Design and Analysis of Algorithms

NUS
National University of Singapore

CS3230

Week 4

Randomized Algorithms

Warut Suksompong

Basics of Probability (Pre-requisite from CS1231)

- If you don’t recall the basics of probability, please review!

- For your convenience, some revision material on probability has been uploaded to LumiNUS.

Use of Probability in Algorithm Analysis

Example: Analysis of Quicksort

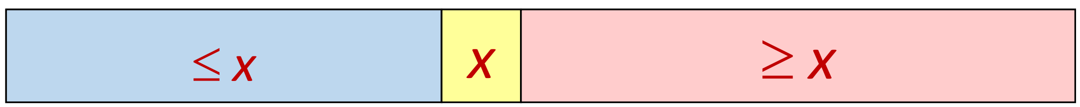
Quicksort

- Proposed by C.A.R. Hoare in 1961.
- A divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).

Divide and Conquer

Quicksort an n-element array:

1. ***Divide:*** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray $\leq x \leq$ elements in upper subarray.



2. ***Conquer:*** Recursively sort the two subarrays.
3. ***Combine:*** Trivial.

Key: *Linear-time partitioning subroutine.*

Pseudocode for Quick Sort

Assume, we select the <u>first element</u> of the array as pivot

QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT$(A, p, q-1)$
            QUICKSORT$(A, q+1, r)$

Initial call: QUICKSORT $(A, 1, n)$

Analysis of Quick Sort

- Let $T(n)$ = worst-case running time on an array of n elements.

- Let $A(n)$ = average-case running time on an array of n elements.

Pseudocode for Quick Sort

QUICKSORT(A, p, r)

if $p < r$

then $q \leftarrow$ PARTITION(A, p, r)

QUICKSORT$(A, p, q-1)$

QUICKSORT$(A, q+1, r)$

Takes $\Theta(n)$ time

Suppose the "pivot" produces the subarrays of size j and $(n - j - 1)$

$$T(n) = T(j) + T(n - j - 1) + \Theta(n)$$

Worst-case of quicksort

- Input sorted or reverse sorted (if we select the first element of an array as pivot).
- Partition around min or max element.
- One side of partition always has no elements.

$$
\begin{aligned}
T(n) &= T(0) + T(n-1) + \Theta(n) \\
&= \Theta(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2) \qquad \textit{(arithmetic series)}
\end{aligned}
$$

Average-case Analysis of Quick Sort

- Let $A(n)$ = average-case running time on an array of n elements.

- Our analysis assumes all input elements are distinct.
  - If duplicates exist, the running time of quicksort is better if we use a better partitioning algorithm.

Average-case Analysis of Quick Sort

0	1	2	3	4	5	6	7	8
6	**11**	**42**	**37**	**24**	**5**	**16**	**27**	**2**
e_3	e_4	e_9	e_8	e_6	e_2	e_5	e_7	e_1

0	1	2	3	4	5	6	7	8
15	**20**	**49**	**41**	**29**	**4**	**23**	**36**	**3**
e_3	e_4	e_9	e_8	e_6	e_2	e_5	e_7	e_1

Let e_i : ith **smallest** element of $\boldsymbol{A}$.

Observation: The execution of **Quick sort** depends upon the permutation of e_i's and **not** on the values taken by e_i's.

Average-case Analysis of Quick Sort

$\boldsymbol{A}(\boldsymbol{n})$ = Average running time for Quick sort on input of size $\boldsymbol{n}$

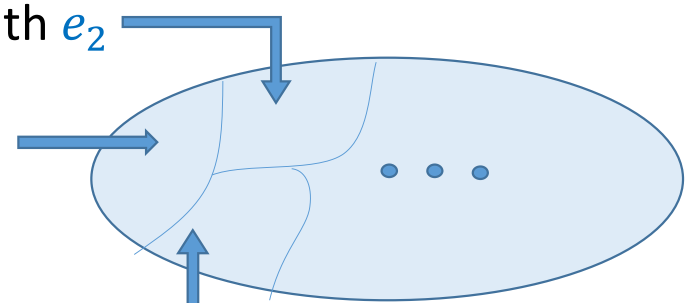
=Expected running time of Quick sort when the input is chosen uniformly at random from the set of all $n!$ Permutations (i.e., expectation is over the random choices of the input).

(average over <u>all possible permutations</u> of $\{e_1, e_2, \ldots, e_n\}$)

Hence, $\boldsymbol{A}(\boldsymbol{n}) = \frac{1}{\boldsymbol{n}!} \sum_{\pi} \boldsymbol{Q}(\pi)$,

where $\boldsymbol{Q}(\pi)$ is the time complexity (or no. of comparisons) when the input is permutation π.

Average-case Analysis of Quick Sort



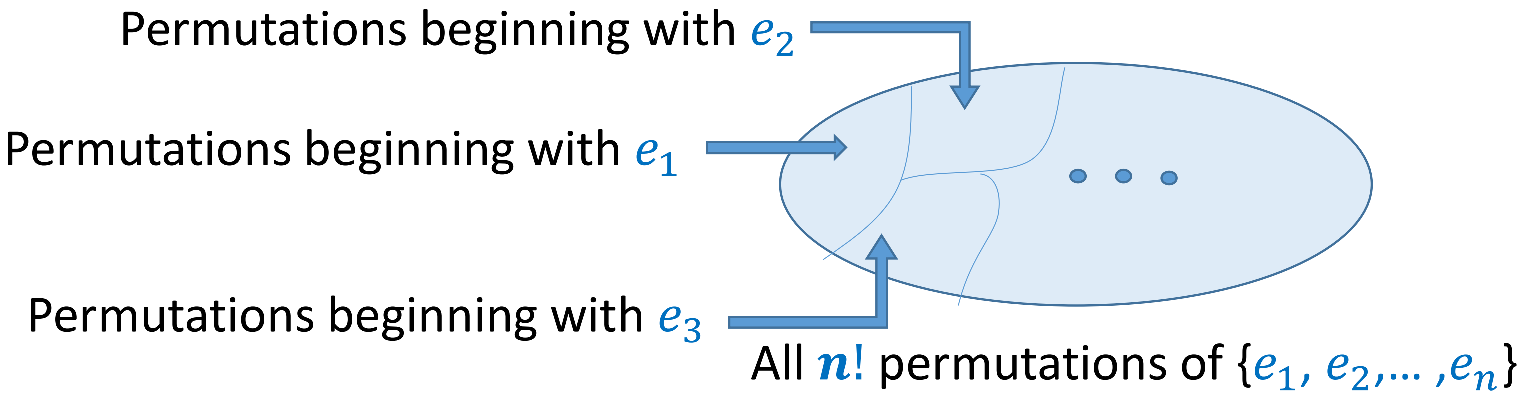
Let **P**($\boldsymbol{i}$) be the set of all those permutations of $\{e_1, e_2, \ldots, e_n\}$ that begin with e_i.

Question: What fraction of all permutations constitutes **P**($\boldsymbol{i}$) ?

Answer: $\frac{1}{\boldsymbol{n}}$

Let $\boldsymbol{G}(\boldsymbol{n}, \boldsymbol{i})$ be the average running time of **QuickSort** over **P**($\boldsymbol{i}$).

Average-case Analysis of Quick Sort



Question: What is the relation between $A(n)$ and $G(n, i)$'s ?

Answer: $A(n) = \frac{1}{n}\sum_{i=1}^{n} G(n, i)$

Observation: We now need to derive an expression for $G(n, i)$**.** For this purpose, we need to have a closer look at the execution of **QuickSort** over **P**(i).

Average-case Analysis of Quick Sort

- $\boldsymbol{G}(\boldsymbol{n}, \boldsymbol{i})$ = average running time of **QuickSort** over **P**($\boldsymbol{i}$).

- So, $\boldsymbol{G}(\boldsymbol{n}, \boldsymbol{i}) = \boldsymbol{A}(\boldsymbol{i} - \mathbf{1}) + \boldsymbol{A}(\boldsymbol{n} - \boldsymbol{i}) + (\boldsymbol{n} - \mathbf{1})$

- We have already seen, $\boldsymbol{A}(\boldsymbol{n}) = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{G}(\boldsymbol{n}, \boldsymbol{i})$

- So, $\boldsymbol{A}(\boldsymbol{n}) = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{A}(\boldsymbol{i} - \mathbf{1}) + \boldsymbol{A}(\boldsymbol{n} - \boldsymbol{i}) + \boldsymbol{n} - \mathbf{1})$

$$= \frac{2}{n} \sum_{i=1}^{n} \boldsymbol{A}(\boldsymbol{i} - \mathbf{1}) + \boldsymbol{n} - \mathbf{1}$$

Average-case Analysis of Quick Sort

$$\boldsymbol{A(n) = \frac{2}{n}\sum_{i=1}^{n} A(i-1) + n - 1}$$

- See lecture notes or in-class presentation for substitute-and-check proof that $A(n) = O(n \log n)$.

Merge Sort vs Quick Sort

No. of Comparisons	Merge Sort	Quick Sort
Average case	$n \log_2 n$	$1.39\, n \log_2 n$
Best case	$n \log_2 n$	$n \log_2 n$
Worst case	$n \log_2 n$	$n(n-1)$

After seeing this table, no one would prefer **Quick sort** to **Merge sort**

But **Quick sort** is still the most preferred algorithm in practice. **Why ?**

Merge Sort vs Quick Sort (in Practice)

Input: a random permutation of $\boldsymbol{n}$ numbers.

No. of repetitions: $\mathbf{1000}$

	$\boldsymbol{n} = \mathbf{100}$	$\boldsymbol{n} = \mathbf{1000}$	$\boldsymbol{n} \geq \mathbf{10000}$
No. of times **Merge sort** outperformed **Quick sort**	$\mathbf{0.1\%}$	$\mathbf{0.02\%}$	$\mathbf{0\%}$

Reasons:

- **Overhead** of temporary storage in Merge Sort
- *Cache* misses

What makes Quick Sort Popular?

No. of repetitions = 1000

No. of times run time exceeds average by	100	1000	10^4	10^5	10^6
10%	190	49	22	10	3
20%	28	17	12	3	0
50%	2	1	1	0	0
100%	0	0	0	0	0

Inference:

As n increases, the chances of deviation from average case

The *reliability* of quick sort

A Serious Problem with Quick Sort

- **Distribution sensitive:** Time taken depends on the initial (input) permutation

- Is real data random?

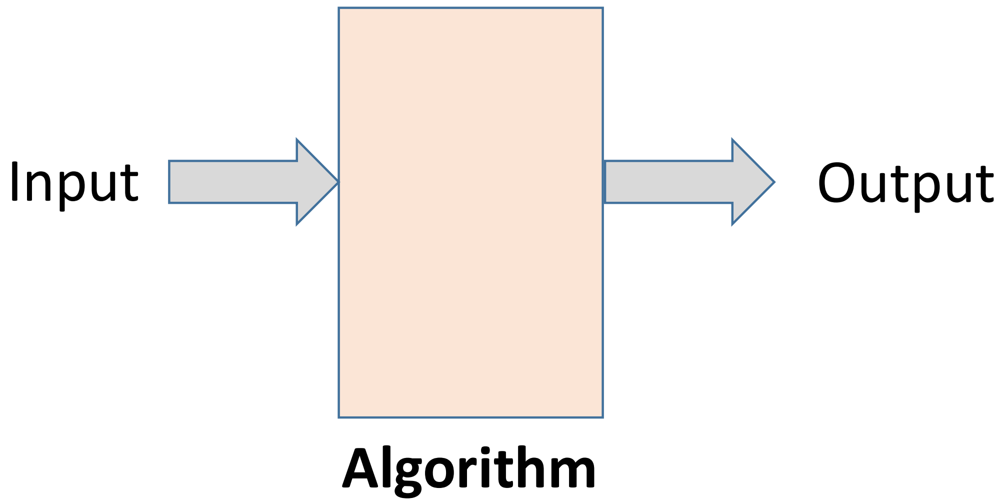
- Can we make Quick Sort distribution insensitive?

Remark: Worst-case vs Average-case

- Worst-case analysis is much more common than average-case

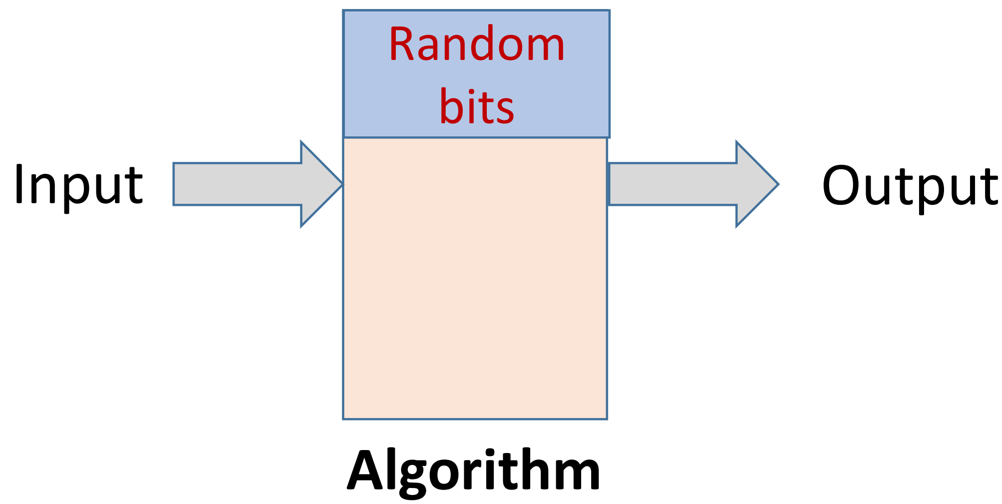
- **Reasons:**
  - It is often easier
  - To get a meaningful average-case result, a reasonable probability model on input is critical, but maybe unavailable or difficult to analyze

Algorithms



- The **output** as well as the **running time** are <u>**functions** only of the **input**</u>.

Randomized Algorithms



- The **output** as well as the **running time** are **functions of the input and random bits chosen**.

A Serious Problem with Quick Sort

- **Distribution sensitive:** Time taken depends on the initial (input) permutation
- Can we make Quick Sort distribution insensitive?

- What if we select the pivots uniformly at random from the array?

Quick Sort

QUICKSORT(A, p, r)

  if $p < r$

    then $q \leftarrow$ PARTITION(A, p, r)

      QUICKSORT$(A, p, q-1)$

      QUICKSORT$(A, q+1, r)$

Pick an element uniformly at random from A and make it the pivot

…ne, we select the first element of the array as pivot

Initial call: QUICKSORT $(A, 1, n)$

A Serious Problem with Quick Sort

- **Distribution sensitive:** Time taken depends on the initial (input) permutation
- Can we make Quick Sort distribution insensitive?

- What if we select the pivots uniformly at random from the array?
- **Distribution insensitive:** Time taken does **not** depend on initial permutation of $\boldsymbol{A}$.
- Time taken **depends** upon the **random** choices of pivot elements.

Analysis of Randomized Quick Sort

Theorem [Colin McDiarmid, **1991**]:

Probability that the run time of Randomized Quick Sort exceeds average by $x\%$ = $n^{-\frac{x}{100}\ln\ln n}$

➔Probability that run time of Randomized quick sort is **double** the average for $n \geq 10^6$ is 10^{-15}

What makes Randomized Algorithms so Popular?

[A study by *Microsoft* in $\mathbf{2008}$]

Title: Cycles, Cells and Platters: An Empirical Analysis of **Hardware Failures** on a Million Consumer PCs

Authors: **Edmund B. Nightingale, John R. Douceur, Vince Orgovan**

Available at : research.microsoft.com/pubs/144888/eurosys84-nightingale.pdf

$\boldsymbol{n \geq 10^6}$

Event	Probability
Your desktop will **crash** during this lecture	$> \mathbf{10^{-7}}$
RandQsort takes time at least **double** the average	$< \mathbf{10^{-15}}$

+ Simplicity

Types of Randomized Algorithms

Randomized Las Vegas Algorithms:

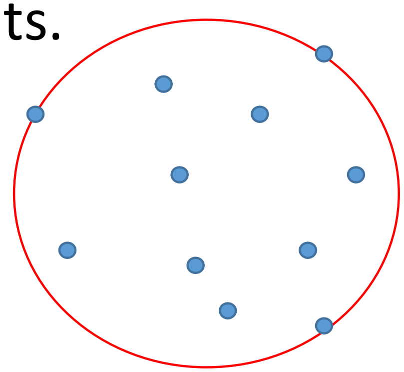
- Output is always correct
- Running time is a **random variable**

Randomized Monte Carlo Algorithms:

- Output may be incorrect with some small probability
- Running time is deterministic.

Motivating Example 1: Smallest Enclosing Circle

Problem definition: Given n points in a plane, compute the smallest radius circle that encloses all n points.



Best deterministic algorithm : [Megiddo, 1983]

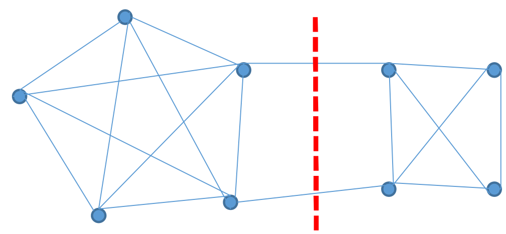
- **O(n)** time complexity, too **complex**, uses **advanced geometry**

Randomized Las Vegas algorithm: [Welzl, 1991]

- **Average O(n)** time complexity, very **simple**, uses **elementary geometry**

Motivating Example 2: Minimum Cut

Problem definition: Given a connected graph **G=(V,E)** on **n** vertices and **m** edges, compute the smallest set of edges whose removal will make G disconnected.



Best deterministic algorithm : [Stoer and **Wagner, 1997]**

- **O(mn)** time complexity.

Randomized Monte Carlo algorithm: [Karger, 1993]

- **O(m log n)** time complexity.
- Error probability: n^{-c} **for any c that we desire**

Motivating Example 3: Primality Testing

Problem definition: Given an n bit integer, determine if it is prime.

Applications:

- RSA-cryptosystem,
- Algebraic algorithms

Best deterministic algorithm : [Agrawal, Kayal and **Saxena, 2002]**

- **O(**n^6 **)** time complexity.

Randomized Monte Carlo algorithm: [Rabin, 1980]

- **O(**$k\ n^2$ **)** time complexity.
- Error probability: 2^{-k} **for any** k **that we desire**
- For k= 50, this probability is 10^{-15}

Randomized Quick Sort

Pick an element uniformly at random from A and make it the pivot

QUICKSORT(A, p, r)
if $p < r$
then $q \leftarrow$ PARTITION(A, p, r)
QUICKSORT$(A, p, q-1)$
QUICKSORT$(A, q+1, r)$

Assumption: All elements are distinct (if not, break the ties arbitrarily)

Notation : e_i : ith smallest element of array $\boldsymbol{A}$.

Randomized Quick Sort Analysis

$\text{QUICKSORT}(A, p, r)$
 if $p < r$
  then $q \leftarrow \text{PARTITION}(A, p, r)$
   $\text{QUICKSORT}(A, p,$
   $\text{QUICKSORT}(A, q+$

Maximum over inputs A of length n

Let $T(n)$ be the worst-case number of comparisons.

Note that $T(n)$ is a **random variable.**

Randomized Quick Sort Analysis

```
QUICKSORT(A, p, r)
    if p < r
        then q ← PARTITION(A, p, r)
             QUICKSORT(A, p, q–1)
             QUICKSORT(A, q+1, r)
```

$$T(n) = n - 1 + T(q - 1) + T(n - q)$$

Let $A(n) = \mathbb{E}[T(n)]$ where the expectation is over the randomness in the algorithm.

Taking expectations and applying linearity of expectations:

$$A(n) = n - 1 + \frac{1}{n}\sum_{q=1}^{n}\left(A(q-1) + A(n-q)\right) = n - 1 + \frac{2}{n}\sum_{q=1}^{n-1} A(q)$$

This is the same as the recurrence for average-case quicksort!

Randomized QuickSelect

```
QUICKSELECT(A, p, r, k)
  if p < r
    then q ← PARTITION(A, p, r)
      if q==k
        then return A[q]
      else if q > k
        then QUICKSELECT(A, p, q – 1, k)
      else QUICKSELECT(A, q+1, r, k-q)
```

- # Comparisons doesn't halve! Becomes $O(n)$. See lecture notes.

Geometric Distribution

- Suppose you flip a fair coin until it comes up heads. What is the expected number of times you need to flip?

- Let X be the number of times. Note that X is a random variable.
- X follows a **geometric distribution** with probability p = 1/2
- Pr[X = 1] = 1/2, Pr[X = 2] = 1/4, Pr[X = 3] = 1/8, …
- **Fact:** E[X] = 1/p

Acknowledgement

- The slides are modified from
  - the slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - the slides from Prof. Surender Baswana
  - the slides from Prof. Leong Hon Wai
  - the slides from Prof. Lee Wee Sun
  - the slides from Prof. Ken Sung
  - the slides from Prof. Diptarka Chakraborty
  - the slides from Prof. Arnab Bhattacharyya