Randomized Algorithms

Fahad Panolan

Indian Institute of Technology Hyderabad, India

3-Sep-2022

Slides from Prof. Chandra Chekuri (modified as needed)

Summary of Previous lectures

Streaming Model

Classical Algorithms: Random Access Model (RAM)

Streaming Model

Classical Algorithms: Random Access Model (RAM)

Streaming Model

- The input consists of m objects/items/tokens $e_1, e_2, \ldots, e_m$ that are seen one by one by the algorithm.
- The algorithm has "limited" memory say for B tokens where $B < m$ (often $B << m$) and hence cannot store all the input
- Want to compute interesting functions over input

Streaming Model

Classical Algorithms: Random Access Model (RAM)

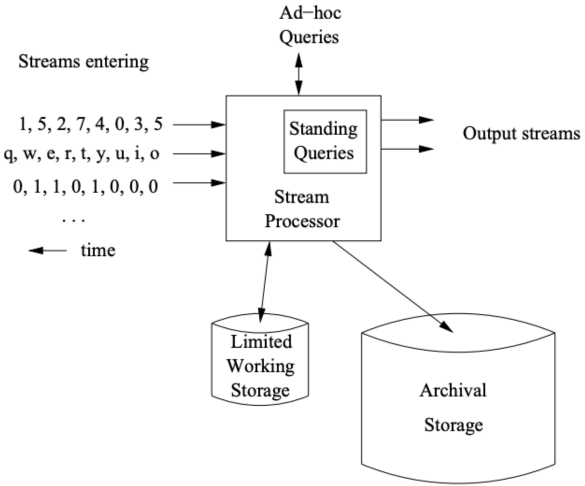
Streaming Model

- The input consists of m objects/items/tokens $e_1, e_2, \ldots, e_m$ that are seen one by one by the algorithm.
- The algorithm has "limited" memory say for B tokens where $B < m$ (often $B << m$) and hence cannot store all the input
- Want to compute interesting functions over input

Some examples:

- Each token is a number from $[n]$
- High-speed network switch: tokens are packets with source, destination IP addresses and message contents.
- Each token is an edge in graph (graph streams)

Streaming model



["Mining of Massive Data Sets" by Leskovec, Rajaraman, Ullman]

Discrete Probability Theory

- Discrete probability space: $(\Omega, \Pr)$ where
  - Ω is a countable set, and
  - $\sum_{\omega \in \Omega} \Pr[w] = 1$.

Discrete Probability Theory

- Discrete probability space: $(\Omega, \Pr)$ where
  - Ω is a countable set, and
  - $\sum_{\omega \in \Omega} \Pr[w] = 1$.
- Elements in Ω are elementary events.

Discrete Probability Theory

- Discrete probability space: $(\Omega, \Pr)$ where
  - Ω is a countable set, and
  - $\sum_{\omega \in \Omega} \Pr[w] = 1$.
- Elements in Ω are <u>elementary events</u>.
- An <u>event</u> is a subset A of Ω and $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$

Discrete Probability Theory

- Discrete probability space: $(\Omega, \Pr)$ where
  - Ω is a countable set, and
  - $\sum_{\omega \in \Omega} \Pr[w] = 1$.
- Elements in Ω are <u>elementary events</u>.
- An <u>event</u> is a subset A of Ω and $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$
- A and B are <u>independent</u> if $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$.

Discrete Probability Theory

- Discrete probability space: $(\Omega, \Pr)$ where
  - Ω is a countable set, and
  - $\sum_{\omega \in \Omega} \Pr[w] = 1$.
- Elements in Ω are <u>elementary events</u>.
- An <u>event</u> is a subset A of Ω and $\Pr[A] = \sum_{\omega \in A} \Pr[\omega]$
- A and B are <u>independent</u> if $\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$.
- Random variables $X : \Omega \mapsto \mathbb{R}$.
- Independent random variables, and expectation and variance of a random variables.
- Probabilistic inequalities: Markov, Chebshev, and Chernoff bounds

Discrete Probability Theory

- Union bound : $\Pr[A_1 \cup A_2 \ldots \cup A_\ell] \leq \sum_{i=1}^{\ell} \Pr[A_i]$.

Discrete Probability Theory

- Union bound : $\Pr[A_1 \cup A_2 \ldots \cup A_\ell] \leq \sum_{i=1}^{\ell} \Pr[A_i]$.
- If $A_1, A_2 \ldots \cup A_\ell$ are mutually disjoint events, then

$$\Pr[A_1 \cup A_2 \ldots \cup A_\ell] = \sum_{i=1}^{\ell} \Pr[A_i].$$

Discrete Probability Theory

- Union bound : $\Pr[A_1 \cup A_2 \ldots \cup A_\ell] \leq \sum_{i=1}^{\ell} \Pr[A_i]$.
- If $A_1, A_2 \ldots \cup A_\ell$ are mutually disjoint events, then

$$\Pr[A_1 \cup A_2 \ldots \cup A_\ell] = \sum_{i=1}^{\ell} \Pr[A_i].$$

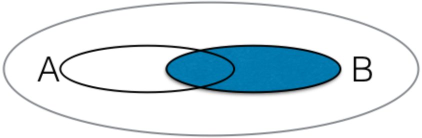
- Law of total probability : Let $E_1, \ldots, E_r \subseteq \Omega$ are mutually disjoint events and $\Omega = \bigcup_i E_i$. Then for any event A,

$$\Pr[A] = \sum_{i} \Pr[A \cap E_i].$$

Conditional Probability

The conditional probability that event A occurs given that event B happened is

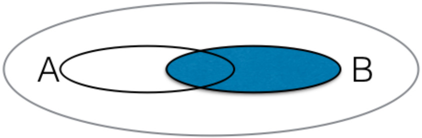
$$\Pr[A|B] = \frac{\Pr[A \cap B]}{\Pr[B]}.$$



Conditional Probability

The conditional probability that event A occurs given that event B happened is

$$\Pr[A|B] = \frac{\Pr[A \cap B]}{\Pr[B]}.$$

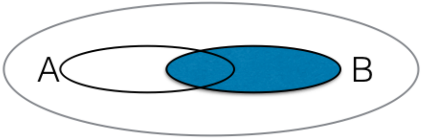


B defines our restricted sample space, we normalize the probabilities by dividing by $\Pr[B]$, so that the sum of the probabilities in B is 1.

Conditional Probability

The conditional probability that event A occurs given that event B happened is

$$\Pr[A|B] = \frac{\Pr[A \cap B]}{\Pr[B]}.$$

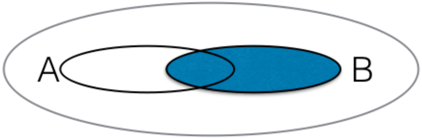


B defines our restricted sample space, we normalize the probabilities by dividing by $\Pr[B]$, so that the sum of the probabilities in B is 1.

Conditional Probability

The conditional probability that event A occurs given that event B happened is

$$\Pr[A|B] = \frac{\Pr[A \cap B]}{\Pr[B]}.$$



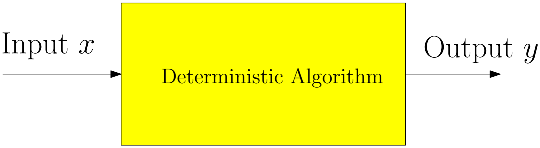
B defines our restricted sample space, we normalize the probabilities by dividing by $\Pr[B]$, so that the sum of the probabilities in B is 1.

Law of total probability : Let $E_1, \ldots, E_r \subseteq \Omega$ are mutually disjoint events and $\Omega = \bigcup_i E_i$. Then for any event A,

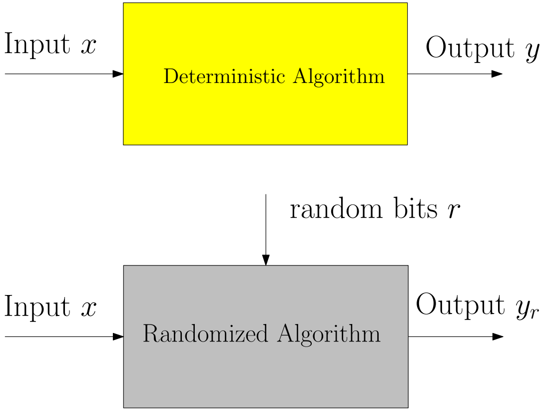
$$\Pr[A] = \sum_i \Pr[A \cap E_i] = \Pr[A|E_i] \cdot \Pr[E_i]$$

Randomized Algorithms

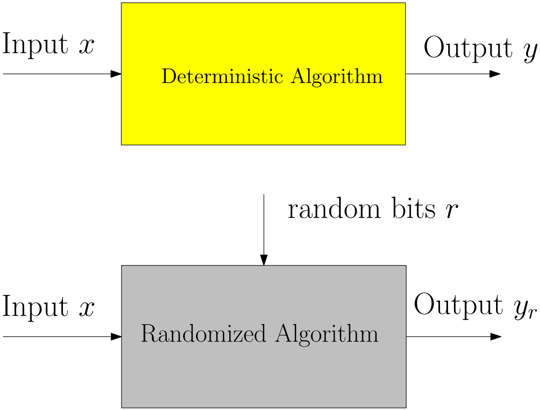
Randomized Algorithms



Randomized Algorithms

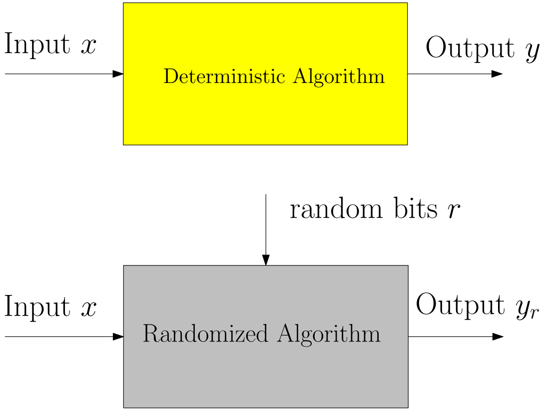


Randomized Algorithms



The behaviour of the algorithm is “not always good”.

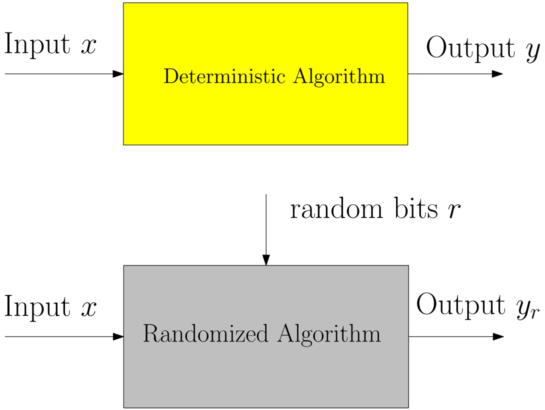
Randomized Algorithms



The behaviour of the algorithm is "not always good".

- Output may not be correct always (Monte Carlo)

Randomized Algorithms



The behaviour of the algorithm is "not always good".

- Output may not be correct always (Monte Carlo)
- Running time may not be fast always (Las Vegas)

Randomized **Quick Sort**

Quick Sort

- Pick a pivot element from array
- Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and equal to pivot.
- Recursively sort the subarrays, and concatenate them.

Randomized **Quick Sort**

- Pick a pivot element **uniformly at random** from the array
- Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and equal to the pivot.
- Recursively sort the subarrays, and concatenate them.

Randomized **Quick Sort**

Recall: Quick Sort can take $\Omega(n^2)$ time to sort array of size n.

Randomized **Quick Sort**

Recall: Quick Sort can take $\Omega(n^2)$ time to sort array of size n.

Theorem

Randomized **Quick Sort** *sorts a given array of length* n *in* $O(n \log n)$ *<u>expected</u> time.*

Randomized **Quick Sort**

Recall: Quick Sort can take $\Omega(n^2)$ time to sort array of size n.

Theorem

Randomized **Quick Sort** *sorts a given array of length* n *in* $O(n \log n)$ *expected time.*

Note: On every input randomized **Quick Sort** takes $O(n \log n)$ time in expectation. On every input it may take $\Omega(n^2)$ time with some small probability.

Analysis

Let $Q(A)$ be number of comparisons done on input array A:

- For $1 \leq i < j < n$ let R_{ij} be the event that rank i element is compared with rank j element.
- X_{ij} is the indicator random variable for R_{ij}. That is, $X_{ij} = 1$ if rank i is compared with rank j element, otherwise 0.

Analysis

Let $Q(A)$ be number of comparisons done on input array A:

- For $1 \leq i < j < n$ let R_{ij} be the event that rank i element is compared with rank j element.

- X_{ij} is the indicator random variable for R_{ij}. That is, $X_{ij} = 1$ if rank i is compared with rank j element, otherwise 0.

$$Q(A) = \sum_{1 \leq i < j \leq n} X_{ij}$$

and hence by linearity of expectation,

$$E[Q(A)] = \sum_{1 \leq i < j \leq n} E[X_{ij}] = \sum_{1 \leq i < j \leq n} \Pr[R_{ij}].$$

Analysis

$\boxed{R_{ij} = \text{rank } i \text{ element is compared with rank } j \text{ element.}}$

Question: What is $\Pr[R_{ij}]$?

Analysis

$\boxed{R_{ij} = \text{rank } i \text{ element is compared with rank } j \text{ element.}}$

Question: What is $\Pr[R_{ij}]$?

	7	5	9	1	3	4	8	6
Ranks:	6	4	8	1	2	3	7	5

Analysis

$\boxed{R_{ij} = \text{rank } i \text{ element is compared with rank } j \text{ element.}}$

Question: What is $\Pr[R_{ij}]$?

	7	5	9	1	3	4	8	6
Ranks:	6	4	8	1	2	3	7	5

Probability of comparing 5 to 8 is $\Pr[R_{4,7}]$.

Analysis

$$\boxed{R_{ij} = \text{rank } i \text{ element is compared with rank } j \text{ element.}}$$

Question: What is $\Pr[R_{ij}]$?

	7	5	9	1	3	4	8	6
Ranks:	6	4	8	1	2	3	7	5

Probability of comparing 5 to 8 is $\Pr[R_{4,7}]$.

- If pivot too small (say 3 [rank 2]). Partition and call recursively:

$$\boxed{7\ 5\ 9\ 1\ 3\ 4\ 8\ 6} \Longrightarrow \boxed{1}\ \boxed{3}\ \boxed{7\ 5\ 9\ 4\ 8\ 6}$$

Decision if to compare 5 to 8 is moved to subproblem.

- If pivot too large (say 9 [rank 8]):

$$\boxed{7\ 5\ 9\ 1\ 3\ 4\ 8\ 6} \Longrightarrow \boxed{7\ 5\ 1\ 3\ 4\ 8\ 6}\ \boxed{9}$$

Decision if to compare 5 to 8 moved to subproblem.

Analysis

- If pivot is 5 (rank 4).

  7 5 9 1 3 4 8 6 $\Longrightarrow$ 1 3 4 | 5 | 7 9 8 6

- If pivot is 8 (rank 7).

  7 5 9 1 3 4 8 6 $\Longrightarrow$ 7 5 1 3 4 6 | 8 | 9

- If pivot in between the two numbers (say 6 [rank 5]):

  7 5 9 1 3 4 8 6 $\Longrightarrow$ 5 1 3 4 | 6 | 7 8 9

  5 and 8 will never be compared to each other.

Analysis

Question: What is $\Pr[R_{i,j}]$?

Conclusion

$R_{i,j}$ happens if and only if :

ith or jth ranked element is the first pivot out of the elements ranked i to j.

Analysis

Question: What is $\Pr[R_{ij}]$?

Analysis

Question: What is $\Pr[R_{ij}]$?

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

Analysis

Question: What is $\Pr[R_{ij}]$?

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

Proof.

Let $a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n$ be elements of A in sorted order. Let $S = \{a_i, a_{i+1}, \ldots, a_j\}$

Observation: If pivot is chosen outside S then all of S either in left array or right array.

Observation: a_i and a_j separated when a pivot is chosen from S for the first time. Once separated no comparison.

Observation: a_i is compared with a_j if and only if either a_i or a_j is chosen as a pivot from S at separation. $\square$

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

Proof.

Let $a_1, \ldots, a_i, \ldots, a_j, \ldots, a_n$ be sort of A. Let $S = \{a_i, a_{i+1}, \ldots, a_j\}$
Observation: a_i is compared with a_j if and only if either a_i or a_j is chosen as a pivot from S at separation.
Observation: Given that pivot is chosen from S the probability that it is a_i or a_j is exactly $\frac{2}{|S|} = \frac{2}{(j-i+1)}$ since the pivot is chosen uniformly at random from the array. $\square$

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

$$\mathbf{E}[Q(A)] = \sum_{1 \le i < j \le n} \Pr[R_{ij}]$$

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

$$\mathbf{E}[Q(A)] = \sum_{1 \leq i < j \leq n} \Pr[R_{ij}] \quad = \quad \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1}$$

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

$$
\begin{aligned}
\mathbf{E}[Q(A)] = \sum_{1 \le i < j \le n} \Pr[R_{ij}] &= \sum_{1 \le i < j \le n} \frac{2}{j-i+1} \\
&= 2 \sum_{i=1}^{n-1} \sum_{i<j}^{n} \frac{1}{j-i+1}
\end{aligned}
$$

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

$$\begin{aligned}
\mathbf{E}[Q(A)] = \sum_{1 \le i < j \le n} \Pr[R_{ij}] &= \sum_{1 \le i < j \le n} \frac{2}{j-i+1} \\
&= 2 \sum_{i=1}^{n-1} \sum_{i<j}^{n} \frac{1}{j-i+1} \\
&\le 2 \sum_{i=1}^{n-1} \sum_{\Delta=2}^{n-i+1} \frac{1}{\Delta}
\end{aligned}$$

Analysis

Lemma

$\Pr[R_{ij}] = \frac{2}{j-i+1}$.

$$
\begin{aligned}
\mathbf{E}[Q(A)] = \sum_{1\le i<j\le n} \Pr[R_{ij}] \quad &= \quad \sum_{1\le i<j\le n} \frac{2}{j-i+1} \\
&= \quad 2\sum_{i=1}^{n-1}\sum_{i<j}^{n} \frac{1}{j-i+1} \\
&\le \quad 2\sum_{i=1}^{n-1}\sum_{\Delta=2}^{n-i+1} \frac{1}{\Delta} \\
&\le \quad 2\sum_{1\le i<n} H_n
\end{aligned}
$$

Analysis

$H_n = \sum_{i=1}^{n} \frac{1}{i}$ is the n'th harmonic number

1. $H_n = \Theta(1)$.
2. $H_n = \Theta(\log \log n)$.
3. $H_n = \Theta(\sqrt{\log n})$.
4. $H_n = \Theta(\log n)$.
5. $H_n = \Theta(\log^2 n)$.

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

- Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
- **Worst-case analysis:** run time on worst input for a given size n.

$$T_{wc}(n) = \max_{x:|x|=n} Q(x).$$

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

- Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
- **Worst-case analysis:** run time on worst input for a given size n.

$$T_{wc}(n) = \max_{x:|x|=n} Q(x).$$

- **Average case analysis:** Assume inputs comes from a probability distribution. Analyze the algorithm's average performance over the distribution over inputs

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

- Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
- **Worst-case analysis:** run time on worst input for a given size n.

$$T_{wc}(n) = \max_{x:|x|=n} Q(x).$$

- **Average case analysis:** Assume inputs comes from a probability distribution. Analyze the algorithm's average performance over the distribution over inputs

Randomized algorithm R for a problem Π:

- Let $R(x)$ be the time for R to run on input x of length $|x|$.

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

- Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
- **Worst-case analysis:** run time on worst input for a given size n.

$$T_{wc}(n) = \max_{x:|x|=n} Q(x).$$

- **Average case analysis:** Assume inputs comes from a probability distribution. Analyze the algorithm's average performance over the distribution over inputs

Randomized algorithm R for a problem Π:

- Let $R(x)$ be the time for R to run on input x of length $|x|$.
- $R(x)$ is a random variable: depends on random bits used by R.

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

- Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
- **Worst-case analysis:** run time on worst input for a given size n.

$$T_{wc}(n) = \max_{x:|x|=n} Q(x).$$

- **Average case analysis:** Assume inputs comes from a probability distribution. Analyze the algorithm's average performance over the distribution over inputs

Randomized algorithm R for a problem Π:

- Let $R(x)$ be the time for R to run on input x of length $|x|$.
- $R(x)$ is a random variable: depends on random bits used by R.
- $\mathbf{E}[R(x)]$ is the expected running time for R on x

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

- Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
- **Worst-case analysis:** run time on worst input for a given size n.

$$T_{wc}(n) = \max_{x:|x|=n} Q(x).$$

- **Average case analysis:** Assume inputs comes from a probability distribution. Analyze the algorithm's average performance over the distribution over inputs

Randomized algorithm R for a problem Π:

- Let $R(x)$ be the time for R to run on input x of length $|x|$.
- $R(x)$ is a random variable: depends on random bits used by R.
- $\mathbf{E}[R(x)]$ is the expected running time for R on x
- Worst-case analysis: expected time on worst input of size n

$$T_{rand-wc}(n) = \max_{x:|x|=n} \mathbf{E}[R(x)].$$

Verifying Matrix Multiplication

Problem

Given three $n \times n$ matrices A, B, C is $AB = C$?

Deterministic algo: $O(n^3)$ (simple) and $O(n^{2.373})$ (complicated).

Verifying Matrix Multiplication

Problem

Given three $n \times n$ matrices A, B, C is $AB = C$?

Deterministic algo: $O(n^3)$ (simple) and $O(n^{2.373})$ (complicated).

Verifying Matrix Multiplication

Problem

Given three $n \times n$ matrices A, B, C is $AB = C$?

Deterministic algo: $O(n^3)$ (simple) and $O(n^{2.373})$ (complicated).

Randomized algo:

- Pick a random $n \times 1$ vector $r \in \{0,1\}^n$.
- Return the answer of the equality $ABr = Cr$.
- Running time: $O(n^2)$!

Verifying Matrix Multiplication

Problem

Given three $n \times n$ matrices A, B, C is $AB = C$?

Deterministic algo: $O(n^3)$ (simple) and $O(n^{2.373})$ (complicated).

Randomized algo:

- Pick a random $n \times 1$ vector $r \in \{0, 1\}^n$.
- Return the answer of the equality $ABr = Cr$.
- Running time: $O(n^2)$!

Theorem

If $AB = C$ then the algorithm will always say YES. If $AB \neq C$ then the algorithm will say YES with probability at most $1/2$. Can repeat the algorithm 100 times independently to reduce the probability of a false positive to $1/2^{100}$.

Analyzing Monte Carlo Algorithms

Randomized algorithm M for a problem Π:

- Let $M(x)$ be the time for M to run on input x of length $|x|$. For Monte Carlo, assumption is that run time is deterministic.
- Running time $T(n) = \max_{x:|x|=n} M(x)$.

Analyzing Monte Carlo Algorithms

Randomized algorithm M for a problem Π:

- Let $M(x)$ be the time for M to run on input x of length $|x|$. For Monte Carlo, assumption is that run time is deterministic.
- Running time $T(n) = \max_{x:|x|=n} M(x)$.
- Output depends on the random bits.
- Let $\Pr[x]$ be the probability that M is correct on x.
- Worst-case analysis: success probability on worst input

$$P_{rand-wc}(n) = \min_{x:|x|=n} \Pr[x].$$

Thank You.