Outline

Introduction

Course Outline and Administrivia

Divide and Conquer
Purpose and Goals of the Course

Design and mathematically analyze efficient algorithms:

▶ An “Algorithm” is any step-by-step procedure or method for solving a problem where each step is simple and unambiguous.

▶ “Efficiency” isn’t measured in seconds but in how many basic steps it takes and how this scales when the size of the problem grows.

▶ “Mathematically analyze” means proving algorithm satisfies certain guarantees, e.g., always terminate within a certain number of steps, always returns correct answer... The course has a lot of math.

Goals of course:

▶ Learn some specific algorithms and specific techniques.

▶ Learn the skill of designing algorithms, i.e., applying general principles and some creativity to solve new problems and analyze new algorithms that you have not seen before.
Is this course going to be hard?

- **Good News:** There’ll be no big project or programming assignments. There’s a lot of math but you can collaborate on the homework.

- **Bad News:** The homework will be designed to make you think and you’ll get stuck sometimes. Best way to learn the skill of algorithm design is to try solving algorithmic problems that challenge you.

- **Math Background:** You’re expected to be able to write rigorous mathematical proofs. May need to brush up on basic math: probability, complex numbers, linear algebra, induction, etc.

- **Algorithms Background:** Official pre-requisite is an undergraduate algorithms class. We’ll go through some undergrad topics but very quickly. Students have taken the class without this but a) they typically had a solid mathematics background and b) spent considerably more time outside the class understanding the material.
Abstraction: Makes it easier to study “efficiency”

- Making an algorithm fast involves many considerations that are architecture specific. For the sake of simplicity and generality, ignore them! We will assume that any location in memory can be accessed a unit cost and measure running time in “basic steps” such as pairwise arithmetic operation and memory accesses.

- Don’t count steps exactly: Consider basic steps $T(n)$ asymptotically as the size of the problem $n$ grows. If, for some function $g(n)$ there exists constants $c, n_0 \geq 0$ such that

$$T(n) \leq cg(n) \text{ for all } n > n_0$$

then we say “$T(n)$ is order $g(n)$” and write $T(n) = O(g(n))$. E.g.,

$$10n^3 = O(n^3) \quad 5n^2 + n + 1000 = O(n^2) \quad n^2 = O(n^5)$$

$$\log_{10} n = O(\log_2 n) \quad (\log n)^{100} = O(n) \quad n^{10000} = O(e^n)$$

- More notation: $T(n) = \Omega(g(n))$ if there exists $c, n_0 > 0$ such that

$$T(n) \geq cg(n) \text{ for } n \text{ larger than } n_0.$$

If $T(n) = O(g(n))$ and $T(n) = \Omega(g(n))$, we write $T(n) = \Theta(g(n))$. 
Outline

Introduction

Course Outline and Administrivia

Divide and Conquer
Basic Stuff

Lectures: Tuesday and Thursday, 11:30 to 12:45 pm via Zoom. Lectures will be recorded and available via Zoom.

Lecturer: Andrew McGregor
  ▶ Email: mcgregor@cs.umass.edu
  ▶ Office hours: Wednesday 9am.

TA: Raghav Addanki.
  ▶ Email: raddanki@cs.umass.edu
  ▶ Office hours: Monday 9am and Thursday 5pm

TA: Md Abdul Aowal.
  ▶ Email: aowal@cs.umass.edu
  ▶ Office hours: Tuesday 2pm and Friday 9:30am

For the quickest response, it’s almost always best to reach us via Piazza, the class forum. Links for office hours are available via Moodle.
Textbooks and Materials

Essential:


Useful Background:

▶ Cormen, Leiserson, Rivest, and Stein. Introduction to Algorithms
▶ Kleinberg and Tardos. Algorithm Design
▶ Dasgupta, Papadimitriou, Vazirani. Algorithms

Specific Topics in More Detail:

▶ Motwani and Raghavan. Randomized Algorithms
▶ Mitzenmacher and Upfal. Probability and Computing
▶ Vazirani. Approximation Algorithms

Websites:

▶ Slides: people.cs.umass.edu/~mcgregor/courses/CS611F20
▶ Quiz: moodle.umass.edu (everyone enrolled already has access)
▶ Homework: gradescope.com (will set up before first deadline)
▶ Discussion: Link is in Moodle. Please join today!
Course Outline

- Preliminaries, Divide and Conquer, FFT (3 lectures)
- Matroids and Greedy Algorithms (4 lectures)
- Dynamic Programming, Shortest Paths, Network Flow (4 lectures)
- Randomized Algorithms (4 lectures)
- Approximation Algorithms for NP-Hard Problems (7 lectures)
- Linear Programming (3 lectures)

Hopefully we'll have time to incorporate some additional material beyond the book, e.g., hashing and streaming, multiplicative weights method, clustering, . . .
Assessment

- **Homework:** 5 or 6 assignments contribute 30% to grade. Collaboration allowed in groups of at most four. You are only allowed to refer to slides and the textbooks; no searching on the web or discussing with anyone outside your group.

- **Quizzes:** Online quizzes contribute 20% to grade. No collaboration.

- **Exams:** There will be one midterm and a final exam. Together exams contribute 40% to grade. No collaboration.

  - Midterm: Date TBA
  - Final: After Thanksgiving.

- **Participation:** 10% of grade will be based on forum participation, i.e., asking good questions and helping other students.

- Cheating results in an F for the course. Email for clarification if anything isn’t clear. All online discussion about the course should

- **Late policy:** No credit will be given for late quizzes or homework. However, we’ll drop everyone’s weakest quiz and homework.
Outline

Introduction

Course Outline and Administrivia

Divide and Conquer
Merge Sort

**Problem:** Given an unsorted list of \( n \) numbers, sort them!

**Algorithm**

1. *Divide list two halves.*
2. *Sort each half.*
3. *Merge the sorted halves.*

Let running time of algorithm be \( T(n) \). Observe that for some constant \( c \), \( T(1) \leq c \) and

\[
T(n) \leq 2T(n/2) + cn
\]
More generally we can consider splitting a problem of size $n$ into $a$ subproblems of size $n/b$.

**Theorem**

Suppose $T(1) \leq c$ and $T(n) \leq aT(n/b) + cn^\alpha$ for $n > 1$ where $a$, $b$, $c$, $\alpha$ are some constants. Then

$$T(n) = \begin{cases} O(n^\alpha) & \text{if } a < b^\alpha \\ O(n^{\log_b a}) & \text{if } a > b^\alpha \\ O(n^\alpha \log n) & \text{if } a = b^\alpha \end{cases}$$

Therefore, Merge-Sort takes $O(n \log n)$ time since $a = 2$, $b = 2$, $\alpha = 1$. 
Proof

- Assume $n$ is a power of $b$ but theorem holds in general.
- Let $W(n) = cn^\alpha$ and repeatedly expand $T(n)$ to get

  \[
  T(n) \leq aT(n/b) + W(n) \\
  \leq a^2 T(n/b^2) + aW(n/b) + W(n) \\
  \leq a^3 T(n/b^3) + a^2 W(n/b^2) + aW(n/b) + W(n) \\
  \leq \ldots \\
  \leq a^{\log_b n} T(1) + a^{\log_b n-1} W(n/b^{\log_b n-1}) + \ldots + aW(n/b) + W(n) \\
  \leq a^{\log_b n} W(n/b^{\log_b n}) + a^{\log_b n-1} W(n/b^{\log_b n-1}) + \ldots + aW(n/b) + W(n) \\
  = cn^\alpha (r^{\log_b n} + r^{\log_b n-1} + \ldots + r + 1) \quad \text{where } r = a/b^\alpha
  \]

- If $r = 1$ and $T(n) \leq cn^\alpha (1 + \log_b n) = O(n^\alpha \log n)$
- If $r < 1$ and

  \[
  T(n) \leq cn^\alpha \left(\frac{1 - r^{1+\log_b n}}{1 - r}\right) = O(n^\alpha)
  \]

- If $r > 1$, then

  \[
  T(n) \leq cn^\alpha \left(\frac{r^{1+\log_b n} - 1}{r - 1}\right) = O(n^\alpha r^{\log_b n}) = O(n^{\log_b a})
  \]