Probabilistic (or Randomized) Algorithms

Factoring and Primality

Is there an efficient algorithm to factor large (say, 1000 digit) integers into their prime factors? This is a wide open question, one of the most important open questions in computer science. In 1977, Rivest, Shamir, and Adleman developed a public-key encryption system now known as *RSA*, whose security depends on the hypothesis that there is no efficient algorithm for factoring such large integers. If there were such an algorithm, RSA and many other cryptographic protocols would fail.

What about the simpler question of deciding whether such a large integer is prime or composite? In 1977, the same year RSA was invented and making news, Solovay and Strassen discovered an algorithm that could determine whether x is prime in time polynomial in n, the number of digits of x. Their algorithm required the use of a random number generator. The "catch" was that, with small probability, the algorithm might report that x is prime even though it is actually composite; that probability of error could be made as small a positive number as you like, say 10^{-9}.

For 25 years, no one knew how to solve this problem in time polynomial in n without using randomization. Solovay and Strassen's discovery was big news in the algorithm community, and researchers began discovering other problems that could either be solved faster or simpler using random numbers than without.

In 2002, though, it was shown that the primality of x could be determined in time polynomial in n without using randomization.

Quicksort

Solovay and Strassen weren't the first to use a random number generator to speed up an algorithm. A much earlier example was Quicksort, invented by Hoare in

1959. You have probably seen Quicksort, but perhaps not the probabilistic version:

To sort $a_1, a_2, \ldots, a_n$, if $n > 1$,

```
1. Choose $p \sim$ Unif (1,$n$), that is, a random and uniform integer in [1,$n$].
2. Let $L = \{a_i \mid a_i < a_p\}$,
       $E = \{a_i \mid a_i = a_p\}$,
       $G = \{a_i \mid a_i > a_p\}$.
3. Recursively sort and output $L$.
   Output $E$.
   Recursively sort and output $R$.
```

What is the running time of this algorithm? If the random choices in step 1 are very unlucky at every level of recursion, $|L| = 0$ and $|G| = n - 1$ and the total running time is $\theta(n^2)$. What about with ordinary luck? The running time is a random variable (because of the randomness in step 1), and the *expected* running time can be shown to be $O(n \log n)$. This is asymptotically the fastest sorting can be done, though it doesn't beat the running time of Mergesort or Heapsort, which each require no randomization.

A. Conditional expectation

Defn. ~~[illegible]~~ Let X be a r.v. and A an event.

$E[X|A] = \sum_x x P(X=x|A)$.

Law of total expectation:

$E[X] = E[X|A]P(A) + E[X|\bar{A}]P(\bar{A})$.

Proof:

$$
\begin{aligned}
E[X] &= \sum_x x p(x) \\
&= \sum_x x(p(x|A)P(A) + p(x|\bar{A})P(\bar{A})) \\
&= \sum_x x p(x|A)P(A) + \sum_x x p(x|\bar{A})P(\bar{A}) \\
&= \left(\sum_x x p(x|A)\right)P(A) + \left(\sum_x x p(x|\bar{A})\right)P(\bar{A}) \\
&= E[X|A)P(A) + E[X|\bar{A}]P(\bar{A})
\end{aligned}
$$

Randomized algorithms

Quicksort (Hoare, 1959)

A. To sort $a_1, a_2, \ldots, a_n$: If $n > 1$,

1. Choose $p \in \{1, 2, \ldots, n\}$ randomly and uniformly, $p \sim \mathrm{Unif}(1,n)$
2. Let $L = \{a_i \mid a_i < a_p\}$,
   $E = \{a_i \mid a_i = a_p\}$,
   $G = \{a_i \mid a_i > a_p\}$.
3. Recursively sort and output L;
   Output E;
   Recursively sort and output G.

B. If unlucky $|L| = 0$ and $|E| = 1$, and time is $\Theta(n^2)$.
But what is the expected time? For simplicity, assume all a_i distinct.
Let r.v. R be the rank of a_p, i.e., $R = |L|$.
Let r.v. X_n be the number of comparisons to sort $a_1, a_2, \ldots, a_n$.

if $n>1$: $X_n = n-1 + X_R + X_{n-1-R}$.

$E[X_n] = n-1 + E[X_R + X_{n-1-R}]$ (linearity)

$= n-1 + \sum_{i=0}^{n-1} E[X_R + X_{n-1-R} \mid R=i]\, P(R=i)$ (law of total expectation)

$= n-1 + \frac{1}{n}\sum_{i=0}^{n-1} E[X_i + X_{n-1-i}]$

$= n-1 + \frac{1}{n}\left(\sum_{i=0}^{n-1} E[X_i] + \sum_{i=0}^{n-1} E[X_{n-1-i}]\right)$ (linearity)

$= n-1 + \frac{2}{n}\sum_{i=0}^{n-1} E[X_i]$

$n E[X_n] = n(n-1) + 2\sum_{i=0}^{n-1} E[X_i]$

$(n-1) E[X_{n-1}] = (n-1)(n-2) + 2\sum_{i=0}^{n-2} E[X_i]$ (substitute $n \to n-1$)

$n E[X_n] - (n-1) E(X_{n-1}] = 2n-2 + 2E[X_{n-1}]$

$n E[X_n] = 2n-2 + (n+1) E[X_{n-1}]$

$$\frac{E[X_n]}{n+1} = \frac{2n-2}{n(n+1)} + \frac{E[X_{n-1}]}{n} \le \frac{2}{n} + \frac{E[X_{n-1}]}{n}$$

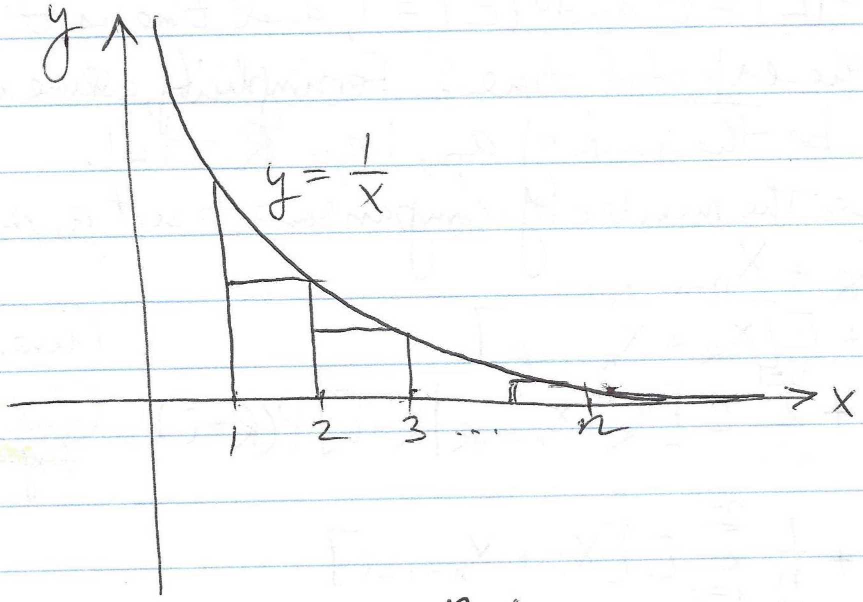
$$\le \frac{2}{n} + \frac{2}{n-1} + \frac{E[X_{n-2}]}{n-1}$$

$$\le \frac{2}{n} + \frac{2}{n-1} + \frac{2}{n-2} + \frac{E[X_{n-3}]}{n-2}$$

$$\le \cdots$$

$$\le \frac{E[X_1]}{2} + 2\sum_{i=2}^{n} \frac{1}{i} = 2\sum_{i=2}^{n} \frac{1}{i} \le 2(H_n - 1) \le 2\ln n$$

$$E[X_n] \le 2(\ln n)(n+1) = 2n\ln n + 2\ln n \le 1.4 n\log_2 n + O(\log n)$$



$$\sum_{i=2}^{n} \frac{1}{i} \le \int_1^n \frac{dx}{x} = \ln x \Big|_1^n = \ln n$$