TABLE OF CONTENTS

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

COURSE MECHANICS

- Course website: https://www.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 to be 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>Input</th>
<th>Matrice Multiplication</th>
<th>Traveling Salesman Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>An array of integers</td>
<td>Two n x n matrices A, b</td>
<td>A set S of cities, and distances between each pair of cities</td>
</tr>
<tr>
<td>Some array of integers</td>
<td>A matrix C=A* b</td>
<td>Shortest possible path that visits each city, and returns to the origin city</td>
</tr>
<tr>
<td>in increasing order</td>
<td></td>
<td></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?

Fundamental (& Fast) Algorithms for Tractable Problems

- MergeSort
- Strassen’s MM
- BFS/DFS
- Dijkstra’s SSSP
- Floyd-Warshall AASP
- Topological Sort
- …

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

Topics to Cover

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 Index: 1 7 4 0 2 1 3 1

Solution: 19 (take all of A[1..8])

**Example 2**

Array Index: 1 1 7 -7 -4 -1 -2 -2 -3 -1

Solution: 0 (take no elements of A)

**Example 3**

Array Index: 1 1 -7 0 0 2 -3 3 -1

Solution: 8 (take A[3..7])

**BENTLEY’S PROBLEM**

A worked example to demonstrate algorithm design
Bentley's Problem: Solution 1

Design: brute force

Try all combinations of \(i, j\).
And for each combination, sum over \(k = 0, \ldots, n\).

max := 0;
for \(i := 1\) to \(n\)
for \(j := 1\) to \(n\)
  // compute \(A[i] + \ldots + A[j]\)
  sum := 0;
  for \(k := i\) to \(j\)
    sum := sum + \(A[k]\);
  // compare to maximum sum observed so far
  if sum > max then max := sum;

Time:

Bentley's Problem: Solution 2

Design: slightly better brute force

Avoid summing over \(k = 0, \ldots, n\).

max := 0;
for \(i := 1\) to \(n\)
  // for each \(j\), compute \(A[i] + \ldots + A[j]\)
  sum := 0;
  for \(j := 1\) to \(n\)
    // update sum by adding the next entry \(A[j]\)
    sum := sum + \(A[j]\);
  // compare to maximum sum observed so far
  if sum > max then max := sum;

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

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:

\[
\begin{array}{c|c|c|c|c|c}
\hline
A & 1 & 2 & 3 & 4 & 5 \\
\hline
i & 1 & 2 & 3 & 4 & 5 \\
\hline
j & 1 & 2 & 3 & 4 & 5 \\
\hline
\end{array}
\]

But both are impossible!

\[
\begin{array}{c|c|c|c|c|c}
\hline
A & 1 & 2 & 3 & 4 & 5 \\
\hline
i & 1 & 2 & 3 & 4 & 5 \\
\hline
j & 1 & 2 & 3 & 4 & 5 \\
\hline
\end{array}
\]

This sum is bigger

So either \(i' > i\) or \(j' > j\)

Let's see how...

Find: maximum subarray going over the middle partition

Find \(i\) that maximizes the sum over \(i, \ldots, n/2\).
Find \(j\) that maximizes the sum over \((n/2 + 1), \ldots, j\).

We can prove \(A[i \ldots j]\) is the maximum subarray going over the middle partition.

\[
\begin{array}{c|c|c|c|c|c}
\hline
A & 1 & 3 & 1 & 5 & 2 & 5 & 3 & 1 \\
\hline
i & 1 & 3 & 1 & 5 & 2 & 5 & 3 & 1 \\
\hline
j & 1 & 3 & 1 & 5 & 2 & 5 & 3 & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
\hline
maxL = 10 & 3 & 5 & 2 & 1 & 0 \\
\hline
maxR = 3 & maxI = 5 & maxJ = 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
\hline
maxM = maxI + maxJ = 5 & maxL = 10 & maxR = 3 \\
\hline
\end{array}
\]

maxL = 10
maxR = 3
maxI = 5
maxJ = 0

Return \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 a \( \text{include}(j) \), \( \text{exclude}(j) \) for all \( j \), then the solution to our problem would be \( \max(\text{include}(n), \text{exclude}(n)) \)

**Example:** computing these recurrences with two arrays

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

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

\[
\text{include}\left[\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array}\right] = \left[\begin{array}{cccc}
1 & 3 & 3 & 7 \\
\end{array}\right]
\]

\[
\text{exclude}\left[\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\end{array}\right] = \left[\begin{array}{cccc}
1 & 2 & 1 & 4 \\
\end{array}\right]
\]

**Full solution** \( \text{max of these two: 8} \)

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

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

Recall the definition:

- \( \text{include}(j) \) = \text{max solution in } A[1..j] \text{ that includes } A[j] 
- \( \text{include}(j) \) = \text{max solution in } A[1..j] \text{ that excludes } A[j] 
- \( \text{include}(j) \) = \text{max solution in } A[1..j] \text{ that excludes } A[j-1] 
- \( \text{include}(j) \) = \text{max solution in } A[1..j] \text{ that includes } A[j-1] 

Recall:

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

**Recursive case:**

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

Let’s turn these recurrences into code...

```cpp
function solveDP(A) {
  define arrays exclude[1..n], include[1..n] 
  exclude[1] = 0
  for j = 2..n
    exclude[j] = max(exclude[j-1], include[j-1] + include[j])
    include[j] = max(include[j-1], exclude[j-1] + exclude[j])
  return max(exclude[n], include[n])
}
```

Do we actually need these *entire* arrays? Only really care about the *last* entry of each...
At this time, include contains exactly \( \text{include}[j-1] \)
And similarly for exclude...

And these contain exactly “exclude[]” and “include[]”

BENTLEY’S PROBLEM: TIME CONSTRAINTS

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

<table>
<thead>
<tr>
<th>( n ) (bit length)</th>
<th>Sol.1</th>
<th>Sol.2</th>
<th>Sol.3</th>
<th>Sol.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.00s</td>
<td>0.00s</td>
<td>0.02s</td>
<td>0.00s</td>
</tr>
<tr>
<td>1000</td>
<td>0.01s</td>
<td>0.01s</td>
<td>1.14s</td>
<td>3.12s</td>
</tr>
<tr>
<td>10k</td>
<td>0.04s</td>
<td>0.08s</td>
<td>2.16s</td>
<td>6.30s</td>
</tr>
<tr>
<td>100k</td>
<td>0.35s</td>
<td>0.84s</td>
<td>16.08s</td>
<td>47.24s</td>
</tr>
<tr>
<td>1M</td>
<td>1.94s</td>
<td>4.39s</td>
<td>264.65s</td>
<td>708.14s</td>
</tr>
<tr>
<td>10M</td>
<td>11.64s</td>
<td>26.94s</td>
<td>1658.56s</td>
<td>4467.36s</td>
</tr>
<tr>
<td>100M</td>
<td>112.35s</td>
<td>258.37s</td>
<td>16920.10s</td>
<td>46294.50s</td>
</tr>
<tr>
<td>1G</td>
<td>1123.5s</td>
<td>2589.37s</td>
<td>169201.00s</td>
<td>462945.00s</td>
</tr>
</tbody>
</table>

HOW ABOUT A MORE MODERN SYSTEM? ☺

AMD Threadripper 3970x (2020)

<table>
<thead>
<tr>
<th>( n ) (bit length)</th>
<th>Sol.1</th>
<th>Sol.2</th>
<th>Sol.3</th>
<th>Sol.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.00s</td>
<td>0.00s</td>
<td>0.02s</td>
<td>0.00s</td>
</tr>
<tr>
<td>1000</td>
<td>0.01s</td>
<td>0.01s</td>
<td>1.14s</td>
<td>3.12s</td>
</tr>
<tr>
<td>10k</td>
<td>0.04s</td>
<td>0.08s</td>
<td>2.16s</td>
<td>6.30s</td>
</tr>
<tr>
<td>100k</td>
<td>0.35s</td>
<td>0.84s</td>
<td>16.08s</td>
<td>47.24s</td>
</tr>
<tr>
<td>1M</td>
<td>1.94s</td>
<td>4.39s</td>
<td>264.65s</td>
<td>708.14s</td>
</tr>
<tr>
<td>10M</td>
<td>11.64s</td>
<td>26.94s</td>
<td>1658.56s</td>
<td>4467.36s</td>
</tr>
<tr>
<td>100M</td>
<td>112.35s</td>
<td>258.37s</td>
<td>16920.10s</td>
<td>46294.50s</td>
</tr>
<tr>
<td>1G</td>
<td>1123.5s</td>
<td>2589.37s</td>
<td>169201.00s</td>
<td>462945.00s</td>
</tr>
</tbody>
</table>

BONUS

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