CSC373

Week 11:
Randomized Algorithms
Randomized Algorithms

Deterministic Algorithm

Input → Deterministic Algorithm → Output

Randomized Algorithm

Input → Randomized Algorithm → Output

Randomness → Randomized Algorithm → Output
Randomized Algorithms

• Running time
  ➢ Harder goal: the running time should *always* be small
    o Regardless of both the input and the random coin flips
  ➢ Easier goal: the running time should be small *in expectation*
    o Expectation over random coin flips
    o 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
    o Once again, the expectation is over random coin flips
    o 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

- Deceptively simple, but incredibly powerful!
- Many 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

    o 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 $B$, $Pr[A \cup B] \leq Pr[A] + Pr[B]$
  - “Probability that at least one of the $n$ events $A_1, \ldots, A_n$ will occur is at most $\sum_i Pr[A_i]$”
  - Typically, $A_1, \ldots, A_n$ are “bad events”
    - You do not want any of them to occur
    - If you can individually bound $Pr[A_i] \leq \frac{1}{2n}$ for each $i$, then probability that at least one them occurs $\leq \frac{1}{2}$
    - Thus, with probability $\geq \frac{1}{2}$, *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 $\varphi = C_1 \land C_2 \land \cdots \land C_m$, 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
  \[ N_d(\tau) = \text{set of all truth assignments which can be obtained by}
  \text{changing the value of at most } d \text{ variables in } \tau \]

• 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 \( N_1(\tau) + \text{oblivious local search} \Rightarrow 3/4\)-apx for Exact Max-2-SAT.
Exact Max-\(k\)-SAT

• Recall our local search
  
  \[ N_d(\tau) = \text{set of all truth assignments which can be obtained by changing the value of at most } d \text{ variables in } \tau \]

• We claimed that \(\frac{3}{4}\)-apx for Exact Max-2-SAT can be generalized to \(\frac{2^k - 1}{2^k}\)-apx for Exact Max-\(k\)-SAT
  
  \[ \text{Algorithm becomes slightly more complicated} \]

• 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 $\frac{1}{2}$ and to FALSE with probability $\frac{1}{2}$

• How good is this?
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

• Let $\tau$ be a random assignment
  ➢ For each clause $C_i$: $\Pr[C_i \text{ is not satisfied}] = \frac{1}{2^k}$ (WHY?)
    o Hence, $\Pr[C_i \text{ is satisfied}] = \frac{(2^k-1)}{2^k}$
  ➢ $E[W(\tau)] = \sum_{i=1}^{m} w_i \cdot \Pr[C_i \text{ is satisfied}]$ (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?
    o Setting the values of $x_1, x_2, \ldots, x_n$
  ➢ 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 $W(\hat{\tau}) \geq E[W(\tau)]$
    o 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...

\[ E[W(\tau)] = \Pr[x_1 = T] \cdot E[W(\tau)|x_1 = T] + \Pr[x_1 = F] \cdot E[W(\tau)|x_1 = F] \]
\[ = \frac{1}{2} \cdot E[W(\tau)|x_1 = T] + \frac{1}{2} \cdot E[W(\tau)|x_1 = F] \]

➢ Hence, \( \max(E[W(\tau)|x_1 = T], E[W(\tau)|x_1 = F]) \geq E[W(\tau)] \)
  o 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...
  o Then we can set \( x_1 \) deterministically without degrading the expected objective value
• After deterministically making the right choice for $x_1$ (say T), we can apply the same logic to $x_2$

$$E[W(\tau)|x_1 = T] = \frac{1}{2} \cdot E[W(\tau)|x_1 = T, x_2 = T]$$
$$+ \frac{1}{2} \cdot E[W(\tau)|x_1 = T, x_2 = F]$$

➢ Pick the better of the two conditional expectations

• Derandomized Algorithm:

➢ For $i = 1, \ldots, n$
  o Let $z_i = T$ if $E[W(\tau)|x_1 = z_1, \ldots, x_{i-1} = z_{i-1}, x_i = T] \geq E[W(\tau)|x_1 = z_1, \ldots, x_{i-1} = z_{i-1}, x_i = F]$, and $z_i = F$ otherwise
  o 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:
    \[ E[W(\tau)|x_1 = z_1, \ldots, x_{i-1} = z_{i-1}, x_i = T] \] and
    \[ E[W(\tau)|x_1 = z_1, \ldots, x_{i-1} = z_{i-1}, x_i = F]? \]
Derandomization

- $E[W(\tau)|x_1 = z_1, ..., x_{i-1} = z_{i-1}, x_i = T]$
  - $\sum_r w_r \cdot \Pr[C_r \text{ is satisfied }|x_1 = z_1, ..., x_{i-1} = z_{i-1}, x_i = T]$
  - Set the values of $x_1, ..., x_{i-1}, x_i$
  - If $C_r$ resolves to TRUE already, the corresponding probability is 1
  - If $C_r$ resolves to FALSE already, the corresponding probability is 0
  - Otherwise, if there are $\ell$ literals left in $C_r$ after setting $x_1, ..., x_{i-1}, x_i$, the corresponding probability is $\frac{2^{\ell-1}}{2^\ell}$

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

• Simple randomized algorithm
  
  \[
  \frac{2^k-1}{2^k} \text{—approximation for Max-}k\text{-SAT}
  \]
  
  \[
  \text{Max-3-SAT } \Rightarrow \frac{7}{8}
  \]
  
  o [Håstad]: This is the best possible assuming \( P \neq NP \)
  
  \[
  \text{Max-2-SAT } \Rightarrow \frac{3}{4} = 0.75
  \]
  
  o The best known approximation is 0.9401 using semi-definite programming and randomized rounding

  \[
  \text{Max-SAT } \Rightarrow \frac{1}{2}
  \]
  
  o Max-SAT = no restriction on the number of literals in each clause
  
  o 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 $1 - \frac{1}{e} \approx 0.6321$ approximation

• Max-SAT:
  ➢ Input: $\varphi = C_1 \land C_2 \land \cdots \land C_m$, 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:
  o \( y_1, ..., y_n \in \{0,1\} \)
    • \( y_i = 1 \) iff variable \( x_i = \text{TRUE} \) in Max-SAT
  o \( z_1, ..., z_m \in \{0,1\} \)
    • \( z_j = 1 \) iff clause \( C_j \) is satisfied in Max-SAT

o Program:

Maximize \( \sum_j w_j \cdot z_j \)

s.t.
\[
\sum_{x_i \in C_j} y_i + \sum_{\bar{x}_i \in C_j} (1 - y_i) \geq z_j \quad \forall j \in \{1, ..., m\}
\]
\[
y_i, z_j \in \{0,1\} \quad \forall i \in \{1, ..., n\}, j \in \{1, ..., m\}
\]
LP Formulation of Max-SAT

• LP relaxation:

  ➢ Variables:

    o \( y_1, \ldots, y_n \in [0,1] \)
      • \( y_i = 1 \) iff variable \( x_i = \text{TRUE} \) in Max-SAT
    o \( z_1, \ldots, z_m \in [0,1] \)
      • \( z_j = 1 \) iff clause \( C_j \) is satisfied in Max-SAT

  o Program:

    Maximize \( \sum_j w_j \cdot z_j \)
    s.t.
    \( \sum_{i \in C_j} y_i + \sum_{\overline{x}_i \in C_j} (1 - y_i) \geq z_j \quad \forall j \in \{1, \ldots, m\} \)
    \( y_i, z_j \in [0,1] \quad \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \)
Randomized Rounding

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

\[
1 - \Pi_{x_i \in C_j} (1 - y_i^*) \cdot \Pi_{\bar{x}_i \in C_j} (y_i^*) \\
\geq 1 - \left( \frac{\Sigma_{x_i \in C_j} (1 - y_i^*) + \Sigma_{\bar{x}_i \in C_j} (y_i^*)}{k} \right)^k \\
\geq 1 - \left( \frac{k - z_j^*}{k} \right)^k
\]

AM-GM inequality 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:

\[ \Pr[C_j \text{ is satisfied}] \geq 1 - \left( \frac{k - z^*_j}{k} \right)^k \geq \left( 1 - \left( 1 - \frac{1}{k} \right)^k \right) \cdot z^*_j \geq \left( 1 - \frac{1}{e} \right) \cdot z^*_j \]

• Hence,

\[ \mathbb{E}[\text{#weight of clauses satisfied}] \geq \left( 1 - \frac{1}{e} \right) \sum_j w_j \cdot z^*_j \geq \left( 1 - \frac{1}{e} \right) \cdot OPT \]
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 \( \blacksquare \)
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 $\frac{3}{4} = 0.75$ approximation!
  o This algorithm can be derandomized.

➢ Recall:
  o “naïve randomized” = independently set each variable to TRUE/FALSE with probability 0.5 each, which only gives $\frac{1}{2} = 0.5$ approximation by itself
Back to 2-SAT

• Max-2-SAT is NP-hard (we didn’t prove this!)
• But 2-SAT can be efficiently solved
  ➢ “Given a 2-CNF formula, check whether all clauses can be satisfied simultaneously.”

• Algorithm:
  ➢ Repeatedly eliminate a clause with one literal & set the literal to true
  ➢ Create a graph with each remaining literal as a vertex
  ➢ For every clause \((x \lor y)\), add two edges: \(\overline{x} \rightarrow y\) and \(\overline{y} \rightarrow x\)
    o \(u \rightarrow v\) means if \(u\) is true, \(v\) must be true
  ➢ Formula is satisfiable iff no path from \(x\) to \(\overline{x}\) or \(\overline{x}\) to \(x\) for any \(x\)
    o Can be checked in polynomial time
Random Walk + 2-SAT

• Here’s a cute randomized algorithm by Papadimitriou [1991]

• Algorithm:
  ➢ Start with an arbitrary assignment.
  ➢ While there is an unsatisfied clause $C = (x \lor y)$
    o Pick one of the two literals with equal probability.
    o Flip the variable value so that $C$ is satisfied.

• But can’t this hurt the other clauses?
  ➢ In a given step, yes.
  ➢ But in expectation, we will still make progress.
Random Walk + 2-SAT

• Theorem:
  ➢ If there is a satisfying assignment $\tau^*$, then this algorithm reaches a satisfying assignment in $O(n^2)$ expected time.

• Proof:
  ➢ Fix a satisfying assignment $\tau^*$
  ➢ Let $\tau_0$ be the starting assignment
  ➢ Let $\tau_i$ be the assignment after $i$ iterations
  ➢ Consider the “hamming distance” $d_i$ between $\tau_i$ and $\tau^*$
    o Number of coordinates in which the two differ
    o $d_i \in \{0,1,\ldots,n\}$
  ➢ Claim: the algorithm hits $d_i = 0$ in $O(n^2)$ iterations in expectation, unless it stops before that
Random Walk + 2-SAT

• **Observation:** \( d_{i+1} = d_i - 1 \) or \( d_{i+1} = d_i + 1 \)
  - Because we change one variable in each iteration.

• **Claim:** \( \Pr[d_{i+1} = d_i - 1] \geq 1/2 \)

• **Proof:**
  - Iteration \( i \) considers an unsatisfied clause \( C = (x \lor y) \)
  - \( \tau^* \) satisfies at least one of \( x \) or \( y \), while \( \tau_i \) satisfies neither
  - Because we pick a literal randomly, w.p. at least \( 1/2 \) we pick one where \( \tau_i \) and \( \tau^* \) differ and decrease the distance
  - **Q:** Why did we need an unsatisfied clause? What if we pick one of \( n \) variables randomly and flip it?
Random Walk 2-SAT

• Answer:
  ➢ We want the distance to decrease with probability at least \( \frac{1}{2} \) no matter how close or far we are from \( \tau^* \)
  ➢ If we are already close, choosing a variable at random will likely choose one where \( \tau \) and \( \tau^* \) already match
  ➢ Flipping this variable will increase the distance with high probability
  ➢ An unsatisfied clause narrows it down to two variables s.t. \( \tau \) and \( \tau^* \) differ on at least one of them
Random Walk + 2-SAT

- Observation: \( d_{i+1} = d_i - 1 \) or \( d_{i+1} = d_i + 1 \)
- Claim: \( \Pr[d_{i+1} = d_i - 1] \geq 1/2 \)

• How does this help?
• How does this help?
  - Can view this as a “Markov chain” and use known results on “hitting time”
  - But let’s prove it using elementary methods
Random Walk + 2-SAT

• For $k > \ell$, define:
  ➢ $T_{k,\ell} =$ expected number of iterations it takes to hit distance $\ell$
    for the first time when you start at distance $k$

• $T_{i+1,i} \leq \frac{1}{2} * 1 + \frac{1}{2} * (1 + T_{i+2,i})$
  $= \frac{1}{2} * (1) + \frac{1}{2} * (1 + T_{i+2,i+1} + T_{i+1,i})$

• Simplifying:
  ➢ $T_{i+1,i} \leq 2 + T_{i+2,i+1} \leq 4 + T_{i+3,i+2} \leq \cdots \leq O(n) + T_{n,n-1} \leq O(n)$
    ○ Uses $T_{n,n-1} = 1$ (Why?)

• $T_{n,0} \leq T_{n,n-1} + \cdots + T_{1,0} = O(n^2)$

Which pillar did we use?
Random Walk + 2-SAT

• Can view this algorithm as a “drunken local search”
  ➢ We are searching the local neighborhood
  ➢ But we don’t ensure that we necessarily improve
  ➢ We just ensure that in expectation, we aren’t hurt
  ➢ Hope to reach a feasible solution in polynomial time

• Schöning extended this technique to $k$-SAT
  ➢ Schöning’s algorithm no longer runs in polynomial time, but this is okay because $k$-SAT is NP-hard
  ➢ It still improves upon the naïve $2^n$
  ➢ Later derandomized by Moser and Scheder [2011]
Schöning’s Algorithm for \( k \)-SAT

**Algorithm:**
- Choose a random assignment \( \tau \)
- Repeat \( 3n \) times (\( n = \# \text{variables} \))
  - If \( \tau \) satisfies the CNF, stop
  - Else, pick an arbitrary unsatisfied clause and flip a random literal in the clause
Schöning’s Algorithm

• Randomized algorithm with one-sided error
  ➢ If the CNF is satisfiable, it finds an assignment with probability at least \((\frac{1}{2} \cdot \frac{k}{k-1})^n\)
  ➢ If the CNF is unsatisfiable, it never finds an assignment

• Expected #times we need to repeat in order to find a satisfying assignment when one exists: \(O\left(2 \left(1 - \frac{1}{k}\right)^n\right)\)
  ➢ For \(k = 3\), this gives \(O(1.3333^n)\)
  ➢ For \(k = 4\), this gives \(O(1.5^n)\)
Best Known Results

• 3-SAT

• Deterministic
  ➢ Derandomized Schöning’s algorithm: $O(1.3333^n)$
  ➢ Best known: $O(1.3303^n)$ [HSSW]
    o If we are assured that there is a unique satisfying assignment: $O(1.3071^n)$ [PPSZ]

• Randomized
  ➢ Nothing better known without one-sided error
  ➢ With one-sided error, best known is $O(1.30704^n)$ [Modified PPSZ]
Random Walk + 2-SAT

• Random walks are not only of theoretical interest
  ➢ WalkSAT is a practical SAT algorithm
  ➢ At each iteration, pick an unsatisfied clause \textit{at random}
  ➢ Pick a variable in the unsatisfied clause to flip:
    o With some probability, pick at random.
    o With the remaining probability, pick one that will make the fewest previously satisfied clauses unsatisfied
  ➢ Restart a few times (avoids being stuck in local minima)

• Faster than “intelligent local search” (GSAT)
  ➢ Flip the variable that satisfies most clauses
Random Walks on Graphs

• Aleliunas et al. [1979]
  ➢ Let $G$ be a connected undirected graph. Then a random walk starting from any vertex will cover the entire graph (visit each vertex at least once) in $O(mn)$ steps.

• Limiting probability distribution
  ➢ In the limit, the random walk will visit a vertex with degree $d_i$ in $\frac{d_i}{2m}$ fraction of the steps

• Markov chains
  ➢ Generalize to directed (possibly infinite) graphs with unequal edge traversal probabilities
Randomization for Sublinear Running Time
Sublinear Running Time

• 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)
Exact algorithms, expected sublinear time
Searching in Sorted List

• **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: $x_{p_i} \leq x_i \leq x_{n_i}$

• **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

• 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
    o Just like binary search achieves for an explicitly sorted array
Searching in Sorted List

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

- 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

• **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 Algorithms

• 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.
Inexact algorithms, expected sublinear time
Estimating Avg Degree in Graph

- **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 $O(\epsilon^{-O(1)}\sqrt{n})$

- **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 Graph

• 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 Graph

• 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 $\overline{d} = \min_i d_{S_i}$

• 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