https://algs4.cs.princeton.edu

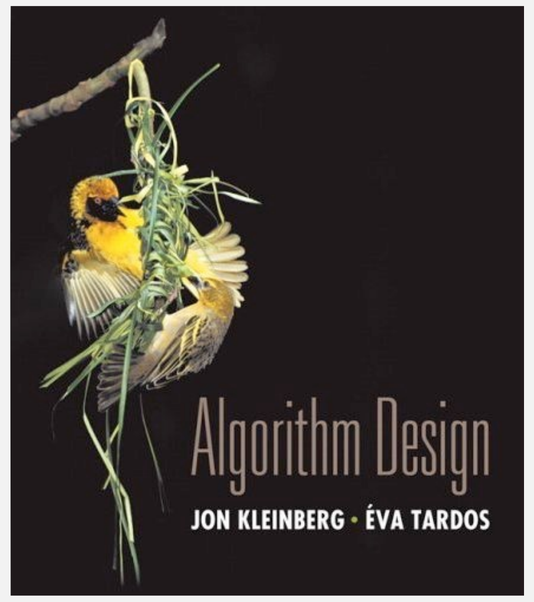
Algorithm Design

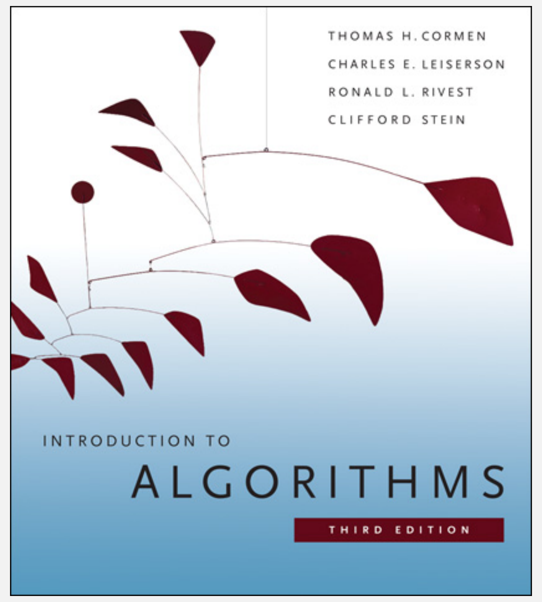
- *analysis of algorithms*
- *greed*
- *network flow*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

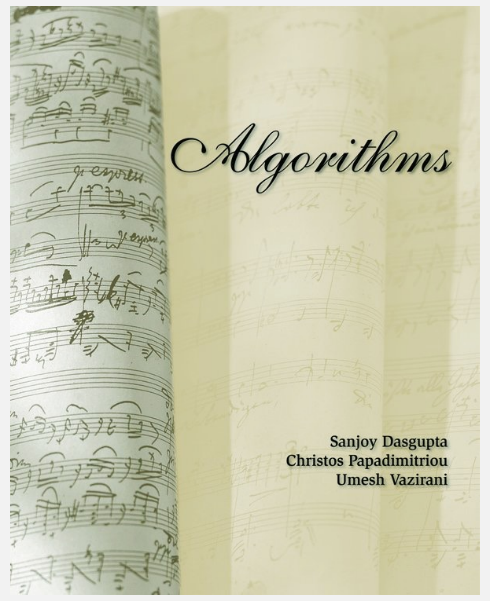
Algorithm design

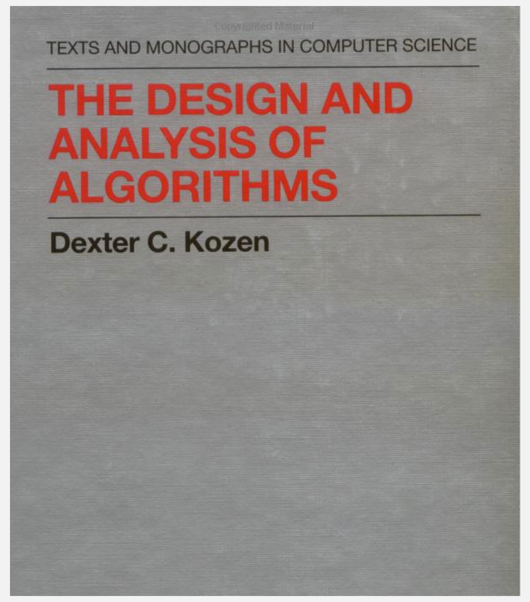
Algorithm design patterns.

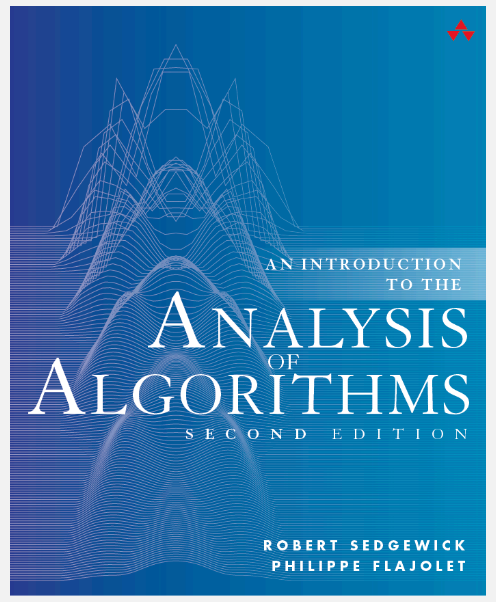
- Analysis of algorithms.
- Greed.
- Network flow.
- Dynamic programming.
- Divide-and-conquer.
- Randomization.

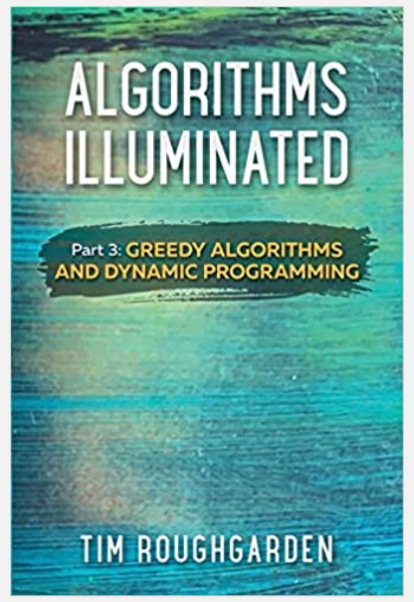












Want more? See COS 340, COS 343, COS 423, COS 445, COS 451, COS 488, ...

INTERVIEW QUESTIONS

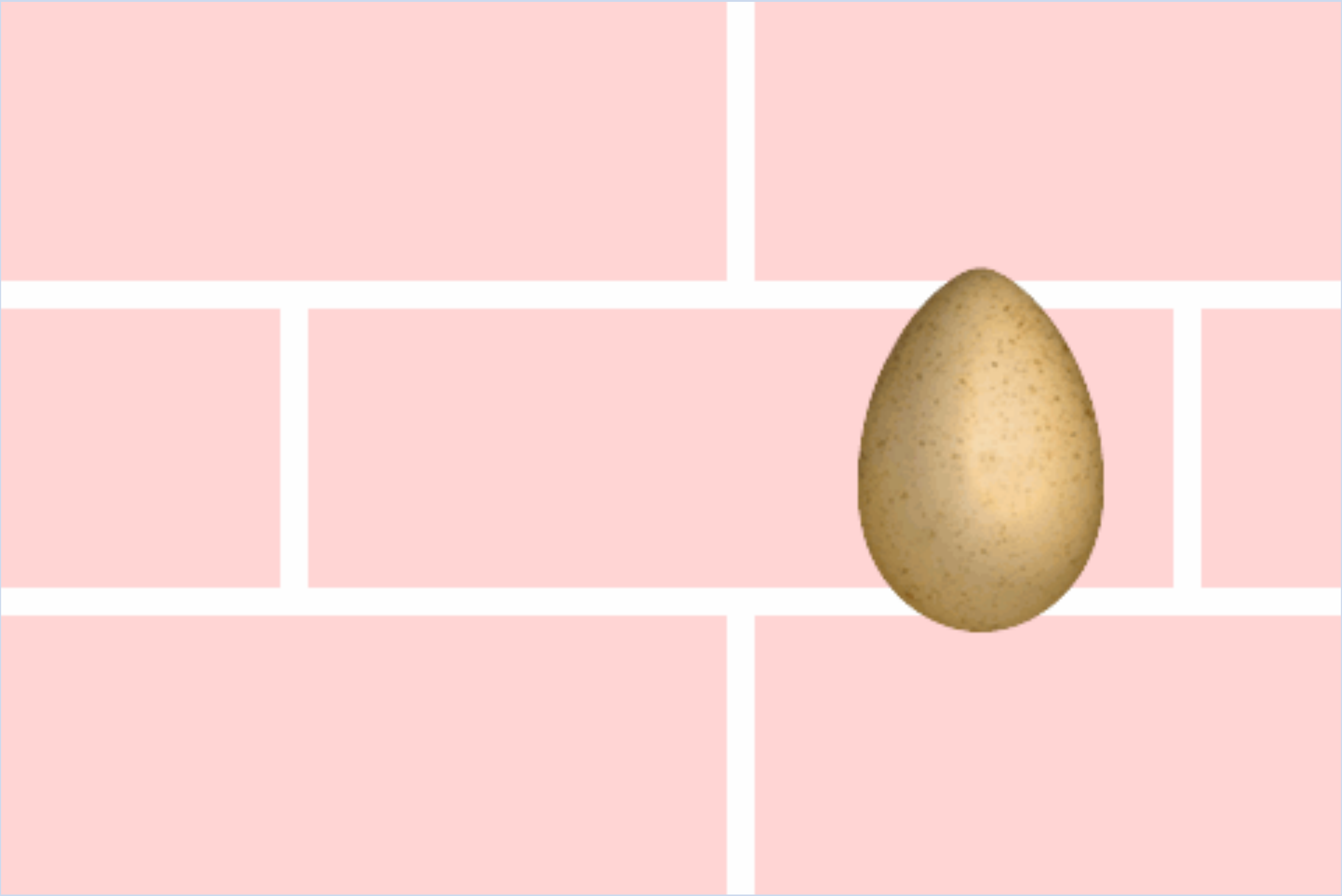
https://algs4.cs.princeton.edu

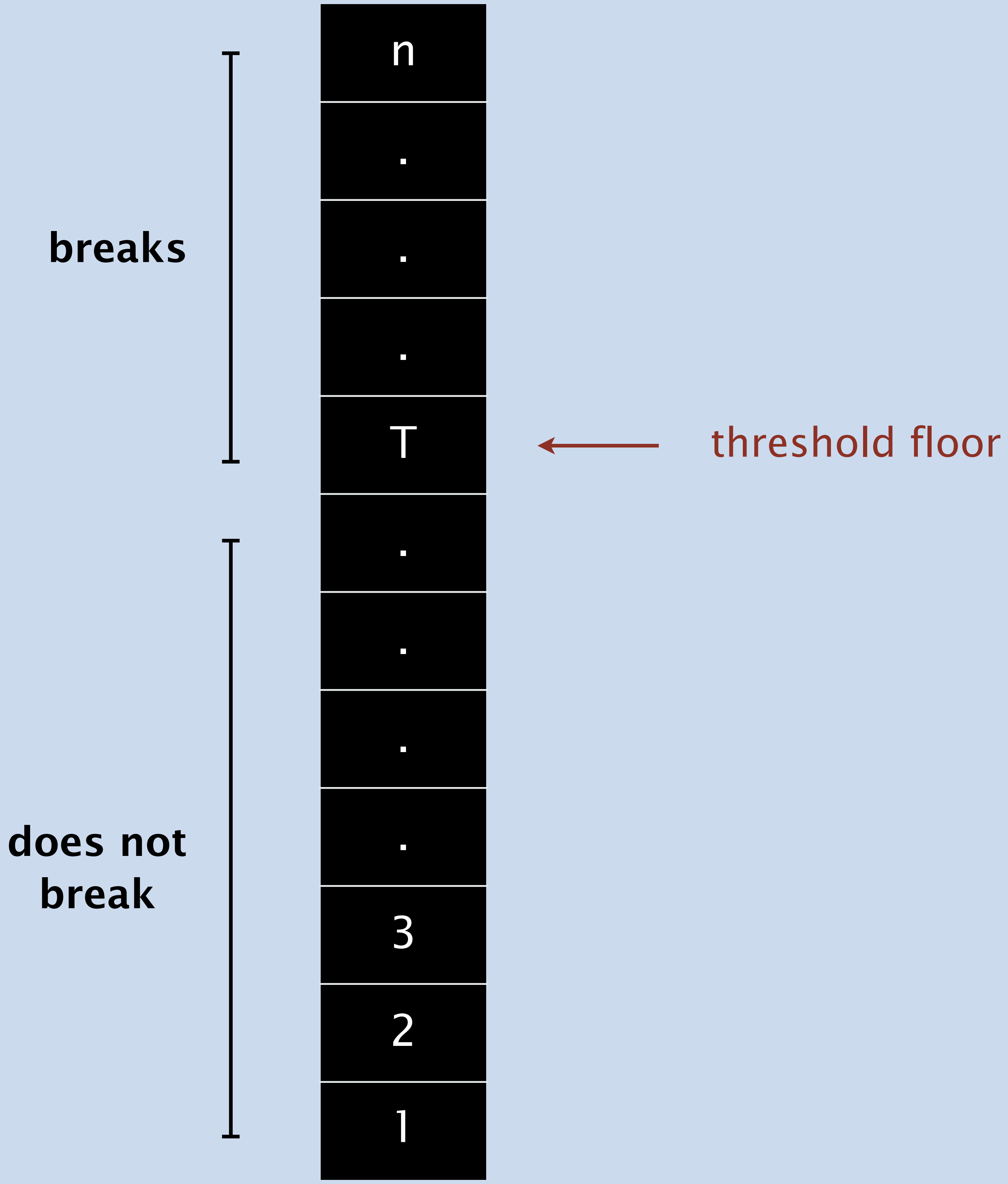
Algorithm Design

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

EGG DROP

Goal. Find T using fewest drops.





Egg drop

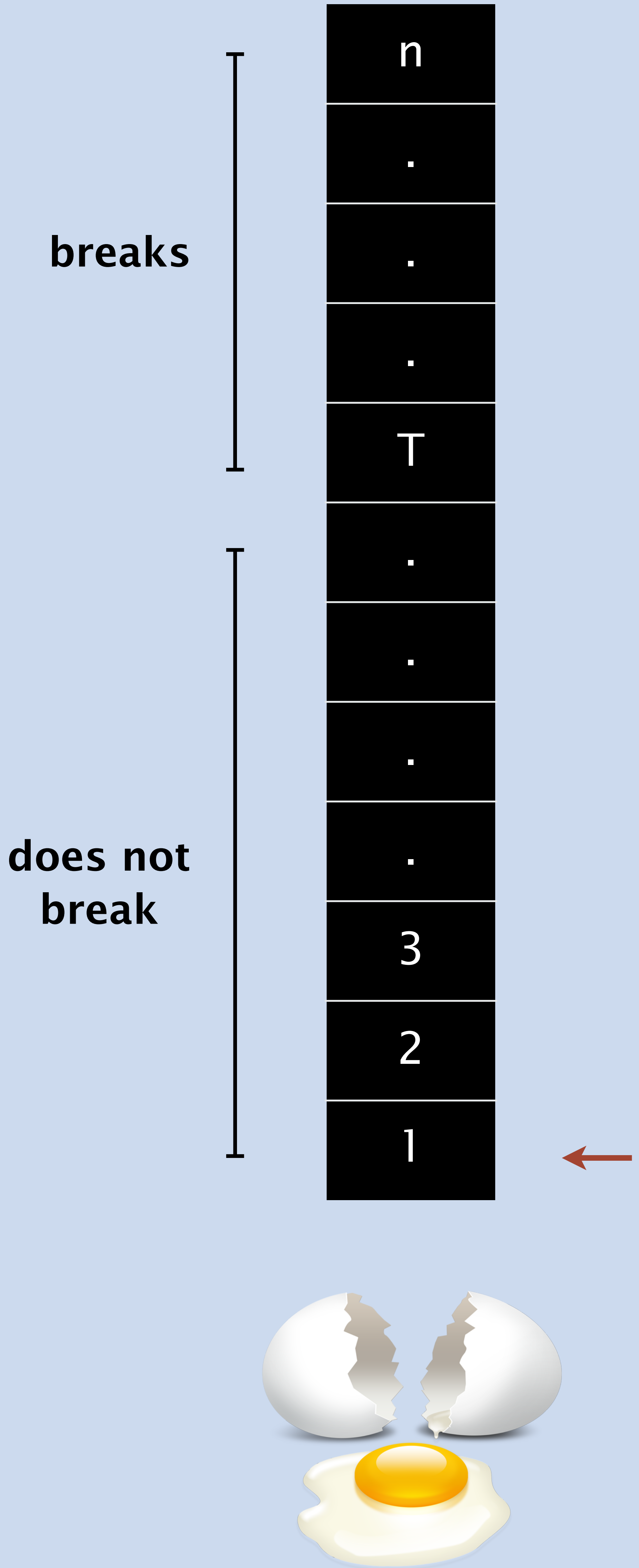
Goal. Find T using fewest drops.

Variant 0. 1 egg.

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

Analysis. 1 egg and T drops.

running time depends upon
a parameter that you don't know a priori



EGG DROP

Goal. Find T using fewest drops.

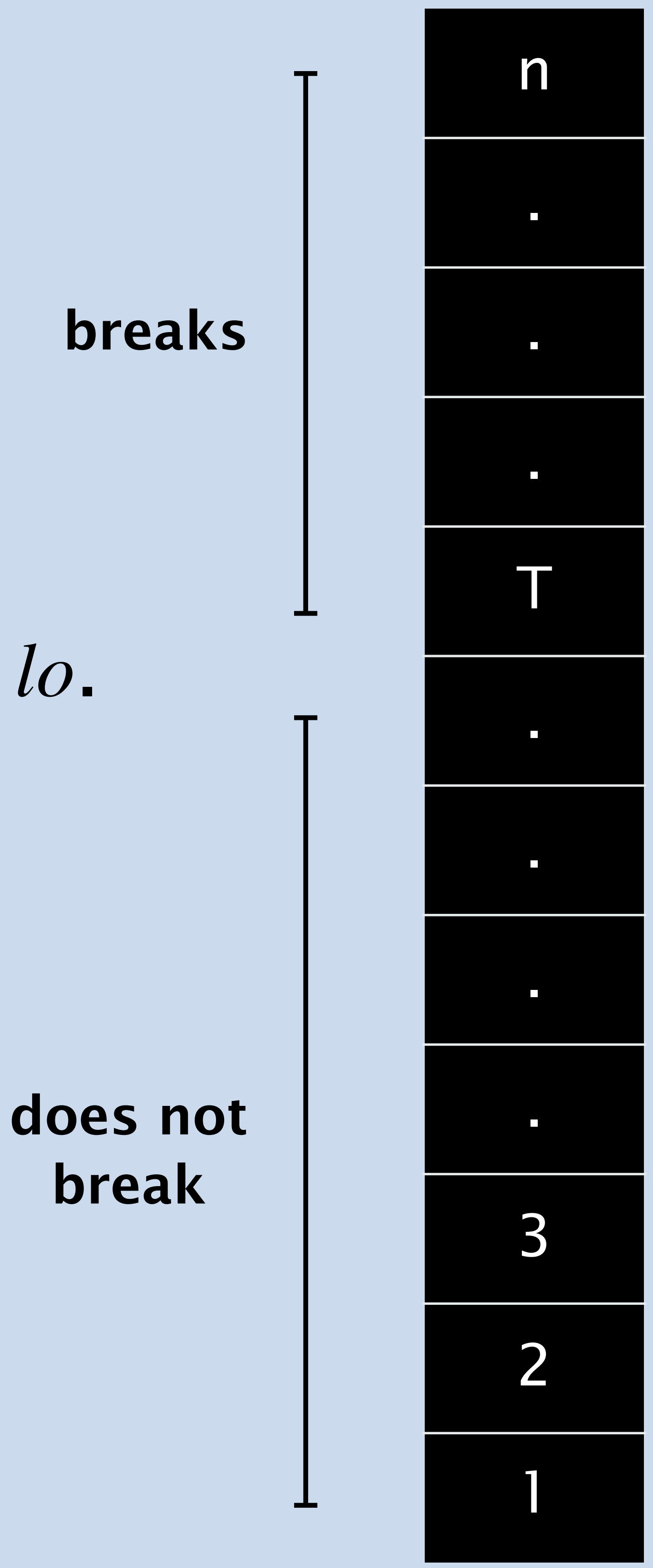
Variant 1. ∞ eggs.

Solution. Binary search for T.

- Initialize $[\text{lo}, \ hi] \ = \ [0, \ n{+}1]$.
- Maintain invariant: egg breaks on floor hi but not on lo.
- Repeat until length of interval is 1:
  - drop on floor $mid = (lo + hi) \, / \, 2$.
  - if it breaks, update $hi = mid$.
  - if it doesn't break, 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.

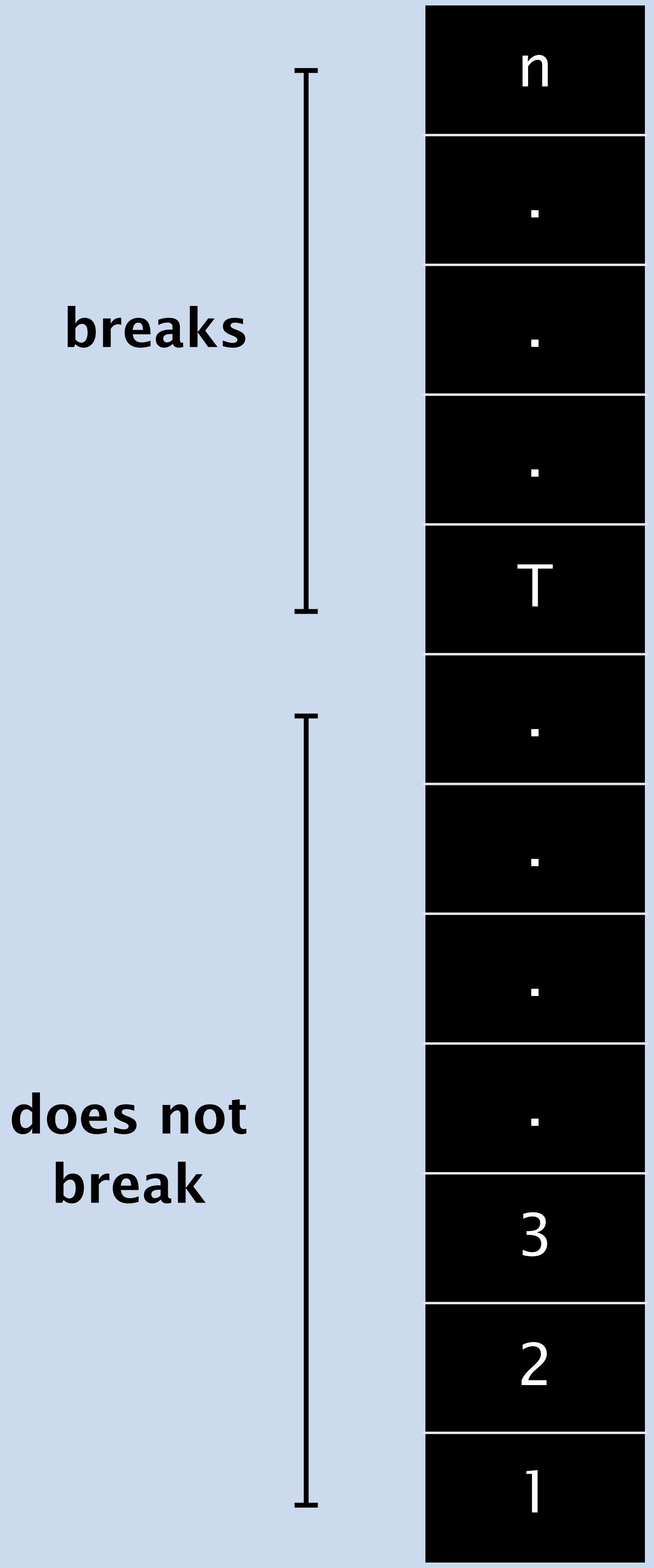
Variant 1′. ∞ eggs and $\Theta(\log T)$ drops.

Solution. Use repeated doubling; then binary search.

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



Algorithm design: quiz 1

Goal. Find T using fewest drops.

Variant 2. 2 eggs.

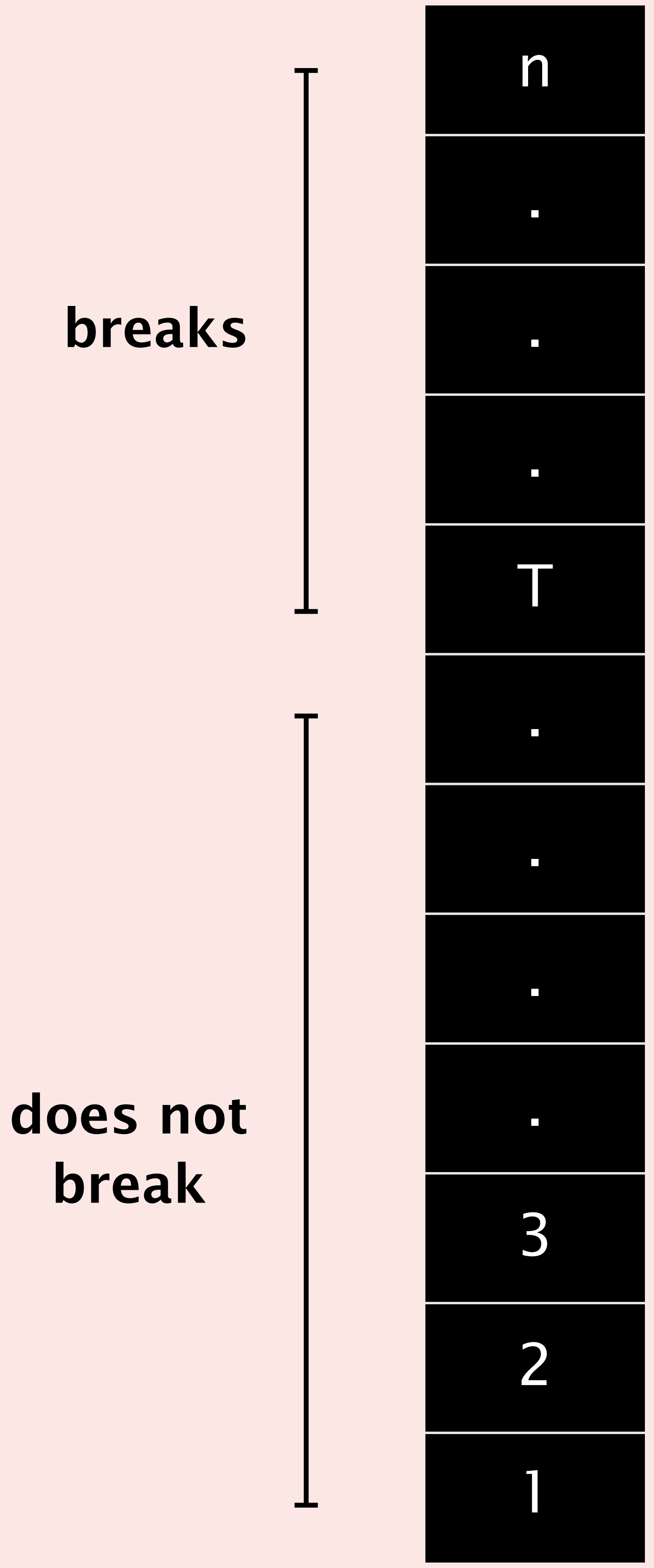
In worst case, how many drops needed as a function of n?

A. $\Theta(1)$

B. $\Theta(\log n)$

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

D. $\Theta(n)$



EGG DROP (ASYMMETRIC SEARCH)

Goal. Find T using fewest drops.

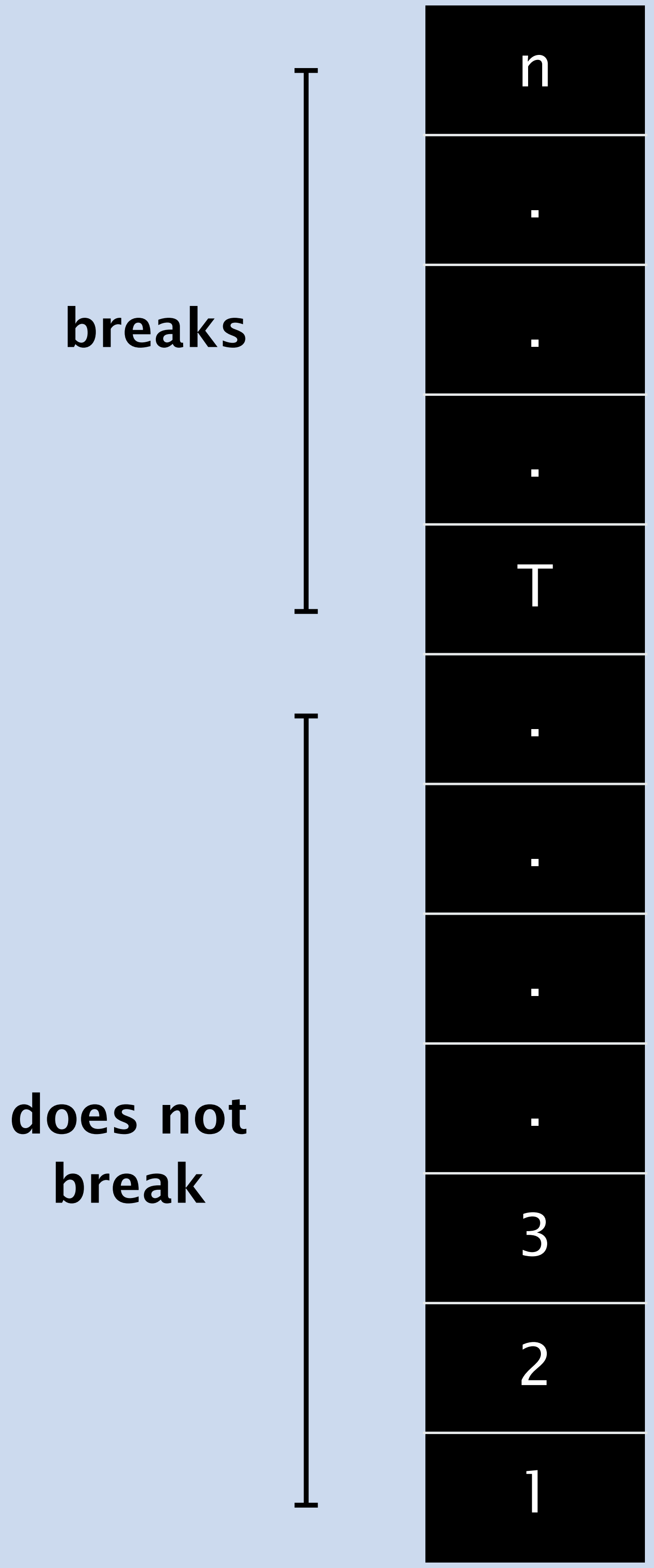
Variant 2. 2 eggs.

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 3 eggs and at most $3\, n^{1/3}$ drops.

https://algs4.cs.princeton.edu

Algorithm Design

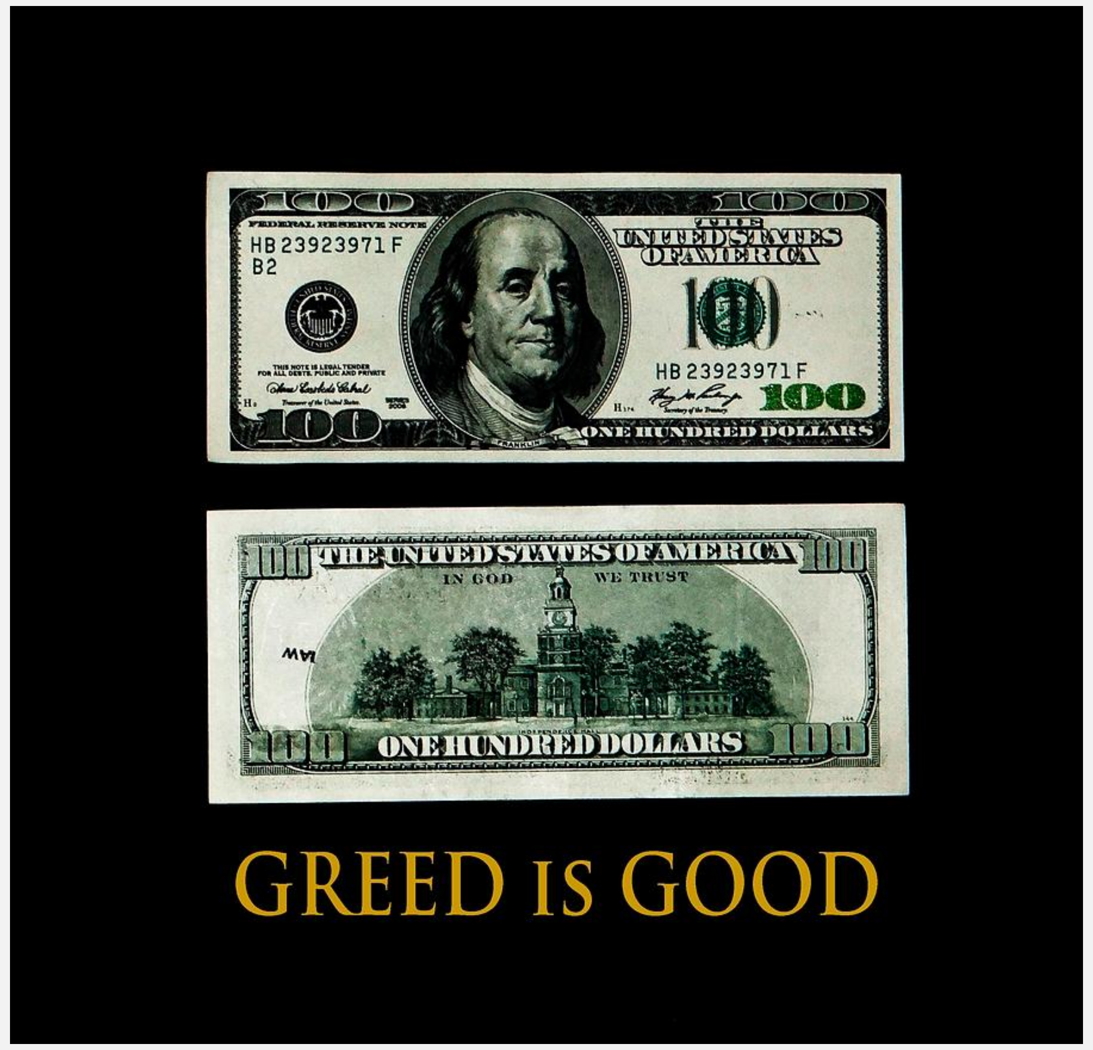
- *analysis of algorithms*
- *greed*
- *network flow*
- *dynamic programming*
- *divide-and-conquer*
- *randomization*

Greedy algorithms

Make locally optimal choices at each step.

Familiar examples.

- Huffman coding.
- Prim's algorithm.
- Kruskal's algorithm.
- Dijkstra's algorithm.



More classic examples.

- Activity scheduling.
- A* search algorithm.
- Gale–Shapley stable marriage.
- ...

Caveat. Greedy algorithm rarely leads to globally optimal solution.

(but is often used anyway, 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

Algorithm design: quiz 2

Is the cashier's algorithm optimal for U.S. coin denominations?

A. Yes, greedy algorithms are always optimal.

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

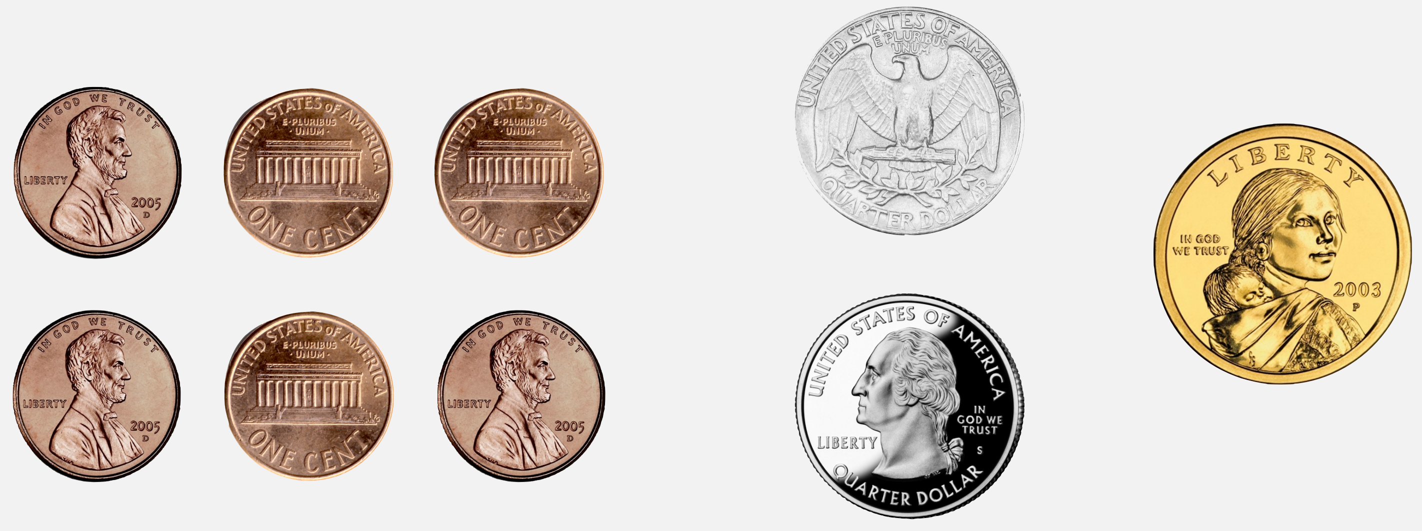
C. Yes, because of special properties of U.S. coin denominations.

D. No.

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

Property 1. Number of pennies $P \le 4$.
Pf. Replace 5 pennies with 1 nickel.

exchange argument



Property 2. Number of nickels $N \le 1$.
Property 3. Number of dimes $D \le 2$.
Property 4. Number of quarters $Q \le 3$.

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

- Properties 2 and 3: $N \le 1$ and $D \le 2$.
- Replace 2 dimes and 1 nickel with 1 quarter.

Property 6. $\underset{\text{contributes at most 4}}{P} + \underbrace{5N + 10D}_{\text{contributes at most 20}} + \underbrace{25Q}_{\text{contributes at most 75}} \le 99.$

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 optimal solution does not take x dollar coins.
- Then, optimal solution satisfies $P + 5\,N + 10\,D + 25\,Q \;\geq\; 100$.
- This contradicts Property 6.

must make change for ≥ 100¢
using only pennies, nickels, dimes, and quarters

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

https://algs4.cs.princeton.edu

Algorithm Design

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

Network flow

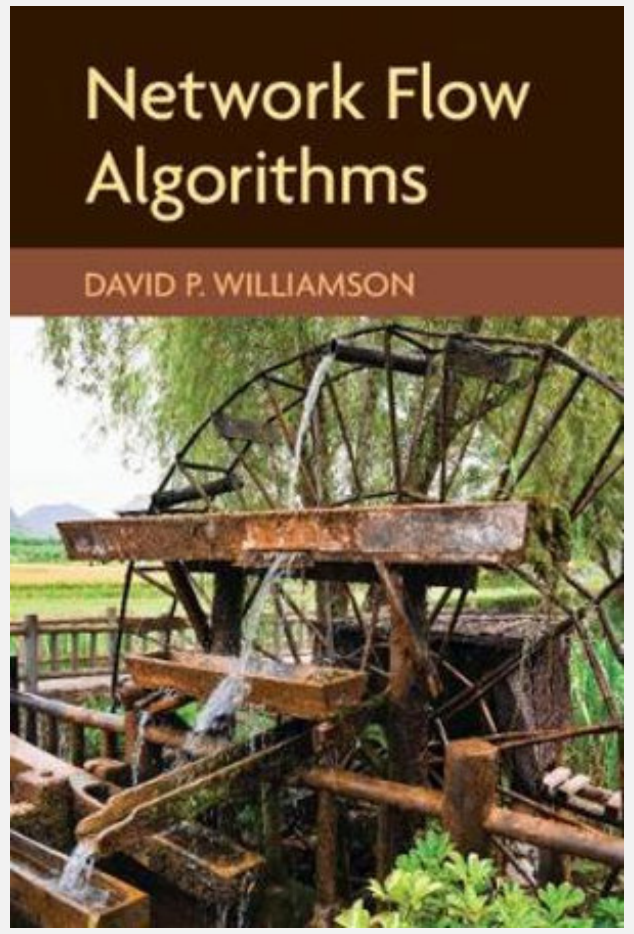
Fundamental problems on edge-weighted graphs and digraphs.

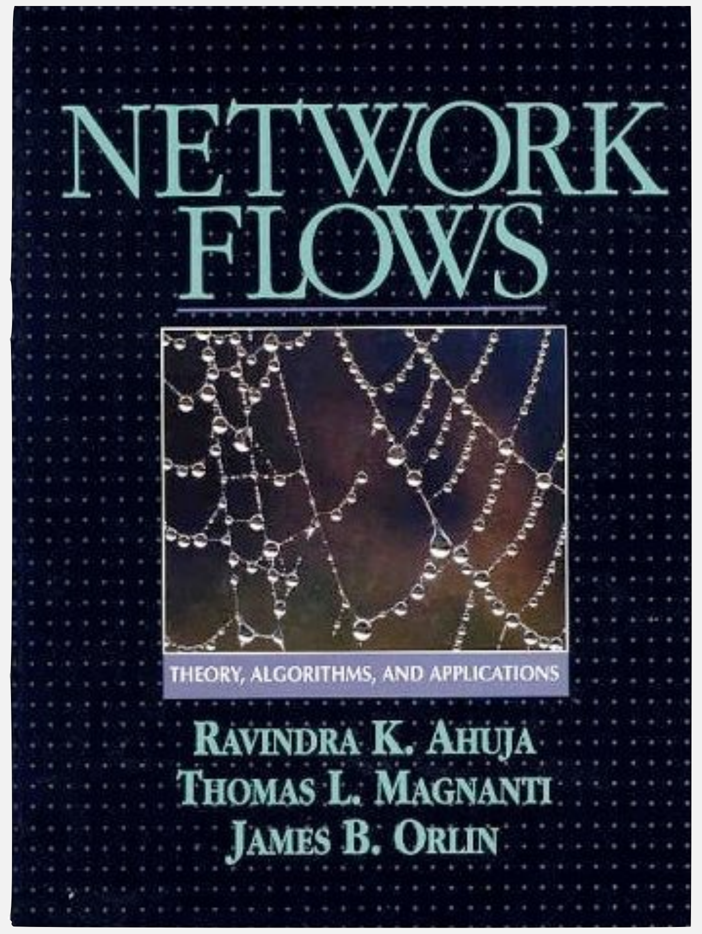
Familiar examples.

- Shortest paths.
- Bipartite matching.
- Maxflow and mincut.
- Minimum spanning tree.

Other classic examples.

- Minimum-cost flow.
- Assignment problem.
- Non-bipartite matching.
- Minimum-cost arborescence.
- ...



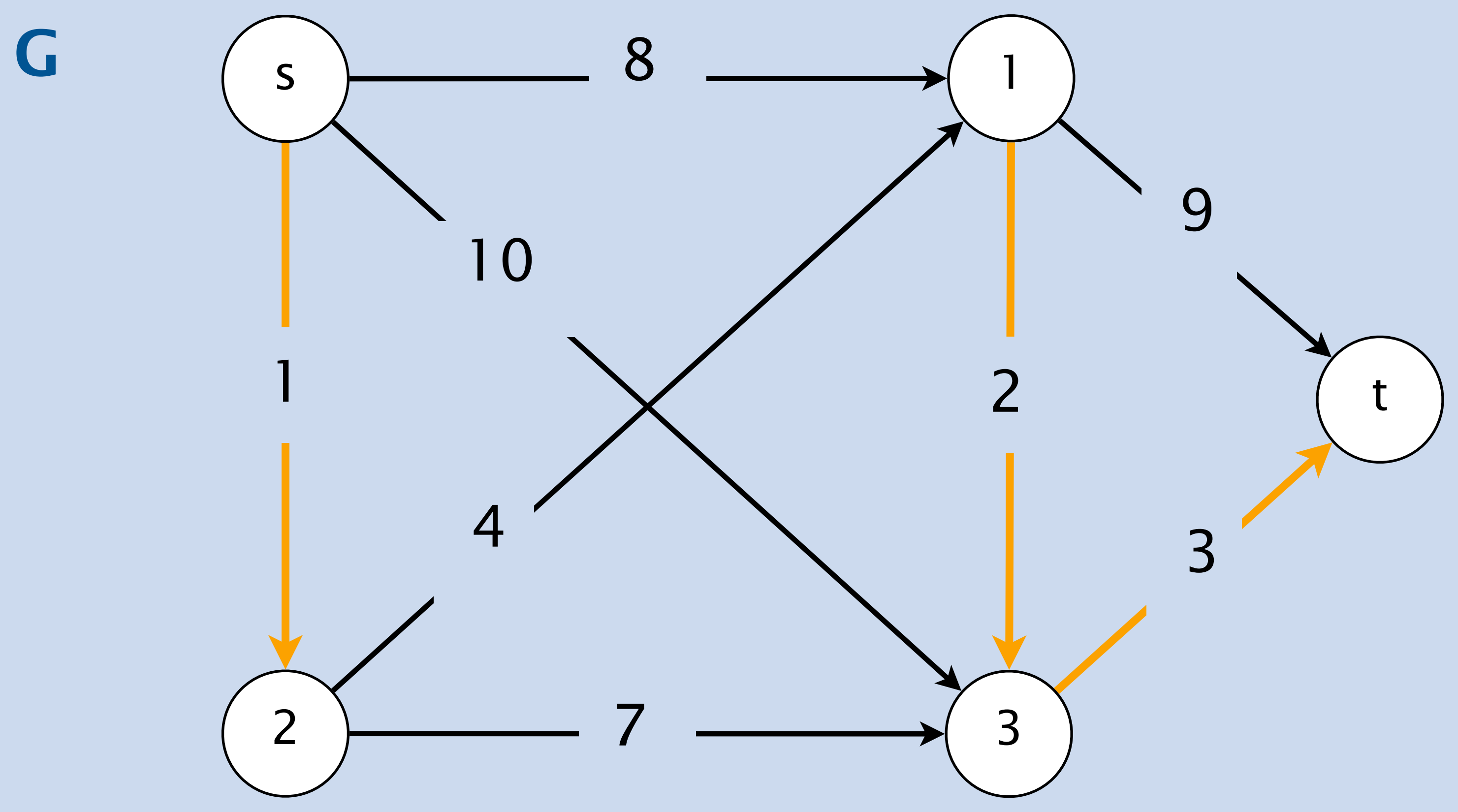


"reduction"

Applications. Many many problems can be modeled using network flow.

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)

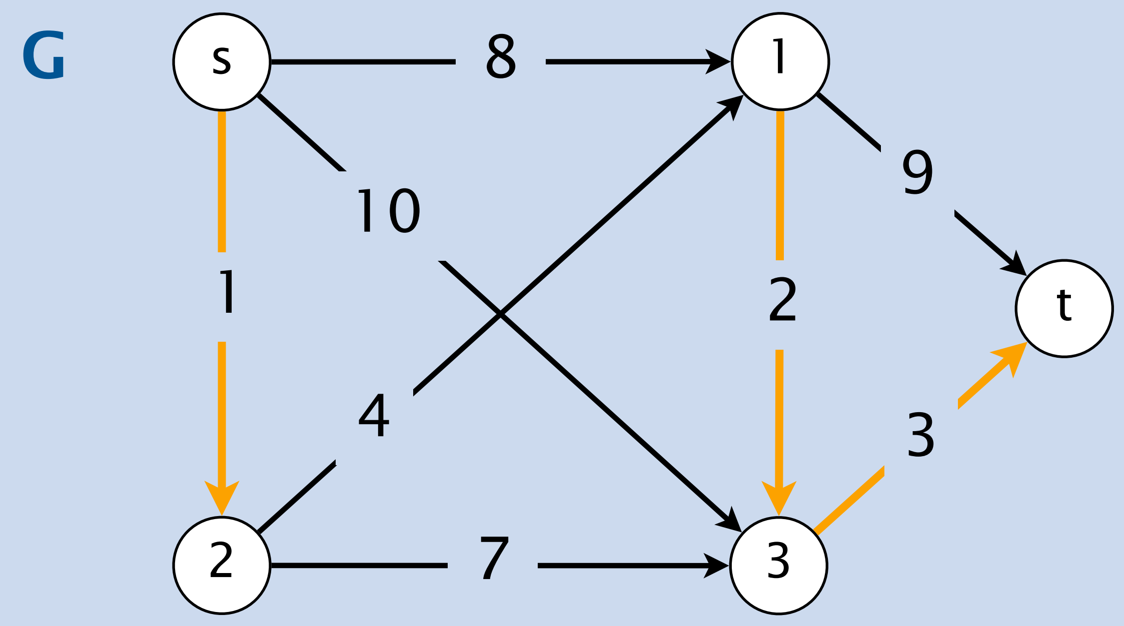
k = 2: s→2→3→t (11)

k = 3: s→2→1→3→t (10)

k = 4: s→2→1→3→t (10)

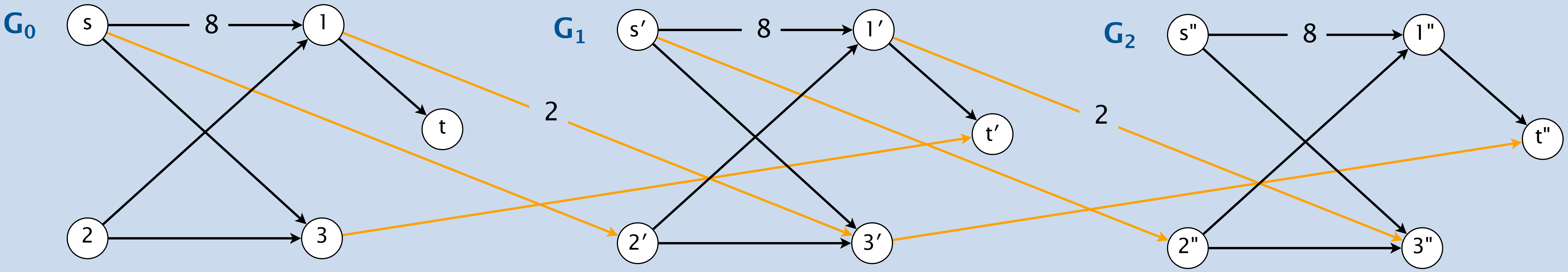
Shortest path with orange and black edges

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.



Solution.

- Create k+1 copies of the vertices in digraph G, labeled $G_0, G_1, \ldots, G_k$.
- 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.



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
- network flow
- *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.

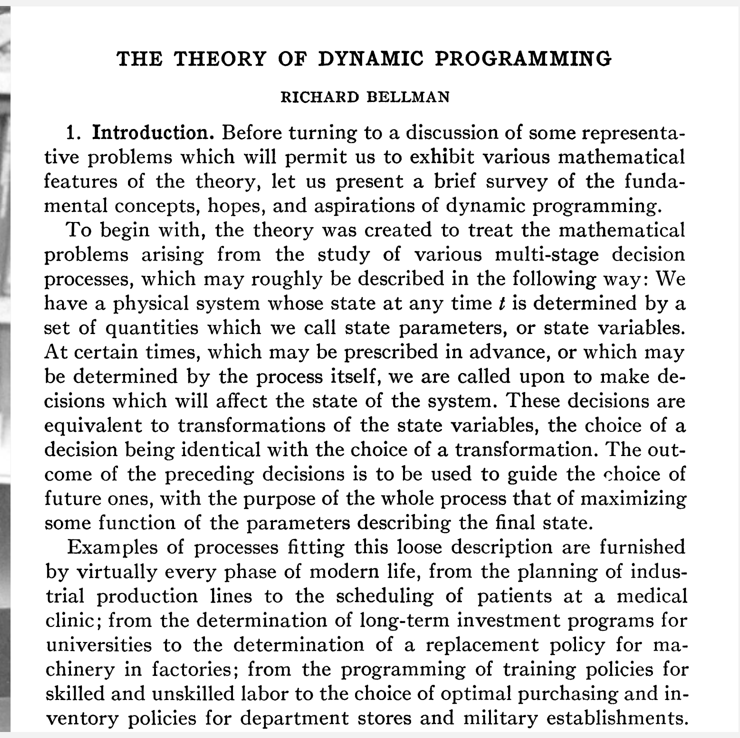
  (caching solutions to subproblems in a table for later reuse)

Familiar examples.

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

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.
- ...



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.

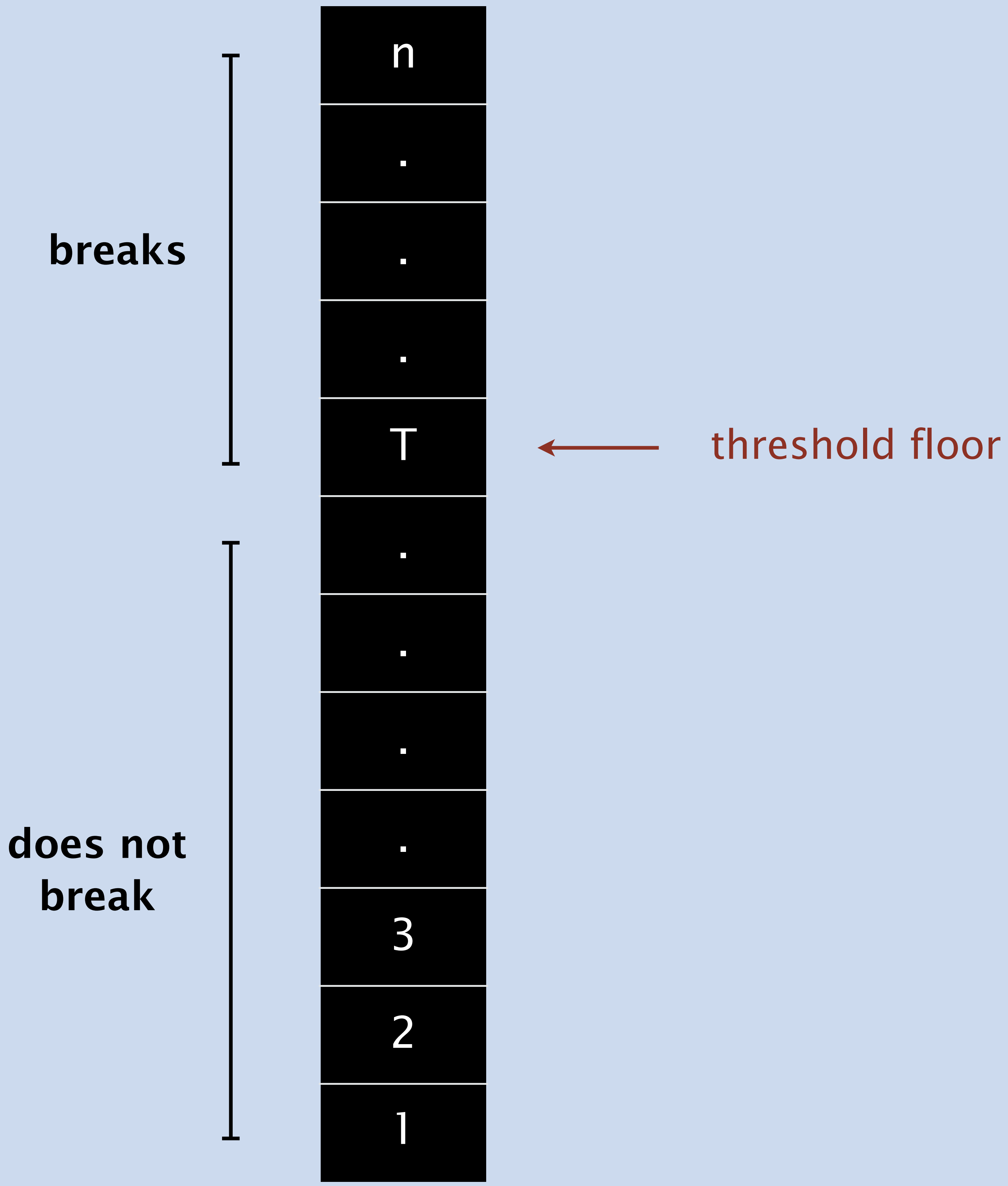
To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time t is determined by a set of quantities which we call state 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

EGG DROP (REVISITED)

Goal. Given m eggs and n floors, find threshold floor using the fewest drops.



EGG DROP (REVISITED)

Goal. Given m eggs and n floors, find threshold floor using the fewest drops.

Subproblems. $OPT(i, j)$ = fewest drops with i eggs and j contiguous floors to check.

Optimal value. $OPT(m, n)$.

best over all x

Multiway choice. To compute $OPT(i, j)$, drop next egg on some floor x between 1 and j.

- Breaks: use fewest drops given $i - 1$ eggs and $x - 1$ floors.
- Does not break: use fewest drops given i eggs and $j - x$ floors.

for a given floor x, take worst of two possibilities

optimal substructure

only number of contiguous floors to check matters (not that they are between 1 and j)

Dynamic programming recurrence.

$$OPT(i,j) = \begin{cases} j & \text{if } i = 1 \\ 0 & \text{if } j = 0 \\ \min\limits_{1 \le x \le j} \left\{ 1 + \max\{ OPT(i-1, x-1), OPT(i, j-x) \} \right\} & \text{if } i > 1 \text{ and } j > 0 \end{cases}$$

Coin changing: bottom-up implementation

Bottom-up DP implementation.

```
// drops[i][j] = min number of drops with i eggs and j floors
int[][] drops = new int[eggs+1][floors+1];

// base cases
for (int j = 1; j <= floors; j++) drops[1][j] = j;
for (int i = 1; i <= eggs; i++)   drops[i][0] = 0;

// dynamic programming recurrence
for (int i = 2; i <= eggs; i++) {
   for (int j = 1; j  <= floors; j++) {
      drops[i][j] = Integer.MAX_VALUE;
      for (int x = 1; x <= j; x++) {
         int temp = 1 + Math.max(drops[i-1][x-1], drops[i][j-x]);
         drops[i][j] = Math.min(temp, drops[i][j]);
      }
   }
}
```

drop #	floor
1	14
2	27
3	39
4	50
5	60
6	69
7	77
8	84
9	90
10	95
11	99
12	100

drop first egg on these floors (until it breaks)

m = 2 eggs, n = 100 floors

(max number of drops = 14)

Algorithm design: quiz 4

What is running time of algorithm as a function of the number of eggs m and the number of floors n?

A. $\Theta(m + n)$

B. $\Theta(m\,n)$

C. $\Theta(m\,n^2)$

D. $\Theta(m^2\,n)$

https://algs4.cs.princeton.edu

Algorithm Design

- *analysis of algorithms*
- *greed*
- *network flow*
- *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.

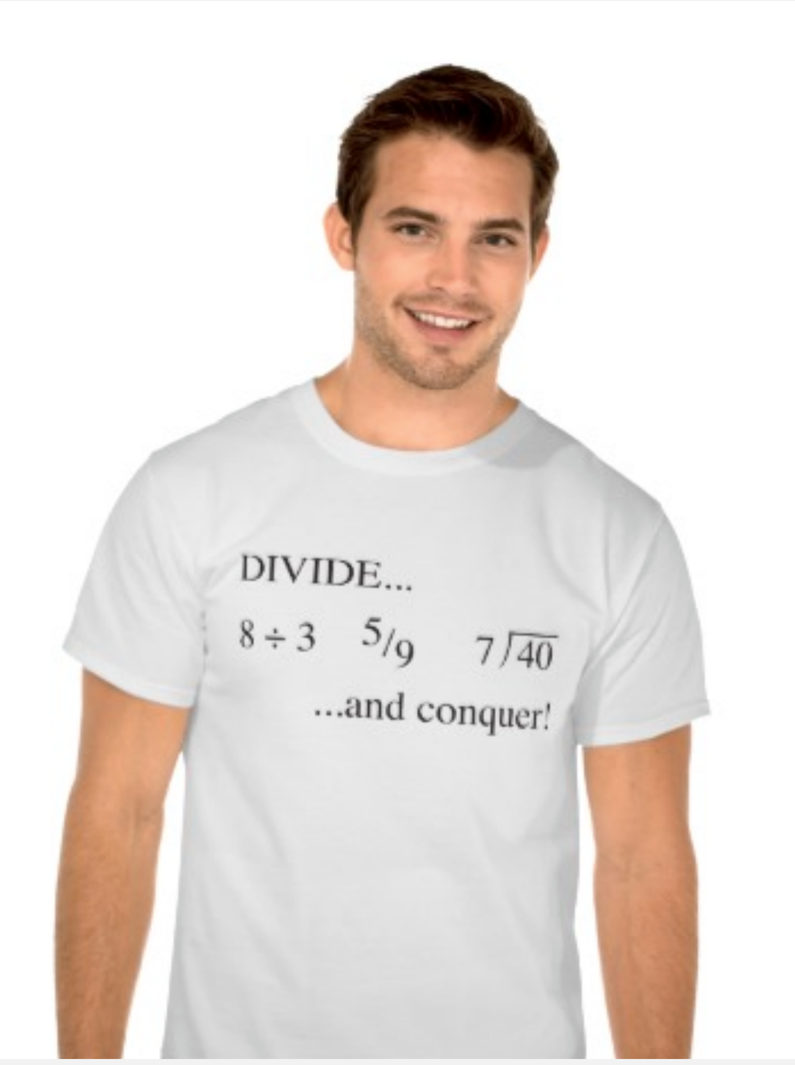
Familiar examples.

- 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*
- *network flow*
- *dynamic programming*
- *divide-and-conquer*
- ***randomization***

Randomized algorithms

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

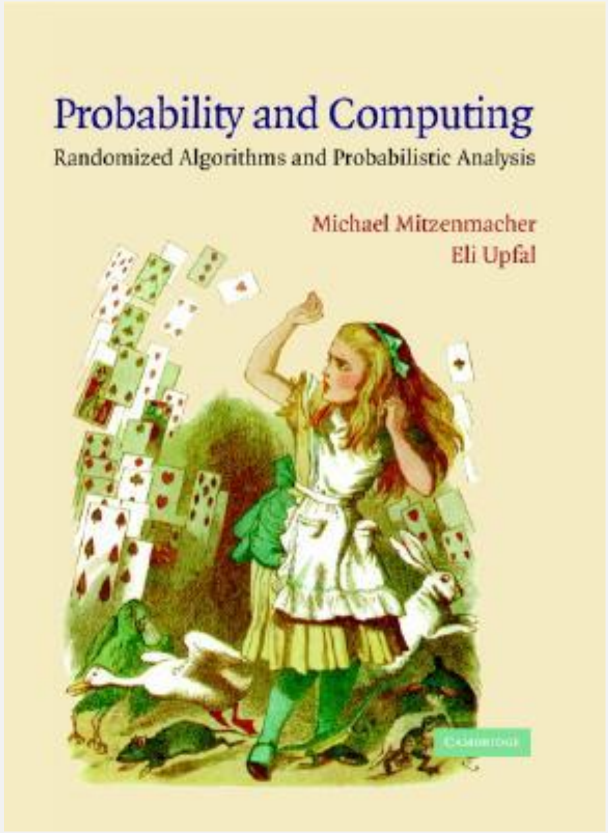
Familiar examples.

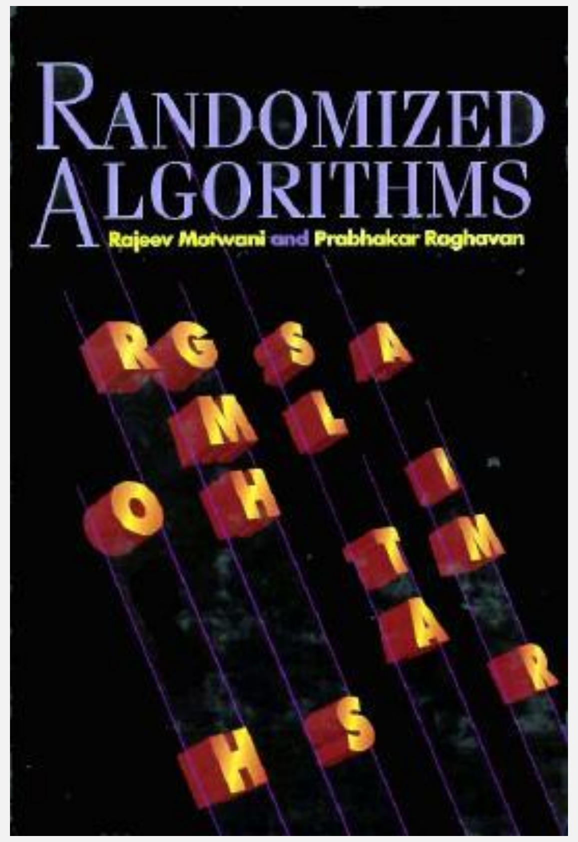
- Quicksort.
- Quickselect.

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 subproblems.

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

A. $\Theta(n)$

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

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

D. $\Theta(n^2)$

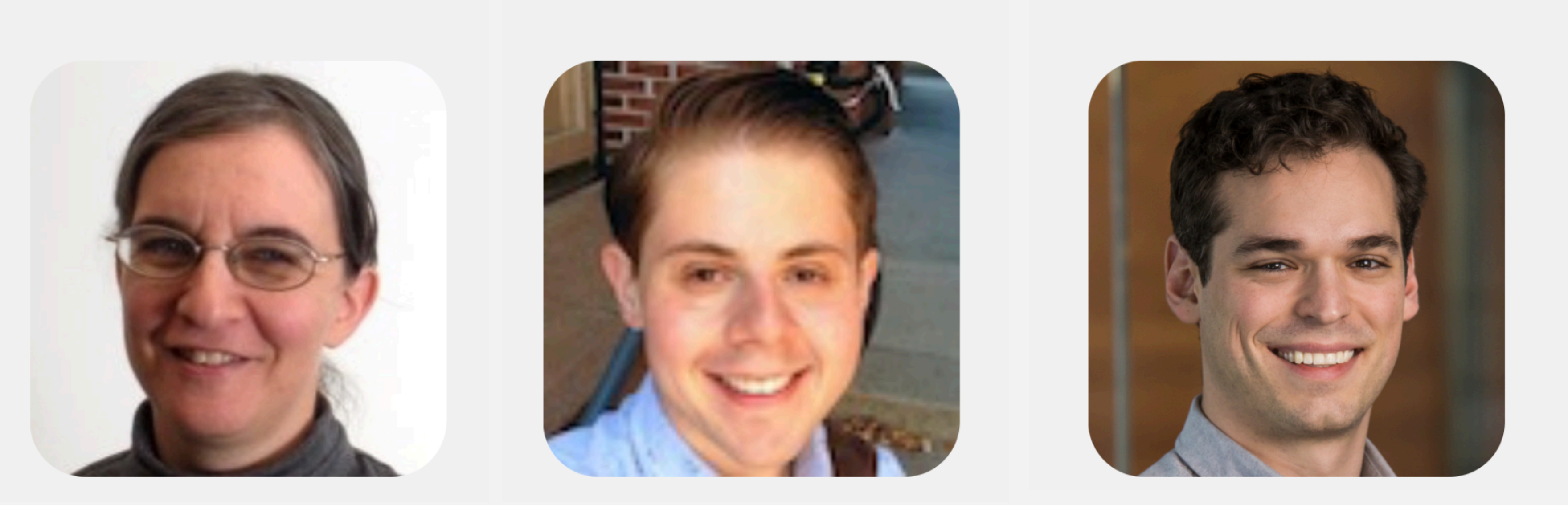
https://algs4.cs.princeton.edu

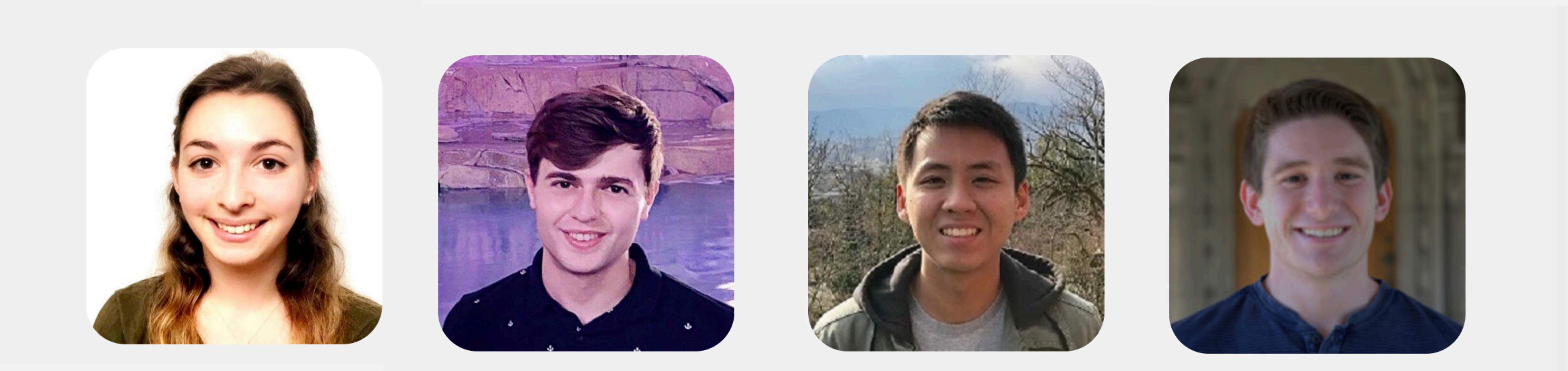
Algorithm Design

- analysis of algorithms
- greed
- network flow
- dynamic programming
- divide-and-conquer
- randomization
- credits

Credits

Faculty senior staff and graduate student AIs.





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

Ed tech. Several developed here at Princeton!

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