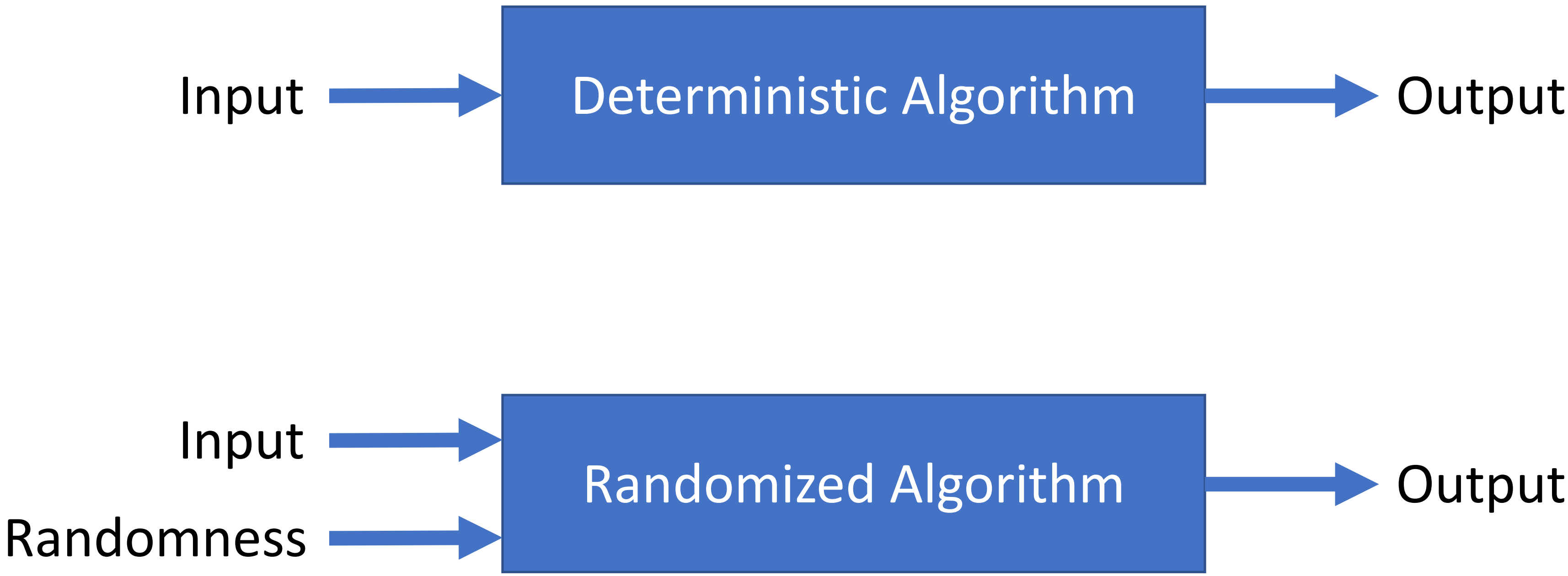
# CSC373

# Week 11: Randomized Algorithms

# Randomized Algorithms



# Randomized Algorithms

- Running time
  - Harder goal: the running time should *always* be small
    - Regardless of both the input and the random coin flips
  - Easier goal: the running time should be small *in expectation*
    - Expectation over random coin flips
    - But it should still be small for every input (i.e. worst-case)
- Approximation Ratio
  - The objective value of the solution returned should, *in expectation*, be close to the optimum objective value
    - Once again, the expectation is over random coin flips
    - The approximation ratio should be small for every input

# Derandomization

- After coming up with a randomized approximation algorithm, one might ask if it can be "derandomized"
  - Informally, the randomized algorithm is making random choices that, in expectation, turn out to be good
  - Can we make these "good" choices deterministically?

- For some problems…
  - It may be easier to first design a simple randomized approximation algorithm and then de-randomize it…
  - Than to try to directly design a deterministic approximation algorithm

# Recap: Probability Theory

- Random variable $X$
  - Discrete
    - Takes value $v_1$ with probability $p_1$, $v_2$ w.p. $p_2$, …
    - Expected value $E[X] = p_1 \cdot v_1 + p_2 \cdot v_2 + \cdots$
    - Examples: coin toss, the roll of a six-sided die, …

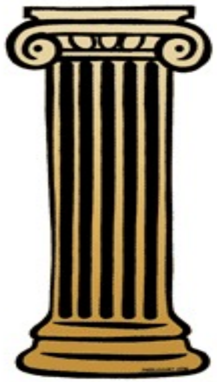
  - Continuous
    - Has a probability density function (pdf) $f$
    - Its integral is the cumulative density function (cdf) $F$
      - $F(x) = \Pr[X \leq x] = \int_{-\infty}^{x} f(t)\, dt$
    - Expected value $E[X] = \int_{-\infty}^{\infty} x\, f(x)\, dx$
    - Examples: normal distribution, exponential distribution, uniform distribution over $[0,1]$, …

# Recap: Probability Theory

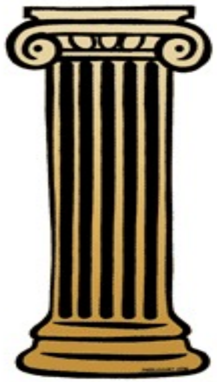
- Things you should be aware of…
  - Conditional probabilities
  - Conditional expectations
  - Independence among random variables
  - Moments of random variables
  - Standard discrete distributions: uniform over a finite set, Bernoulli, binomial, geometric, Poisson, …
  - Standard continuous distributions: uniform over intervals, Gaussian/normal, exponential, …

# Three Pillars

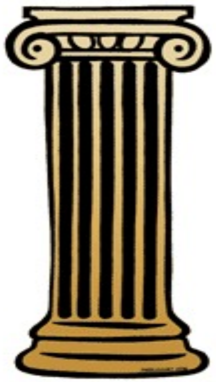
Linearity of Expectation



Union Bound



Chernoff Bound



- Deceptively simple, but incredibly powerful!
- Many many many many probabilistic results are just interesting applications of these three results

# Three Pillars

- Linearity of expectation
  - $E[X+Y] = E[X] + E[Y]$
  - This does *not* require any independence assumptions about $X$ and $Y$
  - E.g. if you want to find out how many people will attend your party on average, just ask each person the probability with which they will attend and sum up the probabilities
    - It does not matter whether some of them are friends and either all will attend together or none will attend

# Three Pillars

- Union bound
  - For any two events $A$ and $B$, $\Pr[A \cup B] \le \Pr[A] + \Pr[B]$
  - "Probability that at least one of the $n$ events $A_1, \dots, A_n$ will occur is at most $\sum_i \Pr[A_i]$"
  - Typically, $A_1, \dots, A_n$ are "bad events"
    - You do not want any of them to occur
    - If you can individually bound $\Pr[A_i] \le {}^{1}/_{2n}$ for each $i$, then probability that at least one them occurs $\le 1/2$
    - Thus, with probability $\ge {}^{1}/_{2}$, *none* of the bad events will occur

- Chernoff bound & Hoeffding's inequality
  - Read up!

# Exact Max-$k$-SAT

# Exact Max-$k$-SAT

- Problem (recall)
  - Input: An exact $k$-SAT formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause $C_i$ has exactly $k$ literals, and a weight $w_i \geq 0$ of each clause $C_i$
  - Output: A truth assignment $\tau$ maximizing the number (or total weight) of clauses satisfied under $\tau$

- Let us denote by $W(\tau)$ the total weight of clauses satisfied under $\tau$

# Exact Max-$k$-SAT

- Recall our local search
  - $N_d(\tau)$ = set of all truth assignments which can be obtained by changing the value of at most $d$ variables in $\tau$

- Result 1: Neighborhood $N_1(\tau) \Rightarrow {}^2/_3$-apx for Exact Max-2-SAT.
- Result 2: Neighborhood $N_1(\tau) \cup \tau^c \Rightarrow {}^3/_4$-apx for Exact Max-2-SAT.
- Result 3: Neighborhood $N_1(\tau)$ + oblivious local search $\Rightarrow {}^3/_4$-apx for Exact Max-2-SAT.

# Exact Max-$k$-SAT

- Recall our local search
  - $N_d(\tau)$ = set of all truth assignments which can be obtained by changing the value of at most $d$ variables in $\tau$

- We claimed that ¾-apx for Exact Max-2-SAT can be generalized to $\frac{2^k-1}{2^k}$-apx for Exact Max-$k$-SAT
  - Algorithm becomes slightly more complicated

- What can we do with randomized algorithms?

# Exact Max-$k$-SAT

- Recall:
  - We have a formula $\varphi = C_1 \land C_2 \land \cdots \land C_m$
  - Variables = $x_1, \ldots, x_n$, literals = variables or their negations
  - Each clause contains exactly $k$ literals

- The most naïve randomized algorithm
  - Set each variable to TRUE with probability ½ and to FALSE with probability ½

- How good is this?

# Exact Max-$k$-SAT

- Recall:
  - We have a formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$
  - Variables = $x_1, \dots, x_n$, literals = variables or their negations
  - Each clause contains exactly $k$ literals

- Let $\tau$ be a random assignment
  - For each clause $C_i$: $\Pr[C_i \text{ is not satisfied}] = {}^{1}/_{2^k}$ (WHY?)
    - Hence, $\Pr[C_i \text{ is satisfied}] = {}^{(2^k-1)}/_{2^k}$
  - $E[W(\tau)] = \sum_{i=1}^{m} w_i \cdot \Pr[C_i \text{ is satisfied}]$ (WHY?)
  - $E[W(\tau)] = \frac{2^k-1}{2^k} \cdot \sum_{i=1}^{m} w_i \geq \frac{2^k-1}{2^k} \cdot OPT$

# Derandomization

- Can we derandomize this algorithm?
  - What are the choices made by the algorithm?
    - Setting the values of $x_1, x_2, \dots, x_n$
  - How do we know which set of choices is good?

- Idea:
  - Do not think about all the choices at once.
  - Think about them one by one.
  - Goal: Gradually convert the random assignment $\tau$ to a deterministic assignment $\hat{\tau}$ such that $W(\hat{\tau}) \geq E[W(\tau)]$
    - Combining with $E[W(\tau)] \geq \frac{2^k-1}{2^k} \cdot OPT$ will give the desired deterministic approximation ratio

# Derandomization

- Start with the random assignment $\tau$ and write…

$$E[W(\tau)] = \Pr[x_1 = T] \cdot E[W(\tau)|x_1 = T] + \Pr[x_1 = F] \cdot E[W(\tau)|x_1 = F]$$
$$= {}^1/_2 \cdot E[W(\tau)|x_1 = T] + {}^1/_2 \cdot E[W(\tau)|x_1 = F]$$

- Hence, $\max(E[W(\tau)|x_1 = T], E[W(\tau)|x_1 = F]) \geq E[W(\tau)]$
  - What is $E[W(\tau)|x_1 = T]$?
    - It is the expected weight when setting $x_1 = T$ deterministically but still keeping $x_2, \ldots, x_n$ random

- If we can compute both $E[W(\tau)|x_1 = T]$ and $E[W(\tau)|x_1 = F]$, and pick the better one…
  - Then we can set $x_1$ deterministically without degrading the expected objective value

# Derandomization

- After deterministically making the right choice for $x_1$ (say T), we can apply the same logic to $x_2$

$$E[W(\tau)|x_1 = T] = {}^1/_2 \cdot E[W(\tau)|x_1 = T, x_2 = T] + {}^1/_2 \cdot E[W(\tau)|x_1 = T, x_2 = F]$$

  - Pick the better of the two conditional expectations

- Derandomized Algorithm:
  - For $i = 1, \dots, n$
    - Let $z_i = T$ if $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T] \geq E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = F]$, and $z_i = F$ otherwise
    - Set $x_i = z_i$

# Derandomization

- This is called *the method of conditional expectations*
  - If we're happy when making a choice at random, we should be at least as happy conditioned on at least one of the possible values of that choice

- Remaining question:
  - How do we compute & compare the two conditional expectations: $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T]$ and $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = F]$?

# Derandomization

- $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T]$
  - $\sum_r w_r \cdot \Pr[C_r$ is satisfied $|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T]$
  - Set the values of $x_1, \dots, x_{i-1}, x_i$
  - If $C_r$ resolves to TRUE already, the corresponding probability is 1
  - Otherwise, if there are $\ell$ literals left in $C_r$ after setting $x_1, \dots, x_{i-1}, x_i$, the corresponding probability is $\frac{2^\ell - 1}{2^\ell}$

- Compute $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = F]$ similarly

# Max-SAT

- Simple randomized algorithm
  - $\frac{2^k-1}{2^k}$ −approximation for Max-$k$-SAT
  - Max-3-SAT ⇒ $^7/_8$
    - [Håstad]: This is the best possible assuming P ≠ NP
  - Max-2-SAT ⇒ $^3/_4 = 0.75$
    - The best known approximation is $0.9401$ using semi-definite programming and randomized rounding
  - Max-SAT ⇒ $^1/_2$
    - Max-SAT = no restriction on the number of literals in each clause
    - The best known approximation is $0.7968$, also using semi-definite programming and randomized rounding

# Max-SAT

- Better approximations for Max-SAT
  - Semi-definite programming is out of the scope
  - But we will see the simpler "LP relaxation + randomized rounding" approach that gives $1 - {}^{1}/_{e} \approx 0.6321$ approximation

- Max-SAT:
  - Input: $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause $C_i$ has weight $w_i \geq 0$ (and can have any number of literals)
  - Output: Truth assignment that approximately maximizes the weight of clauses satisfied

# LP Formulation of Max-SAT

- First, IP formulation:
  - Variables:
    - $y_1, \dots, y_n \in \{0,1\}$
      - $y_i = 1$ iff variable $x_i =$ TRUE in Max-SAT
    - $z_1, \dots, z_m \in \{0,1\}$
      - $z_j = 1$ iff clause $C_j$ is satisfied in Max-SAT
    - Program:

      Maximize $\Sigma_j \, w_j \cdot z_j$
      s.t.
      $\Sigma_{x_i \in C_j} \, y_i + \Sigma_{\bar{x}_i \in C_j} \, (1 - y_i) \geq z_j \quad \forall j \in \{1, \dots, m\}$
      $y_i, z_j \in \{0,1\} \quad \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$

# LP Formulation of Max-SAT

- LP relaxation:
  - Variables:
    - $y_1, \dots, y_n \in [0,1]$
      - $y_i = 1$ iff variable $x_i =$ TRUE in Max-SAT
    - $z_1, \dots, z_m \in [0,1]$
      - $z_j = 1$ iff clause $C_j$ is satisfied in Max-SAT
    - Program:

Maximize $\Sigma_j \, w_j \cdot z_j$
s.t.
$\Sigma_{x_i \in C_j} \, y_i + \Sigma_{\bar{x}_i \in C_j} \, (1 - y_i) \geq z_j \quad \forall j \in \{1, \dots, m\}$
$y_i, z_j \in [0,1] \quad \forall i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$

# Randomized Rounding

- Randomized rounding
  - Find the optimal solution $(y^*, z^*)$ of the LP
  - Compute a random IP solution $\hat{y}$ such that
    - Each $\hat{y}_i = 1$ with probability $y_i^*$ and $0$ with probability $1 - y_i^*$
    - Independently of other $\hat{y}_i$'s
    - The output of the algorithm is the corresponding truth assignment
  - What is $\Pr[C_j \text{ is satisfied}]$ if $C_j$ has $k$ literals?

$$1 - \Pi_{x_i \in C_j} (1 - y_i^*) \cdot \Pi_{\bar{x}_i \in C_j} (y_i^*)$$

$$\underbrace{\geq 1 - \left( \frac{\Sigma_{x_i \in C_j} (1 - y_i^*) + \Sigma_{\bar{x}_i \in C_j} (y_i^*)}{k} \right)^k}_{\text{AM-GM inequality}} \underbrace{\geq 1 - \left( \frac{k - z_j^*}{k} \right)^k}_{\text{LP constraint}}$$

# Randomized Rounding

- Claim
  - $1-\left(1-\frac{z}{k}\right)^k \geq \left(1-\left(1-\frac{1}{k}\right)^k\right) \cdot z$ for all $z \in [0,1]$ and $k \in \mathbb{N}$

- Assuming the claim:

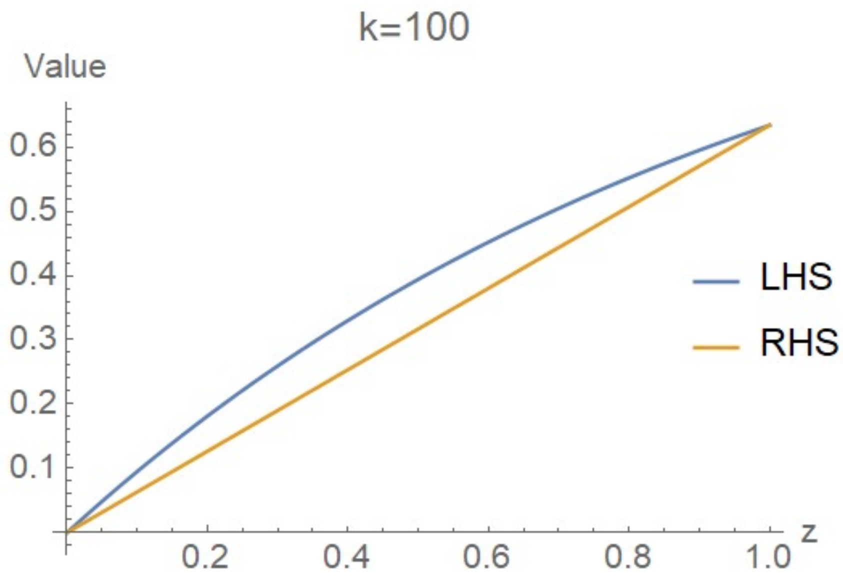
$$\Pr[C_j \text{ is satisfied}] \geq 1-\left(\frac{k-z_j^*}{k}\right)^k \geq \underbrace{\left(1-\left(1-\frac{1}{k}\right)^k\right) \cdot z_j^* \geq \left(1-\frac{1}{e}\right) \cdot z_j^*}_{\text{Standard inequality}}$$

- Hence,

$$\mathbb{E}[\#\text{weight of clauses satisfied}] \geq \underbrace{\left(1-\frac{1}{\mathrm{e}}\right) \sum_j w_j \cdot z_j^* \geq \left(1-\frac{1}{e}\right) \cdot OPT}_{\text{Optimal LP objective} \geq \text{optimal ILP objective}}$$

# Randomized Rounding

- Claim
  - $1 - \left(1 - \frac{z}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot z$ for all $z \in [0,1]$ and $k \in \mathbb{N}$
- Proof of claim:
  - True at $z = 0$ and $z = 1$ (same quantity on both sides)
  - For $0 \leq z \leq 1$:
    - LHS is a convex function
    - RHS is a linear function
    - Hence, LHS $\geq$ RHS ■



# Improving Max-SAT Apx

- Best of both worlds:
  - Run both "LP relaxation + randomized rounding" and "naïve randomized algorithm"
  - Return the best of the two solutions

  - Claim without proof: This achieves a ${}^{3}/_{4} = 0.75$ approximation!
    - This algorithm can be derandomized.

  - Recall:
    - "naïve randomized" = independently set each variable to TRUE/FALSE with probability $0.5$ each, which only gives ${}^{1}/_{2} = 0.5$ approximation by itself

NOT IN SYLLABUS

# Randomization for Sublinear Running Time

# Sublinear Running Time

NOT IN SYLLABUS

- Given an input of length $n$, we want an algorithm that runs in time $o(n)$
  - $o(n)$ examples: $\log n$, $\sqrt{n}$, $n^{0.999}$, $\frac{n}{\log n}$, …
  - The algorithm doesn't even get to read the full input!

- There are four possibilities:
  - Exact vs inexact: whether the algorithm always returns the correct/optimal solution or only does so with high probability (or gives some approximation)
  - Worst-case versus expected running time: whether the algorithm always takes $o(n)$ time or only does so in expectation (but still on every instance)

NOT IN SYLLABUS

# Exact algorithms, expected sublinear time

# Searching in Sorted List

NOT IN SYLLABUS

- Input: A sorted doubly linked list with $n$ elements.
  - Imagine you have an array $A$ with $O(1)$ access to $A[i]$
  - $A[i]$ is a tuple $(x_i, p_i, n_i)$
    - Value, index of previous element, index of next element.
  - Sorted: $x_{p_i} \le x_i \le x_{n_i}$

- Task: Given $x$, check if there exists $i$ s.t. $x = x_i$

- Goal: We will give a randomized + exact algorithm with expected running time $O(\sqrt{n})$!

# Searching in Sorted List

NOT IN SYLLABUS

- Motivation:
  - Often we deal with large datasets that are stored in a large file on disk, or possibly broken into multiple files
  - Creating a new, sorted version of the dataset is expensive
  - It is often preferred to "implicitly sort" the data by simply adding previous-next pointers along with each element
  - Would like algorithms that can operate on such implicitly sorted versions and yet achieve sublinear running time
    - Just like binary search achieves for an explicitly sorted array

# Searching in Sorted List

NOT IN SYLLABUS

**Algorithm:**

- Select $\sqrt{n}$ random indices $R$
- Access $x_j$ for each $j \in R$
- Find "accessed $x_j$ nearest to $x$ in either direction"
  - either the largest among all $x_j \leq x$…
  - or the smallest among all $x_j \geq x$
- If you take the largest $x_j \leq x$, start from there and keep going "next" until you find $x$ or go past its value
- If you take the smallest $x_j \geq x$, start from there and keep going "previous" until you find $x$ or go past its value

# Searching in Sorted List

NOT IN SYLLABUS

- Analysis sketch:
  - Suppose you find the largest $x_j \leq x$ and keep going "next"
  - Let $x_i$ be smallest value $\geq x$
  - Algorithm stops when it hits $x_i$
  - Algorithm throws $\sqrt{n}$ random "darts" on the sorted list
  - Chernoff bound:
    - Expected distance of $x_i$ to the closest dart to its left is $O(\sqrt{n})$
    - We'll assume this without proof!
  - Hence, the algorithm only does "next" $O(\sqrt{n})$ times in expectation

# Searching in Sorted List

NOT IN SYLLABUS

- Note:
  - We don't *really* require the list to be doubly linked. Just "next" pointer suffices if we have a pointer to the first element of the list (a.k.a. "anchored list").

- This algorithm is optimal!
- Theorem: No algorithm that always returns the correct answer can run in $o(\sqrt{n})$ expected time.
  - Can be proved using "Yao's minimax principle"
  - Beyond the scope of the course, but this is a fundamental result with wide-ranging applications

# Sublinear Geometric Algo

NOT IN SYLLABUS

- Chazelle, Liu, and Magen [2003] proved the $\Theta(\sqrt{n})$ bound for searching in a sorted linked list
  - Their main focus was to generalize these ideas to come up with sublinear algorithms for geometric problems
  - Polygon intersection: Given two convex polyhedra, check if they intersect.
  - Point location: Given a Delaunay triangulation (or Voronoi diagram) and a point, find the cell in which the point lies.
  - They provided optimal $O(\sqrt{n})$ algorithms for both these problems.

NOT IN SYLLABUS

# Inexact algorithms, expected sublinear time

# Estimating Avg Degree in

NOT IN SYLLABUS

- Input:
  - Undirected graph $G$ with $n$ vertices
  - $O(1)$ access to the degree of any queried vertex
- Output:
  - Estimate the average degree of all vertices
  - More precisely, we want to find a $(2 + \epsilon)$-approximation in expected time $O\left(\epsilon^{-O(1)}\sqrt{n}\right)$
- Wait!
  - Isn't this equivalent to "given an array of $n$ numbers between 1 and $n - 1$, estimate their average"?
  - No! That requires $\Omega(n)$ time for any constant approximation!
    - Consider an instance with constantly many $n - 1$'s, and all other 1's: you may not discover any $n - 1$ until you query $\Omega(n)$ numbers

# Estimating Avg Degree in

NOT IN SYLLABUS

- Why are degree sequences more special?

- Erdős–Gallai theorem:
  - $d_1 \geq \cdots \geq d_n$ is a degree sequence iff their sum is even and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} d_i$

- Intuitively, we will sample $O(\sqrt{n})$ vertices
  - We may not discover the few high degree vertices but we'll find their neighbors and thus account for their edges anyway!

# Estimating Avg Degree in

NOT IN SYLLABUS

- Algorithm:
  - Take $^8/_\epsilon$ random subsets $S_i \subseteq V$ with $|S_i| = O\left(\frac{\sqrt{n}}{\epsilon}\right)$
  - Compute the average degree $d_{S_i}$ in each $S_i$.
  - Return $\widehat{d} = \min_i d_{S_i}$

- Analysis beyond the scope of this course
  - This gets the approximation right with probability at least $\frac{5}{6}$
  - By repeating the experiment $\Omega(\log n)$ times and reporting the median answer, we can get the approximation right with probability at least $1 - 1/O(n)$ and a bad approximation with the other $1/O(n)$ probability cannot hurt much