# 15-251: Great Ideas in Theoretical Computer Science

Lecture 19: Randomized Algorithms I

*Nov 1st, 2018*

---

**Most Common 3 Random Variables**

---

## Bernoulli Random Variable

$\boldsymbol{X} \sim \text{Bernoulli}(p)$ means:

"$\boldsymbol{X}$ is a Bernoulli random variable with success probability $p$."

X = Bernoulli(p)

$\Pr[\boldsymbol{X} = 1] =$

$\Pr[\boldsymbol{X} = 0] =$

So $\text{range}(\boldsymbol{X}) =$

<u>Check:</u>

$\mathbf{E}[\boldsymbol{X}] =$

## Binomial Random Variable

$\boldsymbol{X} \sim \mathrm{Binomial}(n, p)$ means:

$\boldsymbol{X} = \boldsymbol{X}_1 + \boldsymbol{X}_2 + \cdots + \boldsymbol{X}_n$

where $\boldsymbol{X}_i \sim \mathrm{Bernoulli}(p)$ for all $i \in \{1, 2, \ldots, n\}$,

and the $\boldsymbol{X}_i$'s are independent.

So $\mathrm{range}(\boldsymbol{X}) = \{0, 1, 2, \ldots, n\}$

<u>Check:</u>

$\Pr[\boldsymbol{X} = i] =$

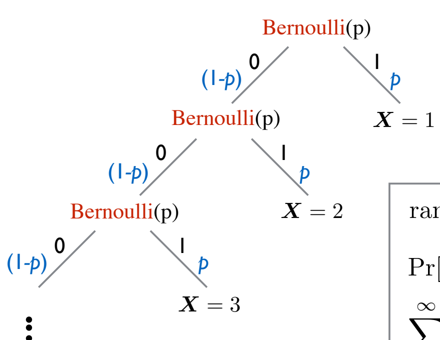
$\mathbf{E}[\boldsymbol{X}] =$

## Geometric Random Variable

$\boldsymbol{X} \sim \mathrm{Geometric}(p)$ means:

"number of p-biased coin flips until we see **H** for the first time."

## Geometric Random Variable




$\mathrm{range}(\boldsymbol{X}) = \{1, 2, 3, \ldots\}$

$\Pr[\boldsymbol{X} = i] =$

$\sum_{i=1}^{\infty} \Pr[\boldsymbol{X} = i] = 1$

(geometric sum)

$\mathbf{E}[\boldsymbol{X}] =$

## Markov's Inequality

A non-negative random variable $X$ is rarely much bigger than its expectation $\mathbf{E}[X]$.

**Theorem:**

Let $X$ be a random variable that is always non-negative. Then for any $c \geq 1$,

**Randomized Algorithms**

## Randomness and algorithms

**How can randomness be used in computation?**

Given some algorithm that solves a problem:

(i) the input can be chosen randomly

(ii) the algorithm can make random choices

Which one will we focus on?

## Randomness and algorithms

**What is a <u>randomized algorithm</u>?**

A *randomized algorithm* is an algorithm that is allowed to "***flip a coin***" (i.e., has access to random bits).

**<u>In 15-251:</u>**

A randomized algorithm is an algorithm that is allowed to call:

## Randomness and algorithms

A randomized algorithm computes a decision/computational problem if ___________________
(what?)

## Deterministic vs Randomized

**Deterministic**

```
def A(x):
    y = 1
    if(y == 0):
        while(x > 0):
            x = x - 1
    return x+y
```

**Randomized**

```
def A(x):
    y = Bernoulli(0.5)
    if(y == 0):
        while(x > 0):
            x = x - 1
    return x+y
```

For any <u>fixed</u> input (e.g. x = 3):

Deterministic:

- the output
- the running time

Randomized:

- the output
- the running time

## Deterministic vs Randomized

A **deterministic algorithm** $A$ computes $f : \Sigma^* \to \Sigma^*$ in time $T(n)$ means:

- **correctness**: $\forall x \in \Sigma^*$,

- **running time**: $\forall x \in \Sigma^*$,

Note: we require worst-case guarantees for correctness and run-time.

## Deterministic vs Randomized

**A Try**

A **randomized algorithm** $A$ computes $f : \Sigma^* \to \Sigma^*$ in time $T(n)$ means:

- **correctness**: $\forall x \in \Sigma^*$ ,

- **running time**: $\forall x \in \Sigma^*$,

**Is this interesting?**

$\boxed{\forall x \in \Sigma^*}$

| | | Correctness | Run-time |
|---|---|---|---|
| Deterministic | | | |
| Randomized | Type 0 | | |
| | Type 1 | | |
| | Type 2 | | |
| | Type 3 | | |

Type 0:
Type 1:
Type 2:
Type 3:

## Example: Battleship

**Input**: An array B with **n/4** 1's and **3n/4** 0's.
**Output**: An index that contains a 1.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

## Example: Battleship

**Input**: An array B with **n/4** 1's and **3n/4** 0's.
**Output**: An index that contains a 1.

| Deterministic | Randomized | |
|---|---|---|
| | Type 1 (Monte Carlo) | Type 2 (Las Vegas) |

## Example: Battleship

**Input**: An array B with **n/4** 1's and **3n/4** 0's.
**Output**: An index that contains a 1.

| | Correctness | Run-time |
|---|---|---|
| **Monte Carlo** | | |
| **Las Vegas** | | |

## Formal Definitions

## Formal Definition: Deterministic

Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

We say that deterministic algorithm $A$ computes $f$ in time $T(n)$ if:

$$\boxed{\forall x \in \Sigma^*,} \qquad A(x) = f(x)$$

$$\boxed{\forall x \in \Sigma^*,} \qquad \#\text{ steps } A(x) \text{ takes is } \leq T(|x|).$$

**Picture:**


**Deterministic:**

## Formal Definition: Monte Carlo

Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

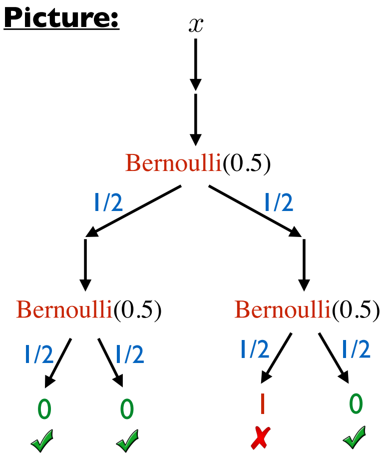
We say that randomized algorithm $A$ is a $T(n)$-time **Monte Carlo algorithm** for $f$ with $\epsilon$ error probability if:

$\forall x \in \Sigma^*,$

$\forall x \in \Sigma^*,$

---

**Picture:**




**Monte Carlo:**

Each input $x$ induces a probability tree.

---

## Formal Definition: Las Vegas

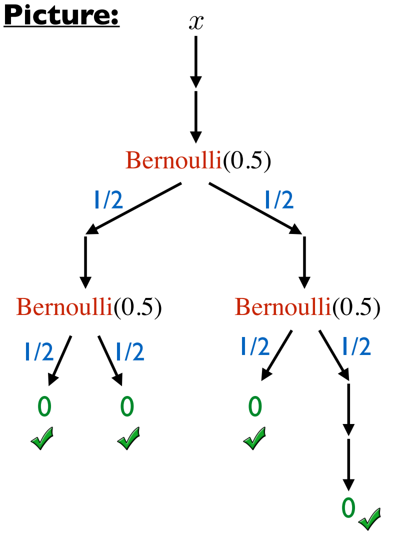
Let $f : \Sigma^* \to \Sigma^*$ be a computational problem.

We say that randomized algorithm $A$ is a $T(n)$-time **Las Vegas algorithm** for $f$ if:

$\forall x \in \Sigma^*,$

$\forall x \in \Sigma^*,$

**Picture:**




**Las Vegas:**

Each input $x$ induces a probability tree.

**More Examples**

**3 IMPORTANT PROBLEMS**

**Integer Factorization**

Input: integer N

Ouput: a prime factor of N

**isPrime**

Input: integer N

Ouput: True if N is prime.

**Generating a (random) n-bit prime**

Input: integer n

Ouput: a (random) n-bit prime

**Most crypto systems start like:**

- pick two random n-bit primes P and Q.
- let N = PQ. (N is some kind of a "key")
- (*more steps…*)

We should be able to do efficiently the following:

- check if a given number is prime.
- generate a random prime.

We should **not** be able to do efficiently the following:

- given N, find P and Q. (the system is broken if we can do this!!!)

## isPrime

```
def isPrime(N):
    if (N < 2):  return False
    maxFactor = round(N**0.5)
    for factor in range(2, maxFactor+1):
        if (N % factor == 0):  return False
    return True
```

Problems:

## isPrime

**Amazing result from 2002:**

There is a poly-time algorithm for isPrime.

Agrawal, Kayal, Saxena

However, best known implementation is ~ $O(n^6)$ time.
Not feasible when $n = 2048$.

## isPrime

So that's **not** what we use in practice.

Everyone uses the Miller-Rabin algorithm (1975).

CMU Professor

The running time is:

Why is the previous result a breakthrough?

## Generating an n-bit prime

```
repeat:
   let N be a random n-bit number
   if isPrime(N): return N
```

**Prime Number Theorem (informal):**

$\Longrightarrow$ expected run-time of the above algorithm ~

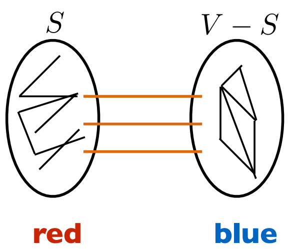
No poly-time deterministic algorithm is known to generate an $n$-bit prime!!!

**Randomized Algorithms meet Approximation Algorithms**

Randomized approximation algorithms for optimization problems

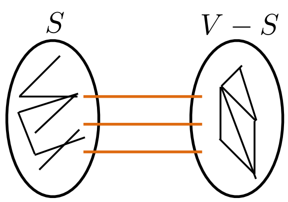
## Cut Problems

**Max Cut Problem** (Ryan O'Donnell's favorite problem):
Given a connected graph $G = (V, E)$ ,
color the vertices **red** and **blue** so that the number of edges with two colors (e = {**u**,**v**}) is maximized.




## Cut Problems

**Max Cut Problem** (Ryan O'Donnell's favorite problem):
Given a connected graph $G = (V, E)$,
find a subset $S \subset V$ such that
number of edges from $S$ to $V - S$ is maximized.

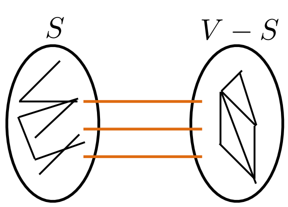



size of the cut = # edges from $S$ to $V - S$.

Max Cut Problem is **NP**-hard!

## Cut Problems

**Min Cut Problem** (my favorite problem):
Given a connected graph $G = (V, E)$,
find a non-empty subset $S \subset V$ such that
number of edges from $S$ to $V - S$ is ***minimized***.




size of the cut = # edges from $S$ to $V - S$.

**Randomized Approximation Algorithm for Max Cut**

**<u>Most Useful Equality in Probability Theory:</u>**

**Linearity of Expectation**

$$\mathbf{E}[\boldsymbol{X} + \boldsymbol{Y}] = \mathbf{E}[\boldsymbol{X}] + \mathbf{E}[\boldsymbol{Y}]$$

$\boldsymbol{X}$ and $\boldsymbol{Y}$ need not be independent!

($\mathbf{E}[\boldsymbol{X} \cdot \boldsymbol{Y}] = \mathbf{E}[\boldsymbol{X}] \cdot \mathbf{E}[\boldsymbol{Y}]$ not always true!)

**<u>Most Useful Type of Random Variable:</u>**

**Indicator Random Variable**

Event —> Random Variable

Let $A$ be an event. The *indicator r.v.* for $A$ is:

$$\boldsymbol{I}_A = \begin{cases} \end{cases}$$

$\Pr[\boldsymbol{I}_A = 1] =$

$\mathbf{E}[\boldsymbol{I}_A] =$

## High Level Idea

Want to compute $\mathbf{E}[X]$:

Write $X = I_1 + I_2 + \cdots + I_n$. (sum of indicator r.v.'s)

Then $\mathbf{E}[X] =$

$=$

$=$

$=$

## Approximation Alg. for Max Cut

## Analysis

**NEXT TIME:**

**Monte Carlo Algorithm for Min Cut**