

## CSE 830: Design and Theory of Algorithms

---

Dr. Charles Ofria

## Overview

- Administrative stuff...
- Rules of the game
- What is an algorithm?
- Systems for studying algorithms
- Some examples
- Lecture schedule overview

## Course Info

- Textbook: Introduction to Algorithms, *Second Edition* by Cormen, Leiserson, Rivest, and Stein
- Web page: <http://www.cse.msu.edu/~cse830/>
- My Info: 2140 Engineering, 355-8389, [ofria@msu.edu](mailto:ofria@msu.edu)
- Office Hours: To be decided (for now, by appointment.)

## Tentative Grading

25% Homework (5 assignments worth 5% each)  
10% Programming Project  
20% Exam I  
20% Exam II  
25% Final Exam

What would you prefer?

- Quizzes? (more frequent, but easier than exams)
- More Programming?
- More take-home assignments?

## Homework

- 5 Homework Assignments
- You must work in small groups ; 2-3 students per group. Only hand in one written set of answers.
- A single homework is allowed to be a week late; otherwise 20 points off per class session.
- Re-grades can go in either direction!

## Project

- You will be given a hard problem and you need to write a program to solve it correctly and efficiently.
- This is *not* a group project - everyone must write their own.
- Prizes will be given out for the best solutions.

## Exams

- There will be three exams over the course of the semester.
- All exam questions will be made available ahead of time - with many others.
- I typically give 140 points of questions on an exam, but you only need to answer 100 points worth.
- Exams are cumulative.

## What is an Algorithm?

According to the *Academic American Encyclopedia*:

An algorithm is a procedure for solving a usually complicated problem by carrying out a precisely determined sequence of simpler, unambiguous steps. Such procedures were originally used in mathematical calculations (the name is a variant of algorithm, which originally meant the Arabic numerals and then "arithmetic") but are now widely used in computer programs and in programmed learning.

## What is an Algorithm?

Algorithms are the ideas behind computer programs.

An algorithm is the thing that stays the same whether the program is in C++ running on a Cray in New York or is in BASIC running on a Macintosh in Katmandu!

To be interesting, an algorithm has to solve the general form of a well-defined problem. An algorithmic problem is specified by describing the set of instances it must work on and what desired properties the output must have.

## Example: Sorting

**Input:** A sequence of  $N$  numbers  $a_1, \dots, a_n$ .

**Output:** the permutation (reordering) of the input sequence such that  $a_1 \leq a_2 \leq \dots \leq a_n$ .

We seek algorithms that are *correct* and *efficient*.

## Correctness

For any algorithm, we must prove that it *always* returns the desired output for all legal instances of the problem.

For sorting, this means even if (1) the input is already sorted, or (2) it contains repeated elements.

## Efficiency

What is "efficiency"?

In what ways can we compare algorithms?

## How do we measure *speed*?

- Compare by processor?
- By compiler?
- What about optimization level?

“Why not use a super-computer?”

## Example: Odd or Even?

What is the best way to determine if a number is odd or even?  
Some of the algorithms we can try are:

- Count up to that number from one and alternate naming each number as odd or even.
- Factor the number and see if there are any twos in the factorization.
- Keep a lookup table of all numbers from 0 to the maximum integer.
- Look at the last bit (or digit) of the number.

## Expressing Algorithms

Three ways of expressing algorithms:

- Implementations
- Pseudo-code
- English

## Types of Algorithms

**Search problems:** find X in the input satisfying property Y

**Structuring problems:** Transform input X to satisfy property Y

**Construction problems:** Build X satisfying Y

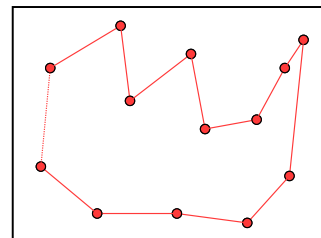
**Decision problems:** Does X satisfy Y?

## Example Problem

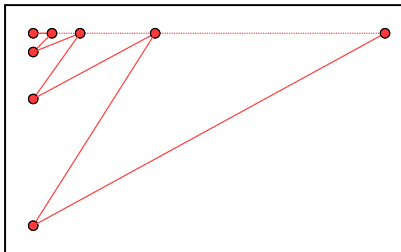
You have a robot arm equipped with a soldering iron. To enable the robot arm to do a soldering job, you must construct an ordering of the contact points. The robot visits (and solders) the first contact point, then visits the second point, third, and so forth until the job is done.

Since robots are expensive, we need to find the order that minimizes the time (I.e. travel distance) it takes to assemble the circuit board.

## Example Circuit

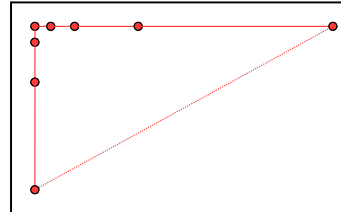


Not Correct!

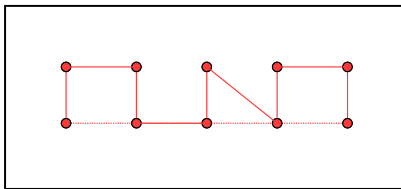


A better algorithm?

What if we always connect the closest pair of points that do not result in a cycle or a three-way branch? We finish when we have a single chain.



Still Not Correct!



A *Correct* Algorithm

We could try all possible orderings of the points, then select the ordering which minimizes the total length:

```
d = ∞
For each of the n! permutations, Pi of the n points,
  if cost(Pi) < d then
    d = cost(Pi)
    Pmin = Pi
return Pmin
```

## Analyzing Algorithms

Algorithms are the only important, durable, and original part of computer science because they can be studied in a machine and language independent way.

How can we study the time complexity of an algorithm if we don't choose a specific machine to measure it on?