Introduction to Randomized Algorithms: QuickSort

Lecture 2

August 27, 2020

Outline

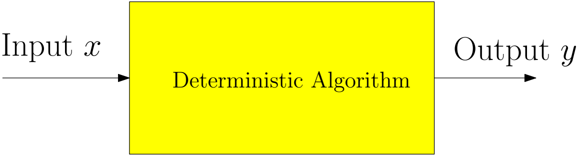
Today

- Randomized Algorithms – Two types
  - Las Vegas
  - Monte Carlo
- Randomized Quick Sort

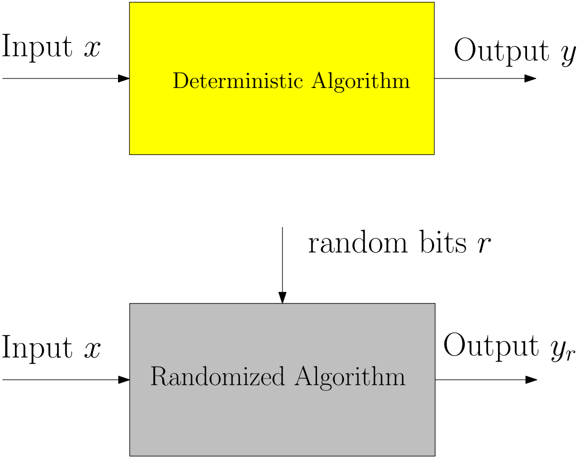
Part I

Introduction to Randomized Algorithms

Randomized Algorithms



Randomized Algorithms



Example: Randomized QuickSort

QuickSort ?

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

Randomized QuickSort

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

Example: Randomized Quicksort

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

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.*

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 $AB = C$?

Example: Verifying Matrix Multiplication

Problem

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

Deterministic algorithm:

1. Multiply A and B and check if equal to C.
2. 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:

1. Pick a random $n \times 1$ vector r.
2. Return the answer of the equality $ABr = Cr$.
3. Running time?

Example: Verifying Matrix Multiplication

Problem

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

Randomized algorithm:

1. Pick a random $n \times 1$ vector r.
2. Return the answer of the equality $ABr = Cr$.
3. Running time? $O(n^2)$!

Example: Verifying Matrix Multiplication

Problem

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

Randomized algorithm:

1. Pick a random $n \times 1$ vector r.
2. Return the answer of the equality $ABr = Cr$.
3. 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?

1. Many many applications in algorithms, data structures and computer science!
2. In some cases only known algorithms are randomized or randomness is provably necessary.
3. Often randomized algorithms are (much) simpler and/or more efficient.
4. Several deep connections to mathematics, physics etc.
5. ...
6. Lots of fun!

Average case analysis vs Randomized algorithms

Average case analysis:

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

Randomized algorithms:

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

Types of Randomized Algorithms

Typically one encounters the following types:

1. **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.

Types of Randomized Algorithms

Typically one encounters the following types:

1. **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.
2. **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).
3. Algorithms whose running time and output may both be random.

Analyzing Las Vegas Algorithms

Deterministic algorithm Q for a problem Π:

1. Let $Q(x)$ be the time for Q to run on input x of length $|x|$.
2. 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:

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

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

Randomized algorithm R for a problem Π:

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

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

Analyzing Monte Carlo Algorithms

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

1. 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.
2. Let $\mathbf{Pr}[x]$ be the probability that M is correct on x.
3. $\mathbf{Pr}[x]$ is a random variable: depends on random bits used by M.
4. Worst-case analysis: success probability on worst input

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

Part II

Randomized Quick Sort

Randomized QuickSort

Randomized QuickSort

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

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

Analysis

What events to count?

- Number of Comparisions.

Analysis

What events to count?

- Number of Comparisions.

What is the probability space?

- All the coin tosses at all levels and parts of recursion.

Analysis

What events to count?

- Number of Comparisions.

What is the probability space?

- All the coin tosses at all levels and parts of recursion.

Too Big!!

Analysis

What events to count?

- Number of Comparisions.

What is the probability space?

- All the coin tosses at all levels and parts of recursion.

Too Big!!

What random variables to define?
What are the events of the algorithm?

Analysis via Recurrence

1. Given array A of size n, let $Q(A)$ be number of comparisons of randomized **QuickSort** on A.
2. Note that $Q(A)$ is a random variable.
3. Let A^i_{left} and A^i_{right} be the left and right arrays obtained if rank i element chosen as pivot.

Let X_i be indicator random variable, which is set to 1 if pivot is of rank i in A, else zero.

$$Q(A) = n + \sum_{i=1}^{n} X_i \cdot \left(Q(A^i_{\text{left}}) + Q(A^i_{\text{right}}) \right).$$

Analysis via Recurrence

1. Given array A of size n, let $Q(A)$ be number of comparisons of randomized **QuickSort** on A.
2. Note that $Q(A)$ is a random variable.
3. Let A^i_{left} and A^i_{right} be the left and right arrays obtained if rank i element chosen as pivot.

Let X_i be indicator random variable, which is set to 1 if pivot is of rank i in A, else zero.

$$Q(A) = n + \sum_{i=1}^{n} X_i \cdot \left(Q(A^i_{\text{left}}) + Q(A^i_{\text{right}}) \right).$$

Since each element of A has probability exactly of $1/n$ of being chosen:

$$\mathbf{E}[X_i] = \mathbf{Pr}[\text{pivot has rank } i] = 1/n.$$

Independence of Random Variables

Lemma

Random variables X_i *is independent of random variables* $Q(A^i_{left})$ *as well as* $Q(A^i_{right})$, i.e.

$$\mathbf{E}\left[X_i \cdot Q(A^i_{left})\right] = \mathbf{E}[X_i]\,\mathbf{E}\left[Q(A^i_{left})\right]$$
$$\mathbf{E}\left[X_i \cdot Q(A^i_{right})\right] = \mathbf{E}[X_i]\,\mathbf{E}\left[Q(A^i_{right})\right]$$

Proof.

This is because the algorithm, while recursing on $Q(A^i_{\text{left}})$ and $Q(A^i_{\text{right}})$ uses new random coin tosses that are independent of the coin tosses used to decide the first pivot. Only the latter decides value of X_i. □

Analysis via Recurrence

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

Analysis via Recurrence

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

We have, for any A:

$$Q(A) = n + \sum_{i=1}^{n} X_i \left(Q(A^i_{\text{left}}) + Q(A^i_{\text{right}}) \right)$$

Analysis via Recurrence

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

We have, for any A:

$$Q(A) = n + \sum_{i=1}^{n} X_i \left(Q(A^i_{\text{left}}) + Q(A^i_{\text{right}}) \right)$$

By linearity of expectation, and independence random variables:

$$\mathbf{E}\Big[Q(A)\Big] = n + \sum_{i=1}^{n} \mathbf{E}[X_i] \Big(\mathbf{E}\Big[Q(A^i_{\text{left}})\Big] + \mathbf{E}\Big[Q(A^i_{\text{right}})\Big] \Big).$$

Analysis via Recurrence

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

We have, for any A:

$$Q(A) = n + \sum_{i=1}^{n} X_i \left(Q(A^i_{\text{left}}) + Q(A^i_{\text{right}}) \right)$$

By linearity of expectation, and independence random variables:

$$\mathbf{E}\Big[Q(A)\Big] = n + \sum_{i=1}^{n} \mathbf{E}[X_i] \Big(\mathbf{E}\Big[Q(A^i_{\text{left}})\Big] + \mathbf{E}\Big[Q(A^i_{\text{right}})\Big] \Big).$$

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

Analysis via Recurrence

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

Analysis via Recurrence

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

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

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

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

Solving the Recurrence

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

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

Solving the Recurrence

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

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

Lemma

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

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. □

Part III

Slick analysis of QuickSort

A Slick Analysis of QuickSort

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

1. For $1 \leq i < j < n$ let R_{ij} be the event that rank i element is compared with rank j element.
2. 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.

A Slick Analysis of QuickSort

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

1. For $1 \leq i < j < n$ let R_{ij} be the event that rank i element is compared with rank j element.
2. 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,

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

A Slick Analysis of QuickSort

R_{ij} = rank i element is compared with rank j element.

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

A Slick Analysis of QuickSort

R_{ij} = rank i element is compared with rank j element.

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

	7	5	9	1	3	4	8	6
With ranks:	6	4	8	1	2	3	7	5

A Slick Analysis of QuickSort

R_{ij} = rank i element is compared with rank j element.

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

	7	5	9	1	3	4	8	6
With ranks:	6	4	8	1	2	3	7	5

As such, probability of comparing **5** to **8** is $\textbf{Pr}[R_{4,7}]$.

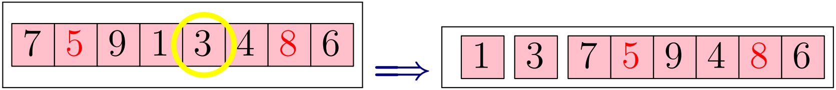
A Slick Analysis of QuickSort

R_{ij} = rank i element is compared with rank j element.

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

	7	5	9	1	3	4	8	6
With ranks:	6	4	8	1	2	3	7	5

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



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

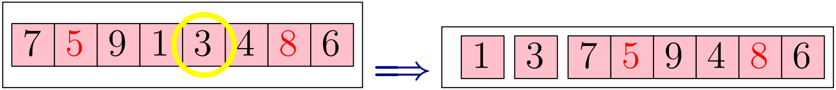
A Slick Analysis of QuickSort

R_{ij} = rank i element is compared with rank j element.

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

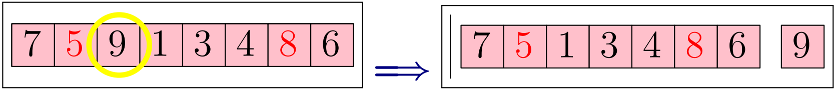
	7	5	9	1	3	4	8	6
With ranks:	6	4	8	1	2	3	7	5

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



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

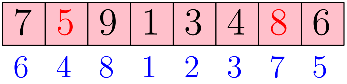
2. If pivot too large (say **9** [rank 8]):



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

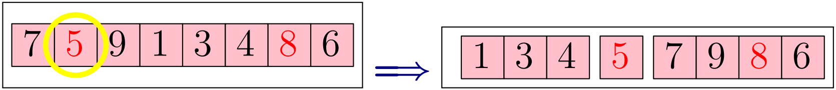
A Slick Analysis of QuickSort

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



As such, probability of comparing **5** to **8** is $\mathbf{Pr}[R_{4,7}]$.

1. If pivot is **5** (rank 4). Bingo!



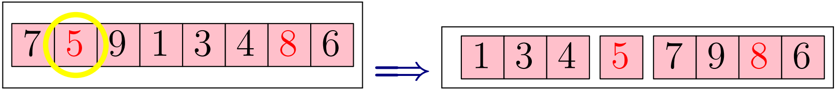
A Slick Analysis of QuickSort

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

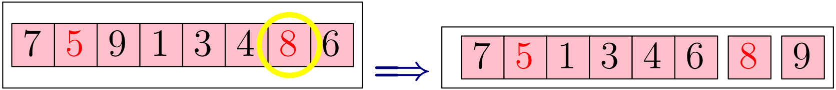
7	5	9	1	3	4	8	6
6	4	8	1	2	3	7	5

As such, probability of comparing **5** to **8** is $\mathbf{Pr}[R_{4,7}]$.

1. If pivot is **5** (rank 4). Bingo!



2. If pivot is **8** (rank 7). Bingo!



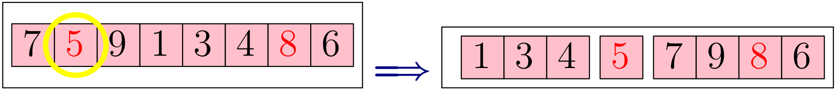
A Slick Analysis of QuickSort

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

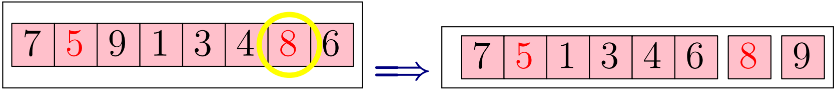
7	5	9	1	3	4	8	6
6	4	8	1	2	3	7	5

As such, probability of comparing **5** to **8** is $\Pr[R_{4,7}]$.

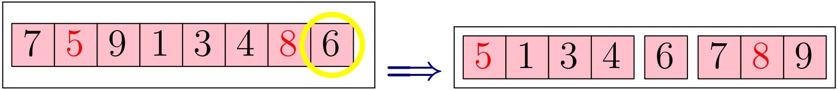
1. If pivot is **5** (rank 4). Bingo!



2. If pivot is **8** (rank 7). Bingo!



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



5 and **8** will never be compared to each other.

A Slick Analysis of QuickSort

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
> ith to jth ranked elements.

Digression

Consider the following experiment:

- Every day John decides whether to wear a tie by tossing a biased coin that comes up heads with probability $p > 0$ (and tails otherwise). He wears a tie if it comes up heads.
- If the coin is heads he tosses an unbiased coin to decide whether to wear a red tie or a blue tie.

Digression

Consider the following experiment:

- Every day John decides whether to wear a tie by tossing a biased coin that comes up heads with probability $p > 0$ (and tails otherwise). He wears a tie if it comes up heads.
- If the coin is heads he tosses an unbiased coin to decide whether to wear a red tie or a blue tie.

Question: What is the probability that John wore a red tie on the first day he wore a tie?

A Slick Analysis of QuickSort

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

A Slick Analysis of QuickSort

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

Lemma

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

A Slick Analysis of QuickSort

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

Lemma

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

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... □

A Slick Analysis of QuickSort

Continued...

Lemma

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

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 $2/|S| = 2/(j - i + 1)$ since the pivot is chosen uniformly at random from the array. □

How much is this?

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

(A) $H_n = \Theta(1)$.

(B) $H_n = \Theta(\log\log n)$.

(C) $H_n = \Theta(\sqrt{\log n})$.

(D) $H_n = \Theta(\log n)$.

(E) $H_n = \Theta(\log^2 n)$.

And how much is this?

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

is equal to

- **(A)** $T_n = \Theta(n)$.
- **(B)** $T_n = \Theta(n \log n)$.
- **(C)** $T_n = \Theta(n \log^2 n)$.
- **(D)** $T_n = \Theta(n^2)$.
- **(E)** $T_n = \Theta(n^3)$.

A Slick Analysis of QuickSort

Continued...

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

Lemma

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

A Slick Analysis of QuickSort

Continued...

Lemma

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

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

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1}$$

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1}$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = 2\sum_{i=1}^{n-1}\sum_{i<j}^{n}\frac{1}{j-i+1}$$

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = 2\sum_{i=1}^{n-1}\sum_{i<j}^{n}\frac{1}{j-i+1}$$

A Slick Analysis of QuickSort

Continued...

Lemma

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

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

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = 2\sum_{i=1}^{n-1}\sum_{i<j}^{n}\frac{1}{j-i+1} \le 2\sum_{i=1}^{n-1}\sum_{\Delta=2}^{n-i+1}\frac{1}{\Delta}$$

$$\le 2\sum_{i=1}^{n-1}(H_{n-i+1}-1) \;\le\; 2\sum_{1\le i<n} H_n$$

A Slick Analysis of QuickSort

Continued...

Lemma

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

$$\mathbf{E}\Big[Q(A)\Big] = 2\sum_{i=1}^{n-1}\sum_{i<j}^{n}\frac{1}{j-i+1} \le 2\sum_{i=1}^{n-1}\sum_{\Delta=2}^{n-i+1}\frac{1}{\Delta}$$

$$\le 2\sum_{i=1}^{n-1}(H_{n-i+1}-1) \le 2\sum_{1\le i<n} H_n$$

$$\le 2nH_n = O(n\log n)$$

Where do I get random bits?

Question: Are true random bits available in practice?

1. Buy them!
2. CPUs use physical phenomena to generate random bits.
3. 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.
4. In practice pseudo-random generators work quite well in many applications.
5. 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.