

Probabilistic Algorithms

- An algorithm that uses random numbers
 - sometimes better to choose randomly than to spend a lot of time to compute optimal choice
- May sometimes not have a guaranteed correct result
 - but: can, with several tries, solve any problem
 - otherwise: a heuristic algorithm
- Numerical probabilistic algorithms
 - confidence interval: answer is in a range with a certain probability
- Monte Carlo algorithms
 - give rarely wrong answer
- Las Vegas algorithms
 - rarely say: don't know

© 2005. 324-1-109

Three Kinds of Probabilistic Algorithms

- Algorithm to answer: “When did Columbus discover America?” — run it 3 times
- Numerical:
 - between 1490 and 1500
 - between 1485 and 1495
 - between 1491 and 1501
- Monte Carlo:
 - 1492, 1492, 357 BC
- Las Vegas:
 - 1492, sorry!, 1492

© 2005. 324-1-110

Probabilistic Algorithms

- Expected time
 - average time for a certain instance
- Average expected time
 - average of the expected times of all instances
- Worst-case expected time
 - worst of the expected times of all instances
- Random number generation
 - needed: function that can return uniformly distributed close-to-random numbers
 - typically, in the range between 0 and 1
 - pseudorandom generator: seed + function
 - example:
 - sequence: $x_i = f(x_{i-1})$ $x_0 = \text{seed}$
 - $f(x) = x^2 \bmod p \cdot q$ p, q large primes = 3 mod 4
 - take last bit of result

© 2005. 324-1-111

Prime Number Test

- Monte Carlo algorithm
- Needed for public-key cryptography
 - need large prime numbers
 - based on the fact that it's very hard to find factors of large numbers
- Guaranteed algorithm
 - try numbers $< \sqrt{n}$ as factors
 - runtime:
 - for a 100-digit number: $\sqrt{10^{100}} = 10^{50} = \text{forever}$
 - but a Monte Carlo algorithm can do it!
 - for a 100-digit-number: $\lg 10^{100} = 100 * \lg 10 = 332. \dots$
 - with 10^{-13} error probability

© 2005. 324-1-112

Monte Carlo Prime Number Test

- Theorem by Fermat:
 - n is prime
 - then: $a^{n-1} \bmod n = 1$ $1 \leq a \leq n-1$
 - can use this to test if a number is prime!
- But who says this does not happen if n is not prime??
 - turns out it happens very rarely
 - at least if a is chosen randomly
 - with some numbers it fails always, smallest: $561 = 3 \cdot 11 \cdot 17$
- Algorithm can be improved so that there are no numbers where it always fails

© 2005. 324-1-113

Monte Carlo Prime Test

```
Boolean isPrime( int n ) {  
    int a = randomInteger( 1, n-1 );  
    return expomod( a, n-1, n ) == 1;  
}
```

- int expomod(int a, int n, int z)
 - calculates $a^n \bmod z$
 - time: $O(\log n)$
 - divide-and-conquer algorithm

© 2005. 324-1-114

Las Vegas Quicksort

- Quicksort:
 - picks target[left] as pivot
 - worst case: $O(n^2)$
 - average case: $O(n \log n)$
 - if array already sorted:
 - worst case!
- Let's pick the pivot randomly
 - target[randomInteger(left, right)]
 - worst case expected time: $O(n \log n)$
- Robin Hood effect
 - slow performance gets equally distributed among instances
 - can still be slow if you are unlucky
 - useful if you are using non-random input

© 2005. 324-1-115

Las Vegas Universal Hashing

- Hashing
 - worst case: $O(n^2)$
 - average case: $O(1)$
- Use Robin Hood effect again
 - worst-case expected time: $O(1)$
- Idea:
 - have a set of hash functions
 - pick one randomly
 - assumption: each hash function gives good results on some inputs
 - $\text{hash}(x) = \text{hash}(y)$ happens only for $1/n$ hash functions
 - n size of hashtable
- Example of a universal set of hash functions:
 - $\text{hash}_{ij}(x) = ((i * x + j) \bmod p) \bmod n$
 - i, j : random integers $< p$
 - p prime number $>$ input size

© 2005. 324-1-116