INTRODUCTION
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

dad buys a Commodore64

year 0
decide to become an astronaut

decide to become a surgeon

decide to become a surgeon on a space station

start college at Penn State

year 10
start grad school at the U

discover computer graphics, realize CS is awesome

software engineer at Raytheon

finish BS in astronomy

year 20

interned at the Chicago Tribune
finish PhD in computer science


Particle-based Sampling and Meshing of Multimaterial Volumes. M. Meyer et al., Vis 2008.

Particle Systems for Efficient and Accurate High-Order Finite Element Visualization M. Meyer et al., TVCG 2006.
postdoc at Harvard University
finish PhD in computer science
WE ARE HERE

Assistant professor at the U in School of Computing and SCI

Postdoc at Harvard University

Finish PhD in computer science

Year 30

![Image of a baby waving](image.png)
what *can* we build?  

**technique-driven**

what *should* we build?  

**problem-driven**

target specific domain problems  
close collaboration with domain experts  
rapid, iterative prototyping  
refine visualizations through user feedback
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 saltwash 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?
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 $\mathbf{N}$

- $\mathbf{N}$ is the problem size
  - sorting an array of $\mathbf{N}$ numbers
  - searching for an item in a set of $\mathbf{N}$ items
  - inserting an item into a set of $\mathbf{N}$ items

amount of work done for these operations usually depends on $\mathbf{N}$

- work required is a function of $\mathbf{N}$
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 vs sort2
Choosing an Algorithm

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

---

**small N**

![Graph showing running time vs. list size for sort 1 and sort 2]
Choosing an Algorithm (medium problem size)

running time (miliseconds) vs N (size of list)

- sort 1
- sort 2
Choosing an Algorithm

As $N$ becomes large, complexity matters!

**TAKE AWAY:**

**AS $N$ BECOMES LARGE COMPLEXITY MATTERS!**

---

**large $N$**

---

**Diagram:**

![Graph showing the relationship between $N$ (size of list) and running time (milliseconds). The graph compares two algorithms: sort 1 and sort 2. As $N$ increases, sort 1 shows a significant increase in running time, while sort 2 remains relatively constant. This highlights the importance of considering algorithm complexity when dealing with large inputs.]
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);
    }
}
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

~30ms/frame for all algorithms in a game

~1 billion Google searches per day, every day
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
This course provides an introduction to tools found throughout computer science -- basic algorithms and data structures that lend themselves naturally to computational problem solving, as well as the problem of engineering computational efficiency in to programs. Students will gain an understanding of classical algorithms (including sorting, searching, tree and graph traversal), and data structures (including linked-lists, trees, graphs, hash tables, and heaps), and an introduction to parallel computing. Students will complete extensive programming assignments that require the implementation and testing of these concepts.

**SCHEDULE**

<table>
<thead>
<tr>
<th>WEEK</th>
<th>DATE</th>
<th>TOPIC</th>
<th>EXAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/13 &amp; 1/15</td>
<td>Introduction &amp; Java review</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/20 &amp; 1/22</td>
<td>OO &amp; generic programming</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1/27 &amp; 1/29</td>
<td>Algorithm analysis &amp; data structures</td>
<td>Midterm 1 on Tues</td>
</tr>
<tr>
<td>4</td>
<td>2/3 &amp; 2/5</td>
<td>Basic sorting</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2/10 &amp; 2/12</td>
<td>Recursive sorting</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2/17 &amp; 2/19</td>
<td>Linked lists</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2/24 &amp; 2/26</td>
<td>Stacks &amp; queues</td>
<td></td>
</tr>
</tbody>
</table>
-programming homework
  - one assignment per week
  - must be done with a partner, except this week
  - all programming to be done in Java 7

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

-labs
  - held most Mondays
  - practice with class topics, 1-on-1 help from TA’s
  - required: they count towards your final grade!
- assignments
- grades
- student-to-student discussion forum
- announcements
-help sessions this Friday
  -9:40am | 10:45am | 11:50am
  -MEB 3225

-getting started with Eclipse
-Java refresher
-assignment posted by Thursday night
  -this will not count towards your grade
good coding practice
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)*
  - student survey *(due Thursday, Jan 15th at 5pm)*

-no lab
  - optional help sessions on Friday