CS 4/56101
Design & Analysis of Algorithms

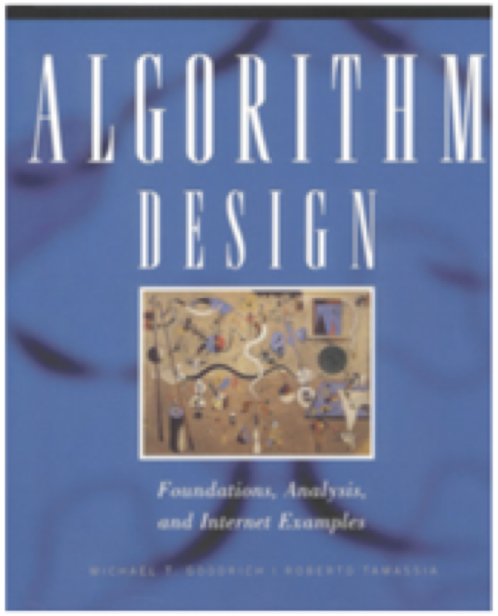
- **Course Website**:
  - http://www.cs.kent.edu/~hmichaud/daa-f18/

- **Instructor**: Heather M. Guarnera
  - Office: MSB 352
  - Email: hmichaud@kent.edu (**Piazza is better**)
  - Office Hours: TR 2:30–3:30, or by appointment

Books

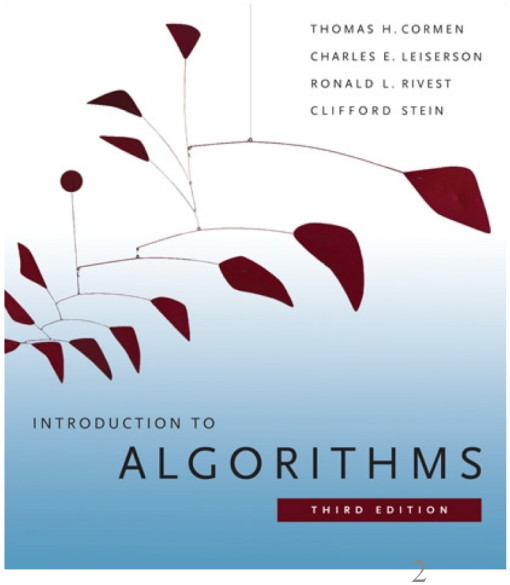
- Textbook:

Algorithm Design: Foundations, Analysis, and Internet Examples, by Michael T. Goodrich and Roberto Tamassia, 1st edition, Wiley, 2001



- An excellent reference:

Introduction to Algorithms, 3rd Edition, by T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, MIT, 2009.



Academic Presence Verification

- Due to federal rules, instructors “must verify that students begin attendance in each course for which they are registered.”
- Required to receive federal financial aid.
- Attendance sheet

Course Requirements

- Homework 40%
  - Good preparation for exams
  - Homework is weighted based on different problems

- Exams
  - Midterm 30% Oct. ??, during class
  - Final 30% Wed Dec. 12, 12:45-3:00pm

- Exam Instructions:
  - Closed book
  - One handwritten sheet (*one side*) allowed

Example: Boss assigns a task

- Given today's prices of pork, grain, sawdust, etc…
- Given constraints on what constitutes a hotdog.
- Make the cheapest hotdog.

Every industry asks these questions.

- Mundane programmer: "Um? Tell me what to code."
- Better: "I learned an algorithm that will work."
- Best: "I can develop an algorithm for you."

How to do this?

Design & Analysis of Algorithms

- Advanced data structures and their analysis
  - Time/space complexity for data structure operations
- Up to date grasp of fundamental problems and solutions
  - How to evaluate algorithms (correctness, complexity)
- Principles and techniques to solve the vast array of unfamiliar problems that arise in a rapidly changing field
  - Notations and abstractions for describing algorithms
  - Approaches to solve
- To think algorithmically like a ‘real’ computer scientist

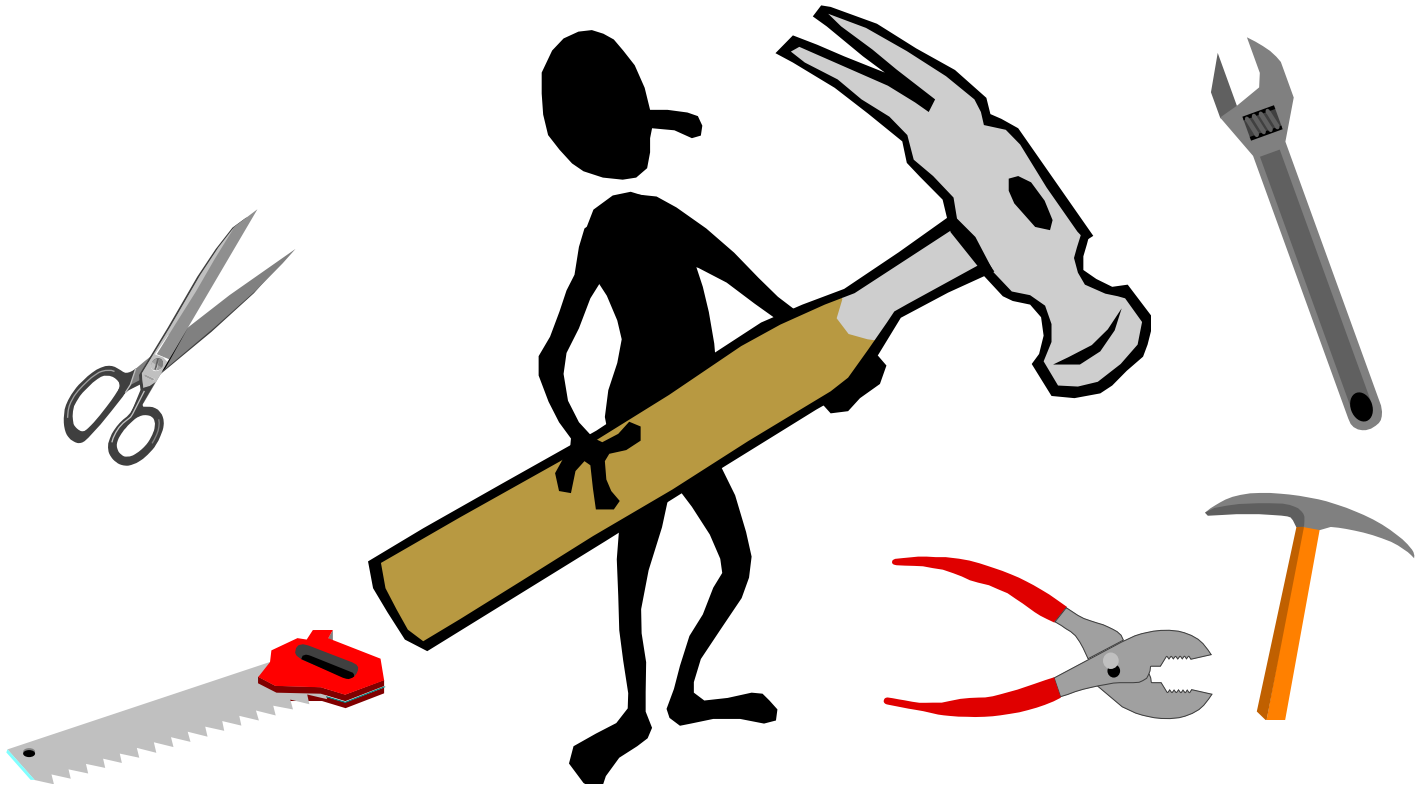
Course Content

- A list of algorithms
  - Learn the code
  - Trace them until you are convinced that they work
  - Implement them.

```
class InsertionSortAlgorithm extends SortAlgorithm
{
   void sort(int a[]) throws Exception {
        for (int i = 1; i < a.length; i++) {
            int j = i;
            int B = a[i];
            while ((j > 0) && (a[j-1] > B)) {
                        a[j] = a[j-1];
                        j--;  }
            a[j] = B;
              }}
```

Course Content

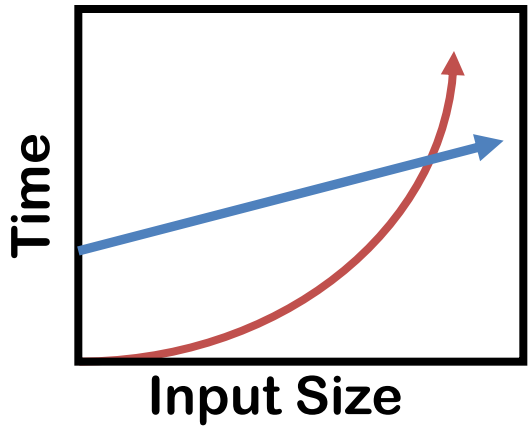
- A survey of algorithmic design techniques
- Abstract thinking
- How to develop new algorithms for any problem that may arise



Start With Some Math

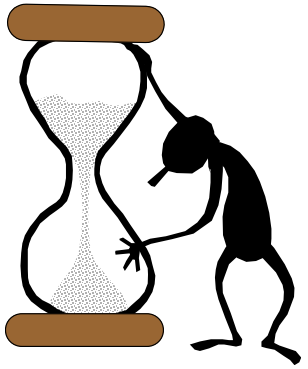
Classifying Functions

$f(i) = n^{\Theta(n)}$



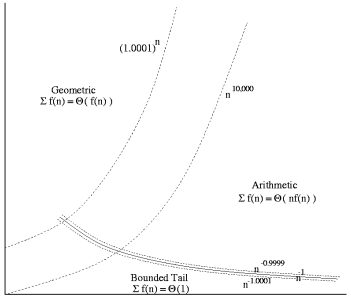
Time Complexity

$t(n) = \Theta(n^2)$



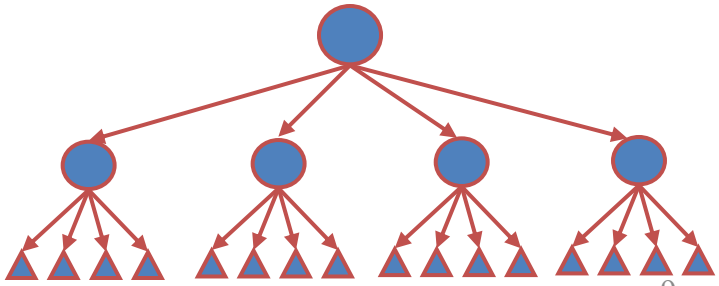
Adding Made Easy

$\sum_{i=1} f(i).$

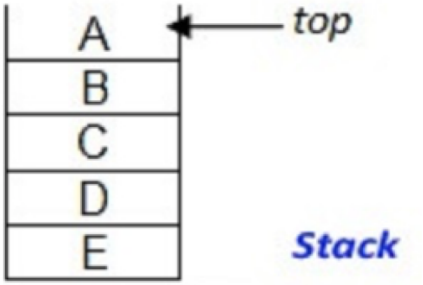


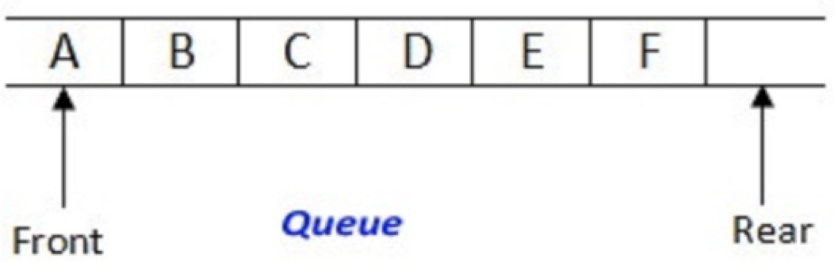
Recurrence Relations

$T(n) = a\ T(n/b) + f(n)$

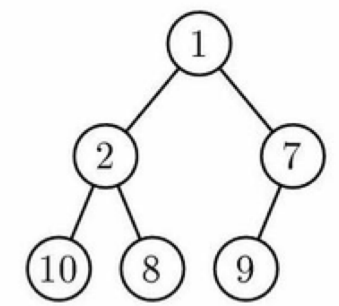


Data Structures

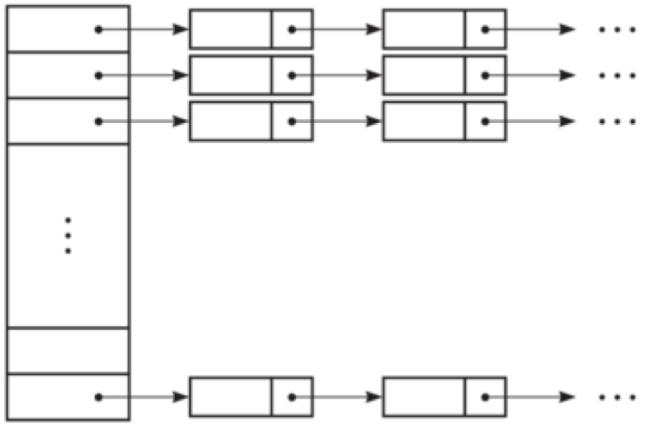




Trees & Heaps

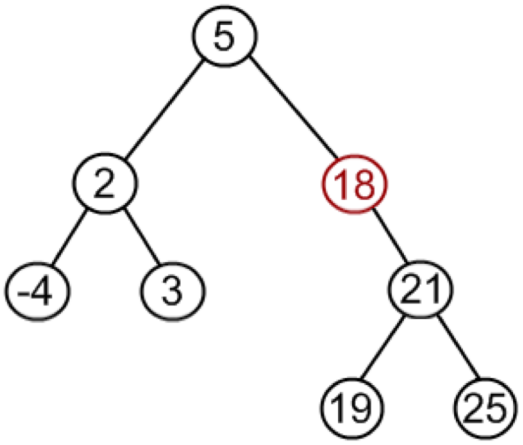


Hash Tables & Dictionaries

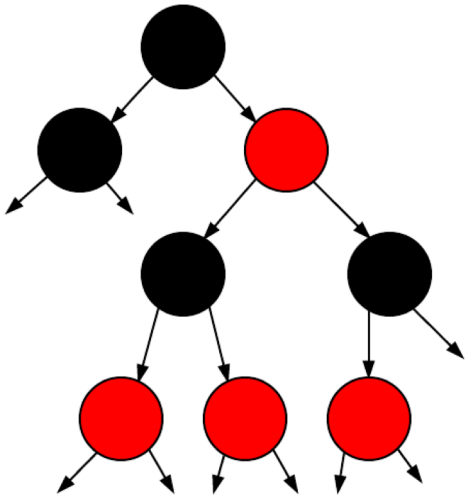


Searching & Sorting

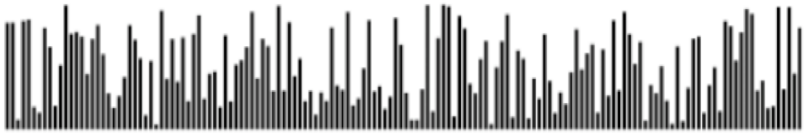
Binary Search Tree

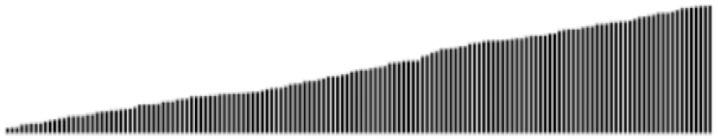


Red Black Trees



Sorting

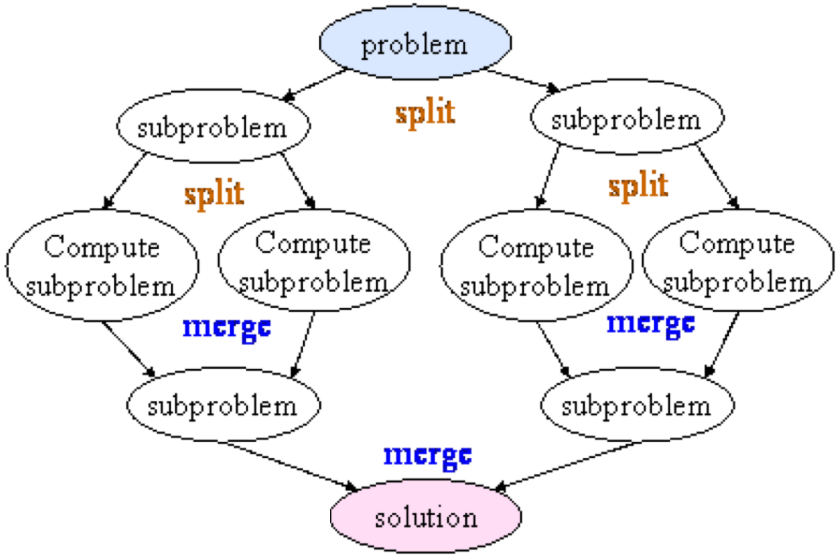




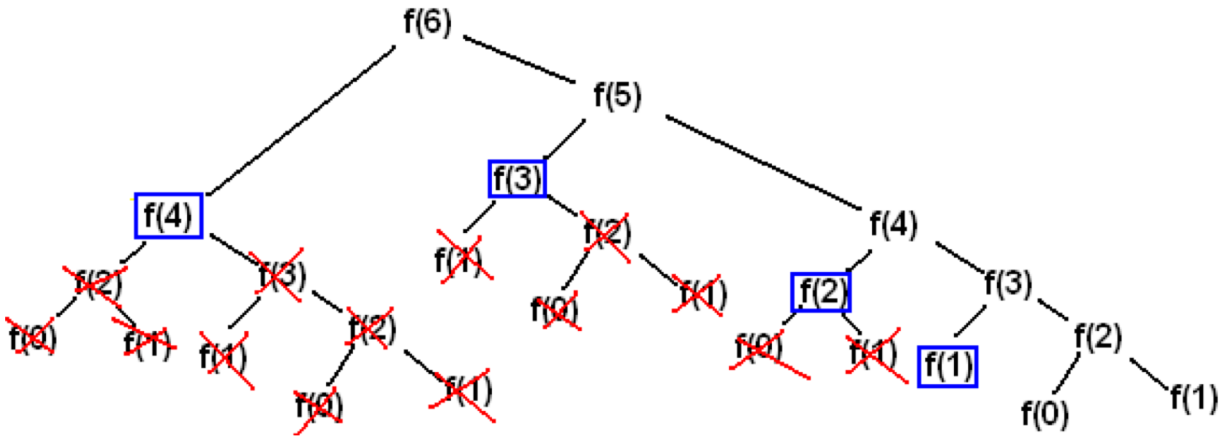
Fundamental Techniques

Greedy Algorithms

Divide and Conquer



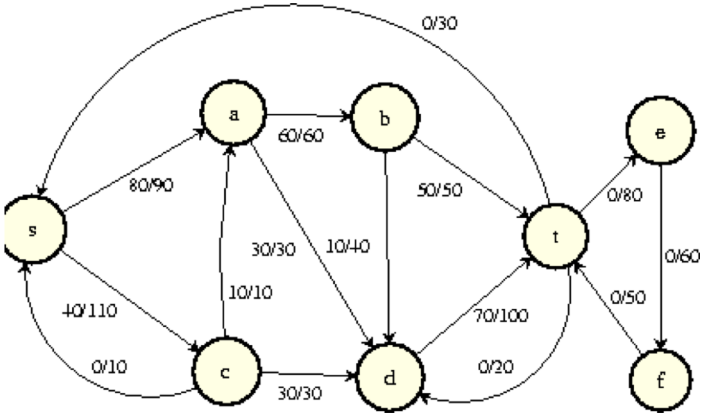
Dynamic Programming



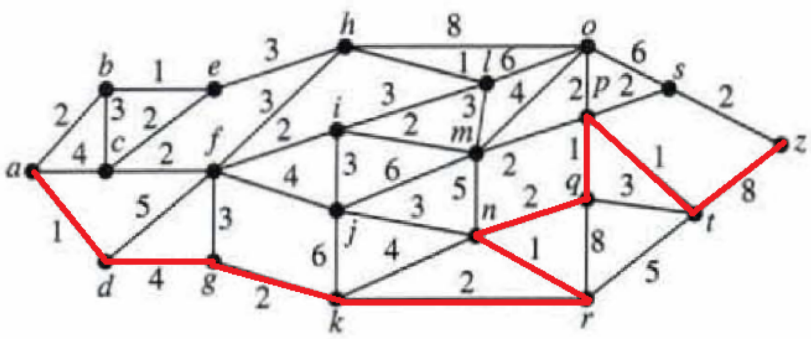
Graph Algorithms

Graph Search

Network Flows



Shortest Path



Text Processing

Pattern Matching

T	H	I	S		I	S		A		S	I	M	P	L	E		E	X	A	M	P	L	E

S	I	M	P	L	E																		
	S	I	M	P	L	E																	
		S	I	M	P	L	E																
			S	I	M	P	L	E															
				S	I	M	P	L	E														
					S	I	M	P	L	E													
						S	I	M	P	L	E												
							S	I	M	P	L	E											
								S	I	M	P	L	E										
									S	I	M	P	L	E									
										S	I	M	P	L	E								

Useful Learning Techniques

- You are expected to read ahead (before the lecture)
  - This will facilitate more productive discussion during class
- Practice explaining
  - You'll be tested on your ability to explain material
- Ask questions
  - Why is it done this way and not that way?
- Guess at potential algorithms for solving a problem
  - Look for input instances where your algorithm is wrong