https://algs4.cs.princeton.edu

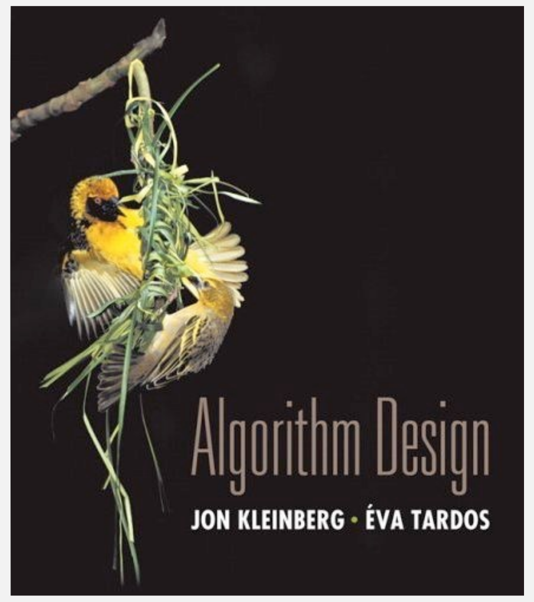
Algorithm Design

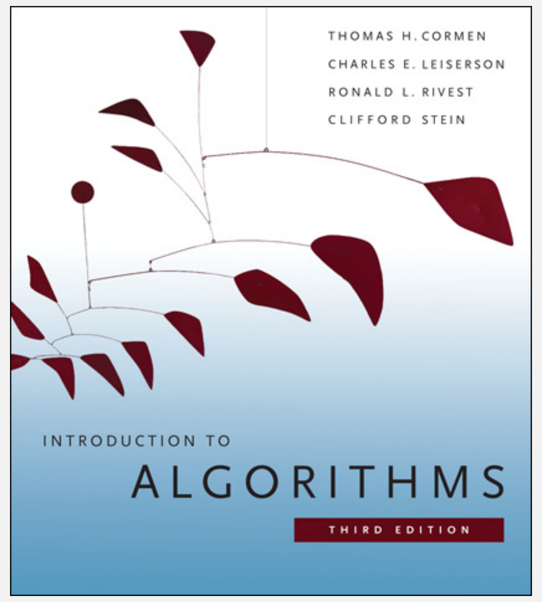
- *analysis of algorithms*
- *greed*
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

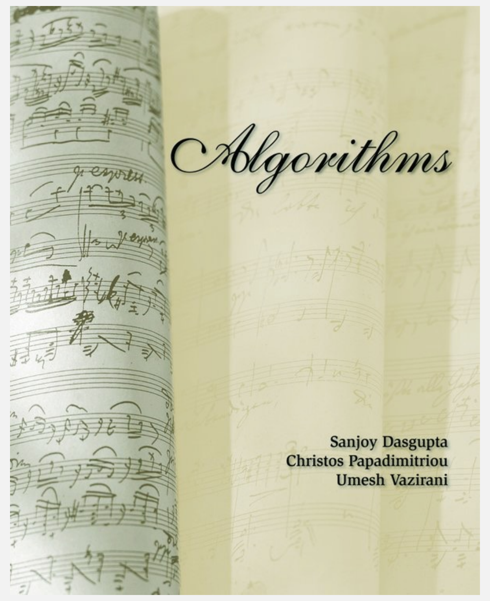
Algorithm design

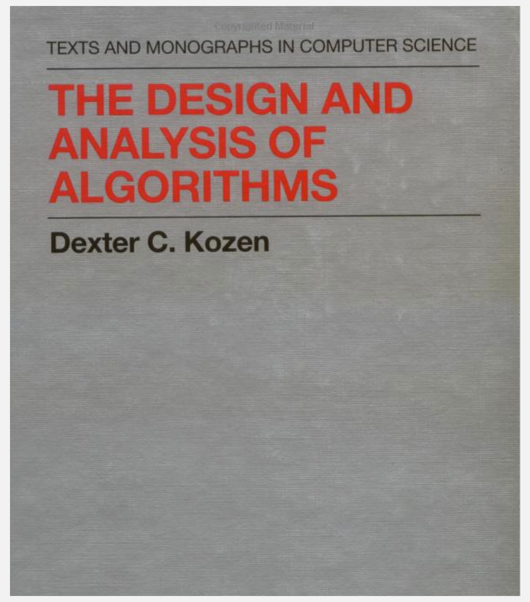
Algorithm design patterns.

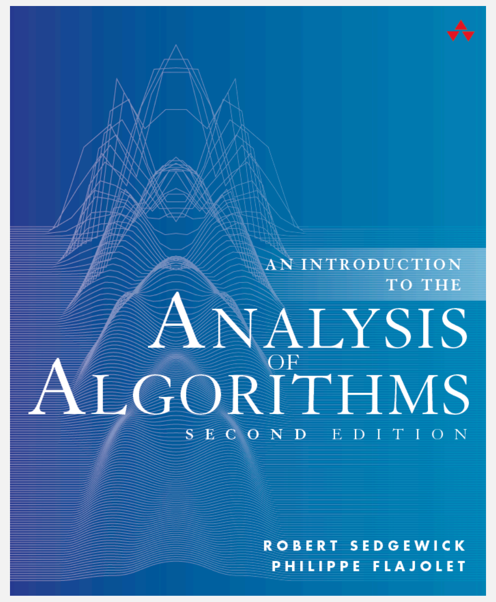
- Analysis of algorithms.
- Greed.
- Reduction.
- Dynamic programming.
- Divide-and-conquer.
- Randomization.

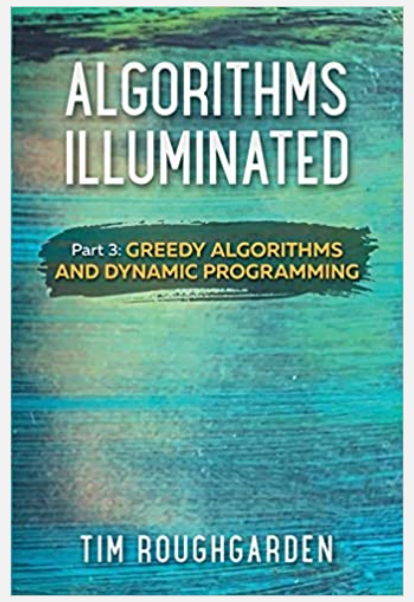












Want more? See COS 240, COS 343, COS 423, COS 445, COS 451, COS 488, ….

INTERVIEW QUESTIONS

Google
GitHub
PANDORA
facebook
RSA SECURITY
Cisco Systems
slack
airbnb
NETFLIX
Square
in
lyft
Morgan Stanley
IBM
nVIDIA
TESLA
JANE STREET
SPACEX
Adobe
DE Shaw & Co
UBER
zoom
PIXAR ANIMATION STUDIOS
ORACLE
YAHOO!
amazon
intel
Microsoft
Akamai

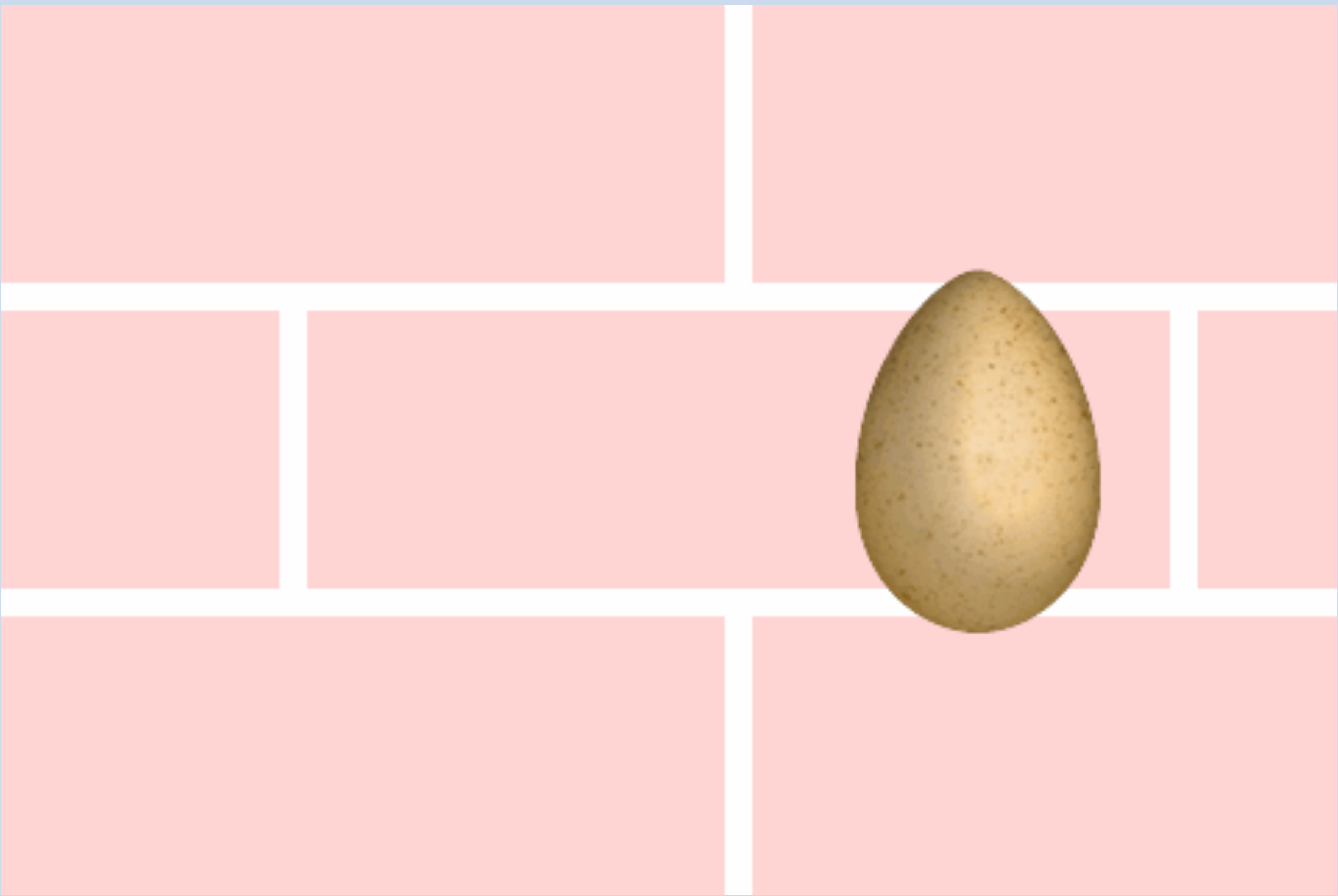
https://algs4.cs.princeton.edu

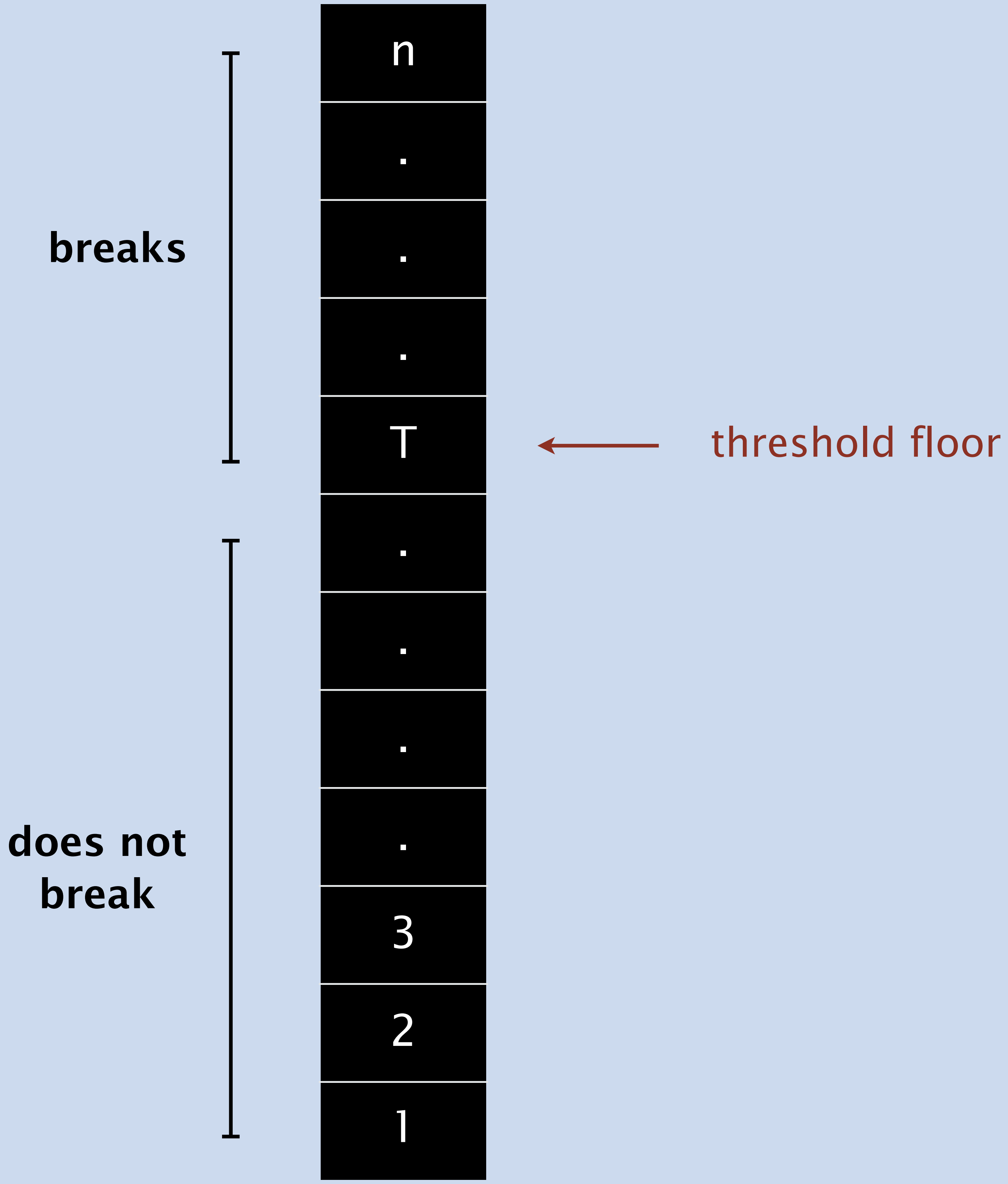
Algorithm Design

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

EGG DROP

Goal. Find T using fewest drops.





EGG DROP

Goal. Find T using fewest drops.

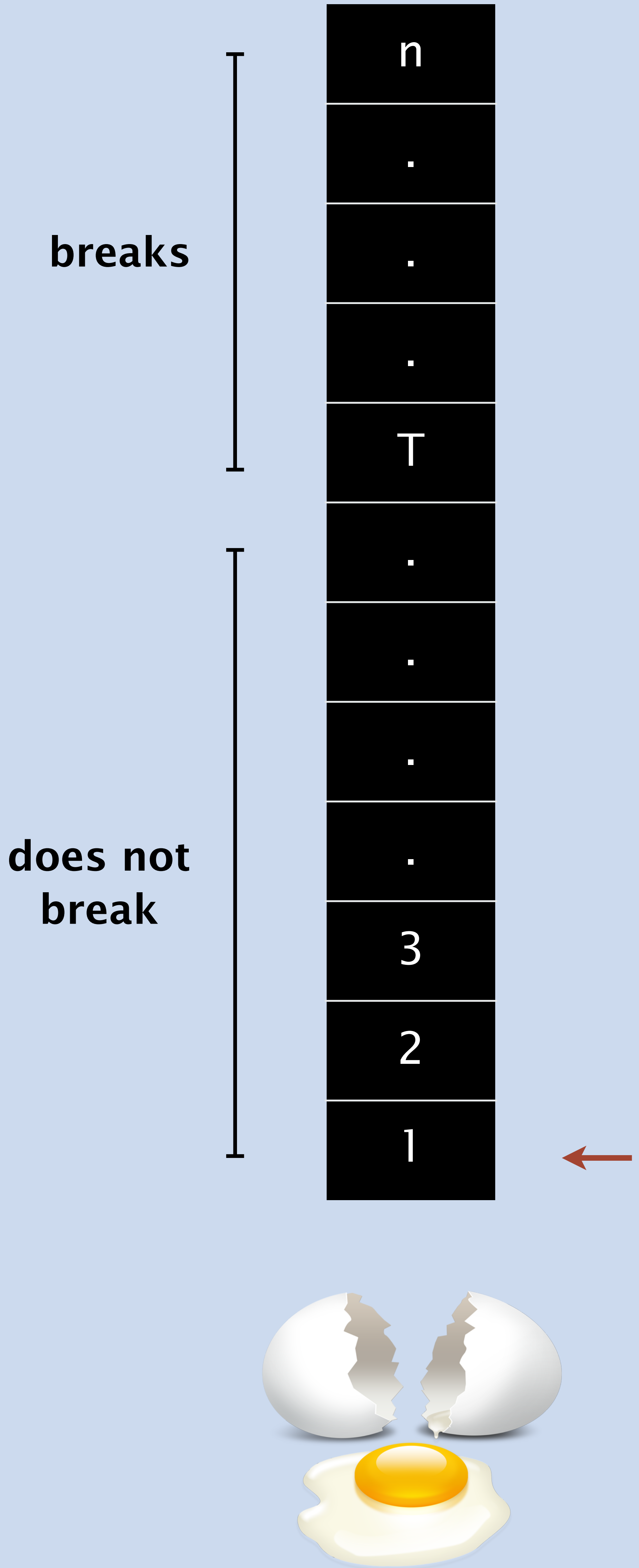
Variant 0. 1 egg.

Solution. Use sequential search: drop on floors $1, 2, 3, \ldots$ 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.

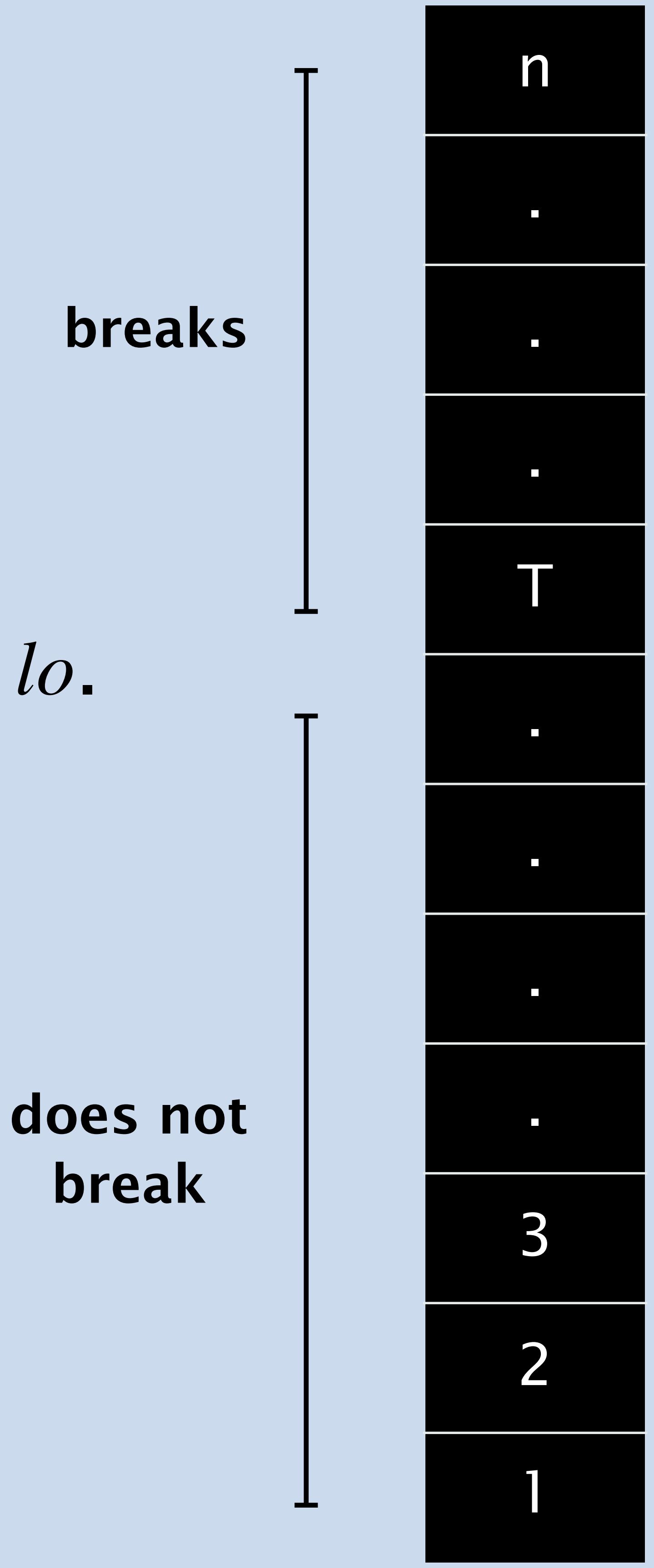
Variant 1. ∞ eggs.

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?



Goal. Find T using fewest drops.

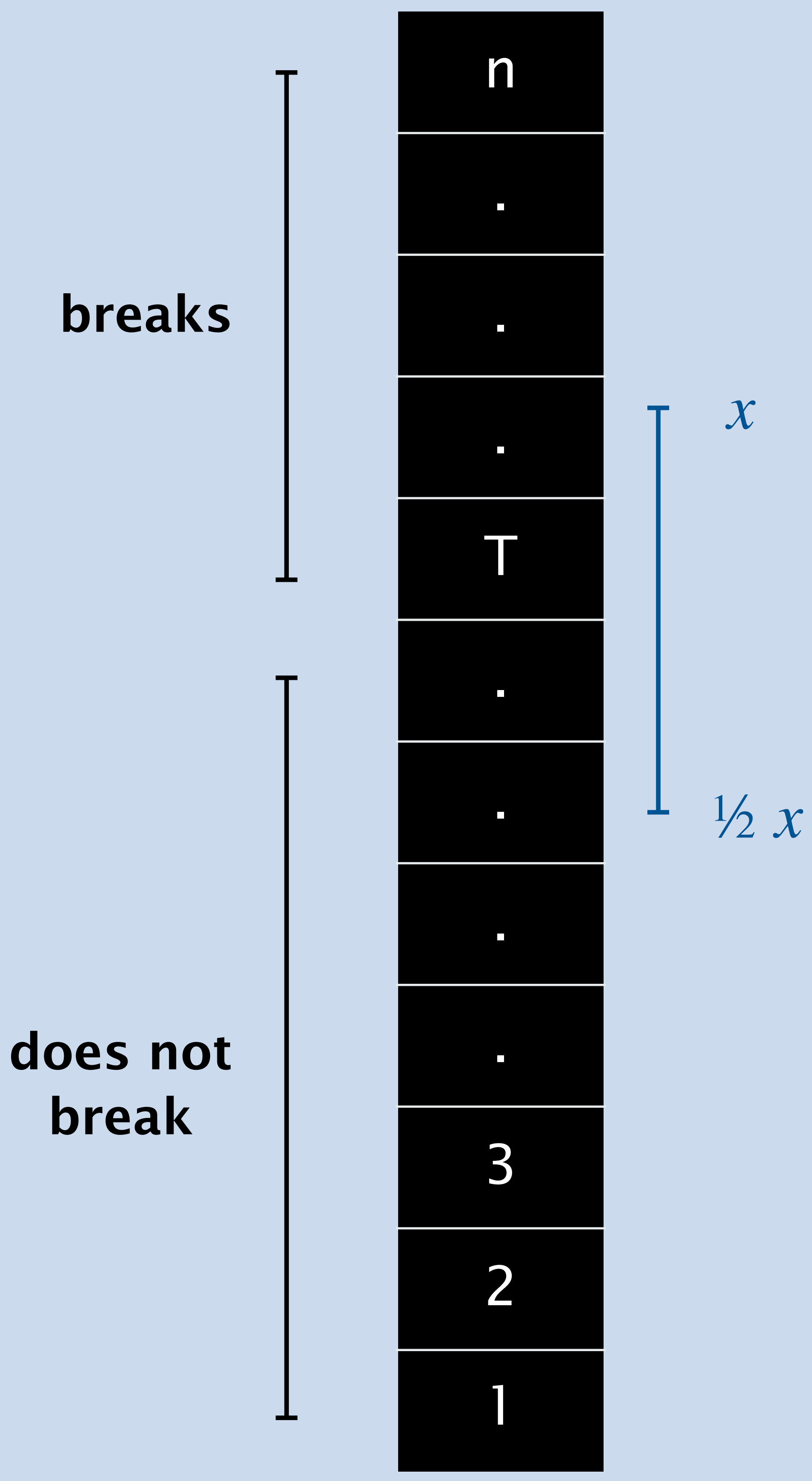
Variant 1′. ∞ eggs and $\Theta(\log T)$ drops.

Solution. Use repeated doubling; then binary search.

- Drop on floors $1, 2, 4, 8, 16, \ldots, x$ 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 $T \le x < 2T$.



Algorithm design: quiz 1

Goal. Find T using fewest drops.

Variant 2. 2 eggs.

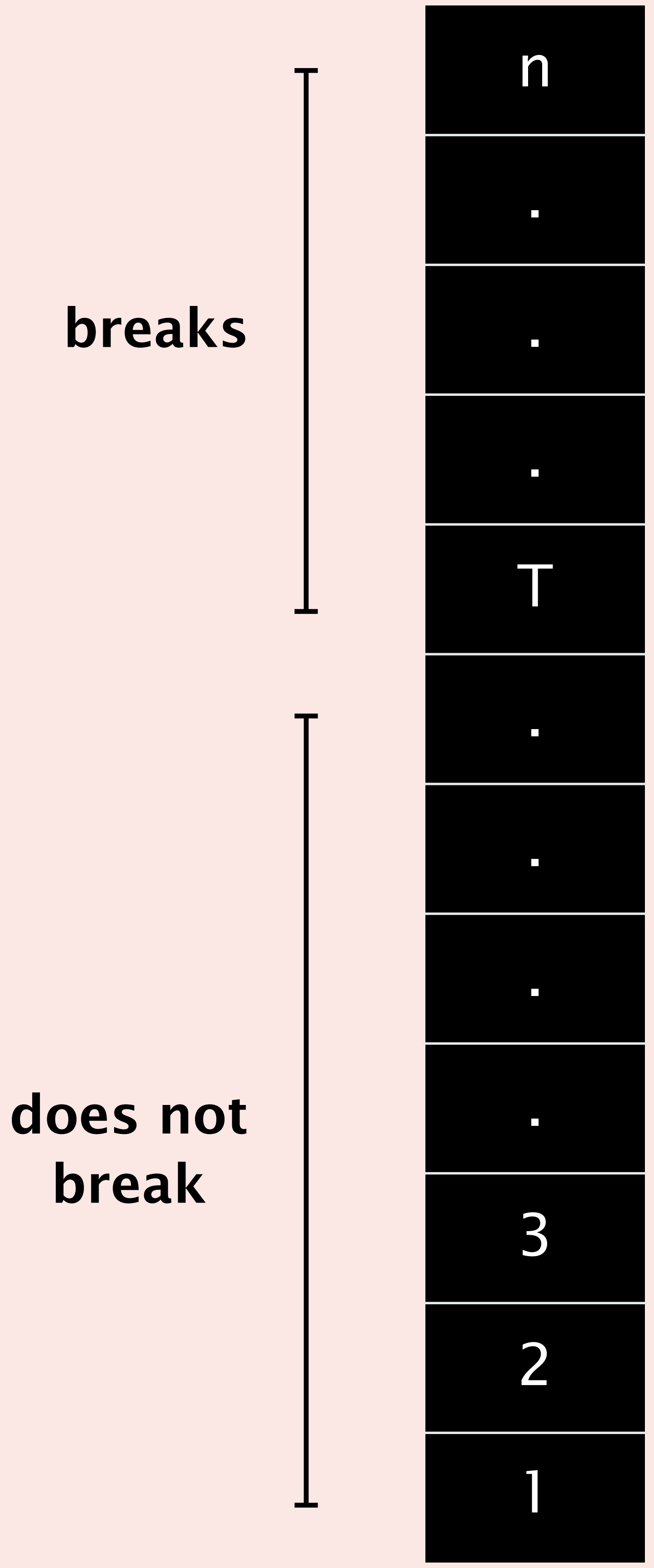
As a function of n, what is the fewest drops that an algorithm can guarantee?

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

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.

https://algs4.cs.princeton.edu

Algorithm Design

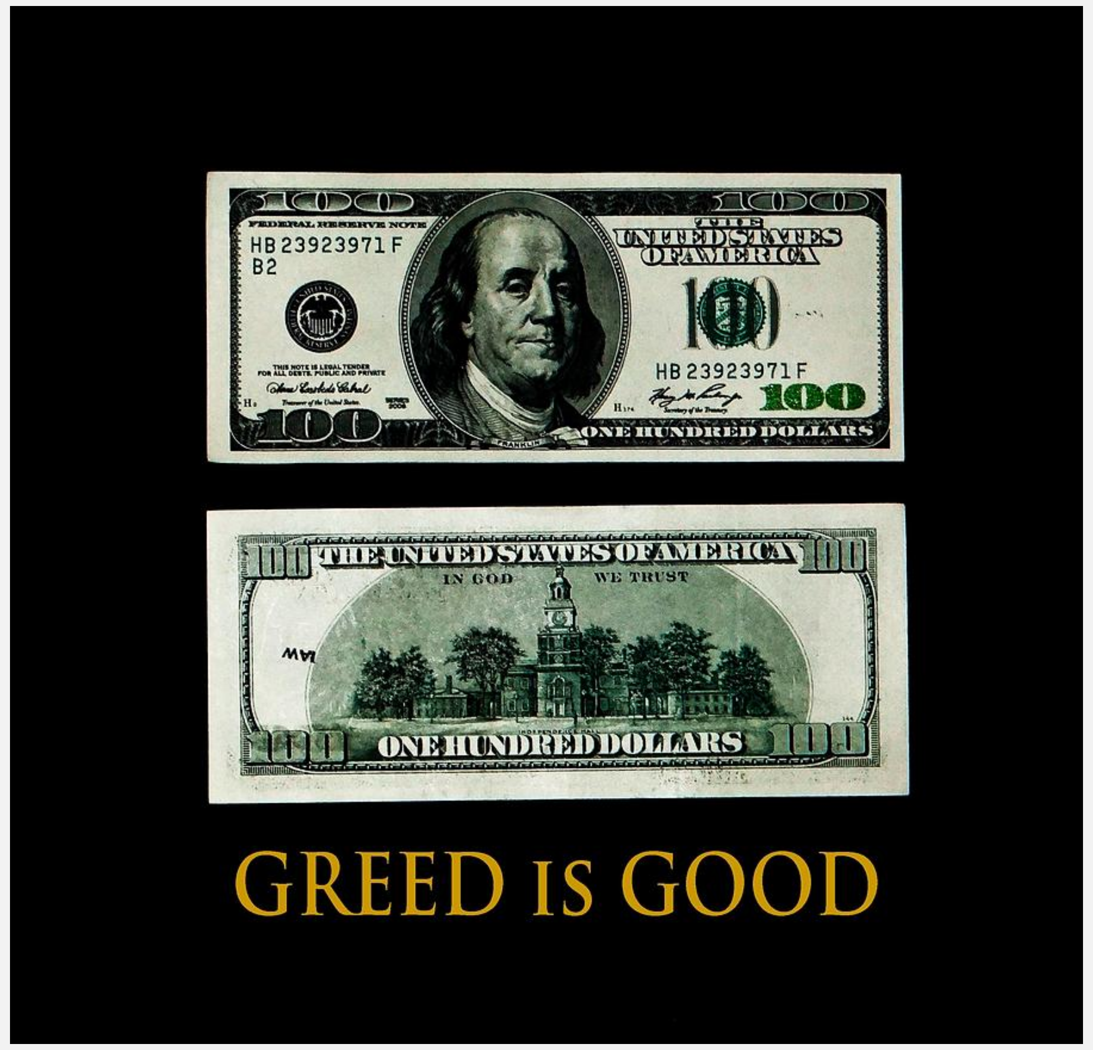
- *analysis of algorithms*
- ***greed***
- *reduction*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

Greedy algorithms

Make locally optimal 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 } ?

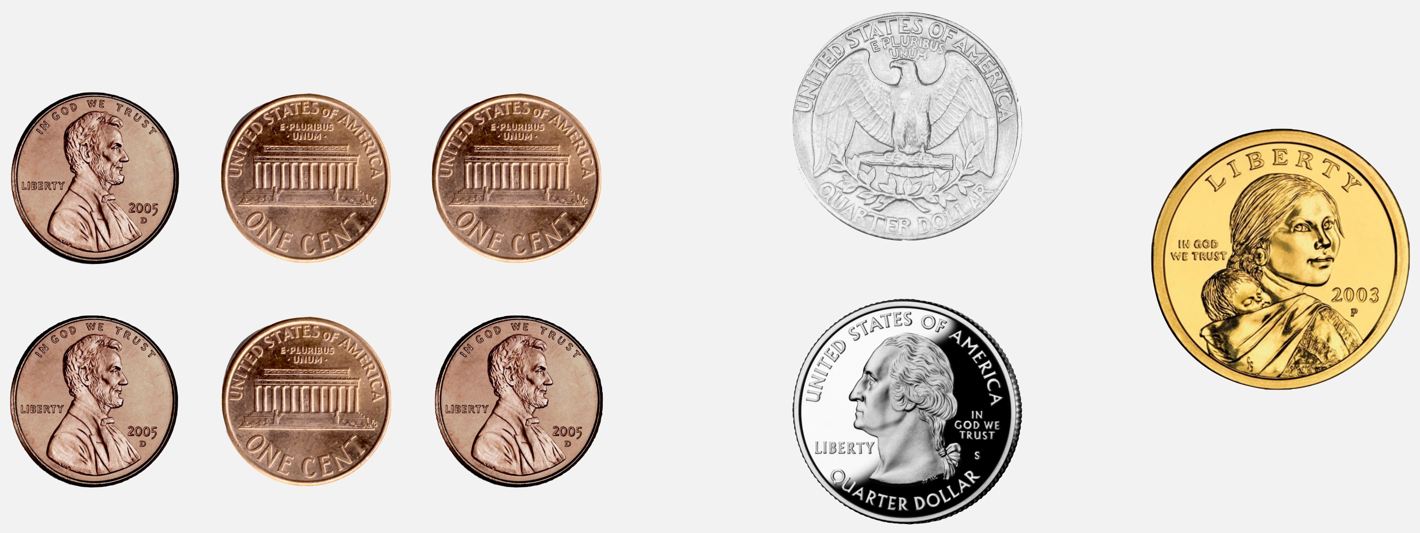
A. Yes, greedy algorithms are always optimal.

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

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

- P: P1 $\Rightarrow$ contributes at most 4
- $5N + 10D$: P5 $\Rightarrow$ contributes at most 20
- $25Q$: 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 ≥ 100¢
using only pennies, nickels, dimes, and quarters

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

https://algs4.cs.princeton.edu

Algorithm Design

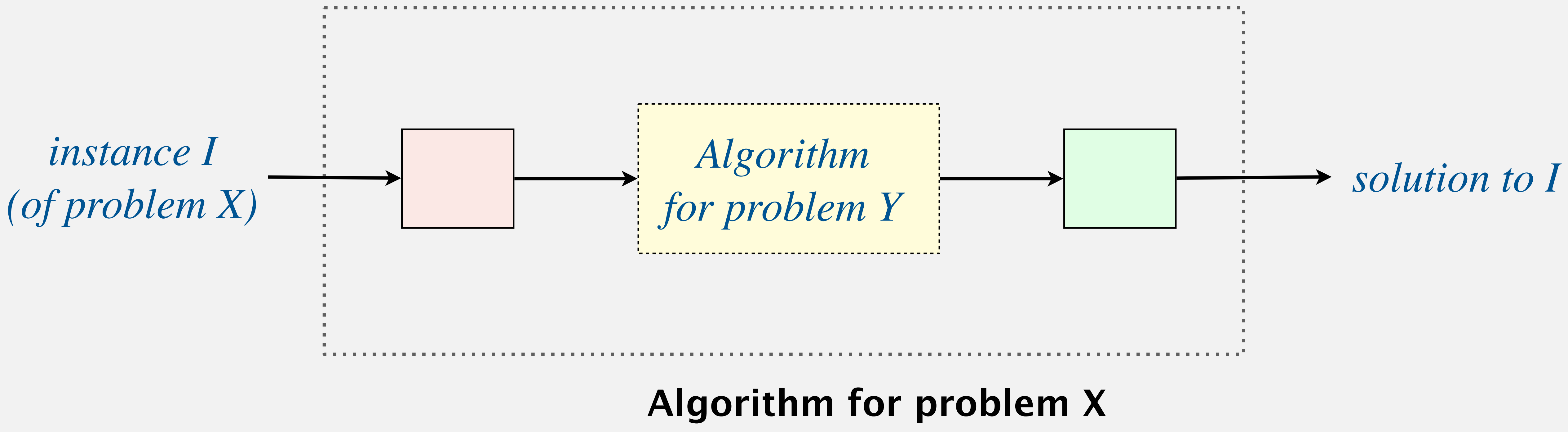
- *analysis of algorithms*
- *greed*
- ***reduction***
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

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. Min energy seam reduces to shortest paths.



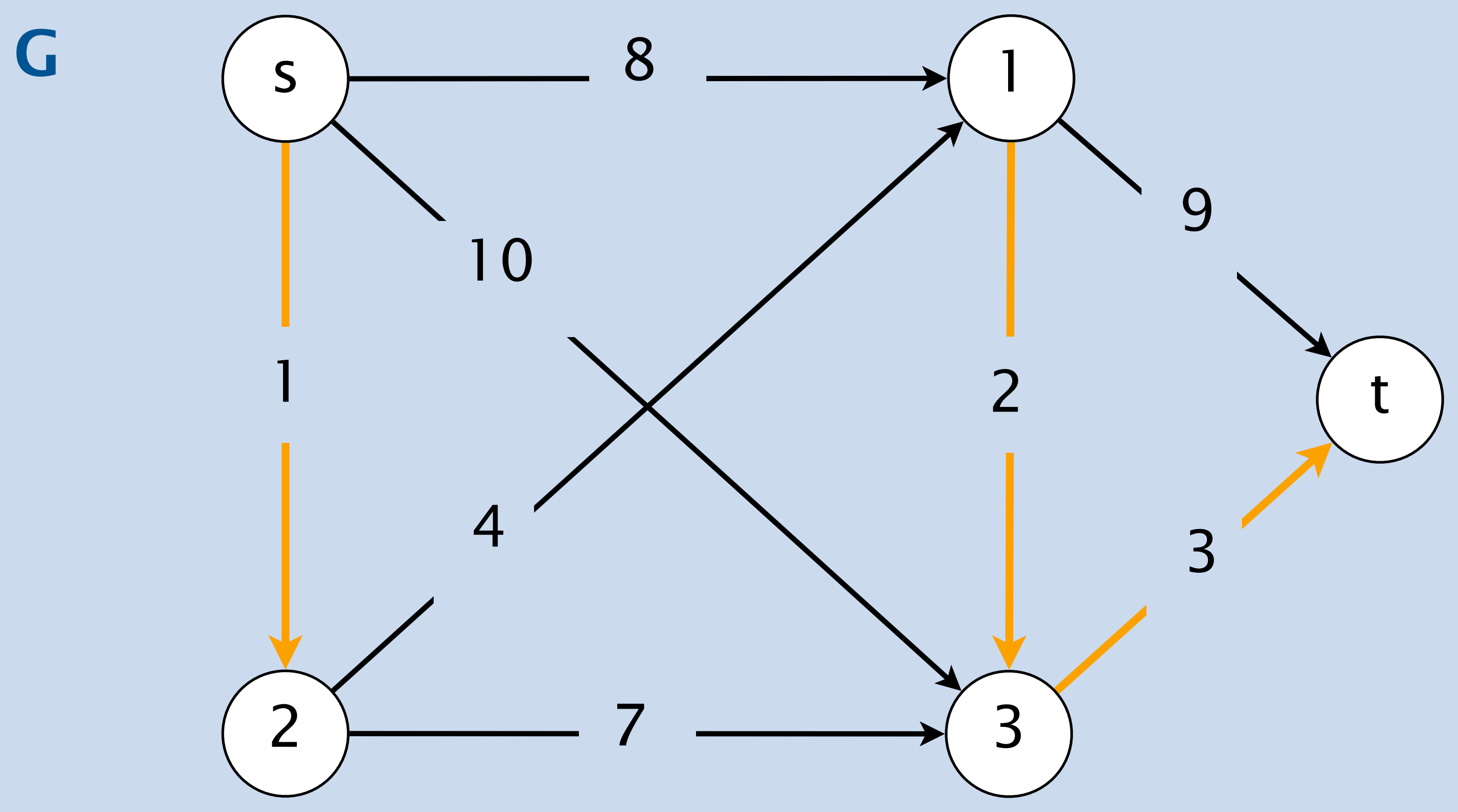
Many many problems reduce to:

- Sorting.
- Shortest paths.
- Suffix array.
- Minimum spanning tree.
- Maximum flow. ← see COS 423
- 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→1→t (17)

k = 1: s→3→t (13)

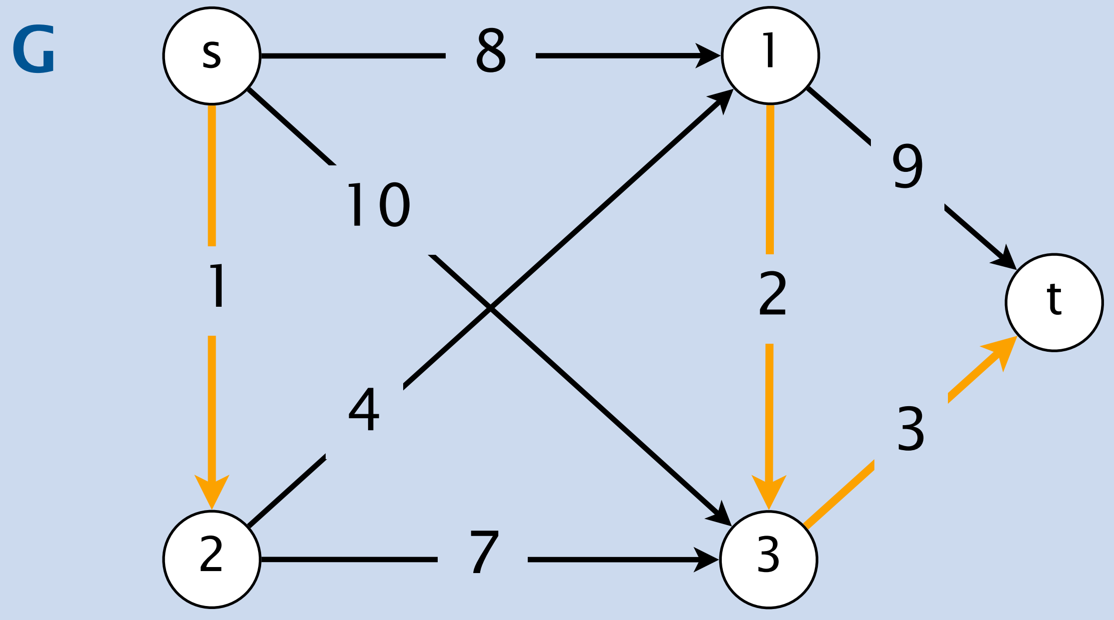
k = 2: s→2→3→t (11)

k = 3: s→2→1→3→t (10)

k = 4: s→2→1→3→t (10)

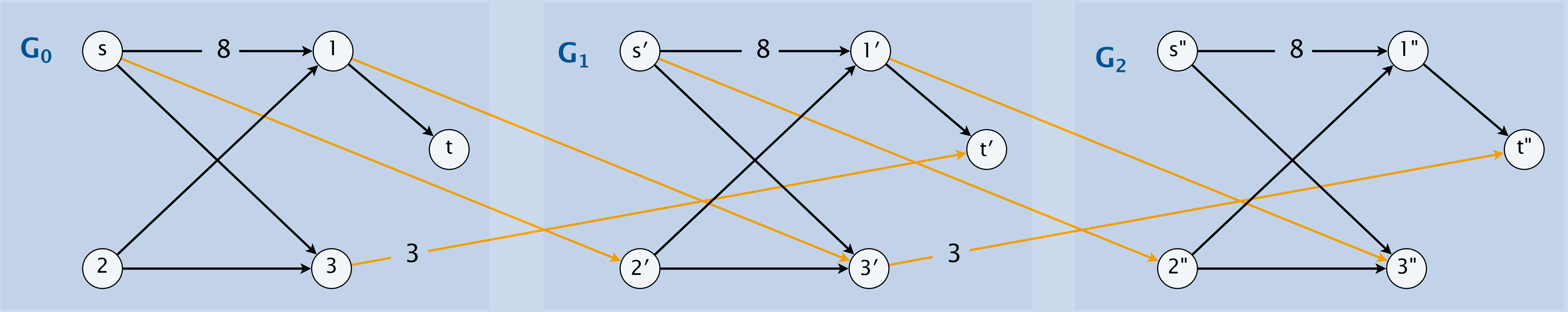
Shortest path with orange and black edges

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 redution to shortest paths:

- Create $k+1$ copies of the vertices in digraph G, labeled $G_0, G_1, \ldots, G_k$.
- For each black edge $v \to w$: add edge from vertex v in graph G_i to vertex w in G_i.
- For each orange edge $v \to w$: 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.



k = 2

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 ≥ V.

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

B. $\Theta(k\ E)$

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

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

https://algs4.cs.princeton.edu

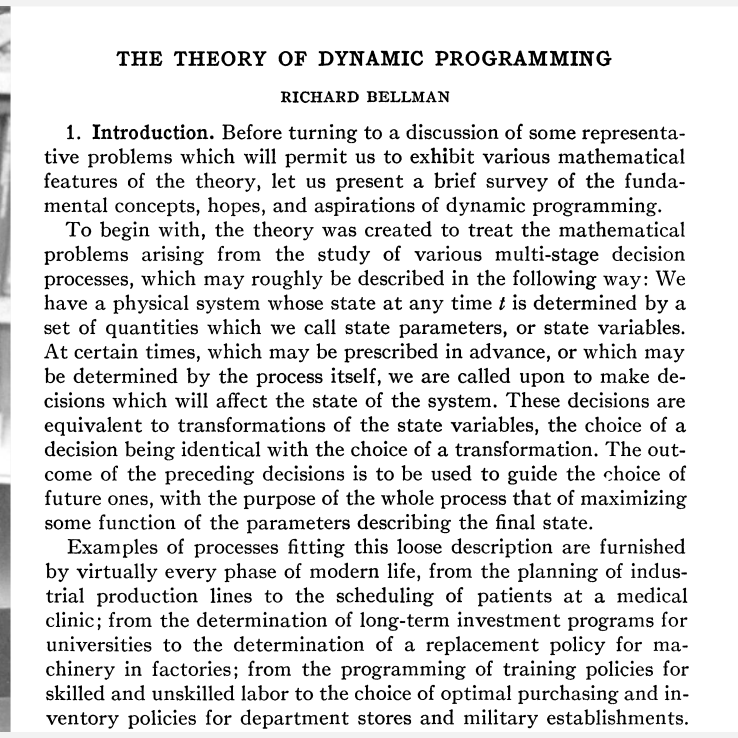
Algorithm Design

- analysis of algorithms
- greed
- 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]



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.

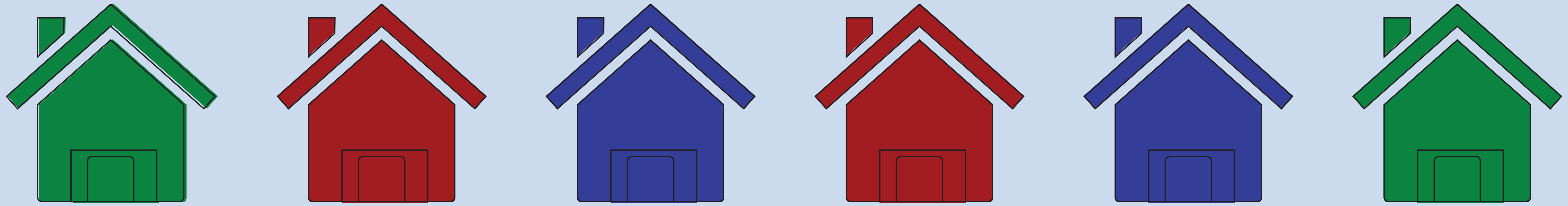
More classic examples.

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

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

Dynamic programming recurrence.

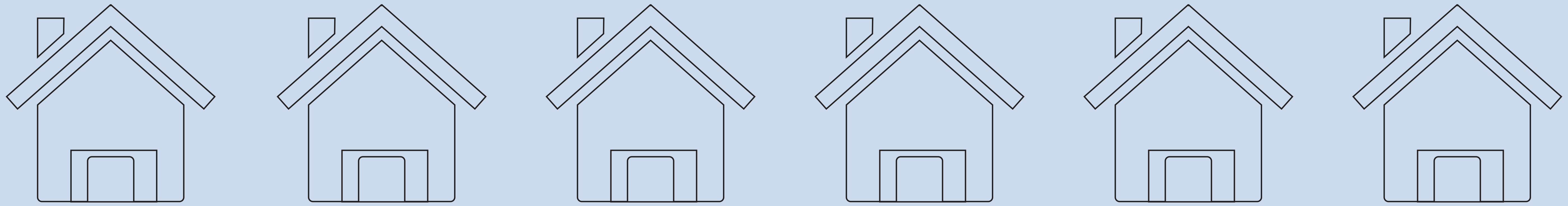
- $R(0) = G(0) = B(0) = 0$
- $R(i) = cost(i, red) + \min \{ G(i-1), B(i-1) \}$
- $G(i) = cost(i, green) + \min \{ B(i-1), R(i-1) \}$
- $B(i) = cost(i, blue) + \min \{ R(i-1), G(i-1) \}$

"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 $B(i+1)$.

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



	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]);
```

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

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

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

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

https://algs4.cs.princeton.edu

Algorithm Design

- analysis of algorithms
- greed
- 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.

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

Problem. 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 $\Theta(n^2)$ algorithm. For each $i < j$, check if $a_i > a_j$.

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

Goal. $\Theta(n \log n)$ time (or better).

Counting inversions: divide-and-conquer

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	1
count inversions in right subarray	0	3	4	7	9	1	2	5	6	8	3
count inversions with one element in each subarray	0	3	4	7	9	1	2	5	6	8	13
output	0	1	2	3	4	5	6	7	8	9	1 + 3 + 13 = 17

Left subarray inversions: 4–3

Right subarray inversions: 5–2 8–2 8–6

Inversions with one element in each subarray: 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

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

COUNTING INVERSIONS: DIVIDE-AND-CONQUER

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	

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.** $\Theta(n^2)$

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.

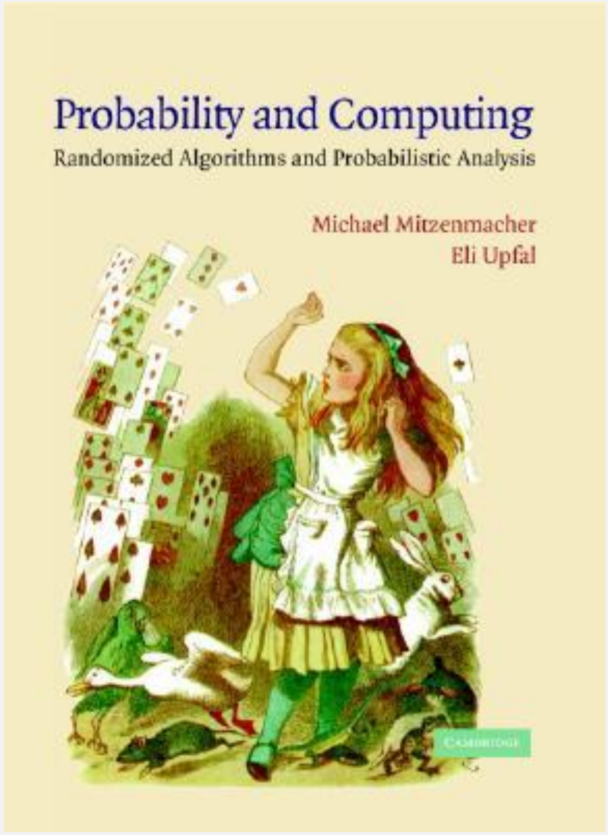
Familiar examples.

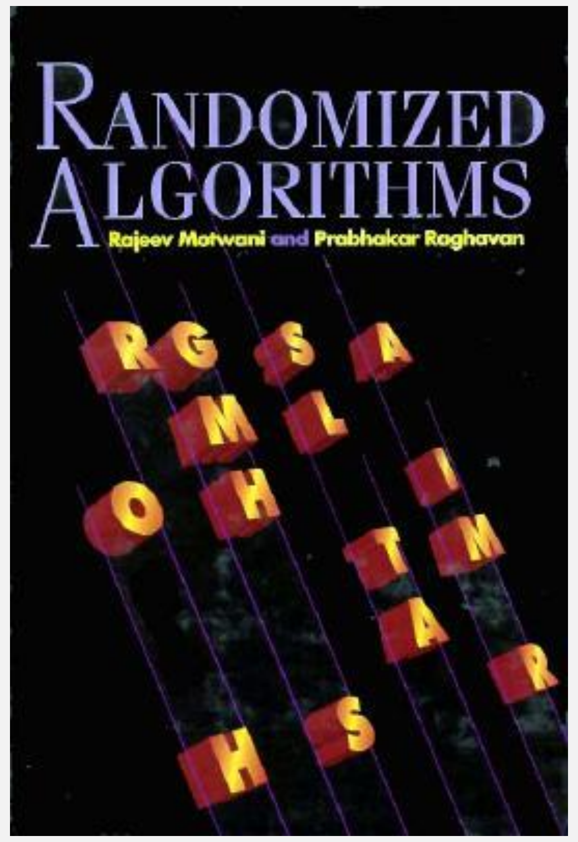
- Quicksort.
- Quickselect.

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'.

	smaller bolts						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						larger nuts			

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. $\Theta(n^2)$

Nuts and bolts

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 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.

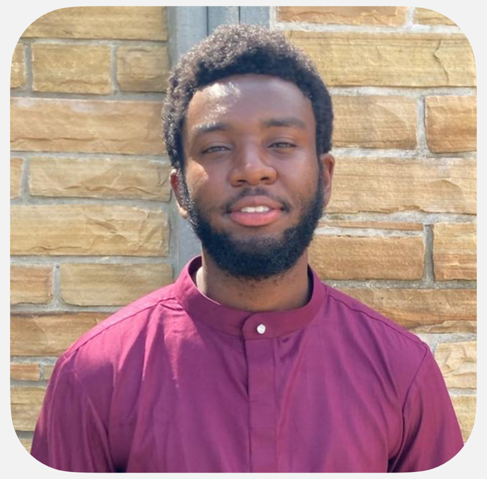
https://algs4.cs.princeton.edu

Algorithm Design

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

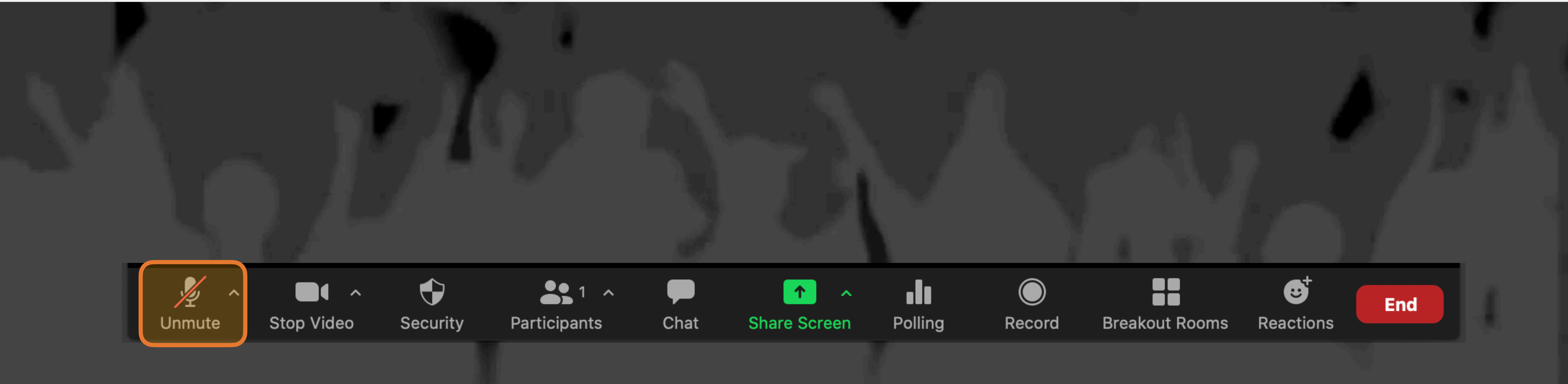
Credits

Co-instructors and graduate student AIs.

Precept facilitators. Alex, Alison, Andrew, Bryan, Dwaipayan, Harvey, Kartik, Liam, Niva, Saumya.

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



A farewell video (from P04, Fall 2018)

A final thought

> *" Algorithms and data structures are love.*
>
> *Algorithms and data structures are life. "*
>
> *— anonymous COS 226 student*