Intro to Algorithms

1. Motivating Problem
2. Algorithm Analysis
3. Graph/Graph Algorithms
4. Greedy Algorithms
5. Dynamic Programming
6. Network Flow
7. NP-completeness

What is an algorthm

1. A step-by-step procedue to solve a program
2. Every program is instantiation of some algorithm

Example: **Sorting**

> **An algorithm solves a general, well-specified problem**
> Given a sequence of keys, as input, produce an output reordering (i.e. a permutation) of keys so that:
>
> **The problem has specific instances**
> [Dopey, Happey, Grumpy] or [3, 5, 7, 1, 2, 3]
>
> **An algorithm takes any possible instance and produces output that has the desired properties**
> e.g.: insertion sortm quicksortm heapsort, ...
>
> **It is hard to design algorithms that are:**
>
> - correct
> - efficient
> - (easily) implmentable
>
> **To do so effectively, we need to know about:**
>
> - algorithm design and modeling techniques

> - existing resources (i.e. don't reinvent the wheel)
>
> **How do you know an algoritm is correct?**
> It produces the correct output on every possible input

- Since there are usually infinitely many inputs, ensuring this is not trivial
- Saying "it's obvious" can be dangerous
- Often one's intution can be tricked by a particular type of input

Example: **Shortest path**
Given a set of points in the plane, what is the shortest tour that visites each point and returns to the beginning

> **An application**
> Consider a robot arm that solders contact points on a circuit board; we want to minimize the movement fo the robot arm

- Starting by visiting any point
- While all points are not visited choose an unvisited point closest to the last visited point and visit it
- return to the first point

> *place good example*
> *place bad example*

Measuring Efficiency

Software and hardware are both continually advancing: advancing software constantly demands a faster CPU, more memory, etc.

Givien a problem:

- What is an efficient algorithm?
- What is the most efficient algorithm?
- Does there even exist an algorithm?

We generally focus on machine-independent measures

> **Measuring Efficiency**
>
> 1. We analyze a "pseudocode" verision of the algorithm
> 2. We assume an idealized model of a machine in which one instruction takes on unit of time

> **Big-O notation**
> Analyze the order of magnitude changes in efficiency as the problem size increases
>
> **We often focus on worst-case performance**
> This is safe, the worst case often occurs frequently and the average case is often just as bad

1. There's no point in finding the fastest algorithm for parts of the program that are not bottlenecks
2. If the program will only be run a few times or time is not an issue (e.g. the program will be left to run overnight), there's no real proint in finding the faswtest algorithm
3. Cover a variety of fundamental algorithm design techniques as applied to a numbner of basic problems
4. Along the way infuse discussion with different types of algorithm analysis
5. Study some lower bounds indicating inherent limitations in finding efficeint algorithms
6. Learn about undecidability: some problmes are simply unsolvable