



<https://algs4.cs.princeton.edu>

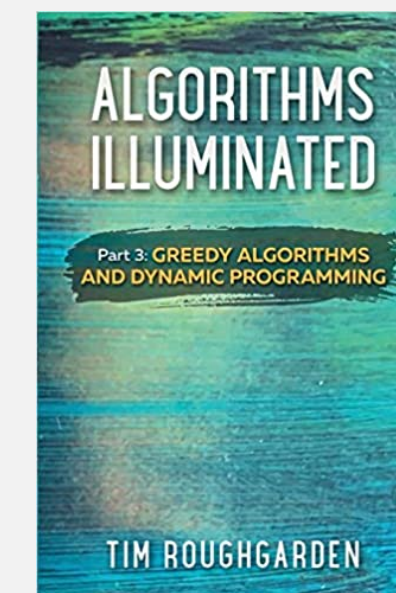
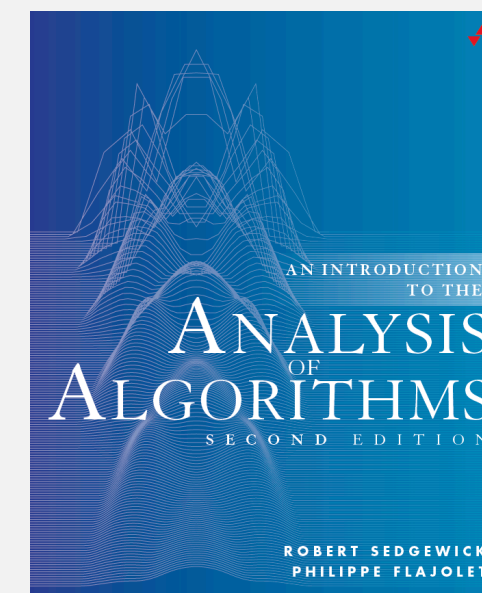
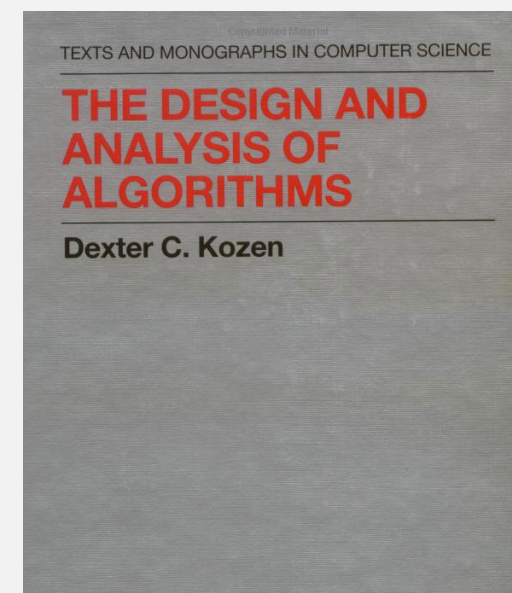
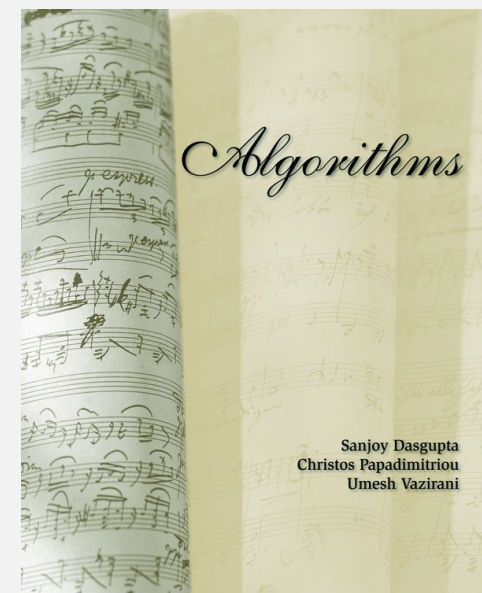
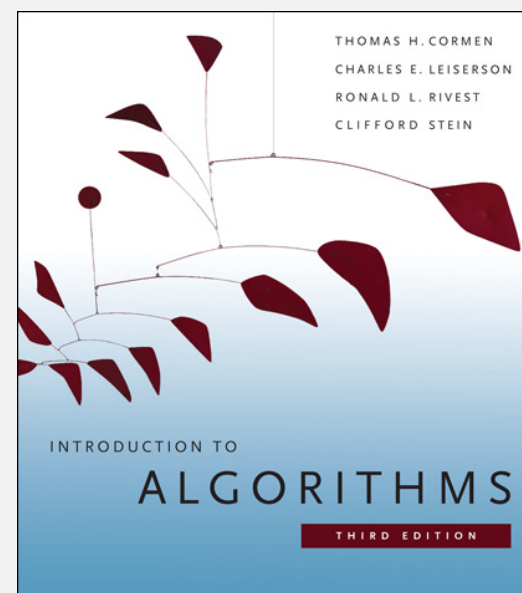
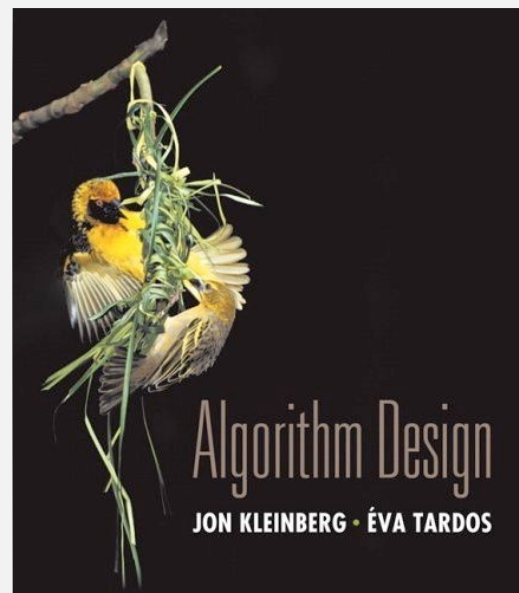
ALGORITHM DESIGN

- *analysis of algorithms*
- *greed*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

Algorithm design

Algorithm design patterns.

- Analysis of algorithms.
- Greed.
- Reduction.
- Dynamic programming.
- Divide-and-conquer.
- Randomization.

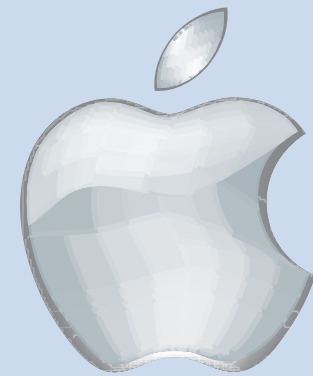


Want more? See COS 240, COS 343, COS 423, COS 445, COS 451, MAT 375,

INTERVIEW QUESTIONS



Google



GitHub



facebook

RSA
SECURITY

airbnb

NETFLIX



slack

CISCO SYSTEMS

Square

in

Morgan Stanley

IBM

lyft



NVIDIA

TESLA

JANE STREET

DE Shaw & Co

SPACEX

Adobe

UBER

zoom

P X A R
ANIMATION STUDIOS

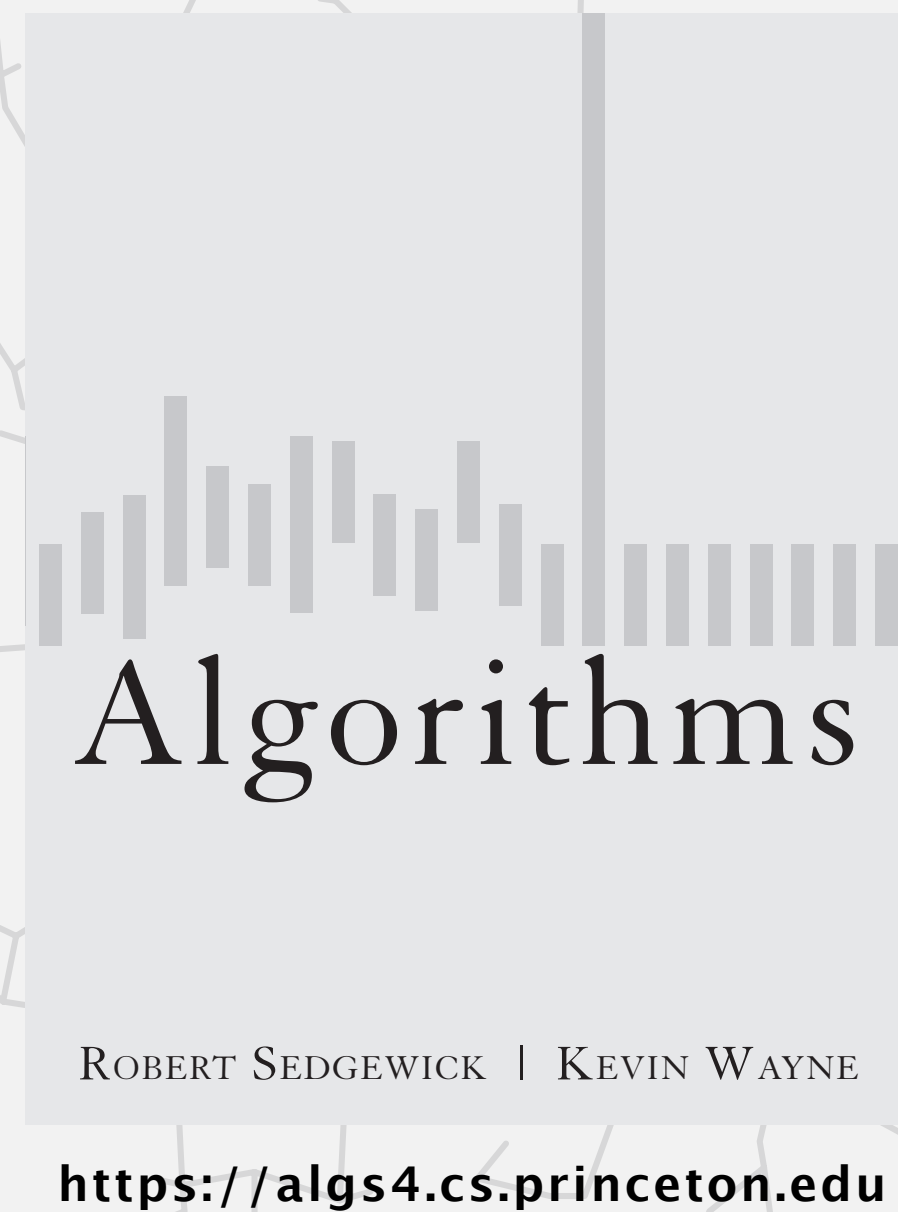
YAHOO!

amazon

intel

Microsoft

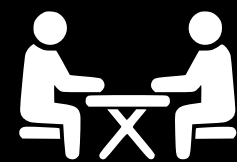
Akamai



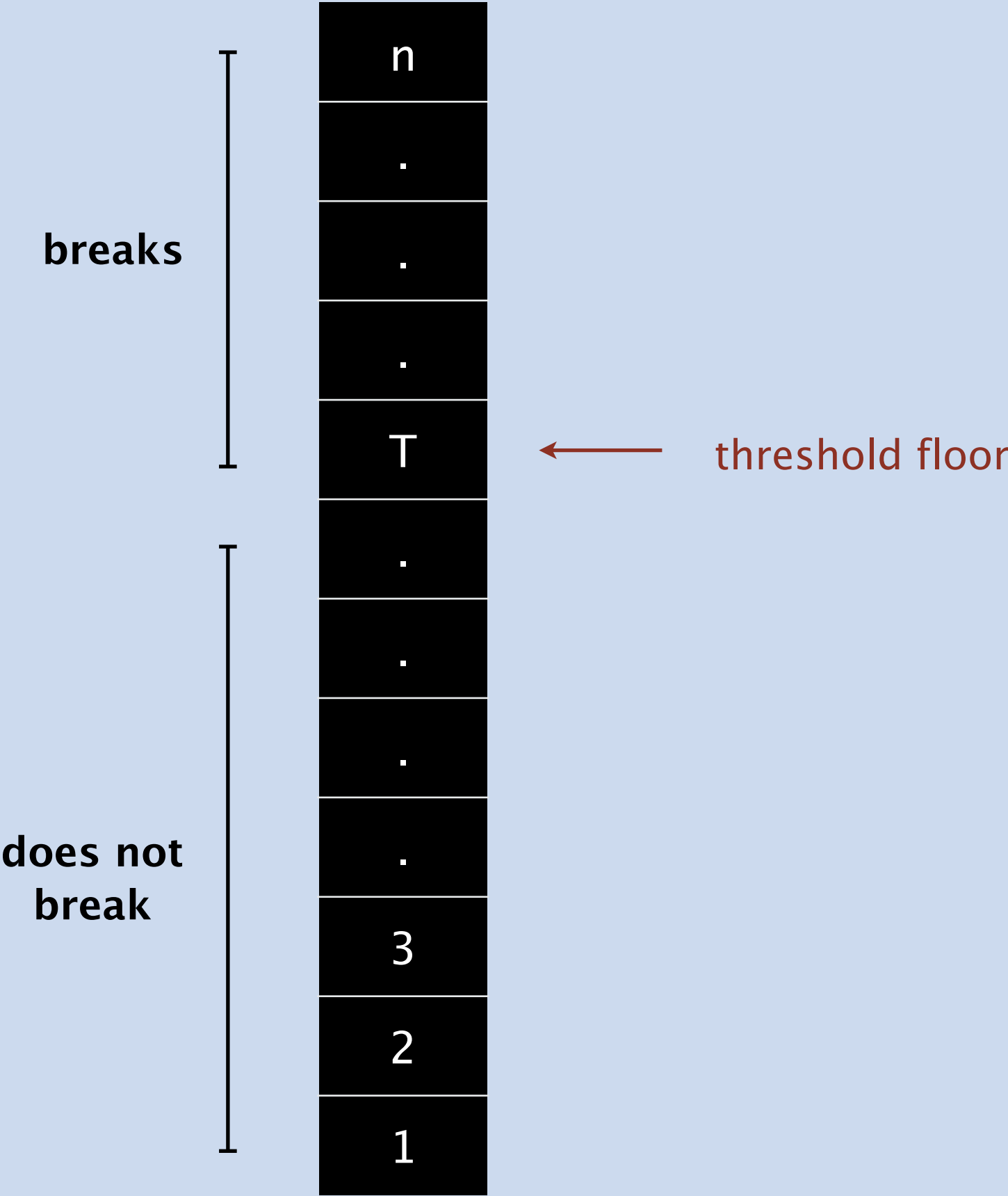
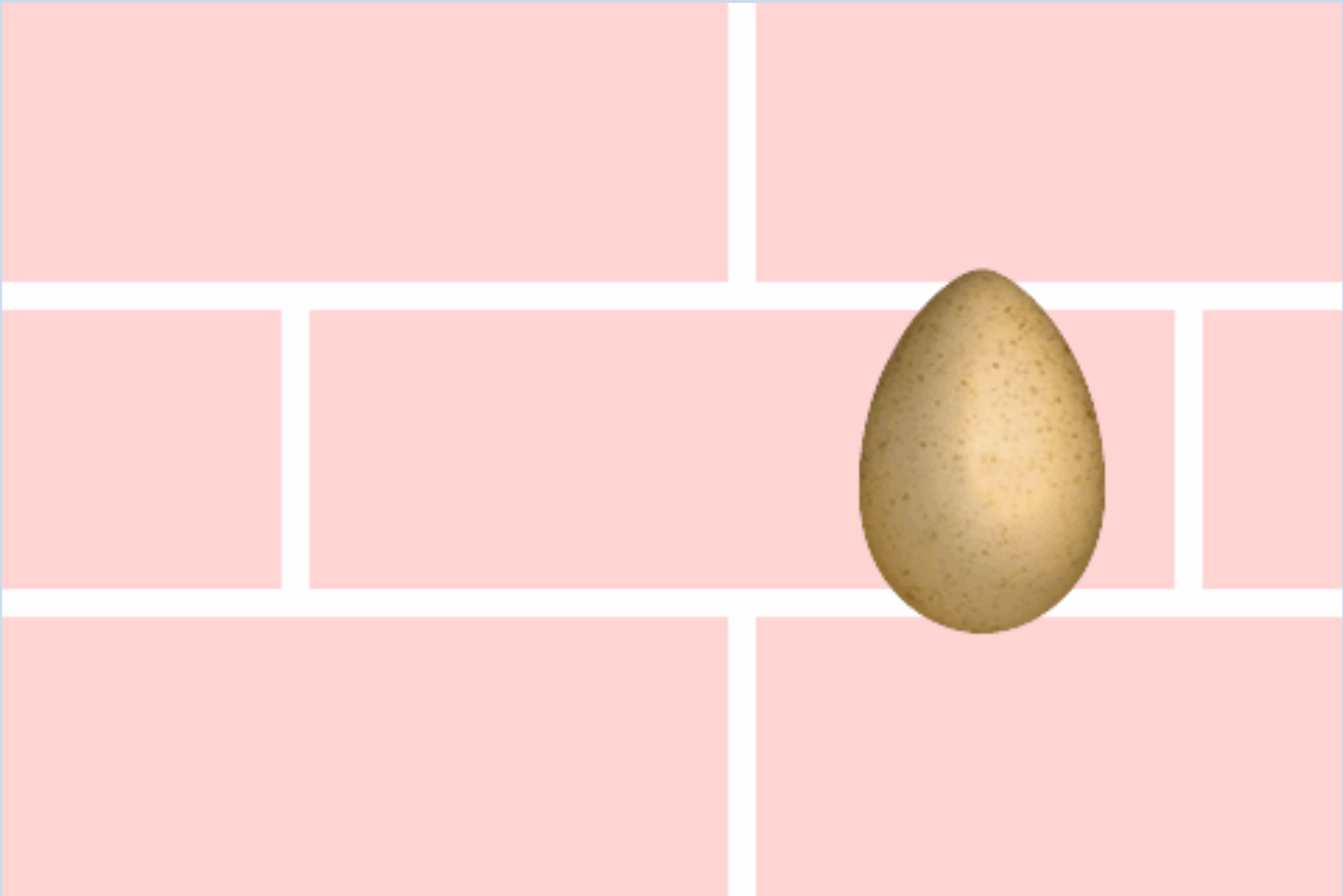
ALGORITHM DESIGN

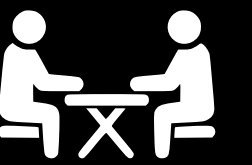
- *analysis of algorithms*
- *greed*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

EGG DROP



Goal. Find T using fewest drops.

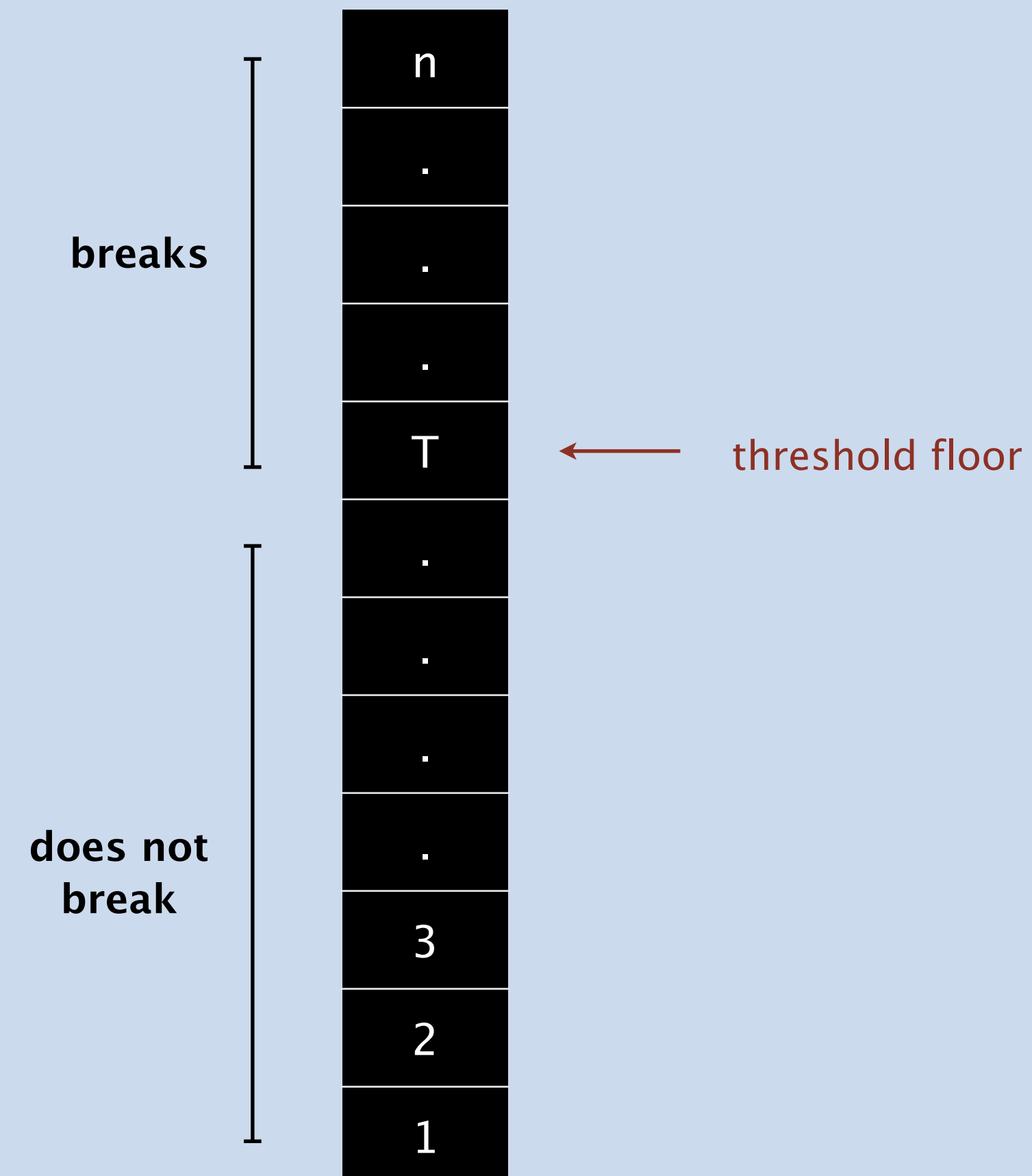




Goal. Find T using fewest drops.

Rules.

- An egg that breaks cannot be reused.
- An egg that survives a fall can be reused.
- The effect of a drop is the same for all eggs.
- An egg can break on floor 1 or survive on floor n .



EGG DROP



Goal. Find T using fewest drops.

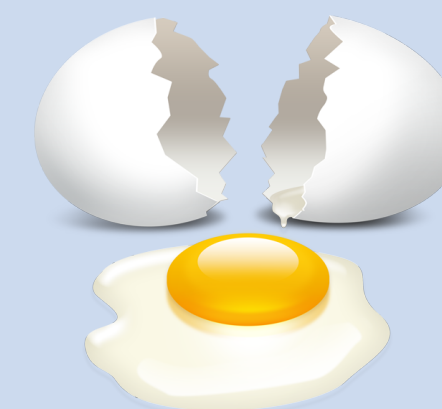
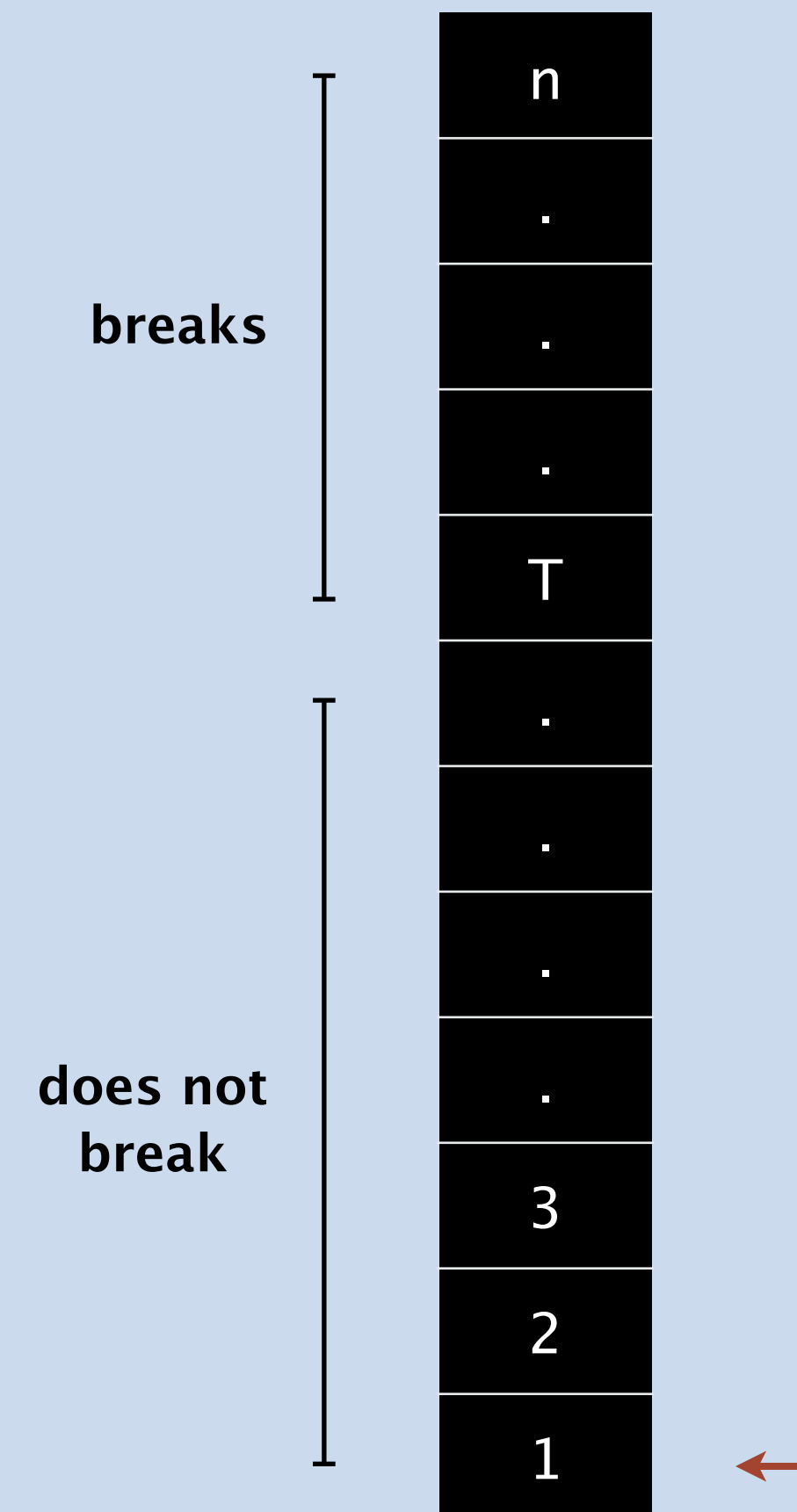
Variant 0. 1 egg.

Solution. Use **sequential search**: drop on floors 1, 2, 3, ... until egg breaks.

Analysis. 1 egg and $\leq n$ drops.

Analysis. 1 egg and T drops.

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



EGG DROP



Goal. Find T using fewest drops.

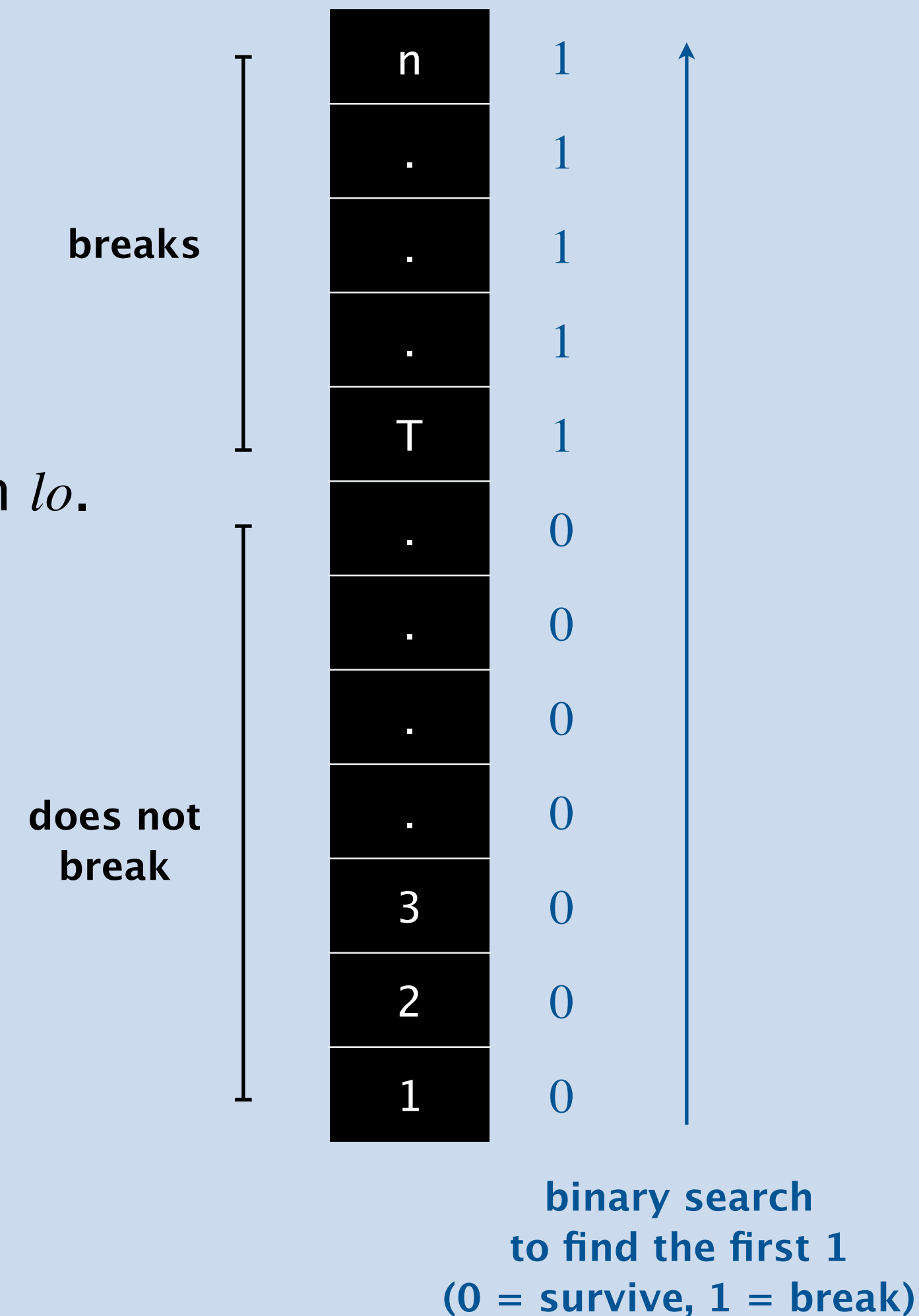
Variant 1. ∞ eggs.

Solution. Binary search for T .

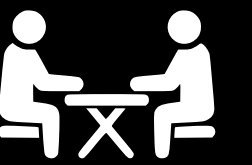
- Initialize $[lo, hi] = [0, n+1]$.
- Maintain invariant: egg breaks on floor hi but not on lo .
- Repeat until length of interval is 1:
 - drop on floor $mid = \lfloor (lo + hi) / 2 \rfloor$.
 - if it breaks, update $hi = mid$.
 - otherwise, update $lo = mid$.

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

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



EGG DROP



Goal. Find T using fewest drops.

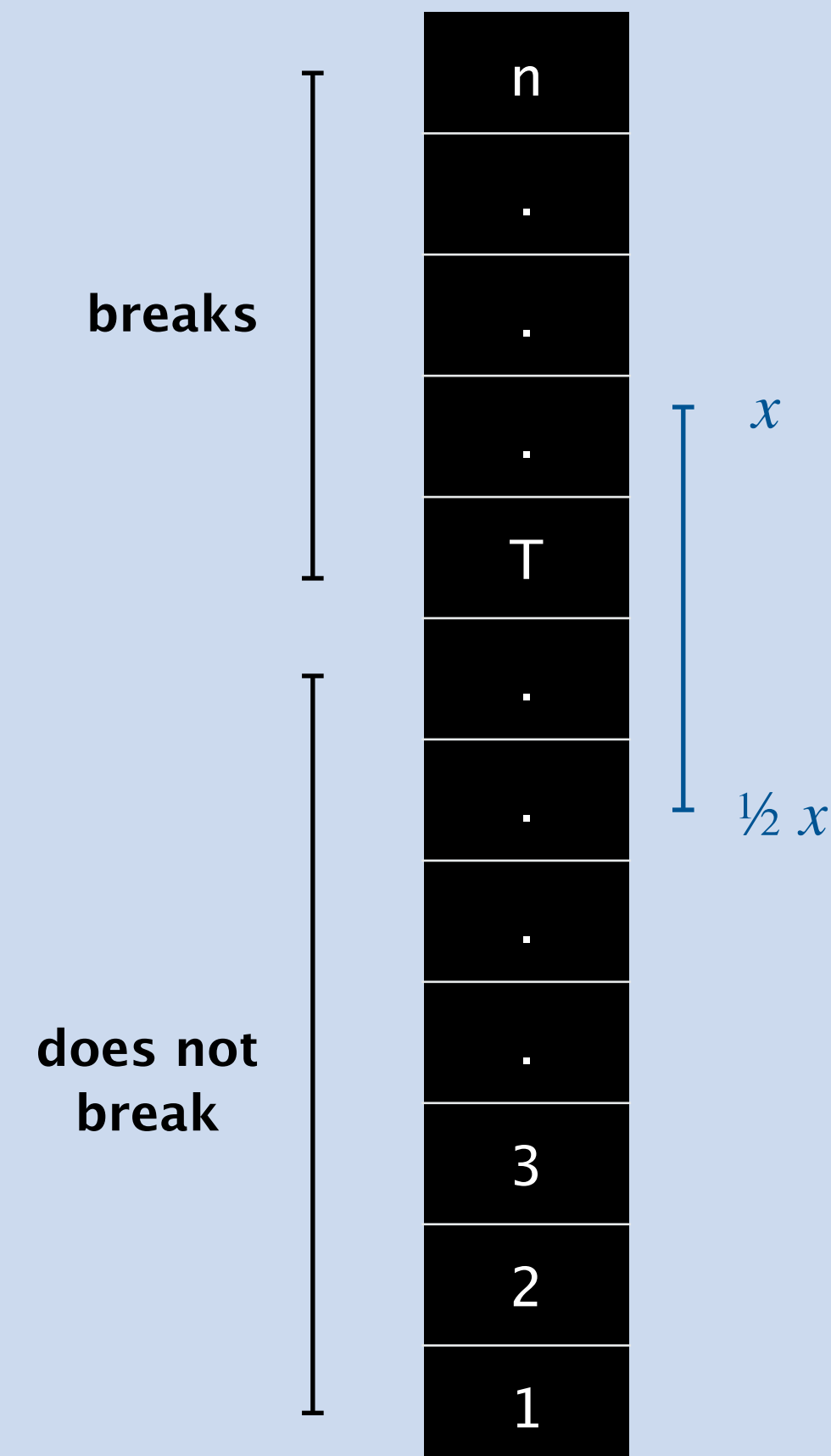
Variant 1'. ∞ eggs and $\Theta(\log T)$ drops.

Solution. Use **repeated doubling**; then **binary search**.

- Drop on floors $1, 2, 4, 8, 16, \dots, x$ to find a floor x such that the egg breaks on floor x but not on $\frac{1}{2}x$.
- Binary search in 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.
- Observe that $T \leq x < 2T$.



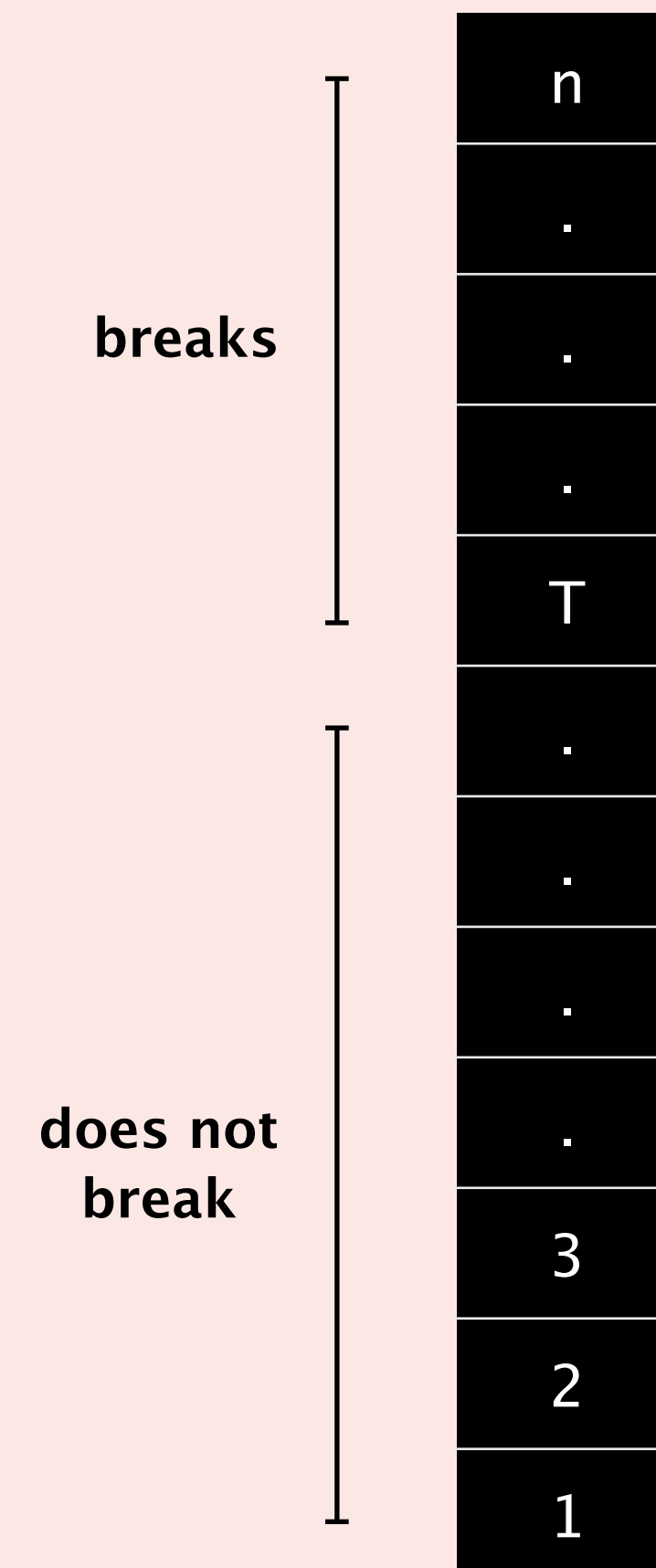


Goal. Find T using fewest drops.

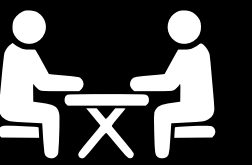
Variant 2. 2 eggs.

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 T using fewest drops.

Variant 2. 2 eggs.

Solution. Use **gridding**; then **sequential search**.

- Drop at floors $\sqrt{n}, 2\sqrt{n}, 3\sqrt{n}, \dots$
until first egg breaks, say at floor $c\sqrt{n}$.
- Sequential search in interval $[c\sqrt{n} - \sqrt{n}, c\sqrt{n}]$.

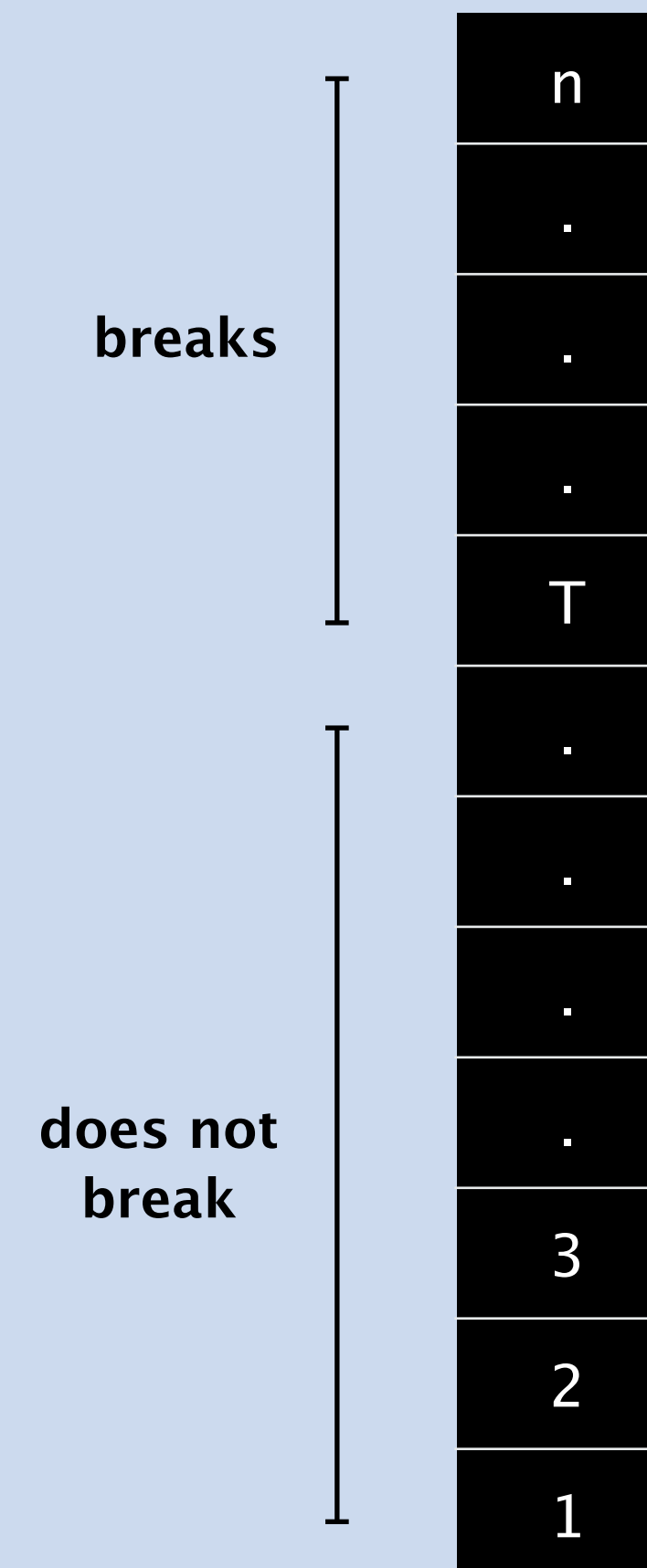
Analysis. At most $2\sqrt{n}$ drops.

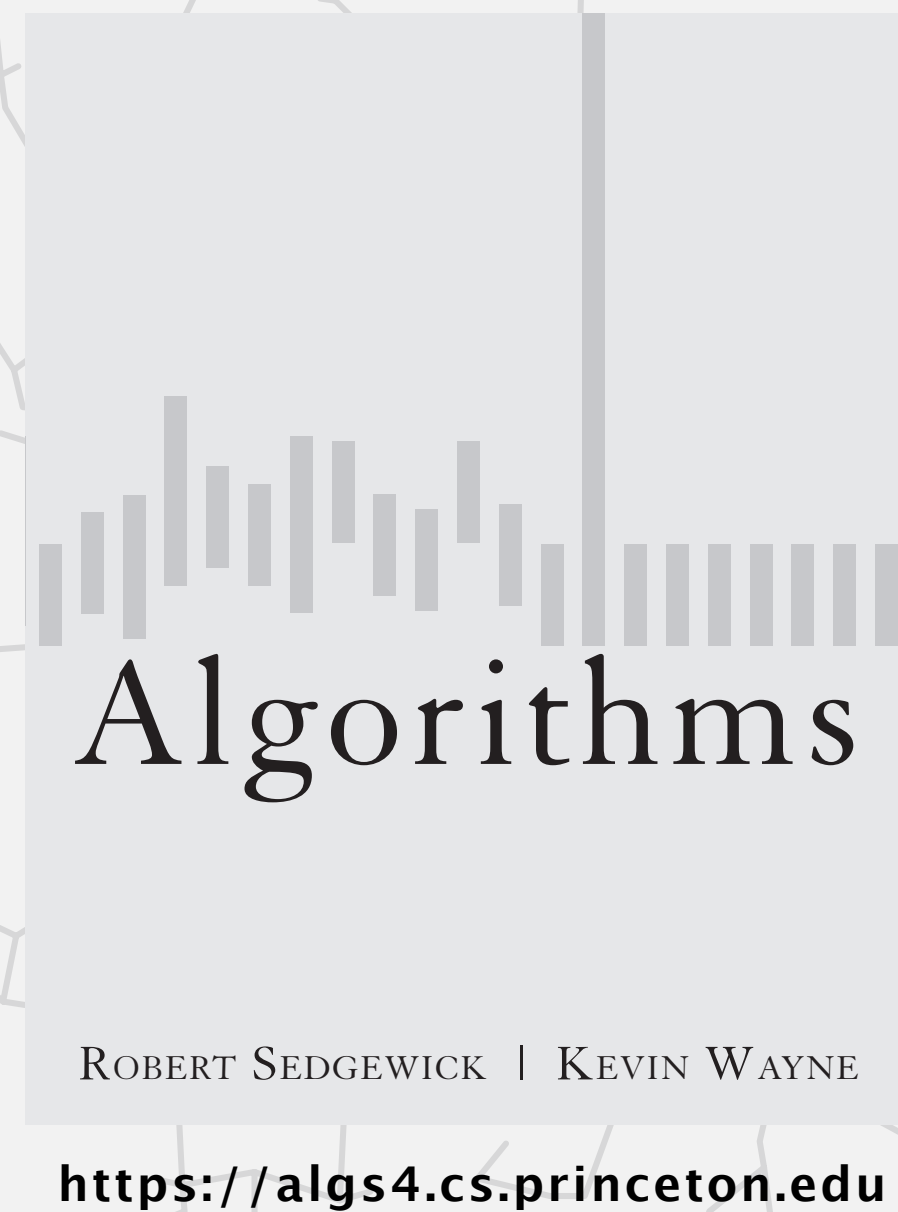
- First egg: $\leq \sqrt{n}$ drops.
- Second egg: $\leq \sqrt{n}$ drops.

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

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

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





ALGORITHM DESIGN

- *analysis of algorithms*
- *greed*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

Greedy algorithms

Make locally optimal, irrevocable, choices at each step.

Familiar examples.

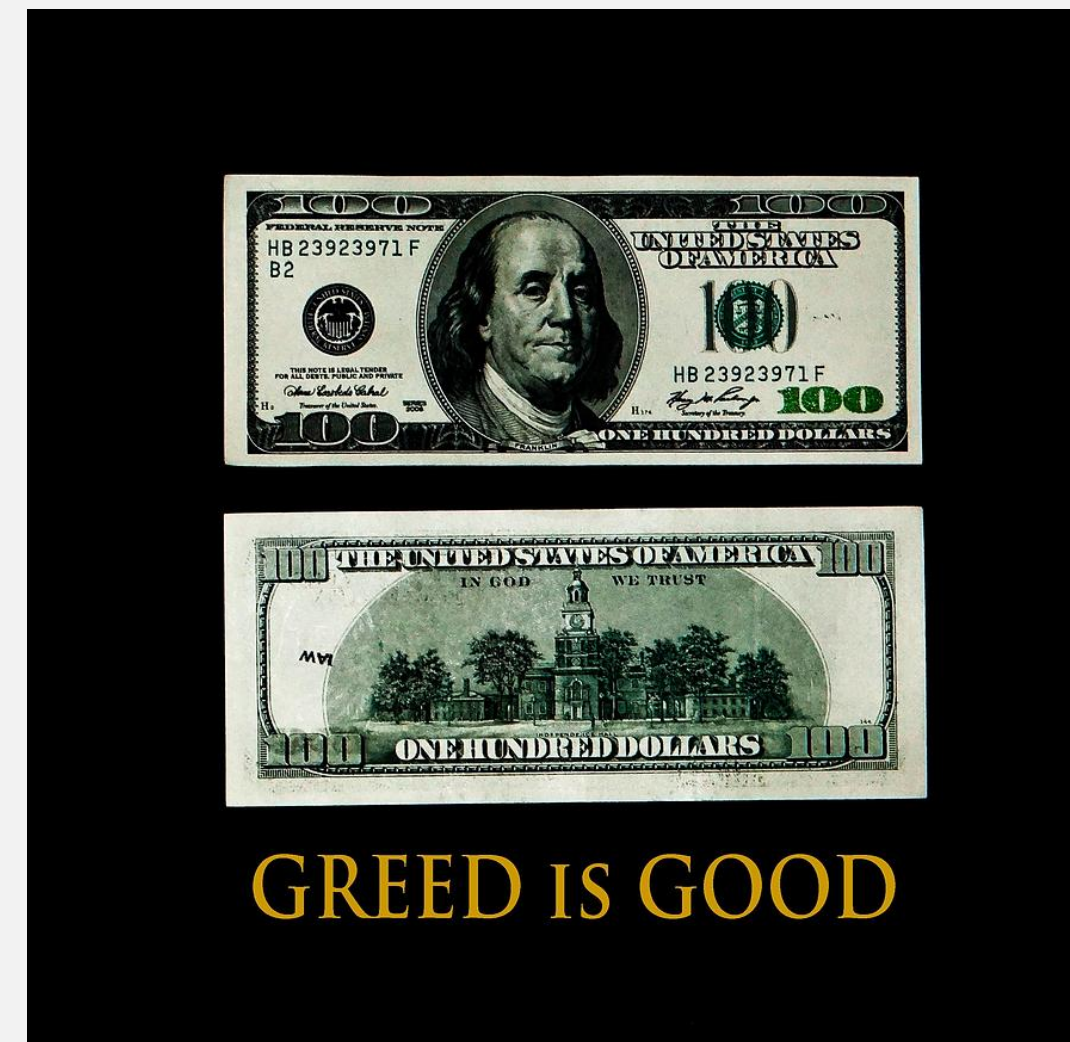
- Prim's algorithm. [for MST]
- Kruskal's algorithm. [for MST]
- Dijkstra's algorithm. [for shortest paths]
- Huffman's algorithm. [for data compression]

More classic examples.

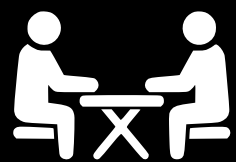
- A* search algorithm.
- Gale-Shapley algorithm for stable marriage.
- Greedy algorithm for matroids.
- ...

Caveat. Greedy algorithms rarely lead to provably optimal solutions.

[but often used anyway in practice, especially for intractable problems]



COIN CHANGING PROBLEM AND CASHIER'S ALGORITHM



Goal. Given U. S. coin denominations { 1, 5, 10, 25, 100 }, devise a method to pay amount to customer using fewest coins.

Ex. 34¢.



6 coins

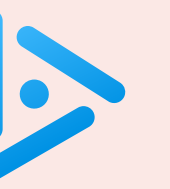


Cashier's (greedy) algorithm. Repeatedly add the coin of the largest value that does not exceed the remaining amount to be paid.

Ex. \$2.89.



10 coins



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 $d_1 < d_2 < \dots < d_n$ provided $d_1 = 1$.
- C. Yes, because of special properties of U.S. coin denominations.
- D. No.

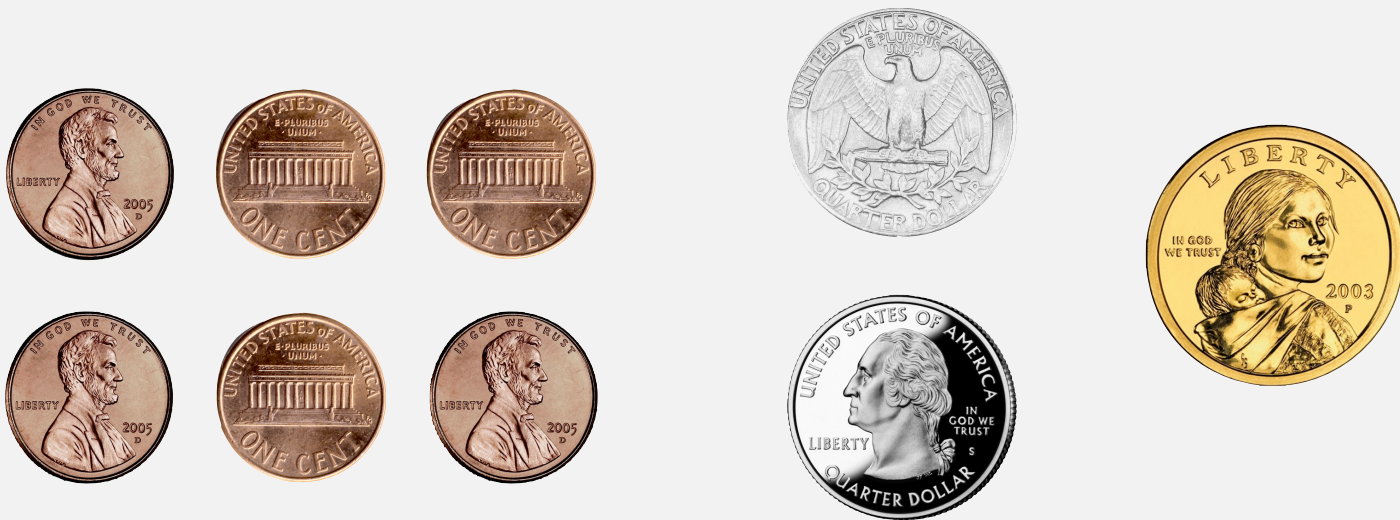


Properties of any optimal solution (for U.S. coin denominations)

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

Pf. Replace 5 pennies with 1 nickel.

← exchange argument



Property 2. Number of nickels $N \leq 1$.

← replace 2 nickels with 1 dime

Property 3. Number of dimes $D \leq 2$.

← replace 3 dimes with 1 quarter and 1 nickel

Property 4. Number of quarters $Q \leq 3$.

← replace 4 quarters with 1 dollar

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

Pf.

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

significance: total amount of change from
pennies, nickels, dimes, and quarters

Property 6. $P + 5N + 10D + 25Q \leq 99$.

P1 \Rightarrow contributes
at most 4

P5 \Rightarrow contributes
at most 20

P4 \Rightarrow contributes
at most 75

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

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

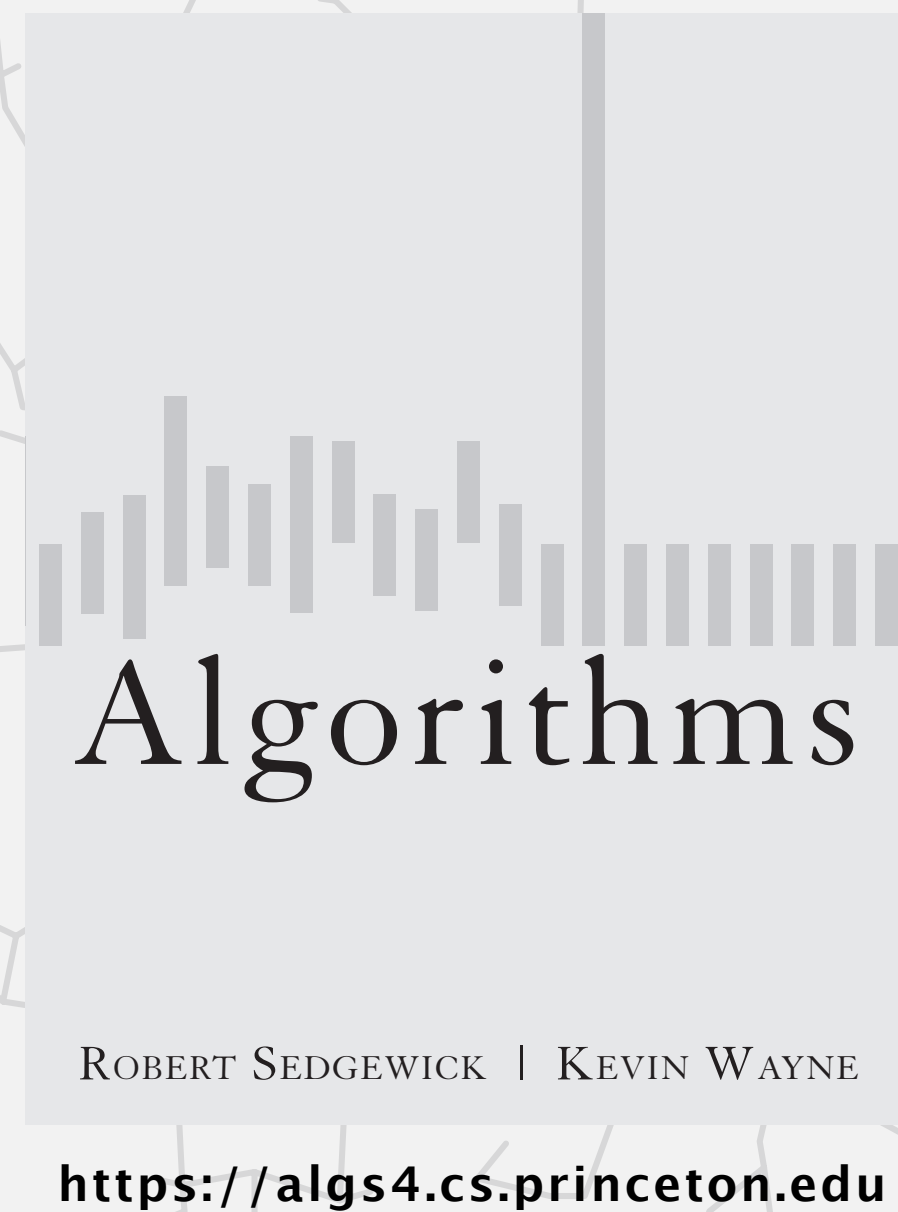
Pf. [for dollar coins]

- Suppose we are changing amount $\$x.yz$.
- Cashier's algorithm takes x dollar coins.
- Suppose (for the sake of contradiction) that an optimal solution takes fewer than x dollar coins.
- Then, optimal solution satisfies $P + 5N + 10D + 25Q \geq 100$.
- This contradicts Property 6:

$$P + 5N + 10D + 25Q \leq 99$$

↑
must make change for $\geq 100\text{¢}$
using only pennies, nickels, dimes, and quarters

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



ALGORITHM DESIGN

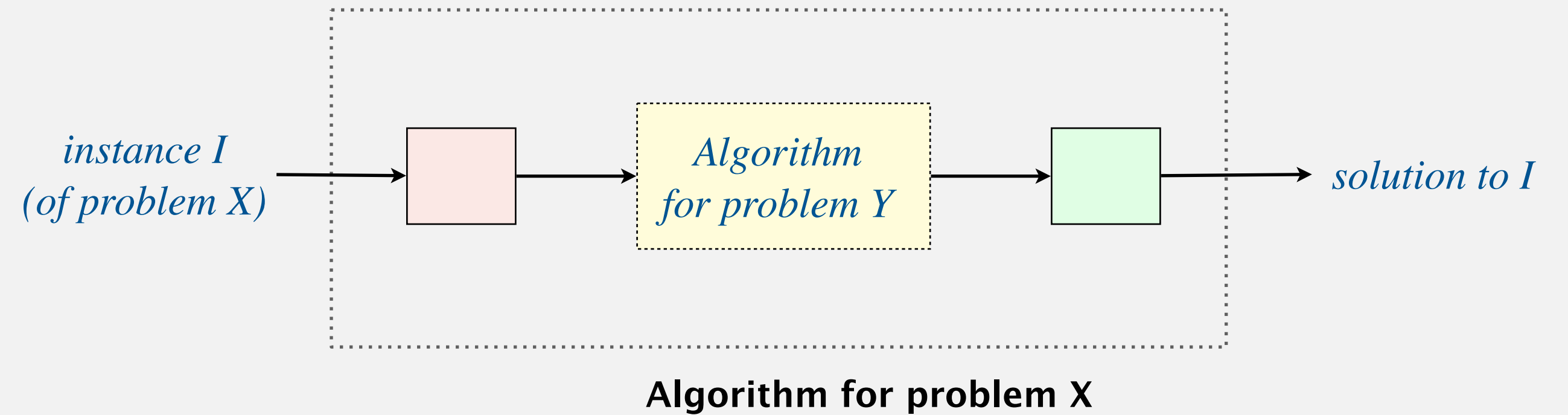
- *analysis of algorithms*
- *greed*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

Reductions

Problem X **reduces to** problem Y if you can solve X by using an algorithm for Y .

Ex 1. Finding the median reduces to sorting.

Ex 2. Bipartite matching reduces to maxflow.

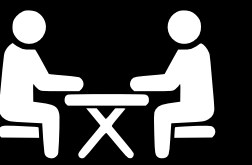


Many many problems reduce to:

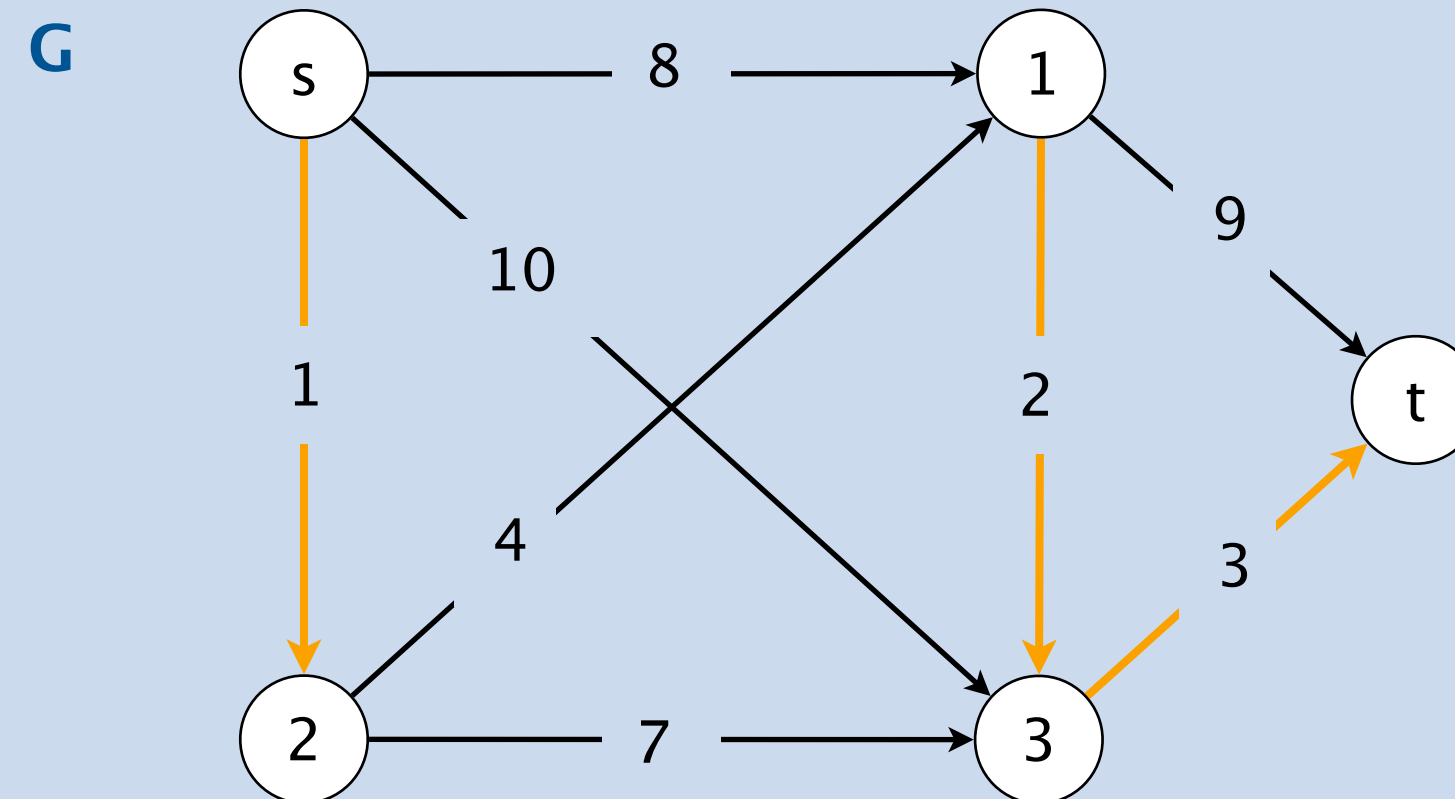
- Sorting.
- Maxflow.
- Suffix array.
- Shortest path.
- Minimum spanning tree.
- Linear/semidefinite programming. ← see ORF 307 or ORF 363
- ...

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

SHORTEST PATH WITH ORANGE AND BLACK EDGES



Goal. Given a digraph, where each edge has a positive weight and is orange or black, find shortest path from s to t that uses at most k orange edges.



$k = 0: s \rightarrow 1 \rightarrow t \quad (17)$

$k = 1: s \rightarrow 3 \rightarrow t \quad (13)$

$k = 2: s \rightarrow 2 \rightarrow 3 \rightarrow t \quad (11)$

$k = 3: s \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow t \quad (10)$

$k = 4: s \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow t \quad (10)$

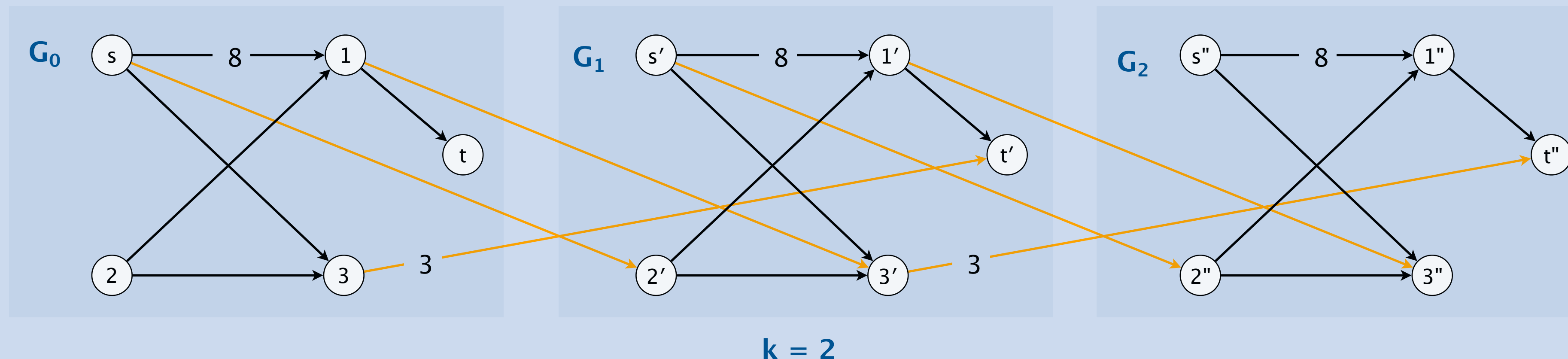
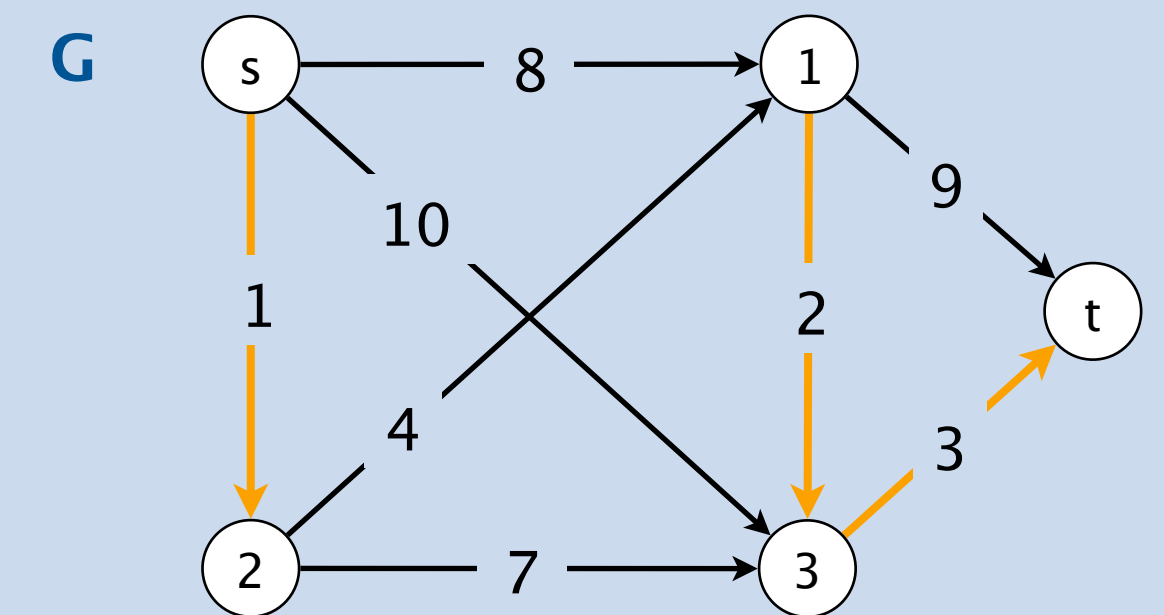
SHORTEST PATH WITH ORANGE AND BLACK EDGES



Goal. Given a digraph, where each edge has a positive weight and is orange or black, find shortest path from s to t that uses at most k orange edges.

A reduction to shortest paths:

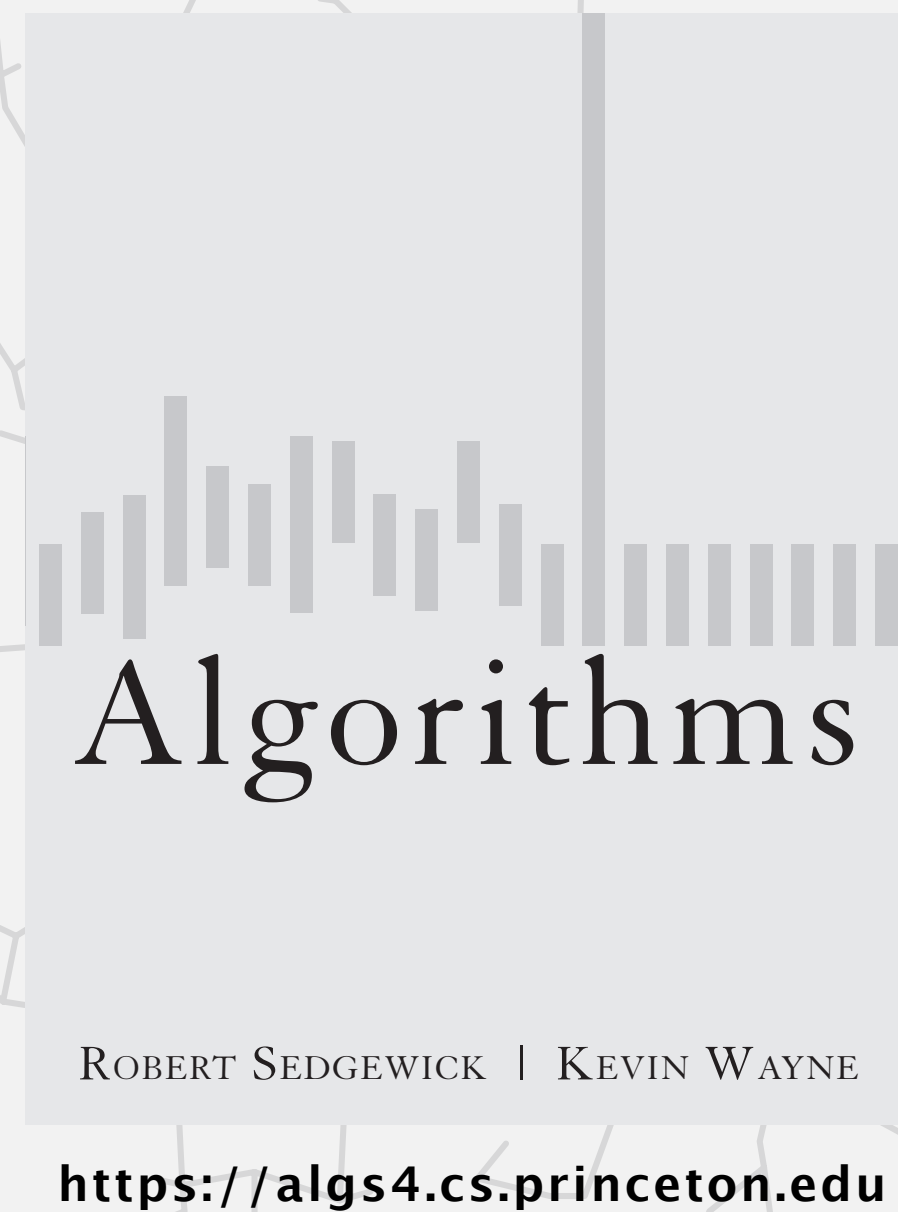
- Create $k+1$ copies of the vertices in digraph G , labeled G_0, G_1, \dots, G_k .
- For each black edge $v \rightarrow w$: add edge from vertex v in graph G_i to vertex w in G_i .
- For each orange edge $v \rightarrow w$: add edge from vertex v in graph G_i to vertex w in G_{i+1} .
- Compute shortest path from s to any copy of t .





What is worst-case running time of algorithm as a function of k , the number of vertices V , and the number of edges E ? Assume $E \geq V$ and $k > 0$.

- A. $\Theta(E \log V)$
- B. $\Theta(k E)$
- C. $\Theta(k E \log V)$
- D. $\Theta(k^2 E \log V)$



ALGORITHM DESIGN

- *analysis of algorithms*
- *greedy*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

Dynamic programming

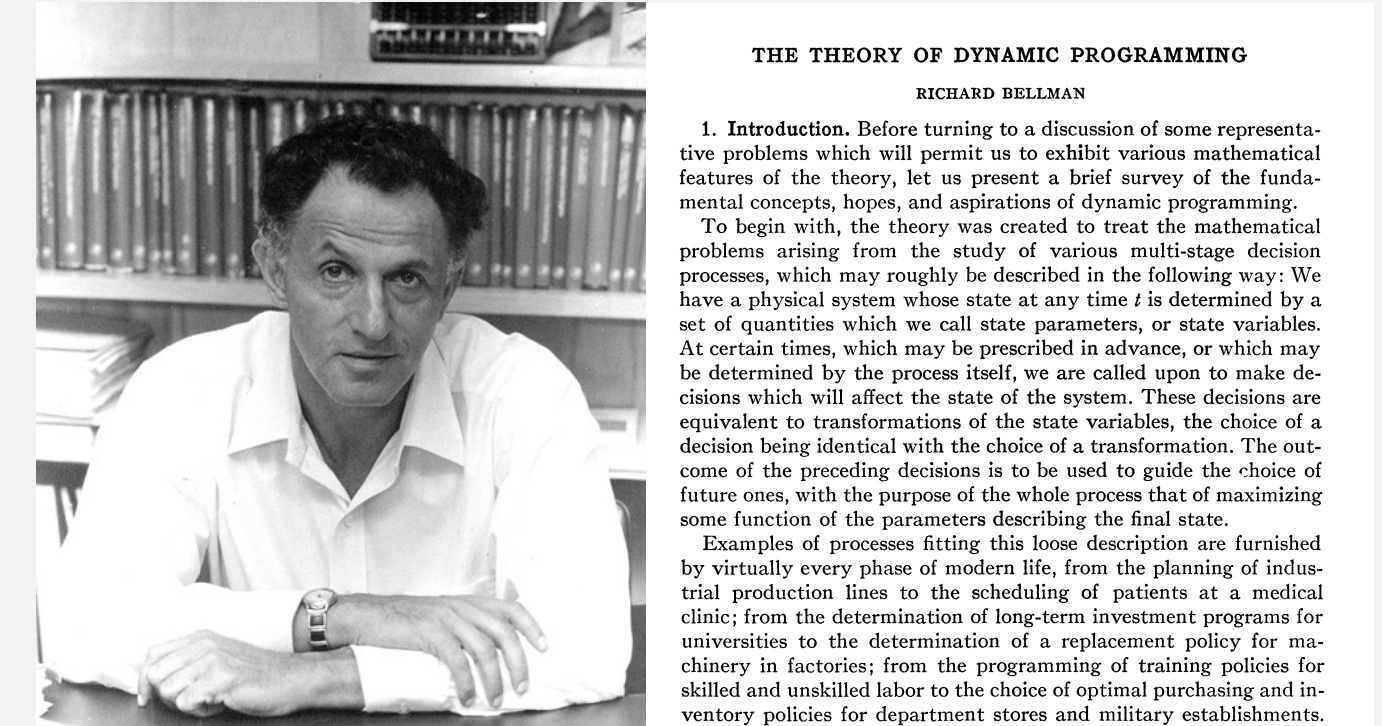
- Break up problem into a series of overlapping subproblems.
- Build up solutions to larger and larger subproblems.
[caching solutions to subproblems in a table for later reuse]

Familiar examples.

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

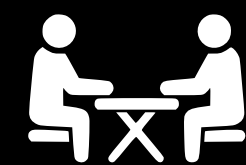
More classic examples.

- Unix diff.
- Viterbi algorithm for hidden Markov models.
- CKY algorithm for parsing context-free grammars.
- Needleman–Wunsch/Smith–Waterman for DNA sequence alignment.
- ...

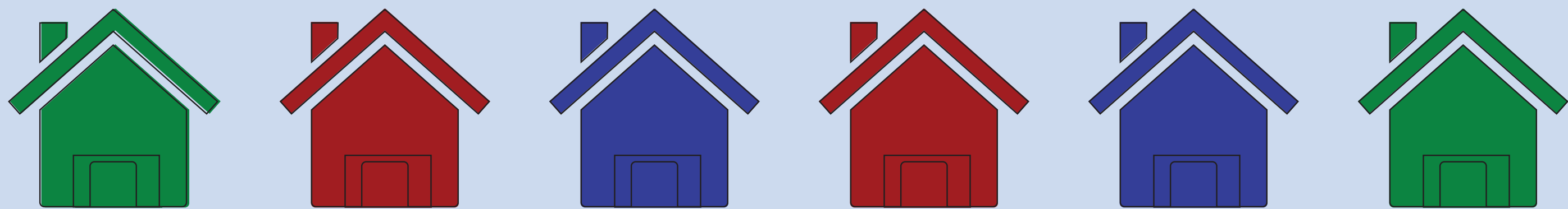


Richard Bellman, *46

HOUSE COLORING PROBLEM



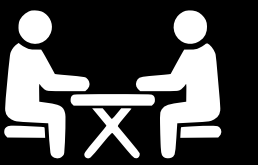
- Goal.** Paint a row of n houses red, green, or blue so that:
- Minimize total cost, where $cost(i, color)$ is cost to paint i given color.
 - No two adjacent houses have the same color.



	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:

- Minimize total cost, where $cost(i, color)$ is cost to paint i given color.
- No two adjacent houses have the same color.

Subproblems.

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

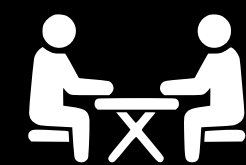
Dynamic programming recurrence.

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

← “optimal substructure”

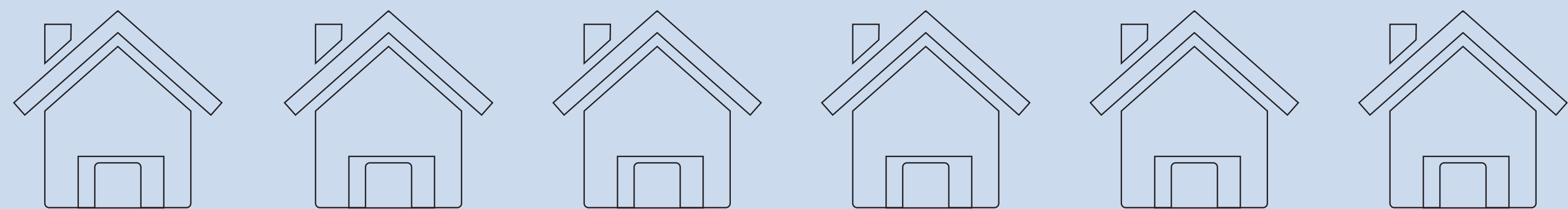
(optimal solution can be constructed from optimal solutions to smaller subproblems)

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 $B(i+1)$.

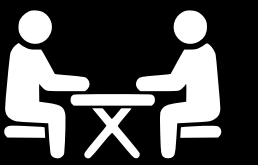
$$\begin{aligned} B(6) &= \text{cost}(6, \text{blue}) + \min \{ R(5), G(5) \} \\ &= 7 + \min \{ 29, 32 \} \\ &= 36 \end{aligned}$$



	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]);
```

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

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

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

Proposition. Takes $\Theta(n)$ time and uses $\Theta(n)$ extra space.



ALGORITHM DESIGN

- *analysis of algorithms*
- *greedy*
- *reduction*
- *dynamic programming*
- ***divide-and-conquer***
- *randomization*

Divide and conquer

- Break up problem into two or more **independent subproblems**.
- Solve each subproblem recursively.
- Combine solutions to subproblems to form solution to original problem.

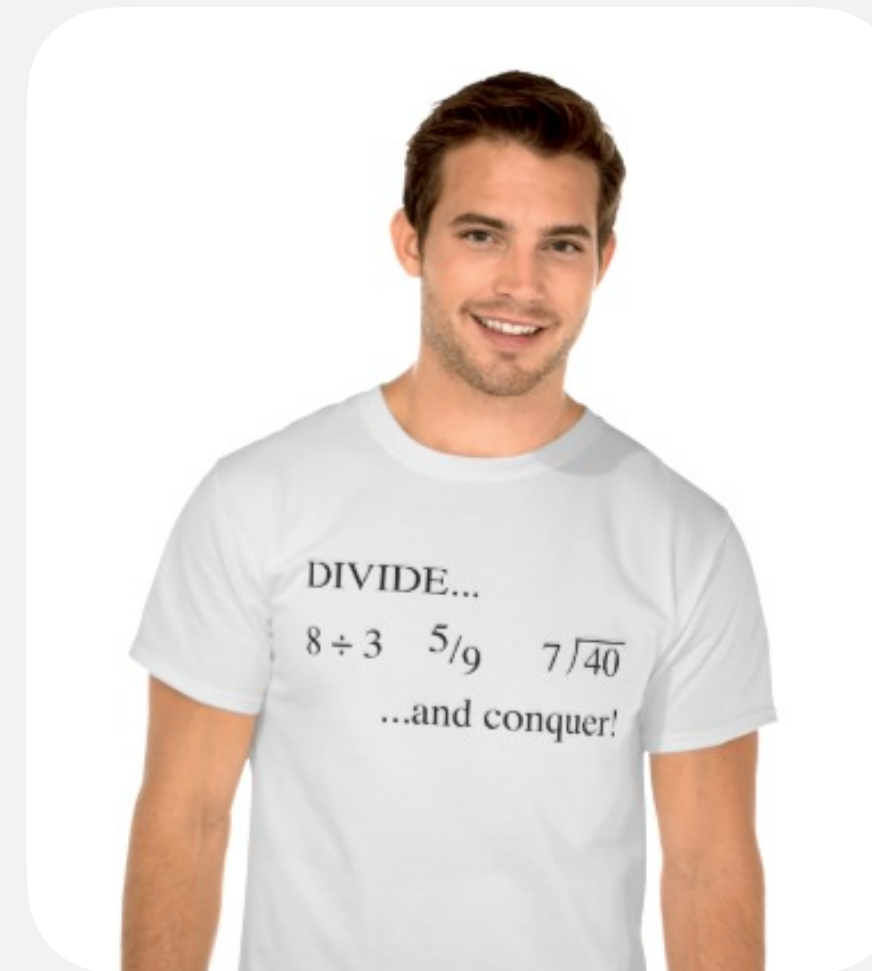
Familiar examples.

- Mergesort.
- Quicksort.

More classic examples.

- Closest pair.
- Convolution and FFT.
- Matrix multiplication.
- Integer multiplication.

...



needs to take COS 226?

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



<https://algs4.cs.princeton.edu>

ALGORITHM DESIGN

- *analysis of algorithms*
- *greed*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- ***randomization***

Randomized algorithms

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



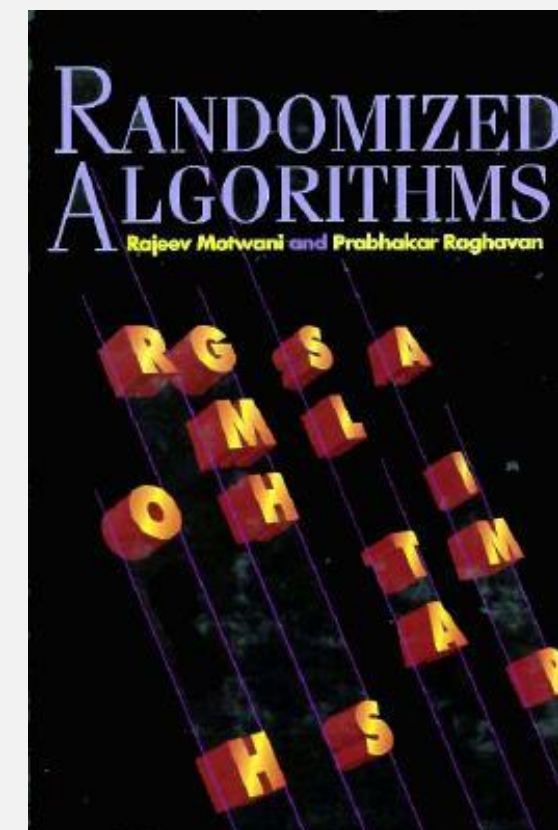
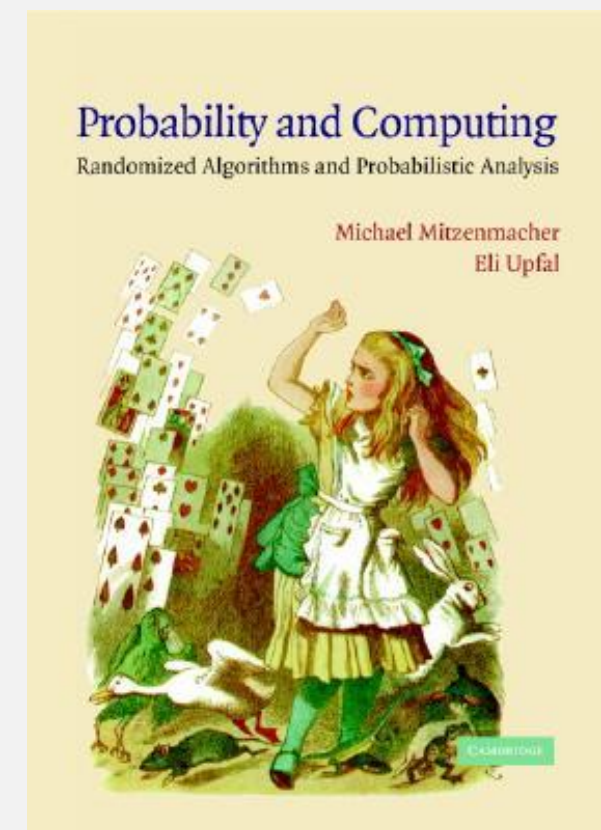
Familiar examples.

- Quicksort.
- Quickselect.

More classic examples.

- Miller–Rabin primality testing.
- Rabin–Karp substring search.
- Polynomial identity testing.
- Volume of convex body.
- Universal hashing.
- Global min cut.

...



NUTS AND BOLTS



Problem. A disorganized carpenter has a mixed pile of n nuts and n bolts.

- The goal is to find the corresponding pairs of nuts and bolts.
- Each nut fits exactly one bolt; each bolt fits exactly one nut.
- By fitting a nut and a bolt together, the carpenter can determine which is bigger.

← 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 makes $O(n \log n)$ compares.

NUTS AND BOLTS

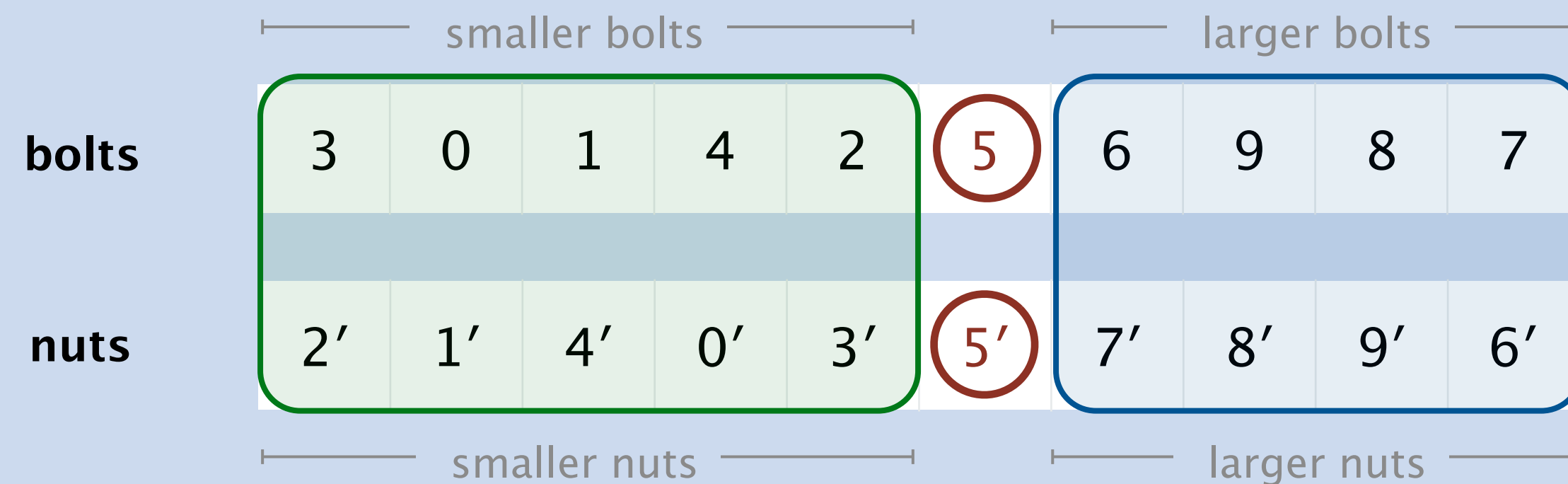


Shuffle. Shuffle the nuts and bolts.

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

Partition.

- Pick leftmost bolt i and compare against all nuts; divide nuts smaller than i from those that are larger than i .
- Let i' be the nut that matches bolt i . Compare i' against all bolts; divide bolts smaller than i' from those that are larger than i' .



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



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. $\Theta(n^2)$



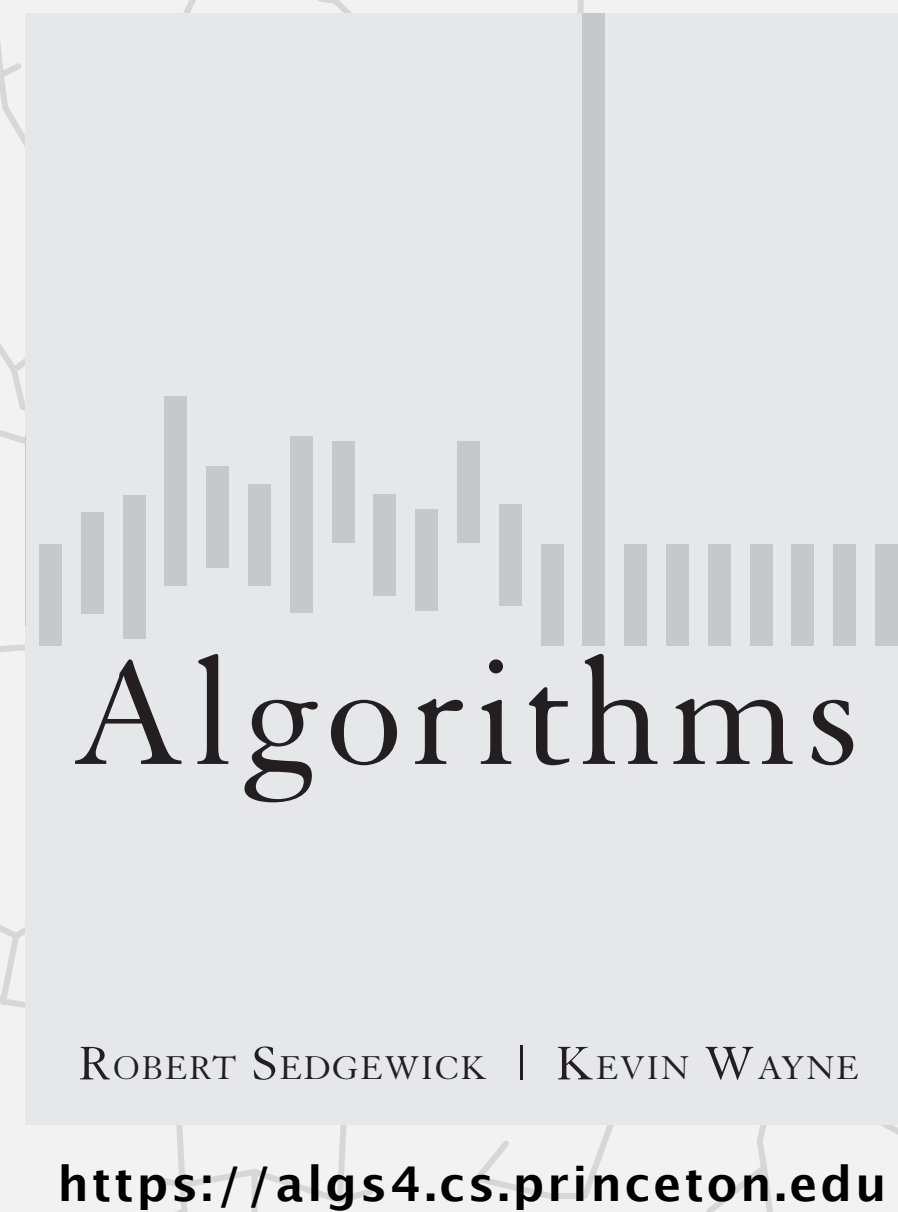
Hiring bonus. Algorithm that takes $O(n \log n)$ time in the **worst case**.

Chapter 27
Matching Nuts and Bolts in $O(n \log n)$ Time
(Extended Abstract)

János Komlós^{1,4} Yuan Ma² 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.



ALGORITHM DESIGN

- *analysis of algorithms*
- *greedy*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*
- *credits*



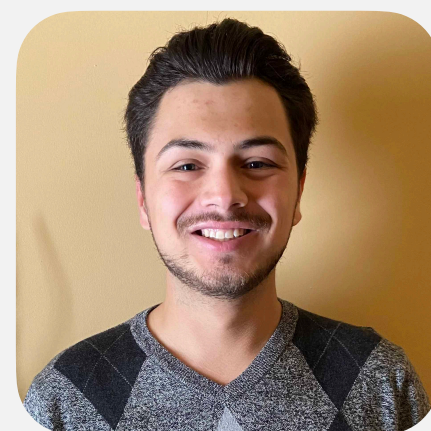
Co-instructors and graduate student preceptors.



Dan Leyzberg



Jérémie Lumbroso



Gabriel Contreras



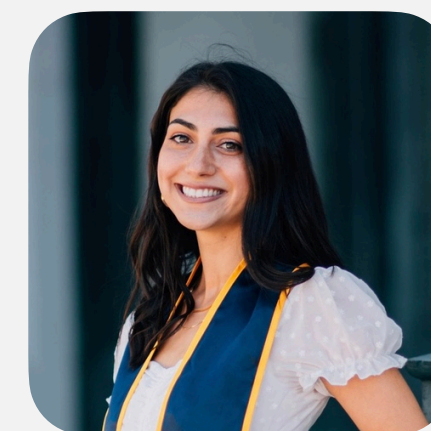
Jennifer Lam



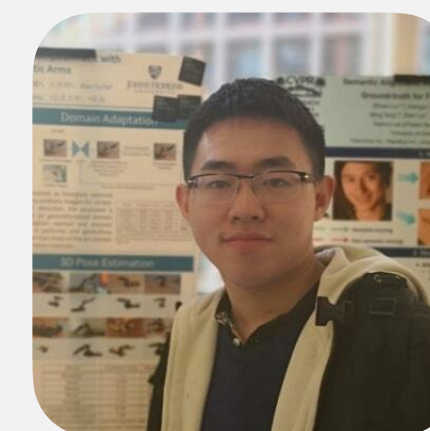
Ross Texeira



Yingxi Lin



Morgan Nanez



Yiming Zuo



Shelley Xia



Sabhya Chhabria

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

A final thought

*“ Algorithms and data structures are love.
Algorithms and data structures are life. ”*
— anonymous COS 226 student

© Copyright 2023 Robert Sedgewick and Kevin Wayne