CMSC 27230: Honors Algorithms, Winter 2025

Homework and Material Covered

Course home | SLIDES | LaTeX HW template | Gradescope | Policy on collaboration and internet use | Texts | Handouts | Categories of exercises | Statistics | Grading | Using previous exercises | #1| #2| #3 | #4| #5| #6 | #P2| #7| #8 | #9| #10| #11 | #12| #13| #14 | #15| #16| #17 | #18| #19| #20 | #21| #22| #23

View the 2025-03-08 15:30 version of this page in PDF

REFRESH your browser to see latest update

Abbreviations used in references to **online material** are explained **here**.

PROVE YOUR ANSWERS unless it is expressly stated that no proof is required.

Clarity and elegance count.

Go to top

REFRESH your browser to see latest update

Use the LaTeX template provided (click "LaTeX HW template" on the banner).

Using previous exercises

When solving an exercise, **you may use any of the lower-numbered DO or HW exercises and Challenge problems, Facts, Theorems -- any lower-numbered results -- without proof**, unless otherwise instructed, but you need to **reference** what you use, and state how you are using such a reference (e.g., "the variable z in the current exercise corresponds to the variable u in the referenced exercise"). If you use any other nontrivial results, you need to prove them. Here "notrivial" is to be understood in comparison with the difficulty of the exercise itself. If in doubt, ask the instructor by email.
The same applies to the online references, e.g., when solving a problem from ASY, you may use any lower-numbered statements from ASY.

You may also use any result from the **LinAlg** book, except where the exercise is essentially identical with the result or a special case of the result from the book.

Presentation of algorithms

Algorithms must be presented in pseudocode (unless the problem expressly states that pseudocode is not required). A pseudocode must clearly show the loop structure and the logic of the algorithm (**for**, **while** loops, **if/then/else** statements). For clarity, employ **end(for)**, **end(while)**, **end(if)** statements (so your grader will not have to rely on indentations to understand the loop structure). Before the pseudocode, you need to **explain verbally, what the algorithm does.** Additionally, please generously comment the lines or sections of your pseudocode, explaining the *meaning* of the steps done. The point is, your work is evaluated by a *human*, please **help the grader understand** what is happening.
Important: **explain your variables** (what do they mean, what role do they play in the algorithm). For each variable, state what **type of object** it is (array, linked list, integer, real, etc.).
Do not use commands from modern programming languages in your pseudocode, their bit complexity may be difficult to evaluate, and each language has their idiosyncrasies that may look confusing. (Writing "$i++$" is ok, but "$i := i+1$" is preferable, and please avoid "$i+=1$".)
Please number the lines of your pseudocode for easier reference.
Several examples of pseudocodes are given in the handouts, study them and view them as models for your pseudocodes.

Go to top

Each time you open this page, **REFRESH** .

Class #26, Fri, Mar 7

Material covered: Turing machine. Layers of a language: $L_n = L \cap \Sigma^n$. Polynomial-time computable languages vs. uniform families of polynomial-size circuits. Proof of the Cook-Levin Theorem. SAT $\prec_{\text{Karp}}$ 3SAT and therefore 3SAT $\in \mathcal{NPC}$. CLIQUE language. 3SAT $\prec_{\text{Karp}}$ CLIQUE (proved). 3DM (3-dim matching) $\prec_{\text{Karp}}$ SUBSET-SUM (mentioned), therefore SUBSET-SUM $\in \mathcal{NPC}$, therefore integer KNAPSACK $\in \mathcal{NPC}$. Knapsack approximable within ϵ (solution of value $\geq$ OPTIMUM$\cdot(1-\epsilon)$) found in time $O(n^3/\epsilon)$: polynomial-time approximation scheme. MAX-3SAT: random substitution satisfies an expected $7m/8$ (out of m) clauses; therefore, OPT approximable within a $1/8$ factor. Approximation within any factor $1/8-\delta$ impossible unless $\mathcal{P}=\mathcal{NP}$: MAX-3SAT is "approximation resistant."

26.10 STUDY "The P and NP complexity classes" handout (referred to below as "P/NP")

26.13 DEF A language L is **$\mathcal{NP}$-complete** if (a) $L \in \mathcal{NP}$ and (b) $(\forall M \in \mathcal{NP})(M \prec_{\text{Karp}} L)$ (all $\mathcal{NP}$-language are Karp-reducible to L).

26.13 REVIEW from class Turing machines. The Cook-Levin Theorem: SAT in $\mathcal{NP}$-complete. Review the sketch of the proof of this theorem given in class.

26.16 DO Expanding the definition of $\mathcal{NP}$-completeness, the the Cook-Levin Theorem states that if $L \in \mathcal{NP}$ then $L \prec_{\text{Karp}}$ SAT. Prove the converse:
If $L \prec_{\text{Karp}}$ SAT then $L \in \mathcal{NP}$.

26.19 HW (11 points) P/NP 3.4 ($\mathcal{P} \subseteq \mathcal{NP}$, shortest witness)

26.22 HW (12 points) For $k \in \mathbb{N}$, let kCOL denote the set of k-colorable graphs. Assuming that 3COL $\in \mathcal{NPC}$, prove that 4COL $\in \mathcal{NPC}$ by constructing a simple Karp-reduction from 3COL to 4COL.

26.25 HW (15 points) Joey misread the definition of the complexity class $\mathcal{NP}$, he forgot to bound the length of the witness. So the language class $\mathcal{JC}$ (Joey's class) has the following definition. Let Σ be a finite alphabet, $|\Sigma| \geq 2$, and let $L \subseteq \Sigma^*$. Then $L \in \mathcal{JC}$ if and only if

$$(\exists M \in \mathcal{P})(\forall x \in \Sigma^*)(x \in L \Leftrightarrow (\exists w \in \Sigma^*)((x, w) \in M))$$

Determine the language class $\mathcal{JC}$.

26.30 DEF The SUBSET-SUM problem is defined as follows. INPUT: a list $(a_1, \ldots, a_k, b)$ of positive integers
QUESTION: Does there exist a subset $S \subseteq [k]$ such that $\sum_{i \in S} a_i = b$?
Let SBSM denote the corresponding language (the set of YES-instances). (A YES-instance is an input for which the answer to the QUESTION is "YES.")
Clearly, SBSM $\in \mathcal{NP}$ (the witness is S).
Examples: $(5, 2, 5, 8, 13) \in$ SBSM (witness: $S = \{1, 4\}$), $(5, 2, 5, 8, 12) \in$ SBSM (witness: $S = \{1, 2, 3\}$), $(5, 2, 5, 8, 11) \notin$ SBSM.
It is known that SBSM $\in \mathcal{NPC}$, by Karp reduction from 3DM (3-dimensional matching).

26.33 DEF The INTEGER KNAPSACK DECISION PROBLEM is defined as follows:
INPUT: a list $(v_1, \ldots, v_\ell, R, w_1, \ldots, w_\ell, W)$ of positive integers
QUESTION: Does there exist a subset $T \subseteq [\ell]$ such that $\sum_{j \in T} v_j \geq R$ and $\sum_{j \in T} w_j \leq W$?
Let KNAP denote the corresponding language (the set of YES-instances).
Clearly, KNAP $\in \mathcal{NP}$ (the witness is T).

26.36 HW (11 points) Assuming that SBSM $\in \mathcal{NPC}$, prove that KNAP $\in \mathcal{NPC}$ by constructing a simple Karp-reduction from SBSM to KNAP.

26.39 XX

Go to top

Class #25, Wed, Mar 5

Material covered: Adjacency matrix of digraph. Powers of the adjacency matrix count walks. Number of triangles counted in Strassen time. Square matrix times vector multiplication accelerated if done simultaneously on many vectors (same matrix). -- The complexity classes $\mathcal{P}$ and $\mathcal{NP}$. Formal definition of $\mathcal{NP}$. Karp reduction among languages. $\mathcal{NP}$-completeness: Cook-Levin Theorem stated. Conjectures: $\mathcal{NP} \neq co\mathcal{NP}$ and $\mathcal{P} \neq \mathcal{NP} \cap co\mathcal{NP}$. Candidate language to demonstrate the last separation: Factoring language: FACT $\in \mathcal{NP} \cap co\mathcal{NP}$ but conjectured not to be in $\mathcal{P}$.

25.10 DEF Adjacency matrix of digraph

Go to top

Class #24, Mon, Mar 3

Material covered: The complexity classes $\mathcal{P}$ and $\mathcal{NP}$. Examples of languages in $\mathcal{NP}$: 3COL (the set of 3-colorable graphs), HAM (the set of Hamiltonian graphs), FACT

$= \{(x, y) \mid x, y \in \mathbb{N}, (\exists d)(2 \le d \le y \wedge d \mid x)$ (the factoring language), COMPOSITES (the set of composite numbers). Witness (proof of membership). Formal definition of $\mathcal{NP}$. Karp-reduction.

Go to top

Class #23, Fri, Feb 28

Material covered: Language classes, complexity classes. Computable functions (also called recursive functions), computable languages (also called recursive languages), decidable decision problems, the complexity class $\mathcal{R}$. Computably enumerable languages (also called recursively enumerable languages), the complexity class $\mathcal{RE}$. The HALTING language. HALTING $\in \mathcal{RE}$. HALTING $\notin \mathcal{R}$, i.e., the Halting problem is undecidable (proved).

23.15 DEF Let P be a computer program in a universal programming language such as Python. Assume P takes input strings $x \in \Sigma_1^*$ and if it halts, it produces and output string $P(x) \in \Sigma_2^*$ where the Σ_i are finite alphabets. If P does not halt (runs forever) on input x then we write $P(x) = \infty$ where ∞ is a special symbol not in Σ_2.
We say that a function $f : \Sigma_1^* \to \Sigma_2^*$ is **computable** if there exists a Python program P that halts (eventually stops) on every input string $x \in \Sigma^*$ and $(\forall x \in \Sigma^*)(f(x) = P(x))$. Computable functions are also called **recursive functions**.

23.18 DEF A **decision problem** is a function $f : \Sigma^* \to \{0, 1\}$. The decision problem f is **decidable** if f is computable and **undecidable** otherwise.

23.21 DEF For a language $L \subseteq \Sigma^*$ the **characteristic function** of L is the function $f_L : \Sigma^* \to \{0, 1\}$ defined by

$$f_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \in \Sigma^* \setminus L \end{cases}$$

So the function f_L answers the membership queries in L: for all $x \in \Sigma^*$ we have $x \in L \Leftrightarrow f_L(x) = 1$ and $x \notin L \Leftrightarrow f_L(x) = 0$.
We say that a **language** $L \subseteq \Sigma^*$ is **computable** (or **recursive**) if its characteristic function is computable, i.e., its **membership problem** is decidable.
The set of **computable languages** is denoted $\mathcal{R}$.

23.24 DEF The language $L \subseteq \Sigma^*$ is **computably enumerable** if there exists a Python program P that halts on input $x \in \Sigma^*$ if and only if $x \in L$. Computably enumerable languages are also called **recursively enumerable**. The set of computably enumerable languages is denoted $\mathcal{RE}$.

23.27 DO The language $L \subseteq \Sigma^*$ is computably enumberable if and only if either $L = \emptyset$ or there exists a computable function of which L is the range, i.e., $L = \{f(x) \mid x \in \Sigma^*\}$.

23.28 DO Prove: $\mathcal{R} \subseteq \mathcal{RE}$.

23.30 DEF The halting problem is to decide, given a Python program P and no input (empty input string) whether P will halt. We denote the corresponding language HALTING $= \{$ halting programs $\}$.

23.32 DO Prove: HALTING $\in \mathcal{RE}$.

23.35 THEOREM HALTING $\notin \mathcal{R}$ (the HALTING problem is undecidable).

23.38 Terminology If $\mathcal{C}$ is a set of languages, we refer to $\mathcal{C}$ as a **language class**. "Important" language classes are called **complexity classes**.
Examples of complexity classes: $\mathcal{P}$, $\mathcal{NP}$, $co\mathcal{NP}$, $\mathcal{R}$, $\mathcal{RE}$, $co\mathcal{RE}$.

23.41 DEF A language L is always given with reference to a finite alphabet which we call the **alphabet of definition** of L, denoted Σ_L. So $L \subseteq \Sigma_L^*$. The **complement** of L is the language $\overline{L} = \Sigma_L^* \setminus L$.
If $\mathcal{C}$ is a language class then $co\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$.

23.44 DO Prove: $\mathcal{R} = co\,\mathcal{R}$.

23.47 DO If $\mathcal{C}$ and $\mathcal{D}$ are language classes and $\mathcal{C} \subseteq \mathcal{D}$ then $co\mathcal{C} \subseteq co\mathcal{D}$. In particular, $\mathcal{R} \subseteq \mathcal{RE} \cap co\mathcal{RE}$.

23.51 HW (12 points) Prove: $\mathcal{R} = \mathcal{RE} \cap co\mathcal{RE}$.

23.XX

Go to top

Class #22, Wed, Feb 26

Material covered: Decision problems $=$ predicates (functions with codomain $\{0,1\}$). The SAT problem: Boolean circuit satisfiability. The $3\,\mathrm{COL}$ problem (3-colorability of graphs). Karp reduction.
$3\,\mathrm{COL} \prec_{\mathrm{Karp}} \mathrm{SAT}$ (proved). Strings, languages. Membership problem. Decision problems $\leftrightarrow$ languages.
The complexity class $\mathcal{P}$. Multiplication of integer matrices computable in polynomial time but NOT in $\mathcal{P}$ because it is not a decision problem. Primality testing: PRIMES $\in \mathcal{P}$ (Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, 2002) (stated). The language class $\mathcal{NP}$.

22.10 XX

Go to top

Class #21, Mon, Feb 24

Material covered: Boolean functions. DAGs. Boolean circuits, gates, wires, variables, literals. Layered circuits. Conjunctive Normal Form (CNF), Disjunctive Normal Form (DNF). The PARITY function. Bounded-depth circuits. The Yao-Håstad theorem, Håstad Switching Lemma (mentioned). Shannon's Theorem (1950): Existence of Boolean functions that require exponential-size circuits. The Probabilistic Method: proof of existence without explicit construction. Derandomization.

21.10 DEF A **Boolean function** in n variables is a function $f : \{0,1\}^n \to \{0,1\}$.

21.13 DO The number of Boolean functions in n variables is 2^{2^n}.

21.16 DEF A DAG (directed acyclic graph) is a digraph without directed cycles.

21.19 DEF A **Boolean circuit** is a DAG of which the vertices are labeled "AND" or "OR" or a variable, and some of the edges are labeled "negation." The vertices are called **gates**, the edges are called **wires**. The **size**

of a circuit is the number of its wires. An **assignment** assigns Boolean values (0 or 1) to each variable. The **fan-in** of a gate is its in-degree, the **fan-out** is its out-degree. An AND-gate of fan-in zero has value 1, and OR-gate of fan-in zero has value 0. One gate is designated to be the **output gate**. The **depth** of the circuit is the length of the longest (directed) path from an input gate to the output gate. --- Given an assignment to the variables, the gates can be evauated inductively, yielding a Boolean output value. This the circuit defines a Boolean function.

21.22 DEF A **literal** is a variable or its negation. A **literal-based circuit** has two kinds of input gates: the variables and their negations, and it has no negations on the wires.

21.25 HW (12 points) If a Boolean circuit has size m then it can be simulated by a literal-based Boolean circuit of size $\le 2m$. "Simulation" means the two circuits compute the same Boolean function.

21.26a DEF In a **layered circuit** is a literal-based circuit where the gates are partitioned into layers L_0, L_1, . . ., L_d where the bottom layer, L_0, consists of the $2n$ literals, and the other layers are alternately AND-layers and OR-layers.

21.26b DEF An AND gate or AND operation is called a **conjunction**. An OR gate or OR operation is called a **disjunction**.
A **disjunctive clause** is a disjunction of a list of literals. Example: $x_2 \vee \overline{x}_3 \vee \overline{x}_5$.
A **conjunctive clause** is a conjunction of list of literals. Example: $x_2 \wedge \overline{x}_3 \wedge \overline{x}_5$.
A **conjunctive normal form** (CNF) is a conjunction of disjunctive clauses: $\bigwedge D_i$ where each D_i is a disjunctive clause.
A **disjunctive normal form** (DNF) is a disjunction of conjunctive clauses: $\bigvee C_i$ where each C_i is a conjunctive clause.

21.27 DO The depth-2 layered circuits are precisely the CNFs and the DNFs.

21.28 HW (10 points) Every Boolean function in n variables can be represented (a) by a DNF of size at most $(n+1)2^n$ and (b) by a CNF of size at most $(n+1)2^n$.
Update, March 2, 22:10. Changed $n \cdot 2^n$ to $(n+1)2^n$.

21.31 DEF The **parity function** in n variables is defined as $\mathrm{PARITY}_n(x_1,\dots,x_n) = (\sum_{i=1}^n x_i \bmod 2)$. Another notation for this function is $x_1 \oplus x_2 \oplus \cdots \oplus x_n$.

21.34 HW (7 points) Construct a CNF for the function $x \oplus y \oplus z$. No proof required.
Hint. Construct a DNF for $x \oplus y \oplus z \oplus 1$ and then negate it.

21.37 HW (14 points, due Thursday, March 13) Prove: a CNF representing the PARITY_n function must have at least $n \cdot 2^{n-1}$ wires.

21.XX More to follow.

21.XX

21.XX

21.XX

Go to top

Class #20, Fri, Feb 21

Material covered: Public-key cryprography. Fermat's little Theorem. The RSA protocol. Modular exponentiation.

20.07 DO Let A denote the 2×2 matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Determine the matrix A^k. Prove your answer. The entries of the matrix will come from a familiar sequence. Hint. Experiment with small values of k. Observe a pattern, make a conjecture. Prove your conjecture by induction.

20.10 STUDY LinAlg Chapter 14.1 (Basic concepts of arithmetic), especially 14.1.9 (Division Theorem) and the rest of Chap.14.1.1, and Chapters 14.1.2 (Divisibility), 14.1.3 (Greatest common divisor), especially Theorem 14.1.39 (Bezout's Lemma), Chpater 14.1.4 (Fundamental Theorem of Arithmetic), Chapter 14.1.5 (Least common multiple).

20.13 STUDY LinAlg Chapter 14.2 (modular arithmetic).

20.16 STUDY "Euclid's algorithm and multiplicative inverse" handout.

20.19 STUDY "Repeated squaring" handout.

20.22 Fermat's little Theorem (FlT) Let p be a prime number and $a \in \mathbb{Z}$ an integer that is not divisible by p. Then $a^{p-1} \equiv 1 \pmod{p}$.

20.25 DO Let p be a prime number and let $n \in \mathbb{N}$ such that $n \equiv 1 \pmod{p-1}$. Then $(\forall a \in \mathbb{Z})(a^n \equiv a \pmod{p})$.

20.28 DO Let $k, \ell, m \in \mathbb{Z}$ such that k and ℓ are relatively prime (i.e., $\gcd(k, \ell) = 1$). Then $(\forall a, b \in \mathbb{Z})(a \equiv b \pmod{k\ell} \Leftrightarrow a \equiv b \pmod{k}$ and $a \equiv b \pmod{\ell})$.

20.31 DO Let $p \neq q$ be prime numbers. Let $M = (p-1)(q-1)$. Let $k \in \mathbb{N}$ such that $k \equiv 1 \pmod{M}$. Then $(\forall a \in \mathbb{Z})(a^k \equiv a \pmod{pq})$.

20.34 The RSA cryptographic scheme Let $p \neq q$ be prime numbers. Let $N = pq$ and $M = (p-1)(q-1)$. Let $e, f \in \mathbb{N}$ such that $ef \equiv 1 \pmod{M}$. (f is the multiplicative inverse of e modulo M.)
The "message space" is the set $\mathcal{X} = \{0, 1, \ldots, N-1\}$.

Alice wishes to receive encrypted messages from her customers. She publishes (N, e) (this is her **public key**). She keeps f private (this is her **private key**). She discards p, q, M.
Encryption: a function $E : \mathcal{X} \to \mathcal{X}$ defined as $E(x) = (x^e \bmod N)$.
Decryption: a function $D : \mathcal{X} \to \mathcal{X}$ defined as $D(y) = (y^f \bmod N)$.

20.36 history. The RSA scheme is named after its inventors, **Ronald Rivest, Adi Shamir, and Leonard Adleman** (1977). This algorithm ushered in the age of modern cryptography, turning cryptography, an ancient art, into science. Public-key cryptography is a key tool of the digital age without which we would not be able to conduct **electronic commerce.** Rivest, Shamir, and Adleman received the 2002 Turing award for their discovery.

The RSA scheme was actually first invented in 1973, a few years before Rivest et al., by British mathematician **Clifford Cocks**, working for the *UK Government Communications Headquarters* (GCHQ). His discovery was **classified** and shared with the *US National Security Agency* (NSA). These agencies recognized the military significance of the discovery but failed to appreciate its commercial value. By keeping it classified until 1997, they prevented it from triggering the cryptographic revolution the RSA annoucement subsequently touched off in academic research, highlighted by the 2012 Turing award shared by Shafi Goldwasser and Silvio Micali.

One wonders what 17th century French mathematician **Pierre de Fermat** (1601-1665) would have thought if someone had foretold that his "little theorem" would become central to a trillion-dollar industry.

20.38 HW (4 points) Prove that both E and D are permutations of the set $\mathcal{X}$ and they are inverses of one another:
$(\forall x \in \mathcal{X})(D(E(x)) = x)$ and $(\forall y \in \mathcal{X})(E(D(y)) = y)$.

20.41 The RSA protocol Customer Bob wishes to send Alice a secret message $x \in \mathcal{X}$ (the **plaintext**).
Step 1. Bob encrypts the message using Alice's public key, producing the **ciphertext** $y = E(x)$.
Step 2. Bob sends y to Alice over a public channel. Eavesdropper Eve listens, learns y, but hopefully cannot decipher the message.
Step 3. Alice decrypts the message using her private key: $x = D(y)$.

20.44 HW (9 points) Assume p and q are n-digit primes. Prove: If Eve can guess the prime p then she can decipher the message in polynomial time. Use only algorithms covered in handouts. Other web and text sources will not be accepted.
Update March 3, 8:40. To justify polynomial time, do not use any fancy multiplication/division of integers. If x, y are integers with $\le n$ digits then computing xy and $(x \bmod y)$ costs $O(n^2)$ bit operations. You don't need to prove this.

20.47 Advice to Alice Do not use prime numbers sold by a vendor. (The vendor can break your encryption.) Do not use someone else's software to generate your primes. Do not use a prime number published on the internet. Create a pair of random n-digit primes on your own.

20.49 Computational complexity assumption. For large n, for a pair of random n-digit primes p, q it is computationally infeasible to factor $N = pq$.
This assumption is a necessary but not sufficient condition of the security of the RSA scheme. The assumption is based on the brainpower of a long line of distinguished mathematicians who worked on this problem. The best factoring algorithms known have performance guarantee of $\exp(O(\sqrt{n}))$.

Public-key cryptography is based by necessity not on the lack of information (N contains full information about its prime factors p and q). The concept of public-key cryptography was invented by **Whitfield Diffie** and **Martin Hellman** (1976, Turing award 2015). Their paper motivated Rivest et al.; RSA was the first, and to this day most successful, implementation of the Diffie-Hellman concept. Essentially the same concept ("non-secret cryptography") was invented earlier (1970) by British cryptographer James H. Ellis at the GCHQ; his classified work motivated his colleague Clifford Cocks (see above, 20.36).

20.52 Recommended movie: ***Sneakers*** (1992), starring Robert Redford. In the thriller, the good guys and the bad guys compete to get hold of a black box capable of factoring integers using an efficient algorithm designed and programmed by a mathematician murdered after he hints at the possibility of an "elegant" algorithm in a lecture at Berkeley.

20.55 Alice's **computational tasks:**
(a) Generate a pair of random n-digit primes, p and q.
(b) Find a number e that is relatively prime to $M := (p-1)(q-1)$.
(c) Compute f, the multiplicative inverse of e modulo M.
(d) Decrypt the ciphertext: compute $(y^f \bmod pq)$ (modular exponentiation).
We shall show that each of these tasks can be solved in polynomial time. (Naturally, (a) requires randomization, the others are deterministic.) Note that (c) and (d) are explicitly solved in handouts.

20.58 FACT A **Bernoulli trial** is an randomized experiment of which the outcome is "success" or "failure." Let p be the probability of success. We repeat the experiment with independent random choices until we encounter "success." Then the expected number of trials is $1/p$.

20.61 HW (6+4 points, due Thursday, March 13) (a) Consider the following Bernoulli trial: pick an n-bit integer at random (initial zeros permitted). We claim success if the number picked is prime. Prove that the probability of success is asymptotically c/n for some constant $c > 0$. Determine the value of c. (b) Explain how this allows Alice to pick a random n-digit prime in expected time polynomial in n. You may refer to an algorithm mentioned (but not discussed) in class (see the slides).

20.64 HW (6+4 points, due Thursday, March 13) (a) Given a positive integer x, let $p(x)$ denote the smallest prime number that does not divide x. Prove: $p(x) \lesssim \ln x$. Use 3.58 CH.
Here $a_n \lesssim b_n$ denotes "less than or asymptotically equal." For positive values a_n and b_n this means that $\limsup a_n/b_n \leq 1$. (b) Explain how this result allows Alice to pick e (deterministically, no randomization) in polynomial time.

20.66 HW (6 points, due Thursday, March 13) Given an n-bit natural number $x \in \mathbb{N}$, compute its rounded square root $\lfloor\sqrt{x}\rfloor$ in time $O(n^3)$. Assume multiplication of n-bit integers takes $O(n^2)$. Don't use a super efficient algorithm such as Newton's method, the target is to make the algorithm simple to analyze within the stated time bound. Use a simple algorithmic technique we have used many times.

20.67 HW (9 points) On Alice's discarded hard drive, Eve finds the value M. Prove: now Eve can factor N in polynomial time and thereby break Alice's encryption. (You don't need to show the "thereby" part, just that Eve can factor N.)
Revised update March 3, 11:50. For the complexity of integer multiplication and division, see 20.44. For taking square roots, rounded (computing $\lfloor\sqrt{x}\rfloor$ for a positive integer x), assume one can do it in $O(n^3)$ if x has n digits (see 20.66).

20.75 HW (14 points, due Thursday, March 13) Given the positive integers k and m (in binary), compute $(F_k \bmod m)$ in polynomial time. State your best exponent in the polynomial time-bound, assuming multiplication of n-bit integers takes $O(n^2)$ (no fancy multiplication). Elegance matters.

Go to top

Class #19, Wed, Feb 19

Material covered: Min-weight spanning tree algorithms: a variety of greedy approaches. Borůvka's a parallel algorithm. The Jarnik (1930) (Prim, 1957) algorithm: growing the tree from a root. Implementation: by slight modification of Dijkstra's RELAX routine. --- Matchings in a graph, matching number $\nu(G)$.

Maximal vs maximum matching. Greedy matching algorithm $(1/2)$-optimal. Polynomial-time algorithms to find maximum matching mentioned: Dénes König (1931) for bipartite graphs, Jack Edmonds for all graphs (1965). Independent sets, independence number $\alpha(G)$. Cliques, clique number $\omega(G) = \alpha(\overline{G})$. Legal coloring of the vertices of a graph. Chromatic number $\chi(G)$. Systems of linear equations: if the input has integer coefficients then Gaussian elmination works in polynomial time (Jack Edmonds, 1964, inventor of the notion of polynomial time). Concern: blow-up of the integers computed along the way. Bit-length of integer determinant.

19.08 HW (14 points) Review the "Knapsack problem" handout. Assume all weights, values and the weight limit are positive integers. Prove that the dynamic programming algorithm described in the handout does NOT run in polynomial time.
Conceptual clarity is paramount. Give a clear statement of what it is that you need to demonstrate. The word "not" should not appear in your description.

19.10 REVIEW DMmini Chapters 6.1 (Graph Theory terminology), especially Terminology 6.1.42, and Chapter 6.4 (Digraph terminology).

19.15 HW (15 points, due March 4) We have learned that the cost of running Dijkstra, in terms of PRIORITY QUEUE operations, is $O(n\cdot\text{INSERT} + m\cdot\text{DECREASE-KEY} + n\cdot\text{EXTRACT-MIN})$. We refer to this expression as the "Dijkstra cost."
Recall that Dijkstra's algorithm requires all edge weights to be non-negative.
Recall that in an edge-weighted digraph, a **negative cycle** is a directed cycle whose edge weights add up to a negative number.
Consider the single-source min-weight path problem with one negative edge but no negative cycle:
INPUT: (V, E, s, w) where (V, E) is a digraph, $s \in V$ (the source [root]), $w : E \to \mathbb{R}$ is a weight functions where $w(e_0) < 0$ for some edge e_0 and $w(e) \geq 0$ for all $e \in E \setminus \{e_0\}$ and there is **no negative cycle**.
OUTPUT: array $[q(v) \mid v \in V]$ where $q(v)$ is the weight of the min-weight path from s to v; array $[p(v) \mid v \in V]$: array of parent links for a tree rooted at s such that a min-cost $s - v$ path for vertices v accessible from s can be found by tracing the parent links from v to s.
Prove: this problem can be solved at Dijkstra cost. The description of your algorithm should be very simple, with reference to Dijkstra's algorithm which you should use as a subroutine. When you invoke Dijkstra, make sure to clearly describe the input to which you are applying Dijkstra.
Warning: we are looking for a min-weight paths, not walks. So no vertex can be used more than once.
Update March 4, 0:30. The "no negative cycle" condition was added.

19.20 HW (9 points) Describe the modification(s) of the RELAX routine required for turning Dijkstra's algorithm into a min-weight spanning tree algorithm. Proof of correctness not required.

19.33 STUDY the "Greedy matching" handout, referred to as "GM" below.
The matching number is denoted $\nu(G)$ where ν is the Greek letter "nu", coded as \nu in (La)TeX.

19.35 DEF The **star graph** star_n is the complete bipartite graph $K_{1,n-1}$. It is a tree with one vertex adjacent to all others.

19.37 DO (a) Verify: the star graphs with $n \geq 2$ vertices have matching number $\nu(\text{star}_n) = 1$. (b) Find all graphs G with $\nu(G) = 1$ that are not star graphs.

19.41 HW (6+7+8 points) (part (c) is due March 13) (a) The greedy matching algorithm always produces a maximal matching. (b) If M is a maximal matching then $\nu(G) \leq 2|M|$. (c) Show that the bound in part (b) is tight in the following sense: Find infinitely many connected graphs $G = (V, E)$ along

with an ordering of their edges such that if the greedy algorithm produces the maximal matching M then $\nu(G) = 2|M|$. Give a very simple description of your graphs G, edge ordering, greedy matching M, and maximum matching in G.
Note that according to parts (a) and (b), the greedy algorithm produces a "(1/2)-approximation" of $\nu(G)$.
Update March 1, 5:30. Part (c) postponed to March 13. This is consistent with Assignment 8 on Gradescope; no change was made on Gradescope.

* * *

19.65 REVIEW Let G be a graph. Recall that $\alpha(G)$ denotes its independence number, $\omega(G) = \alpha(\overline{G})$ its clique number, and $\chi(G)$ its chromatic number. Recall that $\chi(G) \le 2$ if and only if G is bipartite. A **triangle-free** graph is a graph with $\omega(G) \le 2$. Note that all trees are bipartite. Note also that not all triangle-free graphs are bipartite. (Give a counterexample.)

19.68 HW (5 points) Prove: $\alpha(G) \cdot \chi(G) \ge n$.

19.71 DO (a) Prove: $\chi(G) \ge \omega(G)$. (b) Find the smallest graph with $\chi(G) > \omega(G)$.
By "smallest graph" we mean the one with the smallest number of vertices, and among them, the one with the smallest number of edges.
Hint: your graph will be triangle-free.

19.74 HW (7 points) Find the smallest graph with $\omega(G) = 3$ and $\chi(G) = 4$. Give a very simple description of your graph. Prove both claims (about $\omega(G)$ and $\chi(G)$). You do not need to prove that your graph is smallest.

19.76 CH Find the smallest triangle-free graph of chromatic number 4.
Hint. It has 11 vertices. (Do not look it up.)

19.79 CH Prove: for all $k \in \mathbb{N}$ there exists a triangle-free graph of chromatic number k. (Do not look it up.)

19.85 STUDY the "Greedy coloring" handout (GCol)

19.88 DO GCol Problem (a) (greedy coloring not too bad)

19.91 HW (11 points) GCol Problem (b) (greedy coloring is terrible)

19.94 HW (14 points, due March 4) GCol Problem (d) (linear-time implementation)

Go to top

Class #18, Mon, Feb 17

Material covered: Very fast growing functions: TOWER function, Ackermann function. Very slowly growing functions: $\log^*(n)$, inverse Ackerman function. UNION-FIND data structure, hierarchical implementations. Rooted tree in each block ("country"), parent links. Cost: UNION: $O(1)$, FIND: depth of deepest tree. UNION requires decision as to which "country" "wins." Goal: keep the trees shallow. Rule 1: biggest wins, Rule 2: deepest wins. Each rule guarantees depth $\le \log_2 n$. Additional algorithmic step: Collapse of trees after FIND. Amortized complexity of "deepest wins" rule with collapse: inverse Ackerman function (Robert Endre Tarjan). --- Proof of correctness of Kruskal's algorithm completed: reduction of the case of general weight function to the case of distinct weights (previously solved): perturbation method.

18.10 XX

18.10 XX

Go to top

Class #17, Fri, Feb 14

Material covered: Proof of correctness of Kruskal's algorithm assuming all weights are distinct. Alternative greedy approach: growing tree from one source: Jarnik's (Prim's) algorithm

17.10 XX

Go to top

Class #16, Wed, Feb 12

Material covered: "Free" trees, rooted trees. Spanning trees. Min-weight spanning tree of a graph. Greedy algorithm: Kruskal's algorithm. Implementation requires UNION-FIND data structure. Hierarchical implementation of the UNION-FIND data structure.

WARNING: The numbering of the problems below was changed on Feb 13 at 20:25.

16.11 HW (7 points) Recall that a digraph $G=(V,E)$ is **strongly connected** if $(\forall x,y\in V)(y$ is accessible from $x)$.
Given a digraph represented by an array of adjacency lists, determine in linear time whether G is strongly connected. (We work in the unit-cost model; "linear time" means $O(n+m)$ steps where n is the number of vertices and m is the number of edges.) Give a very simple pseudocode; you may use previously studied algorithms as subroutines. (Do not include their pseudocodes.)

* * *

16.20 DEF A **tree** is a connected, cycle-free graph.

16.23 DO Let $G=(V,E)$ be a graph. Prove that G is a tree if and only if for every pair $x,y\in V$ there is a unique $x\dots y$ path in G (i.e., there is one and only one path between x and y).

16.26 DO Prove that every tree is bipartite.

16.29 TERMINOLOGY Let $G=(V,E)$ be a graph. A **maximum path** in G is a longest path (a path of maximum length). A **maximal path** is a path in G that is not a subgraph of a longer path in G.

16.32 HW (5 points) Obviously, every maximum path is maximal. Show that the converse is false. Construct a smallest **connected** counterexample (connected graph, minimum number of vertices, and minimum number of edges among those with the minimum number of vertices). Draw the graph, and highlight two paths, one of them maximal but not maximum, the other maximum. You can draw by hand but include the drawing in your PDF file.
Updated 2-14 at 19:25: the word "connected" was added several times.

16.35 DO Prove: if a tree has $n \geq 2$ vertices then it has a vertex of degree 1. (Hint: the endpoints of any maximal path have degree 1.

16.38 DO A tree with n vertices has $n-1$ edges. (Hint: use the preceding exercise.)

16.42 DEF A **spanning tree** of a graph $G=(V,E)$ is a spanning subgraph $T=(V,F)$ that is a tree.

16.45 DO Prove: a graph G has a spanning tree if and only if G is connected.

16.48 HW (5 points) Describe a linear-time algorithm (in the unit cost model) that takes as input a graph G and either decides that G is disconnected or produces a spanning tree of G. Give a very short answer (a couple of lines) with reference previously studied algorithm(s).

16.51 DO Let G be a connected graph. Show that an edge e belongs to every spanning tree if and only if e does not belong to any cycle in G.

16.54 CH Given a connected graph, find, in linear time, every edge that does not belong to any cycle.

16.57 DEF The **min-weight spanning tree problem** takes as input a connected, edge-weighted graph (V,E,w) where $G=(V,E)$ is a connected graph and $w : E \to \mathbb{R}$ is a real-valued weight function. (The weights can be negative.) The output is a minimum-weight spanning tree. (The weight of a subgraph is the sum of the weights of its edges.)

The following **greedy algorithm** was designed by American mathematician Joseph Kruskal in 1956.

16.61 Kruskal's algorithm
Input: connected, edge-weighted graph (V,E,w)
Output: min-weight spanning tree (V,F)

```
01  sort the edges by weight:   $w(e_1) \leq \cdots \leq w(e_m)$
02  $F := \emptyset$       (: $F$ is a list that collects the edges of the spanning tree :)   (: intended loop invariant: "$(V,F)$ is a forest" :)
03  let $e_i = \{x_i, y_i\}$
04  for $i = 1$ to $m$
05          if $x_i$ and $y_i$ belong to different components of $(V,F)$ then add $e_i$ to $F$
06  end(for)
07  return $F$
```

16.64 Explanation A **greedy algorithm** makes a sequence of choices, choosing each time what seems best at the time. It does this without foresight, and it never reconsiders its choices. For most combinatorial optimization problems, this is a poor strategy, but for the minimum-cost spanning tree problem it produces optimal output.

16.67 DO Show that the statement "(V,F) is a forest" is a **loop invariant** for Kruskal's algorithm.

16.71 Reward problem (correctness) Prove that Kruskal's algorithm produces a minimum-weight spanning tree. (Hint. Solve this first for the case when all weights are distinct; show along the way that in this case the minimum-weight spanning tree is unique.)

16.74 The implementation of Kruskal's algorithm raises delicate **data structure** issues: how do we maintain the connected components of F. See UNION-FIND data structure below.

16.78 (pessimist's algorithm) We might call the greedy algorithm the "optimist's algorithm" ("looks good? grab it!"). The pessimist's algorithm ("looks bad? throw it away") goes as follows. Similar to the greedy algorithm, this algorithm also makes its choices sequentially, without foresight, and never reconsiders.
Input: connected, edge-weighted graph (V, E, w)
Output: min-weight spanning tree (V, F)

```
01    sort the edges by weight:  $w(e_1) \leq \cdots \leq w(e_m)$
02    $F := E$      (: $F$ is a list of edges from which we shall throw away the "bad" edges :)   (: intended loop invariant: "$(V,F)$ is connected" :)
03    (blank line, kept so numbering corresponds to Kruskal)
04    for $i = m$ downto $1$
05        if $(V, F \setminus \{e_i\})$ is connected then remove $e_i$ from $F$
06    end(for)
07    return $F$
```

16.81 Reward problem (correctness) Prove that the pessimist's algorithm returns a min-cost spanning tree.

16.84 CH Is it possible to implement the pessimist's algorithm in nearly linear time? (I do not know the answer.)

Go to top

Class #15, Mon, Feb 10

Material covered: Exponential growth. Simply exponential growth. Exponential growth beats polynomial growth. - Proving the correctness of algorithms: loop invariants.

* * *

EXPONENTIAL GROWTH

15.15 DEF (exponential growth) We say that the function $f : \mathbb{N} \to \mathbb{R}^+$ grows **at least exponentially** if there exist $C > 1, c > 0$ such $f(n) = \Omega(C^{n^c})$.
We say that f grows **at most exponentially** if there exist $C > 1, c > 0$ such $f(n) = O(C^{n^c})$.
If we say that f grows **exponentially** if it grows both at least and at most exponentially.
Example: $f(n) = \mathrm{e}^{\sqrt{n}} + (\sin n) \cdot n^{100}$ grows exponentially.
We say that $f(n)$ grows **at least simply exponentially** if there exists $C > 1$ such $f(n) = \Omega(C^n)$.
We say that $f(n)$ grows **at most simply exponentially** if there exists $C > 1$ such $f(n) = O(C^n)$.
We say that $f(n)$ grows **simply exponentially** if it grows both at least and at most simply exponentially.
Note. Remember that the tower of exponentials notation a^{b^c} (without parentheses) means $a^{(b^c)}$ and this is very different from $(a^b)^c = a^{bc}$.

15.18 HW (4+5 points) Show that $n!$ grows exponentially but not simply exponentially. State all valid values of the pair (C, c) (a) for the lower bound and (b) for the upper bound. Prove your answers.

15.21 HW (5 points) Find a function $f(n)$ such that $f(n)$ grows simply exponentially but there is no C such that $f(n) = \Theta(C^n)$. Give a very simple definition of your f. Prove.

15.24 DO $(\forall C > 1)(n = o(C^n))$. Do not use L'Hôpital's rule. Use only the fact that $(\ln x)/x \to 0$ as $x \to \infty$ and apply a simple substitution of variables.

15.28 DO (simply exponential growth beats polynomial growth) Prove: $(\forall D > 1)(\forall E)(x^E = o(D^x))$ as $x \to \infty$.
Do not use L'Hôpital's rule. Use 15.24 with a substitution of variables. (**Bootstrapping:** inferring a stronger statement from its weaker special case.)
Hint. Let $C := D^{1/E}$.

15.31 DO (exponential growth beats polynomial growth) Prove: $(\forall A > 1, c > 0)(\forall B)(y^B = o(A^{y^c}))$ as $y \to \infty$. Give a very simple proof. Do not use L'Hôpital's rule. Use the preceding exercise with a substitution of variables (bootstrapping). Clearly state your substitution.
Example: $y^{100} = o(1.001^{y^{0.001}})$.

* * *

15.41 STUDY the online handout "Loop invariants" (click "Texts" on the banner). Study especially exercises 6, 8, and 9 of this handout. These exercises constitute the proof of correctness of Dijkstra's algorithm.

15.44 HW (6+6 points) "Loop invariants" handout, Exercise 5(a)(b) ("Is R_2^* a loop invariant for Dijkstra?")

15.51 DO "The car-race problem" handout, problem (a) (optimal route can revisit the same location 100 times)

15.54 HW (7+7 points) "The car-race problem" handout, problems (b)(c). Give your simplest solution to each problem. Solving (c) does not give you credit for solving (b), but you can use your solution to (b) to describe your solution to (c).
To make sure you are viewing that latest version of this handout, check that the last sentence of Problem (c) begins with the phrase "Point out where you are using." If you don't see this phrase, you are looking at an earlier, cached version of this handout. If so, refresh your browser. If that does not help, please clear your browser's cache or try another browser.
Update: typo in handout Tue, Feb 18, 2:00 (am). Item (c) in the handout says "do not use an array with more than $|R|n$ cells." The correct bound is $O(|R|n)$ cells.

15.57 HW (14 points, due Feb 25) "The car-race problem" handout, problem (d). Please do not refer to your solution to the previous week's problems because looking up a previous solution is very time consuming for the graders. Describe a self-contained solution to this problem. (You can lift passages from your previous solution.)

* * *

15.65 STUDY the online handout "Amortized analysis."

15.68 HW (5+8 points) "Amortized analysis" handout, problems (a)(b) (Queue via stacks problem).

Go to top

Class #14, Fri, Feb 7

Material covered: What are the assumptions and conclusion of L'Hôpital's rule? $\ln x = o(x)$ as $x \to \infty$. Bootstrapping: $\ln x = o(x^\epsilon)$. Polynomially bounded functions. Bitlength of rational numbers. Polynomial-time algorithms. Dijkstra's algorithm for rational weights works in polynomial time. Given the positive integer N, the number 2^N cannot be computed in polynomial time because it has exponentially many digits.

14.10 REVIEW L'Hôpital's rule (one of several cases):
Assumptions:
(1) $f(x)$ and $g(x)$ are differentiable real functions on the interval (c,∞) for some $c \in \mathbb{R}$;
(2) $f(x) \to \infty$ and $g(x) \to \infty$ as $x \to \infty$;
(3) the (finite or infinite) limit $L := \lim_{x\to\infty} f'(x)/g'(x)$ exists.
(It is customary to add the assumption that $g'(x) \neq 0$ for all sufficiently large x, but this is implicit in (3).)
Conclusion:
The limit $M := \lim_{x\to\infty} f(x)/g(x)$ exists and $M = L$.
Updated 2-11 at 18:50: previously this item was erroneously numbered "14.20" instead of 14.10.

14.15 L'Hôpital's rule (continued) The same holds if $f(x) \to 0$ and $g(x) \to 0$. Moreover the conclusion in each case remains true if we replace the circumstance $x \to \infty$ everywhere by $x \to d^-$ (limit from the left) for some $d \in \mathbb{R}$ in which case $f(x)$ and $g(x)$ need to be differentiable on an interval of the form (t, d) for some $t < d$, or we replace $x \to \infty$ everywhere by $x \to d^+$ (limit from the right) in which case $f(x)$ and $g(x)$ need to be differentiable on an interval of the form (d, t) for some $t > d$.

14.19 HW (6 points) The 14.10 case of L'Hôpital's rule states that assuming conditions (1) and (2), if the limit L exists then the limit M also exists. Prove that the converse is false in the following sense: assuming (1), (2), and the existence of the limit M, it does NOT follow that the limit L exists, even if we assume that $g'(x) \neq 0$ for all sufficiently large x.

14.22 DO Prove: $\ln x = o(x)$ as $x \to \infty$. Use L'Hôpital's rule.

14.22 DO Prove: (a) $(\forall \epsilon > 0)(\ln x = o(x^\epsilon))$ as $x \to \infty$. Do NOT use L'Hôpital's rule. Just use the preceding exercise and an appropriate substitution of variables.
Bootstrapping is a method to turn a weak result into a much stronger result using simple manipulations. This exercise is an example of bootstrapping.
(b) $(\forall K, \epsilon > 0)((\ln x)^K = o(x^\epsilon))$ as $x \to \infty$. Do NOT L'Hôpital's rule. Just use the preceding exercise and

* * *

POLYNOMIAL GROWTH, POLYNOMIAL TIME

14.27 DEF Let $\mathbb{R}^+$ denote the set of positive real numbers. Let $T : \mathbb{N} \to \mathbb{R}+$ be a function. We say that T is **polynomially bounded** if $(\exists C \in \mathbb{R}^+)(T(n) = O(n^C)$. We say that such a C is a **valid exponent** for T.

14.28 DO Show that $n^2 \log n$ is polynomially bounded, and C is a valid exponent if and only if $C > 2$.

14.31 DO Let $T : \mathbb{N} \to [1,\infty)$ be a function. Show that T is polynomially bounded if and only if $\log_2 T(n) = O(\log n)$.

14.32 DO Show that $n^{\log n}$ is not polynomially bounded.

14.35 DEF (polynomial-time algorithm) Consider a computational task f that maps finite bit-strings to finite bit-strings. For $x \in \{0,1\}^n$ we write $|x| = n$ (the length of the bit-string x). Assume an algorithm $\mathcal{A}$ computes $f(x)$ using at most $T(|x|)$ bit-operations. If $T(n)$ is polynomially bounded, we say that $\mathcal{A}$ is a polynomial-time algorithm.

WARNING1: This concept does not apply to computational tasks where some of the input variables are real (because reals cannot be encoded as finite bit strings). In particular, Dijkstra's algorithm is not "polynomial time." However, it is polynomial-time if the edge weights are rational numbers (see below).
WARNING2: "Bit-operations" does not fully define the model. However, the notion of "polynomial time" is so robust, in all reasonable models of computation it has the same meaning. The range of models includes Turing machines on one end and Random Access Machines (RAMs) on the other. It does not include quantum computers. We will not give a rigorous definition of RAMs, but statements like "linear" or "quadratic" time refer to an intuitive understanding of that model.

14.37 DO Addition and subtraction of integers is "linear time" ($O(n)$, where n is the total number of input bits). The schoolbook method of multiplication of integers (given in binary) is polynomial-time, specifically, quadratic ($O(n^2)$). The Karatsuba method of multiplication of integers is also polynomial time, whose best valid exponent is $\log_2 3$.

14.39 DO (a) Given the positive integers a and m, the remainder $(a \bmod m)$ can be computed in quadratic time using the schoolbok method ("long division"). Note that in this problem, n (the input length) is the total number of bits of a and m. (b) Use Karatsuba's method to compute $(a \bmod m)$ in $O(n^\alpha)$ where $\alpha = \log_2 3$.

* * *

14.51 HW (6 points) Let $b(N)$ denote the number of binary digits of the positive integer N (so $b(N) = \lceil \log_2(N+1) \rceil$). Show that $b(F_k) \sim ck$ for some positive constant c. Determine c.

14.54 HW (5 points) Given k (in binary), show that F_k cannot be computed in polynomial time because the number of digits of F_k is exponentially large in terms of n, the bit-length of the input. (So $n = \lceil \log_2(k+1) \rceil$.) Specifically, show that the bit-length of the output is $\Theta(C^n)$ for some positive constant C. Determine C.

Go to top

Class #13, Wed, Feb 5

Material covered: Review of Dijkstra's algorithm. Cost in terms of the Priority Queue operations: $\le n_0$ calls of INSERT, $\le n_0$ calls of EXTRACT-MIN, and $\le m_0$ calls of DECREASE-KEY where n_0 and m_0 are the number of accessible vertices and edges, respectively. Total cost under HEAP implementation of PRIORITY QUEUE: $O((n+m)\log n)$. Amortized complexity, amortized cost of a sequence of Priority queue operations in the Fibonacci Heaps of Fredman and Tarjan: INSERT $O(1)$, EXTRACT-MIN $O(\log n)$, DECREASE-KEY $O(1)$. Total cost of Dijkstra implementation using Fibonacci heaps: $O(n\log n + m)$. This cost bound is optimal among Dijkstra implementations.

13.10 REVIEW "Dijkstra's algorithm" handout.

13.20 DO (optimality of Fibonacci heaps for Dijkstra - review from class) Prove: given n and $n-1 \le m \le n^2$, there exists a weighted rooted digraph $G(n,m)$ such that under any implementation of the Priority queue, the cost of Dijkstra on $G(n,m)$ will be $\Omega(n\log n + m)$.

13.41 DEF A **DAG (directed acyclic graph)** is a digraph with no directed cycles.

13.44 DO (a) Show that a DAG has at most $\binom{n}{2}$ edges. (b) Show that this bound is tight: for every n, find a DAG with n vertices and $\binom{n}{2}$ edges.

13.47 HW (7 points) Show that even on a DAG, Dijkstra fails if we allow a single negative-weight edge. Make your DAG as small as possible (minimize the number of edges). You do not need to prove that your DAG is smallest. Describe the progress of the algorithm: in each phase, indicate the associated arrays (current status, cost, parent link for each vertex).

Go to top

Class #12, Mon, Feb 3

Material covered: Single-source shortest path problem: BFS. Array representation of Heaps. Weighted, rooted digraphs (V, E, w, s) where (V, E) is a digraph, $w : E \to \mathbb{R}$ is a weight function, and s a "source" (root). Single-source min-weight path problem with non-negative weights solved by Dijkstra's algorithm usig a Priority Queue.

12.10 STUDY "Dijkstra's algorithm" handout.

12.20 REVIEW Priority Queue (abstract data structure), Heap.

Go to top

Class #11, Fri, Jan 31

Material covered: Counting sort. Radix sort. Digraphs. Modelling (undirected) graphs by digraphs. Outdegree, indegree. Directed path, directed cycle. Walk. Accessibility. Computational representation of digraphs: edge list, adjacency array (array of adjacency lists). Single pass scheme. Linear-time algorithms in the unit cost model. Strongly connected digraphs. Single source (root) shortest paths. Breadth-first search (BFS) using a queue (FIFO list)..

11.10 STUDY DMmini Chap. 6.4 "Digraph terminology," 6.4.1 -- 6.4.36.

11.13 CH DMmini 6.4.20 (if each vertex of a digraph G has the same indegree as its outdegree and G is weakly connected then G is strongly connected).

11.16 DEF (reverse digraph) Let $G = (V, E)$ be a digraph. The **reverse** of G is the digraph $G^- = (V, E^-)$ where $E^- = \{(y, x) \mid (x, y) \in E\}$. (The reverse of G is also called the "transpose" of G.)

11.20 STUDY the online handout "Breadth-first search." This handout was updated today (Jan 31). REFRESH your browser before downloading. The bottom line of the document should say "Last updated 01-31-2025." If you don't see this, please let the instructor know.

11.25 Representations of digraphs. We represent the set of vertices as an array, $[v_1, \ldots, v_n]$.
(a) Edge-list representation. We list the edges as a linked list.
(b) Adjacency-array representation. Each cell i in the array of vertices points to a linked list, $\text{adj}[v_i] = [w_1, w_2, \cdots, w_{k_i}]$, the **adjacency list** of v_i, where $k_i = \deg^+(i)$ is the outdegree of v_i and $w_1, \ldots, w_{k_i}$ are the out-neighbors of w_i, listed in any order.

Note that upon reading the number i we can immediately jump to vertex v_i in the array of vertices ("random access"), and then start reading (in order) the outneighbors of v_i. However, deciding whether v_j is an outneighbor of v_i may take k_i steps; we need to walk through all outneighbors of v_i before we can be sure that v_j is not among them.

11.28 HW (conversion between representations, 8+8 points) Let $G = (V, E)$ be a digraph ($E \subseteq V \times V$) where $V = [n] = \{1, \dots, n\}$.
(a) Let G be given in the edge-list representation. Convert this into an adjacency-array representation of G in linear time, $O(n + m)$ where $m = |E|$. Operations with numbers $i \in [n]$ (reading, copying) have unit cost. Describe your algorithm in simple pseudocode. Give a brief reason why your algorithm runs in linear time. You do not need to prove correctness.
(b) Let G be given in adjacency-array representation. Convert this into an edge-list representation of G in linear time, $O(n + m)$. The rules are the same as in part (a).

11.31 STUDY (reverse digraph, worked example) Given a digraph $G = (V, E)$ in adjacency-list representation, construct an adjacency-array representation of the **reverse** digraph $G^- = (V, E^-)$ (each edge reversed) in linear time. Describe your algorithm in a very simple, elegant pseudocode. Assume $V = [n]$.
Solution

INPUT: adj (: array of linked lists representing G :)

```
for $j = 1$ to $n$
   create the empty linked list $\mathrm{adj}^-[j]$    (: initializing the adjacency-array representation of $G^-$ :)
end(for)
for $i = 1$ to $n$
   for $j \in \mathrm{adj}[i]$                     (: visiting each out-neighbor of $i$ in the given order :)
     append $i$ to $\mathrm{adj}^-[j]$
   end(for)
end(for)
return $\mathrm{adj}^-$
```

The reason this algorithm works in linear time is that for every item in the input (adjacency arrays) it performs a constant number of operations (read/copy numbers $i, j \in [n]$, do random access using such a number as an address, advance on a link in a linked list).

2nd solution. Convert the adjacency array representation of G to edge-list representation. Then reverse in a single pass through the edge list. Finally, convert back to adj-array.

11.35 HW (monotone adjacency-lists, 9 points) We say that the adjacency list of a vertex is *monotone* if its neighbors are listed in increasing order. We say that an adjacency-array representation of a digraph is *monotone* if the adjacency list of each vertex is monotone. Given a digraph $G = (V, E)$ in adjacency-array representation, convert this to a monotone adjcency-array representation in linear time. Your algorithm should be very simple, just a couple of lines, with reference to previously seen algorithms. Explain in words what your algorithm is doing.
Briefly reason why your algorithm is correct. Briefly indicate why your algorithm runs in linear time. For the latter, take 11.31 as a model.

11.38 HW (undirectedness test, 8 points, due Feb 11) We say that a digraph G is **undirected** if $G = G^-$. Given a digraph in adjacency-array representation, decide in linear time whether it is undirected. Write a very simple pseudocode, with reference to previously seen algorithms. Explain in words what your algorithm is

doing.
You do not need to reason the correctness of your algorithm. Please add one sentence to reason that it runs in linear time. Take 11.31 as a model.

Go to top

Class #10, Wed, Jan 29

Material covered: Basic concepts of graph theory continued. Degree, Handshake Theorem. Accessibility: an equivalence relation. Walks. Equivalence relations and partitions, blocks of a partition. The Fundamental Theorem of Equivalence relations, equivalence classes. Connected components of a graph: the equivalence classes of the accessibility relation with the edges added within each class. Connected graph.

10.10 REVIEW 03.25 and 03.30 below (Fundamental Theorem of equivakence relations, equivalence classes).

10.15 DO Let x, y be vertices of the graph G. Assume there exists an $x \dots y$ walk in G. Prove that there exists an $x \dots y$ path in G. *Hint.* Show that a shortest $x \dots y$ walk is a path (if we disregard orientation).

Go to top

Class #9, Mon, Jan 27

Material covered: Basic concepts of graph theory. Adjacency relation. Isomorphism. Complement. Complete graphs, cycles, paths. Subgraphs.
All homework due **Tue, Feb 4**, unless expressly stated otherwise. (Remember that some of the problems from previous classes are also due Feb 4.)

09.07 STUDY DMmini Chap 6.1 "Graph theory terminology," including the concepts not discussed in class. **IMPORTANT.** Graph theory terminology in the literature is not uniform. Please use the terminology given in DMmini Chap 6.1. **Do not look up terminology from any other source.**

09.10 NOTATION Let V be a set and $k \in \mathbb{N}_0$. Then $\binom{V}{k}$ denotes the set of k-element subsets of V.
So $\left|\binom{V}{k}\right| = \binom{|V|}{k}$.

09.13 DO Solve DMmini Exercises 6.1.3--6.1.50. We highlight some of these below.

09.16 DO DMmini Ex. 6.1.6(c) (non-isomorphic graphs with the same number of vertices and the same number of edges).

09.18 HW (5 points, due Feb 11) DMmini 6.1.11(c) (number of vertices of a self-complementary graph is 0 or 1 mod 4)

09.19 Reward problem If $n \in \mathbb{N}$ and and $n \equiv 0$ or $1 \pmod 4$ then there exists a self-complementary graph with n vertices.

09.22 HW (6 points) DMmini Chap 6.1 Exercise 6.1.20 (count the spanning subgraphs of K_n with a given number of edges). Give a simple formula involving binomial coefficients. No proof required.

09.25 DO DMmini Ex. 6.1.23 (a graph is bipartite if and only if it has no odd cycle)

09.28 HW (16 points) DMmini Ex. 6.1.34 (draw all non-isomorphic 7-vertex trees). Lose 2 points if you fail to state how many you found. Lose 3 points for each mistake (missing a tree, duplicating a tree [two of your pictures are isomorphic], drawing a tree that does not belong, drawing a graph that is not a tree). You are not required to use LaTeX for this problem, you can draw your answer by hand. Make sure to give your name and state your sources/collaborations.

09.31 DO Define the $k \times \ell$ grid graph $\mathrm{grid}(k, \ell)$, naturally generalizing the picture of $\mathrm{grid}(4, 10)$ in DMmini, Figure 6.6, right before Ex. 6.1.46. Remember that $\mathrm{grid}(k, \ell)$ has $k\ell$ vertices.

09.34 DO Prove that $\mathrm{grid}(k, \ell)$ is bipartite.

09.37 HW (6 points) DMmini Ex. 6.1.49 (count the shortest paths between two opposite corners of $\mathrm{grid}(k, \ell)$). Give a simple formula involving a binomial coefficient. No proof required.

09.41 DO Prove: if $k, \ell \geq 2$ and k or ℓ is even then $\mathrm{grid}(k, \ell)$ is Hamiltonian (has a Hamilton cycle).

09.43 HW (7 points, due Feb 11) Prove: if both k and ℓ are odd then $\mathrm{grid}(k, \ell)$ is not Hamiltonian. Give a very simple proof. Remember: you can use lower-numbered exercises without proof. (State which exercises you are using.) Elegance matters; you lose points if your solution is more than three lines.

Go to top

Class #8, Fri, Jan 24

Material covered: MERGE-SORT reviewed. Recurrence, analysis. Data item: ID, keys, links. Abstract data structures defined by requests they serve. List (NEXT, LAST, FIRST), Queue (Enqueue, Dequeue: FIFO list), Stack (Push, Pop: FILO list), Priority Queue (INSERT, EXTRACT-MIN). Implementation: HEAP. Topology: complete binary tree. Depth: $d = \lfloor \log_2 n \rfloor$. HEAP condition. Repairing violation of HEAP condition: bubbling up/down. INSERT: bubbling up, $\leq d$ comparisons. EXTRACT-MIN: bubbling down, $\leq 2d$ comparisons. HEAP-SORT: $\leq 2n \log_2 n$ comparisons. HEAPIFY: $O(n)$ comparisons.

* * *

08.25 Linked lists Let $n \in \mathbb{N}$. Integers $i \in [n]$ will be referred to as *tokens*. They can be represented by binary strings of length $\lceil \log_2(n+1) \rceil$.
A linked list is a sequence of at most n *nodes*. Each node has an *address* (a token) and contains a fixed number of *keys* (objects of any kind, e.g., real numbers) and a fixed number of *tokens*. Two such tokens are mandatory: the *next* link (the address of the next node on the list) and *head* link (the address where the list begins). If we are at the end of the list then the "next" link leads to "NIL" (indicator that the list has ended). A *doubly linked list* also has a "prev" link to the preceding node and a "tail" link, pointing to the end of the list.

08.28 Linked list instructions We maintain the address of a "current node." At unit cost we can follow a link, updating the address of the current node. Further operations:

- $\mathrm{create}(L)$ -- creates an empty list L
- update key in current node
- update link in current node

- $\mathrm{APPEND}(L, t)$ where L is a list and t is a key or list of keys; this ooperation adds a new node at the end of the list. The new node includes the list t of keys and the required links
- next-node (follow the "next" link, update "current node")
- previous node
- DELETE: delete current node. This is accomplished by updating a single link: next(prev):=next(current), and updating the current node to *prev*
- INSERT node before/after current node (update links)

In the **unit cost model**, each of these operations incur unit cost.

* * *

Go to top

Class #7, Wed, Jan 22

Material covered: The decision tree model. Information theory lower bound. Sorting by comparisons. Information theoretic lower bound $\log(n!) \sim n\log_2 n$. Matching asymptotic upper bound in the comparison model where bookkeeping is free: Insertion Sort. Basic **data structures:** arrays, linked lists. Unit cost model: unit cost of arithmetic on indices in an array and copying links in a linked list. Binary search can be performed in an array but not in a linked list. Insertion is $O(1)$ in a linked list but $\Omega(n)$ in an array. FIND is $\Omega(n)$ in a linked list. Insertion sort incurs quadratic cost in each of these models. MERGE-SORT: asymptotically optimal number of comparisons ($\sim n\log_2 n$) and optimal order of magnitude of the bookkeeping cost ($O(n\log_2 n)$). Evaluation of the corresponding recurrences.

All HW problems **due Jan 28** unless expressly stated otherwise.

07.10 DO (very important) solve problems P2.06--P2.47 below (Problem session #2) (click #P2 on the banner)

07.15 HW (6 points) Prove: $(\forall n \in \mathbb{N})(n! > (n/\mathrm{e})^n)$. Give a very simple proof (2 lines). Use the power-series expansion of e^x.

07.18 DO Use the preceding exercise to show that $\log_2(n!) \sim n\log_2 n$. Do not use Stirling's formula.

07.25 HW (7 points) (MERGE) Let $A[1..m]$ and $B[1..n]$ be two sorted lists of real numbers: $A[1] \le A[2] \le \dots \le A[m]$ and $B[1] \le B[2] \le \dots \le B[n]$. Given these lists as inputs, create the merged list $C[1..m+n]$ where $C[1] \le C[2] \le \dots \le C[m+n]$.
Example. $A = [2,4,4,6]$, $B = [1,3,4,5,6,9]$, $C = [1,2,4,4,4,5,6,6,9]$.
Merge the lists making at most $m+n-1$ comparisons. (In this problem, the only thing we can do with real numbers is comparing them. All else is bookkeeping (recording the outcomes of the comparisons) and computation with integers $1,\dots,m+n-1$, all of which comes for free, we only count comparisons.) Write a simple pseudocode. Prove that you are using at most $m+n-1$ comparisons. Proof of correctness not required.

07.28 HW (14 points, due Tue, Feb 11) (MERGE optimal) Let $m = n$ in the preceding exercise. Prove that the algorithm of the preceding exercise is optimal in this case: $2n-2$ comparisons are not sufficient to create the merged list.
Warning: this is not about a particular method of merging; we are up against all conceivable methods.

Conceptual clarity is paramount.

07.31 HW (16 points, due Tue, Feb 11) (MERGE not optimal) Show that the two lists of Ex. 07.25 can be merged using $m \cdot \lceil \log_2(n+1) \rceil$ comparisons. Describe your algorithm in elegant pseudocode. Do not ignore rounding and do not ignore the possibility that some of the entries can be equal. You need to justify the exact upper bound claimed.
Note that if $m = o(n/\log_2 n)$ then this is much faster than the algorithm of Ex. 07.25.

07.34 (MERGE-SORT) This is a Divide-and-Conquer algorithm. It takes as input an array of n reals, $A[1,\dots,n]$, and produces their sorted array $B[1,\dots,n]$ $(B[1] \le B[2] \le \dots \le B[n])$.
Example: $A = [7,4,1,9,5]$, $B = [1,4,5,7,9]$.
The algorithm proceeds as follows:

```
Input: array $A[r, r+1, \dots, s]$ of $s-r+1$ reals

if $s = r$ then return $A$    //* ($A$ is sorted)
else  $A_1 := A[r,\dots,\lfloor (r+s)/2 \rfloor]$ and $A_2 := A[\lfloor (r+s)/2 \rfloor + 1,\dots,s]$.   //* splitting $A$ in half
  return $B := MERGE(SORT[A_1], SORT[A_2])$   //* recursively sort the half-size lists, then merge the results
```

Analysis. Let $R(n)$ denote the number of comparisons performed by the MERGE-SORT algorithm and let $T(n)$ denote the total cost, including bookkeeping. Then we have
(*) $\quad R(1) = 0 \quad$ and for $n \ge 2 \quad R(n) \le R(\lfloor n/2 \rfloor) + R(\lceil n/2 \rceil) + (n-1)$

and

(**) $\quad T(1) = 0 \quad$ and for $n \ge 2 \quad T(n) \le T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$

07.37 HW/CH Use the method of reverse inequalities to prove: $R(n) \le n \log_2 n$ **(a) HW (8 points)** for $n = 2^k$ $(k \in \mathbb{N}_0)$ **(b) CH** for all $n \in \mathbb{N}$.

07.39 HW (12 points, due Tue, Feb 4) Assume $T(n)$ is monotone non-decreasing. Use the method of reverse inequalities to prove $T(n) = O(n \log_2 n)$ $(n \in \mathbb{N})$.
Hint. First prove this for $n = 2^k$ $(k \in \mathbb{N}_0)$. Then reduce the general case to this special case.

* * *

07.51 DO (ternary decision tree) In a ternary decision tree, each query has 3 possible answers. Prove that to select an object out of N possible objects, the decision tree must have depth $\ge \log_3 N$ (i.e., we need to ask at least $\log_3 N$ questions in the worst case to identify the selected object).

07.54 DO/Reward (12 coins) (a) DO Given 12 coins, one of which is fake, we want to find the fake coin and also decide whether it is heavier or lighter. Show that this can be done with 3 measurements on a balance. (The genuine coins all have the same weight; the fake coin has a different weight. A balance has two trays. You can put any number of coins on each tray. The outcome of a measurement is "Left heavy," "Right heavy" or "Equal.") **(b) REWARD PROBLEM** Find an *oblivious* algorithm for this problem, i.e., specify your measurements (subsets of coins to put in each tray) in advance (your choice of which coins to put in each tray must not depend on the outcomes of previous measurements). Show that 3 measurements still suffice. (Do NOT hand in your solution. But do let me know if your proof of correctness is really elegant, not based on looking at several cases.)

07.57 HW (more than 12 coins) Given n coins, one of which is fake, we want to find the fake coin and also decide whether it is heavier or lighter, using measurements on a balance.
(a) (5 points) Prove: if $n = 14$ then 3 measurements do not suffice. Make your proof very simple.
(b) (13 points) Prove: if $n = 13$ then 3 measurements do not suffice.
Solving (b) will NOT earn you the 5-point credit for (a). You need to give a very simple separate solution to (a).

07.61 HW (15 points, due Tue, Feb 4) (Evenly splitting the fake coins): Suppose we have $2n$ coins, some of which are fake. All the fake coins have equal weight, and they are lighter than the genuine coins (which also have equal weight). Using $\sim \log_2 n$ measurements on a balance, divide the coins into two sets of n coins of nearly equal weight. "Nearly equal" means if the number of fake coins is even, they must be evenly split between the two parts; if their number is odd, one part must have exactly one more fake coin than the other part. The number of measurements made must be asymptotically equal to $\log_2 n$.
Write your algorithm in elegant pseudocode. Explain your variables.

07.64 HW (5+9 points, due Tue, Feb 4) (Find min) Given n real numbers, we want to determine their *min* in the comparison model. (Comparisons cost one unit each, bookkeeping is free.) Prove for every n: **(a)** $n-1$ comparisons suffice. Give a simple pseudocode. Explain your variables. **(b)** $n-2$ comparisons do not suffice.

07.67 HW (10 points) (Find min and max, due Tue, Feb 4) Given $n \geq 1$ real numbers, we want to determine both their *min* and their *max* in the comparison model. Prove: $\lfloor 3n/2 \rfloor$ comparisons suffice. (Note that this is rather surprising; how can determining the *min* reduce the cost of determining the *max*?)

* * *

07.71 HW (7+7+7+4 points) (Strassen's matrix multiplication): If we multiply two $n \times n$ real matrices according to the textbook definition, we need to perform n^3 multiplications of real numbers and $n^2(n-1) = \Theta(n^3)$ additional additions of real numbers. In 1969, Volker Strassen found that this was not optimal: in analogy with Karatsuba's algorithm to multiply polynomials, Strassen reduced the multiplication of two $n \times n$ matrices to the multiplication of 7 pairs of $n/2 \times n/2$ matrices (as opposed to 8 pairs, which would be the obvious way, mimicking the textbook multiplication of 2×2 matrices), plus a constant number of additions and subtractions of such matrices and some bookkeeping. The cost of the reduction is $O(n^2)$ (it involves a fixed number of additions and subtractions of $n/2 \times n/2$ matrices and some bookkeeping).
Here and below we assume n **is a power of** 2.
(a) First consider the model where we charge one unit cost for the multiplication of two real numbers; addition, subtraction, and bookkeeping are free. Let $M(n)$ denote the cost of multiplying two $n \times n$ matrices in this model. Observe that, according to Strassen, $M(n) \leq 7M(n/2)$. Moreover, $M(1) = 1$. **(a1)** Prove: $M(n) \leq n^\beta$ where $\beta = \log_2 7 \approx 2.807$. Use direct calculation and induction; do not use the method of reverse inequalities. In particular, you cannot assume that the upper bound has the form $M(n) \leq n^c$ for some c. **(a2)** Prove also that the inequality $M(n) < n^\beta$ does NOT follow for any value of $n = 2^k$ from the recurrence and the initial condition given. Clarify the LOGIC of this statement: state, what exactly needs to be shown.
(b1) Now, factor in the cost of additions, subtractions, and bookkeeping. State the recurrent inequality for the complexity $T(n)$ under this cost model (number of arithmetic operations on real numbers and bookkeeping). Use the big-Oh notation to estimate the cost of additions, subtractions, bookkeeping.
(b2) Conclude that $T(n) = O(n^\beta)$ using a previous exercise solved using the method of reverse inequalities. Which exercise?

Go to top

Problem session #2, Fri, Jan 17

Material covered: Conceptual midunderstandings regarding limits and asymptotic equality in the light of solutions submitted to the problem "logarithm of asymptotic equality" (ASY 6.5ab). The problems about asymptotic equality in the P2 sequence below are modeled after those issues.

P2.06 Review ASY Sections 1--6.

P2.08 Convention In all exercises in the P2 sequence, (a_n), (b_n), etc., are sequences of reals.

P2.10 Terminology: "asymptotically equal" (not "asymptotically equivalent")

P2.13 Terminology: the limit of a sequence **IS** L (as opposed to "the limit *goes to* L"). Notation: $\lim_{n\to\infty} a_n = L$. The limit of a sequence is a number or $\pm\infty$, it does not "go" anywhere. We also say that the terms of the sequence **go to** L or **approach** L. Notation: $a_n \to \infty$. The statement $\lim_{n\to\infty} a_n = L$ is synonymous with the statement $a_n \to L$. Never write "$\lim_{n\to\infty} a_n \to L$," this is meaningless.

P2.16 DO Explain why the statement "$(\forall n)(a_n \to L)$" makes no sense. (See the answer at end of this sequence of exerises.)

P2.19 DO Explain why the statement "$a_n \to L$ for large n" makes no sense and reflects a lack of understanding of the concept of limits and the basics of logic, especially the meaning of quantifiers. (The answer is the same as to the preceding exercise.) (Note that the statement "$a_n \to L$" has the same meaning as the statement "$a_k \to L$." Now what would you think of the statement "$a_k \to \infty$ for large n"?)

P2.22 DO Explain why the statement "$(\forall n)(a_n \sim b_n)$" makes no sense.

P2.25 DO Explain why the statement "$a_n \sim b_n$ for large n" makes no sense.

P2.27 Terminology Let $P(n)$ be a sequence of T/F statements. We say that $P(n)$ is **eventually true** if $P(n)$ is true for all sufficiently large n, i.e., if $(\exists n_0)(\forall n > n_0)(P(n)$ is true$)$.
Example. Let (a_n) be a sequence of reals. Then we say that "a_n is eventually zero" if $a_n = 0$ for all sufficiently large n.

P2.29 DO Show that $a_n \to 0$ DOES NOT mean that (a_n) is eventually zero.

P2.31 DO Say in simple, plain English, understandable to the educated person in the street, that "a_n is NOT eventually non-zero." (Find the answer at the end of this sequence of exercises.)

P2.34 DO Assume $(\forall n)(a_n \ge 1.01)$ and $a_n \sim b_n$. Show that (a) it DOES NOT follow from these assumptions that b_n is eventually ≥ 1.01, but (b) it DOES follow that b_n is eventually ≥ 1.00999.

P2.37 TERMINOLOGY Let (a_n) be a sequence of reals such that $(\forall n)(a_n > 1)$. We say that the sequence (a_n) is **bounded away** from 1 if there exists a positive constant c such that eventually $a_n \ge 1 + c$.

P2.41 DO Assume $(\forall n)(a_n > 1)$ and $a_n \sim b_n$. Prove: if (a_n) is bounded away from 1 then (b_n) is also bounded away from 1.

P2.44 DO Assume $a_n \sim b_n$. Show that it does NOT follow that $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$. You need to give a counterexample, i.e., two sequences, a_n and b_n, that are asymptotically equal but for which $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$ does not hold. Note that this does NOT mean that $\lim_{n\to\infty} a_n \neq \lim_{n\to\infty} b_n$ holds. Why? How is this even possible? Isn't "$\neq$" the negation of equality? See the answer at the end of this sequence of exercises.

P2.47 DO Assume $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n \neq 0$. In particular, we are making the assumption that these limits exist. Show that it does NOT follow that $a_n \sim b_n$. (See a hint at the end of this sequence of exercises.)

P2.51 DO Assume $c_n \to 0$ and $(\forall n)(d_n > 0)$. (a) Show that it DOES NOT follow that $c_n/d_n \to 0$. (b) Show that $c_n/d_n \to 0$ DOES follow under the stronger assumption that d_n is bounded away from zero.

P2.54 DO Assume $(\forall n)(a_n, b_n > 0)$ and $c_n \to 0$. Assume further that $a_n \sim b_n$. (a) Show that it DOES NOT follow that $\lim_{n\to\infty} a_n/(b_n + c_n) = \lim_{n\to\infty} a_n/b_n$. Show that this conclusion DOES follow if b_n is bounded away from zero.

Answer to P2.16 Because "$a_n \to L$" is a statement about the entire sequence (a_n) and not about its individual terms.

Answer to P2.31 "a_n is infinitely often zero."

Answer to P2.44 Because these limits may not exist. Let (a_n) be a sequence of nonzero numbers without a limit. Then $a_n \sim a_n$ but the satement $\lim_{n\to\infty} a_n = \lim_{n\to\infty} a_n$ is false.

Hint to P2.47 Let the limit be ∞.

Go to top

Class #6, Fri, Jan 17

Material covered: Dynamic programming: the Knapsack problem.

06.10 STUDY the online handout "Knapsack problem" (click "Handouts" on the banner)

06.15 HW (14 points) Solve the Problem stated in the online handout "All-ones square: a dynamic programming example" (click "Handouts" on the banner)

Go to top

Class #5, Wed, Jan 15

Material covered: Asymptotic notation: big-Oh, big-Omega. Evaluation of recurrent inequalities: the Method of Reverse Inequalities. Rigorous analysis of the Karatsuba algorithm.

05.10 STUDY the online handout "Evaluation of recurrent inequalties: The method of reverse inequalities." (Click "Handouts" on the banner.)

05.15 HW (Optimality of our analysis of the Karatsuba algorithm) (9 points) Let the function $S : \mathbb{N} \to \mathbb{N}$ be monotone non-decreasing, i.e., $(\forall n \in \mathbb{N})(S(n+1) \geq S(n))$. Assume that

$(\forall n \in \mathbb{N})(S(n) > 0)$ and $S(n) \le 3S(n/2) + O(n)$ for every $n \ge 2$ that is a power of 2. From this information we derived that $S(n) = O(n^\alpha)$ where $\alpha = \log_2 3 \approx 1.58$.
Prove that $S(n) = o(n^\alpha)$ does NOT follow from the same information (namely, from the recurrent inequality given and the positivity of $S(n)$). Most of the credit goes for the simplicity and **conceptual clarity** of your answer. Clearly state what exactly needs to be proved.
(For the little-oh notation used in this exercise, see 4.16 below.)
Updated 01-21 7:40. The assumption that S is monotone non-decreasing has been added.

05.18 HW (12 points) Let $T(n)$ denote the cost of a Divide-and-Conquer algorithm on inputs of size n, so $T(n) > 0$ for all $n \in \mathbb{N}$. Assume that the function $T(n)$ is monotone non-decreasing, i.e., $(\forall n \in \mathbb{N})(T(n+1) \ge T(n))$. Assume further that for all $n \ge 2$ we have $T(n) \le 7 \cdot T(\lceil n/2 \rceil) + O(n^2)$.
Prove that $T(n) = O(n^\beta)$ for some constant β. Find the smallest value of β for which this conclusion follows from the assumptions. You don't need to prove that your β is optimal.
For a real number x, the "ceiling" notation $\lceil x \rceil$ denotes the smallest integer k such that $x \le k$. For example, $\lceil 3.4 \rceil = 4$, $\lceil 3 \rceil = 3$, and $\lceil -3.4 \rceil = -3$.
Solving the problem with n restricted to powers of 2 earns you **8 points.** Elegance matters.

05.22 HW (5 points) Let N be a k-bit positive integer. Prove: $k = \lceil \log_2(N+1) \rceil$. Elegance matters.

05.25 DO Let (a_n) and (b_n) be sequences of positive numbers such that the limit $L = \lim_{n\to\infty} a_n/b_n$ exists and $0 < L < \infty$. Prove: $a_n = \Theta(b_n)$.

05.28 HW (4 points) Find two sequences, (a_n) and (b_n), of positive numbers such that $a_n = \Theta(b_n)$ but the limit $\lim_{n\to\infty} a_n/b_n$ does not exist. No proof required.

05.31 STUDY **congruences** from Sec. 5 of the online handout "Euclid's algorithm and multiplicative inverse."

05.34 STUDY the "Knapsack problem" online handout to get clarity on the notion of **pseudocodes** we use in this class.

05.37 REVIEW the **Communication Complexity** model (Class #1: click "#1" on the banner).

05.41 HW (18 points) (Communication Complexity) Alice has access to an n-bit integer $X = \overline{x_0 x_1 \dots x_{n-1}}$, Bob has access to an n-bit integer $Y = \overline{y_0 y_1 \dots y_{n-1}}$. (Initial zeros are permitted; $x_i, y_i \in \{0,1\}$.) Alice and Bob share a k-bit positive integer q such that $X \not\equiv Y \pmod{q}$. (This fact is known to both of them, they don't need to test it.) The numbers k and q are known to both of them. The task before Alice and Bob is to find i such that $x_i \ne y_i$. Show that they can accomplish this with no more than $\lceil \log_2 n \rceil \cdot (k+1)$ bits of communication. Describe the protocol in **pseudocode**.
Your protocol should be deterministic. If you do not include a correct pseudocode, just describe the procedure in clear and unambiguous language, you lose 6 points. If you fail to prove the correctness of your procedure, you lose 5 points.
To **typeset congruences in LaTeX**, here is an example: the LaTeX code for the equation "$X \not\equiv Y \pmod{q}$" is "`X \not\equiv Y \pmod{q}`".
Updated 01-17 18:40. Previously the communication complexity bound was erroneously stated as $\lceil \log_2 n \rceil \cdot k$.

Go to top

Class #4, Mon, Jan 13

Material covered: Polynomials, degree. Degree of the zero polynomial. Divide and Conquer: the Karatsuba method of multiplication of polynomials.

04.10 STUDY the online handout "Divide and Conquer: the Karatsuba algorithm." (Click "Handouts" on the banner.)

04.13 STUDY the online handout "Binary search." (Click "Handouts" on the banner.)

04.16 STUDY DMmini Sections 2.3 and 2.4 (asymptotic notation: little-oh, little-omega, big-Oh, big-Omega, big-Theta)

Go to top

Class #3, Fri, Jan 10

All solutions are due Tue, Jan 14, by 23:00, unless expressly stated otherwise.

Material covered: Analysis of the error probability of the randomized RYS protocol for the identity relation $(X \overset{?}{=} Y)$. Asymptotic equality of sequences. Relations, equivalence relations.

03.15 STUDY ASY Sections 1--6.

03.20 DO Solve all exercises from ASY Sec 1-6.

03.25 STUDY Functions, relations, equivalence relations, partitions: REAS22, classes 6 and 7.

03.30 DO Solve REAS22 6.10-6.157 and REAS22 7.10--7.57, especially 7.45 (proof of the Fundamental Theorem of Equivalence Relations).

03.35 HW (5 points) ASY 3.28 (limit of quotients of Fibonacci numbers assuming the limit exists)

03.38 (a)(b) HW (4+4 points) ASY 4.26(a)(b) (sum of limsup's vs limsup of sum)

03.41 HW (8 points) ASY 5.8d (asymptotics of $\sqrt{n^2+1}-n$)

03.44 HW (6 points) ASY 5.12 (asymptotics of middle binomial coefficient)

03.47(a)(b) HW (7+9 points) ASY 6.5(a)(b) (taking the logarithm of asymptotic equality)

03.50 HW (5 points) For the positive integer m, let $\nu(m)$ denote the number of distinct prime divisors of m. (Examples: $\nu(54)=2$, $\nu(60)=3$, $\nu(1)=0$.)
Prove: $\nu(m) \le \log_2(m)$.

03.55 CH Let p_n denote the n-th prime number. So $p_1=2$, $p_2=3$, $p_3=5, \dots$, $p_{25}=97, \dots$. Consider the statement

$$p_n \sim n \ln n\,.$$

Prove that this statement is equivalent to the PNT **(Prime Number Theorem).**

03.58 CH For the real number x let $P(x)$ denote the product of all primes $p \le x$. For instance, $P(9.73) = 2 \cdot 3 \cdot 5 \cdot 7 = 210$ and $P(1.99) = 1$ (why?). Consider the statement

$\ln P(x) \sim x$ (as $x \to \infty$).

Prove that this statement is equivalent to the PNT (Prime Number Theorem).

03.62 Notation For a real number x let $\nu^*(x) = \max_{k \le x} \nu(k)$. For instance, $\nu^*(372.6) = 4$ because $\nu(210) = 4$ and for all integers $k < 2310$ we have $\nu(k) \le 4$. (Why?)
Updated 01-14 19:40. Previously "2320" was written instead of 2310 and the last part of the last sentence erroneously stated that "for all integers $k < 2320$ other than $k = 210$ we have $\nu(k) \le 3$." This is false, for instance, $\nu(420) = 4$ and $\nu(2310) = 5$. The conclusion that $\nu^*(372.6) = 4$ remains valid.

03.65 CH Prove:

$$\nu^*(x) \sim \frac{\ln(x)}{\ln\ln(x)} \,.$$

You may use the PNT and all exercises above.

03.69 CH For all $x \ge 0$ prove $P(x) \le 4^x$. Your proof should be elementary; in particular, do not use any form of the PNT. The proof should not be longer than one page. If you get stuck, you may ask the instructor for a hint.

03.72 CH Assume the prime power p^t divides the binomial coefficient $\binom{n}{k}$. Prove: $p^t \le n$.

03.76 CH (weak PNT, Chebyshev, 1850) There exist positive constants c and C such that for all sufficiently large x we have $cx/\ln(x) \le \pi(x) \le Cx/\ln x$. In other words, $\pi(x) = \Theta(x/\ln x)$. (See the Θ notation in DMmini, Section 2.4.)
Give an elementary proof of this result (do not use the much stronger PNT in any form). Use problems 3.69 and 3.72 above. (The proof sketched here was found by Paul Erdős around 1934 while he was an undergraduate in Budapest.)

03.79 Remark. Chebyshev obtained constants c, C that were quite close to 1 $(c = 0.921, C = 1.106)$. From this result he derived (1852) **Bertrand's postulate** that for every positive integer n there is a prime number between n and $2n$.

03.82 Remark. ***Pafnuty Lvovich Chebyshev*** (1821-1894), the founding father of Russian mathematics, also proved a couple of years earlier (1848) that if the limit $\lim_{x\to\infty} \dfrac{\pi(x)}{x/\ln(x)}$ exists, it must be 1 (i.e., proving the convergence of this quotient would suffice to prove the PNT).

03.85 Remark. The PNT was proved half a century later (1896) independently by French mathematician ***Jacques Salomon Hadamard*** (1865-1963) and Belgian mathematician ***Charles Jean de la Vallée Poussin*** (1866-1962). Both of them used complex analysis, namely, Riemann's zeta function, following the ideas of German mathematician ***Bernhard Riemann's*** revolutionary 1859 memoir, the only paper Riemann had published about the distribution of prime numbers. A few years earlier Riemann introduced Riemannian manifolds that would 60 years later turn out to describe the geometry of the universe, forming the mathematical foundation of Einstein's general relativity theory. Riemann (1826-1866), regarded as one of the greatest innovators of mathematics, died of tuberculosis at the age of 39. At the time of his death, his housekeeper discarded his unpublished manuscripts, in all likelihood an immeasurable loss to mathematics.

Go to top

Class #2, Wed, Jan 8

Material covered: Randomized algorithms. Amplification of success probability. Worst-case analysis of randomized algorithms. Randomized communication complexity of the identity function: the Yao-Rabin-Simon protocol. Asymptotic equality. The Prime Number Theorem. Analysis of the YRS protocol.

Go to top

Class #1, Mon, Jan 6

Material covered: Model of computation. Computations task. Decision problem. Cost function. Size of input. Worst-case complexity of an algorithm, worst-case complexity of a computational task. Analysis of algorithms: correctness, cost. --- The communication complexity model. The Mehlhorn--Schmidt Theorem: lower bound on communication complexity in terms of matrix rank. Communication complexity of the identity function.

01.05 DEF A **computational task** is a function $f : D \to K$ mapping a domain D to a codomain K. We call D the set of *inputs* and K the set of potential *outputs*. We say that f is a **decision problem** if $K = \{0, 1\}$.

01.08 DEF Let $\mathbb{N}_0$ denote the set of non-negative integers. With each input x we associate a **size** $|x|$ which is a non-negative integer, so the size is a function $D \to \mathbb{N}_0$. An **algorithm** for f is a method to compute $f(x)$ given an input $x \in D$. The **model of computation** specifies the types of algorithms we consider and assigns a **cost** $C(\mathcal{A}, x) \in \mathbb{N}_0$ to each algorith $\mathcal{A}$ and input x.

01.11 DEF The **worst-case complexity of the algorithm** $\mathcal{A}$ is a function $C_{\mathcal{A}} : \mathbb{N}_0 \to \mathbb{N}_0$ where $C_{\mathcal{A}}(n) = \max_{x\in D, |x|=n} C(\mathcal{A}, x)$.
The **worst-case complexity of the computational task** f is a function $C_f : \mathbb{N}_0 \to \mathbb{N}_0$ where

$$C_f(n) = \min_{\mathcal{A}} C_{\mathcal{A}}(n) = \min_{\mathcal{A}} \max_{x\in D, |x|=n} C(\mathcal{A}, n)$$

where the minimum is taken over all algorithms $\mathcal{A}$ that compute f.

01.14 DEF Let $\mathcal{A}(x)$ denote the output produced by the algorithm $\mathcal{A}$ on input x. The **worst-case analysis** of the algorithm $\mathcal{A}$ for the computational task f in the given model of computation has two parts: (a) **correctness:** we need to verify that $(\forall x \in D)(\mathcal{A}(x) = f(x))$ (b) **worst-case complexity:** we need to compute or estimate, for all $n \in \mathbb{N}_0$ (or for all sufficiently large $n \in \mathbb{N}_0$), the complexity $C_{\mathcal{A}}(n)$.

* * *

The COMMUNICATION COMPLEXITY model (added 1-15)

01.20 DEF The **communication complexity** model (Andrew Chi-Chih Yao 1979: discrete model, a variation of a continuous model by Harold Abelson 1978) In this model, the domain D of a computational task $f : D \to K$ is split as a Cartesian product $D = A \times B$. Alice and Bob are two computing agents with unlimited computational power; any computation they perform has zero cost. The function f is known to both agents. They collaboratively wish to evaluate the function f at input (X, Y) where $X \in A$ is known to

Alice and $Y \in B$ is known to Bob. They need to communicate with each other because each of them misses part of the input, and they have to pay one unit for each bit communicated. So the cost is the number of bits communicated. The goal is for one of them to find the value $f(X,Y)$. They take turns communicating bit strings to the other. In advance they agree on a **communication protocol** (algorithm) that determines what bit-string each of them sends to the other on their turn; this bit-string is determined by the history of the communication and the input available to the communicating party. (This is similar to a bidding system in the card game of Bridge.)
We write $CC(f)$ to denote the **communication complexity** of the function f, i.e., the number of bits communicated under the best protocol for the worst input (X,Y).

01.23 DO (trivial upper bound) Assume $A = B = \{0,1\}^n$ and $K = \{0,1\}$. Then $CC(f) \le n$ for any function $f : A \times B \to \{0,1\}$. This is achieved by the trivial protocol: Alice sends the entire string X to Bob. Bob then has the entire input (X,Y) and computes $f(X,Y)$ at no cost.

01.26 DEF The **communication matrix** M_f corresponding to the function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is the $2^n \times 2^n$ $(0,1)$-matrix (each entry is 0 or 1) whose rows and columns are labeled by the $(0,1)$-strings of length n (there are 2^n such strings) and the entry corresponding to row X and column Y is $f(X,Y)$.

01.30 Theorem (Kurt Mehlhorn and Erik M. Schmidt, 1982)

$$CC(f) \ge \log_2 \operatorname{rk}(M_f)$$

01.34 DO Consider the task for Alice and Bob to decide whether $X = Y$ where $X, Y \in \{0,1\}^n$. We call this the **identity function** and denote it by id_n:

$$\mathrm{id}_n(X,Y) = \begin{cases} 1 & \text{if } X = Y \\ 0 & \text{if } X \ne Y \end{cases}$$

Prove: $CC(\mathrm{id}_n) = n$.
Proof. The communication matrix M_{id_n} corresponding to this task is the $2^n \times 2^n$ identity matrix. The rank of this matrix is 2^n, therefore $CC(\mathrm{id}_n) \ge \log_2 2^n = n$. On the other hand we know that $CC(f) \le n$ for all functions. □

* * *

View the instructor's class material from previous years

Return to the Department of Computer Science home page

Course home

Go to top