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 soon 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)
WHY IS CS341 IMPORTANT FOR YOU?

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

CS 341 is a required course for all CS

WHAT IS A COMPUTATIONAL PROBLEM?

1. Informally: A description of input, and the desired output

WHAT IS AN ALGORITHM?

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

EXAMPLES OF COMPUTATIONAL PROBLEMS

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

ANALYSIS OF ALGORITHMS

1. Every software program uses resources
   - CPU instructions → we call this time
   - Memory (RAM) → we call this space
2. Others: I/O, network bandwidth/messages, locks... (not covered in this course)
3. Analysis is the study of how many resources an algorithm uses
4. Usually using big-O notation (to ignore constant factors)
This course mainly covers: Serial, deterministic, exact

Topics to Cover

**Fundamental (Fast) Algorithms for Tractable Problems**
- MergeSort
- QuickSort
- BFS/DFS
- Dijkstra's
- Kruskal or Prim
- Topological Sort
- ...

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

**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

CS341: Before → After

1. Fundamental Algorithms
2. Fundamental Design Paradigms
3. Tractability/Intractability

Math Techniques for Algorithm Analysis

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 today: 1 1 7 4 0 2 1 3 1 1
```
Solution: 19 (take all of A[1..8])

**Example 2**
```
Array today: -1 -7 -4 -1 -2 -1 -3 -1 -1
```
Solution: 0 (take no elements of A)

**Example 3**
```
Array today: 1 1 7 4 0 2 1 3 1 1
```
Solution: 8 (take A[3..7])
Bentley's Problem: Solution 1 - Design: brute force

```
max := 0;
for i := 1 to n do
    for j := 1 to n do
        // compute A[i] + ... + A[j]
        sum := 0;
        for k := i to j do
            sum := sum + A[k];
        // compare to maximum sum observed so far
        if sum > max then max := sum;
```

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

Time:

Bentley's Problem: Solution 2 - Design: slightly better brute force

```
max := 0;
for i := 1 to n do
    for j := i to n do
        // compute A[i] + ... + A[j]
        sum := 0;
        for k := i to j do
            sum := sum + A[k];
        // compare to maximum sum observed so far
        if sum > max then max := sum;
```

Avoid summing over k = i...j

Time:

Bentley's Problem: Solution 3

Divide-and-Conquer can also be used here.
Divide an array into two equally-sized parts.
Our solution must either be entirely in the left part, or entirely in the right part, or it must be crossing the partition line.

Case 1: optimal sol'n is entirely in L
Case 2: optimal sol'n is entirely in R
Case 3: optimal sol'n crosses the partition

Let's see how...

Find: maximum subarray going over the middle partition

We can prove B[i...j] is the maximum subarray going over the middle partition!

WHY A[i...j] IS MAXIMAL

• Suppose not for contradiction
• Then some A[i'...j'] that crosses the partition has a larger sum

This sum is bigger
So either $i' < i$ or $j' > j$
Bentley's Problem: Solution 4
Design: dynamic programming

- Define: \text{include}(j) = \text{maximum sum of consecutive entries in array } A[1..j]
  
- Define: \text{exclude}(j) = \text{maximum sum of consecutive entries in array } A[1..j]
  
- Observe: If we could solve for \text{include}(j), \text{exclude}(j) for all \( j \), then the solution to our problem would be \( \max(\text{include}(n), \text{exclude}(n)) \)

Recall the definition:
- \text{include}(j) = \max \text{ of } \{ A[1..j], \text{include}(j-1) \}
- \text{exclude}(j) = \max \text{ of } \{ A[1..j], \text{exclude}(j-1) \}

Example: computing these recurrences relations with two arrays

### Base case:
- \text{include}(1) = A[1]
- \text{exclude}(1) = 0

### Inductive step:
- \text{include}(j) = \max(\{ A[1..j], \text{include}(j-1) \})
- \text{exclude}(j) = \max(\{ A[1..j], \text{exclude}(j-1) \})

**Recall:**
- \text{Base case:} \text{include}(1) = A[1]
- \text{Base case:} \text{exclude}(1) = 0

**Recursive case:**
- \text{include}(j) = \max(\text{include}(j-1), \text{exclude}(j-1))
- \text{exclude}(j) = \max(\text{include}(j-1), \text{exclude}(j-1))
At this time, include contains exactly \( \text{include}[j-1] \) and similarly for \( \text{exclude} \)… And these contain exactly \( \text{exclude}[n] \) and \( \text{include}[n] \) respectively.

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

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

**BENTLEY’S PROBLEM: TIME CONSTRAINTS**

<table>
<thead>
<tr>
<th>Max. size of problem solved as</th>
<th>Sol. 1</th>
<th>Sol. 2</th>
<th>Sol. 3</th>
<th>Sol. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>1 mL</td>
<td>10 mL</td>
<td>100 mL</td>
<td>1 L</td>
</tr>
<tr>
<td>10 mL</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>100 mL</td>
<td>0.04s</td>
<td>0.12s</td>
<td>0.35s</td>
<td>0.5s</td>
</tr>
<tr>
<td>1 L</td>
<td>0.42s</td>
<td>1.4s</td>
<td>5.8s</td>
<td>52h</td>
</tr>
<tr>
<td>2 L</td>
<td>4.2s</td>
<td>16 s</td>
<td>24 s</td>
<td>142 days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time if ( n ) increases</th>
<th>x 2</th>
<th>x 2</th>
<th>x 2+</th>
<th>x 4</th>
<th>x 8</th>
</tr>
</thead>
</table>

**HOW ABOUT A MORE MODERN SYSTEM? 😊**

**AMD Threadripper 3970x (2020)**

<table>
<thead>
<tr>
<th>Sol. 1</th>
<th>Sol. 2</th>
<th>Sol. 3</th>
<th>Sol. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>mL</td>
<td>mL</td>
<td>mL</td>
<td>mL</td>
</tr>
<tr>
<td>1 mL</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>10 mL</td>
<td>0.04s</td>
<td>0.12s</td>
<td>0.35s</td>
</tr>
<tr>
<td>100 mL</td>
<td>0.42s</td>
<td>1.4s</td>
<td>5.8s</td>
</tr>
<tr>
<td>1 L</td>
<td>4.2s</td>
<td>16 s</td>
<td>24 s</td>
</tr>
<tr>
<td>2 L</td>
<td>142 days</td>
<td>142 days</td>
<td>142 days</td>
</tr>
</tbody>
</table>

**Pentium II (circa 1997)**

<table>
<thead>
<tr>
<th>Sol. 1</th>
<th>Sol. 2</th>
<th>Sol. 3</th>
<th>Sol. 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>mL</td>
<td>mL</td>
<td>mL</td>
<td>mL</td>
</tr>
<tr>
<td>1 mL</td>
<td>0.02s</td>
<td>0.02s</td>
<td>0.02s</td>
</tr>
<tr>
<td>10 mL</td>
<td>0.04s</td>
<td>0.12s</td>
<td>0.35s</td>
</tr>
<tr>
<td>100 mL</td>
<td>0.42s</td>
<td>1.4s</td>
<td>5.8s</td>
</tr>
<tr>
<td>1 L</td>
<td>4.2s</td>
<td>16 s</td>
<td>24 s</td>
</tr>
<tr>
<td>2 L</td>
<td>142 days</td>
<td>142 days</td>
<td>142 days</td>
</tr>
</tbody>
</table>

**BONUS**

- Trevor’s study-song of the day.
- Tool - Descending.
- [youtu.be/PcSoLwFisaw](youtu.be/PcSoLwFisaw)