INTRODUCTION

cs2420 | Introduction to Algorithms and Data Structures | Fall 2016
today...
- meet the teaching staff
- what is this course about?
- why should you care?
- nuts & bolts
- good coding practice
meet the teaching staff
born in Martinsville, VA

year 0
born in Martinsville, VA

year 0
dad buys a Commodore64
decide to become an astronaut
decide to become a surgeon

decide to become an astronaut

year 10
decide to become an astronaut

decide to become a surgeon

decide to become a surgeon on a space station

year 10
decide to become an astronaut

year 10

decide to become a surgeon

decide to become a surgeon on a space station

start college at Penn State
finish BS in astronomy

year 20
software engineer at Raytheon

finish BS in astronomy

year 20
discover computer graphics, realize CS is awesome

software engineer at Raytheon

finish BS in astronomy

year 20
start grad school at the U

discover computer graphics, realize CS is awesome

software engineer at Raytheon

finish BS in astronomy

year 20
finish PhD in computer science

year 30
postdoc at Harvard University

finish PhD in computer science

*year 30*
assistant professor at the U in School of Computing and SCI
postdoc at Harvard University
finish PhD in computer science
assistant professor at the U in School of Computing and SCI
postdoc at Harvard University
finish PhD in computer science
Night
Louise Bogan

The cold remote islands
And the blue estuaries
Where what breathes, breathes
The restless wind of the inlets
And what drinks, drinks
The incoming tide:

Where shell and weed
Wait upon the salt wash of the sea
And the clear nights of stars
Swing their lights westward
To set behind the land;

Where the pulse clinging to the rocks
Renews itself forever:
Where, again on cloudless nights,
The water reflects
The firmament's partial setting;

-O remember
In your narrowing dark hours
That more things move
Than blood in the heart.
what is this course about?
fundamentals of coding

- how to analyze your algorithms
  - improve efficiency
  - make good coding choices

- recursion
  *def.* Recursive loop: See “recursive loop”.

- basic sorting algorithms
  - one of most studied operations in CS

- elementary data structures
  - provide mechanism for what we can do with data
why should you care?
You’re traveling to NYC and plan to get around using public transit. Write an algorithm to compute the fastest travel time between any two locations in the city at any time of the day.
You’re traveling to NYC and plan to get around using public transit. Write an algorithm to compute the fastest travel time between any two locations in the city at any time of the day.

Describe an algorithm to efficiently look up a word in a dictionary.
You’re traveling to NYC and plan to get around using public transit. Write an algorithm to compute the fastest travel time between any two locations in the city at any time of the day.

Describe an algorithm to efficiently look up a word in a dictionary.

Given a destination webpage and starting webpage, determine if the destination is reachable via embedded links.
You’re traveling to NYC and plan to get around using public transit. Write an algorithm to compute the fastest travel time between any two locations in the city at any time of the day.

Describe an algorithm to efficiently look up a word in a dictionary.

Given a destination webpage and starting webpage, determine if the destination is reachable via embedded links.

Given a sentence as a string, reverse the characters in each word.
You’re traveling to NYC and plan to get around using public transit. Write an algorithm to compute the fastest travel time between any two locations in the city at any time of the day.

Describe an algorithm to efficiently look up a word in a dictionary.

Given a destination webpage and starting webpage, determine if the destination is reachable via embedded links.

Given a sentence as a string, reverse the characters in each word.
why do(n’t) algorithms matter?

- many different ways to solve a problem
  - one method may take 1ms longer per item....
  - computers operate on LARGE numbers of items
    - millions
    - billions
    - ... or more
why do(n’t) algorithms matter?

- many different ways to solve a problem
  - one method may take 1ms longer per item....
  - computers operate on LARGE numbers of items
    - millions
    - billions
    - ... or more

$1 \times 10^{12} \times \text{(minuscule amount of time)} = \text{large amount of time}$
why do(n’t) algorithms matter?

- many different ways to solve a problem
  - one method may take 1ms longer per item....
  - computers operate on LARGE numbers of items
    - millions
    - billions
    - ... or more

\[ 1 \times 10^{12} \times \text{(minuscule amount of time)} = \text{large amount of time} \]

- this matters, but not as much as algorithmic complexity
-we refer to unspecified integer quantities as \( N \)
  - \( N \) is the problem size
    - sorting an array of \( N \) numbers
    - searching for an item in a set of \( N \) items
    - inserting an item into a set of \( N \) items

- amount of work done for these operations usually depends on \( N \)
  - work required is a function of \( N \)
why DO algorithms matter?

- algorithms don’t always require $N$ steps for $N$ items
  - could be linear, quadratic, logarithmic, …
  - called the complexity of an algorithm

- $N^2$ is much MUCH bigger than $N$
  - what if $N == 1$ million?

- we only care about large $N$
sort1 versus sort2
Choosing an Algorithm

How important is it to pick the best algorithm for the job?

- small problem size

<table>
<thead>
<tr>
<th>N (size of list)</th>
<th>running time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.001</td>
</tr>
<tr>
<td>20</td>
<td>0.002</td>
</tr>
<tr>
<td>30</td>
<td>0.003</td>
</tr>
<tr>
<td>40</td>
<td>0.004</td>
</tr>
<tr>
<td>50</td>
<td>0.005</td>
</tr>
<tr>
<td>60</td>
<td>0.006</td>
</tr>
<tr>
<td>70</td>
<td>0.007</td>
</tr>
<tr>
<td>80</td>
<td>0.008</td>
</tr>
<tr>
<td>90</td>
<td>0.009</td>
</tr>
<tr>
<td>100</td>
<td>0.010</td>
</tr>
</tbody>
</table>

- sort 1
- sort 2

- small N
Choosing an Algorithm

- **medium**

![Graph showing running time vs. N (size of list)]

- **sort 1**
- **sort 2**

*running time (milliseconds)*

N (size of list)
Choosing an Algorithm

As $N$ becomes large, complexity matters!

- Sort 1
- Sort 2
Choosing an Algorithm (large problem size)

As $N$ becomes large, complexity matters!

**Graph:**
- $N$ (size of list)
- Running time (milliseconds)

**Take away:**
as $N$ becomes large, complexity matters!
void sort1(int[] arr) {
    for(int i = 0; i < arr.length-1; i++) {
        int j, minIndex;
        for(j = i+1, minIndex = i; j < arr.length; j++)
            if(arr[j] < arr[minIndex])
                minIndex = j;
        swap(arr, i, minIndex);
    }
}
void sort1(int[] arr) {
    for (int i = 0; i < arr.length-1; i++) {
        int j, minIndex;
        for (j = i+1, minIndex = i; j < arr.length; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        swap(arr, i, minIndex);
    }
}

best case: $O(N^2)$
average case: $O(N^2)$
worst case: $O(N^2)$
```java
void sort1(int[] arr) {
    for(int i = 0; i < arr.length-1; i++) {
        int j, minIndex;
        for(j = i+1, minIndex = i; j < arr.length; j++)
            if(arr[j] < arr[minIndex])
                minIndex = j;
        swap(arr, i, minIndex);
    }
}
```

selection sort

<table>
<thead>
<tr>
<th>best case</th>
<th>average case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
</tr>
</tbody>
</table>

```java
void sort2(int[] arr, int beg, int end) {
    if (end > beg + 1) {
        int piv = arr[beg], l = beg + 1, r = end;
        while (l < r) {
            if (arr[l] <= piv)
                l++;
            else
                swap(arr, l, --r);
        }
        swap(arr, --l, beg);
        sort2(arr, beg, l);
        sort2(arr, r, end);
    }
}
```

<table>
<thead>
<tr>
<th>best case</th>
<th>average case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(N \log N)$</td>
<td>$O(N \log N)$</td>
<td>$O(N^2)$</td>
</tr>
</tbody>
</table>
void sort1(int[] arr) {
    for (int i = 0; i < arr.length-1; i++) {
        int j, minIndex;
        for (j = i+1, minIndex = i; j < arr.length; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        swap(arr, i, minIndex);
    }
}

void sort2(int[] arr, int beg, int end) {
    if (end > beg + 1) {
        int piv = arr[beg], l = beg + 1, r = end;
        while (l < r) {
            if (arr[l] <= piv)
                l++;
            else
                swap(arr, l, --r);
        }
        swap(arr, --l, beg);
        sort2(arr, beg, l);
        sort2(arr, r, end);
    }
}
the difference between 1ms and 30ms doesn’t matter if you run the algorithm once…
Complexity matters

the difference between 1ms and 30ms doesn’t matter if you run the algorithm once...

... but this is rarely the case in computing

~1 billion Google searches per day, every day

~30ms/frame for all algorithms in a game

~5 billion videos streamed on YouTube every day
data structures & algorithms matter

- for large $N$, the difference between $O(N \log N)$ and $O(N^2)$ is HUGE!
data structures & algorithms matter

- for large $N$, the difference between $O(N \log N)$ and $O(N^2)$ is HUGE!

- is running time the only measure of efficiency?
data structures & algorithms matter

-for large $N$, the difference between $O(N \log N)$ and $O(N^2)$ is HUGE!

-is running time the only measure of efficiency?

-transitioning from cs1410 to cs2420

  cs1410: correct algorithms (or just code?) to solve problems

  cs2420: correct algorithms analyzed for efficiency; advanced structures for intuitive organization of data
nuts & bolts
http://www.eng.utah.edu/~cs2420/
- **programming homework**
  - one assignment per week (except final project)
    - *out Wednesdays, due next Wednesday at midnight*
  - approx. half must be done with a partner
  - 1-day late policy, 10% off
  - all programming to be done in Java 8

- **exams**
  - two midterms, held during class time
  - one final

- **labs**
  - will focus on technical issues
  - **required**: they count towards your final grade!
- **programming homework**
  - one assignment per week (except final project)
  - *out Wednesdays, due next Wednesday at midnight*
  - approx. half must be done with a partner
  - 1-day late policy, 10% off
  - all programming to be done in Java 8

- **exams**
  - two midterms, held during class time
  - one final

- **labs**
  - will focus on technical issues
  - **required**: they count towards your final grade!
<table>
<thead>
<tr>
<th>Grade</th>
<th>100-93</th>
<th>92-90</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>89-87</td>
<td>B+</td>
</tr>
<tr>
<td></td>
<td>86-83</td>
<td>B</td>
</tr>
<tr>
<td>A-</td>
<td>82-80</td>
<td>B-</td>
</tr>
<tr>
<td></td>
<td>79-77</td>
<td>C+</td>
</tr>
<tr>
<td></td>
<td>76-73</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>72-70</td>
<td>C-</td>
</tr>
<tr>
<td></td>
<td>69-67</td>
<td>D+</td>
</tr>
<tr>
<td></td>
<td>66-63</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>62-60</td>
<td>D-</td>
</tr>
<tr>
<td></td>
<td>59-0</td>
<td>E</td>
</tr>
</tbody>
</table>
cheating policy*

- cheating is:
  - sharing, retyping, looking at, or supplying a copy
  - “copy” includes your own work from a previous instantiation of this course

- cheating is not:
  - discussing concepts, answering question about concepts, or helping a classmate understand how to use tools and software

- penalty is a failing grade and formal sanctions

- we will run code checkers on all assignments, including a corpus of several years worth of assignments

- use GitHub carefully (or, just don’t use it!)

*I really really really hate having to talk about this!*
- assignments
- grades
- student-to-student discussion forum
- announcements
getting in touch

teach-cs2420@list.eng.utah.edu
TA office hours

Events shown in time zone: Mountain Time

SYLLABUS

PREREQUISITES  Students are expected to have a working knowledge of Java and to be able to complete the tasks required in CS1410.

OBJECTIVES  The following are the expected outcomes for students completing this course:

- Students will learn algorithms for basic searching and sorting, as well as the asymptotic behavior of each.
- Students will become proficient implementing and using basic data structures fundamental to computer science including arrays, linked lists, stacks, queues, graphs, trees, binary search trees, Huffman trees, hash tables, binary heaps, and priority queues. Students will reason about the
tough love slide...
tough love slide...

- this is the gate-keeper course for the CS major
  - getting [A || A-] means likely getting to major status
  - getting [B+ || B] is a gray area
  - getting [<= B-] is a no-go
tough love slide...

- this is the gate-keeper course for the CS major
  - getting [A || A-] means likely getting to major status
  - getting [B+ || B] is a gray area
  - getting [<= B-] is a no-go

-historically...
  - 25% of students get A’s
  - 25% of students get B’s
  - 30% of students drop or fail
good coding practice
the nature of programming

- requires more time than you think
  - more time consuming than 4-credit hours may imply
the nature of programming

- requires more time than you think
  - more time consuming than 4-credit hours may imply

- when is a program done?
  - when it compiles?
the nature of programming

requires more time than you think
  more time consuming than 4-credit hours may imply

when is a program done?
  when it compiles?

can the time required to code and debug a program be reduced?
  YES! by practicing good software engineering
phases of software development

- requirements gathering
  - read and understand assignment specs, ask questions

- planning | design | analysis
  - outline how to solve a problem, determine algorithms, write pseudocode

- construction
  - write code, debug syntactic errors

- testing
  - test thoroughly to find semantic errors and boundary cases

- maintenance
using SE in assignments

- careful planning and coding can save hours of debugging

- learn from your mistakes: anticipate errors
  - misspellings, typos, off-by-one errors

- thorough, organized testing will detect more errors

- pay attention to the way you design, code, debug, test — habits form quickly!
testing

-white-box
  -test with knowledge of the program’s inner-workings — from the programmer’s perspective
    -unit testing, boundary analysis

-black-box
  -test only with knowledge of the program’s interface — from the user’s perspective
    -stress testing

-test-first model
  -write acceptance tests before writing any code
good coding style

- benefits the programmer and all other readers of the program

-components:
  - descriptive names (variables, methods, classes)
  - clear expressions, straightforward control flow
  - consistency, conventions, and language idioms
  - comments!

-well-written code is often smaller, has fewer errors, and is easier to extend and modify
SE in cs2420

- start practicing good coding style for its own rewards, not just credit

- try applying SE to each assignment
  - learn from development process on previous assignments
  - make necessary improvements on future assignments

- cs3500 (Software Practice 1) will cover SE principles more thoroughly
this week...
- **reading**
  - chapters 1 & 2

- **homework**
  - proficiency exam *(do not hand in)*
  - assignment 1 out on Wednesday
  - Ryan here on Wednesday to discuss expectations

- **lab**
  - first lab on Friday