Why Do We Need EECS 233?

- We'll be studying fundamental *data structures*, as well as an introduction to *algorithms* that use these data structures to solve common problems correctly and efficiently.

- What is a data structure? A way of organizing on a collection of information or data
  - sequences: lists, stacks, and queues
  - trees and hash tables
  - heaps and priority queues
  - graphs, etc

- What is an algorithm? A method to solve problems
  - A procedure: takes an input and produces results
  - Provides step-by-step "instructions" for solving a problem or accomplishing a task

Data Structures and Algorithms

- Used universally, fundamental to all computer applications
- software in applications
  - Windows, Mac, iPhone/iPad
  - electronic devices: DVD player, TV
- databases
  - credit card transactions
  - medical records
  - map software
- internet:
  - network routing
  - google, yahoo, etc, web search
- computer aided design, simulation, and optimization
  - VLSI, scheduling , planning
  - scientific research, climate simulation,
  - rendering algorithms, games, artificial intelligence, robotics
- and much, much more ...

Example : Searching in a phonebook

- Data structure: phonebook representation
  - an array of 1,000,000 names (names[0..999999]) and an array of corresponding telephone numbers (phones[0..999999])
- Algorithm:
  - the procedure to find the telephone number by name

```
findNumber(person) {
    P = 0
    while (P < 1000000) {
            Compare the P-th person in the array and person
            if (names[P] == person)
                    return phones[P]
            else
                    P++
    }
    return NOT_FOUND
}
```

- How many comparisons are required, on average?

Example cont'd:
Same problem, different data structure

- Different data structure: a sorted array by name alphabetically, names[0..999999], and the corresponding array of numbers, phones[0..999999]

```
findNumber(person) {
    low = 0
    high = 999999

    while (low <= high) {
            P = floor((low + high) / 2)
            if (person == names[P])
                return numbers[P]
        else
                if (person < names[P])
                    high = P – 1
            else
                low = P + 1
        }
        return NOT_FOUND
}
```

- How many comparisons are required, on average?

Example cont'd: So What to Use?

- Much of the class is learning a way of *thinking,* i.e algorithmically
- Is a sorted array is always better?
- We may need to efficiently perform different operations:
  - search for a telephone number
  - insert a new telephone number
  - remove a telephone number
- Some data structures provide better performance than others for a particular application.
- More generally, we'll learn how to characterize and compare the efficiency of different data structures and their associated algorithms to perform different operations (for different applications).
  - Time efficiency: how many steps are needed?
  - Space efficiency: how much storage is needed?

week	lec.	Date	Topic	Notes	Assignments
1	1	Tue, Aug 30	Course Introduction		
	2	Thu, Sep 1	Abstract Data Types	Ch.1	
2	3	Tue, Sep 6	Recursion	Ch.1.3	W1 out
	4	Thu, Sep 8	Algorithm Analysis	Ch.2	
3	5	Tue, Sep 13	Linked Lists	Ch.3.1-2	W1 due; P1 out
	6	Thu, Sep 15	List Operations	Ch.3.3-5	
4	7	Tue, Sep 20	Stacks and Queues	Ch.3.6-7	
	8	Thu, Sep 22	Basics of Trees	Ch.4.1	P1 due; W2 out
5	9	Tue, Sep 27	Types of Trees	Ch.4.2-3	
	10	Thu, Sep 29	AVL Trees	Ch.4.4	W2 due; P2 out
6	11	Tue, Oct 4	More Trees	Ch.4.5-9	
	12	Thu, Oct 6	Heaps and Priority Queues	Ch.6.1-3	
7	13	Tue, Oct 11	Applications of Heaps	Ch.6.4	
		Thu, Oct 13	Review		P2 due; W3 out
8		Tue, Oct 18	*Midterm Exam*		
	14	Thu, Oct 20	Hashing	Ch.5.1-4	
9		Tue, Oct 25	*Fall break - no class*		

9		Tue, Oct 25	*Fall break - no class*		
	15	Thu, Oct 27	Hashing implementation 1	Ch.5	
10	16	Tue, Nov 1	Hashing Implementation 2	Ch.5	W3 due; P3 out
	17	Thu, Nov 3	Simple Sorting Algorithms	Ch.7.1-4	
11	18	Tue, Nov 8	Bubble Sort and Quick Sort	Ch.7.7	
	19	Thu, Nov 10	Merge Sort	Ch.7.6	
12	20	Tue, Nov 15	Bucket Sort and External Sort	Ch.7.9-10	P3 due
	21	Thu, Nov 17	Graphs	Ch.9.1	W4 out
14	22	Tue, Nov 22	Graph Traversal, DFS, BFS	Ch.9.1-3	P4 out
		Thu, Nov 24	*Thanksgiving holiday - no class*		
15	23	Tue, Nov 29	Shortest Path Problem	Ch.9.3	
	24	Thu, Dec 1	Minimum Spanning Trees	Ch.9.5	
16	25	Tue, Dec 6	Advanced Topics		W4 due
		Thu, Dec 8	Review		P4 due
		TBA	*Final Exam*		

9		Tue, Oct 25	*Fall break - no class*		
	15	Thu, Oct 27	Hashing implementation 1	Ch.5	
10	16	Tue, Nov 1	Hashing Implementation 2	Ch.5	W3 due; P3 out
	17	Thu, Nov 3	Simple Sorting Algorithms	Ch.7.1-4	
11	18	Tue, Nov 8	Bubble Sort and Quick Sort	Ch.7.7	
	19	Thu, Nov 10	Merge Sort	Ch.7.6	
12	20	Tue, Nov 15	Bucket Sort and External Sort	Ch.7.9-10	P3 due
	21	Thu, Nov 17	Graphs	Ch.9.1	**W4 out**
14	22	Tue, Nov 22	Graph Traversal, DFS, BFS	Ch.9.1-3	**P4 out**
		Thu, Nov 24	*Thanksgiving holiday - no class*		
15	23	Tue, Nov 29	Shortest Path Problem	Ch.9.3	
	24	Thu, Dec 1	Minimum Spanning Trees	Ch.9.5	
16	25	Tue, Dec 6	Advanced Topics		**W4 due**
		Thu, Dec 8	Review		**P4 due**
		TBA	*Final Exam*		

What is a good programmer?

- Understands problem and how to solve it
  - knows effective solutions to common problems
  - has knowledge of a wide range of tools and how to use them
  - can find new techniques and teach themselves about them
- Has sense of elegance: chooses best among many solutions
- Has a sense of style: sensible names and organization
- Implements efficiently and correctly:
  - thinks through design and implementation before coding
  - breaks down problem into subproblems
  - tests every line thoroughly and efficiently
  - prioritize among competing goals, has a sense of the big picture and the overall goals
- communicates effectively what the program does, writes maintainable code
- comprehension: can read, understand, and debug other code

The practice of programming

- Style
- Algorithms and Data Structures
- Design and Implementation
- Debugging
- Testing
- Performance
- Portability