

CMSC 451: Design and Analysis of Computer Algorithms

Sorelle Friedler

Who Are You? Why are you Here?

Calendar / Course Overview

- Homework 20%
- Group Project 20%
- Midterm 25%
- Final 35%

- Plan and work in advance!

Academic Integrity

- Lots of group work - don't work outside your group

- Cite everything!

Rules and Reminders

- Quiet cell phones
- Be on time
- Come to class and be an active participant
- Laptops in class only if really needed
- Read the book
- Stay organized and ahead of your work

Getting Help

- Get help as soon as you need it (and not when you don't)
- Office hours: MTuWThF 11 – 12 in the TA room
- Email: sorelle@cs.umd.edu
 - I will usually respond within 24 hours
- Slides will be posted online

Preview

What we'll be doing:

- Algorithmic analysis / Complexity
- Algorithmic techniques by type

Why we care:

- Useful
- Interesting

Computer Scientists are Lazy

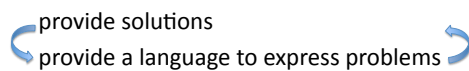
- If we can identify broad categories of algorithms and solve them, then when we see those problems again, they're easy to solve.
- If we encounter a new problem, it's probably similar to a solved one, or a few solved problems put together.

Algorithm Design Overview

2 fundamental algorithmic design components:

- Get to the mathematical core of the problem
- Identify the appropriate techniques based on the problem structure

Algorithms both



Problem's We'll Consider

- Illustrate basic problem paradigms
- Drawn from fundamental issues throughout computer science
- Often discrete => implicit search over many combinatorial possibilities
 - Goal: find solution with particular characteristics

Prerequisites / Review

- Efficiency analysis
- Graphs
- Basic data structures – arrays, stacks, etc.
- Mathematical maturity – logs, limits, etc.
- If you're not confident about these topics, talk to me and read chapters 2 and 3 for homework.

Understanding Efficiency

- Goal: Algorithms should be fast and not use too much space
- We'll concentrate mostly on time analysis
- What does this mean quantitatively?
 - This shouldn't be dependent on platforms, problem instances, input sizes

Worst-case Analysis

Given input of size n what is the largest possible bound on the running time?

How do we know if the bounds are good?

- Consider the brute-force algorithm, i.e. searching through the solution space
- Algorithms better than brute-force have some nice “trick”
- Want: polynomial running time

Precise Notions of Growth

For $n \geq 0$, $n_0 \geq 0$, $c > 0$, $f(n)$, algorithm time $T(n)$...

Upper bounds

- If $T(n) \leq c f(n)$ for all $n \geq n_0$ then $T(n) = O(f(n))$

Lower bounds

- If $T(n) \geq c f(n)$ for all $n \geq n_0$ then $T(n) = \Omega(f(n))$

Tight bounds

- If $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$ then $T(n) = \Theta(f(n))$

Linear Time Algorithms: $O(n)$

- The algorithm's running time is at most a constant factor times the input size
- Process the input in a single pass spending constant time on each item
 - Max algorithm
- Charging scheme, each item of input is charged once
 - Sorted list merge

$O(n \log n)$ time

Frequent running time in cases when algorithms involve:

- Sorting
- Divide and conquer

Quadratic Time: $O(n^2)$

- Every pair of points is processed
 - Nearest neighbor
- Nested loops

$O(n^k)$ Time

- Consider all subsets of points of size k

Slower than Polynomial Time

- All subsets of points of any size: $O(2^n)$
- Number of ways to match n items with each other: $O(n!)$

Sublinear Time

- Query time in a binary search tree: $O(\log n)$
- In general, if we can throw away a *constant fraction* of the input with each step of the algorithm, we can achieve sublinear time.

Graph review

- Graph: A set of nodes and edges that connect nodes
- Undirected graph (or just graph): Edges indicate a symmetric relationship
- Directed graph: Edges indicate an asymmetric relationship
- Edges/nodes can have weights

Why do we care about graphs?

Lots of problems in different contexts can be modeled as graphs:

- Transportation network
- Communication network
- Information network
- Social network
- Dependencies

Remember: we want a clean mathematical model for problems

Graph Properties

- Path: sequence of nodes where each consecutive pair is joined by an edge
- Cycle: a path in which the first node in the path is the same as the last
- Connected graph: an undirected graph in which any two nodes have a path between them
- Tree: a connected graph with no cycle

Tree properties

- Root: a chosen node that identifies the orientation of the tree (with it as the top)
 - Leaf nodes: the last nodes on any path from the root
- Given any edge along a path from the root to a leaf node
- Parent: the node closer to the root
 - Child: the node farther from the root
- Similarly: ancestors and descendants