

# Introduction to Randomized Algorithms: Quick Sort and Quick Selection

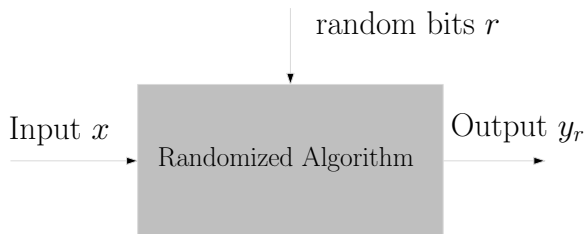
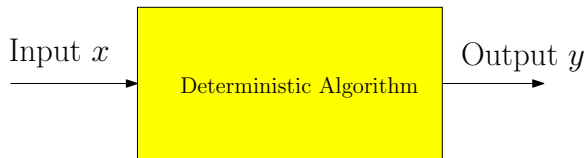
Lecture 14

March 10, 2011

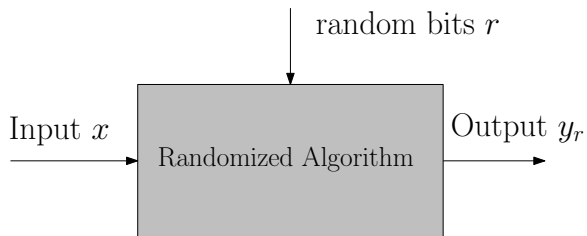
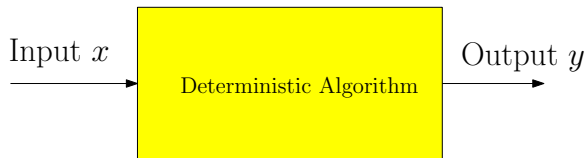
# Part I

## Introduction to Randomized Algorithms

# Randomized Algorithms



# Randomized Algorithms



# Example: Randomized QuickSort

## QuickSort [?]

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

## Randomized QuickSort

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

# Example: Randomized Quicksort

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

## Theorem

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

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

# Example: Randomized Quicksort

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

## Theorem

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

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

# Example: Randomized Quicksort

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

## Theorem

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

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



# Example: Verifying Matrix Multiplication

## Problem

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

Deterministic algorithm:

- Multiply  $A$  and  $B$  and check if equal to  $C$ .
- Running time?  $O(n^3)$  by straight forward approach.  $O(n^{2.37})$  with fast matrix multiplication (complicated and impractical).

# Example: Verifying Matrix Multiplication

## Problem

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

Deterministic algorithm:

- Multiply  $A$  and  $B$  and check if equal to  $C$ .
- Running time?  $O(n^3)$  by straight forward approach.  $O(n^{2.37})$  with fast matrix multiplication (complicated and impractical).

# Example: Verifying Matrix Multiplication

## Problem

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

Deterministic algorithm:

- Multiply  $A$  and  $B$  and check if equal to  $C$ .
- Running time?  $O(n^3)$  by straight forward approach.  $O(n^{2.37})$  with fast matrix multiplication (complicated and impractical).

# Example: Verifying Matrix Multiplication

## Problem

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

Randomized algorithm:

- Pick a random  $n \times 1$  vector  $r$ .
- 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  $1/2^{100}$ .*

# Example: Verifying Matrix Multiplication

## Problem

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

Randomized algorithm:

- Pick a random  $n \times 1$  vector  $r$ .
- 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  $1/2^{100}$ .*

# Example: Verifying Matrix Multiplication

## Problem

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

Randomized algorithm:

- Pick a random  $n \times 1$  vector  $r$ .
- 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  $1/2^{100}$ .*

# Example: Verifying Matrix Multiplication

## Problem

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

Randomized algorithm:

- Pick a random  $n \times 1$  vector  $r$ .
- 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  $1/2^{100}$ .

# Why randomized algorithms?

- Many many applications in algorithms, data structures and computer science!
- In some cases only known algorithms are randomized or randomness is provably necessary.
- Often randomized algorithms are (much) simpler and/or more efficient.
- Several deep connections to mathematics, physics etc.
- ...
- Lots of fun!



# Where do I get random bits?

**Question:** Are true random bits available in practice?

- Buy them!
- CPUs use physical phenomena to generate random bits.
- Can use pseudo-random bits or semi-random bits from nature. Several fundamental unresolved questions in complexity theory on this topic. Beyond the scope of this course.
- In practice pseudo-random generators work quite well in many applications.
- The model is interesting to think in the abstract and is very useful even as a theoretical construct. One can *derandomize* randomized algorithms to obtain deterministic algorithms.

# Average case analysis vs Randomized algorithms

## Average case analysis:

- Fix a deterministic algorithm.
- Assume inputs comes from a probability distribution.
- Analyze the algorithm's *average* performance over the distribution over inputs.

## Randomized algorithms:

- Algorithm uses random bits in addition to input.
- Analyze algorithms *average* performance over the given input where the average is over the random bits that the algorithm uses.
- On each input behaviour of algorithm is random. Analyze worst-case over all inputs of the (average) performance.

# Discrete Probability

We restrict attention to finite probability spaces.

## Definition

A discrete probability space is a pair  $(\Omega, \Pr)$  consists of finite set  $\Omega$  of *elementary* events and function  $\Pr : \Omega \rightarrow [0, 1]$  which assigns a probability  $\Pr[\omega]$  for each  $\omega \in \Omega$  such that  $\sum_{\omega \in \Omega} \Pr[\omega] = 1$ .

## Example

An unbiased coin.  $\Omega = \{H, T\}$  and  $\Pr[H] = \Pr[T] = 1/2$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

# Discrete Probability

We restrict attention to finite probability spaces.

## Definition

A discrete probability space is a pair  $(\Omega, \Pr)$  consists of finite set  $\Omega$  of *elementary* events and function  $\Pr : \Omega \rightarrow [0, 1]$  which assigns a probability  $\Pr[\omega]$  for each  $\omega \in \Omega$  such that  $\sum_{\omega \in \Omega} \Pr[\omega] = 1$ .

## Example

An unbiased coin.  $\Omega = \{H, T\}$  and  $\Pr[H] = \Pr[T] = 1/2$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

# Discrete Probability

And more examples

## Example

A biased coin.  $\Omega = \{H, T\}$  and  $\Pr[H] = 2/3, \Pr[T] = 1/3$ .

## Example

Two independent unbiased coins.  $\Omega = \{HH, TT, HT, TH\}$  and  $\Pr[HH] = \Pr[TT] = \Pr[HT] = \Pr[TH] = 1/4$ .

## Example

A pair of (highly) correlated dice.

$\Omega = \{(i, j) \mid 1 \leq i \leq 6, 1 \leq j \leq 6\}$ .

$\Pr[i, i] = 1/6$  for  $1 \leq i \leq 6$  and  $\Pr[i, j] = 0$  if  $i \neq j$ .

## Definition

Given a probability space  $(\Omega, \Pr)$  an **event** is a subset of  $\Omega$ . In other words an event is a collection of elementary events. The probability of an event  $\mathbf{A}$ , denoted by  $\Pr[\mathbf{A}]$ , is  $\sum_{\omega \in \mathbf{A}} \Pr[\omega]$ . The complement of an event  $\mathbf{A} \subseteq \Omega$  is the event  $\Omega \setminus \mathbf{A}$  frequently denoted by  $\bar{\mathbf{A}}$ .

# Events

## Examples

### Example

A pair of independent dice.  $\Omega = \{(i, j) \mid 1 \leq i \leq 6, 1 \leq j \leq 6\}$ .

- Let **A** be the event that the sum of the two numbers on the dice is even. Then  $\mathbf{A} = \{(i, j) \in \Omega \mid (i + j) \text{ is even}\}$ .

$$\Pr[\mathbf{A}] = |\mathbf{A}|/36 = 1/2.$$

- Let **B** be the event that the first die has 1. Then  $\mathbf{B} = \{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)\}$ .

$$\Pr[\mathbf{B}] = 6/36 = 1/6.$$

# Independent Events

## Definition

Given a probability space  $(\Omega, \Pr)$  and two events  $A, B$  are **independent** if and only if  $\Pr[A \cap B] = \Pr[A] \Pr[B]$ . Otherwise they are *dependent*. In other words  $A, B$  independent implies one does not affect the other.

## Example

Two coins.  $\Omega = \{HH, TT, HT, TH\}$  and  $\Pr[HH] = \Pr[TT] = \Pr[HT] = \Pr[TH] = 1/4$ .

- $A$  is the event that the first coin is heads and  $B$  is the event that second coin is tails.  $A, B$  are independent.
- $A$  is the event that the two coins are different.  $B$  is the event that the second coin is heads.  $A, B$  independent.



# Independent Events

## Definition

Given a probability space  $(\Omega, \Pr)$  and two events  $A, B$  are **independent** if and only if  $\Pr[A \cap B] = \Pr[A] \Pr[B]$ . Otherwise they are *dependent*. In other words  $A, B$  independent implies one does not affect the other.

## Example

Two coins.  $\Omega = \{HH, TT, HT, TH\}$  and  $\Pr[HH] = \Pr[TT] = \Pr[HT] = \Pr[TH] = 1/4$ .

- $A$  is the event that the first coin is heads and  $B$  is the event that second coin is tails.  $A, B$  are independent.
- $A$  is the event that the two coins are different.  $B$  is the event that the second coin is heads.  $A, B$  independent.

# Independent Events

## Examples

### Example

**A** is the event that both are not tails and **B** is event that second coin is heads. **A, B** are dependent.

# Random Variables

## Definition

Given a probability space  $(\Omega, \Pr)$  a (real-valued) random variable  $X$  over  $\Omega$  is a function that maps each elementary event to a real number. In other words  $X : \Omega \rightarrow \mathbb{R}$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

- $X : \Omega \rightarrow \mathbb{R}$  where  $X(i) = i \bmod 2$ .
- $Y : \Omega \rightarrow \mathbb{R}$  where  $Y(i) = i^2$ .

## Definition

A **binary random variable** is one that takes on values in  $\{0, 1\}$ .

# Random Variables

## Definition

Given a probability space  $(\Omega, \Pr)$  a (real-valued) random variable  $X$  over  $\Omega$  is a function that maps each elementary event to a real number. In other words  $X : \Omega \rightarrow \mathbb{R}$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

- $X : \Omega \rightarrow \mathbb{R}$  where  $X(i) = i \bmod 2$ .
- $Y : \Omega \rightarrow \mathbb{R}$  where  $Y(i) = i^2$ .

## Definition

A **binary random variable** is one that takes on values in  $\{0, 1\}$ .

# Random Variables

## Definition

Given a probability space  $(\Omega, \Pr)$  a (real-valued) random variable  $X$  over  $\Omega$  is a function that maps each elementary event to a real number. In other words  $X : \Omega \rightarrow \mathbb{R}$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

- $X : \Omega \rightarrow \mathbb{R}$  where  $X(i) = i \bmod 2$ .
- $Y : \Omega \rightarrow \mathbb{R}$  where  $Y(i) = i^2$ .

## Definition

A **binary random variable** is one that takes on values in  $\{0, 1\}$ .

# Indicator Random Variables

Special type of random variables that are quite useful.

## Definition

Given a probability space  $(\Omega, \Pr)$  and an event  $A \subseteq \Omega$  the indicator random variable  $X_A$  is a binary random variable where  $X_A(\omega) = 1$  if  $\omega \in A$  and  $X_A(\omega) = 0$  if  $\omega \notin A$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ . Let  $A$  be the event that  $i$  is divisible by 3. Then  $X_A(i) = 1$  if  $i = 3, 6$  and 0 otherwise.

# Indicator Random Variables

Special type of random variables that are quite useful.

## Definition

Given a probability space  $(\Omega, \Pr)$  and an event  $A \subseteq \Omega$  the indicator random variable  $X_A$  is a binary random variable where  $X_A(\omega) = 1$  if  $\omega \in A$  and  $X_A(\omega) = 0$  if  $\omega \notin A$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ . Let  $A$  be the event that  $i$  is divisible by 3. Then  $X_A(i) = 1$  if  $i = 3, 6$  and 0 otherwise.

# Expectation

## Definition

For a random variable  $\mathbf{X}$  over a probability space  $(\Omega, \Pr)$  the **expectation** of  $\mathbf{X}$  is defined as  $\sum_{\omega \in \Omega} \Pr[\omega] \mathbf{X}(\omega)$ . In other words, the expectation is the average value of  $\mathbf{X}$  according to the probabilities given by  $\Pr[\cdot]$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

- $\mathbf{X} : \Omega \rightarrow \mathbb{R}$  where  $\mathbf{X}(i) = i \bmod 2$ . Then  $\mathbf{E}[\mathbf{X}] = 1/2$ .
- $\mathbf{Y} : \Omega \rightarrow \mathbb{R}$  where  $\mathbf{Y}(i) = i^2$ . Then  $\mathbf{E}[\mathbf{Y}] = \sum_{i=1}^6 \frac{1}{6} \cdot i^2 = 91/6$ .



# Expectation

## Definition

For a random variable  $\mathbf{X}$  over a probability space  $(\Omega, \Pr)$  the **expectation** of  $\mathbf{X}$  is defined as  $\sum_{\omega \in \Omega} \Pr[\omega] \mathbf{X}(\omega)$ . In other words, the expectation is the average value of  $\mathbf{X}$  according to the probabilities given by  $\Pr[\cdot]$ .

## Example

A 6-sided unbiased die.  $\Omega = \{1, 2, 3, 4, 5, 6\}$  and  $\Pr[i] = 1/6$  for  $1 \leq i \leq 6$ .

- $\mathbf{X} : \Omega \rightarrow \mathbb{R}$  where  $\mathbf{X}(i) = i \bmod 2$ . Then  $\mathbf{E}[\mathbf{X}] = 1/2$ .
- $\mathbf{Y} : \Omega \rightarrow \mathbb{R}$  where  $\mathbf{Y}(i) = i^2$ . Then  $\mathbf{E}[\mathbf{Y}] = \sum_{i=1}^6 \frac{1}{6} \cdot i^2 = 91/6$ .

# Expectation

## Proposition

For an indicator variable  $X_A$ ,  $E[X_A] = \Pr[A]$ .

## Proof.

$$\begin{aligned} E[X_A] &= \sum_{y \in \Omega} X_A(y) \Pr[y] \\ &= \sum_{y \in A} 1 \cdot \Pr[y] + \sum_{y \in \Omega \setminus A} 0 \cdot \Pr[y] \\ &= \sum_{y \in A} \Pr[y] \\ &= \Pr[A]. \end{aligned}$$

# Linearity of Expectation

## Lemma

Let  $\mathbf{X}, \mathbf{Y}$  be two random variables over a probability space  $(\Omega, \Pr)$ .  
Then  $\mathbf{E}[\mathbf{X} + \mathbf{Y}] = \mathbf{E}[\mathbf{X}] + \mathbf{E}[\mathbf{Y}]$ .

## Proof.

$$\begin{aligned}\mathbf{E}[\mathbf{X} + \mathbf{Y}] &= \sum_{\omega \in \Omega} \Pr[\omega] (\mathbf{X}(\omega) + \mathbf{Y}(\omega)) \\ &= \sum_{\omega \in \Omega} \Pr[\omega] \mathbf{X}(\omega) + \sum_{\omega \in \Omega} \Pr[\omega] \mathbf{Y}(\omega) = \mathbf{E}[\mathbf{X}] + \mathbf{E}[\mathbf{Y}].\end{aligned}$$



## Corollary

$$\mathbf{E}[a_1 \mathbf{X}_1 + a_2 \mathbf{X}_2 + \dots + a_n \mathbf{X}_n] = \sum_{i=1}^n a_i \mathbf{E}[\mathbf{X}_i].$$

# Linearity of Expectation

## Lemma

Let  $\mathbf{X}, \mathbf{Y}$  be two random variables over a probability space  $(\Omega, \Pr)$ .  
Then  $\mathbf{E}[\mathbf{X} + \mathbf{Y}] = \mathbf{E}[\mathbf{X}] + \mathbf{E}[\mathbf{Y}]$ .

## Proof.

$$\begin{aligned}\mathbf{E}[\mathbf{X} + \mathbf{Y}] &= \sum_{\omega \in \Omega} \Pr[\omega] (\mathbf{X}(\omega) + \mathbf{Y}(\omega)) \\ &= \sum_{\omega \in \Omega} \Pr[\omega] \mathbf{X}(\omega) + \sum_{\omega \in \Omega} \Pr[\omega] \mathbf{Y}(\omega) = \mathbf{E}[\mathbf{X}] + \mathbf{E}[\mathbf{Y}].\end{aligned}$$



## Corollary

$$\mathbf{E}[\mathbf{a}_1 \mathbf{X}_1 + \mathbf{a}_2 \mathbf{X}_2 + \dots + \mathbf{a}_n \mathbf{X}_n] = \sum_{i=1}^n \mathbf{a}_i \mathbf{E}[\mathbf{X}_i].$$

# Types of Randomized Algorithms

Typically one encounters the following types:

- **Las Vegas randomized algorithms:** for a given input  $x$  output of algorithm is always correct but the running time is a random variable. In this case we are interested in analyzing the *expected* running time.
- **Monte Carlo randomized algorithms:** for a given input  $x$  the running time is deterministic but the output is random; correct with some probability. In this case we are interested in analyzing the *probability* of the correct output (and also the running time).
- Algorithms whose running time and output may both be random.

# Analyzing Las Vegas Algorithms

*Deterministic* algorithm  $\mathbf{Q}$  for a problem  $\Pi$ :

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

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

*Randomized* algorithm  $\mathbf{R}$  for a problem  $\Pi$ :

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

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

# Analyzing Las Vegas Algorithms

*Deterministic* algorithm **Q** for a problem  $\Pi$ :

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

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

*Randomized* algorithm **R** for a problem  $\Pi$ :

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

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

# Analyzing Monte Carlo Algorithms

*Randomized* algorithm **M** for a problem  $\Pi$ :

- Let  $\mathbf{M}(\mathbf{x})$  be the time for **M** to run on input  $\mathbf{x}$  of length  $|\mathbf{x}|$ . For Monte Carlo, assumption is that run time is deterministic.
- Let  $\mathbf{Pr}[\mathbf{x}]$  be the probability that **M** is correct on  $\mathbf{x}$ .
- $\mathbf{Pr}[\mathbf{x}]$  is a random variable: depends on random bits used by **M**.
- Worst-case analysis: success probability on worst input

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



## Part II

# Randomized Quick Sort and Selection

# Randomized QuickSort

## Randomized QuickSort

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

# Example

- array: 16, 12, 14, 20, 5, 3, 18, 19, 1

# Analysis via Recurrence

- Given array  $\mathbf{A}$  of size  $n$  let  $Q(\mathbf{A})$  be number of comparisons of randomized **QuickSort** on  $\mathbf{A}$ .
- Note that  $Q(\mathbf{A})$  is a random variable
- Let  $\mathbf{A}_{\text{left}}^i$  and  $\mathbf{A}_{\text{right}}^i$  be the left and right arrays obtained if:

pivot is of rank  $i$  in  $\mathbf{A}$ .

$$Q(\mathbf{A}) = n + \sum_{i=1}^n \Pr[\text{pivot has rank } i] \left( Q(\mathbf{A}_{\text{left}}^i) + Q(\mathbf{A}_{\text{right}}^i) \right)$$

Since each element of  $\mathbf{A}$  has probability exactly of  $1/n$  of being chosen:

$$Q(\mathbf{A}) = n + \sum_{i=1}^n \frac{1}{n} \left( Q(\mathbf{A}_{\text{left}}^i) + Q(\mathbf{A}_{\text{right}}^i) \right)$$

# Analysis via Recurrence

- Given array  $\mathbf{A}$  of size  $n$  let  $Q(\mathbf{A})$  be number of comparisons of randomized **QuickSort** on  $\mathbf{A}$ .
- Note that  $Q(\mathbf{A})$  is a random variable
- Let  $\mathbf{A}_{\text{left}}^i$  and  $\mathbf{A}_{\text{right}}^i$  be the left and right arrays obtained if:

pivot is of rank  $i$  in  $\mathbf{A}$ .

$$Q(\mathbf{A}) = n + \sum_{i=1}^n \Pr[\text{pivot has rank } i] \left( Q(\mathbf{A}_{\text{left}}^i) + Q(\mathbf{A}_{\text{right}}^i) \right)$$

Since each element of  $\mathbf{A}$  has probability exactly of  $1/n$  of being chosen:

$$Q(\mathbf{A}) = n + \sum_{i=1}^n \frac{1}{n} \left( Q(\mathbf{A}_{\text{left}}^i) + Q(\mathbf{A}_{\text{right}}^i) \right)$$

# Analysis via Recurrence

Let  $T(n) = \max_{A:|A|=n} E[Q(A)]$  be the worst-case expected running time of randomized **QuickSort** on arrays of size  $n$ .

We have, for any  $A$ :

$$Q(A) = n + \sum_{i=1}^n \Pr[\text{pivot has rank } i] \left( Q(A_{\text{left}}^i) + Q(A_{\text{right}}^i) \right)$$

Therefore, by linearity of expectation:

$$E[Q(A)] = n + \sum_{i=1}^n \Pr[\text{pivot of rank } i] \left( E[Q(A_{\text{left}}^i)] + E[Q(A_{\text{right}}^i)] \right).$$

$$\Rightarrow E[Q(A)] \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

# Analysis via Recurrence

Let  $T(n) = \max_{A:|A|=n} E[Q(A)]$  be the worst-case expected running time of randomized **QuickSort** on arrays of size  $n$ .

We have, for any  $A$ :

$$Q(A) = n + \sum_{i=1}^n \Pr[\text{pivot has rank } i] \left( Q(A_{\text{left}}^i) + Q(A_{\text{right}}^i) \right)$$

Therefore, by linearity of expectation:

$$E[Q(A)] = n + \sum_{i=1}^n \Pr[\text{pivot of rank } i] \left( E[Q(A_{\text{left}}^i)] + E[Q(A_{\text{right}}^i)] \right).$$

$$\Rightarrow E[Q(A)] \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

# Analysis via Recurrence

Let  $T(n) = \max_{A:|A|=n} E[Q(A)]$  be the worst-case expected running time of randomized **QuickSort** on arrays of size  $n$ .

We have, for any  $A$ :

$$Q(A) = n + \sum_{i=1}^n \Pr[\text{pivot has rank } i] \left( Q(A_{\text{left}}^i) + Q(A_{\text{right}}^i) \right)$$

Therefore, by linearity of expectation:

$$E[Q(A)] = n + \sum_{i=1}^n \Pr[\text{pivot of rank } i] \left( E[Q(A_{\text{left}}^i)] + E[Q(A_{\text{right}}^i)] \right).$$

$$\Rightarrow E[Q(A)] \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$



# Analysis via Recurrence

Let  $T(n) = \max_{A:|A|=n} E[Q(A)]$  be the worst-case expected running time of randomized **QuickSort** on arrays of size  $n$ .

We have, for any  $A$ :

$$Q(A) = n + \sum_{i=1}^n \Pr[\text{pivot has rank } i] \left( Q(A_{\text{left}}^i) + Q(A_{\text{right}}^i) \right)$$

Therefore, by linearity of expectation:

$$E[Q(A)] = n + \sum_{i=1}^n \Pr[\text{pivot of rank } i] \left( E[Q(A_{\text{left}}^i)] + E[Q(A_{\text{right}}^i)] \right).$$

$$\Rightarrow E[Q(A)] \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

# Analysis via Recurrence

Let  $T(n) = \max_{A:|A|=n} E[Q(A)]$  be the worst-case expected running time of randomized **QuickSort** on arrays of size  $n$ .

We derived:

$$E[Q(A)] \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

Note that above holds for any  $A$  of size  $n$ . Therefore

$$\max_{A:|A|=n} E[Q(A)] = T(n) \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

# Analysis via Recurrence

Let  $T(n) = \max_{A:|A|=n} E[Q(A)]$  be the worst-case expected running time of randomized **QuickSort** on arrays of size  $n$ .

We derived:

$$E[Q(A)] \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

Note that above holds for any  $A$  of size  $n$ . Therefore

$$\max_{A:|A|=n} E[Q(A)] = T(n) \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i)).$$

# Solving the Recurrence

$$T(n) \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i))$$

with base case  $T(1) = 0$ .

Lemma

$T(n) = O(n \log n)$ .

Proof.

(Guess and) Verify by induction. □

# Solving the Recurrence

$$T(n) \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i))$$

with base case  $T(1) = 0$ .

**Lemma**

$T(n) = O(n \log n)$ .

**Proof.**

(Guess and) Verify by induction. □

# Solving the Recurrence

$$T(n) \leq n + \sum_{i=1}^n \frac{1}{n} (T(i-1) + T(n-i))$$

with base case  $T(1) = 0$ .

Lemma

$T(n) = O(n \log n)$ .

Proof.

(Guess and) Verify by induction. □

# A Slick Analysis of QuickSort

Let  $Q(\mathbf{A})$  be number of comparisons done on input array  $\mathbf{A}$ :

- For  $1 \leq i < j \leq n$  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  $0$ .

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

and hence by linearity of expectation,

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

# A Slick Analysis of QuickSort

Let  $Q(\mathbf{A})$  be number of comparisons done on input array  $\mathbf{A}$ :

- For  $1 \leq i < j \leq n$  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  $0$ .

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

and hence by linearity of expectation,

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



# A Slick Analysis of QuickSort

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

Lemma

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

Proof.

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

**Observation:** If pivot is chosen outside  $\mathbf{S}$  then all of  $\mathbf{S}$  either in left array or right array.

**Observation:**  $a_i$  and  $a_j$  separated when a pivot is chosen from  $\mathbf{S}$  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  $\mathbf{S}$  at separation...

# A Slick Analysis of QuickSort

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

Lemma

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

Proof.

Let  $a_1, \dots, a_i, \dots, a_j, \dots, a_n$  be elements of  $\mathbf{A}$  in sorted order. Let

$$\mathbf{S} = \{a_i, a_{i+1}, \dots, a_j\}$$

**Observation:** If pivot is chosen outside  $\mathbf{S}$  then all of  $\mathbf{S}$  either in left array or right array.

**Observation:**  $a_i$  and  $a_j$  separated when a pivot is chosen from  $\mathbf{S}$  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  $\mathbf{S}$  at separation... □

# A Slick Analysis of QuickSort

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

Lemma

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

Proof.

Let  $\mathbf{a}_1, \dots, \mathbf{a}_i, \dots, \mathbf{a}_j, \dots, \mathbf{a}_n$  be elements of  $\mathbf{A}$  in sorted order. Let  $\mathbf{S} = \{\mathbf{a}_i, \mathbf{a}_{i+1}, \dots, \mathbf{a}_j\}$

**Observation:** If pivot is chosen outside  $\mathbf{S}$  then all of  $\mathbf{S}$  either in left array or right array.

**Observation:**  $\mathbf{a}_i$  and  $\mathbf{a}_j$  separated when a pivot is chosen from  $\mathbf{S}$  for the first time. Once separated no comparison.

**Observation:**  $\mathbf{a}_i$  is compared with  $\mathbf{a}_j$  if and only if either  $\mathbf{a}_i$  or  $\mathbf{a}_j$  is chosen as a pivot from  $\mathbf{S}$  at separation...  $\square$

# A Slick Analysis of QuickSort

Continued...

## Lemma

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

## Proof.

Let  $\mathbf{a}_1, \dots, \mathbf{a}_i, \dots, \mathbf{a}_j, \dots, \mathbf{a}_n$  be sort of  $\mathbf{A}$ . Let

$$\mathbf{S} = \{\mathbf{a}_i, \mathbf{a}_{i+1}, \dots, \mathbf{a}_j\}$$

**Observation:**  $\mathbf{a}_i$  is compared with  $\mathbf{a}_j$  if and only if either  $\mathbf{a}_i$  or  $\mathbf{a}_j$  is chosen as a pivot from  $\mathbf{S}$  at separation.

**Observation:** Given that pivot is chosen from  $\mathbf{S}$  the probability that it is  $\mathbf{a}_i$  or  $\mathbf{a}_j$  is exactly  $2/|\mathbf{S}| = 2/(j - i + 1)$  since the pivot is chosen uniformly at random from the array. □

# A Slick Analysis of QuickSort

Continued...

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

Lemma

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

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

# A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\begin{aligned} E[Q(A)] &= \sum_{1 \leq i < j \leq n} \Pr[R_{ij}] = \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = 2 \sum_{i=1}^{n-1} \sum_{i < j}^n \frac{1}{j-i+1} \\ &= 2 \sum_{i=1}^{n-1} (H_{n-i+1} - 1) \leq 2 \sum_{1 \leq i < n} H_n \\ &\leq 2nH_n = O(n \log n) \end{aligned}$$

# Randomized Quick Selection

**Input** Unsorted array **A** of **n** integers

**Goal** Find the **j**th smallest number in **A** (*rank j* number)

## Randomized Quick Selection

- Pick a pivot element *uniformly at random* from the array
- Split array into 3 subarrays: those smaller than pivot, those larger than pivot, and the pivot itself.
- Return pivot if rank of pivot is **j**
- Otherwise recurse on one of the arrays depending on **j** and their sizes.

# Algorithm for Randomized Selection

**Assume** for simplicity that **A** has distinct elements.

**QuickSelect**(**A**, **j**):

Pick pivot **x** uniformly at random from **A**

Partition **A** into **A<sub>less</sub>**, **x**, and **A<sub>greater</sub>** using **x** as pivot

**if** ( $|\mathbf{A}_{\text{less}}| = j - 1$ ) **then**

**return** **x**

**if** ( $|\mathbf{A}_{\text{less}}| \geq j$ ) **then**

**return** **QuickSelect**(**A<sub>less</sub>**, **j**)

**else**

**return** **QuickSelect**(**A<sub>greater</sub>**,  $j - |\mathbf{A}_{\text{less}}| - 1$ )



# Analysis via Recurrence

- Given array  $\mathbf{A}$  of size  $n$  let  $Q(\mathbf{A})$  be number of comparisons of randomized selection on  $\mathbf{A}$  for selecting rank  $j$  element.
- Note that  $Q(\mathbf{A})$  is a random variable
- Let  $\mathbf{A}_{\text{less}}^i$  and  $\mathbf{A}_{\text{greater}}^i$  be the left and right arrays obtained if pivot is rank  $i$  element of  $\mathbf{A}$ .
- Algorithm recurses on  $\mathbf{A}_{\text{less}}^i$  if  $j < i$  and recurses on  $\mathbf{A}_{\text{greater}}^i$  if  $j > i$  and terminates if  $j = i$ .

$$Q(\mathbf{A}) = n + \sum_{i=1}^{j-1} \Pr[\text{pivot has rank } i] Q(\mathbf{A}_{\text{greater}}^i) + \sum_{i=j+1}^n \Pr[\text{pivot has rank } i] Q(\mathbf{A}_{\text{less}}^i)$$

# Analysis via Recurrence

- Given array  $\mathbf{A}$  of size  $n$  let  $Q(\mathbf{A})$  be number of comparisons of randomized selection on  $\mathbf{A}$  for selecting rank  $j$  element.
- Note that  $Q(\mathbf{A})$  is a random variable
- Let  $\mathbf{A}_{\text{less}}^i$  and  $\mathbf{A}_{\text{greater}}^i$  be the left and right arrays obtained if pivot is rank  $i$  element of  $\mathbf{A}$ .
- Algorithm recurses on  $\mathbf{A}_{\text{less}}^i$  if  $j < i$  and recurses on  $\mathbf{A}_{\text{greater}}^i$  if  $j > i$  and terminates if  $j = i$ .

$$Q(\mathbf{A}) = n + \sum_{i=1}^{j-1} \Pr[\text{pivot has rank } i] Q(\mathbf{A}_{\text{greater}}^i) + \sum_{i=j+1}^n \Pr[\text{pivot has rank } i] Q(\mathbf{A}_{\text{less}}^i)$$

# Analyzing the Recurrence

As in **QuickSort** we obtain the following recurrence where  $T(n)$  is the worst-case expected time.

$$T(n) \leq n + \frac{1}{n} \left( \sum_{i=1}^{j-1} T(n-i) + \sum_{i=j}^n T(i-1) \right).$$

## Theorem

$$T(n) = O(n).$$

## Proof.

(Guess and) Verify by induction (see next slide). □

# Analyzing the recurrence

## Theorem

$$T(n) = O(n).$$

Prove by induction that  $T(n) \leq \alpha n$  for some constant  $\alpha \geq 1$  to be fixed later.

**Base case:**  $n = 1$ , we have  $T(1) = 0$  since no comparisons needed and hence  $T(1) \leq \alpha$ .

**Induction step:** Assume  $T(k) \leq \alpha k$  for  $1 \leq k < n$  and prove it for  $T(n)$ . We have by the recurrence:

$$\begin{aligned} T(n) &\leq n + \frac{1}{n} \left( \sum_{i=1}^{j-1} T(n-i) + \sum_{i=j}^n T(i-1) \right) \\ &\leq n + \frac{\alpha}{n} \left( \sum_{i=1}^{j-1} (n-i) + \sum_{i=j}^n (i-1) \right) \quad \text{by applying induction} \end{aligned}$$

# Analyzing the recurrence

$$\begin{aligned}T(n) &\leq n + \frac{\alpha}{n} \left( \sum_{i=1}^{j-1} (n-i) + \sum_{i=j}^n (i-1) \right) \\&\leq n + \frac{\alpha}{n} \left( (j-1)(2n-j)/2 + (n-j+1)(n+j-2)/2 \right) \\&\leq n + \frac{\alpha}{2n} (n^2 + 2nj - 2j^2 - 3n + 4j - 2) \\&\quad \text{above expression maximized when } j = (n+1)/2: \text{ calculus} \\&\leq n + \frac{\alpha}{2n} (3n^2/2 - n) \quad \text{substituting } (n+1)/2 \text{ for } j \\&\leq n + 3\alpha n/4 \\&\leq \alpha n \quad \text{for any constant } \alpha \geq 4\end{aligned}$$

# Comments on analyzing the recurrence

- Algebra looks messy but intuition suggest that the median is the hardest case and hence can plug  $j = n/2$  to simplify without calculus
- Analyzing recurrences comes with practice and after a while one can see things more intuitively

**John Von Neumann:**

*Young man, in mathematics you don't understand things. You just get used to them.*

# Notes

# Notes



# Notes

# Notes