Data Structures and Algorithms
Probability in Computer Science

CS 225
G Carl Evans

April 9, 2023

Department of Computer Science

Slides by Brad Solomon
Fundamentals of Probability (Correction)

Linearity of Expectation: For any two random variables $X$ and $Y$,\[ E[X + Y] = E[X] + E[Y] \]

\[
= \sum_{x} \sum_{y} (x + y)Pr\{X = x, Y = y\}
\]

\[
= \sum_{x} \sum_{y} Pr\{X = x, Y = y\} + \sum_{y} \sum_{x} Pr\{X = x, Y = y\}
\]

\[
= \sum_{x} x \cdot Pr\{X = x\} + \sum_{y} y \cdot Pr\{Y = y\}
\]

does not depend on independence!
Randomization in Algorithms

1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

3. Use randomness inside algorithm to approximate solution in fixed time
Average-Case Analysis: BST

Let $S(n)$ be the average total internal path length over all BSTs that can be constructed by uniform random insertion of $n$ objects.

**Claim:** $S(n)$ is $O(n \log n)$

N=0: 

N=1:
Average-Case Analysis: BST

Let $S(n)$ be the average total internal path length over all BSTs that can be constructed by uniform random insertion of $n$ objects.

N=3:
Average-Case Analysis: BST

Let $S(n)$ be the average total internal path length over all BSTs that can be constructed by uniform random insertion of $n$ objects.

Let $0 \leq i \leq n - 1$ be the number of nodes in the left subtree.

Then for a fixed $i$, $S(n) = (n - 1) + S(i) + S(n - i - 1)$
Average-Case Analysis: BST

Let \( S(n) \) be the **average** total internal path length **over all BSTs** that can be constructed by uniform random insertion of \( n \) objects

\[
S(n) = (n - 1) + \frac{1}{n} \sum_{i=0}^{n-1} S(i) + S(n - i - 1)
\]
Average-Case Analysis: BST

\[ S(n) = (n - 1) + \frac{2}{n} \sum_{i=1}^{n-1} S(i) \]

\[ S(n) = (n - 1) + \frac{2}{n} \sum (ci \ln i) \]

\[ S(n) \leq (n - 1) + \frac{2}{n} \int_{1}^{n} (cx \ln x)dx \]

\[ S(n) \leq (n - 1) + \frac{2}{n} \left( \frac{cn^2}{2} \ln n - \frac{cn^2}{4} + \frac{c}{4} \right) \approx cn \ln n \]
Average-Case Analysis: BST

Summary: All operations are on average $O(log n)$

Randomness:

Assumptions:
Expectation Analysis: Randomized Quicksort
Expectation Analysis: Randomized Quicksort
Expectation Analysis: Randomized Quicksort

In randomized quicksort, the selection of the pivot is random.

**Claim:** The expected comparisons is $O(n \log n)$ *for any input!*

Let $X$ be the total comparisons and $X_{ij}$ be an indicator variable:

$$X_{ij} = \begin{cases} 
1 & \text{if } \text{ith object compared to } j\text{th} \\
0 & \text{if } \text{ith object not compared to } j\text{th} 
\end{cases}$$

Then...
Key Ideas

1. Never compare $X_i$ with $X_i$

2. Never compare $X_i$ and $X_j$ more than once
Expectation Analysis: Randomized Quicksort

Claim: $E[X_{ij}] = \frac{2}{j-i+1}$

Base Case: (N=2)
Expectation Analysis: Randomized Quicksort

Claim: $E[X_{i,j}] = \frac{2}{j-i+1}$  \hspace{1cm} \textbf{Induction:} Assume true for all inputs of $< n$
Expectation Analysis: Randomized Quicksort

\[ E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \]

\[ E[X_{ij}] = \frac{2}{j - i + 1} \]
Expectation Analysis: Randomized Quicksort

\[ E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E[X_{ij}] \]

\[ E[X_{ij}] = \frac{2}{j - i + 1} \]

\[ E[X] = \sum_{i=1}^{n} 2 \left( \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n - i + 1} \right) \]

\[ E[X] = \sum_{i=1}^{n} 2(H_{n-1} - 1) \leq 2n \cdot H_n \leq 2n \ln n \]
Expectation Analysis: Randomized Quicksort

**Summary:** Randomized quick sort is $O(n \log n)$ regardless of input

**Randomness:**

**Assumptions:**
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}
Probabilistic Accuracy: Fermat primality test

Pick a random $a$ in the range $[2, p - 2]$

If $p$ is prime and $a$ is not divisible by $p$, then $a^{p-1} \equiv 1 (mod p)$

But… *sometimes* if $n$ is composite and $a^{n-1} \equiv 1 (mod n)$
Probabilistic Accuracy: Fermat primality test

<table>
<thead>
<tr>
<th></th>
<th>$a^{p-1} \equiv 1 \pmod{p}$</th>
<th>$a^{p-1} \not\equiv 1 \pmod{p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$ is prime</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p$ is not prime</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Probabilistic Accuracy: Fermat primality test

Let’s assume \( \alpha = .5 \)

First trial: \( a = a_0 \) and prime test returns ‘prime!’

Second trial: \( a = a_1 \) and prime test returns ‘prime!’

Third trial: \( a = a_2 \) and prime test returns ‘not prime!’

Is our number prime?

What is our \textbf{false positive} probability? Our \textbf{false negative} probability?
Probabilistic Accuracy: Fermat primality test

Summary: Randomized algorithms can also have fixed (or bounded) runtimes at the cost of probabilistic accuracy.

Randomness:

Assumptions:
Types of randomized algorithms

A **Las Vegas** algorithm is a randomized algorithm which will always give correct answer if run enough times but has no fixed runtime.

A **Monte Carlo** algorithm is a randomized algorithm which will run a fixed number of iterations and may give the correct answer.
Next Class: Randomized Data Structures

Sometimes a data structure can be too ordered / too structured

Randomized data structures rely on expected performance

Randomized data structures ‘cheat’ tradeoffs!