A BIT ABOUT ME

• Interests
  • Big multi-core systems
  • Fast data structures and algorithms
  • ...
  • Gamer
  • Ex-metal drummer
COURSE MECHANICS

• Ask questions!
• Need an account in the student.cs environment
• Course website:
  • https://www.student.cs.uwaterloo.ca/~cs341/
  • Syllabus, calendar, policies, slides, assignments...
• Piazza: Join it. Use it to ask even more questions.
• Learn: Grades and assignment solutions
COURSE SYLLABUS

• Read it
• Mark important dates
ACADEMIC OFFENSES

• Sharing ideas: OK
• Sharing code / answers: BAD
• Using solutions from elsewhere: BAD
• Don’t cheat
GRADING SCHEME FOR CS 341

• Midterm (25%)
  • Tuesday, Feb. 26, 2019, 7:00-8:50 PM
• Assignments (30%)
  • There will be five assignments.
  • Work alone
• Final (45%)

Due Friday Jan. 25
Due Friday Feb. 8
Due Friday March 1
Due Friday March 22
Due Friday April 5
ASSIGNMENTS

• All sections have **same** assignments, midterm and final
• Assignments due at 6:00 PM on the due date
  • **No late submissions**
  • Notify me **long before** the deadline of severe problems that will cause you to miss an assignment
TEXTBOOK

  • Available **online for free** via the library website!
• You are expected to know
  • entire textbook sections, as listed on course website or Learn
  • all the material presented in class
WHY IS CS 341 IMPORTANT FOR YOU?

• Algorithms is the heart of CS
• Appears in later courses
• Appears in **technical interviews**
• Designing algorithms is creative work; can be fun!
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Introductory material</td>
</tr>
<tr>
<td>• What are <em>algorithms</em> and <em>computational problems</em>?</td>
</tr>
<tr>
<td>• What is algorithm <em>analysis</em>?</td>
</tr>
<tr>
<td>• The 3SUM problem</td>
</tr>
<tr>
<td>• Reductions</td>
</tr>
<tr>
<td>• Order notation</td>
</tr>
<tr>
<td>• Loop analysis techniques</td>
</tr>
</tbody>
</table>

Only *some* of this covered today!
WHAT IS AN ALGORITHM?

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

Think of an algorithm as a software program.

Input $\rightarrow$ *algorithm* $\rightarrow$ output

WHAT IS A COMPUTATIONAL PROBLEM?

• Informally: A description of some (class of) input, and the desired output
# EXAMPLES OF COMPUTATIONAL PROBLEMS

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorting</td>
<td>An array of integers (in arbitrary order)</td>
<td>Same array of integers in <strong>increasing</strong> order</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>Two ( n \times n ) matrices ( A, B )</td>
<td>( C = A \times 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>
SORTING

Selection Sort - 95 comparisons, 67 array accesses, 0.50 ms delay

http://panthema.net/2013/sound-of-sorting
MATRIX MULTIPLICATION

• Input: two n x n matrices A and B

\[
\begin{array}{ccc}
2 & 1 & 5 \\
3 & 2 & 2 \\
1 & 4 & 6 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 3 & 4 \\
2 & 1 & 1 \\
3 & 7 & 2 \\
\end{array}
\]

• Output: an n x n matrix representing A * B

\[
\begin{array}{ccc}
19 & 41 & 18 \\
13 & 25 & 19 \\
27 & 49 & 20 \\
\end{array}
\]
TRAVELING SALESMAN PROBLEM

• Input: map of cities & distances between each pair

• Output: shortest path that (1) visits all cities, and (2) starts and ends at the same city

How would you provide this input to a program?
TRAVELING SALESMAN PROBLEM

• Input: map of cities & distances between each pair

• Output: shortest path that (1) visits all cities, and (2) starts and ends at the same city

How would you output this path to the user?
ANALYSIS OF ALGORITHMS

• Every software program uses resources
• Algorithms are no exception
• Example Resources:
  • CPU instructions/cycles → typically called time
  • Memory (RAM) → typically called space
  • Network I/O or communication
    (not covered in this course)
ANALYSIS OF ALGORITHMS

- **Analysis** answers questions about **how many** resources an algorithm uses:
  - How much **time** does it take to run Algorithm X?
  - How much **space** does Algorithm X use?
- In this course we will typically analyze **time**
  - Formally: number of **CPU instructions** performed
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

• **Exact vs Approximate**
  - Exact: exact solution to the problem
  - Approximate: only approximate solution to the problem

This course: Serial, deterministic, exact
DO ALL PROBLEMS HAVE FAST SOLUTIONS?

• For some problems, such as the traveling salesman problem, we have only found exponential time algorithms.
  • These algorithms exponentially longer to solve the problem as the number of cities increases!
  • Informally: adding one city doubles the runtime
  • We want polynomial time solutions
• Open question (P vs NP): are there problems so hard that any solution must take exponential time on some inputs?
TRACTABLE VS INTRACTABLE

• Terminology used in the course
• Informally: a problem is tractable if:
  • An algorithm exists that can solve the problem for any possible input in polynomial time
• Informally: a problem is intractable if:
  • It is (probably) not tractable
  • I.e., all known algorithms that solve the problem take exponential time for some inputs
CS 341 Diagram

Fundamental (& Fast) Algorithms to Tractable Problems
- MergeSort
- Strassen's MM
- BFS/DFS
- Dijkstra's SSSP
- Kruskal's MST
- Floyd Warshall APSP
- 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

Other (Last Lecture)
- Parallel Algorithms
CS 341: Before → After

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

Math Techniques for Algorithm Analysis
SOME REAL MATERIAL

• Let’s briefly study the **sorting** problem
• Input: array **X** containing **n** integers **X[1]**, **X[2]**, ..., **X[n]**

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>2</th>
<th>37</th>
<th>5</th>
<th>9</th>
<th>55</th>
<th>20</th>
</tr>
</thead>
</table>

• Output: the **same** array, but with the **n** integers appearing in **increasing order**

|   |   2 |   5 |   9 |  10 |  20 |  37 |  55 |
A SIMPLE SORTING ALGORITHM

• **Selection sort**
  • Idea is simple
  • Find the smallest element and swap it with X[1]
  • Find the next smallest element and swap it with X[2]
  • ...
  • Repeat until array is sorted
SelectionSort Simulation

\[ i = 1 \]

\[
\begin{array}{ccccccc}
10 & 2 & 37 & 5 & 9 & 55 & 20 \\
\end{array}
\]

minElement: 2
minIndex: 2
SelectionSort Simulation

minElement: 5
minIndex: 4
SelectionSort Simulation

\[ i = 3 \]

\[
\begin{array}{ccccccc}
2 & 5 & 37 & 10 & 9 & 55 & 20 \\
\end{array}
\]

minElement: 9
minIndex: 5
SelectionSort Simulation

i = 4

minElement: 10
minIndex: 4
SelectionSort Simulation

i = 5

minElement: 20
minIndex: 7
SelectionSort Simulation

i = 6

minElement: 37
minIndex: 7
SelectionSort Simulation

| 2 | 5 | 9 | 10 | 20 | 37 | 55 |

Final Output
procedure selectionSort(Array X of size n):
    for i = 1 to n {
        let minIndex = i;
        for j = i+1 to n {
            if X[j] < X[minIndex]
                minIndex = j
        }
        X[i] <-> X[minIndex] (swap in place)
    }
return X
ANALYSIS

- Recall: Input is array \( X \) of size \( n \)
- We refer to \( n \) the **problem size**
- When we analyze the algorithm, we will express the **running time** as a function of \( n \)
  - Running time **must** to be a function of \( n \). Why?
- Recall: **running time** = \# of CPU instructions
ANALYZING THIS ALGORITHM

```java
procedure selectionSort(Array X of size n):
    for i = 1 to n {
        let minIndex = i;
        for j = i+1 to n {
            if X[j] < X[minIndex]
                minIndex = j
        }
        X[i] <-> X[minIndex] (swap in place)
    }
    return X
```

- Covered roughly in a few different ways on the board: \(c \cdot n^2\) steps, where \(c\) is some constant