Today:
- primality testing wrapup
- Monte Carlo vs. Las Vegas
- with high probability
- Quicksort (basic & randomized)
- randomly built binary search trees
- analysis
Miller–Rabin primality test $(N, k)$:
- if $N = 2$ then PRIME
- if $N$ even then COMPOSITE
- if $N$ is a perfect power then COMPOSITE
  $\implies N = a^b$ for $a, b > 1 \implies b \leq \log N = n$
  $\implies O(n)$ powers $= O(n^2)$ multiplies $= O(n^4)$ bit ops.
  [as in Problem 1-3(a)]
- let $N - 1 = u 2^e$ where $u$ is odd
- repeat $k$ times:
  - pick $A \in \{1, 2, \ldots, N-1\}$ uniformly at random
  - if $A^{N-1} \mod N \neq 1$ then COMPOSITE
    (Fermat's Little Theorem: $A^{N-1} = 1 \mod N$ prime)
  - compute $A^{N-1\over 2}, A^{N-1\over 4}, \ldots, A^{u}$ mod $N$
    until reaching value $\neq 1$
    - if it's $\neq -1$ then COMPOSITE
    - PROBABLY PRIME (with probability $1 - 1/2^k$)

Analysis:
- if $N$ is prime, always says PROB. PRIME
- if $N$ is composite but not Charmichael
  $A^{N-1} \mod N \neq 1$ for some $A$
then each Fermat test reveals with prob. $\geq 1/2$
- if $N$ is Charmichael
  then each square-root test reveals with prob. $\geq 1/2$
  [see Lecture 3 notes for proof]
Types of randomized algorithms:

<table>
<thead>
<tr>
<th></th>
<th>poly. time?</th>
<th>correct?</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monte Carlo: [1949]</td>
<td>always</td>
<td>w.h.p.</td>
<td>Miller-Rabin</td>
</tr>
<tr>
<td>Las Vegas: [1979]</td>
<td>w.h.p.</td>
<td>always</td>
<td>Rand-Quicksort</td>
</tr>
</tbody>
</table>

With high probability

Exponentially high probability: $\geq 1 - \frac{1}{2^{cn}}$

for desired constant $c > 0$

(very nice when you can get it)

- e.g. Miller-Rabin with $k = cn$
- $c$ is a parameter influencing the algorithm
- want higher probability? increase $c$

Polynomially high probability: $\geq 1 - \frac{1}{n^c}$

for desired constant $c > 0$

(default meaning of “with high probability”)

$\Rightarrow$ if applied polynomially many times, still no failures w.h.p.: (even if dependent!)

$Pr[\text{fail}_1 \text{ or fail}_2 \text{ or } \cdots \text{ or fail}_n] \\ \leq Pr[\text{fail}_1^3] + Pr[\text{fail}_2^3] + \cdots + Pr[\text{fail}_n^3] \\ \leq \frac{1}{n^c} + \frac{1}{n^c} + \cdots + \frac{1}{n^c} \\ = \frac{n^a}{n^c} = n^{a-c} \\ = \frac{1}{n^{c'}}$ where $c' = c-a$

(i.e. just increase $c$ to compensate)
Quicksort: [C.A.R. Hoare 1962]
- comparison sort
- based on divide & conquer
- but work is in divide instead of combine
- in place: $O(1)$ extra space beyond array
- practical (with tuning)
- different versions:
  - basic: good in average case (random input)
  - randomized: good in expectation, for all inputs
  - deterministic: good in worst case -> [Rec.3]

Algorithm: array $A$

1. divide:
   - pick pivot element $x \in A$
     - basic: $x = A[1]$
     - randomized: $x = A[\text{uniform}\{1,2,\ldots,n\}]$
     - deterministic: $x = \text{median}(A)$
       (see in Recitation 3 how in $O(n)$ time)
   - partition $A$ into elts. $\leq x$, $x$, elts. $> x$:

2. conquer: recursively sort elts. $\leq x$ ($A[1..i-1]$)
3. combine: nothing

3. like Merge Sort
**Partition** \((A, p, q)\): 
- work on subarray \(A[p..q]\)
- basic pivot: first elt. of array

\[
x = A[p] \\
i = p \\
\text{for } j = p+1 \text{ to } q:\n\text{if } A[j] \leq x:\n\quad i += 1 \\

return \(i\)

(Basic) **Quicksort** \((A, p, r)\): 
- sort \(A[p..r]\), \(n = r - p + 1\)

\[
i \geq r: \text{return} \\
q = \text{Partition}(A, p, r) \\
\text{QuickSort}(A, p, q-1) \\
\text{QuickSort}(A, q+1, r) \\
\]

**Example:** 3 1 8 2 6 7 5

```
  3 1 8 2 6 7 5
   1 2         8 6 7 5
    2           6 7 5
     5          7
```

Randomized-QuickSort: use random pivot instead of \(A[p]\)
**BST-build** (A):

\[
T = \text{empty tree} \\
\text{for } i = 1 \text{ to } n: \\
\quad \text{BST-insert}(T, A[i])
\]

- makes exactly the same comparisons as Quicksort, but in a different order

\[ \Rightarrow \text{running time} = \text{sum of depths of nodes} \]

- if we randomly permute the array (= Randomized-Quicksort) then \[ E[\text{running time}] = \Theta(n \log n) \]

\[ \Rightarrow \quad = \Theta(n) \]

*Worst case:* sorted or reverse sorted

```
\text{arithmetic series} \quad T(n) = T(0) + T(n-1) + \Theta(n) \\
\quad = \Theta(n^2)
```

*Best case:* pivot always median

\[ T(n) = 2T(n/2) + \Theta(n) \]

\[ = \Theta(n \log n) \]
Theorem: depth of node in randomly built BST = recursion depth of elt. in Randomized-QuickSort = \(O(\log n)\) with (poly.) high probability ⇒ total Quicksort running time (sum of depths) = \(O(n \log n)\) w.h.p. (by Union Bound)

Proof: consider an element \(z\) in array
- with prob. \(\geq \frac{1}{2}\), size of containing subarray reduces by \(\frac{3}{4}\) factor:
  
  so like flipping a fair coin, and each tail reduces \(n \Rightarrow \frac{3}{4} \cdot n\)
⇒ done after \(\log_{4/3} n = O(\log n)\) tails

so how many coin flips till \(c \log n\) tails?

Aside:
\[
E[\text{\# tails in } d \log n \text{ flips}] = E\left[\sum_{i=1}^{d \log n} F_i\right] \quad \text{where } F_i = \begin{cases} 1 & \text{if flip } i \text{ tails} \\ 0 & \text{else} \end{cases} \quad \text{\uparrow \text{indicator random var.}} \\
= \sum_{i=1}^{d \log n} E[F_i] \quad \left< \text{linearity of expectation} \right> \\
= \sum_{i=1}^{d \log n} \frac{1}{2} \\
= \frac{1}{2} d \log n \quad \text{so looks fine if we set } d \gg 2c
Claim: \( \geq c \lg n \) tails in \( O(\lg n) \) coin flips, w.h.p.

Proof:

\# configurations with \( k \) heads = \( \binom{d \lg n}{k} \)

\# configs. with < \( k \) heads = \( \sum_{i=0}^{k-1} \binom{d \lg n}{i} \)

\[ \frac{d \lg n}{i} = \binom{d \lg n - i + 1}{i} \frac{1}{i} \binom{d \lg n}{i-1} \]  
\( \Rightarrow \) (super)geometric sum

\[ \frac{d \lg n}{i} \geq 2 \text{ for } i \leq \frac{1}{3} \cdot d \lg n \]

\[ \sum_{i=0}^{d \lg n - 1} \binom{d \lg n}{i} < \binom{d \lg n}{d \lg n} \]

assuming \( d \geq 3c \)

\[ \leq \binom{d \lg n + 1}{d \lg n + 1} 2 \]

\[ \leq \binom{d \lg n + 1}{1/3 \cdot d \lg n} 2 \frac{(d/3 - c) \lg n}{d} \]

< all configurations = \( 2^{d \lg n} \)

\[ \Rightarrow \Pr \left( \# \text{ heads } < c \lg n \right) \leq \frac{1}{2} \frac{(d/3 - c) \lg n}{d} \]

\[ = \frac{1}{n} \frac{d/3 - c}{d} \]

\[ = \frac{1}{n} \alpha \text{ for } d = 3(n+c) \]

\[ \Rightarrow \# \text{ heads } \geq c \lg n \text{ with high probability } \square \]