible queries.
- Why?

1 subarray of length 5
2 subarrays of length 4
3 subarrays of length 3
4 subarrays of length 2
5 subarrays of length 1
A Different Approach

- There are only $\Theta(n^2)$ possible RMQs in an array of length $n$.
- If we precompute all of them, we can answer RMQ in time $O(1)$ per query.
Building the Table

- One simple approach: for each entry in the table, iterate over the range in question and find the minimum value.
- How efficient is this?
  - Number of entries: $\Theta(n^2)$.
  - Time to evaluate each entry: $O(n)$.
  - Time required: $O(n^3)$.
- The runtime is $O(n^3)$ using this approach. Is it also $\Theta(n^3)$?
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each entry in yellow requires at least \( n / 2 = \Theta(n) \) work to evaluate.

There are roughly \( n^2 / 8 = \Theta(n^2) \) entries here.

Total work required: \( \Theta(n^3) \)
A Different Approach

- Naïvely precomputing the table is inefficient.
- Can we do better?
- **Claim:** We can precompute all subarrays in time $\Theta(n^2)$ using dynamic programming.
A Different Approach

- Naïvely precomputing the table is inefficient.
- Can we do better?
- **Claim:** We can precompute all subarrays in time $\Theta(n^2)$ using dynamic programming.
A Different Approach

• Naïvely precomputing the table is inefficient.
• Can we do better?
• **Claim:** We can precompute all subarrays in time $\Theta(n^2)$ using dynamic programming.
We'll say that an RMQ data structure has time complexity \( \langle p(n), q(n) \rangle \) if

- preprocessing takes time at most \( p(n) \) and
- queries take time at most \( q(n) \).

We now have two RMQ data structures:

- \( \langle O(1), O(n) \rangle \) with no preprocessing.
- \( \langle O(n^2), O(1) \rangle \) with full preprocessing.

These are two extremes on a curve of tradeoffs: no preprocessing versus full preprocessing.

**Question:** Is there a “golden mean” between these extremes?
Another Approach: *Block Decomposition*
A Block-Based Approach

- Split the input into $O(n / b)$ blocks of some “block size” $b$.
- Here, $b = 3$. 

```
31  41  59  26  53  58  97  93  23  84  62  64  33  83  27
```
A Block-Based Approach

• Split the input into $O(n / b)$ blocks of some “block size” $b$.
  • Here, $b = 3$.
• Compute the minimum value in each block.

| 31 | 41 | 59 | 26 | 53 | 58 | 97 | 93 | 23 | 84 | 62 | 64 | 33 | 83 | 27 |
A Block-Based Approach

• Split the input into $O(n / b)$ blocks of some “block size” $b$.
  • Here, $b = 3$.

• Compute the minimum value in each block.
Analyzing the Approach

- Let's analyze this approach in terms of $n$ and $b$.
- Preprocessing time:
  - $O(b)$ work on $O(n / b)$ blocks to find minima.
  - Total work: $O(n)$.
- Time to evaluate $\text{RMQ}_A(i, j)$:
  - $O(1)$ work to find block indices (divide by block size).
  - $O(b)$ work to scan inside $i$ and $j$'s blocks.
  - $O(n / b)$ work looking at block minima between $i$ and $j$.
  - Total work: $O(b + n / b)$. 

<table>
<thead>
<tr>
<th>31</th>
<th>41</th>
<th>59</th>
<th>26</th>
<th>53</th>
<th>58</th>
<th>97</th>
<th>93</th>
<th>23</th>
<th>84</th>
<th>62</th>
<th>64</th>
<th>33</th>
<th>83</th>
<th>27</th>
</tr>
</thead>
</table>
Intuiting $O(b + n / b)$

- As $b$ increases:
  - The $b$ term rises (more elements to scan within each block).
  - The $n / b$ term drops (fewer blocks to look at).
- As $b$ decreases:
  - The $b$ term drops (fewer elements to scan within a block).
  - The $n / b$ term rises (more blocks to look at).
- Is there an optimal choice of $b$ given these constraints?
Optimizing $b$

- What choice of $b$ minimizes $b + n / b$?
- Start by taking the derivative:
  \[
  \frac{d}{db}(b+n/b) = 1 - \frac{n}{b^2}
  \]
- Setting the derivative to zero:
  \[
  1 - \frac{n}{b^2} = 0
  \]
  \[
  1 = \frac{n}{b^2}
  \]
  \[
  b^2 = n
  \]
  \[
  b = \sqrt{n}
  \]
- Asymptotically optimal runtime is when $b = n^{1/2}$.
- In that case, the runtime is
  \[
  O(b + n / b) = O(n^{1/2} + n / n^{1/2}) = O(n^{1/2} + n^{1/2}) = O(n^{1/2})
  \]
Summary of Approaches

• Three solutions so far:
  • Full preprocessing: \( O(n^2), O(1) \).
  • Block partition: \( O(n), O(n^{1/2}) \).
  • No preprocessing: \( O(1), O(n) \).

• Modest preprocessing yields modest performance increases.

• Question: Can we do better?
A Second Approach: *Sparse Tables*
An Intuition

• The \( \langle O(n^2), O(1) \rangle \) solution gives fast queries because every range we might look up has already been precomputed.

• This solution is slow overall because we have to compute the minimum of every possible range.

• **Question:** Can we still get constant-time queries without preprocessing all possible ranges?
An Observation
An Observation
The Intuition

- It's still possible to answer any query in time $O(1)$ without precomputing RMQ over all ranges.
- If we precompute the answers over too many ranges, the preprocessing time will be too large.
- If we precompute the answers over too few ranges, the query time won't be $O(1)$.
- **Goal:** Precompute RMQ over a set of ranges such that
  - There are $o(n^2)$ total ranges, but
  - there are enough ranges to support $O(1)$ query times.
Some Observations
The Approach

- For each index $i$, compute RMQ for ranges starting at $i$ of size 1, 2, 4, 8, 16, ..., $2^k$ as long as they fit in the array.
  - Gives both large and small ranges starting at any point in the array.
  - Only $O(\log n)$ ranges computed for each array element.
  - Total number of ranges: $O(n \log n)$.
- **Claim:** Any range in the array can be formed as the union of two of these ranges.
Creating Ranges
Creating Ranges

- 7
- 4
- 4
Doing a Query

• To answer $\text{RMQ}_A(i, j)$:
  
  • Find the largest $k$ such that $2^k \leq j - i + 1$.
    
      – With the right preprocessing, this can be done in time $O(1)$; you'll figure out how in Problem Set One.
  
  • The range $[i, j]$ can be formed as the overlap of the ranges $[i, i + 2^k - 1]$ and $[j - 2^k + 1, j]$.
  
  • Each range can be looked up in time $O(1)$.
  
  • Total time: $O(1)$. 
Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.
Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$. 
Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$.
Precomputing the Ranges

- There are $O(n \log n)$ ranges to precompute.
- Using dynamic programming, we can compute all of them in time $O(n \log n)$. 

![Diagram showing ranges and binary representations]
Sparse Tables

- This data structure is called a *sparse table*.
- It gives an $\langle O(n \log n), O(1) \rangle$ solution to RMQ.
- This is asymptotically better than precomputing all possible ranges!
The Story So Far

• We now have the following solutions for RMQ:
  • Precompute all: \(\langle O(n^2), \ O(1) \rangle\).
  • Sparse table: \(\langle O(n \log n), \ O(1) \rangle\).
  • Blocking: \(\langle O(n), \ O(n^{1/2}) \rangle\).
  • Precompute none: \(\langle O(1), \ O(n) \rangle\).

• Can we do better?
A Third Approach: *Hybrid Strategies*
Blocking Revisited

This is just RMQ on the block minima!
Blocking Revisited

This is just RMQ inside the blocks!
The Setup

- Here's a new possible route for solving RMQ:
  - Split the input into blocks of some block size $b$.
  - For each of the $O(n/b)$ blocks, compute the minimum.
  - *Construct an RMQ structure on the block minima.*
  - *Construct RMQ structures on each block.*
  - Combine the local RMQ answers to solve RMQ globally.
- This technique of splitting a problem into a bunch of smaller pieces unified by a larger piece is common in data structure design.
Combinations and Permutations

- The decomposition we just saw isn't a single data structure; it's a framework for data structures.
- We get to choose
  - the block size,
  - which RMQ structure to use on top, and
  - which RMQ structure to use for the blocks.
- Summary and block RMQ structures don't have to be the same type of RMQ data structure – we can combine different structures together to get different results.
The Framework

• Suppose we use a \( \langle p_1(n), q_1(n) \rangle \)-time RMQ solution for the block minima and a \( \langle p_2(n), q_2(n) \rangle \)-time RMQ solution within each block.

• Let the block size be \( b \).

• In the hybrid structure, the preprocessing time is

\[
O(n + p_1(n / b) + (n / b) p_2(b))
\]

<table>
<thead>
<tr>
<th>31</th>
<th>41</th>
<th>59</th>
<th>26</th>
<th>53</th>
<th>58</th>
<th>84</th>
<th>62</th>
<th>64</th>
<th>33</th>
<th>83</th>
<th>27</th>
</tr>
</thead>
</table>
The Framework

• Suppose we use a $\langle p_1(n), q_1(n) \rangle$-time RMQ solution for the block minima and a $\langle p_2(n), q_2(n) \rangle$-time RMQ solution within each block.

• Let the block size be $b$.

• In the hybrid structure, the preprocessing time is

$$O(n + p_1(n / b) + (n / b) p_2(b))$$

• The query time is

$$O(q_1(n / b) + q_2(b))$$
A Sanity Check

- The $(O(n), O(n^{1/2}))$ block-based structure from earlier uses this framework with the $(O(1), O(n))$ no-preprocessing RMQ structure and $b = n^{1/2}$.
- According to our formulas, the preprocessing time should be

$$O(n + p_1(n / b) + (n / b) p_2(b))$$
$$= O(n + 1 + n / b)$$
$$= O(n)$$

- The query time should be

$$O(q_1(n / b) + q_2(b))$$
$$= O(n / b + b)$$
$$= O(n^{1/2})$$

- Looks good so far!

For Reference

\[
p_1(n) = O(1) \quad q_1(n) = O(n) \quad p_2(n) = O(1) \quad q_2(n) = O(n) \quad b = n^{1/2}
\]
An Observation

- A sparse table takes time $O(n \log n)$ to construct on an array of $n$ elements.
- With block size $b$, there are $O(n / b)$ total blocks.
- Time to construct a sparse table over the block minima: $O((n / b) \log (n / b))$.
- Since $\log (n / b) = O(\log n)$, the time to build the sparse table is at most $O((n / b) \log n)$.
- **Cute trick:** If $b = \Theta(\log n)$, the time to construct a sparse table over the minima is
  
  $O((n / b) \log n) = O((n / \log n) \log n) = O(n)$
One Possible Hybrid

- Set the block size to $\log n$.
- Use a sparse table for the top-level structure.
- Use the “no preprocessing” structure for each block.
- Preprocessing time:

$$O(n + p_1(n / b) + (n / b) p_2(b))$$

$$= O(n + n + n / \log n)$$

$$= O(n)$$

- Query time:

$$O(q_1(n / b) + q_2(b))$$

$$= O(1 + \log n)$$

$$= O(\log n)$$

- An $\langle O(n), O(\log n) \rangle$ solution!

For Reference

$p_1(n) = O(n \log n)$
$q_1(n) = O(1)$
$p_2(n) = O(1)$
$q_2(n) = O(n)$

$b = \log n$
Another Hybrid

- Let's suppose we use the $\langle O(n \log n), O(1) \rangle$ sparse table for both the top and bottom RMQ structures with a block size of $\log n$.

- The preprocessing time is

$$O(n + p_1(n / b) + (n / b) p_2(b))$$
$$= O(n + n + (n / \log n) b \log b)$$
$$= O(n + (n / \log n) \log n \log \log n)$$
$$= O(n \log \log n)$$

- The query time is

$$O(q_1(n / b) + q_2(b))$$
$$= O(1)$$

- We have an $\langle O(n \log \log n), O(1) \rangle$ solution to RMQ!

For Reference

\[
p_1(n) = O(n \log n) \\
q_1(n) = O(1) \\
p_2(n) = O(n \log n) \\
q_2(n) = O(1) \\
b = \log n
\]
One Last Hybrid

- Suppose we use a sparse table for the top structure and the \(O(n), O(\log n)\) solution for the bottom structure. Let's choose \(b = \log n\).

- The preprocessing time is
  
  \[
  O(n + p_1(n / b) + (n / b) p_2(b))
  \]
  
  \[
  = O(n + n + (n / \log n) b)
  \]
  
  \[
  = O(n + n + (n / \log n) \log n)
  \]
  
  \[
  = O(n)
  \]

- The query time is
  
  \[
  O(q_1(n / b) + q_2(b))
  \]
  
  \[
  = O(1 + \log \log n)
  \]
  
  \[
  = O(\log \log n)
  \]

- We have an \(O(n), O(\log \log n)\) solution to RMQ!
Where We Stand

- We've seen a bunch of RMQ structures today:
  - No preprocessing: $\langle O(1), O(n) \rangle$
  - Full preprocessing: $\langle O(n^2), O(1) \rangle$
  - Block partition: $\langle O(n), O(n^{1/2}) \rangle$
  - Sparse table: $\langle O(n \log n), O(1) \rangle$
  - Hybrid 1: $\langle O(n), O(\log n) \rangle$
  - Hybrid 2: $\langle O(n \log \log n), O(1) \rangle$
  - Hybrid 3: $\langle O(n), O(\log \log n) \rangle$
Where We Stand

We've seen a bunch of RMQ structures today:

- **No preprocessing:** \(O(1), O(n)\)
- **Full preprocessing:** \(O(n^2), O(1)\)
- **Block partition:** \(O(n), O(n^{1/2})\)
- **Sparse table:** \(O(n \log n), O(1)\)
- **Hybrid 1:** \(O(n), O(\log n)\)
- **Hybrid 2:** \(O(n \log \log n), O(1)\)
- **Hybrid 3:** \(O(n), O(\log \log n)\)
Where We Stand

We've seen a bunch of RMQ structures today:

- No preprocessing: \( \langle O(1), O(n) \rangle \)
- Full preprocessing: \( \langle O(n^2), O(1) \rangle \)
- Block partition: \( \langle O(n), O(n^{1/2}) \rangle \)
- Sparse table: \( \langle O(n \log n), O(1) \rangle \)
- Hybrid 1: \( \langle O(n), O(\log n) \rangle \)
- Hybrid 2: \( \langle O(n \log \log n), O(1) \rangle \)
- Hybrid 3: \( \langle O(n), O(\log \log n) \rangle \)
Is there an $\langle O(n), O(1) \rangle$ solution to RMQ?

Yes!
Next Time

- **Cartesian Trees**
  - A data structure closely related to RMQ.

- **The Method of Four Russians**
  - A technique for shaving off log factors.

- **The Fischer-Heun Structure**
  - A deceptively simple, asymptotically optimal RMQ structure.