Lecture 27: Randomized Algorithms I

Stefano Tessaro
tessaro@cs.washington.edu
Announcements

• HW8 due Friday, **no extensions**
• Final instructions have been posted
• Practice finals have been posted
  – Discussed in Sections on Thursday
• Review section on Wednesday
• Please complete the class evaluation!
Algorithms can be randomized

Random rand = new Random();
int value = rand.nextInt(50);

double random = Math.random() * 49 + 1;

Random rand = new SecureRandom();
int value = rand.nextInt(50);

Of course, not really random. But outcome can be approximate well by appropriate random variable.

As an aside: For strong cryptographic random generator, finding non-random behavior would be a breakthrough.
Randomized Algorithms – Two types

• **Las Vegas:** Guaranteed correct output
  – Running time is a random variable $T(n)$.
  – Complexity measured in terms of $\mathbb{E}(T(n))$

• **Monte Carlo:** Guaranteed running time
  – Output is a random variable
  – Can make errors
  – Quality measured in terms of error probability
Quicksort – Recap

(Assume for simplicity no repeated elements)

Algorithm $\text{QuickSort}(A)$:  // $A$ is array of size $n$

1) If $n \in \{0,1\}$ then return $A$
2) Choose pivot $p$ from $A$
3) Let $A_0 =$ elements of $A$ which are $< p$
4) Let $A_1 =$ elements of $A$ which are $> p$
5) Return $\text{QuickSort}(A_0) \| p \| \text{QuickSort}(A_1)$

Can be done with $n - 1$ comparisons
Recursion Tree – Good Pivot

\[ [1, 7, 3, 5, 2, 8, 10, 4, 15, 6] \]

\[ [1, 3, 2, 4] \]

\[ [1, 2] \]

\[ [1, 2] \]

\[ [7, 10, 15, 6] \]

\[ [7, 6] \]

\[ [7, 6] \]

\[ \text{even split} = O(\log n) \text{ recursion levels} \]
Recursion Tree – Bad Pivot

\[ [1, 7, 3, 5, 2, 8, 10, 4, 15, 6] \]

\[ [] \]

\[ [7, 3, 5, 2, 8, 10, 4, 15, 6] \]

\[ [2] \]

\[ [7, 5, 8, 10, 4, 15, 6] \]

\[ [] \]

\[ [7, 5, 8, 10, 15, 6] \]

\[ [5] \]

\[ [7, 8, 10, 15] \]

\[ \ldots \]

uneven split = \( \Omega(n) \) recursion levels
A Las Vegas Algorithm – Randomized Quicksort

**Algorithm QuickSort(A):**  // A is array of size n

1) If \( n \in \{0,1\} \) then return \( A \)
2) Pivot \( p \) – random element from \( A \)
3) Let \( A_0 = \) elements of \( A \) which are \(< p\)  
4) Let \( A_1 = \) elements of \( A \) which are \(> p\)
5) Return \( \text{QuickSort}(A_0) \parallel p \parallel \text{QuickSort}(A_1) \)

Can be done with \( n - 1 \) comparisons
Goal – Count comparisons

- $T(n) = \#$ of comparisons on input $n$-element array
  - **Goal:** Compute $E(T(n))$ (Approximation of expected runtime)
- $e_1 < e_2 < \ldots < e_n$ are distinct elements of the array (when sorted)
  - $X_{ij} = 1$ if element $e_i < e_j$ are ever compared, 0 else
  - Two elements can be compared at most once (one of them must be a pivot)!
  - Therefore: $E(T(n)) = E(\sum_{i<j} X_{ij}) = \sum_{i<j} E(X_{ij}) = \sum_{i<j} P(X_{ij} = 1)$
Example: $e_i = 3, e_j = 7$

$[1, 7, 3, 5, 2, 8, 10, 4, 15, 6]$

$[1, 3, 2, 4]$  
$[7, 10, 15, 6]$

Never compared, because first pivot 5 separates them into two different sub-arrays by being between 3 and 7.

Therefore: $X_{ij} = 0$
Example: $e_i = 7$, $e_j = 10$

Comparison:

$[1, 7, 3, 5, 2, 8, 10, 4, 15, 6]$

$[1, 3, 2, 4]$

$[7, 10, 15, 6]$

Compared, because on the same side for pivot 5, then one of them is chosen as pivot.

Therefore: $X_{ij} = 1$
Summarizing

Recall: $e_1 < e_2 < \cdots < e_n$ are distinct elements of the array

$X_{ij}$ is determined by following process:

- Pick (random) pivot $p$
- If $p \in [e_i, e_j]$ then
  - If $p = e_i$ or $p = e_j$ then $X_{ij} = 1$
  - If $p \neq e_i, e_j$ then $X_{ij} = 0$
- Else try another round

$A_k = X_{ij}$ is set after exactly $k$ iterations

\[
\mathbb{P}(X_{ij} = 1 | A_k) = \frac{2}{j - i + 1}
\]

\[
\mathbb{P}(X_{ij} = 1) = \sum_k \mathbb{P}(A_k) \cdot \mathbb{P}(X_{ij} = 1 | A_k) = \frac{2}{j - i + 1} \sum_k \mathbb{P}(A_k) = \frac{2}{j - i + 1}
\]
Randomized Quicksort – Wrapping up

\[ \mathbb{P}(X_{ij} = 1) = \sum_k \mathbb{P}(A_k) \cdot \mathbb{P}(X_{ij} = 1|A_k) = \frac{2}{j - i + 1} \sum_k \mathbb{P}(A_k) = \frac{2}{j - i + 1} \]

\[ \mathbb{E}(T(n)) = \sum_{i<j} \mathbb{P}(X_{ij} = 1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} \]

\[ = \sum_{i=1}^{n-1} \sum_{j=2}^{n-i+1} \frac{2}{j} \leq 2 \sum_{i=1}^{n-1} \sum_{j=1}^{n} \frac{1}{j} = 2 \sum_{i=1}^{n-1} H_n \leq 2nH_n \sim 2n \ln n \]
Monte Carlo Algorithms – Primality Testing

Question: Is an integer $N$ prime?

• Is 7 prime?
• Is 25 prime?
• Is 23 prime?
• Is 7919 prime?
  – Yes! 1000$^{th}$ prime!
• Is 1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413 prime?
  – No ;) [It’s the product of two large primes, very hard to factor!]
Primality – Deterministic Complexity

• Trivial algorithm runs in time (roughly) $O(N \log N)$
  – Check divisibility by every integer $1 < i < N$
  – Can be optimized to $O(\sqrt{N} \log N)$ [Why?]
• Breakthrough result (Agrawal–Kayal–Saxena, 2006): Primality testing in $O((\log N)^{7.5})$
  – Much better, but still not very practical …
• Testing primality is very useful in cryptography (and elsewhere)

Note: Deciding whether an integer is prime or not is much easier than finding the prime factors if it is not prime!
Primality – The Miller-Rabin Test

Running time can be made (almost) $O(\log(N)^2)$

**Algorithm IsPrime($N$):**

- $s = 0, d = N - 1$
- while $d$ is even do
  - $d = \frac{d}{2}; \quad s = s + 1$
- Pick uniformly $b \in \{1, 2, ..., N - 1\}$
- $t = b^d$
- For $i = 1$ to $s$ do
  - If $t = N - 1$ then return $\checkmark$
  - $t = t^2 \mod N$
- Return $\times$

At the end of loop: $N - 1 = 2^s d$

Checks whether $N - 1$ is any of $b^d, b^{2d}, b^{4d}, ..., b^{2^{s-1}d} \mod N$
Theorem. The Miller-Rabin test satisfies the following properties:

- If $N$ is prime, then $\Pr(\text{IsPrime}(N)) = 1$
- If $N$ is not prime, then $\Pr(\text{IsPrime}(N)) \leq 1/4$

For better guarantees:
- Repeat $k$ times, return prime only if all tests return prime.
- If $N$ is not prime, prob. of incorrectly identifying it as prime is $\leq 4^{-k} = 2^{-2k}$
  - E.g., $k = 64$, we have $2^{-128}$