ALGORITHMS AND DATA STRUCTURES II

Lecture 12

Randomized Algorithms,

Lecturer: K. Markov

markov@u-aizu.ac.jp

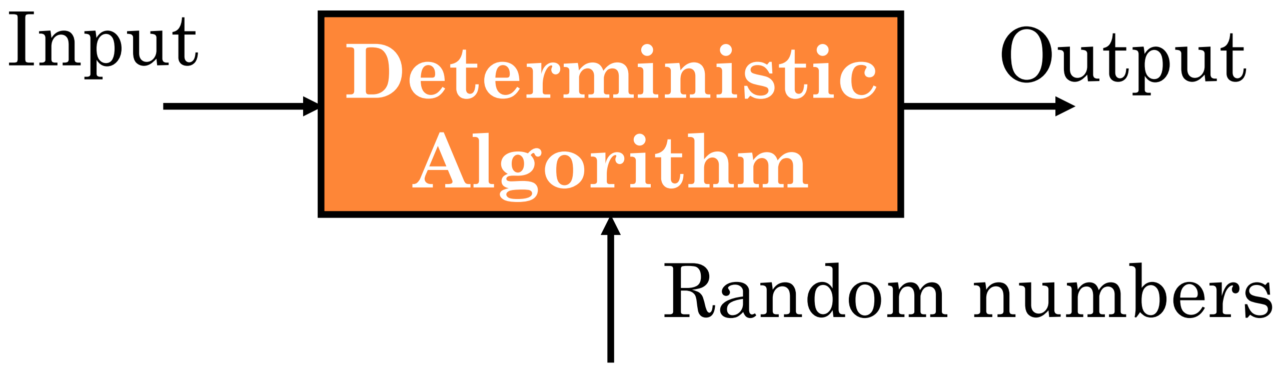
RANDOMIZED ALGORITHMS

- An algorithm is called **randomized** if it uses:
  - a random number to make a decision at least once during the computation and
  - its computation time is determined not only by the input data but also by the values of a random number generator.

DETERMINISTIC ALGORITHMS

- Deterministic algorithm always solves the problem correctly.
- Deterministic algorithm runs at least O(…) fast, i.e. for the worst case.

RANDOMIZED ALGORITHMS



- Randomized algorithm takes a source of random numbers and makes random choices during execution.
- Behavior (running time) can vary even with a fixed input.

RANDOMIZED ALGORITHMS

- Why use randomness?
  - To avoid worst-case behavior: randomness can (probabilistically) guarantee average case behavior.
  - To achieve efficient approximate solutions to intractable problems.

RANDOMIZED ALGORITHMS

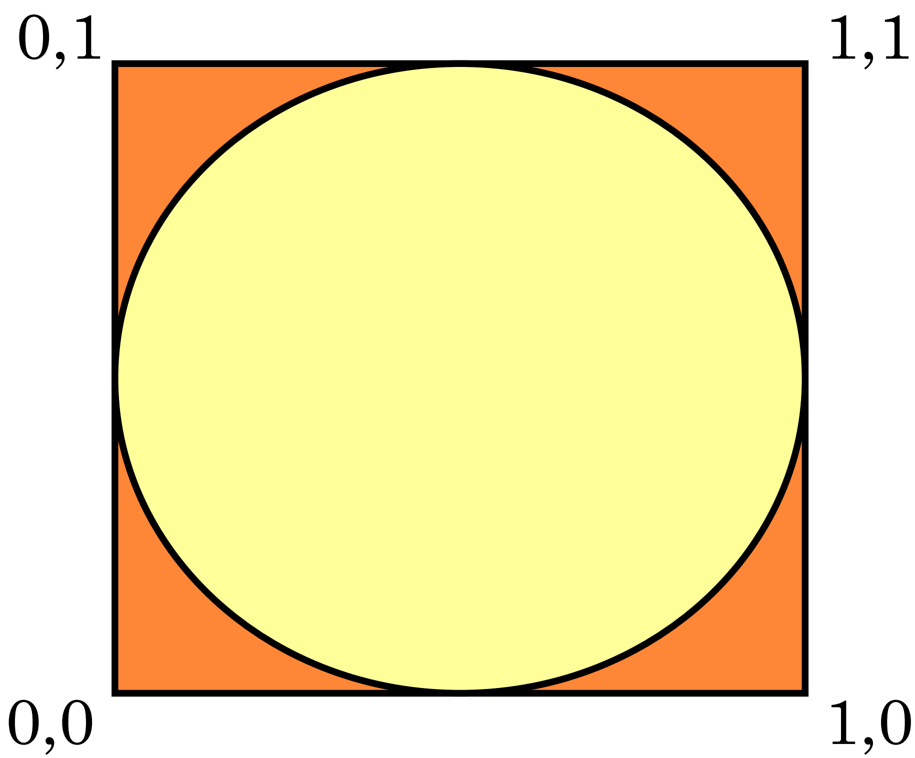
- Two main types:
  - **Monte Carlo**
    - Runs for a fixed number of steps.
    - If there is no solution, returns "no".
    - If there is a solution, finds it with some probability (i.e. > 0.5).
  - **Las Vegas**
    - Always produces the correct answer.
    - Running time is random.

RANDOMIZED ALGORITHMS

- Example: Find π using randomized algorithm.

Square area = 1
Circle area= π/4

The probability a random point in square is in circle = π/4



π = 4 * points in circle / points

RANDOMIZED ALGORITHMS

- Example: Find π using randomized algorithm.

```
def PI (n):
    inCircle = 0
    for i=1 to n:
        x = rand()
        y = rand()
        d = (x - 0.5)^2 + (y - 0.5)^2
        if d < 0.25:
            inCircle = inCircle + 1
    return 4 * inCircle / n
```

RANDOMIZED ALGORITHMS

- Example: Find π using randomized algorithm (result)

n: 1	4.0	4.0	0.0
n: 2	2.0	4.0	4.0
n: 4	3.0	4.0	3.0
...			
n: 64	3.0625	3.125	3.0625
...			
n: 1024	3.16796875	3.13671875	3.1640625
...			
n: 16384	3.12622070	3.14038085	3.1279296
n: 131072	3.13494873	3.14785766	3.1376647
n: 1048576	3.14015579	3.14387893	3.1411247

RANDOMIZED ALGORITHMS

- Example: Randomized **quicksort** algorithm.

  - It is a divide-and-conquer method of sorting.
  - It works by partitioning the sequence S into two parts, then sorting the parts independently.

RANDOMIZED ALGORITHMS

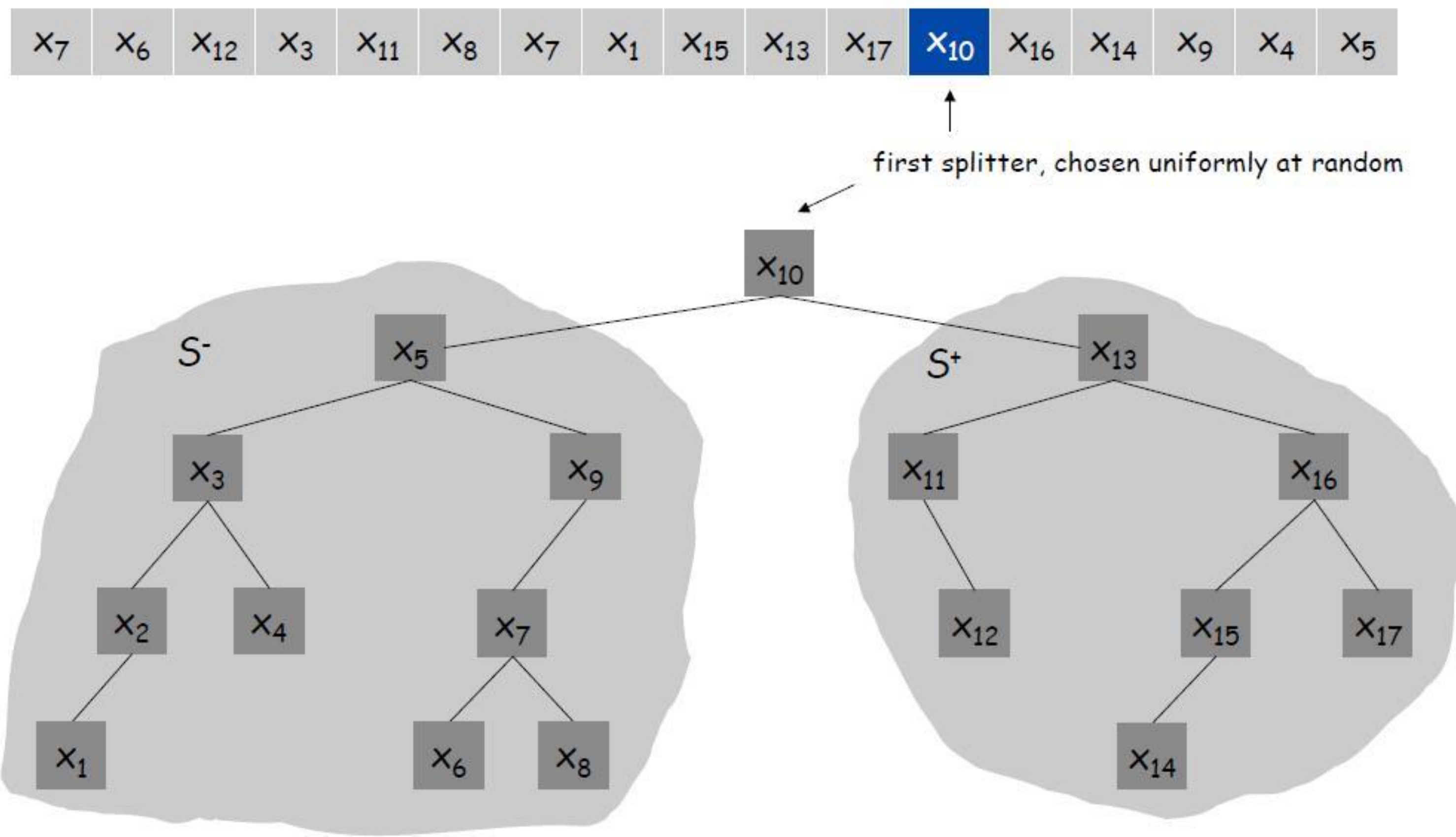
- Example: Randomized **quicksort**.

```
def quicksort (S):
    if |S| = 0: return
    choose a_i ∈ S randomly
    for each a ∈ S:
        if a < a_i : put a in S-
        if a > a_i : put a in S+
    quicksort (S-)
    print a_i
    quicksort (S+)
```

RANDOMIZED ALGORITHMS

- Example: Randomized **quicksort**.

Binary tree representation of splitters.



RANDOMIZED ALGORITHMS

- Example: Randomized **quicksort**.
  - **Running time for deterministic QS.**
    - [Best case.] Select the median element as the splitter: quicksort makes $\Theta(n \log n)$ comparisons.
    - [Worst case.] Select the smallest element as the splitter: quicksort makes $\Theta(n^2)$ comparisons.
  - **Randomized QS**. Protect against worst case by choosing splitter at random.

RANDOMIZED ALGORITHMS

- Example: Primality test.
  - The primality test provides the probability of whether or not a **large number** is prime.
  - Several theorems including Fermat's theorem provide idea of primality test.
  - Cryptography schemes such as RSA algorithm are heavily based on primality test.

RANDOMIZED ALGORITHMS

- Example: Primality test.
  - A Naïve Algorithm:
    - Pick any integer $\mathbf{P}$ that is greater than $\mathbf{2}$.
    - Try to divide $\mathbf{P}$ by all odd integers starting from $\mathbf{3}$ to square root of $\mathbf{P}$.
    - If $\mathbf{P}$ is divisible by any one of these odd integers, we can conclude that $\mathbf{P}$ is not prime.
    - The worst case is that we have to go through all odd number testing cases up to square root of $\mathbf{P}$.
    - Time complexity is $\mathbf{O}(\sqrt{\boldsymbol{n}})$

RANDOMIZED ALGORITHMS

- Example: Primality test.
  - Fermat's Theorem: *If P is prime and $0 < A < P$ then $A^{P-1} = 1 \pmod{P}$.*
  - Given a number P, we can choose a particular A (e.g., 2) with $0 < A < P$ and calculate $A^{p-1} \pmod{P}$:
    - If $A^{P-1} \neq 1 \pmod{P}$ then P is not prime.
    - If $A^{P-1} = 1 \pmod{P}$ then P is **probably** prime.

RANDOMIZED ALGORITHMS

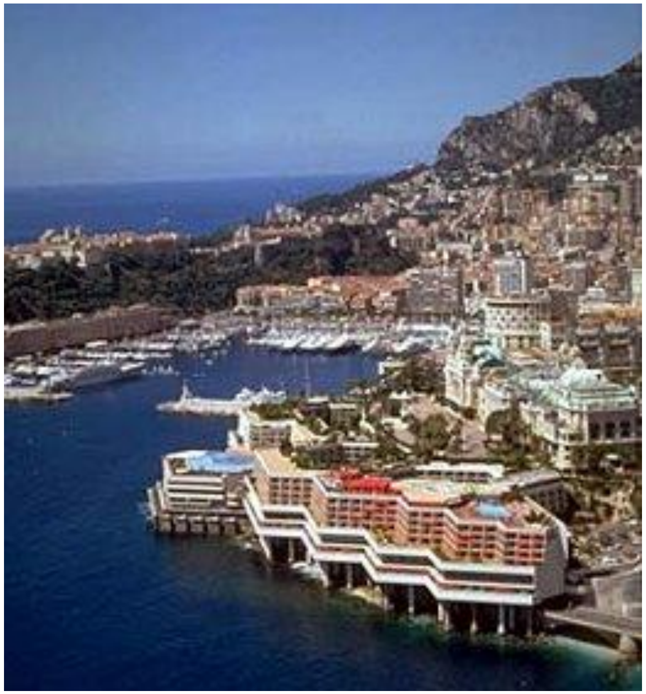
- Example: Primality test.
  - We can randomize the above algorithm by choosing $1 < A < P$ at random.
  - For an A chosen at random if $A^{p\text{-}1} \neq 1 \pmod{P}$ then we say P is not prime otherwise we accept P is prime.
    - If actually P is not a prime – it is a mistake.
    - The probability of mistake is k in each computation. If for m independent computations the algorithm says that P is prime, the probability that P is a prime is at least $1\text{-}k^m$.

RANDOMIZED ALGORITHMS

- Advantages of randomized algorithms:

  - Simplicity.
  - Performance.
  - For many problems, a randomized algorithm is the simplest, the fastest, or both.

RANDOMIZED ALGORITHMS



Monte Carlo or Las Vegas?

RANDOMIZED ALGORITHMS

- Applications and scope:
  - Number-theoretic algorithms:
    - **Primality testing** (Monte Carlo).
  - Data structures:
    - Sorting – **quicksort** (Las Vegas)
    - Order statistics, searching, computational geometry.
  - Algebraic identities:
    - Polynomial and matrix identity verification. Interactive proof systems.

RANDOMIZED ALGORITHMS

- Applications and scope:
  - Mathematical programming:
    - Faster algorithms for linear programming. Rounding linear program solutions to integer program solutions.
  - Graph algorithms:
    - Minimum spanning trees, shortest paths, minimum cuts.
  - Counting and enumeration:
    - Matrix permanent. Counting combinatorial structures.

RANDOMIZED ALGORITHMS

- Applications and scope:
  - Parallel and distributed computing:
    - Deadlock avoidance, distributed consensus.
  - Probabilistic existence proofs:
    - Show that a combinatorial object arises with non-zero probability among objects drawn from a suitable probability space.

That's All for Today!