CPS 430/590.06 **Prof. John Reif**
Design and Analysis of Algorithms **Fall 2013**
Homework 6 **Matt Dickenson**

Question 1

Given a vector of probabilities p of length n, we wish to compute the probability of k successes in n trials, with probability of success i in the i^{th} trial. Obviously if $k > n$, the probability is zero.

One (inefficient) way to go about this is to iterate over all of the possible subsets S of n that could offer k successes and $n-k$ failures. For each of those subsets $s \in S$, we can multiply the probabilities of the successes (s_a), $\prod_{i \in s_a} p_i$, and the complements of the probabilities for the failures (s_b), $\prod_{j \in s_b}(1-p_j)$. This gives us the formula

$$Pr(k\text{successes}) = \sum_{s \in S} \prod_{i \in s_a} p_i \prod_{j \in s_b}(1-p_j)$$

However, there are $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ elements in S, which is quite large and makes this algorithm inefficient.

A more efficient approach uses recursion, alternating between addition and subtraction to avoid double-counting. Again, if $k > n$, we return zero. The base case of the recursion is $k = 0$, in which case we return the product of all failure probabilities, $\prod_j = 1^n(1-p_j$. For $0 < k <= n$,

$$Pr(k\text{successes}) = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{n} (\frac{p_j}{1-p_j})^i Pr(k-i \text{ successes})(-1)^{i-1}$$

Notice that the inner term $\frac{\sum_{j=1}^{n}(p_j}{1-p_j)=C}$ is constant for a given problem, so we can compute it once at the beginning:

$$Pr(k\text{successes}) = \frac{1}{k} \sum_{i=1}^{k} C^i Pr(k-i \text{ successes})(-1)^{i-1}$$

We are then left with a sum of size k and $k+1$ recursive calls. Using n as an upper bound on k, this is $O(n \times n + n) = O(n^2)$, as desired.

Question 2

1. Assume $P = NP$.
2. The factorization of large numbers, $F(n)$, can be verified in polynomial time, that is $F(n) \in NP$. $P = NP$ implies $F(n) \in P$.
3. Suppose an adversary initially has the public key function $P(M) = M^e mod n$ for the message M, but not the secret key $S(C) = C^d mod n$ or the value of n. $S(P(M)) = M = (P(S(M))$.

4. If $F(n) \in P$ then n can be factorized in polynomial time. Because e and d are small primes, the adversary can find $S(C)$ by trial and error (iteration) in polynomial time.

5. Because $S(C)$ can be computed in polynomial time, the adversary can use the encrypted transmission $P(M)$ to compute $M = S(P(M))$.

6. Thus, if $P = NP$, an adversary can break RSA encryption.