CS341: Algorithms

Lecture 00: Introduction

Yao-Liang Yu

School of Computer Science
University of Waterloo

January 3, 2017
Plan for Today

1 Course Logistics

2 Course Review

3 First Examples
Plan

1 Course Logistics

2 Course Review

3 First Examples
Course Information

- **Sections & Instructors**
  - Lec 1 & 3: Semih, TR 2:30 - 5:20, MC 2034
  - Lec 2 & 4: Doug, TR 10:00 - 11:20, RCH 207
  - Lec 5: Yao, TR 8:30 - 9:50, MC 4040

- **Scheduled office hours**
  - Semih, ??
  - Doug, R 1:30 - 2:30
  - Yao, T 10:00 - 11:00, DC 3206 or by email appointment
    yaoliang.yu@uwaterloo.ca
Resources

● Website (still under construction)
  https://www.student.cs.uwaterloo.ca/~cs341/
  For syllabus, calendar, policies, etc.

● Piazza (will be invited)
  piazza.com/uwaterloo.ca/winter2017/cs341
  For announcements, questions, discussions, etc.

● Learn https://learn.uwaterloo.ca
  For slides, assignments, solutions, grades, etc.

● Textbook: Introduction to Algorithms (3rd), by Cormen, Leiserson, Rivest and Stein, MIT press, available in bookstore

● TAs: Aayush, Dimitrios, Eric, Jian, Jose, Shayan, Shikha Y-L. Yu (UW)
Please silence your cell phone and other electronic devices before class.

Questions encouraged, but please refrain from talking in class.

Use your laptop only for course related materials.

Bottom line: Do not disturb others.
## Coursework

<table>
<thead>
<tr>
<th></th>
<th>out date</th>
<th>due date</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>Jan 6</td>
<td>Jan 20</td>
<td>5%</td>
</tr>
<tr>
<td>A2</td>
<td>Jan 20</td>
<td>Feb 3</td>
<td>5%</td>
</tr>
<tr>
<td>A3</td>
<td>Feb 3</td>
<td>Feb 17</td>
<td>5%</td>
</tr>
<tr>
<td>A4</td>
<td>Feb 17</td>
<td>Mar 10</td>
<td>5%</td>
</tr>
<tr>
<td>A5</td>
<td>Mar 10</td>
<td>Mar 31</td>
<td>5%</td>
</tr>
</tbody>
</table>

| Midterm  | STC 1012 & AHS 1689 | Mar 2, 7:00 – 8:50pm | 25% |

| Final     | ??         | ??          | 50% |

- All 5 sections have the same coursework and grading
- **No late policy**
- Emergencies considered if formal proof provided
Academic Integrity

- Good discussion online:
  
  http://www.math.uwaterloo.ca/navigation/Current/cheating_policy.shtml

- Ignorance is no excuse!

- Do your work on your own.

- Discussion is fine or even encouraged, but no sharing of text or code.

- Common mistakes
  
  ▶ excessive collaboration with other students
  ▶ use solutions from other sources

- Possible penalties
  
  ▶ first offense: 0% for that assignment, -5% on final grade
  ▶ second offense: expulsion is possible
Plan

1. Course Logistics
2. Course Review
3. First Examples
Prerequisites

- CS 240: standard data structures
- Mathematical maturity
  - proof by induction
  - proof by contradiction
- Preferably, some programming experience
Why is CS 341 important for you?

- Heart of CS
- Wide applications
- Building stones
- Intellectual challenge
- Job hunting
- etc.

*must do-well course for CS students!*
What is an algorithm?

"An algorithm is a finite, definite, effective procedure, with some input and some output."
— Donald Knuth

A procedure that efficiently computes an “answer” to a problem.
Algorithmic problems

- Sorting, e.g. rank the students/websites by their GPA/relevance
- Searching, e.g. retrieve a patient’s record, if exists in database
- Strings, e.g. ctrl/cmd + F
- Graphs, e.g. shortest trip from Waterloo to Toronto
- Geometry, e.g. closest pair among $n$ points on the plane
- Numerical, e.g. approximate $\sqrt{2}$
Algorithm design techniques

- Brute-force
- Divide and conquer
- Greedy
- Dynamic Programming
- Reduction
- Heuristic Search (not in this course)
- Mathematical Programming (not in this course)

Simplicity is an appeal.
Type of algorithms

- serial vs. parallel
- exact vs. approximate
- deterministic vs. probabilistic
- offline vs. online
- etc.

*In this course, we deal with serial, exact, deterministic and offline algorithms (most of the time).*
Algorithm analysis

- Is your algorithm correct?
  “Yes. I ran it xx times and it always output correctly.”

- Is your algorithm efficient, in time, space, communication, etc.?
  “Yes. I ran it...”

- Is there an “optimal” algorithm?
  No in general. For specific measures yes sometimes.

- Finding an optimal (in certain sense) algorithm is a great achievement!

  *We focus on formal proofs.*
Efficiency: zoomed in

- Most of time we analyze running time complexity of algorithm
- Worst-case vs. average case

\[
\max_{I \in \mathcal{I}(P)} T(A, I) \quad \text{vs.} \quad \frac{1}{|\mathcal{I}(P)|} \sum_{I \in \mathcal{I}(P)} T(A, I)
\]

- Upper bound vs. lower bound

Bear in mind that sometimes analysis might not be tight.

- A hierarchy of time complexity
  - linear time \(O(n)\), polynomial time \(O(n^k)\) | exponential time \(O(2^n)\)

- Undecidable problems that cannot be solved by any algorithm...
Plan

1. Course Logistics
2. Course Review
3. First Examples
Sorting

Problem


Sorting: same array of integers in increasing order $\forall i, A[i] \leq A[i+1]$

Algorithm 1: SelectionSort (array $A$ of size $n$)

1. for $i = 1$ to $n$ do
2.     $minIndex = i$
3.     for $j = i+1$ to $n$ do
5.             $minIndex = j$
7. return $A$
Example
Correctness of SelectionSort

On the $i$-th iteration, $\text{SelectionSort}$ selects the $i$-th smallest element $A[\text{minIndex}]$ and swaps it with $A[i]$. 
Efficiency of SelectionSort

Algorithm 1: SelectionSort (array A of size n)

1. for $i = 1$ to $n$ do
2.     $\text{minIndex} = i$
3.     for $j = i + 1$ to $n$ do
4.         if $A[j] < A[\text{minIndex}]$ then
5.             $\text{minIndex} = j$
6.     $A[i] \leftrightarrow A[\text{minIndex}]$ // in-place swap
7. return A

- Outer loop: $n$ iterations
- Inner loop: $n - i$ iterations for the $i$-th outer iteration
- In total: $\sum_{i=1}^{n} (n - i) = n^2 - \frac{n(n+1)}{2}$ comparisons, $n$ swaps
- Complexity $O(n^2)$
How to sort in decreasing order?

reduce to something familiar
negate, sort increasingly, negate
or modify SelectionSort directly

Algorithm 2:

```
SelectionSort (array A of size n)
1 for i = 1 to n do
2 maxIndex = i
3 for j = i + 1 to n do
4 if A[j] > A[maxIndex] then
5 maxIndex = j
7 return A
```
How to sort in decreasing order?

- reduce to something familiar

  negate, sort increasingly, negate

- or modify SelectionSort directly

---

**Algorithm 3: SelectionSort** (array $A$ of size $n$)

1. for $i = 1$ to $n$ do
2.     $\text{maxIndex} = i$
3.     for $j = i + 1$ to $n$ do
4.         if $A[j] > A[\text{maxIndex}]$ then
5.             $\text{maxIndex} = j$
6.         end if
7.     end for
8.     $A[i] \leftrightarrow A[\text{maxIndex}]$  // in-place swap
9. end for
10. return $A$
Nice properties of SelectionSort

- in-place: no extra memory is needed
Nice properties of SelectionSort

- in-place: no extra memory is needed

  All right, we do need $O(1)$ extra memory...

Nice properties of **SelectionSort**

- **in-place**: no extra memory is needed

  \textit{All right, we do need }O(1)\textit{ extra memory...}

- **stable**: if for \(i < j\), \(A[i] = A[j]\) then after **SelectionSort** \(A[i]\) still proceeds \(A[j]\)

  \textit{Suppose students are sorted alphabetically, then after **SelectionSort** by GPA, students with the same GPA are still alphabetically sorted.}