Overview

Lecture 1

Akshar Varma

3rd July, 2023

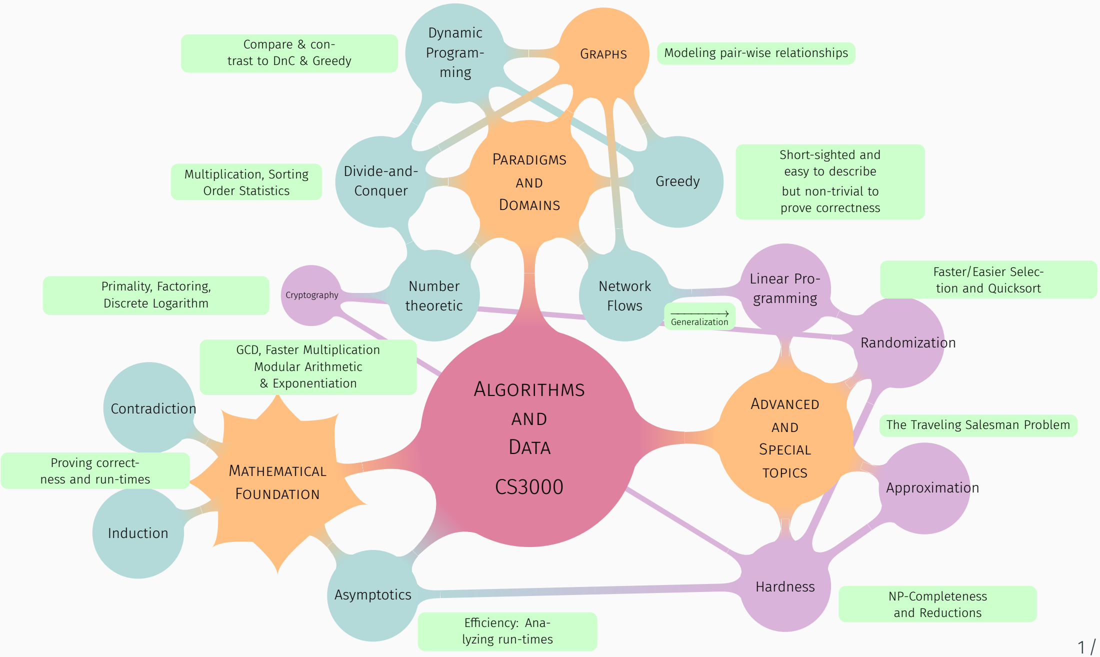
CS3000 Algorithms and Data

Map of CS3000 Algorithms Landscape

Goals and expected outcomes

1. Map of CS3000 Algorithms Landscape

“Algorithms are the computational content of proofs.” — Robert Harper, Benjamin C. Pierce et al.


Algorithms and Data
CS3000
Paradigms and Domains
Graphs
Modeling pair-wise relationships
Dynamic Programming
Compare & contrast to DnC & Greedy
Divide-and-Conquer
Multiplication, Sorting Order Statistics
Greedy
Short-sighted and easy to describe but non-trivial to prove correctness
Number theoretic
Cryptography
Primality, Factoring, Discrete Logarithm
GCD, Faster Multiplication Modular Arithmetic & Exponentiation
Network Flows
Generalization
Linear Programming
Faster/Easier Selection and Quicksort
Randomization
Advanced and Special topics
The Traveling Salesman Problem
Approximation
Hardness
NP-Completeness and Reductions
Mathematical Foundation
Contradiction
Proving correctness and run-times
Induction
Asymptotics
Efficiency: Analyzing run-times

2. Goals and expected outcomes

Course Goals

Algorithms:

- Well-specified procedure (recipe) for turning some *input* into some *output*.
- Efficiency in terms of resource constraints like time and space; parameterized by size of input (typically denoted using n).

This course will focus on

- various algorithm design strategies;
- proving correctness of algorithms;
- and analyzing efficiency (run-time) of algorithms.

Learning Outcomes

- **Create and Validate:** Efficient/performant algorithms for unseen but “everyday” problems.
- **Compare and Choose:** Among different algorithmic paradigms to choose most appropriate strategy.
- **Analyze and Evaluate:** Problem statements and constraints to determine their impact on algorithm design.
- **Modify and Employ:** Known algorithms and examples to new scenarios.
- **Research and Select:** Existing algorithms/implementations based on resource constraints.
- **Read, Examine, Recognize and Duplicate:** Algorithm descriptions given a reasonable amount of detail.

Drone Drop Durability

- *Question:* From what height can a drone fall without breaking apart?
- We want height $x \in \{1, 2, \ldots, n\}$ such that drones break when falling from height $\geq x$. There is a known maximum height n after which the drone definitely breaks.

- *Test setup:* Try dropping a drone from height h, if it breaks, $x \leq h$ otherwise $x > h$.
- Assume that repeated dropping doesn't harm drone at all. No RSI (Repetitive Strain Injury).

- We'll look at what's minimum drops needed to find x when:
    1. No limit on number of drones. If one breaks, use next.
    2. There is only one test drone. If it breaks, you're done.
    3. There are two test drones. You have a *single* spare.

Drone Drop Durability – solutions (derived in-person)

1. No limit on number of drones. If one breaks, use next.

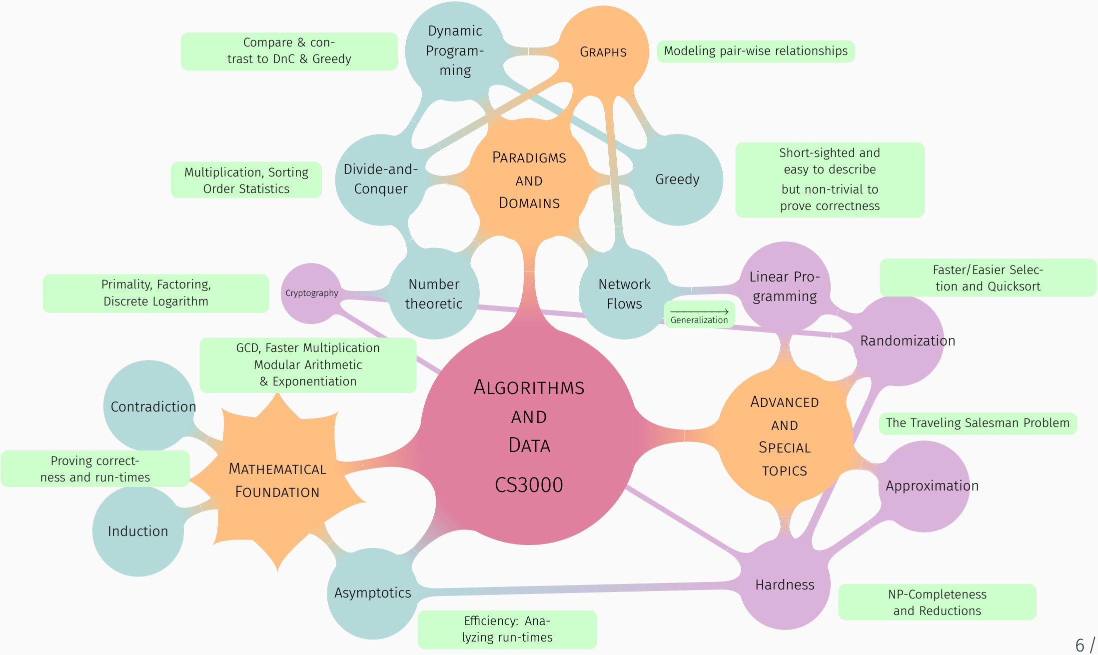
   Binary search.
   Need potentially $\log n$ drones, but drops also bounded by $\log n$.

2. There is only one test drone. If it breaks, you're done.

   Cannot do better than testing heights one by one. Potentially need n drops.

3. There are two test drones. You have a single spare.

   Use bracketing/binning. First drone tries at multiples of $\sqrt{n}$. Whenever it breaks, we have found a $(i \cdot \sqrt{n}, (i+1) \cdot \sqrt{n}\,]$ bracket/bin in which to test spare drone.



Algorithms and Data
CS3000
Paradigms and Domains
Graphs
Modeling pair-wise relationships
Dynamic Programming
Compare & contrast to DnC & Greedy
Divide-and-Conquer
Multiplication, Sorting Order Statistics
Greedy
Short-sighted and easy to describe but non-trivial to prove correctness
Number theoretic
Primality, Factoring, Discrete Logarithm
Cryptography
GCD, Faster Multiplication Modular Arithmetic & Exponentiation
Network Flows
Generalization
Linear Programming
Faster/Easier Selection and Quicksort
Randomization
Advanced and Special topics
The Traveling Salesman Problem
Approximation
Hardness
NP-Completeness and Reductions
Mathematical Foundation
Contradiction
Proving correctness and run-times
Induction
Asymptotics
Efficiency: Analyzing run-times