Probabilistic Algorithms: What, Why, and How

A Deep Dive into Randomness in Computing

Sailesh Dahal

May 12, 2025

Kathmandu University

Outline

What are Probabilistic Algorithms?

What is a Probabilistic Algorithm?

Definition
An algorithm that makes random choices during execution to influence its behavior or output.

What is a Probabilistic Algorithm?

Definition
An algorithm that makes random choices during execution to influence its behavior or output.

- Output or performance may vary on different runs
- Two main types: Las Vegas (always correct, time varies), Monte Carlo (time fixed, may err)

Types of Probabilistic Algorithms

- **Las Vegas:** Always correct, random running time

Types of Probabilistic Algorithms

- **Las Vegas:** Always correct, random running time
- **Monte Carlo:** Fixed running time, may give incorrect result with small probability

Types of Probabilistic Algorithms

- **Las Vegas:** Always correct, random running time
- **Monte Carlo:** Fixed running time, may give incorrect result with small probability
- **Example:** Randomized Quicksort (Las Vegas)

Types of Probabilistic Algorithms

- **Las Vegas:** Always correct, random running time
- **Monte Carlo:** Fixed running time, may give incorrect result with small probability
- **Example:** Randomized Quicksort (Las Vegas)
- **Example:** Primality testing (Monte Carlo)

Why Probabilistic Algorithms?

Why Randomness?

Motivation

- Simpler algorithms

Why Randomness?

Motivation

- Simpler algorithms
- Better expected performance

Why Randomness?

Motivation

- Simpler algorithms
- Better expected performance
- Avoid worst-case scenarios

Why Randomness?

Motivation

- Simpler algorithms
- Better expected performance
- Avoid worst-case scenarios
- Useful for large-scale and distributed systems

Real-World Motivation

- Web search (PageRank)

Real-World Motivation

- Web search (PageRank)
- Load balancing (power of two choices)

Real-World Motivation

- Web search (PageRank)
- Load balancing (power of two choices)
- Hashing (universal hash functions)

Real-World Motivation

- Web search (PageRank)
- Load balancing (power of two choices)
- Hashing (universal hash functions)
- Primality testing (Miller-Rabin)

How do Probabilistic Algorithms Work?

How: Randomization in Algorithms

Key Idea
Use random choices to influence the algorithm's path or output.

How: Randomization in Algorithms

Key Idea
Use random choices to influence the algorithm's path or output.

- Random pivot in Quicksort

How: Randomization in Algorithms

Key Idea
Use random choices to influence the algorithm's path or output.

- Random pivot in Quicksort
- Random walks in graphs

How: Randomization in Algorithms

Key Idea
Use random choices to influence the algorithm's path or output.

- Random pivot in Quicksort
- Random walks in graphs
- Random sampling

Example: Randomized Quicksort

Randomized Quicksort: Step 1 (Initial Array)

Consider the array:

7	5	9	1	3	4	8	6

Randomized Quicksort: Step 1 (Initial Array)

Consider the array:

7	5	9	1	3	4	8	6

Suppose the random pivot chosen is 5:

7	5	9	1	3	4	8	6

Randomized Quicksort: Step 2 (Partitioning)

Partition the array into:

- Left: 1, 3, 4 (all < 5)
- Middle: 5 (pivot)
- Right: 7, 9, 8, 6 (all > 5)

Randomized Quicksort: Step 2 (Partitioning)

Partition the array into:

- Left: 1, 3, 4 (all < 5)
- Middle: 5 (pivot)
- Right: 7, 9, 8, 6 (all > 5)

1	3	4	5	7	9	8	6

Randomized Quicksort: Step 3 (Left Subarray)

Recurse on the left subarray $[1, 3, 4]$.

Randomized Quicksort: Step 3 (Left Subarray)

Recurse on the left subarray $[1, 3, 4]$. Suppose the random pivot is 3:

1	3	4

Randomized Quicksort: Step 3 (Left Subarray)

Recurse on the left subarray $[1, 3, 4]$. Suppose the random pivot is 3:

1	3	4

Partition:

- Left: 1
- Middle: 3
- Right: 4

Randomized Quicksort: Step 4 (Right Subarray)

Recurse on the right subarray $[7, 9, 8, 6]$.

Randomized Quicksort: Step 4 (Right Subarray)

Recurse on the right subarray $[7, 9, 8, 6]$. Suppose the random pivot is 8:

7	8	9	6

Randomized Quicksort: Step 4 (Right Subarray)

Recurse on the right subarray $[7, 9, 8, 6]$. Suppose the random pivot is 8:

7	8	9	6

Partition:

- Left: 7, 6
- Middle: 8
- Right: 9

Randomized Quicksort: Step 5 (Continue Recursion)

Continue recursively on each subarray until all are of length 1.

Randomized Quicksort: Step 5 (Continue Recursion)

Continue recursively on each subarray until all are of length 1. The final sorted array is:

1	3	4	5	6	7	8	9

Quicksort Implementation (Python)

```python
import random
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = random.choice(arr)
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)
```

Analysis: Recurrence and Expectation

Quicksort Recurrence

Expected Comparisons

$$T(n) \le n + \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i))$$

Quicksort Recurrence

Expected Comparisons

$$T(n) \leq n + \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i))$$

Base case: $T(1) = 0$

Quicksort Recurrence

Expected Comparisons

$$T(n) \leq n + \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i))$$

Base case: $T(1) = 0$ Solution: $T(n) = O(n \log n)$

Slick Analysis: Indicator Variables

- $Q(A)$: Number of comparisons on input A

Slick Analysis: Indicator Variables

- $Q(A)$: Number of comparisons on input A
- X_{ij}: Indicator for whether elements i and j are compared

Slick Analysis: Indicator Variables

- $Q(A)$: Number of comparisons on input A
- X_{ij}: Indicator for whether elements i and j are compared
- $E[Q(A)] = \sum_{i<j} Pr[R_{ij}]$

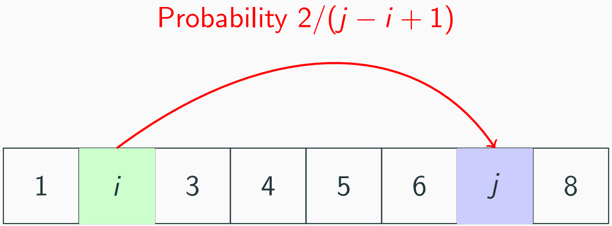
Slick Analysis: Indicator Variables

- $Q(A)$: Number of comparisons on input A
- X_{ij}: Indicator for whether elements i and j are compared
- $E[Q(A)] = \sum_{i<j} Pr[R_{ij}]$
- $Pr[R_{ij}] = \frac{2}{j-i+1}$

Slick Analysis: Indicator Variables

- $Q(A)$: Number of comparisons on input A
- X_{ij}: Indicator for whether elements i and j are compared
- $E[Q(A)] = \sum_{i<j} Pr[R_{ij}]$
- $Pr[R_{ij}] = \frac{2}{j-i+1}$

Visualization



Harmonic Numbers in Analysis

Harmonic Number
$H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

Harmonic Numbers in Analysis

Harmonic Number
$H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

Summation in Quicksort
$E[Q(A)] \leq 2nH_n = O(n \log n)$

Example: Coin Toss Probability

Coin Toss Example

Experiment

John tosses a biased coin (probability p of heads) to decide whether to wear a tie. If heads, tosses a fair coin to pick red or blue tie.

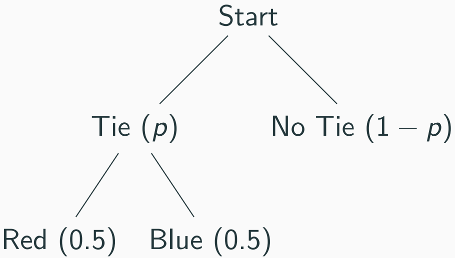
Coin Toss Example

Experiment
John tosses a biased coin (probability p of heads) to decide whether to wear a tie. If heads, tosses a fair coin to pick red or blue tie.

Question
What is the probability John wears a red tie on the first day he wears a tie?

Experiment
John tosses a biased coin (probability p of heads) to decide whether to wear a tie. If heads, tosses a fair coin to pick red or blue tie.

Question
What is the probability John wears a red tie on the first day he wears a tie?



Random Bits in Practice

Where do Random Bits Come From?

- Hardware random number generators

Where do Random Bits Come From?

- Hardware random number generators
- Pseudo-random number generators (PRNGs)

Where do Random Bits Come From?

- Hardware random number generators
- Pseudo-random number generators (PRNGs)
- Physical phenomena (thermal noise, radioactive decay)

Where do Random Bits Come From?

- Hardware random number generators
- Pseudo-random number generators (PRNGs)
- Physical phenomena (thermal noise, radioactive decay)
- In practice, PRNGs are sufficient for most applications

Probabilistic Data Structures

What is a Probabilistic Data Structure?

Definition
Data structures that use randomization or probabilistic techniques to achieve space or time efficiency, often allowing for small errors (e.g., false positives).

What is a Probabilistic Data Structure?

Definition
Data structures that use randomization or probabilistic techniques to achieve space or time efficiency, often allowing for small errors (e.g., false positives).

- Useful for large-scale data, streaming, or approximate answers
- Examples: Bloom filter, Count-Min Sketch, HyperLogLog

Bloom Filter: What and Why?

What is a Bloom Filter?
A space-efficient, probabilistic data structure for set membership queries.

Bloom Filter: What and Why?

What is a Bloom Filter?
A space-efficient, probabilistic data structure for set membership queries.

- Answers: "Is x in the set?"
- May return false positives, but never false negatives
- Very compact compared to hash sets

How Does a Bloom Filter Work?

1. Start with a bit array of m bits, all set to 0

How Does a Bloom Filter Work?

1. Start with a bit array of m bits, all set to 0
2. Use k independent hash functions

How Does a Bloom Filter Work?

1. Start with a bit array of m bits, all set to 0
2. Use k independent hash functions
3. To add an element, set k bits (one per hash) to 1

How Does a Bloom Filter Work?

1. Start with a bit array of m bits, all set to 0
2. Use k independent hash functions
3. To add an element, set k bits (one per hash) to 1
4. To check membership, test if all k bits are 1

Bloom Filter Example

Suppose $m = 8$ bits, $k = 2$ hash functions, and we insert $\{\text{cat}, \text{dog}\}$.

Bloom Filter Example

Suppose $m = 8$ bits, $k = 2$ hash functions, and we insert $\{\text{cat}, \text{dog}\}$.

Bit Array After Insertion

1	0	1	1	0	0	0	1

Bloom Filter Example

Suppose $m = 8$ bits, $k = 2$ hash functions, and we insert $\{\text{cat}, \text{dog}\}$.

Bit Array After Insertion

1	0	1	1	0	0	0	1

- To check if "cat" is in the set, hash and check the corresponding bits
- If all are 1, answer is "possibly in set"; if any is 0, "definitely not in set"

Bloom Filter: Trade-offs

- **False positives:** May say "in set" when not

Bloom Filter: Trade-offs

- **False positives:** May say "in set" when not
- **No false negatives:** Never says "not in set" if it is

Bloom Filter: Trade-offs

- **False positives:** May say "in set" when not
- **No false negatives:** Never says "not in set" if it is
- **Space efficient:** Much smaller than explicit set

Bloom Filter: Trade-offs

- **False positives:** May say "in set" when not
- **No false negatives:** Never says "not in set" if it is
- **Space efficient:** Much smaller than explicit set
- **No deletions:** Standard Bloom filters do not support removing elements

References i