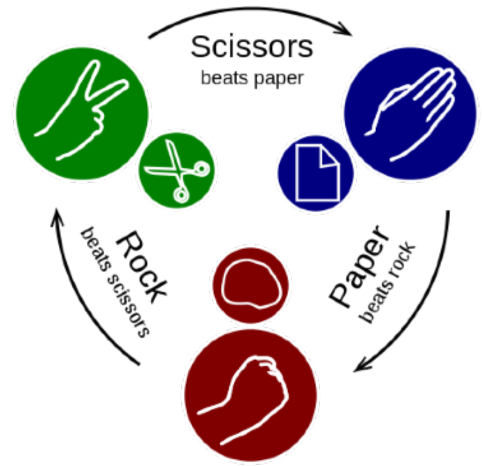
CSE202
Design and Analysis of Algorithms

Week 6 — Randomized Algorithms 1: Principle & First Examples

Randomness is Useful

For several problems,
the best known solution
uses randomness



Best strategy
in some games

Factoring an integer N

Deterministic: $O(N^{1/4+\epsilon})$

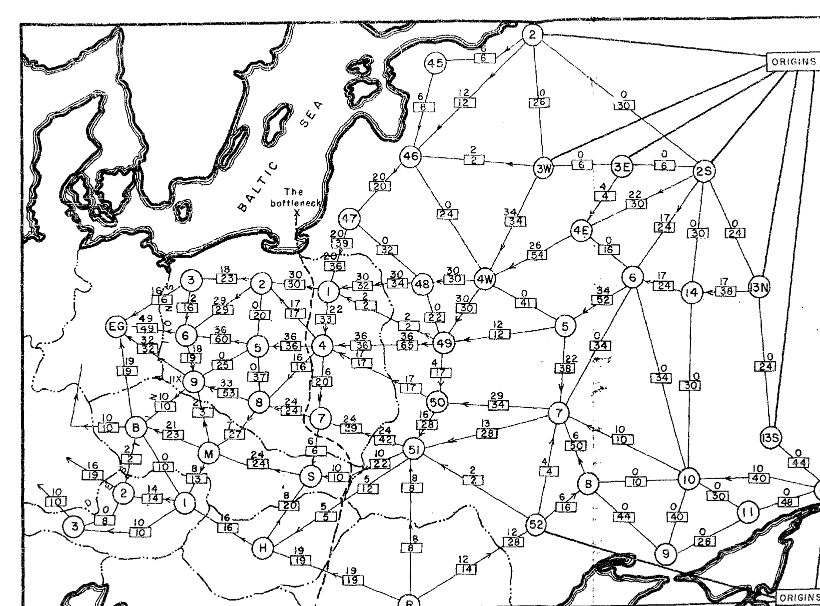
Probabilistic: $O(\exp((\log N)^{1/3+\epsilon}))$

Min-Cut in a Graph
with m edges, n vertices

$O(mn + n^2 \log n)$

$O(n^2 \log n)$

Next tutorial



Two Flavours of Randomized Algorithms

Las Vegas: always gives the correct answer. Running time is a random variable.

Monte Carlo: sometimes incorrect, but with bounded probability.

I. Toy Monte Carlo Example: Freivalds' Algorithm

Simple Way to Check Matrix Product

Input:
3 $n \times n$ matrices

$$\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix} \times \begin{pmatrix} & & \\ & B & \\ & & \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} & & \\ & C & \\ & & \end{pmatrix}$$

Direct approach: compute $D = C - AB$ and test whether $D = 0$.

Complexity: matrix multiplication, $O(n^{2.38})$ (in theory)

Freivalds' Algorithm

1. Pick a random v uniformly in $\{0,1\}^n$
2. Compute $w := Cv - A(Bv)$
3. Return $(w = 0)$

Complexity

$O(n^2)$

optimal

Might be wrong, but not too often

Probability of Error

The only possible error is when $D \neq 0$ but $w = 0$.

$$\Pr(D \neq 0 \wedge w = 0) \leq ??$$

$$\Pr(D \neq 0 \wedge w = 0) \leq \Pr(d^t \cdot v = 0), \; d \text{ non-zero row of } D$$

$$= \Pr\left(d_i v_i = -\sum_{j>i} d_j v_j\right), \quad \text{for the first } d_i \neq 0$$

$d_i v_i$ takes 2 values, each with proba 1/2,
while the sum is independent of d_i

$$\leq 1/2.$$

Drawing the coordinates of v from a set of size k reduces the bound to $1/k$.

Boosting the Probability

Repeating the algorithm k times leads to

$$\boxed{\Pr(k \text{ errors}) \le 1/2^k\,.}$$

$k = 10$ means a probability $\le 0.1\,\%$

$k = 100$ means a probability $\le 10^{-30}$

If your algorithm has 90% chance of being wrong, iterating it 66 times reduces the probability of error to < 0.1%.

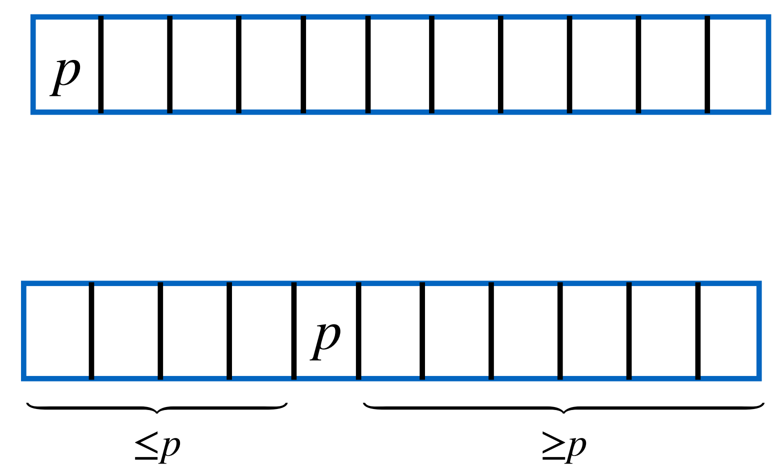
II. Quicksort

One of the top 10 algorithms of the 20th century.
IEEE J. Comput. in Science & Engineering

Recall: QuickSort Partitioning (CSE103)

Input: an array of *n* comparable elements
a pivot *p* among them

Output: array partitioned around *p*;
new index of *p*.



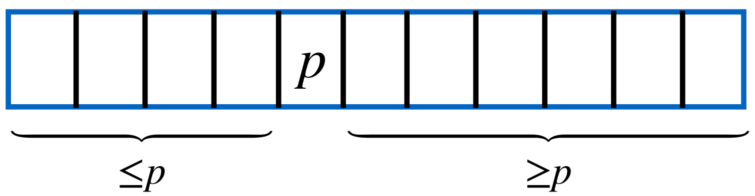
Runs in-place.

```
def partition(A,lo,hi):
    p = A[lo]; i = lo; j = hi
    while True:
        for i in range(i+1,hi):
            if A[i]>=p: break
        for j in range(j-1,lo-1,-1):
            if A[j]<=p: break
        if i>=j: break
        A[i],A[j]=A[j],A[i]
    A[lo],A[j]=A[j],A[lo]
    return j
```

Complexity:
$n - 1$ comparisons
(all of them with p)

Exercise:
identify appropriate variants/invariants and prove correctness.

Recall Quicksort from CSE 103



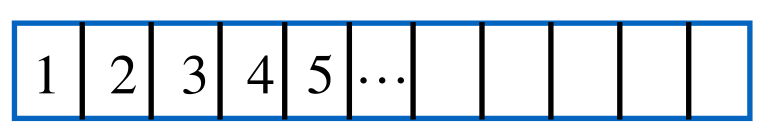
1. Partition
2. Sort subarrays recursively

Deterministic variant:

```
def sort(A):
    quicksort(A,0,len(A))

def quicksort(A,lo,hi):
    if hi<=lo+1: return
    q = partition(A,lo,hi)
    quicksort(A,lo,q)
    quicksort(A,q+1,hi)
```

Quadratic worst-case on a sorted array:



. 1 is compared with the n-1 other entries

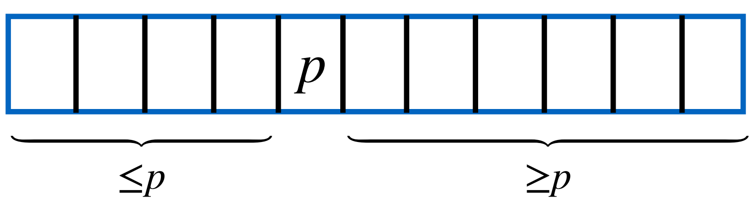
. sorting is called recursively on A[2:n]

$(n-1)+(n-2)+\cdots = n(n-1)/2 = O(n^2)$ comparisons.

Average-Case Complexity

Strong Hypothesis*: the keys are distinct and all permutations of the input are equally likely*

Observation: this property is preserved by partitioning



$$C_0 = 0 \qquad C_n := \text{num. comparisons}$$

$$C_n = \underbrace{n-1}_{\text{partition}} + C_{i-1} + C_{n-i} \quad \text{if pivot at index } i \quad \text{prob. } 1/n$$

Average number of comparisons $E_n := \mathbb{E}(C_n)$

$$E_n = n - 1 + \sum_{i=1}^{n} \frac{E_{i-1} + E_{n-i}}{n}$$

$$= 2(n+1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right) - 4n$$

$$= 2n\log n + 2(\gamma - 2)n + O(1) \approx 1.39 n \log_2 n - 2.85n$$

Perfect partitioning:
$C_n \leq n - 1 + 2C(\lceil n/2 \rceil)$
$= O(n \log n)$
(Master Theorem)

Euler's constant
$\gamma \approx 0.577$

Proof
on the blackboard

Randomized Version

A simple change:

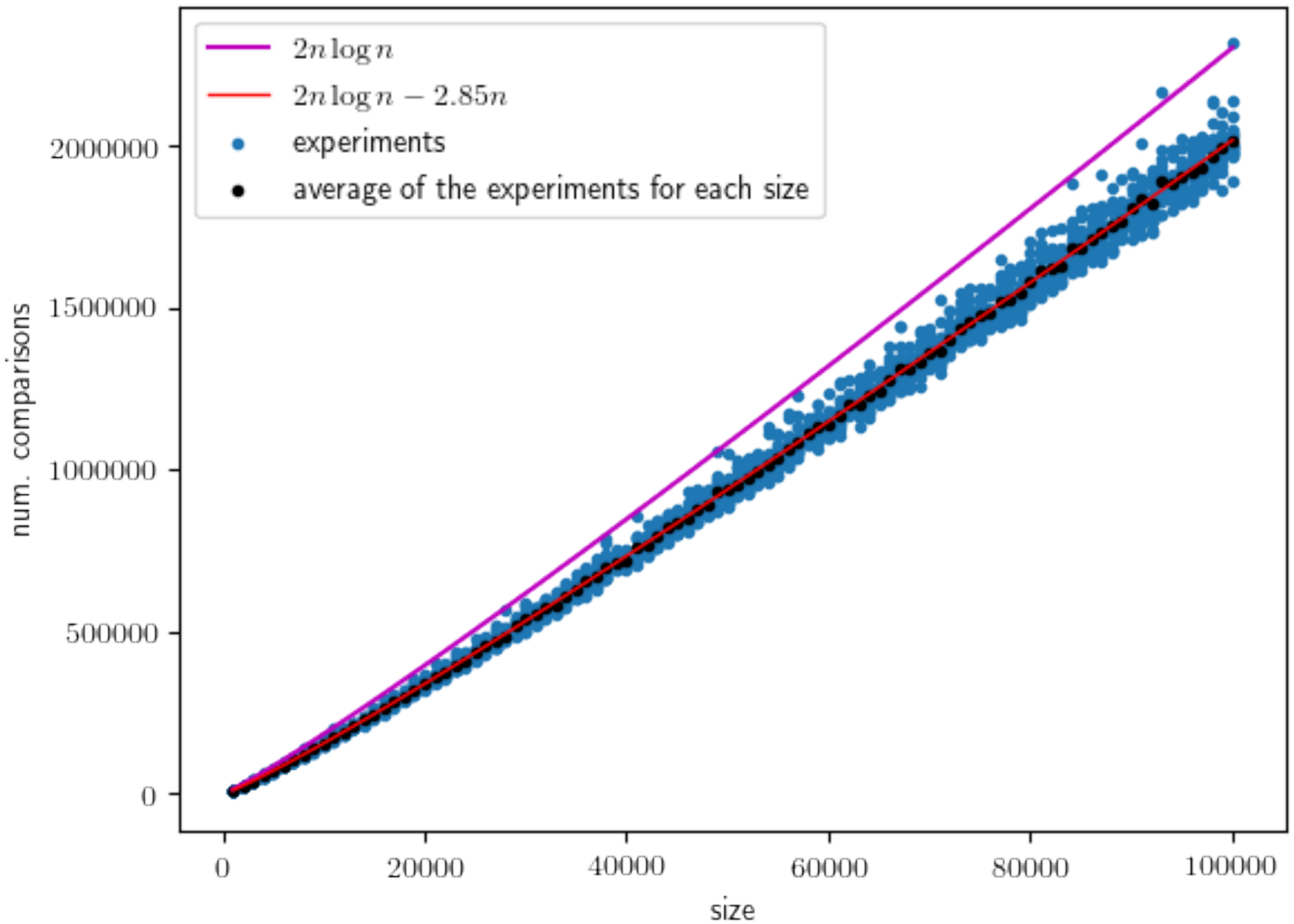
```
import random
def sort(A):
    random.shuffle(A)
    quicksort(A,0,len(A))
```

randomize the input

(in $O(n)$ ops)

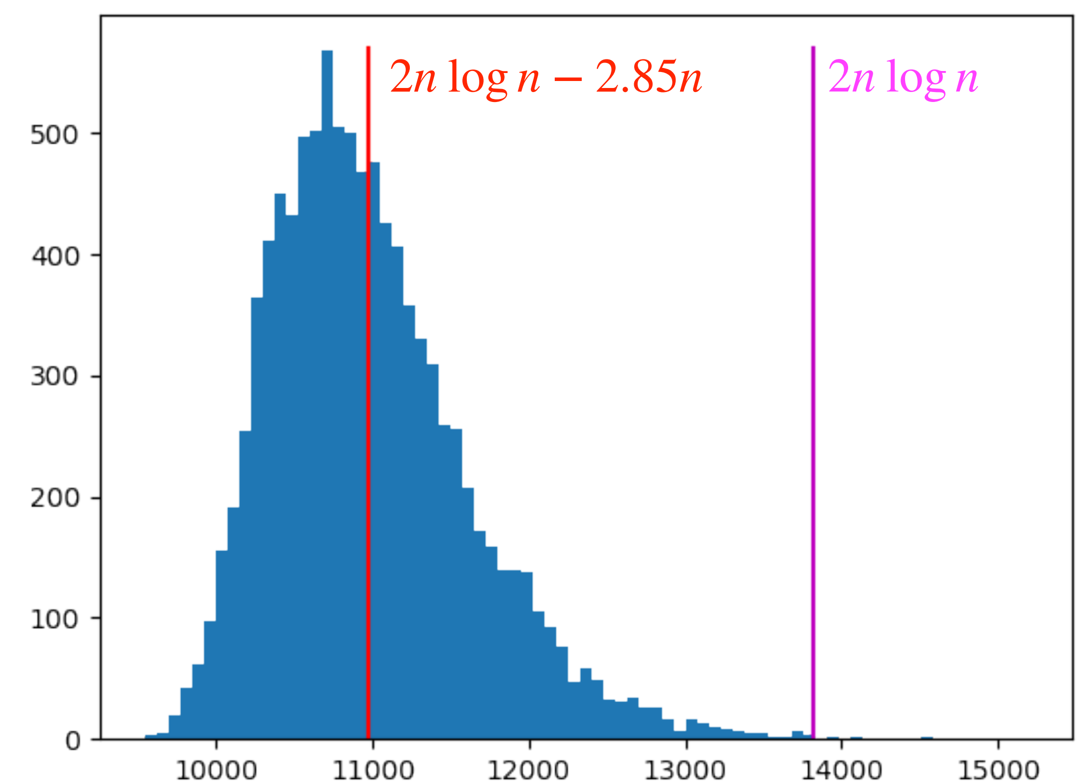
Now, for arbitrarily bad input,
the expected number of comparisons is
$\approx 2n \log n - 2.85n$

The Average has Predictive Value



number of comparisons
for sizes 1,000 to 100,000
(20 experiments per size)

10,000 experiments in size 1,000,
sorted by number of comparisons



The worst-case remains quadratic, but unlikely

Bounding the Bad Cases

Variance of the number of comparisons:

Proof:
3 pages of technical computation. Omitted

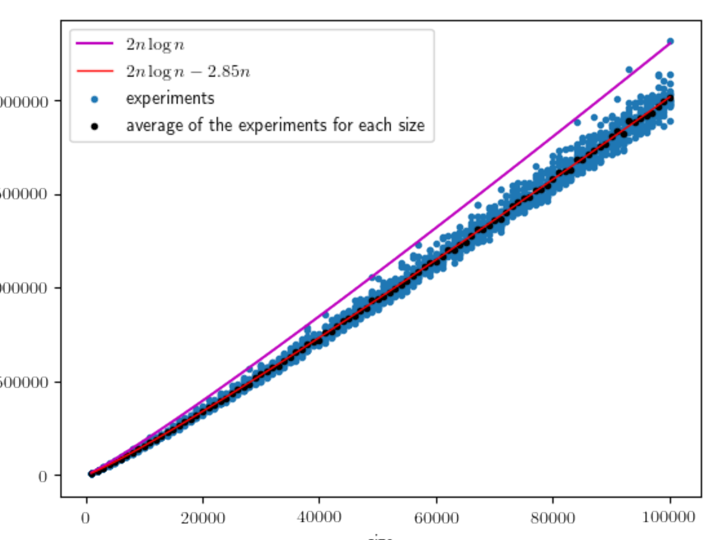
$$V_n := \mathbb{E}((C_n - E_n)^2) = (7 - 2\pi^2/3)\, n^2 + O(n \log n), \quad n \to \infty\,.$$

Standard deviation:

$$\sigma_n := \sqrt{V_n} \approx 1.93\, n$$

Chebyshev's inequality:

$$\Pr(\,|\,C_n - E_n\,| \geq k\sigma_n) \leq \frac{1}{k^2}\,.$$



n	1,000	10,000	100,000
$\Pr(C_n \geq 3\, n \log n)$	$\leq 3\; 10^{-3}$	$\leq 3\; 10^{-3}$	$\leq 3\; 10^{-3}$
$\Pr(C_n \geq 10\, n \log n)$	$\leq 2\; 10^{-4}$	$\leq 8\; 10^{-5}$	$\leq 5\; 10^{-5}$
$\Pr(C_n \geq 0.1\, n^2)$	$\leq 6\; 10^{-5}$	$\leq 5\; 10^{-7}$	$\leq 5\; 10^{-9}$

Quicksort vs Mergesort

	Quicksort	Mergesort
running time	$n \log n$	$n \log n$
in place?	yes	no
extra space	$\log n$	n
deterministic	no	yes

Quicksort is the fastest general-purpose sort.
R. Sedgewick

Further Improvements

Median-of-three partitioning for better pivots:

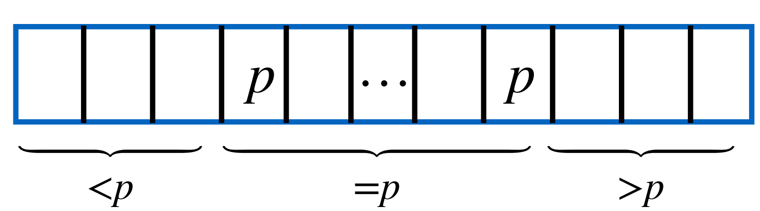
Take 3 elements at random;
use their median as the pivot;
(bonus) the other two as sentinels.

$\sim \frac{12}{7} n \log n$
30% improvement

Cutoff to insertion sort: stop recursion at size ≈10

One sweep of insertion sort on the whole array

Three-way partitioning for duplicate keys



See exercise

III. QuickSelect

QuickSelect

Select: $(A := \{a_1, \dots, a_n\}, k) \mapsto x \in A$ s.t. $|\{a \in A \mid a \le x\}| = k$

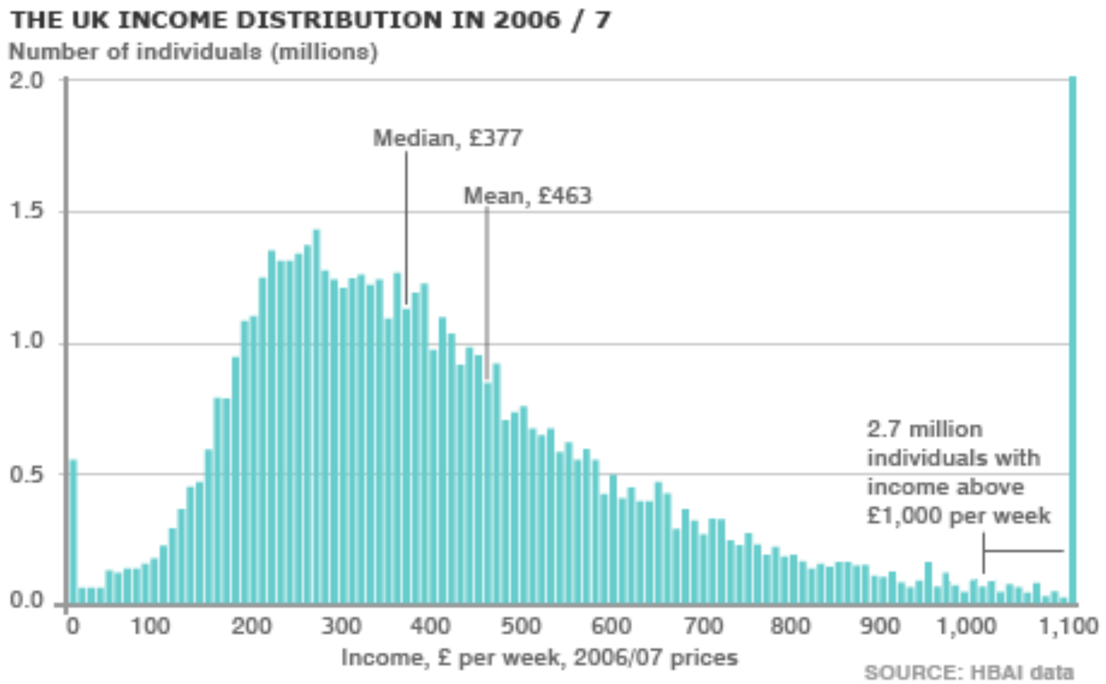
Median: Select with $k = \lfloor n/2 \rfloor$

Sorting gives an algorithm in $O(n \log n)$ comparisons

Median vs Mean:

```
def select(A,k):
    random.shuffle(A)
    return quickselect(A,0,len(A),k)

def quickselect(A,lo,hi,k):
    q = partition(A,lo,hi)
    if q==k: return A[q]
    if q>k: return quickselect(A,lo,q,k)
    return quickselect(A,q+1,hi,k)
```



Only a linear number of comparisons!

Simple Linear Upper Bound

Bound by recursion on the biggest side

$$E_n \le n - 1 + \frac{1}{n}\sum_{i=1}^{n} E_{\max(i-1,n-i)}$$

$$\le n - 1 + \frac{2}{n}\sum_{i=\lceil n/2\rceil}^{n-1} E_i + \begin{cases} \frac{1}{n}E_{\lfloor n/2\rfloor}, & \text{if } n \text{ is odd,} \\ 0 & \text{otherwise.} \end{cases}$$

Then, by induction $E_n \le 4n + 1.$

Average number of comparisons: $\sim 2(\ln 2 + 1)n \approx 3.39\, n$ (much more difficult)

Proof. $$n - 1 + \frac{2}{n}\sum_{i=\lceil n/2\rceil}^{n-1}(4i+1) + \frac{1}{n}(2n+1)$$

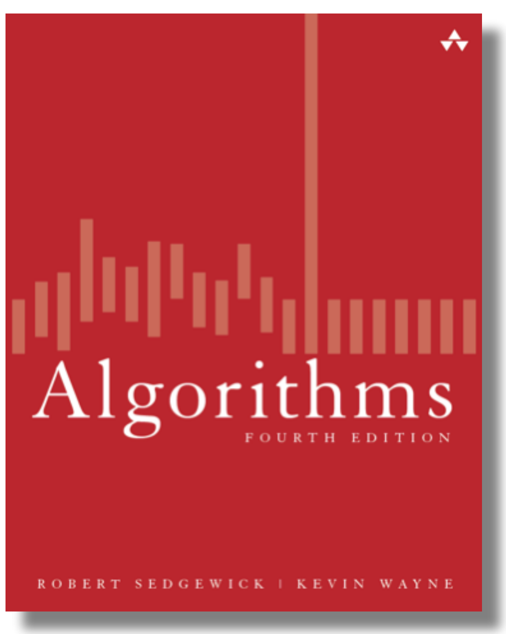
$$\le n - 1 + \frac{2}{n}\left(n(2n-1) - \frac{n}{2}(n-1)\right) + \frac{1}{n}(2n+1)$$

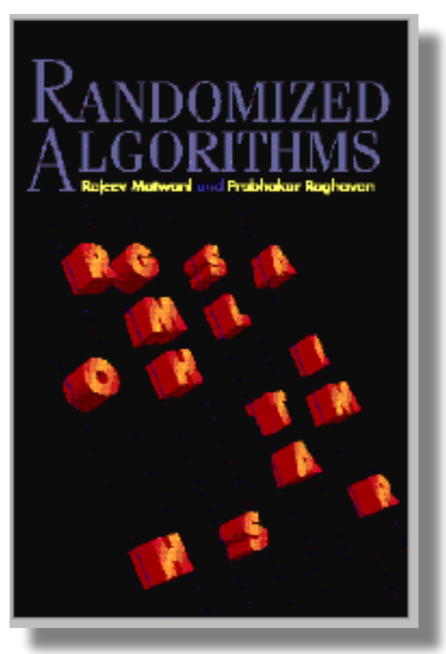
$$= 4n + 1/n\,.$$

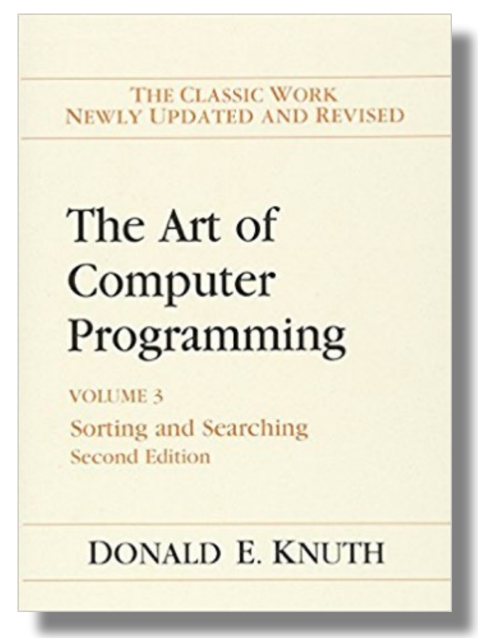
References for this lecture

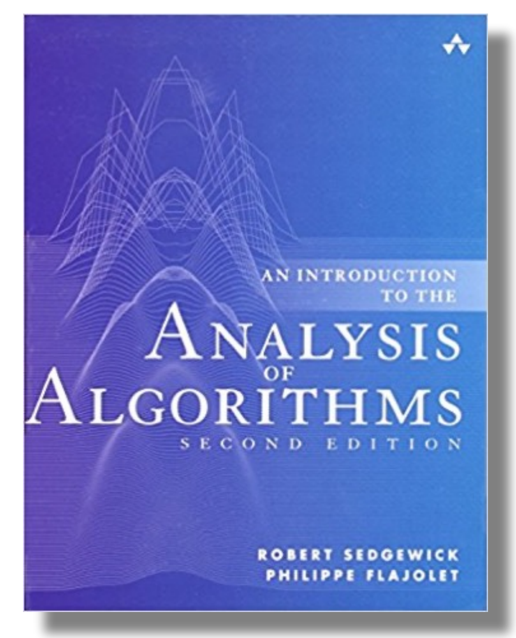
The slides are designed to be self-contained.

They were prepared using the following books that I recommend if you want to learn more:









Next

Assignment this week: variations on quicksort

Next tutorial: a Monte-Carlo algorithm for min-cut

Next week: Vacation time!

Feedback

Moodle for the slides, TDs and exercises.

Questions or comments: Bruno.Salvy@inria.fr