CS 483

Jana Kosecka

CS Dept.

4444 Eng. Building

kosecka@gmu.edu

Course Info

Course webpage:

- from the syllabus on

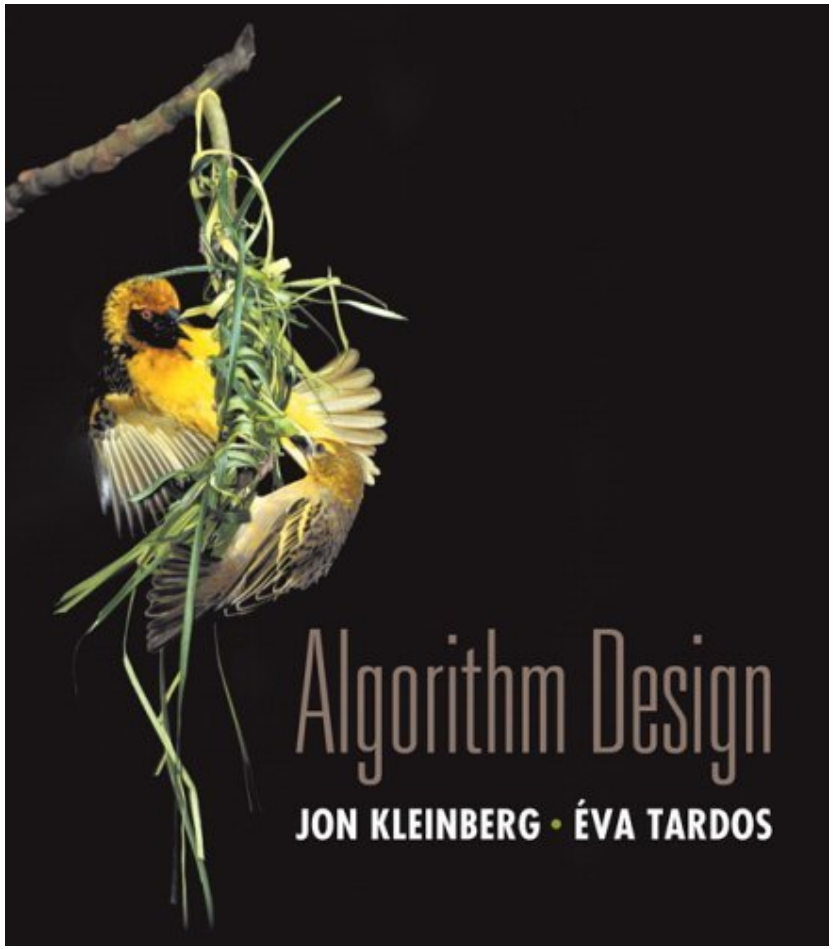
  http://cs.gmu.edu/courses/

Information you will find

- course syllabus, time table
- office hours
- pdf copies of the lectures
- handouts, practice problems

Prerequisites

- Data structures and algorithms (CS 310)
- Formal methods and models (CS 330)
- Calculus (MATH 113, 114, 213)
- Discrete math (MATH 125)
- Ability to program in a high-level language that supports recursion



Grades

- Short Quizes every 2 weeks (40%)
- Practice Problems
- Midterm Exam 30%
- Final Exam 30%
- Make-up tests will **NOT** be given for missed examinations

Other Important Info

Email

- make sure your gmu mail is activated
- send only from your gmu account; mails might be filtered if you send from other accounts
- when you send emails, put [CS483] in your subject header

Goal of the Course

- Design efficient algorithms and analyze their complexity

- **Analysis:**
- what are the computational resources needed ?
- time, storage, #processors, programs, communications

- **What is an algorithm:**

- Recipe to solve a problem
- Clear specification of the problem
- What is the input ? What is the output ?
- How long does it take, under particular circumstances ?
- What are the memory requirements ? (space)

Examples of algorithms

- sorting algorithms – everywhere
- routing, graph theoretic algorithms
- number theoretic algorithms, cryptography
- web search
- triangulation- graphics, optimization problems
- string matching (computational biology),
- cryptography – security

Shortest Paths

Given a graph, find the shortest path in the graph connecting the start and goal vertices.

What is a graph?

How do you represent the graph?

How do you formalize the problem?

How do you solve the problem?

Shortest Paths

What is the most naive way to solve the shortest path problem?

- EX: a graph with only 4 nodes
- How much time does your method take?
- Can we do better?
- How do we know our method is optimal? (i.e., no other methods can be more efficient.)

Shortest Paths

Given a graph, find the shortest path in the graph that visits each vertex exactly once.

- How do you formalize the problem?
- How do you solve the problem?
- How much time does your method take?
- Can we do better?

Hard Problems

We are able to solve many problems, but there are many other problems that we cannot solve efficiently

- we can solve the shortest path between two vertices efficiently
- but we cannot efficiently solve the shortest path problem that requires that path to visit each vertex exactly once

Course Topics

Week 1: Algorithm Analysis (growth of functions)
Week 2: Analysis
Week 3: Graph Algorithms
Week 4: Greedy Algorithms
Week 5: Divide and Conquer
Week 6: Dynamic Programming
Week 7: Max Flow
Week 8: Approximation Algorithms
Week 9: Local Search
Week 10: Randomized Algorithms
Week 11: NP completeness

See updates on the course webpage

Warning & Suggestions

Please don't take this class if you

- You do not have the mathematics and/or CS prerequisites
- You are not able to make arrangements to come to GMU to take the exams on-site
- You are working full-time and taking another graduate level computer science class
- You are not able to spend a minimum of 9~12 hours a week outside of class reading the material and doing practice problem sets

Sorting

Problem: Sort real numbers in **nondecreasing** order

Input: A sequence of n numbers $\langle a_1, \ldots, a_n \rangle$

Output: A permutation $\langle a_1', \ldots, a_n' \rangle$ s.t. $a_1' \leq a_2' \leq \ldots \leq a_n'$

Why do we need to sort?

Sorting

Sorting is important, so
there are **many** sorting algorithms

Selection sort
Insertion sort
Library sort
Shell sort
Gnome sort
Bubble sort
Comb sort
Binary tree sort
Topological sort

Sorting

Algorithms in general

We will be concerned with efficiency

Memory requirements

Independent of the computer speed

How to design algorithms

What is the easiest (or most naive) way to do sorting?

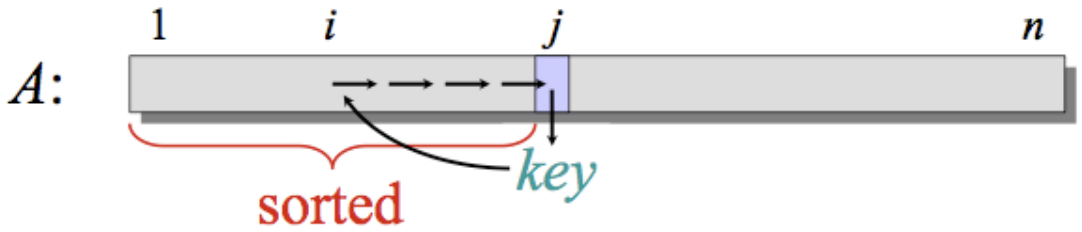
- **EX**: sort 3,1,2,4
- how efficient is your method?
- We will look at two sorting algorithms

Insertion Sort

If you ever sorted a deck of cards, you have done insertion sort

If you don't remember, this is how you sort the cards:

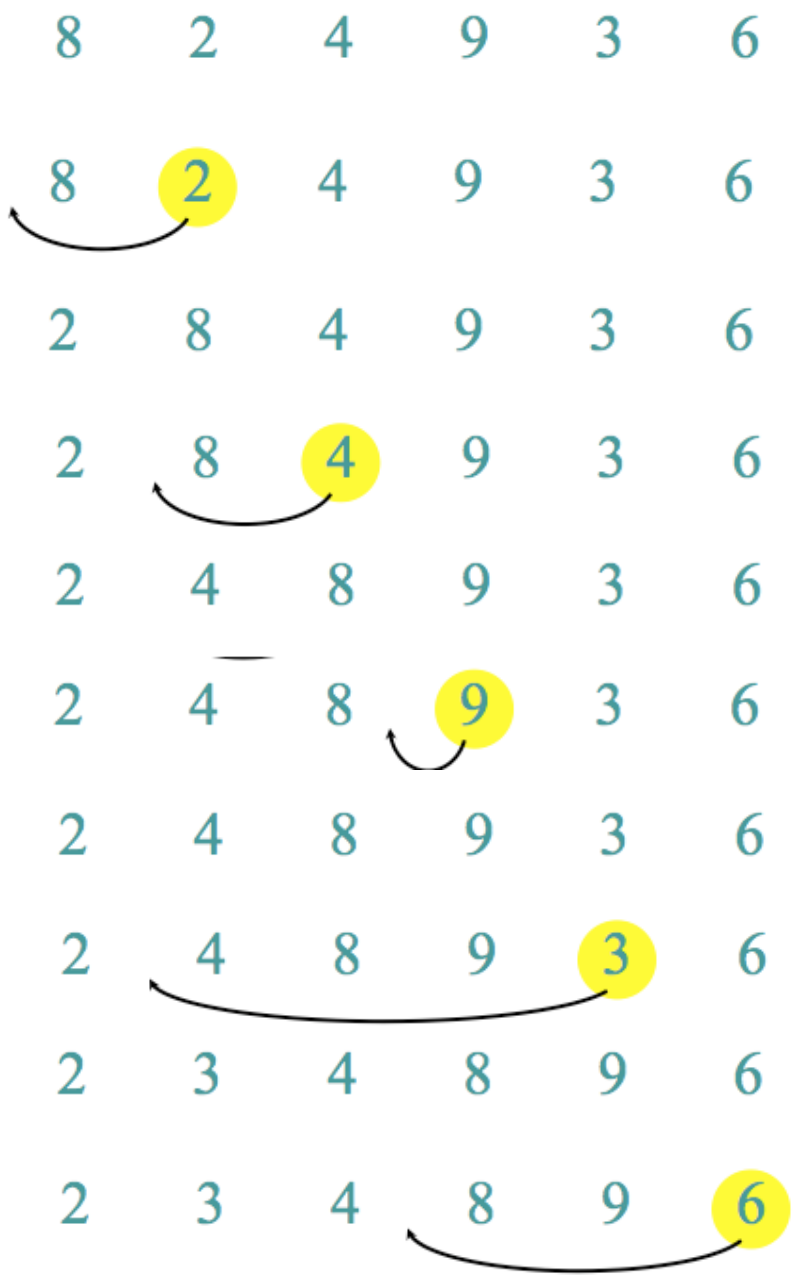
- you sort the card one by one
- assuming the first i cards are sorted, now "sort" the $(i+1)$-th card



Insertion Sort

```
1: for j ← 2 to n do
2:     Temp ← A[j]
3:     i ← j − 1
4:     while i > 0 and A[i] > Temp do
5:         A[i + 1] ← A[i]
6:         i ← i − 1
7:     end while
8:     A[i + 1] ← Temp
9: end for
```

- **EX**: 4, 6, 1, 3, 7, 9, 2


8 2 4 9 3 6
8 2 4 9 3 6
2 8 4 9 3 6
2 8 4 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 4 8 9 3 6
2 3 4 8 9 6
2 3 4 8 9 6

Analyze Insertion Sort

- Is it correct?
- How efficient/slow is insertion sort?
- Characterize running time as a function of input size
- Compute running time of each statement
- Sum up the running times

Insertion Sort

	Cost	times
1: **for** $j \leftarrow 2$ to n **do**	c_1	n
2: Temp $\leftarrow A[j]$	c_2	n-1
3: $i \leftarrow j - 1$	c_4	n-1
4: **while** $i > 0$ and $A[i] >$ Temp **do**	c_5	$\sum_{j=2}^{n} t_j$
5: $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
6: $i \leftarrow i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
7: **end while**		
8: $A[i+1] \leftarrow$ Temp	c_8	n-1
9: **end for**		

- **EX**: 4, 6, 1, 3, 7, 9, 2

Running time – constant amount of time is needed to execute each line, we need to count how many times each line will be executed

t_j- number of times test in line 5 is executed for that value of j

For while and for loop the test is usually executed one more time then the loop body

Insertion Sort

Analysis

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \quad \textit{(n-1)}$$

Best case

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

Worst case - input in the reverse order

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j + c_5 \sum_{j=2}^{n} (j - 1) + c_6 \sum_{j=2}^{n} (j - 1) + c_7 \quad \textit{(n-1)}$$

$$T(n) = an^2 + bn + c$$

Algorithm analysis

Running time

depends on the size of the input (10 vs 100000)

On the type of the input (sorted, partially sorted)

Independent Speed of the computer

Kinds of analysis:

Worst Case analysis max time on any input

Average Case T(n) = average time over all inputs of size n assuming some distribution

Algorithm analysis

Use **pseudocode** description in the language independent way use proper indentation

Analysis of the running time of the algorithm:

How much time does it take ?

(Cost per operation * number of operations)

Choosing the basic operations and how long they take

Too detailed, constant factors do not matter

Machine independent time, Assymptotic Analysis

We would like to ignore machine dependent constants

characterize the running time T(n) as $n \to \infty$ Θ