0. Announcements (points)

- NO QUIZZES for the rest of the semester.

1. fastPow Redux! (points)

Design an efficient algorithm to compute $A^E \mod N$ (modular exponentiation) where $A$, $E$, and $N$ each have at most $n$ bits, and analyze its time complexity.

```plaintext
1: function fastPow(A, E, N)
2:   if $E = 0$ then
3:       return 1
4:   end if
5:   if $E \equiv 0 \mod 2$ then
6:       return fastPow($A^2 \mod N$, $E/2$, N)
7:   end if
8:   return $A * fastPow(A^2 \mod N, (E-1)/2, N)$
9: end function
```

Checking the conditions on lines 2 and 5 take constant time, as does calculating $E/2$ or $(E-1)/2$ ($E$ is written in binary...). So the work of the algorithm is in calculating $A^2 \mod N$ and multiplying by $A$. From lecture we know multiplication is $O(n^{log_2 3})$, and division is $n^2$, and hence mod is also $n^2$. Therefore our recurrence is $T(E) = T(E/2) + O(n^2)$. Noting that it is constant work before each recursive call and we make $log_2 E = n$ recursive calls overall, this algorithm runs in time $O(n^{1+2}) = O(n^3)$.

2. Atlantic City to Monte Carlo. (points)

Suppose you are given a randomized algorithm that solves $f : \Sigma^* \to \Sigma^*$ in expected time $T(n)$ and with $\varepsilon$ probability of error (i.e., the algorithm gambles both with correctness and running time). Show that for any constant $\varepsilon' > 0$, there is a Monte Carlo algorithm computing $f$ with running time $O(T(n))$ and error probability $\varepsilon + \varepsilon'$.

Let $A$ be the algorithm given to us. We want to construct a Monte Carlo algorithm $A'$, which works as follows: run $A$ for $T(n)/\varepsilon'$ steps. If it halts, give its answer. If not, declare “failure”. We can fail in two ways: either the simulation of $A$ halts and gives the wrong answer, or the simulation of $A$ does not halt in $T(n)/\varepsilon'$ steps. The first happens with probability at most $\varepsilon$, and the second happens with probability at most $\varepsilon'$ (by Markov). Using the union bound, the error prob is $\varepsilon + \varepsilon'$.

Here is another way of thinking about it. For any fixed input, the original algorithm induces a probability tree representing the computation. We know that at most $\varepsilon'$ fraction of the leaves are too deep (i.e., running time exceeds $T(n)/\varepsilon'$). And we know at most $\varepsilon$ fraction of the leaves give the wrong answer. By union bound, at most $\varepsilon + \varepsilon'$ fraction of the leaves either give the wrong answer or are too deep. If the wrong leaves intersect with deep ones, that would better, but in the worst case, wrong leaves and deep ones would be disjoint.

**Technicality Alert:** There is a small technical issue here. Algorithm $A'$ needs to be able to compute $T(|x|)$ from $x$ in $O(T(|x|))$ time. This is indeed the case for most $T(\cdot)$ that we care about.
3. Max number of Min cuts (points)
Show that a graph can have at most \( n(n-1)/2 \) distinct minimum cuts.
In the notes.

4. (EXTRA) PITiful polynomials (points)
Consider the PIT problem: given as input a polynomial, written using any of
\[ \Sigma = \{ (, +, - \} \cup \{ x_i : i \in \mathbb{N} \} \cup \mathbb{Q}, \]
calculate whether the polynomial is equal to 0.

Example input:
- \((x_1 x_1 x_1 + x_3)(x_5 + x_1)\) (which is not 0).
- \((x_1 + x_2)(x_1 - x_2) - x_1 x_1 - x_2 x_2\) (which is 0).

(a) Before we solve this problem, we need a lemma which you might find helpful.

**Lemma 1 (Schwartz-Zippel)** If \( P \) is a non-zero polynomial on variables \( x_1, \ldots, x_n \), and is of degree at most \( d \), then if we draw each \( x_i \) uniformly from any set \( S \subseteq \mathbb{R} \),
\[ \Pr[P(x_1, x_2, \ldots, x_n) = 0] \leq \frac{d}{|S|}. \]

Remark: You can think of this lemma as a kind of multivariable fundamental theorem of algebra.

Hint: You can prove this by induction on \( n \). You probably know the base case, and don’t forget FToA in the inductive step.

We prove this by induction on \( n \).

**Base Case** \( n = 1 \). This follows from the fundamental theorem of algebra, which says the polynomial can have at most \( d \) roots overall.

**Inductive Step.** Assume that every polynomial \( P' \) on \( n - 1 \) variables with degree \( d' \) has
\[ \Pr[P'(x_1, x_2, \ldots, x_{n-1}) = 0] \leq d'/|S|. \]
Rewrite our polynomial \( P \) grouping the \( x_n \) terms, so in the end we get an expression of the form
\[ P(x_1, \ldots, x_n) = \sum_{i=1}^{d} x_n^i Q_i(x_1, \ldots, x_{n-1}). \]
Since \( P \) is non-zero, some \( Q_i \) is non-zero. Fix \( i \) to be the largest with this property. Now fix values \( x_1^*, x_2^*, \ldots, x_{n-1}^* \), all except \( x_n \), with \( Q_i(x_1^*, \ldots, x_{n-1}^*) \neq 0 \). By union bound we have
\[ \Pr[P(x_1^*, \ldots, x_{n-1}^*, x_n) = 0] \leq \Pr[Q_i(x_1, \ldots, x_{n-1}) = 0] \]
\[ + \Pr[P(x_1^*, \ldots, x_{n-1}^*, x_n) = 0], \] (1)
if we take the randomness in the second probability over \( x_n \) only, i.e., treating the \( x_1^*, \ldots, x_{n-1}^* \) as fixed. We can also think of this step as conditioning over all possible values of \( x_1, \ldots, x_{n-1} \).

By the induction hypothesis, the first quantity is upper-bounded by \((d - i)/|S|\), since \( Q_i \) can have degree at most \( d - i \). The second probability is upper-bounded by the probability if we take \( x_1^*, \ldots, x_{n-1}^* \) as fixed, and take the randomness just over \( x_n \). In this way, we just need to calculate \( \Pr[P(x_1^*, \ldots, x_{n-1}^*, x_n) = 0] \) for fixed \( x_1^*, \ldots, x_{n-1}^* \). By the fundamental theorem of algebra, this polynomial in \( x_n \) has at most \( i \) roots, since it is of degree at most \( i \). Therefore
\[ \Pr[P(x_1^*, \ldots, x_{n-1}^*, x_n) = 0] \leq i/|S|, \]
Combining these two bounds into (1) yields an upper bound of \( d/|S| \), completing the proof.
(b) Come up with an efficient randomized algorithm to solve this problem with error probability $\varepsilon$.
(Hint: Schwartz-Zippel Lemma)

```
function PIT(P)
    Calculate the degree $d$ of the polynomial.
    Pick $S = \{1, \ldots, 2d\}$.
    Choose $N$ such that $1/2^N < \varepsilon$ ($N > \log_2(1/\varepsilon)$).
    for $i \in [1..N]$ do
        Pick $x_1, \ldots, x_n$ each i.i.d. uniformly from $S$.
        if $P(x_1, \ldots, x_n) \neq 0$ then
            return False
        end if
    end for
    return True
end function
```

By Schwartz-Zippel, the probability the algorithm makes a mistake on any iteration is at most $1/2$, so the probability we make a mistake on any iteration is at most $1/2^N < \varepsilon$. The algorithm is poly-time because $N$ is polynomial in the input, and evaluating $P$ is polynomial time given $x_1, \ldots, x_n$.

(c) Something to ponder: some people believe that randomization gives you no more power than determinism. If we believe them, then try to come up with a deterministic algorithm to solve this problem.