Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

Algorithm Design

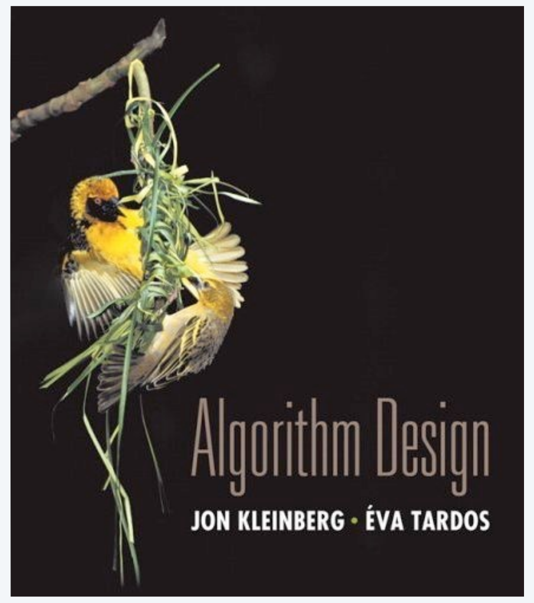
- *analysis of algorithms*
- *greedy algorithms*
- *poly-time reductions*
- *dynamic programming*
- *divide-and-conquer*
- *randomized algorithms*

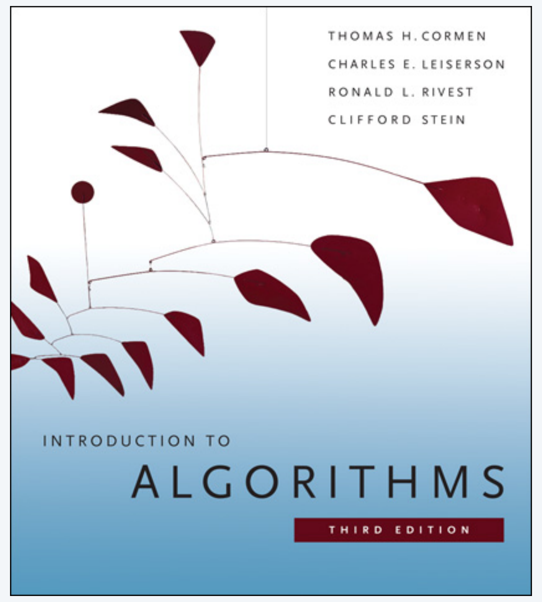
Last updated on 12/3/25 9:27PM

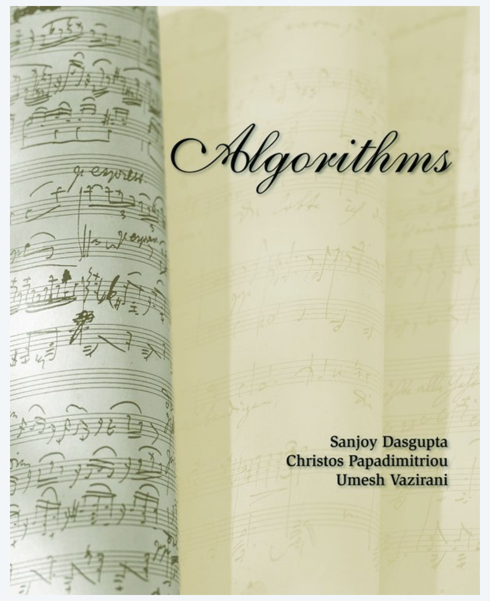
Algorithm design paradigms

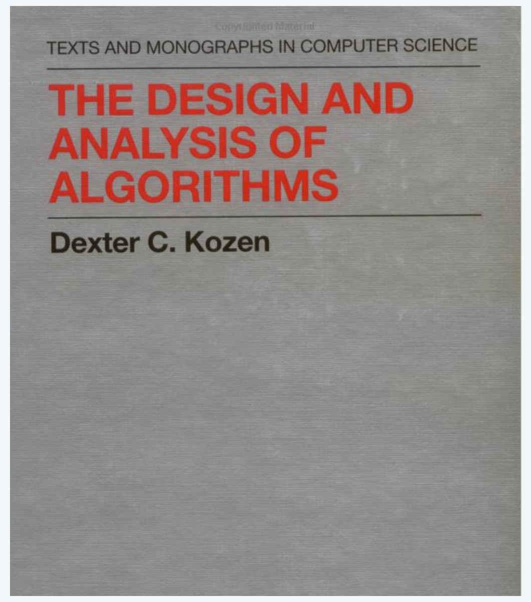
High-level strategies for constructing algorithms.

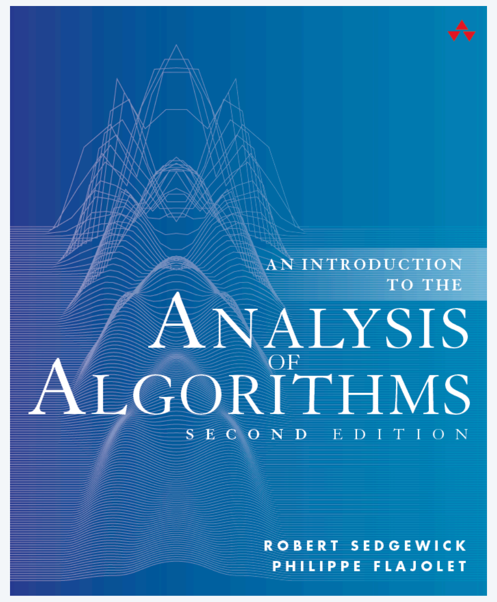
- Analysis of algorithms.
- Greedy algorithms.
- Reductions.
- Dynamic programming.
- Divide-and-conquer.
- Randomized algorithms.

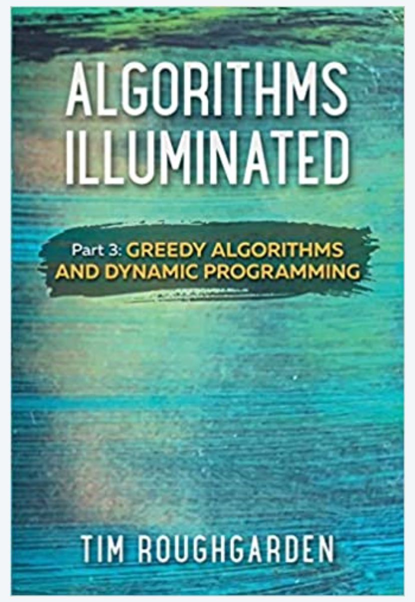







TEXTS AND MONOGRAPHS IN COMPUTER SCIENCE
THE DESIGN AND ANALYSIS OF ALGORITHMS
Dexter C. Kozen





Want more? See COS 240, COS 330, COS 343, COS 423, COS 445, COS 451, MAT 375, MAT 478, …

Interview Questions

Algorithms

Robert Sedgewick | Kevin Wayne

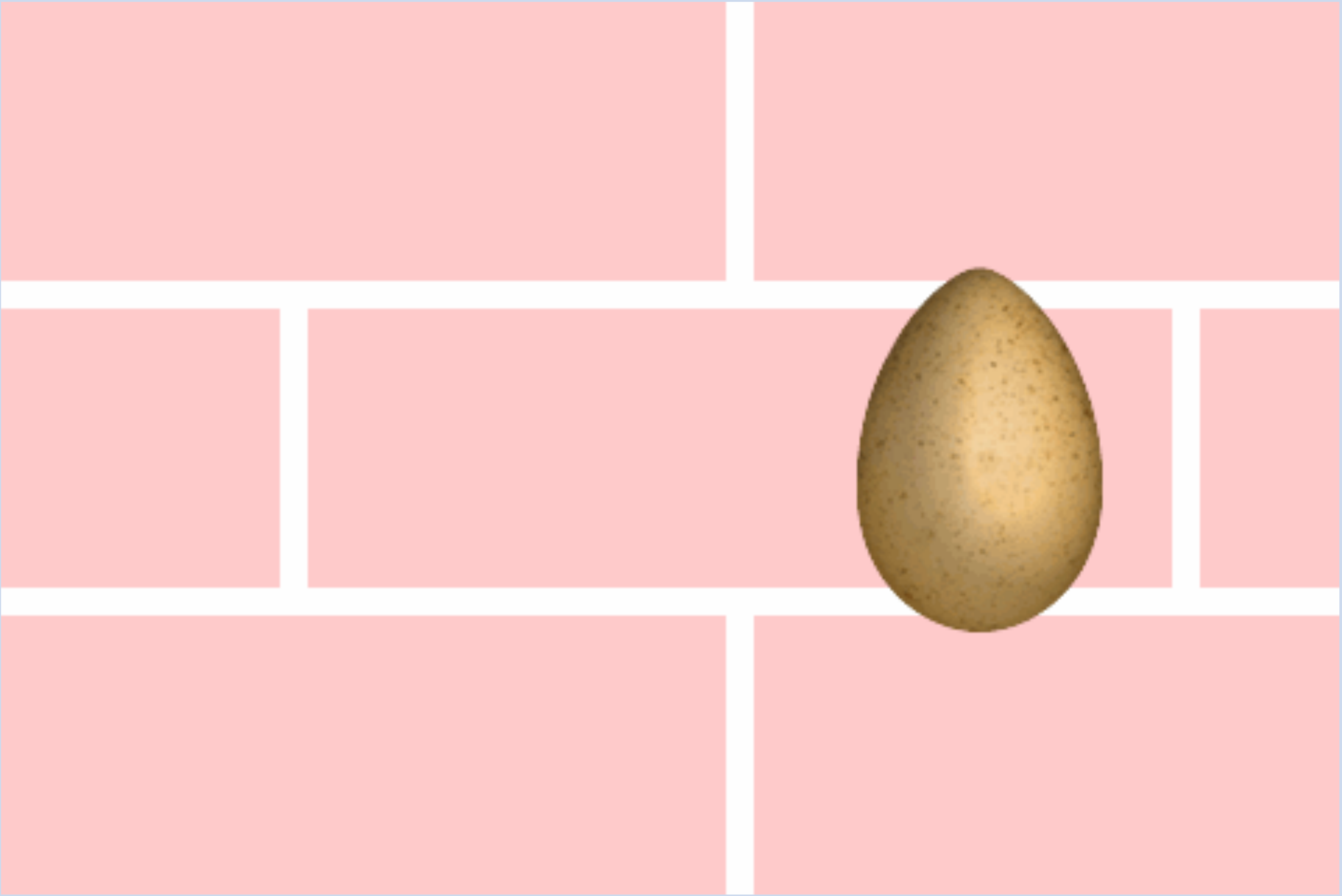
https://algs4.cs.princeton.edu

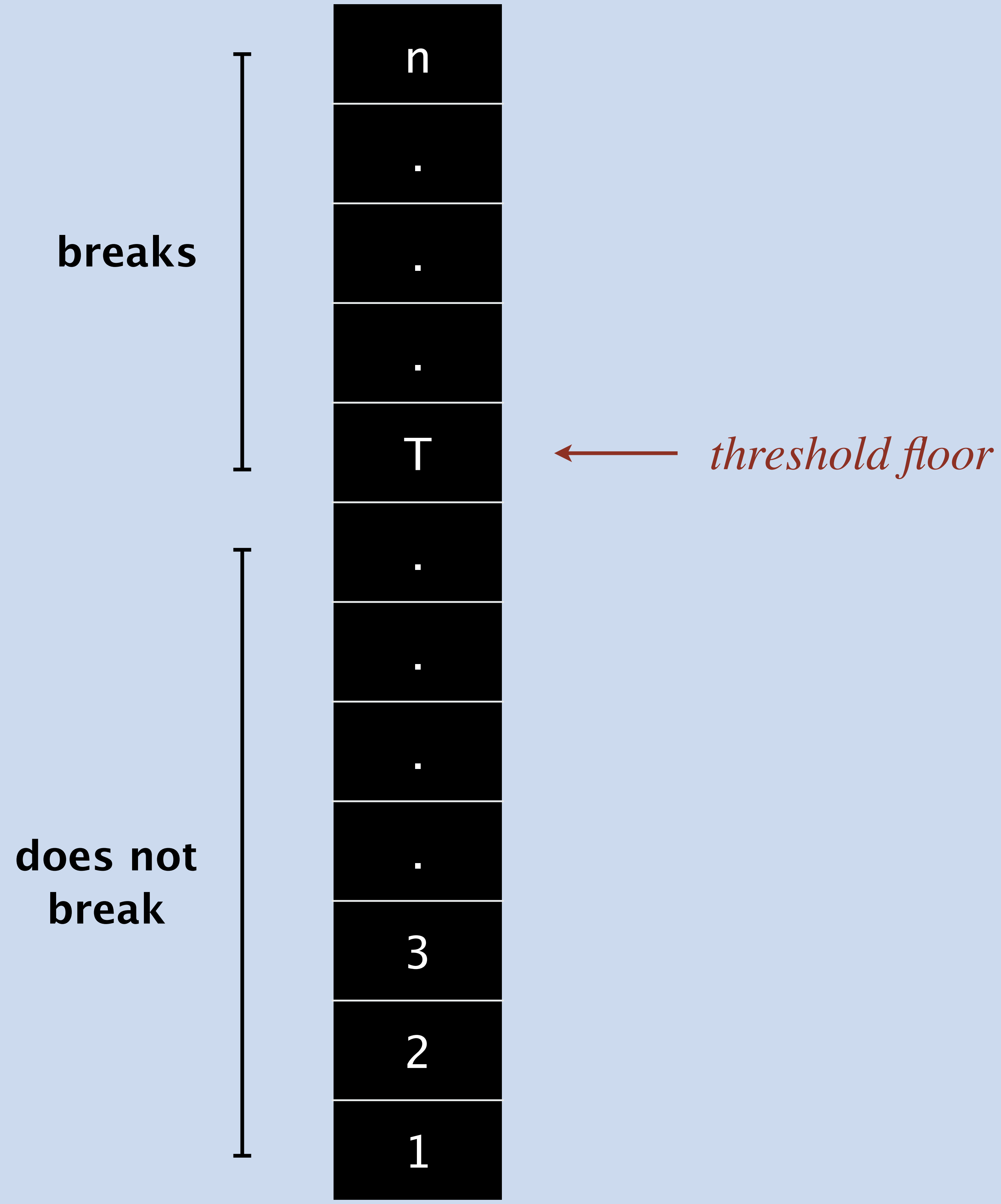
Algorithm Design

- ‣ analysis of algorithms
- ‣ greedy algorithms
- ‣ poly-time reductions
- ‣ dynamic programming
- ‣ divide-and-conquer
- ‣ randomized algorithms

Egg drop

Goal. Find threshold floor T using as few drops as possible.



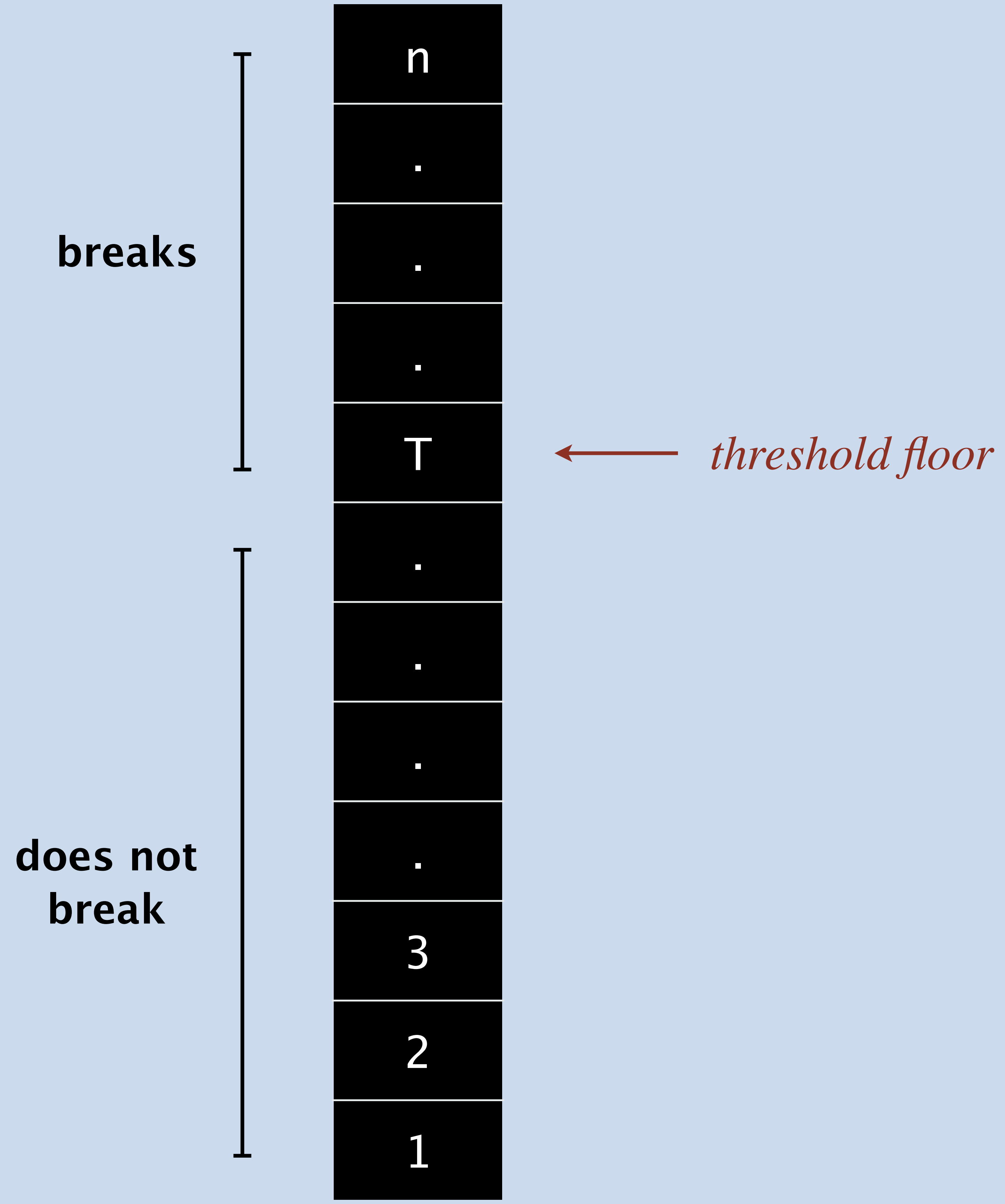


Egg drop

Goal. Find threshold floor T using as few drops as possible.

Rules.

- An egg breaks if dropped from any floor $\geq T$.
- An egg does not break if dropped from any floor $< T$.
- An egg that breaks cannot be reused.
- An egg that does not break can be reused.
- The effect of a drop is the same for all eggs.



EGG DROP

Goal. Find threshold floor T using as few drops as possible.

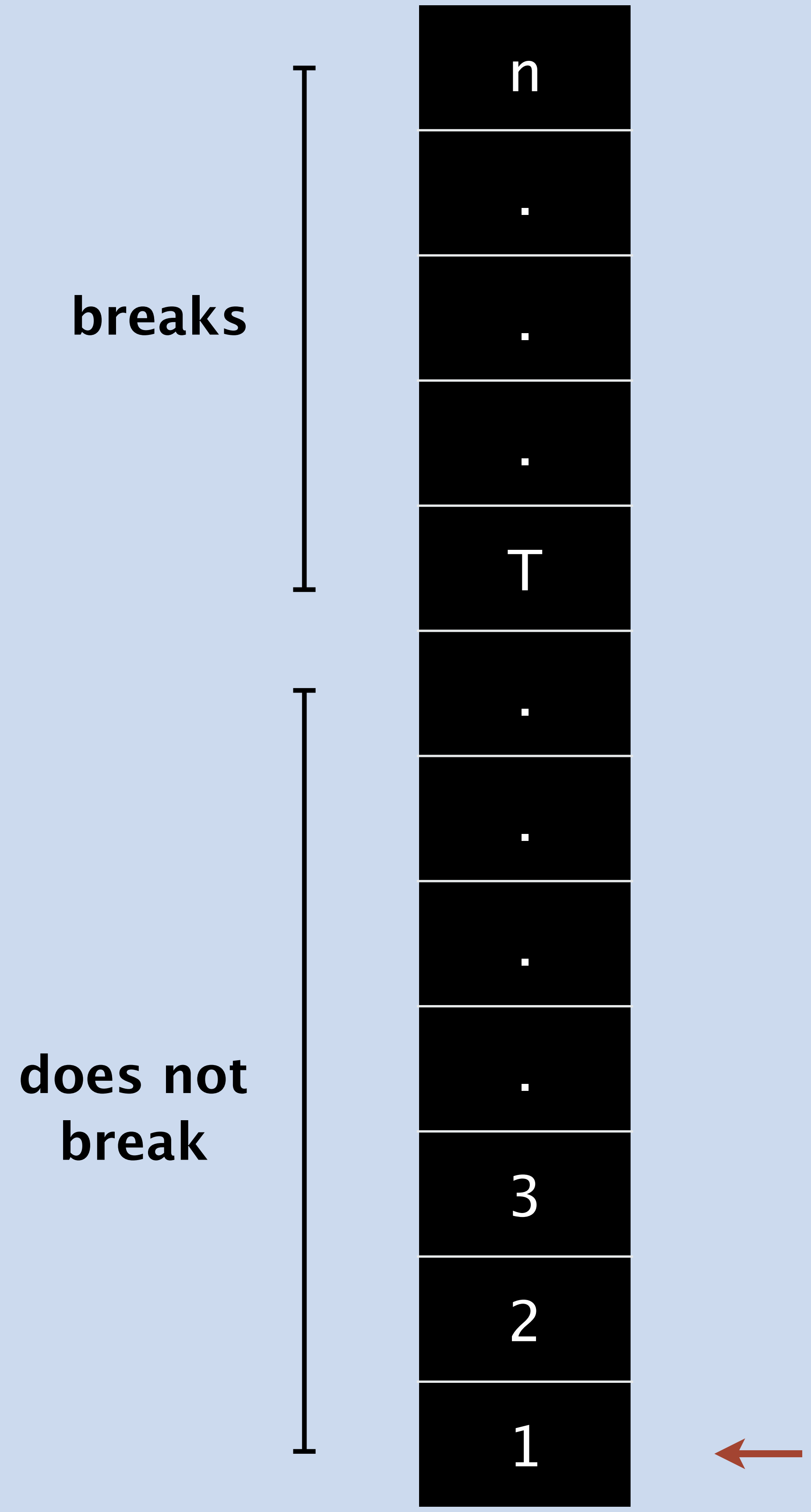
Variant 0. 1 egg.

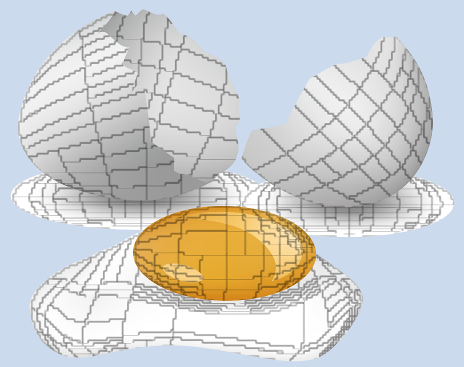
Solution. Use sequential search: drop from floors $1, 2, 3, \ldots$ until the egg breaks.

Analysis. 1 egg and at most n drops.

Analysis. 1 egg and exactly T drops.

# drops depends on a parameter you don't know a priori





Egg drop

Goal. Find threshold floor T using as few drops as possible.

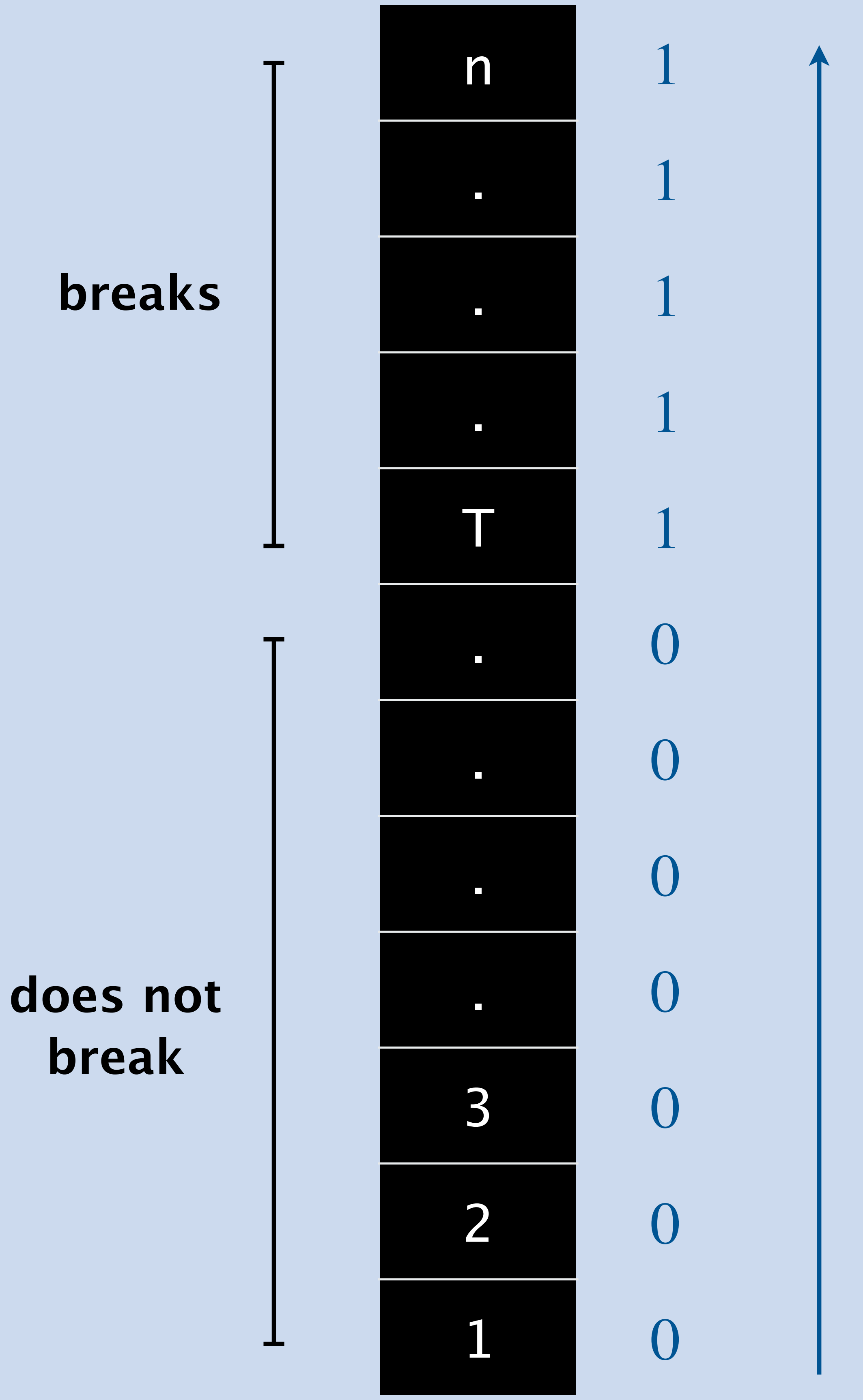
Variant 1. ∞ eggs.

Solution. Binary search for T.

- Initialize $[lo, hi] = [0, n+1]$.
- Loop invariant: egg breaks on floor hi but not on lo.
- Repeat until length of interval is 1:
  - drop from floor $mid = \lfloor (lo + hi) / 2 \rfloor$.
  - if it breaks, update $hi = mid$.
  - otherwise, update $lo = mid$.

Analysis. $\sim \log_2 n$ eggs, $\sim \log_2 n$ drops.

Suppose T is much smaller than n.
Can you guarantee $O(\log T)$ drops?



binary search
to find the first 1
(0 = survive, 1 = break)

Egg drop

Goal. Find threshold floor T using as few drops as possible.

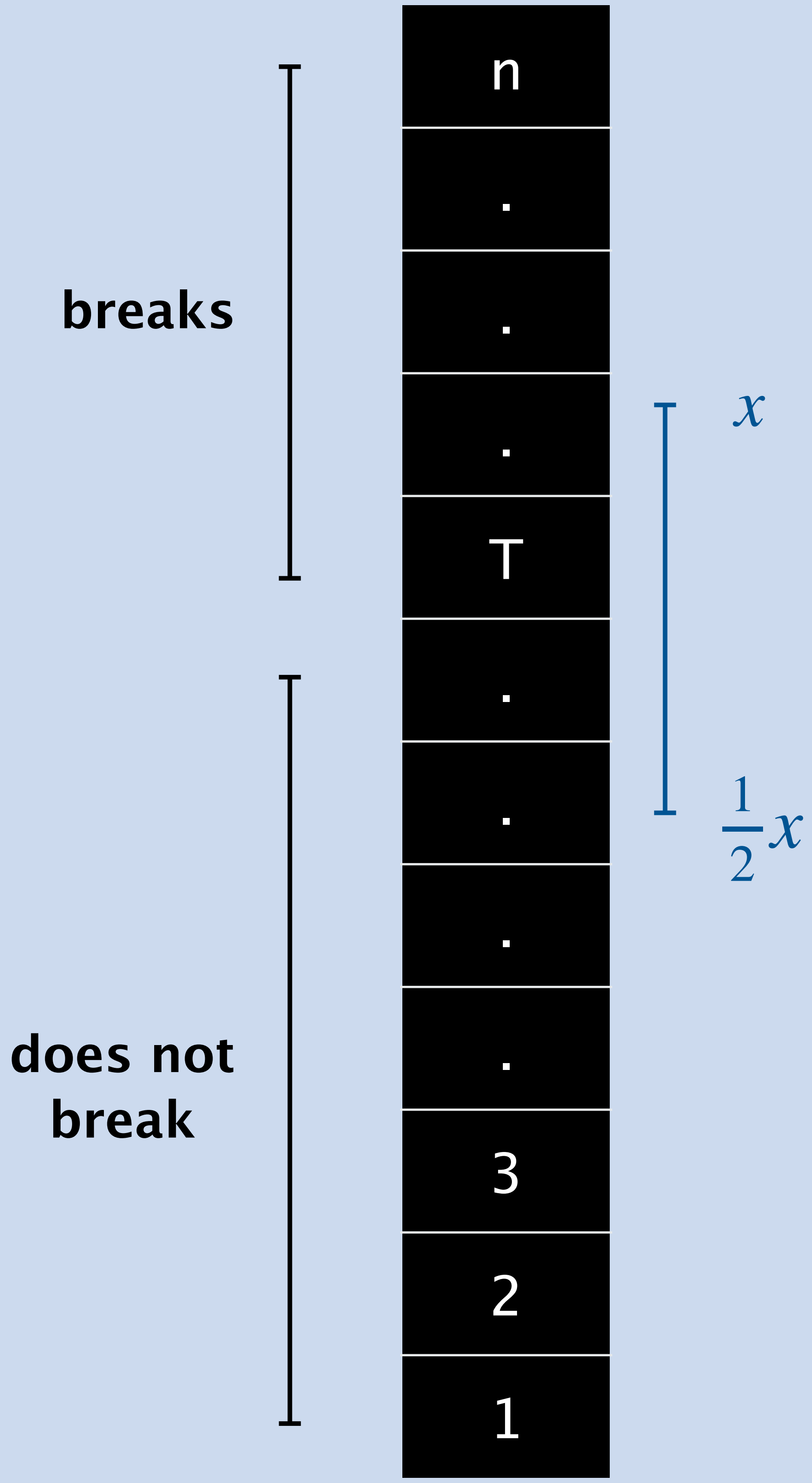
Variant $1'$. ∞ eggs and $O(\log T)$ drops.

Solution. Use repeated doubling, then binary search.

- Drop from floors $1, 2, 4, 8, 16, \ldots, x$ until you find a floor x where the egg breaks from x but does not break from $\frac{1}{2}x$.
- Then, binary search the interval $[\frac{1}{2}x, \ x]$.

Analysis. $\sim \log_2 T$ eggs, $\sim 2 \log_2 T$ drops.

- Repeated doubling: 1 egg and $1 + \log_2 x$ drops.
- Binary search: $\sim \log_2 x$ eggs and $\sim \log_2 x$ drops.
- Total: $\sim \log_2 x$ eggs and $\sim 2 \log_2 x$ drops.
- And because $T \le x < 2T$, the total is $\Theta(\log T)$.



Algorithm design: poll 1

Goal. Find threshold floor T using as few drops as possible.

Variant 2. 2 eggs.

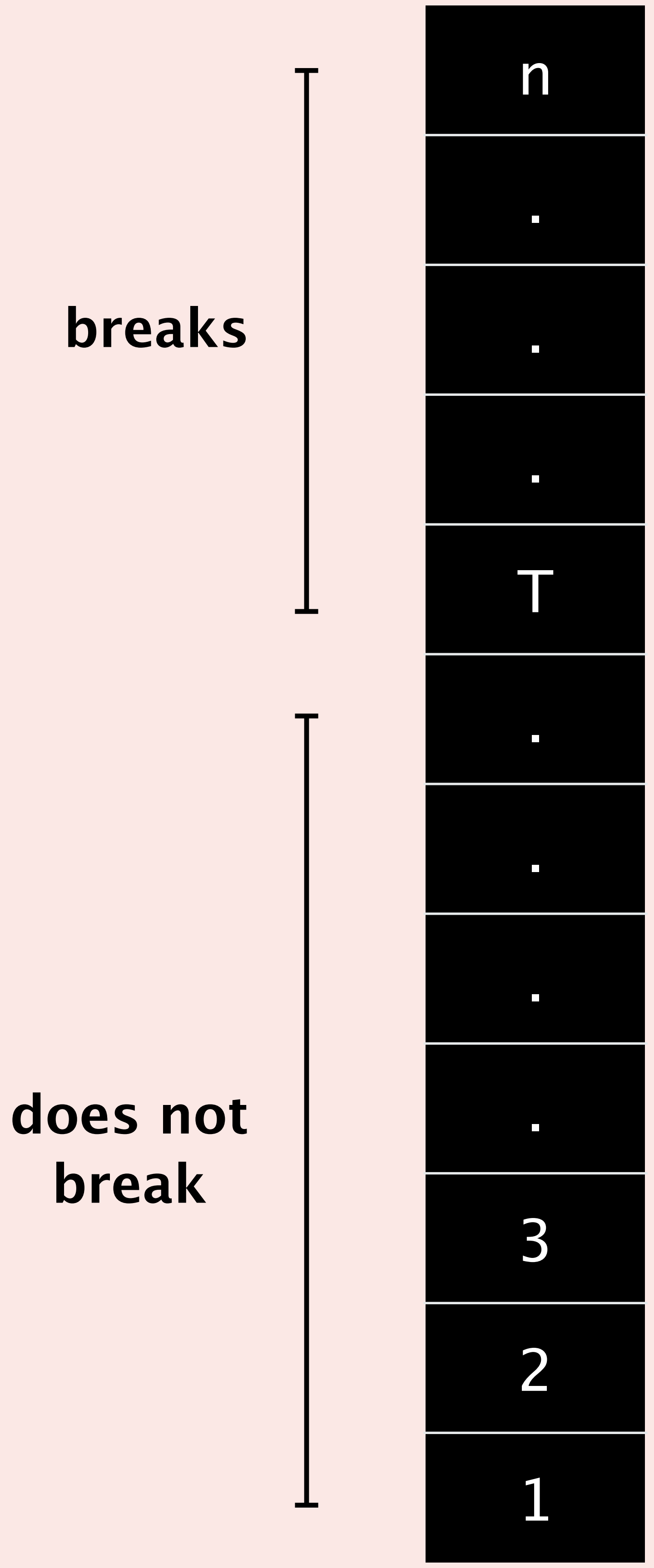
As a function of n, what is the fewest drops that an algorithm can guarantee?

A. $\Theta(1)$

B. $\Theta(\log n)$

C. $\Theta(\sqrt{n})$

D. $\Theta(n)$



Egg drop (asymmetric search)

Goal. Find threshold floor T using as few drops as possible.

Variant 2. 2 eggs.

Solution. Use gridding, then sequential search.

- Drop from floors $\sqrt{n},\ 2\sqrt{n},\ 3\sqrt{n},\ \ldots,\ c\sqrt{n}$ until the first egg breaks.
- Using second egg, sequentially search the interval $[(c-1)\sqrt{n},\ c\sqrt{n}]$.

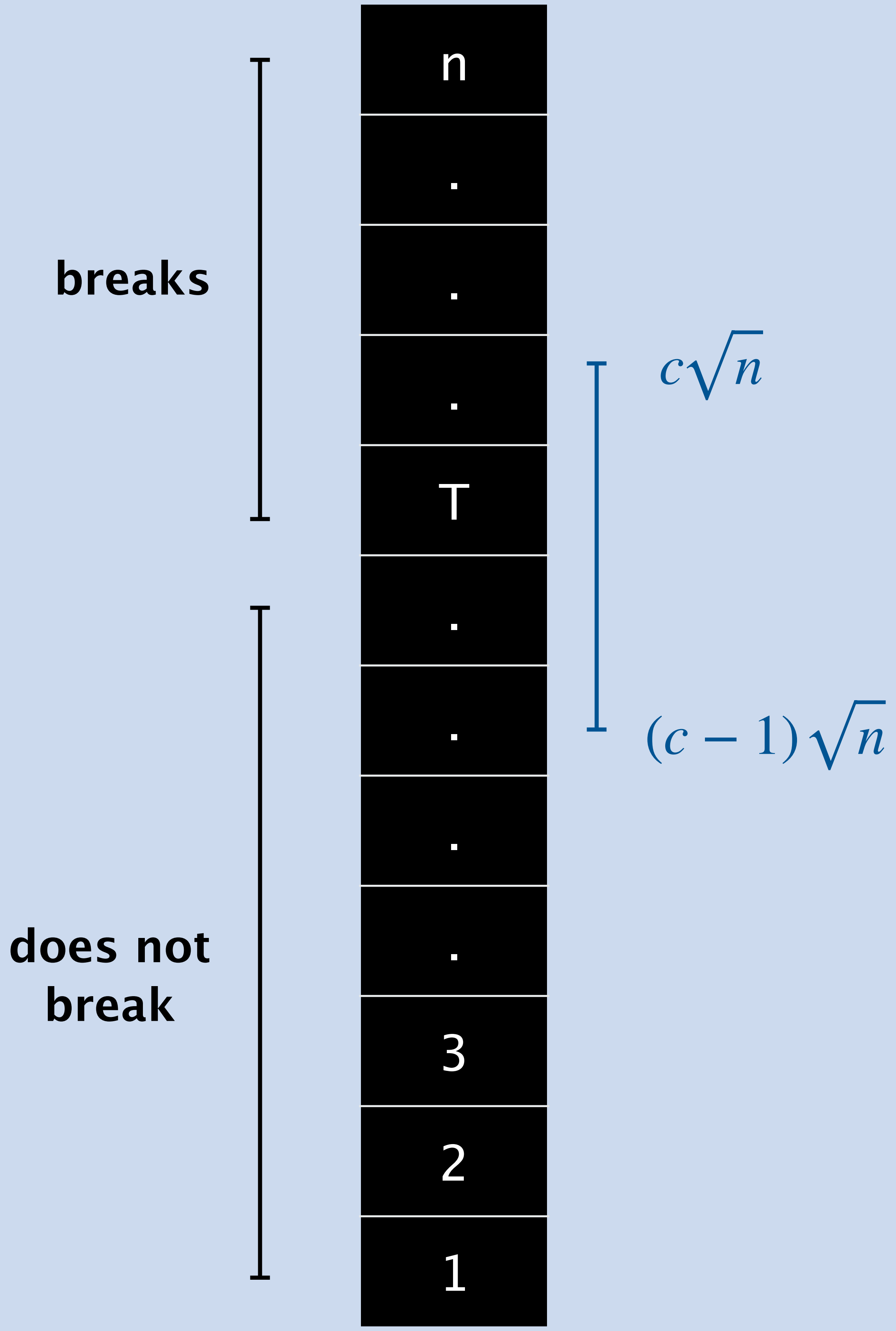
Analysis. Total drops $\le 2\sqrt{n}$.

- First egg: $\le \sqrt{n}$ drops.
- Second egg: $\le \sqrt{n}$ drops.

Signing bonus 1. Use 2 eggs and $\le \sqrt{2n}$ drops.

Signing bonus 2. Use 2 eggs and $O(\sqrt{T})$ drops.

Signing bonus 3. Use 3 eggs and $O(n^{1/3})$ drops.



Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

Algorithm Design

- ▸ *analysis of algorithms*
- ▸ *greedy algorithms*
- ▸ *poly-time reductions*
- ▸ *dynamic programming*
- ▸ *divide-and-conquer*
- ▸ *randomized algorithms*

Greedy algorithms

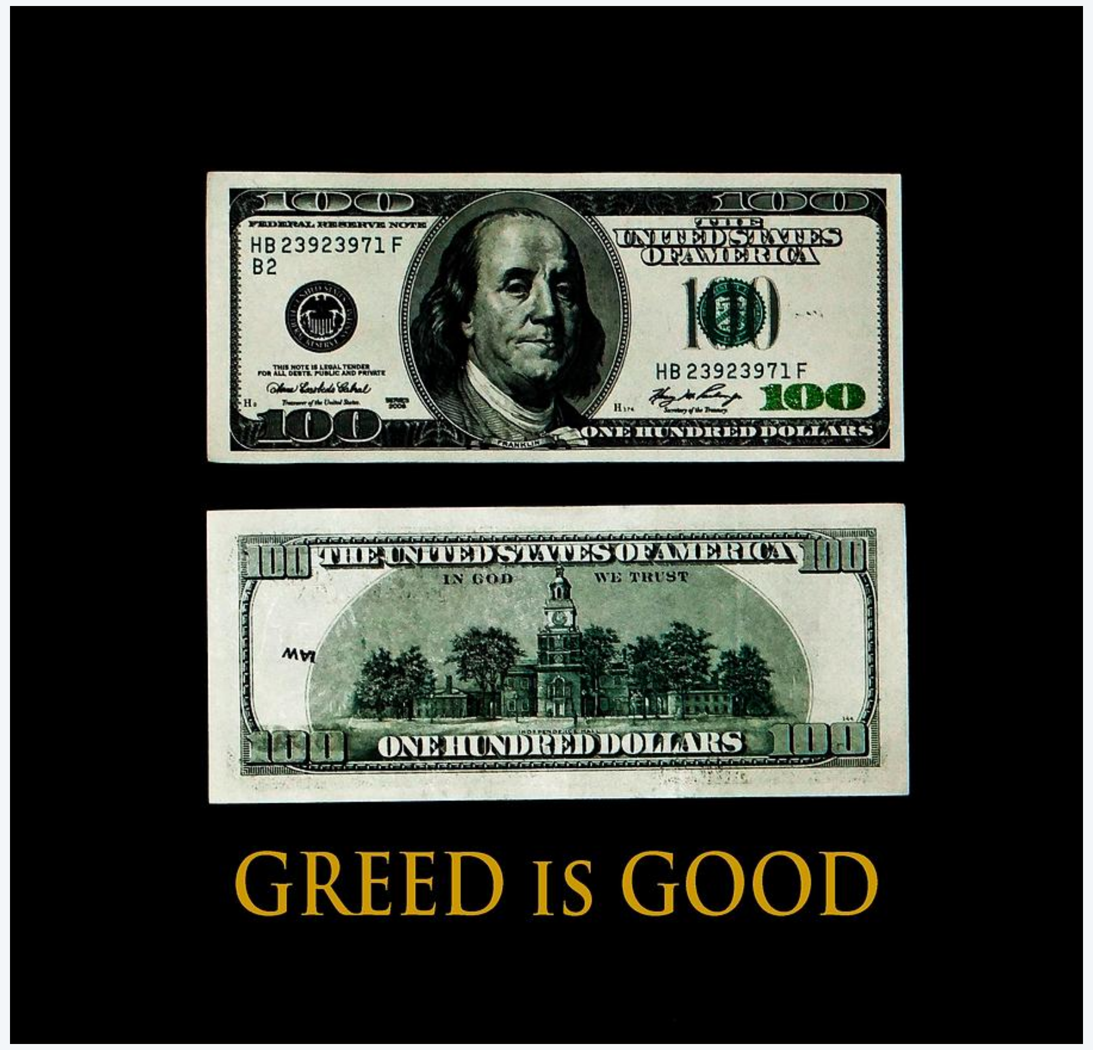
Make locally optimal, irrevocable, choices at each step.

Familiar examples.

- Prim's algorithm. [for MST]
- Kruskal's algorithm. [for MST]
- Dijkstra's algorithm. [for shortest paths]

More classic examples.

- A* search algorithm (artificial intelligence).
- Gale–Shapley algorithm (stable marriage).
- Huffman coding (data compression).
- Greedy basis algorithm (matroids).
- ...



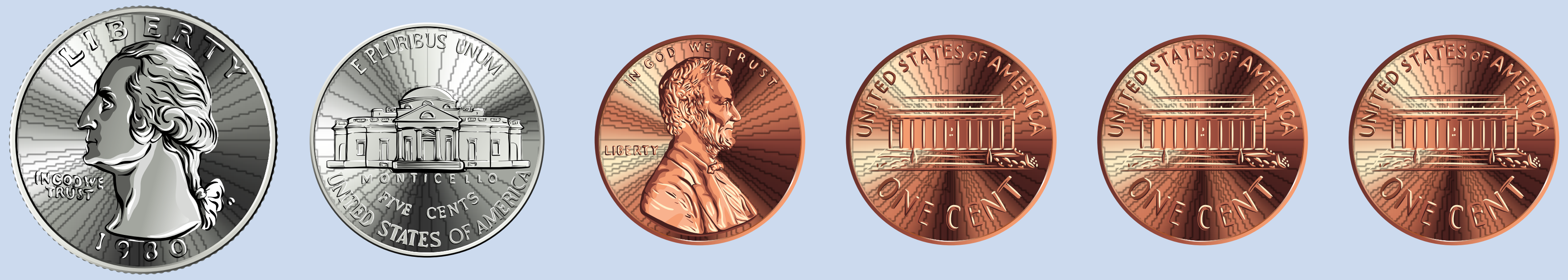
Caveat. Greedy algorithms rarely lead to provably optimal solutions.

[but often used anyway in practice, especially for **NP**-hard optimization problems]

Coin changing problem: greedy algorithm

Goal. Given U. S. coin denominations $\{ 1, 5, 10, 25, 100 \}$, devise a method to make change using fewest coins.

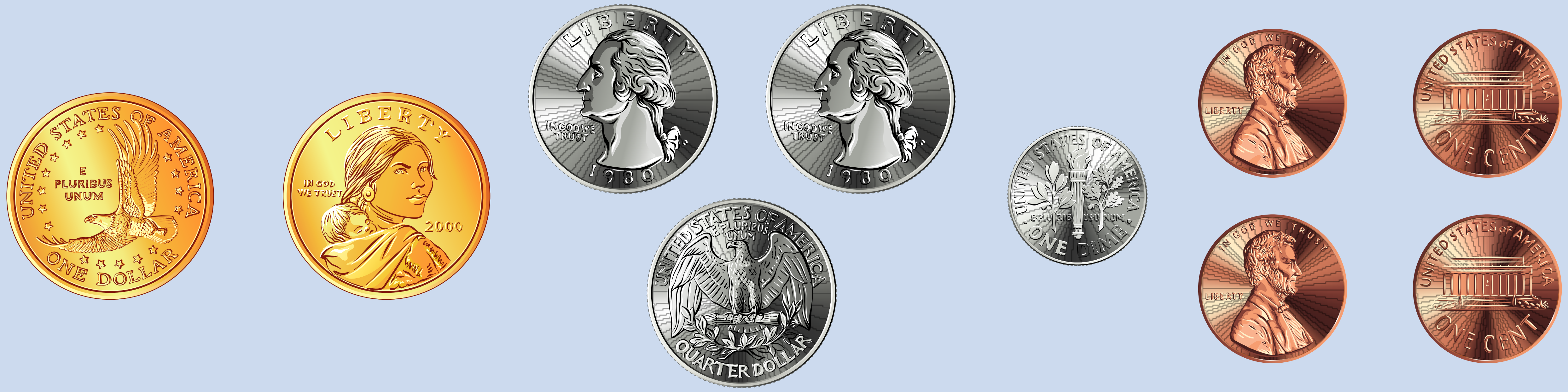
Ex. 34¢.



6 coins

Cashier's (greedy) algorithm. Repeatedly choose the largest coin value that does not exceed the remaining amount.

Ex. $2.89.



10 coins

Algorithm design: poll 2

Is the cashier's algorithm optimal for U.S. coin denominations $\{ 1, 5, 10, 25, 100 \}$?

A. Yes, greedy algorithms are always optimal.

B. Yes, for any set of coin denominations $d_1 < d_2 < \ldots < d_n$ provided $d_1 = 1$.

C. Yes, because of special structural properties of U.S. coin denominations.

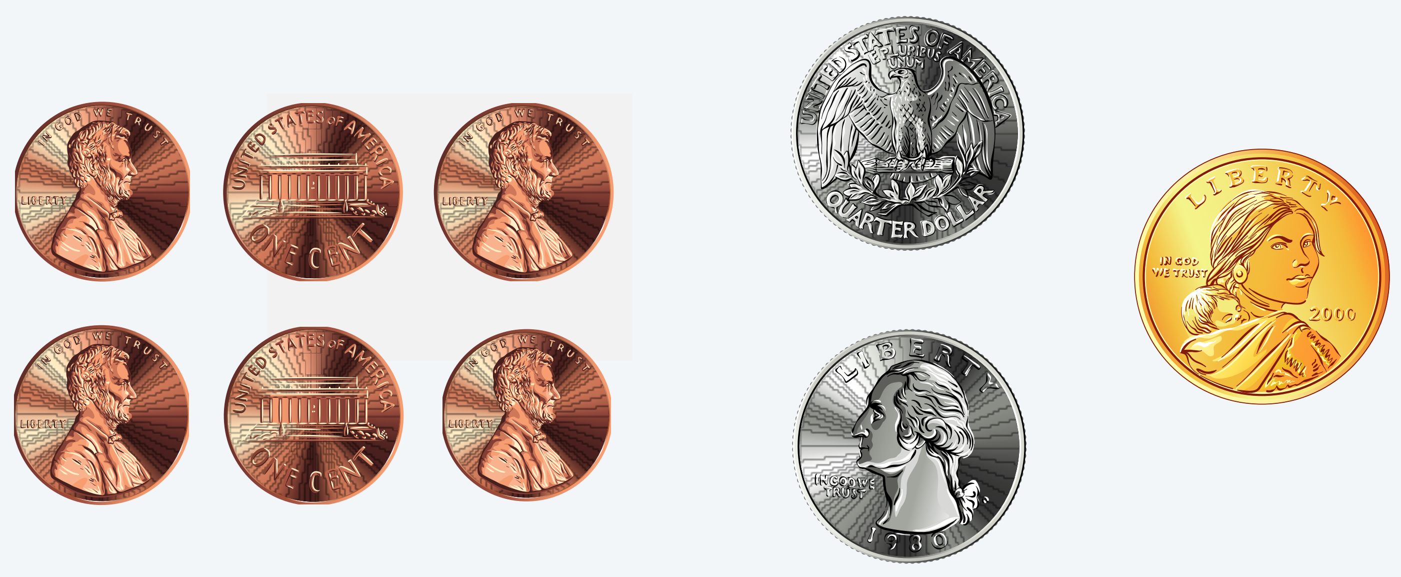
D. No.

Properties of any optimal solution (for U.S. coin denominations)

Property 1. Number of pennies $P \le 4$.

Pf. Replace 5 pennies with 1 nickel.

← *exchange argument*



Property 2. Number of nickels $N \le 1$. ← *replace 2 nickels with 1 dime*

Property 3. Number of dimes $D \le 2$. ← *replace 3 dimes with 1 quarter and 1 nickel*

Property 4. Number of quarters $Q \le 3$. ← *replace 4 quarters with 1 dollar*

Property 5. $N + D \le 2$.

Pf.

- From Properties 2 and 3, $N \le 1$ and $D \le 2$.
- If $N = 1$ and $D = 2$, replace with 1 quarter.

Property 6. $$\overbrace{\underbrace{P}_{\text{P1} \implies \textit{contributes at most } 4} + \underbrace{5N + 10D}_{\text{P5} \implies \textit{contributes at most } 20} + \underbrace{25Q}_{\text{P4} \implies \textit{contributes at most } 75}}^{\textit{significance: total amount of change from pennies, nickels, dimes, and quarters}} \le 99.$$

Optimality of cashier's algorithm (for U.S. coin denominations)

Proposition. Cashier's algorithm yields the unique optimal solution for denominations $\{ 1, 5, 10, 25, 100 \}$.

Pf. [for dollar coins]

- Suppose that we are making change for $\$x.yz$.
- Cashier's algorithm uses x dollar coins.
- Suppose (for the sake of contradiction) that an optimal solution uses fewer than x dollar coins.
- Then, the remaining amount ($\geq 100¢$) must be made using only pennies, nickels, dimes, and quarters, so $P + 5N + 10D + 25Q \ \geq \ 100$.
- But Property 6 says $P + 5N + 10D + 25Q \ \leq \ 99$, a contradiction.

[similar arguments justify greedy strategy for quarters, dimes, and nickels]

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

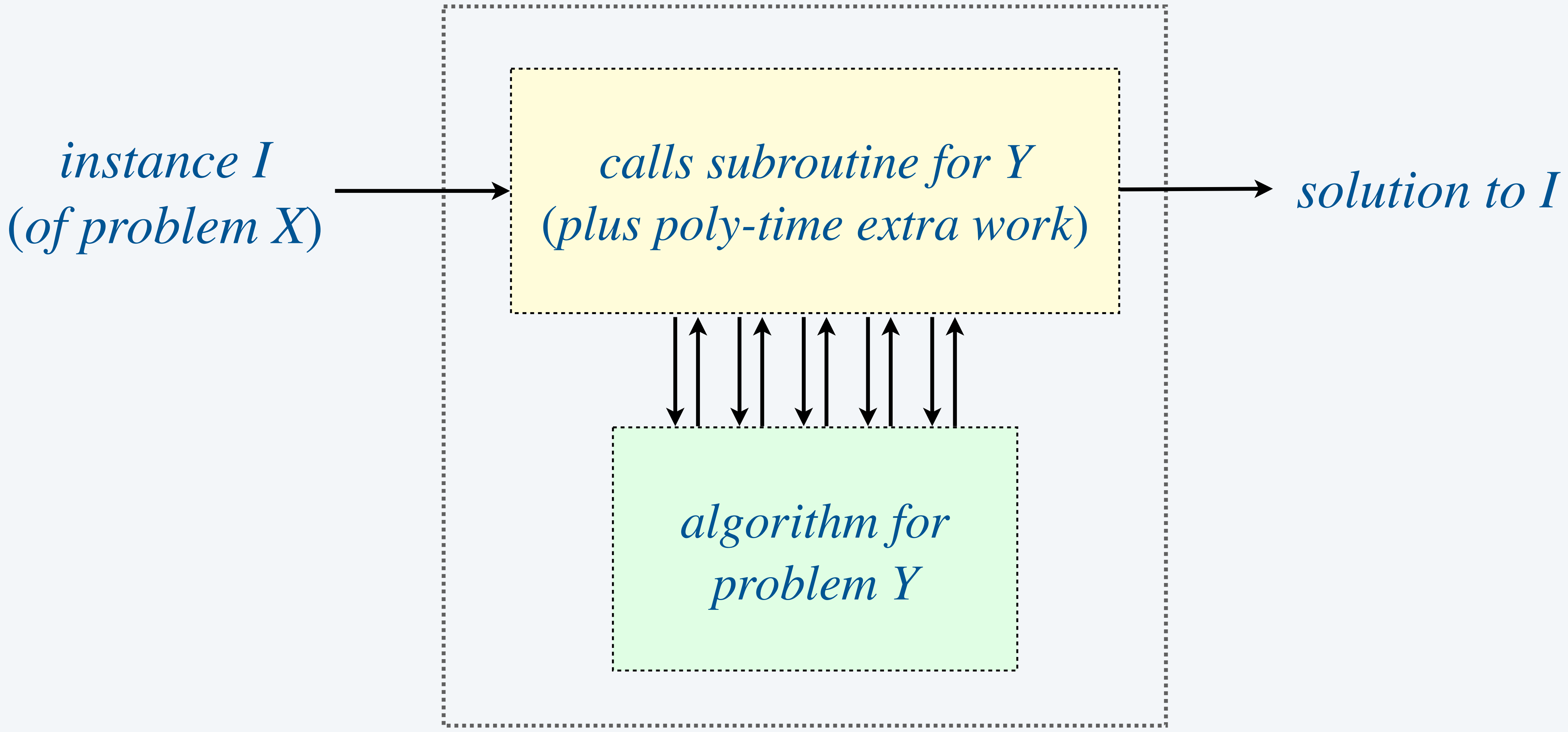
Algorithm Design

- ‣ *analysis of algorithms*
- ‣ *greedy algorithms*
- ‣ *poly-time reductions*
- ‣ *dynamic programming*
- ‣ *divide-and-conquer*
- ‣ *randomized algorithms*

Poly-time reductions

Problem X poly-time reduces to problem Y if there is an algorithm for X that

- makes a polynomial number of calls to an algorithm for Y, and
- performs poly-time extra work (besides those calls).



algorithm for problem X

Ex 1. The median-finding problem reduces to sorting.

Ex 2. Bipartite matching reduces to maxflow.

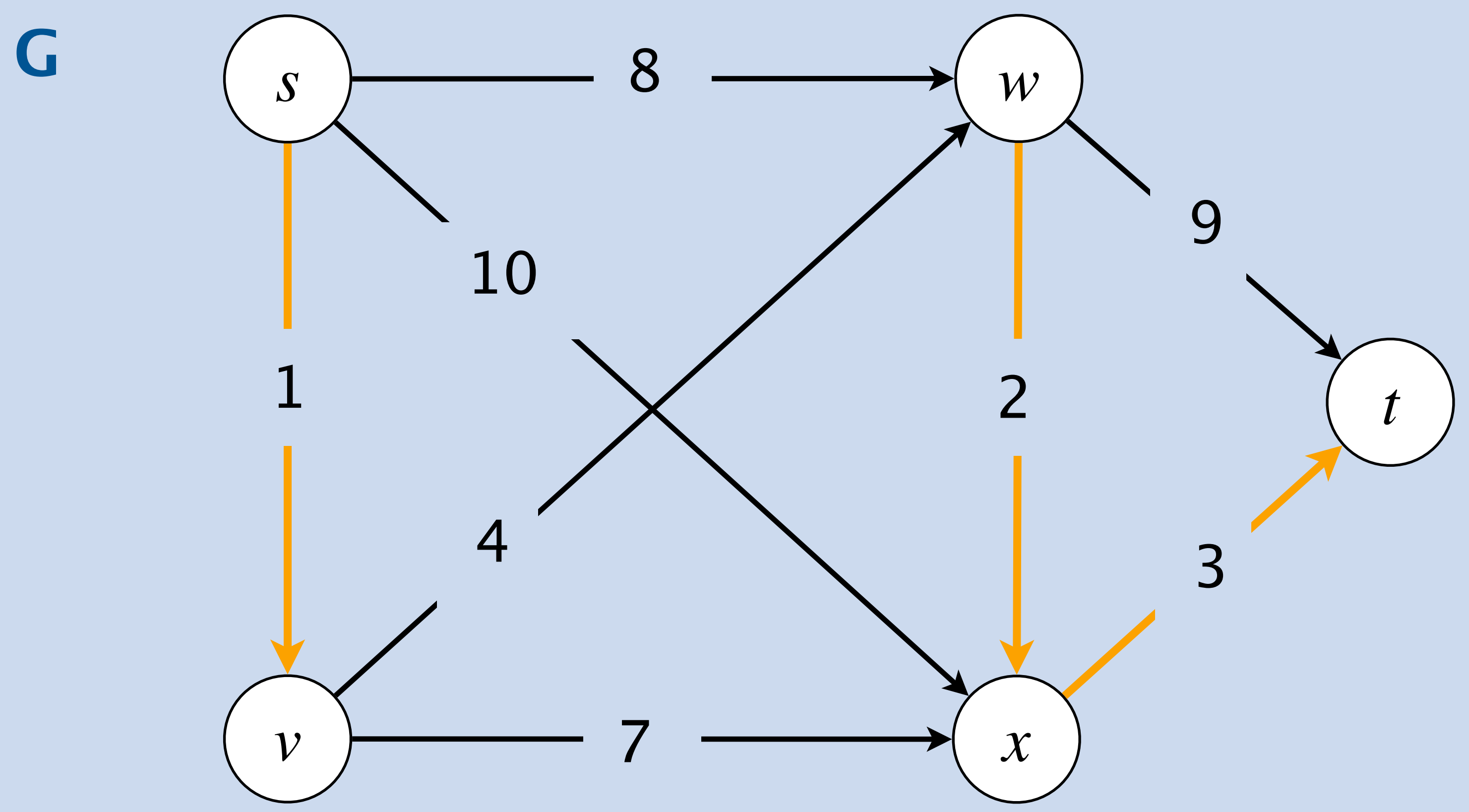
Many, many important problems reduce to:

- Sorting.
- Maxflow.
- Suffix arrays. ← *see* COS 343
- Shortest paths.
- Linear programming. ← *see* ORF 307 *or* ORF 363
- ...

Note. Reductions also play a central role in computational complexity (e.g., **NP**-completeness).

Shortest path with orange and black edges

Goal. Given a digraph with positive edges weights in which each edge is colored orange or black, and an integer k, find a shortest $s \rightsquigarrow t$ path that uses $\leq k$ orange edges.



k = 0: s→w→t (17)

k = 1: s→x→t (13)

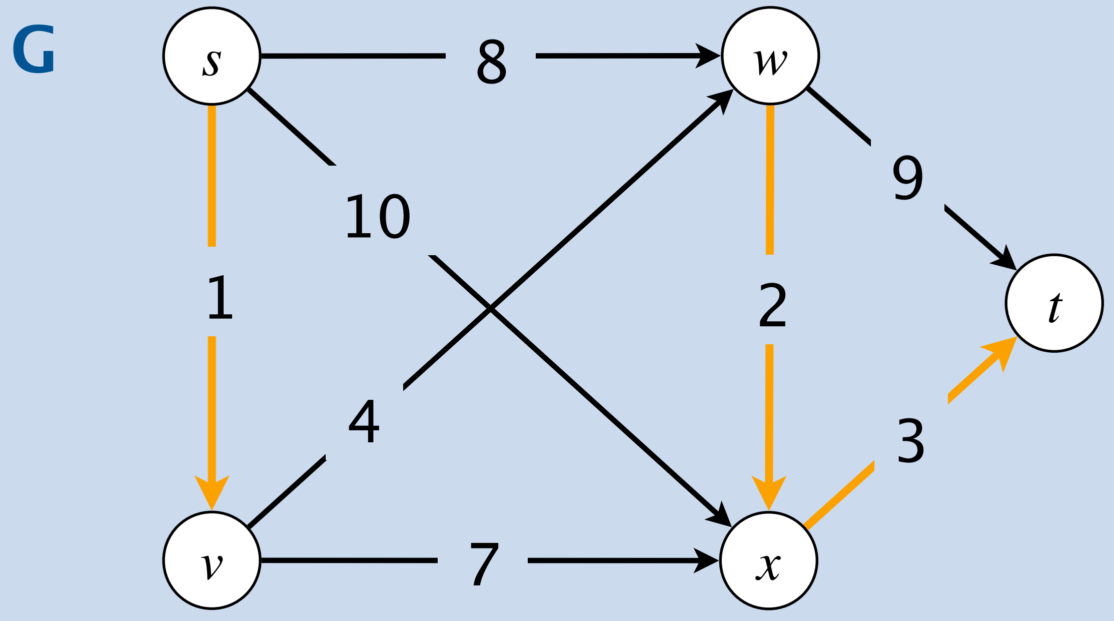
k = 2: s→v→x→t (11)

k = 3: s→v→w→x→t (10)

k = 4: s→v→w→x→t (10)

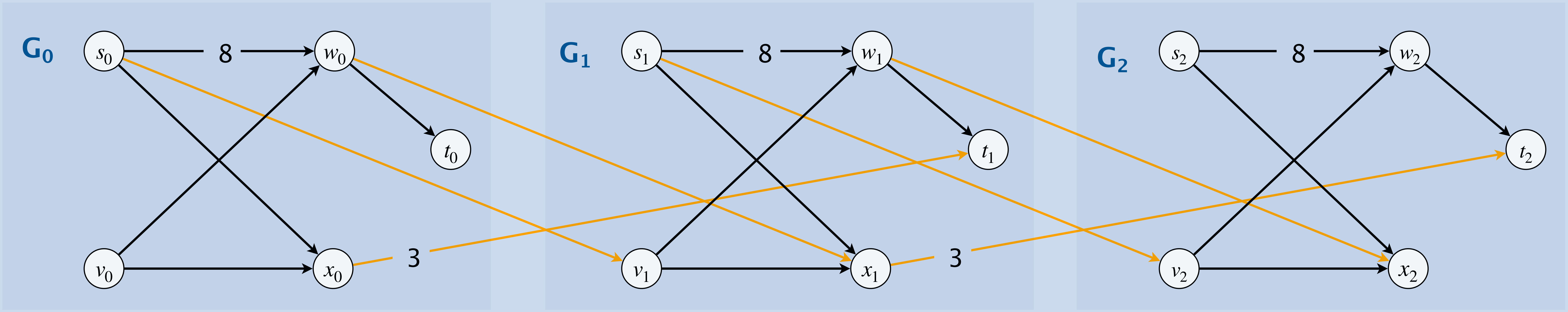
Shortest path with orange and black edges

Goal. Given a digraph with positive edges weights in which each edge is colored orange or black, and an integer k, find a shortest $s \rightsquigarrow t$ path that uses $\leq k$ orange edges.



A poly-time reduction to the single-source shortest paths problem:

- Create $k+1$ copies of the vertices in digraph G, labeled G_0, G_1, …, G_k.
- For each black edge $v \rightarrow w$ in G: add an edge $v_i \rightarrow w_i$ in G_i.
- For each orange edge $v \rightarrow w$ in G: add an edge $v_i \rightarrow w_{i+1}$ from G_i to G_{i+1}.
- Compute shortest paths from s_0 and select the path to the nearest t_i.



k = 2

Algorithm design: poll 3

What is the algorithm's worst-case running time as a function of k, V, and E ?

Assume $E \geq V$ and $k \geq 1$.

A. $\Theta(E \log V)$

B. $\Theta(k E)$

C. $\Theta(k E \log V)$

D. $\Theta(k^2 E \log V)$

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

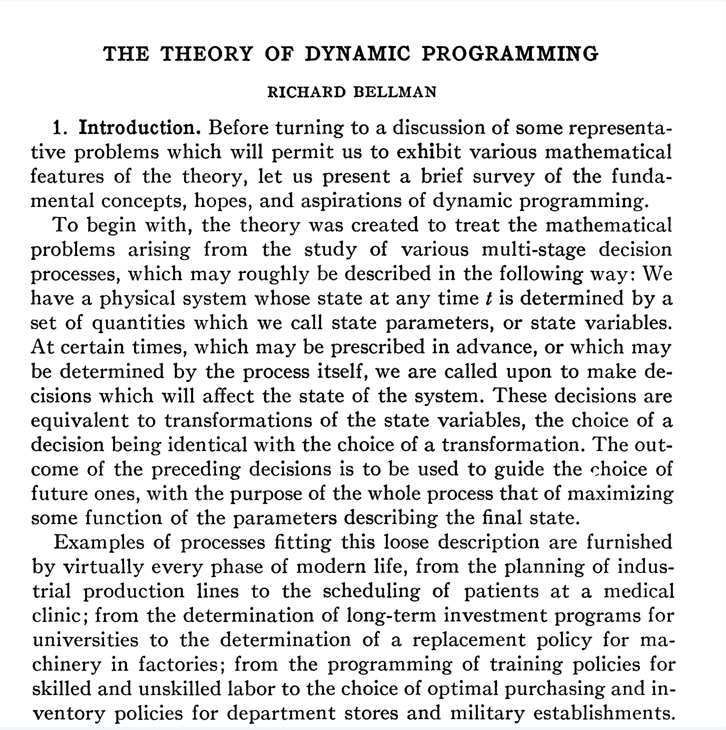
Algorithm Design

- analysis of algorithms
- greedy algorithms
- poly-time reductions
- dynamic programming
- divide-and-conquer
- randomized algorithms

Dynamic programming

- Decompose a complex problem into simpler, overlapping subproblems.
- Build up solutions to progressively larger subproblems.

  (caching intermediate results in a table for efficient reuse)



THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state parameters, or state variables. At certain times, which may be prescribed in advance, or which may be determined by the process itself, we are called upon to make decisions which will affect the state of the system. These decisions are equivalent to transformations of the state variables, the choice of a decision being identical with the choice of a transformation. The outcome of the preceding decisions is to be used to guide the choice of future ones, with the purpose of the whole process that of maximizing some function of the parameters describing the final state.

Examples of processes fitting this loose description are furnished by virtually every phase of modern life, from the planning of industrial production lines to the scheduling of patients at a medical clinic; from the determination of long-term investment programs for universities to the determination of a replacement policy for machinery in factories; from the programming of training policies for skilled and unskilled labor to the choice of optimal purchasing and inventory policies for department stores and military establishments.

Richard Bellman, *46

Familiar examples.

- Bellman-Ford.
- Seam carving.
- Shortest paths in DAGs.

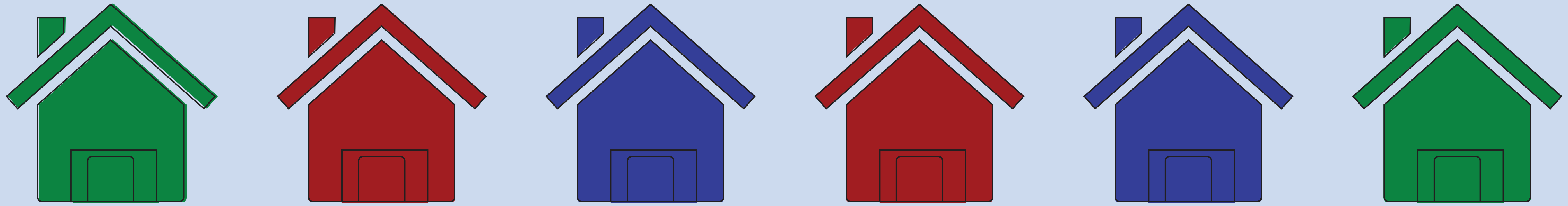
More classic examples.

- Unix diff (file comparison).
- Viterbi (hidden Markov models).
- Cocke-Kasami-Younger (parsing context-free grammars).
- Needleman-Wunsch/Smith-Waterman (DNA sequence alignment).
- ...

House coloring problem

Goal. Paint a row of n houses red, green, or blue so that:

- Total cost is minimized, where $cost(i, color)$ is the cost to paint house i that color.
- No two adjacent houses have the same color.



	1	2	3	4	5	6
$cost(i, red)$	7	6	7	8	9	20
$cost(i, green)$	3	8	9	22	12	8
$cost(i, blue)$	16	10	4	2	5	7

cost to paint house i the given color

(3 + 6 + 4 + 8 + 5 + 8 = 34)

House coloring problem: dynamic programming formulation

Goal. Paint a row of n houses red, green, or blue so that:

- Total cost is minimized, where $cost(i, color)$ is the cost to paint house i that color.
- No two adjacent houses have the same color.

Subproblems.

- $R(i)$ = min cost to paint houses $1, \ldots, i$ with house i red.
- $G(i)$ = min cost to paint houses $1, \ldots, i$ with house i green.
- $B(i)$ = min cost to paint houses $1, \ldots, i$ with house i blue.
- Optimal cost $= \min \{ R(n), G(n), B(n) \}$.

Dynamic programming recurrence.

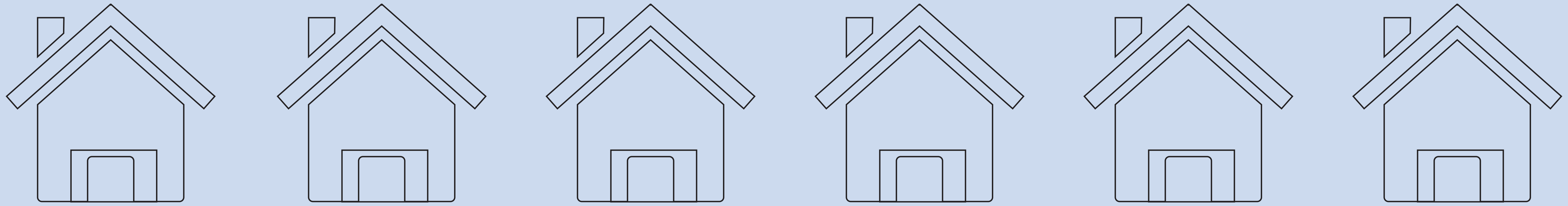
- $R(0) = G(0) = B(0) = 0$
- $R(i) = cost(i, red) + \min \{ G(i-1), B(i-1) \}$
- $G(i) = cost(i, green) + \min \{ B(i-1), R(i-1) \}$
- $B(i) = cost(i, blue) + \min \{ R(i-1), G(i-1) \}$

"optimal substructure"
(optimal solution can be constructed from optimal solutions to smaller subproblems)

HOUSE COLORING: TRACE

Bottom-up DP trace. Given $R(i)$, $G(i)$, and $B(i)$, easy to compute $R(i{+}1)$, $G(i{+}1)$, and $B(i{+}1)$.

$$\begin{aligned} B(6) &= cost(6, blue) + \min\{\, R(5),\ G(5) \,\} \\ &= 7 + \min\{\, 29,\ 32 \,\} \\ &= 36 \end{aligned}$$



	0	1	2	3	4	5	6
$R(i)$	0	7	9	20	21	29	46
$G(i)$	0	3	15	18	35	32	34
$B(i)$	0	16	13	13	20	26	36

cost to paint houses 1, 2, …, i with house i the given color

House coloring: bottom-up implementation

Bottom-up DP implementation.

```
int[] r = new int[n+1];
int[] g = new int[n+1];
int[] b = new int[n+1];

for (int i = 1; i <= n; i++) {
   r[i] = cost[i][RED]   + Math.min(g[i-1], b[i-1]);
   g[i] = cost[i][GREEN] + Math.min(b[i-1], r[i-1]);
   b[i] = cost[i][BLUE]  + Math.min(r[i-1], g[i-1]);
}

return min3(r[n], g[n], b[n]);
```

$$R(i) = cost(i, red) + \min \{ G(i-1),\ B(i-1) \}$$

$$G(i) = cost(i, green) + \min \{ B(i-1),\ R(i-1) \}$$

$$B(i) = cost(i, blue) + \min \{ R(i-1),\ G(i-1) \}$$

Performance. Computes optimal value in $\Theta(n)$ time; uses $\Theta(n)$ extra space.

Remark. Can reconstruct an optimal solution using backtracing.

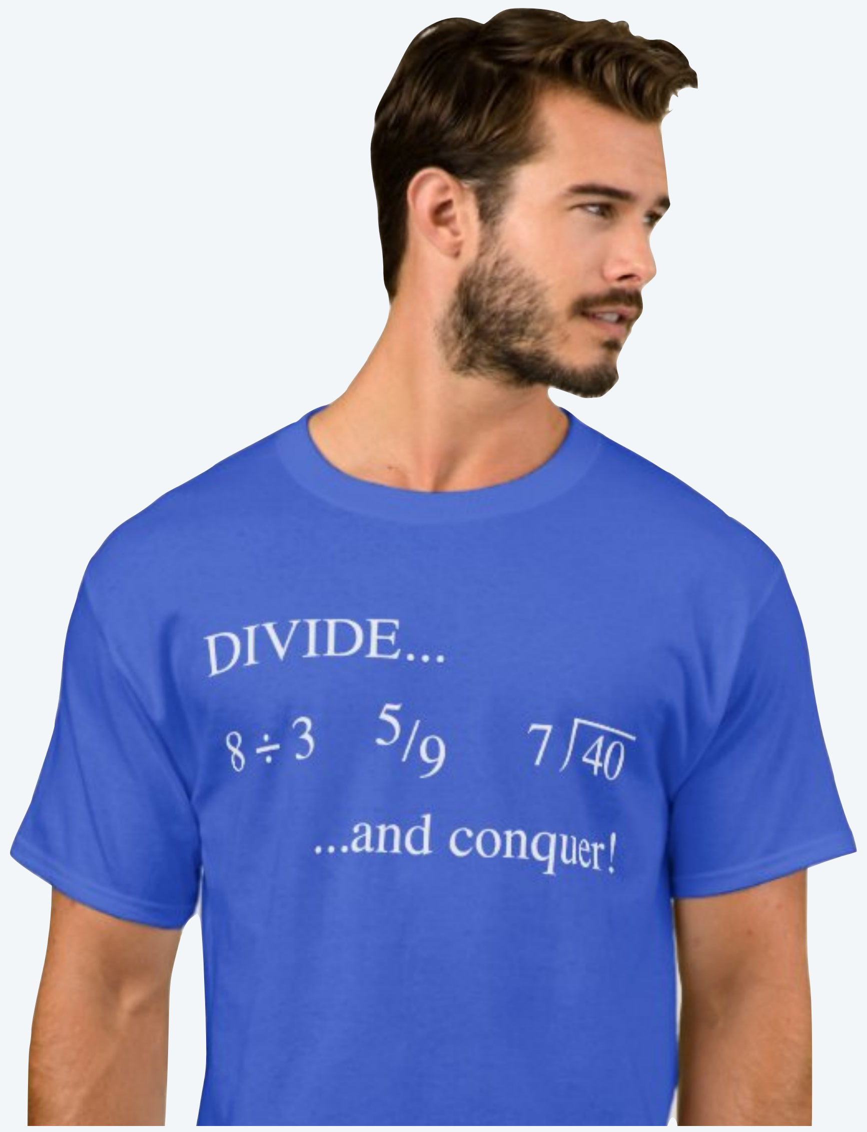
https://algs4.cs.princeton.edu

Algorithm Design

- *analysis of algorithms*
- *greedy algorithms*
- *poly-time reductions*
- *dynamic programming*
- *divide-and-conquer*
- *randomized algorithms*

Divide and conquer

- Break the problem into two or more independent subproblems.
- Solve each subproblem recursively.
- Combine subproblem solutions to solve the original problem.



cosplaying a COS 226 student?

Familiar examples.

- Mergesort.
- Quicksort.

More classic examples.

- Cooley–Tukey FFT (convolution).
- Shamos–Hoey algorithm (closest pair).
- Strassen's algorithm (matrix multiplication).
- Karatsuba's algorithm (integer multiplication).

…

Prototypical usage. Turn a brute-force $\Theta(n^2)$ algorithm into a $\Theta(n \log n)$ algorithm.

Personalized recommendations

Music site tries to match your song preferences with others.

- Your ranking of songs: $0, 1, \ldots, n-1$.
- My ranking of songs: $a_0, a_1, \ldots, a_{n-1}$.
- Music site consults database to find people with similar tastes.

Kendall-tau distance. Number of inversions between two rankings.

Inversion. Songs i and j are inverted if $i < j$, but $a_i > a_j$.

	A	B	C	D	E	F	G	H
you	0	1	2	3	4	5	6	7
me	0	2	3	1	4	5	7	6

3 inversions: 2-1, 3-1, 7-6

Counting inversions

Goal. Given a permutation of length n, count the number of inversions.

0	2	3	1	4	5	7	6

3 inversions: 2–1, 3–1, 7–6

Brute-force algorithm. For each $i < j$ check if $a_i > a_j$.

Running time. Takes $\Theta(n^2)$ time.

A bit better. Run insertion sort; return number of exchanges.

Goal. Algorithm that takes $O(n \log n)$ time.

Counting inversions: divide-and-conquer algorithm

input

0	4	3	7	9	1	5	8	2	6

count inversions in left subarray

0	3	4	7	9	1	5	8	2	6

4–3

1

count inversions in right subarray

0	3	4	7	9	1	2	5	6	8

5–2 8–2 8–6

3

count inversions with one element in each subarray

0	3	4	7	9	1	2	5	6	8

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

13

← *this step seems to require $\Theta(n^2)$ time*

output

0	1	2	3	4	5	6	7	8	9

1 + 3 + 13 = 17

COUNTING INVERSIONS: DIVIDE-AND-CONQUER ALGORITHM

input	0	4	3	7	9	1	5	8	2	6	
count inversions in left subarray and sort	0	3	4	7	9	1	5	8	2	6	1
count inversions in right subarray and sort	0	3	4	7	9	1	2	5	6	8	3
count inversions with one element in each sorted subarray	0	3	4	7	9	1	2	5	6	8	13
and merge into sorted whole	0	1	2	3	4	5	6	7	8	9	

Algorithm design: poll 5

What is running time of algorithm as a function of n ?

A. $\Theta(n)$

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

C. $\Theta(n \log^2 n)$

D. $\Theta(n^2)$

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

https://algs4.cs.princeton.edu

ALGORITHM DESIGN

- ▸ *analysis of algorithms*
- ▸ *greedy algorithms*
- ▸ *poly-time reductions*
- ▸ *dynamic programming*
- ▸ *divide-and-conquer*
- ▸ *randomized algorithms*

Randomized algorithms

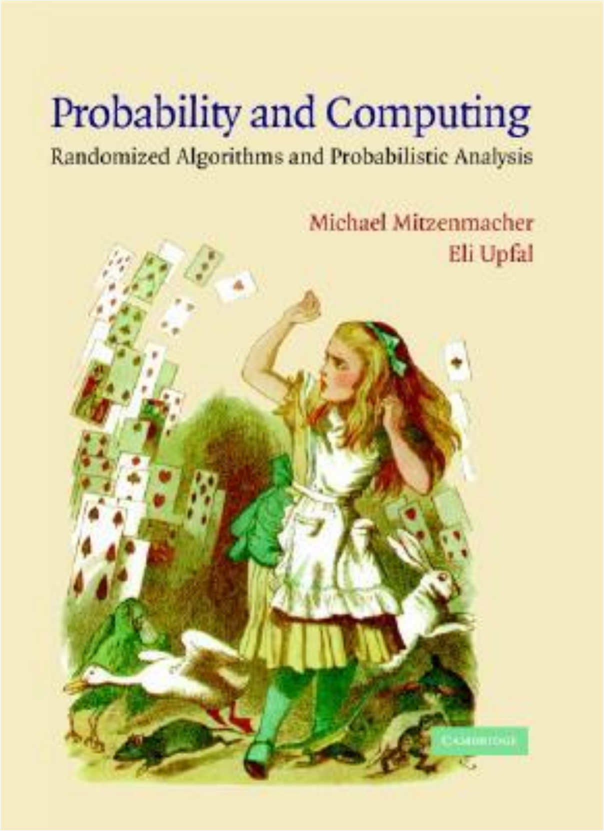
Algorithm whose performance (or output) depends on the results of random coin flips.

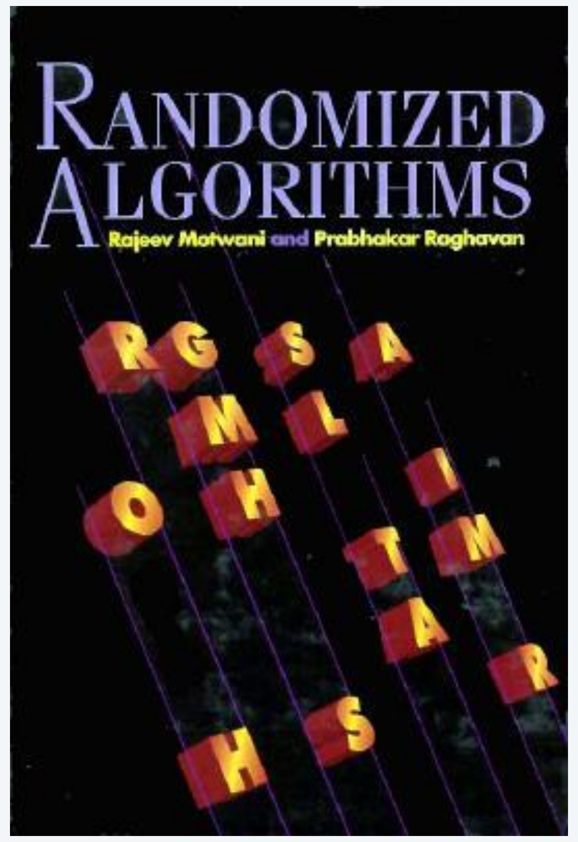
Familiar examples.

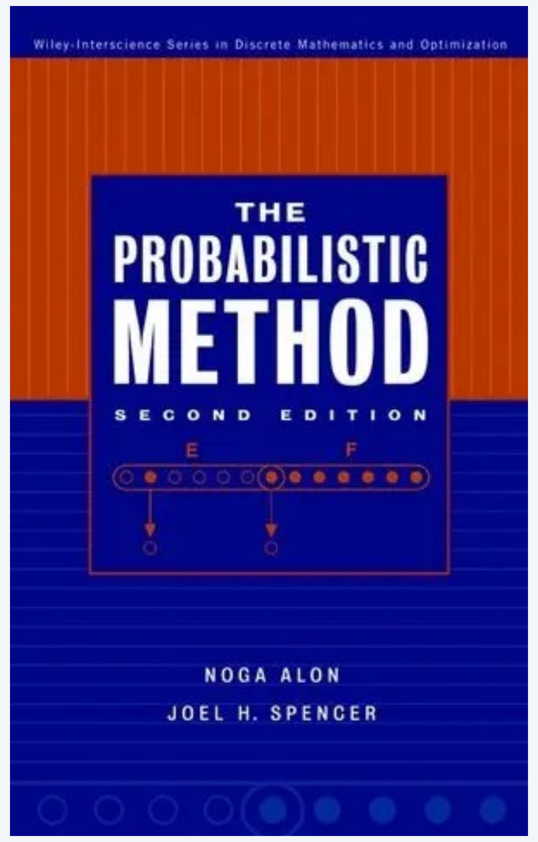
- Quicksort (sorting).
- Quickselect (selection).
- Karger's algorithm (global mincut).

More classic examples.

- Miller-Rabin (primality testing).
- Rabin-Karp (substring search).
- Polynomial identity testing.
- Volume of convex body.
- Universal hashing.
- …


Probability and Computing
Randomized Algorithms and Probabilistic Analysis
Michael Mitzenmacher
Eli Upfal
Cambridge




Wiley-Interscience Series in Discrete Mathematics and Optimization
THE
PROBABILISTIC
METHOD
SECOND EDITION
E
F
NOGA ALON
JOEL H. SPENCER

Nuts and bolts problem

Goal. Given a jumbled pile of n nuts and n bolts, match each nut to its corresponding bolt.

- Each nut fits exactly one bolt; each bolt fits exactly one nut.
- Can compare a nut to a bolt to see which is larger. ←

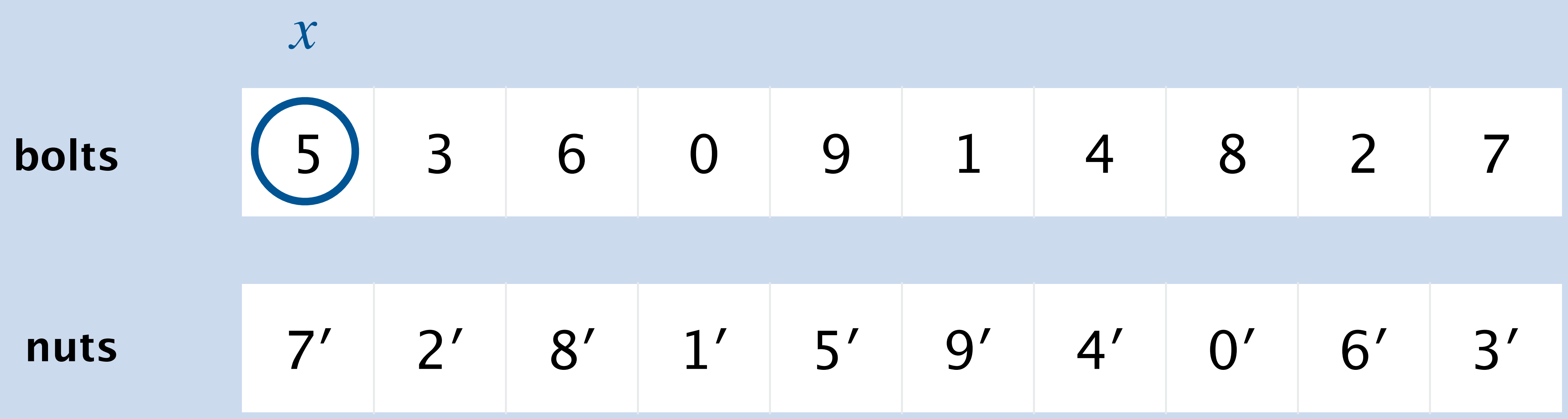
but cannot directly compare two nuts or two bolts

Brute-force algorithm. Compare each bolt to each nut: $\Theta(n^2)$ compares.

Challenge. Design an algorithm that uses only $O(n \log n)$ compares.

Nuts and bolts problem: randomized algorithm

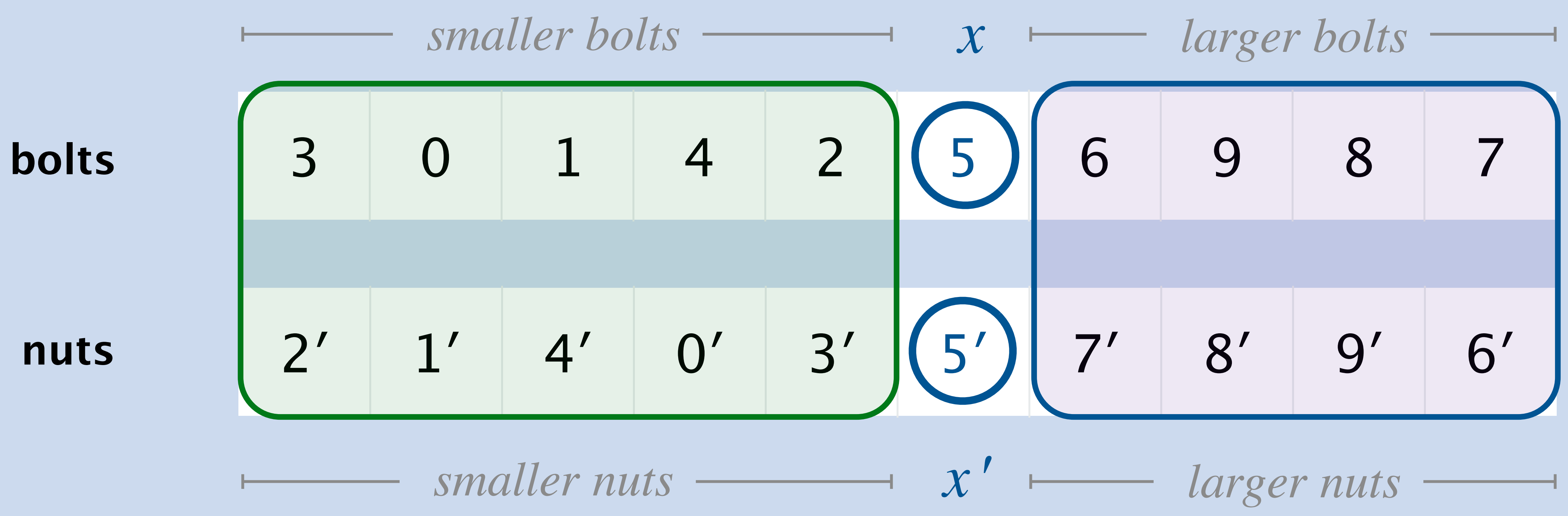
Shuffle. Randomly shuffle the nuts and bolts.



	x									
bolts	5	3	6	0	9	1	4	8	2	7
nuts	7′	2′	8′	1′	5′	9′	4′	0′	6′	3′

Partition.

- Pick the leftmost bolt x. Compare x against all nuts; partition nuts into those smaller than x and those larger than x.
- Let x' be the nut that matches bolt x. Compare x' against all bolts; partition bolts into those smaller than x' and those larger than x'.



	smaller bolts					x	*larger bolts*			
bolts	3	0	1	4	2	5	6	9	8	7
nuts	2′	1′	4′	0′	3′	5′	7′	8′	9′	6′
	smaller nuts					x'	*larger nuts*			

Divide-and-conquer. Recursively solve the two independent subproblems.

Algorithm design: poll 6

What is the expected running time of the randomized algorithm as a function of n ?

A. $\Theta(n)$

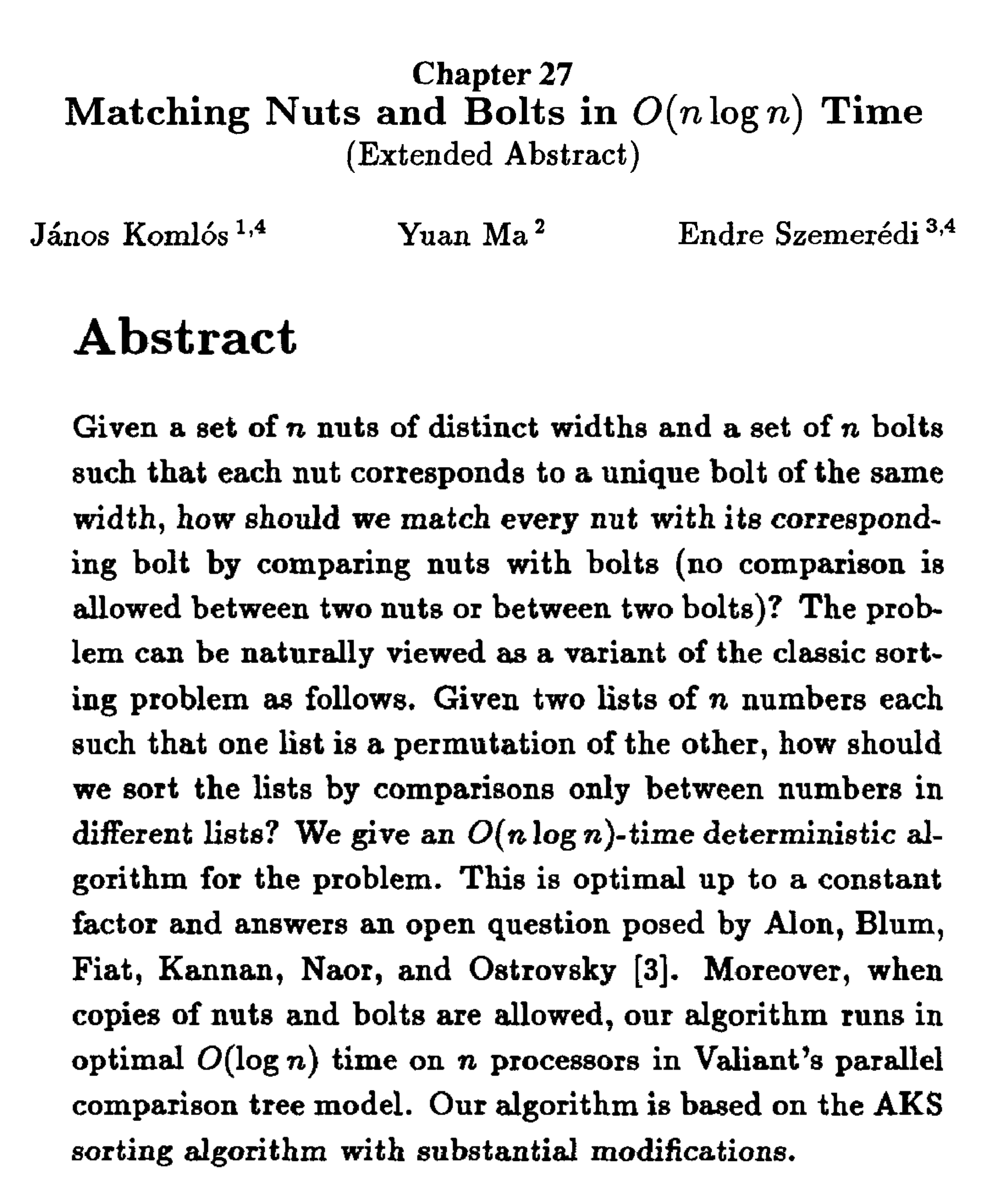
B. $\Theta(n \log n)$

C. $\Theta(n \log^2 n)$

D. $\Theta(n^2)$

Nuts and bolts problem: deterministic algorithm

Hiring bonus. Design an algorithm for the problem that takes $O(n \log n)$ time in the worst case.



Chapter 27

Matching Nuts and Bolts in $O(n \log n)$ Time

(Extended Abstract)

János Komlós[1,4] Yuan Ma[2] Endre Szemerédi[3,4]

Abstract

Given a set of n nuts of distinct widths and a set of n bolts such that each nut corresponds to a unique bolt of the same width, how should we match every nut with its corresponding bolt by comparing nuts with bolts (no comparison is allowed between two nuts or between two bolts)? The problem can be naturally viewed as a variant of the classic sorting problem as follows. Given two lists of n numbers each such that one list is a permutation of the other, how should we sort the lists by comparisons only between numbers in different lists? We give an $O(n \log n)$-*time deterministic algorithm* for the problem. This is optimal up to a constant factor and answers an open question posed by Alon, Blum, Fiat, Kannan, Naor, and Ostrovsky [3]. Moreover, when copies of nuts and bolts are allowed, our algorithm runs in optimal $O(\log n)$ time on n processors in Valiant's parallel comparison tree model. Our algorithm is based on the AKS sorting algorithm with substantial modifications.

Credits

Co-instructors and preceptors.

Prof. Marcel Dall'Agnol

Prof. Maryam Hedayati

Viola Chen

Alkin Kaz

Jiatong Lu

Dexin Zhang

Mingkun Zhao

Undergrad graders and lab TAs. Apply to be one next semester!

A final thought

Credits

media	source	license
Egg Drop	New York Times	
Broken Egg	Adobe Stock	education license
Greed is Good	Dennis Dugan	
Coin Changing	unknown	
U.S. Coins	Adobe Stock	education license
Cash Register	Adobe Stock	education license
Richard Bellman	Wikipedia	
Divide-and-Conquer T-Shirt	Zazzle	
Coin Toss	clipground.com	CC BY 4.0
Nuts and Bolts	Adobe Stock	education license
Crowd Cheering	YouTube	