CS583 Lecture 01

Jyh-Ming Lien

some materials here are based on Prof. Shehu, and Prof. Wang's past lecture notes

Course Info

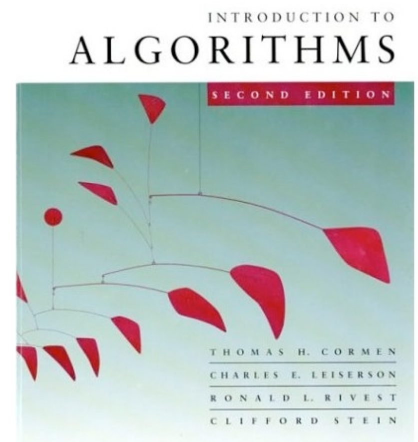
- course webpage:
  - from the syllabus on http://cs.gmu.edu/
  - http://cs.gmu.edu/~jmlien/teaching/09_spring_cs583/
- Information you will find
  - course syllabus
  - time table
  - problem sets
  - pdf copies of the lectures
  - office hours

Prerequisite

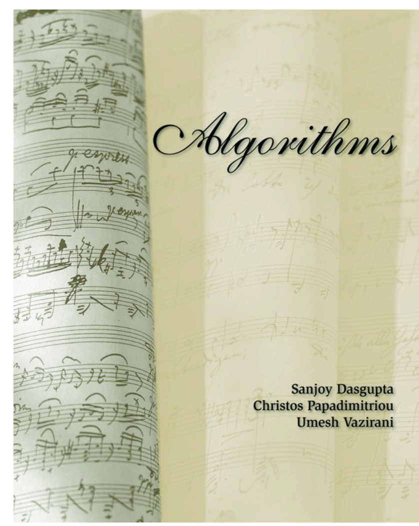
- Data structures and algorithms (CS 310)
- Formal methods and models (CS 330)
- Calculus (MATH 113, 114, 213)
- Discrete math (MATH 125)
- Ability to program in a high-level language that supports recursion

Textbook

- **Introduction to Algorithms** by T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, The McGraw-Hill Companies, 2nd Edition (2001)



- I also recommend you read the following book: **Algorithms**, by Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani, McGraw-Hill, 2006



Grades

- Quizzes (every week) 30%
- Programming Assignment 10%
- Midterm Exam (March 18) 30%
- Final Exam (May 6) 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 [CS583] in your subject header

OK, lets start

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
- Flash sort
- Bucket sort
- Radix sort
- Counting sort
- Pigeonhole sort
- Quicksort
- Heap sort
- Smooth sort
- ... many more

Sorting

- What is the easiest (or most naive) way to do sorting?
  - **EX**: sort 3,1,2,4
  - how efficient is your method?

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

Insertion Sort

for $j \leftarrow 2$ to n **do**
  $\text{Temp} \leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ and $A[i] > \text{Temp}$ **do**
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i - 1$
  end while
  $A[i+1] \leftarrow \text{Temp}$
end for

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

Analyze Insertion Sort

- Is it correct?
- What are the properties of insertion sort
  - stable? in-place? on-line?
- How efficient/slow is insertion sort?

Merge Sort

- how to sort one number quickly?
- how to sort two numbers quickly?
- how to sort three numbers quickly?
- can you generalize this to n numbers?

Merge Sort

if $p < r$ **then**
    $q \leftarrow (p + r)/2$
    Mergesort(A, p, q)
    Mergesort$(A, q + 1, r)$
    Merge(A, p, q, r)
end if

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

Analyze Merge Sort

- Is it correct?
- What are the properties of merge sort
  - stable? in-place? on-line?
- How efficient/slow is merge sort?

Insertion vs. Merge sort

- Which algorithm would you prefer and why?
- Which one is faster? by how much?
- Which one requires more space? by how much?

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

- Jan 28: Algorithm Analysis (growth of functions, recurrence, randomized analysis)
- Feb 04: Sorting & Order Statistics
- Feb 11: Dynamic Programming
- Feb 18: Greedy Algorithms
- Feb 25: Graph Algorithms (basic graph search, topological sort, ...)
- Mar 04: Minimum Spanning Tree
- Mar 25: Single-Source Shortest Paths
- Apr 01: All-Pairs Shortest Paths
- Apr 08: Maximum Flow
- Apr 15: Linear Programming
- Apr 22: NP completeness

- **See details and updates on the course webpage**

Warning

- Please don't take this class if you
  - You do not have the mathematics 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

Suggestions

- **Don’t fall behind** - maintain a steady effort
- **Take the homework (quizzes, practice problems) seriously** - these are the only ways to exercise for the exams
- **Make use of office hours** - we are here to help, but it will be more helpful if you can think about the questions in advance
- **Read the materials before the class** and ask during the class- this prepares you for the quizzes
- **Form study groups** - things become easier if there is joint force