## Types of Randomized Algorithms

<table>
<thead>
<tr>
<th>Type</th>
<th>Correctness</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 0</td>
<td>always (w.p. 1)</td>
<td>always $\leq T(n)$</td>
</tr>
<tr>
<td>Type 1</td>
<td>w.h.p.</td>
<td>always $\leq T(n)$</td>
</tr>
<tr>
<td>Type 2</td>
<td>always</td>
<td>w.h.p. $\leq T(n)$</td>
</tr>
<tr>
<td>Type 3</td>
<td>w.h.p.</td>
<td>w.h.p. $\leq T(n)$</td>
</tr>
</tbody>
</table>

- **Type 0**: may as well be deterministic
- **Type 1**: “Monte Carlo algorithm”
- **Type 2**: “Las Vegas algorithm”
- **Type 3**: Can be converted to type 1. (exercise)
CASE STUDY

Randomized Algorithms for Cut Problems
Max Cut 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!
Simple algorithm:
1) For each vertex, flip a coin
2) If Heads, place it in S

Expected number of cut edges: $|E|/2$
Min Cut Problem

Given a connected graph $G = (V, E)$, find a non-empty subset $S \subset V$ such that the number of edges from $S$ to $V - S$ is minimized.

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

(how many possible “cuts” are there?)
Randomized Monte Carlo Algorithm for Min Cut
Example run 1

Select an edge randomly:
Green edge selected.
Contract that edge.

Size of min-cut: 2
Contraction algorithm for min cut

Example run 1

Select an edge randomly:

Green edge selected.

Contract that edge.  (delete self loops)

Size of min-cut: 2
Contraction algorithm for min cut

Example run 1

Select an edge randomly: Purple edge selected.

Contract that edge. (delete self loops)

Size of min-cut: 2
Contraction algorithm for min cut

**Example run 1**

Select an edge randomly:
- Purple edge selected.

Contract that edge.
- (delete self loops)

Why delete self loops?

Size of min-cut: 2
Contraction algorithm for min cut

Example run 1

Select an edge randomly:

Blue edge selected.

Contract that edge.  (delete self loops)

Size of min-cut: 2
Contraction algorithm for min cut

Example run 1

Select an edge randomly: Blue edge selected.

Contract that edge. (delete self loops)

Size of min-cut: 2
When two vertices remain, you have your cut:

\[ S = \{a, b, c, d\} \quad V \setminus S = \{e\} \quad \text{size: 2} \]
Contraction algorithm for min cut

Example run 2

Select an edge randomly:

Green edge selected.

Contract that edge. (delete self loops)
Contraction algorithm for min cut

Example run 2

Select an edge randomly:

Green edge selected.

Contract that edge. (delete self loops)
Contraction algorithm for min cut

Example run 2

Select an edge randomly:
Yellow edge selected.

Contract that edge.  (delete self loops)
Contraction algorithm for min cut

Example run 2

Select an edge randomly:

Yellow edge selected.

Contract that edge. (delete self loops)
Contraction algorithm for min cut

Example run 2

Select an edge randomly:
Red edge selected.

Contract that edge. (delete self loops)
Contraction algorithm for min cut

Example run 2

Select an edge randomly: Red edge selected.

Contract that edge. (delete self loops)
Contraction algorithm for min cut

Example run 2

When two vertices remain, you have your cut:

$$S = \{a\} \quad V \setminus S = \{b, c, d, e\} \quad \text{size: 3}$$
Contraction algorithm for min cut

\[ G = G_0 \longrightarrow G_1 \longrightarrow G_2 \longrightarrow \cdots \longrightarrow G_{n-2} \]

\( n \) vertices \quad \text{contract} \quad \text{contract} \quad \text{contract} \quad \text{contract} \quad 2 \) vertices

\( n - 2 \) iterations
Observation (cut invariance):
For any $i$: A cut in $G_i$ of size $k$ corresponds exactly to a cut in $G$ of size $k$. 
Theorem:
Let $G = (V, E)$ be a connected graph with $n$ vertices. The probability that the contraction algorithm will output a min-cut is $\geq 1/n^2$

Should we be impressed?
- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\sim 2^n$)
- There is a way to boost the probability of success to $1 - \frac{1}{e^n}$ (and still remain in polynomial time)
Proof of Theorem
Let \( k \) be the size of a minimum cut.

Which of the following are true (can select more than one):

1) For \( G = G_0 \), \( k \leq \min_v \deg_G(v) \) \((\forall v, k \leq \deg_G(v))\)

2) For \( G = G_0 \), \( k \geq \min_v \deg_G(v) \)

3) For every \( G_i \), \( k \leq \min_v \deg_{G_i}(v) \) \((\forall v, k \leq \deg_{G_i}(v))\)

4) For every \( G_i \), \( k \geq \min_v \deg_{G_i}(v) \)
For every $G_i$, $k \leq \min_v \deg_{G_i}(v)$
i.e., for every $G_i$ and every $v \in G_i$, $k \leq \deg_{G_i}(v)$

**Why?**

**Short Answer:** A single vertex $v$ forms a cut of size $\deg(v)$.

**Example:**

This cut has size $\deg(a) = 3$

Same cut exists in original graph (cut invariance)

So $k \leq 3$. 
Proof of theorem

Fix some minimum cut

\[ k = |F| \]

\[ n_i = \text{# vertices in } G_i \]

\[ m_i = \text{# edges in } G_i \]

\[ n = n_0, \quad m = m_0 \]

Will show \( \Pr[\text{algorithm outputs } F] \geq 1/n^2 \)

(Note \( \Pr[\text{success}] \geq \Pr[\text{algorithm outputs } F] \))
Proof of theorem

Fix some minimum cut.

\[ k = |F| \]
\[ n_i = \# \text{ vertices in } G_i \]
\[ m_i = \# \text{ edges in } G_i \]
\[ n = n_0, \quad m = m_0 \]

When does the algorithm output \( F \) ?

What if it never picks an edge in \( F \) to contract?
Then it will output \( F \).

What if the algorithm picks an edge in \( F \) to contract?
Then it cannot output \( F \).
Proof of theorem

Fix some minimum cut.

\[ k = |F| \]
\[ n_i = \# \text{ vertices in } G_i \]
\[ m_i = \# \text{ edges in } G_i \]
\[ n = n_0, \quad m = m_0 \]

\[ \Pr[\text{success}] \geq \]

\[ \Pr[\text{algorithm outputs } F] = \]

\[ \Pr[\text{algorithm never contracts an edge in } F] = \]
\[ \Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \]
\[ E_i = \text{“an edge in } F \text{ is contracted in iteration } i\text{”} \]
Proof of theorem

\[ E_i = \text{“an edge in } F \text{ is contracted in iteration } i \text{.”} \]

**Goal:** \[ \Pr[E_1 \cap E_2 \cap \cdots \cap E_{n-2}] \geq 1/n^2. \]

\[
\Pr[E_1 \cap E_2 \cap \cdots \cap E_{n-2}]
= \Pr[E_1] \cdot \Pr[E_2|E_1] \cdot \Pr[E_3|E_1 \cap E_2] \cdots \\
= \Pr[E_{n-2}|E_1 \cap E_2 \cap \cdots \cap E_{n-3}]
\]

\[ \Pr[E_1] = 1 - \Pr[E_1] = 1 - \frac{\# \text{ edges in } F}{\text{total } \# \text{ edges}} = 1 - \frac{k}{m} \]

want to write in terms of \( k \) and \( n \).
Proof of theorem

\[ E_i = "\text{an edge in } F \text{ is contracted in iteration } i."\]

**Goal:** \( \Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2. \)

**Observation:** \( \forall v \in V : k \leq \deg(v) \)

**Recall:**
\[
\sum_{v \in V} \deg(v) = 2m \quad \Rightarrow \quad 2m \geq kn
\]
\[
\Rightarrow m \geq \frac{kn}{2}
\]

\[
\Pr[\overline{E_1}] = 1 - \frac{k}{m} \geq 1 - \frac{k}{kn/2} = \left(1 - \frac{2}{n}\right)
\]
Proof of theorem

\[ E_i = \text{“an edge in } F \text{ is contracted in iteration } i \text{.”} \]

**Goal:** \[ \Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq \frac{1}{n^2}. \]

\[
\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \\
\geq \left(1 - \frac{2}{n}\right) \cdot \Pr[\overline{E_2}\mid \overline{E_1}] \cdot \Pr[\overline{E_3}\mid \overline{E_1} \cap \overline{E_2}] \cdots \\
\Pr[\overline{E_{n-2}}\mid \overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]
\]

\[
\Pr[\overline{E_2}\mid \overline{E_1}] = 1 - \Pr[\overline{E_2}\mid \overline{E_1}] = 1 - \frac{k}{\# \text{ remaining edges}}
\]

want to write in terms of \( k \) and \( n \)
Proof of theorem

\[ E_i = \text{“an edge in } F \text{ is contracted in iteration } i \text{.”} \]

**Goal:** \( \Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2. \)

Let \( G' = (V', E') \) be the graph after iteration 1.

**Observation:** \( \forall v \in V' : k \leq \deg_{G'}(v) \)

\[
\sum_{v \in V'} \deg_{G'}(v) = 2|E'| \implies 2|E'| \geq k(n-1)
\]
\[
\therefore |E'| \geq \frac{k(n-1)}{2}
\]

\[
\Pr[\overline{E_2}|\overline{E_1}] = 1 - \frac{k}{|E'|} \geq 1 - \frac{k}{k(n-1)/2} = \left(1 - \frac{2}{n-1}\right)
\]
Proof of theorem

\[ E_i = \text{“an edge in } F \text{ is contracted in iteration } i.\text{”} \]

**Goal:** \[ \Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq 1/n^2. \]

\[
\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \\
\geq \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{2}{n-1}\right) \cdot \Pr[\overline{E_3} | \overline{E_1} \cap \overline{E_2}] \cdots \\
\Pr[\overline{E_{n-2}} | \overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-3}}]
\]
Proof of theorem

\[ E_i = \text{“an edge in } F \text{ is contracted in iteration } i \text{.”} \]

**Goal:** \( \Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \geq \frac{1}{n^2}. \)

\[
\Pr[\overline{E_1} \cap \overline{E_2} \cap \cdots \cap \overline{E_{n-2}}] \\
\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \cdots \left(1 - \frac{2}{n-(n-4)}\right) \left(1 - \frac{2}{n-(n-3)}\right) \\
= \left(\frac{n}{n}\right) \left(\frac{n-3}{n-1}\right) \left(\frac{n-4}{n}\right) \cdots \left(\frac{2}{4}\right) \left(\frac{1}{3}\right) \\
= \frac{2}{n(n-1)} \geq \frac{1}{n^2}
\]
Theorem:
Let $G = (V, E)$ be a connected graph with $n$ vertices. The probability that the contraction algorithm will output a min-cut is $\geq 1/n^2$.

Should we be impressed?

- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\sim 2^n$)
- There is a way to boost the probability of success to $1 - \frac{1}{e^n}$ (and still remain in polynomial time)
Boosting Phase
Boosting phase

Run the algorithm $t$ times using fresh random bits.

Output the minimum among $F_i$’s.

larger $t$  $\rightarrow$  better success probability

What is the relation between $t$ and success probability?
Let $A_i$ = “in the i’th repetition, we don’t find a min cut.”

$$\Pr[\text{error}] = \Pr[\text{don’t find a min cut}]$$

$$= \Pr[A_1 \cap A_2 \cap \cdots \cap A_t]$$

ind. events

$$= \Pr[A_1] \Pr[A_2] \cdots \Pr[A_t]$$

$$= \Pr[A_1]^t \leq \left(1 - \frac{1}{n^2}\right)^t$$
Boosting phase

World's most useful inequality: \( \forall x \in \mathbb{R} : 1 + x \leq e^x \)
Boosting phase

\[ \Pr[\text{error}] \leq \left( 1 - \frac{1}{n^2} \right)^t \]

**World’s most useful inequality:** \( \forall x \in \mathbb{R} : 1 + x \leq e^x \)

Let \( x = -1/n^2 \)

\[ \Pr[\text{error}] \leq (1 + x)^t \leq (e^x)^t = e^{xt} = e^{-t/n^2} \]

\[ t = n^3 \implies \Pr[\text{error}] \leq e^{-n^3/n^2} = 1/e^n \implies \]

\[ \Pr[\text{success}] \geq 1 - \frac{1}{e^n} \]
Conclusion for min cut

We have a polynomial-time algorithm that solves the min cut problem with probability $1 - 1/e^n$

- Theoretically, not equal to 1
- Practically, equal to 1
Important Note

Boosting is not specific to Min-cut algorithm

We can boost the success probability of Monte Carlo algorithms via repeated trials
Example of a Las Vegas Algorithm: Quicksort

Doesn't gamble with correctness. Gambles with running time.
On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return $S$
Quicksort Algorithm

On input $S = (x_1, x_2, \ldots, x_n)$
- If $n \leq 1$, return $S$
- Pick uniformly at random a “pivot” $x_m$
**Quicksort Algorithm**

<table>
<thead>
<tr>
<th>8</th>
<th>2</th>
<th>7</th>
<th>99</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On input $S = (x_1, x_2, \ldots, x_n)$
- If $n \leq 1$, return $S$
- Pick uniformly at random a “pivot” $x_m$
Quicksort Algorithm

On input $S = (x_1, x_2, \ldots, x_n)$
- If $n \leq 1$, return $S$
- Pick uniformly at random a “pivot” $x_m$
- Compare $x_m$ to all other $x$’s
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
Quicksort Algorithm

On input $S = (x_1, x_2, \ldots, x_n)$
- If $n \leq 1$, return $S$
- Pick uniformly at random a “pivot” $x_m$
- Compare $x_m$ to all other $x$’s
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
Quicksort Algorithm

On input \( S = (x_1, x_2, \ldots, x_n) \)

- If \( n \leq 1 \), return \( S \)
- Pick uniformly at random a “pivot” \( x_m \)
- Compare \( x_m \) to all other \( x \)’s
- Let \( S_1 = \{ x_i : x_i < x_m \} \), \( S_2 = \{ x_i : x_i > x_m \} \)
### Quicksort Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>0</th>
<th>4</th>
<th>8</th>
<th>7</th>
<th>99</th>
<th>5</th>
</tr>
</thead>
</table>

$S_1$ $S_2$

On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return $S$
- Pick uniformly at random a “pivot” $x_m$
- Compare $x_m$ to all other $x$’s
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
- Recursively sort $S_1$ and $S_2$
On input \( S = (x_1, x_2, \ldots, x_n) \)

- If \( n \leq 1 \), return \( S \)
- Pick uniformly at random a “pivot” \( x_m \)
- Compare \( x_m \) to all other \( x \)'s
- Let \( S_1 = \{ x_i : x_i < x_m \} \), \( S_2 = \{ x_i : x_i > x_m \} \)
- Recursively sort \( S_1 \) and \( S_2 \).
Quicksort Algorithm

On input $S = (x_1, x_2, \ldots, x_n)$

- If $n \leq 1$, return $S$
- Pick uniformly at random a “pivot” $x_m$
- Compare $x_m$ to all other $x$’s
- Let $S_1 = \{x_i : x_i < x_m\}$, $S_2 = \{x_i : x_i > x_m\}$
- Recursively sort $S_1$ and $S_2$. 
- Return $[S_1, x_m, S_2]$
Worst case scenario:
Suppose we always end up picking the smallest or largest element as the pivot.

For an input like

\[
\begin{array}{ccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

how many comparisons would we make?

\[
(n - 1) + (n - 2) + \cdots + 2 + 1 = \frac{n(n - 1)}{2} = \Omega(n^2)
\]

Recursive relation for the number of comparisons:

\[
T(n) = T(n - 1) + (n - 1)
\]
Quicksort Algorithm Analysis

Best case scenario:

What is the best choice of pivot?

Total number of comparisons:

\[ T(n) = T(|S_1|) + T(|S_2|) + (n - 1) \]
\[ T(n) = T(k) + T(n - k - 1) + (n - 1) \]

For \( k \approx n/2 \), \( T(n) = O(n \log n) \)
Expected Running Time?

For fun, let’s look at the expected number of comparisons.

Let \( X \) = number of comparisons

What is \( \mathbb{E}[X] \) ?

How can we bound \( \mathbb{E}[X] \) ?

Indicator r.v.’s + Linearity of expectation
Quicksort: Expected number of comparisons

Indicator r.v.’s + Linearity of expectation

Let $X_{ij} = \# \text{ times } x_i \text{ and } x_j \text{ get compared}$

So: $X = \sum_{1 \leq i < j \leq n} X_{ij}$

How many times do $x_i$ and $x_j$ get compared? 0 or 1

$X_{ij} = \begin{cases} 1 & \text{if } x_i \text{ and } x_j \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$
Quicksort: Expected number of comparisons

Indicator r.v.'s + Linearity of expectation

Let $X_{ij} = \# \text{ times } x_i \text{ and } x_j \text{ get compared}.$

So: $X = \sum_{1 \leq i < j \leq n} X_{ij}$

$\implies \mathbf{E}[X] = \sum_{1 \leq i < j \leq n} \mathbf{Pr}[x_i \text{ and } x_j \text{ are compared}]$

Claim: $\mathbf{Pr}[y_i \text{ and } y_j \text{ are compared}] = \frac{2}{j - i + 1}$

Proof: Define $Y^{ij} = \{y_i, y_{i+1}, \ldots, y_j\}$

Consider the algorithm when a pivot $p$ is being chosen.

Case 1: $p \notin Y^{ij} \implies Y^{ij} \subseteq S_1$ or $Y^{ij} \subseteq S_2$

Case 2: $p \in Y^{ij}$ but $p \neq y_i$ and $p \neq y_j$

$\implies y_i \in S_1$ and $y_j \in S_2$ (and $y_i$ and $y_j$ never compared)

Case 3: $p = y_i$ or $p = y_j$ (and $y_i$ and $y_j$ are compared)

$\sum_{i=1}^{n} f(i) \approx \int_{x=1}^{n} f(x) \, dx$

$\sum_{i=1}^{n} \frac{1}{i} \approx \ln(n)$

$\sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_{x=1}^{n} \frac{1}{x} \, dx \leq 1 + \ln(n)$

$E[X] = \sum_{1 \leq i < j \leq n} \frac{2}{j - i + 1} = 2 \sum_{1 \leq i < j \leq n} \frac{1}{j - i + 1} \leq \ln(n)$

$\sum_{1 \leq i < j \leq n} \frac{1}{j - i + 1} \leq \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{2}$

$\leq \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$

So: $E[X] \leq 2n \ln n$

$\implies E[X] \leq 2n \ln n$
Quicksort Algorithm

This is a Las Vegas algorithm:
- always gives the correct answer
- running time can vary depending on our luck

Expected run-time is $=\text{expected number of comparisons}$:

$$\leq 2n \ln n = O(n \log n).$$

In practice, it is basically the fastest sorting algorithm!
Randomness adds an interesting dimension to computation

Randomized algorithms can be faster and more elegant than their deterministic counterparts

There are some interesting problems for which:
- there is a poly-time randomized algorithm,
- we can’t find a poly-time deterministic algorithm

Another (morally) million dollar question:

Is $P = \text{BPP}$?