CSC373 Notes

Max Xu

January 11, 2026

Contents

§1 Day 1: Intro (Jan 06, 2026)

This was taken off the slides from past years.

1.1 About this Class

This class is about designing algorithms to solve problems.

We will be:

- Designing fast algorithms
  - Divide and conquer
  - Greedy algorithms
  - Dynamic programming
  - Network flow
  - Linear programming
- Proving no fast algorithms are likely possible
  - Reductions and NP-completeness
- What to do if no fast algorithms are possible
  - Approximation algorithms
  - Randomized algorithms

When we analyze an argument, we do correctness and running-time proofs.

§2 Day 2: Intro Redux (Jan 08, 2026)

This course is now about the thought process behind solutions of problems. We use the **RAM Computational Model**.

A proof is a convincing argument:

- Convince your TA for marks
- Convince employer that your program does what it claim it does
- Convince yourself that you're not producing word salad

Sometimes, formal verification is used for mission-critical applications, where unit tests may not have sufficient coverage. We use semi-formal proofs in this course, to prove specific results (as opposed to more general ones, like in math).

2.1 Divide & Conquer

The general framework is to

- Break a problem into two smaller subproblems of the same type
- Solve each problem recursively and independently
- Quickly combine solutions from subproblems to form a solution to a bigger part of the problem

'Quick/cheap' means that the step count is in $O(f(n))$ where f is a polynomial.

Recurrence relations are often encountered while analyzing the running time of divide-and-conquer algorithms. We take the master theorem from (CLRS) for granted, a general result about the asymptotic behavior of certain types of recurrences.

Theorem 2.1 (CLRS Master Theorem)

Let $a \geq 1$, $b > 1$. Have $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence $T(n) = aT(n/b) + f(n)$, where n/b is interpreted as $\lceil \frac{n}{b} \rceil$ or $\lfloor \frac{n}{b} \rfloor$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) \in O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) \in \Theta(f(n))$

n/b describes the size of the subproblems, and $f(n)$ describes the step count required to merge/divide the subproblems to form a solution of size n. The Master Theorem handles the leaf-heavy, balanced, and root-heavy case in that order.

Example 2.2

Algorithms considered divide-and-conquer include:
- Closest pair in $\mathbb{R}^2$, with non-degeneracy assumption
- Karatsuba's Algorithm
- Strassen's Algorithm

There was also a brief discussion about galactic algorithms.