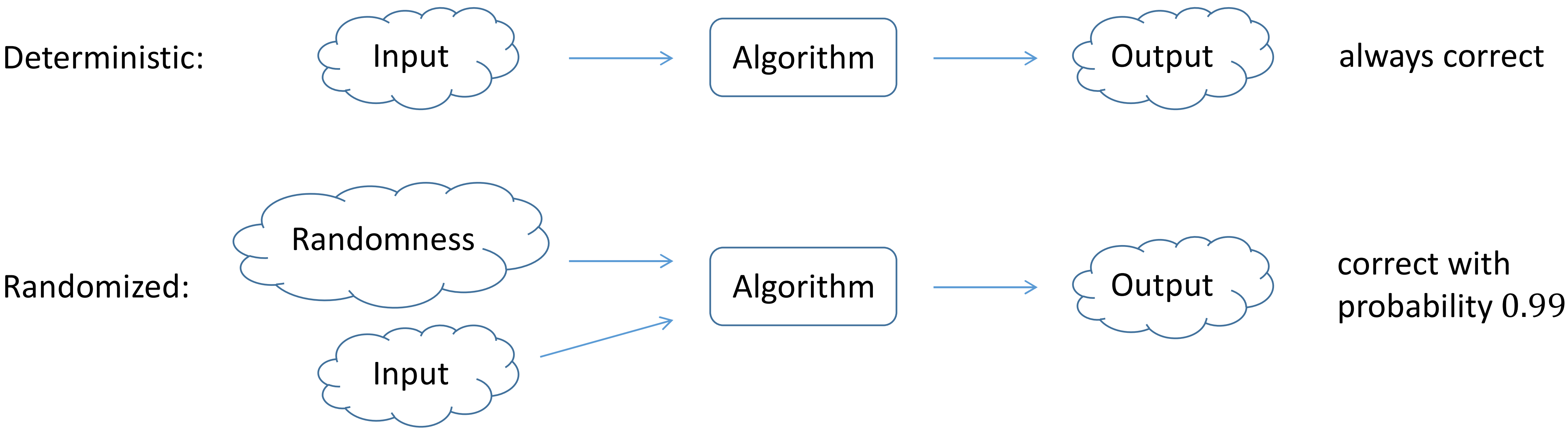
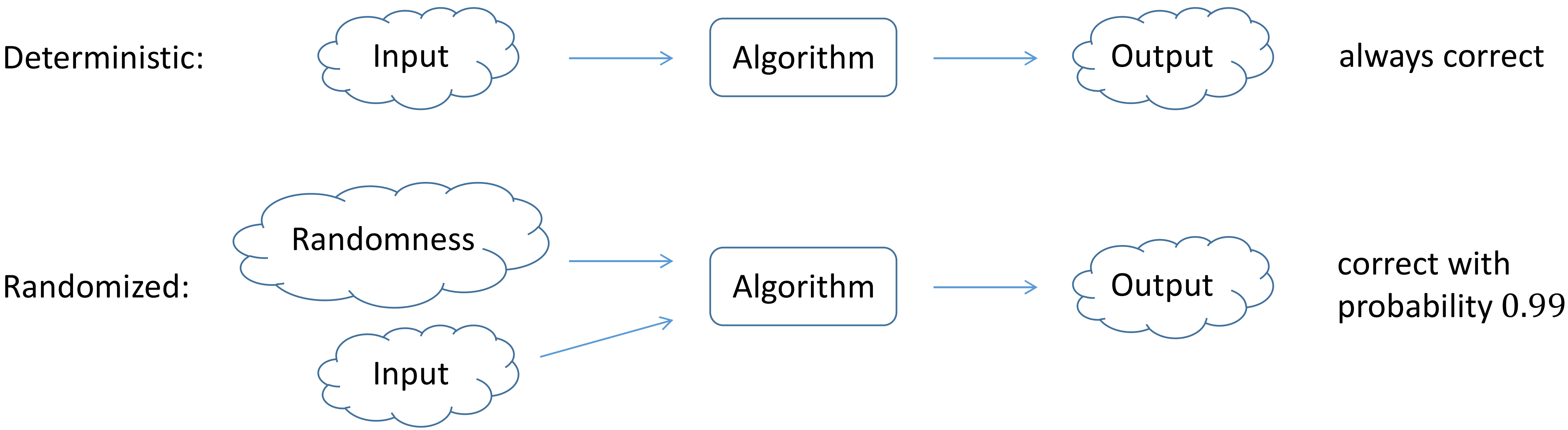
CS3230 – Design and Analysis of Algorithms
(S1 AY2024/25)

Lecture 5: Randomized Algorithms

Randomized algorithms

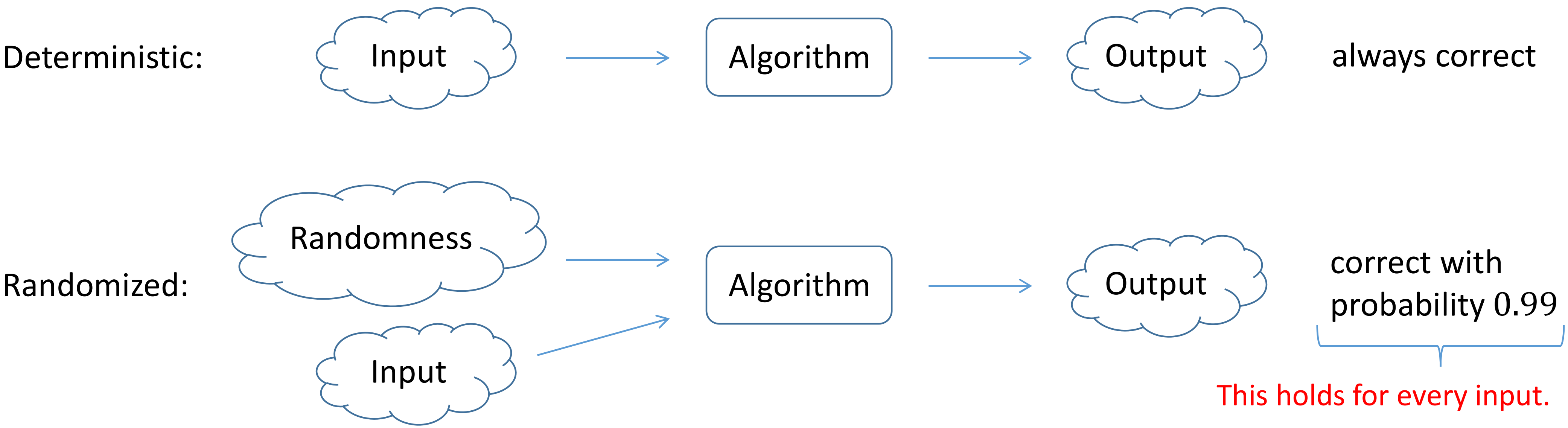


Randomized algorithms



Goal: Utilize randomization to develop algorithms that are more efficient or simpler than their deterministic counterparts, at the cost of allowing a small error probability.

Randomized algorithms



This holds for every input.

- Still do a worst-case analysis over all possible inputs.
- Randomized complexity $\neq$ average-case complexity.

Here algorithm can perform badly for some inputs.

Verification of matrix multiplication

- Given three $n \times n$ matrices A, B, and C, check if $AB = C$.

- **A naïve algorithm:**
  - Calculate AB using a matrix multiplication algorithm.

Verification of matrix multiplication

- Given three $n \times n$ matrices A, B, and C, check if $AB = C$.

- **A naïve algorithm:**
  - Calculate AB using a matrix multiplication algorithm.

- The time complexity of matrix multiplication:
  - Basic algorithm: $O(n^3)$.
  - Strassen's algorithm: $O(n^{2.807\cdots})$.

Question: Can we do better?

Freivalds' algorithm

$\mathbf{Freivalds}(A, B, C)$

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.

For each $i \in [n]$ independently:

- $v_i = 0$ with probability $\frac{1}{2}$.
- $v_i = 1$ with probability $\frac{1}{2}$.

Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.

For each $i \in [n]$ independently:

- $v_i = 0$ with probability $\frac{1}{2}$.
- $v_i = 1$ with probability $\frac{1}{2}$.

This can be done in $O(n^2)$ time via three matrix-vector multiplication.

Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

For each $i \in [n]$ independently:

- $v_i = 0$ with probability $\frac{1}{2}$.
- $v_i = 1$ with probability $\frac{1}{2}$.

This can be done in $O(n^2)$ time via three matrix-vector multiplication.

Analysis of Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

- If $AB = C$, then $ABv = Cv$, so the algorithm always decides $AB = C$ correctly.

- From now on, we focus on the case where $AB \neq C$.

Analysis of Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

- From now on, we focus on the case where $AB \neq C$.
- $C^* = \begin{pmatrix} c^*_{1,1} & \cdots & c^*_{1,n} \\ \vdots & \ddots & \vdots \\ c^*_{n,1} & \cdots & c^*_{n,n} \end{pmatrix} = AB - C$
- $u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = C^* v$

Analysis of Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

- From now on, we focus on the case where $AB \neq C$.
- $C^* = \begin{pmatrix} c^*_{1,1} & \cdots & c^*_{1,n} \\ \vdots & \ddots & \vdots \\ c^*_{n,1} & \cdots & c^*_{n,n} \end{pmatrix} = AB - C$
- $u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = C^* v$

The algorithm outputs an **incorrect** answer $\xleftrightarrow{\text{if and only if}}$ $ABv = Cv$ $\xleftrightarrow{\text{if and only if}}$ $u_k = 0$ for all $k \in [n]$

$\mathbf{Pr}[\text{Freivalds' algorithm is successful}] = \mathbf{Pr}\big[u_k \neq 0 \text{ for some } k \in [n]\big]$

Analysis of Freivalds' algorithm

$\mathbf{Freivalds}(A, B, C)$

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

- From now on, we focus on the case where $AB \neq C$.
- $C^* = \begin{pmatrix} c^*_{1,1} & \cdots & c^*_{1,n} \\ \vdots & \ddots & \vdots \\ c^*_{n,1} & \cdots & c^*_{n,n} \end{pmatrix} = AB - C$
- $u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = C^* v$

The algorithm outputs an **incorrect** answer $\xleftrightarrow{\text{if and only if}}$ $ABv = Cv$ $\xleftrightarrow{\text{if and only if}}$ $u_k = 0$ for all $k \in [n]$

$\mathbf{Pr}[\text{Freivalds' algorithm is successful}] = \mathbf{Pr}\left[u_k \neq 0 \text{ for some } k \in [n]\right]$

Analysis of Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

- From now on, we focus on the case where $AB \neq C$.
- $C^* = \begin{pmatrix} c^*_{1,1} & \cdots & c^*_{1,n} \\ \vdots & \ddots & \vdots \\ c^*_{n,1} & \cdots & c^*_{n,n} \end{pmatrix} = AB - C$
- $u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = C^* v$

- Since $AB \neq C$, there exist (i, j) such that $c^*_{i,j} \neq 0$.
- $u_i = c^*_{i,1} v_1 + c^*_{i,2} v_2 + \cdots + c^*_{i,j} v_j + \cdots + c^*_{i,n} v_n = (\ldots) + c^*_{i,j} v_j$

$$\mathbf{Pr}[\text{Freivalds' algorithm is successful}] = \mathbf{Pr}\big[u_k \neq 0 \text{ for some } k \in [n]\big]$$

Analysis of Freivalds' algorithm

Freivalds(A, B, C)

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

- From now on, we focus on the case where $AB \neq C$.
- $C^* = \begin{pmatrix} c^*_{1,1} & \cdots & c^*_{1,n} \\ \vdots & \ddots & \vdots \\ c^*_{n,1} & \cdots & c^*_{n,n} \end{pmatrix} = AB - C$
- $u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = C^* v$

- Since $AB \neq C$, there exist (i, j) such that $c^*_{i,j} \neq 0$.
- $u_i = c^*_{i,1} v_1 + c^*_{i,2} v_2 + \cdots + c^*_{i,j} v_j + \cdots + c^*_{i,n} v_n = \underbrace{(\dots)}\, + c^*_{i,j} v_j$

- Once we reveal the random numbers $\{v_1, v_2, \dots, v_n\} \setminus \{v_j\}$, this term is **fixed**.
- After fixing this term, there is **exactly one choice** of v_j that makes $u_i = 0$.

$\mathbf{Pr}$[Freivalds' algorithm is successful] $= \mathbf{Pr}\left[u_k \neq 0 \text{ for some } k \in [n]\right]$

Analysis of Freivalds' algorithm

$\mathbf{Freivalds}(A, B, C)$

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

At least one of them makes makes $u_i \neq 0$:

- $v_j = 0$ with probability $\frac{1}{2}$.
- $v_j = 1$ with probability $\frac{1}{2}$.

- Since $AB \neq C$, there exist (i, j) such that $c^*_{i,j} \neq 0$.
- $u_i = c^*_{i,1} v_1 + c^*_{i,2} v_2 + \cdots + c^*_{i,j} v_j + \cdots + c^*_{i,n} v_n = \underbrace{(\ldots)} + c^*_{i,j} v_j$

- Once we reveal the random numbers $\{v_1, v_2, \ldots, v_n\} \setminus \{v_j\}$, this term is **fixed**.
- After fixing this term, there is **exactly one choice** of v_j that makes $u_i = 0$.

$\triangleright$ $\mathbf{Pr}[u_i \neq 0] \geq \frac{1}{2}$

$\mathbf{Pr}[\text{Freivalds' algorithm is successful}] = \mathbf{Pr}\big[u_k \neq 0 \text{ for some } k \in [n]\big]$

Analysis of Freivalds' algorithm

$\mathbf{Freivalds}(A, B, C)$

- Choose $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to be a uniformly random column vector from $\{0,1\}^n$.
- Check if $ABv = Cv$.
- **If** $ABv = Cv$, **then** output $AB = C$.
- **If** $ABv \neq Cv$, **then** output $AB \neq C$.

$\mathbf{Pr}[\text{Freivalds' algorithm is successful}] \geq \frac{1}{2}$

- Since $AB \neq C$, there exist (i, j) such that $c^*_{i,j} \neq 0$.
- $u_i = c^*_{i,1} v_1 + c^*_{i,2} v_2 + \cdots + c^*_{i,j} v_j + \cdots + c^*_{i,n} v_n = \underbrace{(\ldots)} + c^*_{i,j} v_j$

- Once we reveal the random numbers $\{v_1, v_2, \ldots, v_n\} \setminus \{v_j\}$, this term is **fixed**.
- After fixing this term, there is **exactly one choice** of v_j that makes $u_i = 0$.

$\triangleright$ $\mathbf{Pr}[u_i \neq 0] \geq \frac{1}{2}$

Technique: Principle of deferred decision

In the analysis of Freivalds' algorithm, we fix the variables $\{v_1, v_2, \dots, v_n\} \setminus \{v_j\}$ and only consider the randomness in v_j.

- Why are we allowed to do this?

Technique: Principle of deferred decision

In the analysis of Freivalds' algorithm, we fix the variables $\{v_1, v_2, \ldots, v_n\} \setminus \{v_j\}$ and only consider the randomness in v_j.

- Why are we allowed to do this?

Principle of deferred decision:
- If we can show that $\mathbf{Pr}[\mathcal{E} \mid X = x] \geq p$ for every x, then $\mathbf{Pr}[\mathcal{E}] \geq p$.

$$\mathbf{Pr}[\mathcal{E}] = \sum_x \mathbf{Pr}[\mathcal{E} \mid X = x] \cdot \mathbf{Pr}[X = x] \geq p \cdot \sum_x \mathbf{Pr}[X = x] = p.$$

Technique: Success probability amplification

We only show that Freivalds' algorithm is **incorrect** with a probability of **at most** $\frac{1}{2}$.

Case $(AB = C)$:
- The algorithm answers $AB = C$ correctly.

Case $(AB \neq C)$:
- The algorithm answers $AB \neq C$ with a probability of at least $1/2$.
- The algorithm answers $AB = C$ with a probability of at most $1/2$.

successful with a probability of **at least** $\frac{1}{2}$.

Technique: Success probability amplification

We only show that Freivalds' algorithm is **incorrect** with a probability of **at most** $\mathbf{\frac{1}{2}}$.

Case $(AB = C)$:

- The algorithm answers $AB = C$ correctly.

Case $(AB \neq C)$:

- The algorithm answers $AB \neq C$ with a probability of at least $1/2$.
- The algorithm answers $AB = C$ with a probability of at most $1/2$.

Claim: The error probability can be reduced to **at most** $\boldsymbol{f}$ by repeating the algorithm for $t = \left\lceil \log \frac{1}{f} \right\rceil$ times.

- If all t outputs are $AB = C$, return $AB = C$.
- Otherwise, return $AB \neq C$.

Technique: Success probability amplification

We only show that Freivalds' algorithm is **incorrect** with a probability of **at most** $\mathbf{\frac{1}{2}}$.

Case $(AB = C)$:
- The algorithm answers $AB = C$ correctly.

Case $(AB \neq C)$:
- The algorithm answers $AB \neq C$ with a probability of at least $1/2$.
- The algorithm answers $AB = C$ with a probability of at most $1/2$.

- If $AB = C$, Freivalds' algorithm always answers $AB = C$ correctly.
- If $AB \neq C$, the probability that Freivalds' algorithm answers $AB = C$ for all $t = \left\lceil \log \frac{1}{f} \right\rceil$ iterations is at most $\frac{1}{2^t} \leq f$.

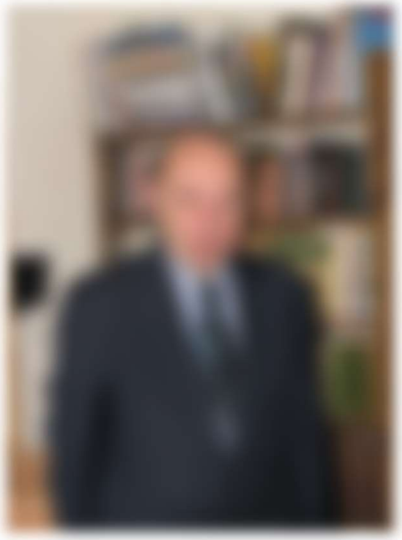
Claim: The error probability can be reduced to **at most** $\boldsymbol{f}$ by repeating the algorithm for $t = \left\lceil \log \frac{1}{f} \right\rceil$ times.

- If all t outputs are $AB = C$, return $AB = C$.
- Otherwise, return $AB \neq C$.

Question 1 @ VisuAlgo online quiz

Who is the **Master of Algorithms** pictured below?

- László Babai
- Rūsiņš Freivalds
- Leonid Levin
- Volker Strassen



Coupon collector's problem

- There are n different types of coupons.
- Once you obtain all n types of coupons, you may receive a prize.
- Each box of cereals contains a random coupon.
- How many boxes must you buy to collect all n types of coupons?

Coupon collector's problem

- There are n different types of coupons.
- Once you obtain all n types of coupons, you may receive a prize.
- Each box of cereals contains a random coupon.
- How many boxes must you buy to collect all n types of coupons?

m

Balls and bins:

- Throw m balls into n bins randomly and independently.
- What is the probability that every bin contains at least one ball?

Coupon collector's problem

- There are n different types of coupons.
- Once you obtain all n types of coupons, you may receive a prize.
- Each box of cereals contains a random coupon.
- How many boxes must you buy to collect all n types of coupons?

$\updownarrow$ m

Balls and bins:

- Throw m balls into n bins randomly and independently.
- What is the probability that every bin contains at least one ball?

- m balls $\leftrightarrow$ m cereal boxes.
- n bins $\leftrightarrow$ n coupons.
- Every bin contains at least one ball $\leftrightarrow$ All n types of coupons have been collected.

Balls and bins

- Throw m balls into n bins randomly and independently.
- What is the probability that every bin contains at least one ball?

Balls and bins

- Throw m balls into n bins randomly and independently.
- What is the probability that every bin contains at least one ball?

- Consider one bin.
- The probability that the bin contains zero balls is $\left(1-\frac{1}{n}\right)^m \leq e^{-\frac{m}{n}}$.

Useful inequality: $1+x \leq e^x$

Balls and bins

- Throw m balls into n bins randomly and independently.
- What is the probability that every bin contains at least one ball?

- Consider one bin.
- The probability that the bin contains zero balls is $\left(1-\frac{1}{n}\right)^m \leq e^{-\frac{m}{n}}$.

Union bound: The probability that **at least one** bin contains zero balls is **at most** $n\left(1-\frac{1}{n}\right)^m \leq ne^{-\frac{m}{n}}$.

Balls and bins

- Throw m balls into n bins randomly and independently.
- What is the probability that every bin contains at least one ball?

- Consider one bin.
- The probability that the bin contains zero balls is $\left(1-\frac{1}{n}\right)^m \le e^{-\frac{m}{n}}$.

$\bigtriangledown$

Union bound: The probability that **at least one** bin contains zero balls is **at most** $n\left(1-\frac{1}{n}\right)^m \le ne^{-\frac{m}{n}}$.

The probability is at most $1/n$ if $m \ge 2n\lceil \ln n \rceil$.

Coupon collector's problem

- There are n different types of coupons.
- Once you obtain all n types of coupons, you may receive a prize.
- Each box of cereals contains a random coupon.
- How many boxes must you buy to collect all n types of coupons?

Answer: Buying $m = 2n\lceil \ln n \rceil \in \Theta(n \log n)$ boxes guarantees a success probability of at least $1 - \frac{1}{n}$.

- m balls $\leftrightarrow$ m cereal boxes.
- n bins $\leftrightarrow$ n coupons.
- Every bin contains at least one ball $\leftrightarrow$ All n types of coupons have been collected.

Technique: Union bound

- You want to upper bound the probability that a bad event $\mathcal{E}$ occurs.

- You know that $\mathcal{E} = \mathcal{E}_1 \vee \mathcal{E}_2 \vee \cdots \vee \mathcal{E}_n$.

- **Union bound**:
  - $\Pr[\mathcal{E}] = \Pr[\mathcal{E}_1 \vee \mathcal{E}_2 \vee \cdots \vee \mathcal{E}_n] \leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \cdots + \Pr[\mathcal{E}_n]$.

- To make sure that $\Pr[\mathcal{E}] \leq f$, it suffices that $\Pr[\mathcal{E}_i] \leq \frac{f}{n}$ for each $i \in [n]$.

Expected value

- **Expected value:**
  - $\mathbb{E}[X] = \sum_x x \cdot \Pr[X = x]$, where the sum ranges over all possible outcomes x of the random variable X.

Technique: Markov inequality

- **Markov inequality:**
  - If X is a non-negative random variable and $a > 0$, then
$$\Pr\left[X \geq a \cdot \mathbb{E}[X]\right] \leq \frac{1}{a}.$$

Technique: Markov inequality

- **Markov inequality:**
  - If X is a non-negative random variable and $a > 0$, then
$$\Pr\big[X \geq a \cdot \mathbb{E}[X]\big] \leq \frac{1}{a}.$$

- **Proof:**

$$\begin{aligned}
\mathbb{E}[X] = \sum_{x} x \cdot \Pr[X = x] \; &\geq \sum_{x \geq a \cdot \mathbb{E}[X]} x \cdot \Pr[X = x] \\
&\geq \sum_{x \geq a \cdot \mathbb{E}[X]} a \cdot \mathbb{E}[X] \cdot \Pr[X = x] \\
&= a \cdot \mathbb{E}[X] \cdot \sum_{x \geq a \cdot \mathbb{E}[X]} \Pr[X = x] \\
&= a \cdot \mathbb{E}[X] \cdot \Pr\big[X \geq a \cdot \mathbb{E}[X]\big] \qquad \triangleright \qquad \Pr\big[X \geq a \cdot \mathbb{E}[X]\big] \leq \frac{1}{a}
\end{aligned}$$

Technique: Markov inequality

- **Markov inequality:**
  - If X is a non-negative random variable and $a > 0$, then
$$\Pr\left[X \geq a \cdot \mathbb{E}[X]\right] \leq \frac{1}{a}.$$

- **Application:**

$\boxed{\text{The expected runtime of } \mathcal{A} \text{ is at most } t}$

$\triangledown$

$\boxed{\text{The runtime of } \mathcal{A} \text{ is at most } 100 \cdot t \text{ with probability at least } 0.99}$

$$\Pr[\text{runtime} \geq 100 \cdot t] \leq \Pr[\text{runtime} \geq 100 \cdot \text{expected runtime}] \leq \frac{1}{100}$$

Technique: Markov inequality

- **Markov inequality:**
  - If X is a non-negative random variable and $a > 0$, then

$$\Pr\left[X \geq a \cdot \mathbb{E}[X]\right] \leq \frac{1}{a}.$$

- **Application:**

The **expected** time complexity of $\mathcal{A}$ is $O(n \log n)$

$\bigtriangledown$

The time complexity of $\mathcal{A}$ is $O(n \log n)$ **with probability at least $\mathbf{0.99}$**

Technique: Linearity of expectation

- **Linearity of expectation:**
  - If $X = A + B$, then $\mathbb{E}[X] = \mathbb{E}[A] + \mathbb{E}[B]$.
  - More generally, if $X = \sum_{i=1}^{n} X_i$, then $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i]$.

Technique: Linearity of expectation

- **Linearity of expectation:**
  - If $X = A + B$, then $\mathbb{E}[X] = \mathbb{E}[A] + \mathbb{E}[B]$.
  - More generally, if $X = \sum_{i=1}^{n} X_i$, then $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i]$.
- **Proof:**

$$
\begin{aligned}
\mathbb{E}[X] = \sum_x x \cdot \Pr[X = x] &= \sum_x x \cdot \Pr[A + B = x] \\
&= \sum_x \sum_b x \cdot \Pr[(A = x - b) \wedge (B = b)] \qquad \boxed{\Pr[A + B = x] = \sum_b \Pr[(A = x - b) \wedge (B = b)]} \\
&= \sum_a \sum_b (a + b) \cdot \Pr[(A = a) \wedge (B = b)] \qquad \boxed{a = x - b} \\
&= \sum_a \sum_b a \cdot \Pr[(A = a) \wedge (B = b)] + \sum_b \sum_a b \cdot \Pr[(A = a) \wedge (B = b)] \\
&= \sum_a a \cdot \sum_b \Pr[(A = a) \wedge (B = b)] + \sum_b b \cdot \sum_a \Pr[(A = a) \wedge (B = b)] \\
&= \sum_a a \cdot \Pr[A = a] + \sum_b b \cdot \Pr[B = b] \\
&= \mathbb{E}[A] + \mathbb{E}[B]
\end{aligned}
$$

Technique: Indicator random variables

- Let $\mathcal{E}$ be an event.
- The indicator random variable $\mathbf{1}_{\mathcal{E}}$ for $\mathcal{E}$ is defined as

$$\mathbf{1}_{\mathcal{E}} = \begin{cases} 1, & \text{if } \mathcal{E} \text{ occurs,} \\ 0, & \text{otherwise.} \end{cases}$$

- **Observation:** $\mathbb{E}[\mathbf{1}_{\mathcal{E}}] = \Pr[\mathcal{E}]$.

Hashing

- **Hash table:**
  - A is an array of length n.
- **Hash function:**
  - h is a mapping from some universe U to the indices of the array $\{1,2,\ldots,n\}$.

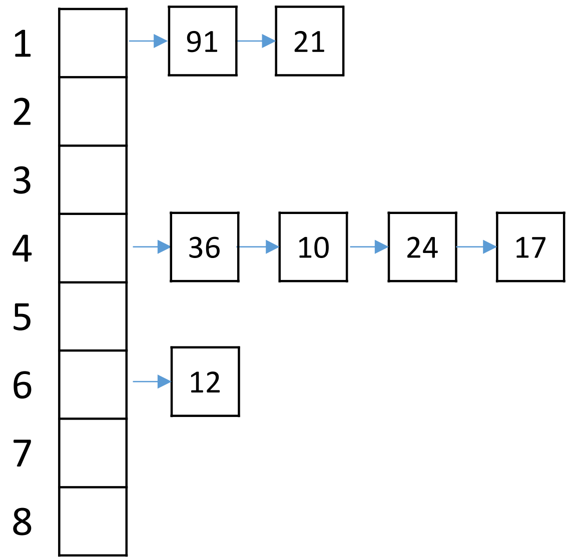
- Insert(v): If v is not in $A[h(v)]$, store v in $A[h(v)]$.
- Search(v): Check if v is in $A[h(v)]$.
- Delete(v): If v is in $A[h(v)]$, remove v from $A[h(v)]$.

Chain hashing

- A **linked list** is created if a position contains more than one element.
- The cost of an operation is linear in the size of the linked list.

- **Hash table:**
  - A is an array of length n.
- **Hash function:**
  - h is a mapping from some universe U to the indices of the array $\{1,2,\ldots,n\}$.

- Insert(v): If v is not in $A[h(v)]$, store v in $A[h(v)]$.
- Search(v): Check if v is in $A[h(v)]$.
- Delete(v): If v is in $A[h(v)]$, remove v from $A[h(v)]$.



Balls and bins

- Throw m balls into n bins randomly and independently.

with a **uniformly random** hash function h

	Balls	**Bins**
Hashing	Elements to be stored	Hash table
Coupon collector	Cereal boxes	Coupons

Balls and bins

- Throw m balls into n bins randomly and independently.

	Balls	**Bins**
Hashing	Elements to be stored	Hash table
Coupon collector	Cereal boxes	Coupons

with a **uniformly random** hash function h → Hashing

How to analyze this?

▽

Number of balls in a bin ⟷ Size of the linked list in an array slot ⟷ Cost of an operation

Question 2 @ VisuAlgo online quiz

- Consider one bin.
- What is the **expected** number of balls in the bin?
  - $\frac{m}{n}$
  - $1 + \frac{m}{n}$
  - $1 + \frac{m}{n} - \frac{1}{n}$
  - $\max\left\{1, \frac{m}{n}\right\}$

Balls and bins:

- Throw m balls into n bins randomly and independently.

Question 3 @ VisuAlgo online quiz

- Consider one ball.
- What is the **expected** number of balls in the bin that contains the selected ball?
  - $\frac{m}{n}$
  - $1 + \frac{m}{n}$
  - $1 + \frac{m}{n} - \frac{1}{n}$
  - $\max\left\{1, \frac{m}{n}\right\}$

Balls and bins:

- Throw m balls into n bins randomly and independently.

Quick sort

- **Input:** an array $A[1..n]$ of n numbers. ← Assume that all numbers are distinct.

- **Partition:**
  - Select a number in $A[1..n]$ as the **pivot**.
  - Rearrange the array to satisfy the condition: $A = [\overbrace{\dots\dots\dots\dots\dots\dots}^{A_S}\ \text{pivot}\ \overbrace{\dots\dots\dots\dots\dots\dots}^{A_L}]$

$\forall x \in A_S, x \leq \text{pivot}$ $\qquad$ $\forall x \in A_L, x \geq \text{pivot}$

- **Recursion:**
  - Recursively sort A_S and A_L.

Randomized quick sort

- **Input:** an array $A[1..n]$ of n numbers.

Assume that all numbers are distinct.

- **Partition:**

  uniformly at random

  - Select a number in $A[1..n]$ as the **pivot**.
  - Rearrange the array to satisfy the condition:

$$A = [\underset{\forall x \in A_S,\, x \leq \text{pivot}}{\overbrace{\ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots}^{A_S}}\ \text{pivot}\ \underset{\forall x \in A_L,\, x \geq \text{pivot}}{\overbrace{\ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots}^{A_L}}\ ]$$

- **Recursion:**
  - Recursively sort A_S and A_L.

Randomized quick sort

- **Input:** an array $A[1..n]$ of n numbers. ← Assume that all numbers are distinct.

uniformly at random

- **Partition:**
  - Select a number in $A[1..n]$ as the **pivot**.
  - Rearrange the array to satisfy the condition:

$$A = [\underbrace{\ldots\ldots\ldots\ldots\ldots\ldots}_{\forall x \in A_S,\, x \le \text{pivot}}^{A_S}\ \text{pivot}\ \underbrace{\ldots\ldots\ldots\ldots\ldots\ldots}_{\forall x \in A_L,\, x \ge \text{pivot}}^{A_L}]$$

This step requires comparing the **pivot** with all other numbers.

- **Recursion:**
  - Recursively sort A_S and A_L.

Observation: running time of quick sort $\in \Theta$(number of comparisons)

Analysis of randomized quick sort

- $(a_1, a_2, \ldots, a_n)$ = the numbers of A in the sorted order.
- $X_{i,j}$ = the number of comparisons made between a_i and a_j.

Linearity of expectation

$$\mathbb{E}[\text{number of comparisons}] = \sum_{1 \le i < j \le n} \boxed{\mathbb{E}[X_{i,j}]}$$

Just need to know how to calculate this.

Analysis of randomized quick sort

- **Observation:**
  - The number $X_{i,j}$ of comparisons made between a_i and a_j is either 0 or 1.
  - We write $\mathcal{E}_{i,j}$ to denote the event $X_{i,j} = 1$.
    - $\Pr[\mathcal{E}_{i,j}] = \mathbb{E}[X_{i,j}]$ ⟵ $X_{i,j}$ is the indicator random variable for $\mathcal{E}_{i,j}$.

Analysis of randomized quick sort

- **Observation:**
  - The number $X_{i,j}$ of comparisons made between a_i and a_j is either 0 or 1.
  - We write $\mathcal{E}_{i,j}$ to denote the event $X_{i,j} = 1$.
    - $\boxed{\Pr[\mathcal{E}_{i,j}] = \mathbb{E}[X_{i,j}]}$ ⟵ $X_{i,j}$ is the indicator random variable for $\mathcal{E}_{i,j}$.

- **Claim:** For any $1 \le i < j \le n$, $\Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

Before a number in $(a_i, \dots, a_j)$ is selected as a pivot, the numbers in $(a_i, \dots, a_j)$ must belong to the same array.

- **Claim:** For any $1 \leq i < j \leq n$, $\Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

Before a number in $(a_i, \dots, a_j)$ is selected as a pivot, the numbers in $(a_i, \dots, a_j)$ must belong to the same array.

The first number chosen as a pivot in $(a_i, \dots, a_j)$:

- a_i or a_j
- not a_i or a_j

- **Claim:** For any $1 \le i < j \le n$, $\Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

Before a number in $(a_i, \dots, a_j)$ is selected as a pivot, the numbers in $(a_i, \dots, a_j)$ must belong to the same array.

The first number chosen as a pivot in $(a_i, \dots, a_j)$:

- a_i or a_j → The algorithm will compare the pivot with all other numbers in the current array. → $\mathcal{E}_{i,j}$ occurs.
- not a_i or a_j

- **Claim:** For any $1 \le i < j \le n$, $\Pr\left[\mathcal{E}_{i,j}\right] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

Before a number in $(a_i, \dots, a_j)$ is selected as a pivot, the numbers in $(a_i, \dots, a_j)$ must belong to the same array.

The first number chosen as a pivot in $(a_i, \dots, a_j)$:

- a_i or a_j: The algorithm will compare the pivot with all other numbers in the current array. → $\mathcal{E}_{i,j}$ occurs.
- not a_i or a_j: a_i and a_j will belong to different arrays in the recursive calls. → $\mathcal{E}_{i,j}$ does not occur.

- **Claim:** For any $1 \le i < j \le n$, $\Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

Before a number in $(a_i, \ldots, a_j)$ is selected as a pivot, the numbers in $(a_i, \ldots, a_j)$ must belong to the same array.

The first number chosen as a pivot in $(a_i, \ldots, a_j)$:

- a_i or a_j: The algorithm will compare the pivot with all other numbers in the current array. → $\mathcal{E}_{i,j}$ occurs.
- not a_i or a_j: a_i and a_j will belong to different arrays in the recursive calls. → $\mathcal{E}_{i,j}$ does not occur.

A comparison is made between a_i and a_j. ⟷ if and only if ⟷ The first number chosen as a pivot in $(a_i, \ldots, a_j)$ is a_i or a_j.

- **Claim:** For any $1 \leq i < j \leq n$, $\Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

Before a number in $(a_i, \ldots, a_j)$ is selected as a pivot, the numbers in $(a_i, \ldots, a_j)$ must belong to the same array.

Each number in $(a_i, \ldots, a_j)$ is **equally likely** to be the first one chosen as a pivot in $(a_i, \ldots, a_j)$.

A comparison is made between a_i and a_j.

if and only if

The first number chosen as a pivot in $(a_i, \ldots, a_j)$ is a_i or a_j.

- **Claim:** For any $1 \leq i < j \leq n$, $\Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$.

A comparison is made between a_i and a_j.

Analysis of randomized quick sort

$$\boxed{\mathbb{E}[X_{i,j}] = \Pr[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}}$$

$$\mathbb{E}[\text{number of comparisons}] = \sum_{1\le i<j\le n} \mathbb{E}[X_{i,j}] = \sum_{1\le i<j\le n} \frac{2}{j-i+1}$$

$$= 2\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j-i+1}$$

$$= 2\sum_{i=1}^{n} \underbrace{\left(\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n-i+1}\right)}_{O(\log n)} \in O(n\log n)$$

Analysis of randomized quick sort

Theorem: The expected running time of randomized quick sort is $O(n \log n)$.

$\Big\downarrow$ Markov inequality

Randomized quick sort finishes in $O(n \log n)$ time with probability at least 0.99.

Two types of randomized algorithms

- Las Vegas algorithms:
  - The output is **always correct**.
  - The time complexity guarantee is only **in expectation**.

Randomized quick sort

- Monte Carlo algorithms:
  - The output is correct only **with some probability**.
  - The time complexity guarantee holds **with probability 1**.

Freivalds' algorithm

Which one is stronger?

Two types of randomized algorithms

- Las Vegas algorithms:
  - The output is **always correct**.
  - The time complexity guarantee is only **in expectation**.

Randomized quick sort

- Monte Carlo algorithms:
  - The output is correct only **with some probability**.
  - The time complexity guarantee holds **with probability** $\mathbf{1}$.

Freivalds' algorithm

Which one is stronger?

We can always turn a Las Vegas algorithm into a Monte Carlo algorithm via Markov inequality.

Discussions

Average-case running time of a deterministic version of an algorithm

They can be very different (in general).

Expected running time of a randomized version of an algorithm

Discussions

Average-case running time of a deterministic version of an algorithm

They can be very different (in general).

Expected running time of a randomized version of an algorithm

Can we apply the analysis of randomized quick sort to do the average-case analysis of deterministic quick sort, and vice versa?

Average-case **number of comparisons** for deterministic quick sort

Are they the same (not just asymptotically)?

Expected **number of comparisons** for randomized quick sort

Acknowledgement

- The slides are modified from previous editions of this course and similar course elsewhere.
- **List of credits:**
  - Surender Baswana
  - Arnab Bhattacharya
  - Diptarka Chakraborty
  - Yi-Jun Chang
  - Erik Demaine
  - Steven Halim
  - Sanjay Jain
  - Wee Sun Lee
  - Charles Leiserson
  - Hon Wai Leong
  - Wing-Kin Sung