CS341: ALGORITHMS (S22)
Lecture 1: course overview and Bentley’s problem
Readings: CLRS Chapter 1

Trevor Brown
https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca

TABLE OF CONTENTS
- Course mechanics
- Overview of course material
- Worked example: Bentley’s problem
  - Multiple solutions, demonstrating different algorithm design techniques

COURSE MECHANICS
- Hybrid course
  - Lectures 1 to 11 online (maybe more)
  - In person Q&A / Discussion / Tutorial in class time
  - Remaining lectures in person
- Course website: https://student.cs.uwaterloo.ca/~cs341/
  - Syllabus, calendar, policies, slides, assignments...
  - Read this and mark important dates.
- Keep up with the lectures: Material builds over time...
- Piazza: For questions and announcements.

ASSESSMENTS
- All sections have same assignments, midterm and final
  - Notify us long before the deadline of severe problems that will cause you to miss an assignment
  - Midterm and final are anticipated to be in person
    - If university policy makes this impossible, we will transition to take-home exams
  - See website for grading scheme

TEXTBOOK
- Introduction to Algorithms, Third Edition
  - Cormen, Leiserson, Rivest and Stein
  - Available for free via library website!
  - You are expected to know
    - entire textbook sections, as listed on course website
    - all the material presented in lectures
      [unless we explicitly say you aren’t responsible for it]
ACADEMIC OFFENSES

- Beware plagiarism
  - High level discussion about solutions with individual students is OK
  - Don’t take written notes away from such discussions
  - Class-wide discussion of solutions is not OK (until the deadline)

COURSE OVERVIEW
Sketching out the road ahead

WHY IS CS341 IMPORTANT FOR YOU?

- Algorithms is the heart of CS
  - It appears often in later courses
  - It dominates technical interviews
    - Master this material... make your interviews easy!
- Designing algorithms is creative work
  - Useful for some of the more interesting jobs out there
  - And, you want to graduate...

WHAT IS AN ALGORITHM?

- Informally: A description of input, and the desired output

WHAT IS A COMPUTATIONAL PROBLEM?

- Informally: A well-defined procedure (sequence of steps) to solve a computational problem

EXAMPLES OF COMPUTATIONAL PROBLEMS

<table>
<thead>
<tr>
<th>Sorting</th>
<th>Matrix Multiplication</th>
<th>Traveling Salesman Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>An array of integers (in arbitrary order)</td>
<td>Two n x n matrices A, B</td>
</tr>
<tr>
<td>Desired output</td>
<td>Some array of integers in increasing order</td>
<td>A set S of cities, and distances between each pair of cities</td>
</tr>
<tr>
<td></td>
<td>A matrix C=A*B</td>
<td>Shortest possible path that visits each city, and returns to the origin city</td>
</tr>
</tbody>
</table>

ANALYSIS OF ALGORITHMS

- Every software program uses resources:
  - CPU instructions → we call this time
  - Memory (RAM) → we call this space
  - Others: I/O, network bandwidth/messages, locks... (not covered in this course)
- Analysis is the study of how many resources an algorithm uses
  - Usually using big-O notation (to ignore constant factors)
TAXONOMY OF ALGORITHMS

- Serial vs Parallel
  - Serial: One instruction at a time
  - Parallel: Multiple instructions at once

Deterministic vs Randomized
- D: On multiple runs on same input, always do same thing
- R: On multiple runs on same input, may do different things
  Example: flip a coin, and base your next action on the result

Exact vs Approximate
- Exact: exact solution to the problem
- Approximate: produce something “close” to a solution

This course mainly covers: Serial, deterministic, exact

TRACTABILITY: DO ALL PROBLEMS HAVE FAST SOLUTIONS?

- For some problems, such as the traveling salesman problem, we have only found exponential time algorithms.
- These algorithms take exponentially longer to solve the problem as the number of cities increases!
- Informally: adding one city doubles the runtime
- This severely limits our ability to solve “real world” inputs...
- Is there a way around this limitation? Or should we stop trying?
Open question (P vs NP): is it possible to solve such problems in polynomial time?

Mathematical Tools to Analyze Algorithms
- Big-oh notation
- Recursion Tree
- Master method
- Substitution method
- Exchange Arguments
- Greedy-stays-ahead Arguments

Intractable Problems
- P vs NP
- Poly-time reductions
- Undecidability

Common Algorithm Design Paradigms
- Divide-and-Conquer
- Greedy
- Dynamic Programming
- Exhaustive search / brute force

Fundamental (& Fast) Algorithms for Tractable Problems
- Mergesort
- QuickSort
- BFS/DFS
- Floyd-Warshall APSP
- Topological Sort
- ...

Topics to Cover

BENTLEY’S PROBLEM
A worked example to demonstrate algorithm design

Bentley’s Problem (introductory example)
Given an array of n integers, A[1], ..., A[n], find the maximum sum of consecutive entries of A (return 0 if all entries of A are negative).

Example 1
Array index
1 | 7 | 4 | 0 | 2 | 1 | 3 | 1
Solution: 19
(take all of A[1..8])

Example 2
Array index
3 | -7 | -4 | -1 | -2 | -3 | -1
Solution: 0
take no elements of A

Example 3
Array index
-1 | -2 | -9 | 0 | 2 | 1 | 3 | -1
Solution: 8
(take A[3..7])

CS341: Before → After

1. Fundamental Algorithms
2. Fundamental Design Paradigms
3. Tractability/Intractability
Bentley's Problem: Solution 1
Design: brute force

Try all combinations of $i$, $j$.
And for each combination, sum over $k = i$ to $j$.

Time: 

Bentley's Problem: Solution 2
Avoid summing over $k = i$ to $j$.

Design: slightly better brute force

Bentley's Problem: Solution 3

Divide-and-Conquer can also be used here.

Our solution must either be entirely in the left part, or entirely in the right part, or it must be crossing the partition line.

WHY $A[i \ldots j]$ IS MAXIMAL

Suppose not for contradiction.

Then some $A[i \ldots j]$ that crosses the partition has a larger sum.

But both are impossible!

This sum is bigger.

So either $\sum_{i} > \sum_{j}$

or $\sum_{j} > \sum_{i}$

Therefore: Find the maximum subarray for left part ($\text{max}(L)$) and right part ($\text{max}(R)$) (done by recursive call). Find the maximum subarray "going over the middle partition line" ($\text{max}(M)$).

This can be done in linear time $O(n)$.

The solution is $\text{max}(\text{max}(L), \text{max}(R), \text{max}(M))$. Return $\text{max}(10, 3, 5) = 10$.
Bentley's Problem: Solution 4

- Define: \( \text{include}(j) = \text{maximum sum of consecutive entries in array } A[1..j] \) if the sum must include \( A[j] \)

- Define: \( \text{exclude}(j) = \text{maximum sum of consecutive entries in array } A[1..j] \) if the sum must exclude \( A[j] \)

- Observe: if we could solve for \( \text{include}(j), \text{exclude}(j) \) for all \( j \), then the solution to our problem would be \( \max( \text{include}(n), \text{exclude}(n) ) \)

Example: computing these recurrence relations with two arrays

**Base case:** \( \text{include}(1) = A[1] \)

\[ \text{include}(j) = \max( A[j], A[j] + \text{include}(j-1) ) \]

**Base case:** \( \text{exclude}(1) = 0 \)

\[ \text{exclude}(j) = \max( \text{include}(j-1), \text{exclude}(j-1) ) \]

Full solution is \( \max \) of these two: 8

Return \( \max(4, 4, 8) = 8 \)

**Recurrence relations, recursion tree methods, master theorem...**

This result is really quite good... but can we do asymptotically better?

Recurrence relations, recursion tree methods, master theorem...
At this time, include contains exactly \( \text{include}[j] \) and similarly for \( \text{exclude} \)...

And these contain exactly \( \text{exclude}[n] \) and \( \text{include}[n] \)

Some running time, but only \( O(1) \) space (besides the input array)

\[ \text{function solveDP2(A)} \]
\[ \text{define arrays exclude[1..n], include[1..n]} \]
\[ \text{exclude}[1] = 0 \]
\[ \text{include}[1] = 1 \]
\[ \text{for } j = 2 \ldots n \]
\[ \text{exclude}[j] = \max \{ \text{exclude}[j-1], \text{exclude}[j-1] \} \]
\[ \text{include}[j] = \max \{ \text{include}[j], \text{exclude}[j-1], \text{include}[j-1], \text{include}[j] \} \]
\[ \text{return max(exclude[1], include[1])} \]

\[ \text{function solveDP2_spaceefficient(A)} \]
\[ \text{exclude} = 0 \]
\[ \text{include} = 1 \]
\[ \text{for } j = 2 \ldots n \]
\[ \text{exclude} = \max \{ \text{include}, \text{exclude} \} \]
\[ \text{include} = \max \{ \text{A}[j], \text{A}[j] + \text{include} \} \]
\[ \text{return max(exclude, include)} \]

BENTLEY’S PROBLEM: TIME CONSTRAINTS

- Consider solutions implemented in C
- Some values measured (on a Pentium II)
- Some estimated from other measurements
- \( \varepsilon \) represents time under 0.01s

| Max. size problem solved in | Time to \( \varepsilon \) | Time to estimate | Time to run | Max. size problem solved in
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( 8 \times 2 )</td>
<td>0.012</td>
<td>36.7</td>
<td>529</td>
<td>( 8 \times 8 )</td>
</tr>
<tr>
<td>( 16 \times 2 )</td>
<td>0.017</td>
<td>4.6</td>
<td>530</td>
<td>( 128 \times 8 )</td>
</tr>
<tr>
<td>( 32 \times 2 )</td>
<td>0.195</td>
<td>12 hours</td>
<td>3,700</td>
<td>( 512 \times 8 )</td>
</tr>
<tr>
<td>( 64 \times 2 )</td>
<td>2.168</td>
<td>50 days</td>
<td>3.7M</td>
<td>( 1K \times 8 )</td>
</tr>
<tr>
<td>( 128 \times 2 )</td>
<td>24.57</td>
<td>1.5 years</td>
<td>&gt; age of life</td>
<td>( 10K \times 8 )</td>
</tr>
<tr>
<td>( 256 \times 2 )</td>
<td>24.57</td>
<td>&gt; age of universe</td>
<td>&gt; age of universe</td>
<td>( 100K \times 8 )</td>
</tr>
<tr>
<td>( 512 \times 2 )</td>
<td>24.57</td>
<td>&gt; age of universe</td>
<td>&gt; age of universe</td>
<td>( 1M \times 8 )</td>
</tr>
<tr>
<td>( 1K \times 2 )</td>
<td>24.57</td>
<td>&gt; age of universe</td>
<td>&gt; age of universe</td>
<td>( 10M \times 8 )</td>
</tr>
<tr>
<td>( 10K \times 2 )</td>
<td>24.57</td>
<td>&gt; age of universe</td>
<td>&gt; age of universe</td>
<td>( 100M \times 8 )</td>
</tr>
</tbody>
</table>

HOW ABOUT A MORE MODERN SYSTEM? ☺

- AMD Threadripper 3970x (2020)
- Pentium II (circa 1997)

<table>
<thead>
<tr>
<th>Sol 1</th>
<th>Sol 2</th>
<th>Sol 3</th>
<th>Sol 4</th>
<th>Sol 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AMD Threadripper 3970x (2020)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium II (circa 1997)</td>
</tr>
</tbody>
</table>

BONUS

- Trevor’s study-song of the day
- Tool - Descending
  - youtu.be/PcS0kWFlscw