CSC 421: Algorithm Design & Analysis

Spring 2016

Analyzing problems

- revisiting Greedy Paths, Futoshiki
- interesting problem: residence matching
- lower bounds on problems
  - decision trees, adversary arguments, problem reduction

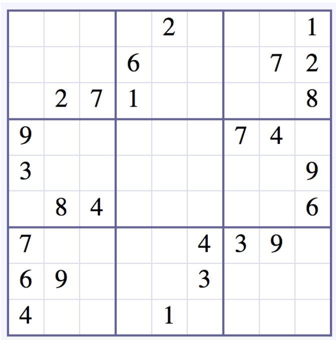
Debrief of Greedy Paths

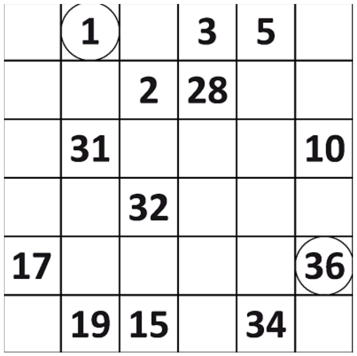
- finding min & max elevations
- adjusting the grayscale
- displaying the greedy paths
  - displaying a single greedy path
- finding & displaying the optimal greedy path

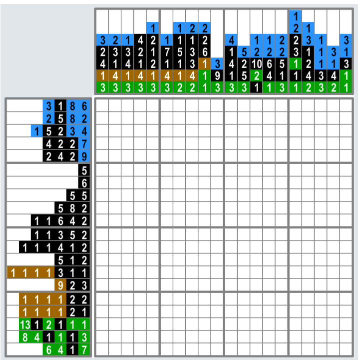
revisiting Futoshiki

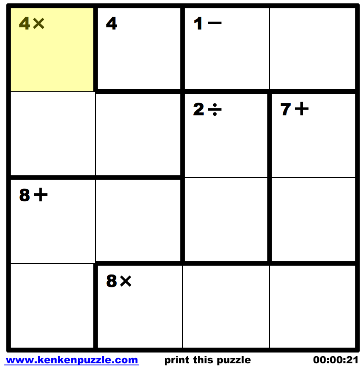
Futoshiki is similar to many grid-based puzzles

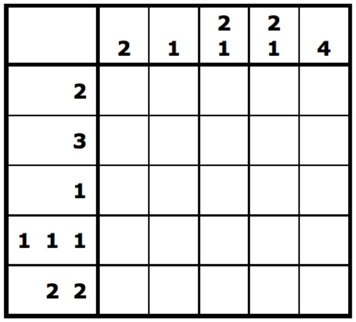
- Sudoku, KenKen, Nonogrids, Hidoku, Pic-a-Pix, Flow Free



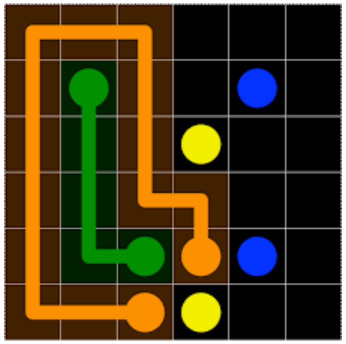








April 10 - 5 x 5
© Kevin Stone



all are based on filling a grid with values (numbers, colors, …) that meet some constraints

Generic backtracking approach

```
while grid is not filled
   find an open cell in the grid
   for each possible value (number, color, …)
      try placing that value in the cell
      if it meets the constraints,
         then try to fill the rest of the grid
      if can't fill the rest, then backtrack
         (i.e., erase the value and continue looping
          through the remaining values)
```

Interesting problem: residence matching

each year, the National Residence Matching Program matches 40,000+ med school graduates with residency programs

- each graduate ranks programs by order of preference
- each program ranks students by order of preference

pairing graduates & programs in a way that makes everyone (reasonably) happy is an extremely complex task

- want to ensure that the pairings are *stable*, i.e., no grad and program would prefer each other over their assigned matches

  e.g., suppose G_1 listed $P_1 > P_2$; and P_1 listed $G_1 > G_2$

  the match $\{G_1 \rightarrow P_2, G_2 \rightarrow P_1\}$ is unstable, since both G_1 and P_1 would prefer $G_1 \rightarrow P_1$

since 1952, the NRMP has utilized an algorithm for processing all residency requests and assigning stable matches to graduates

(this general problem is known as the *stable matching* or *stable marriage problem*)

Stable matching example

can specify preferences either by two tables of rankings

<u>grad's preferences</u>

	1st	2nd	3rd
G_1:	P_2	P_1	P_3
G_2:	P_2	P_3	P_1
G_3:	P_3	P_2	P_1

<u>program's preferences</u>

	1st	2nd	3rd
P_1:	G_2	G_3	G_1
P_2:	G_3	G_1	G_2
P_3:	G_2	G_3	G_1

or via a combined rankings matrix

<u>ranking matrix</u>

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

$G_1 \rightarrow P_1$, $G_2 \rightarrow P_2$, $G_3 \rightarrow P_3$ is unstable

- G_1 would prefer P_2 over P_1
- P_2 would prefer G_1 over G_2

$G_1 \rightarrow P_1$, $G_2 \rightarrow P_3$, $G_3 \rightarrow P_2$ is stable

Stable match algorithm (Gale-Shapley)

1. start with all the grads and programs being unassigned
2. while there are unassigned grads, select an unassigned grad (S_u):
    a. have S_u chooses the next program on S_u's preference list (P_n)
    b. if P_n is unassigned, it (tentatively) accepts S_u
    c. otherwise, it compares S_u with its current match (S_m)
        i. if P_n prefers S_u to S_m, it switches its assignment to S_u (releasing S_m)

ranking matrix

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

initially, $\{G_1, G_2, G_3\}$ unassigned

suppose we select G_1
G_1 chooses P_2
P_2 is unassigned, so it accepts G_1

now, $\{G_1 \rightarrow P_2\}$ & $\{G_2, G_3\}$ unassigned

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

suppose we select G_2
G_2 chooses P_2
P_2 is assigned G_1 and prefers G_1, so no change

Stable match algorithm (Gale-Shapley)

ranking matrix

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

still, $\{G_1 \rightarrow P_2\}$ & $\{G_2, G_3\}$ unassigned

suppose we select G_2 again
G_2 now chooses P_3
P_3 is unassigned, so it accepts G_2

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

now, $\{G_1 \rightarrow P_2, G_2 \rightarrow P_3\}$ & $\{G_3\}$ unassigned

we select G_3
G_3 chooses P_3
P_3 is assigned G_2 and prefers G_2, so no change

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

still, $\{G_1 \rightarrow P_2, G_2 \rightarrow P_3\}$ & $\{G_3\}$ unassigned

we select G_3
G_3 now chooses P_2
P_2 is assigned G_1 but prefers G_3, so switches

Stable match algorithm (Gale-Shapley)

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

now, $\{G_2 \rightarrow P_3, G_3 \rightarrow P_2\}$ & $\{G_1\}$ unassigned

we select G_1
G_1 chooses P_2
P_2 is assigned G_3 and prefers G_3, so no change

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

still, $\{G_2 \rightarrow P_3, G_3 \rightarrow P_2\}$ & $\{G_1\}$ unassigned

we select G_1
G_1 now chooses P_1
P_1 is unassigned, so it accepts G_1

	P_1	P_2	P_3
G_1	2\3	1\2	3\3
G_2	3\1	1\3	2\1
G_3	3\2	2\1	1\2

now, $\{G_1 \rightarrow P_1, G_2 \rightarrow P_3, G_3 \rightarrow P_2\}$

this is a stable match

Analysis of the Gale-Shapley Algorithm

the algorithm produces a stable matching in no more than N^2 iterations

the stable matching produced is always *graduate-optimal*, meaning each grad gets the highest rank program on his/her list under any stable marriage

- the graduate-optimal matching is unique for a given set of grad/program preferences
- originally, the NRMP used a variant of this algorithm with the roles reversed, producing a *program-optimal* matching

the NRMP algorithm now allows for couples to apply together

- this more complex problem turns out to be nP-complete (LATER)
- as a result, the algorithm may produce a partial matching, with unassigned grads going into a secondary Scramble pool

Lloyd Shapley was awarded the 2012 Nobel Prize in Economics for his work and analysis of matching algorithms

Analyzing problems

for most of this class, we have focused on devising algorithms for a given problem, then analyzing those algorithms

selection sort a list of numbers → $O(N^2)$

find shortest path between v_1 & v_2 in a graph (Dijkstra's) → $O(V^2)$

does that mean sorting & path finding are equally hard problems?

we know of a more efficient algorithm for sorting

merge sort → $O(N \log N)$

does that mean it is an easier problem?

Proving lower bounds

to characterize the difficulty of a problem (not a specific algorithm), must be able to show a lower bound on possible algorithms

- can be shown that comparison-based sorting requires $\Omega(N \log N)$ steps
- similarly, shortest path for an undirected graph requires $\Omega(E + V \log V)$ steps

establishing a lower bound for a problem can tell us

- when a particular algorithm is as good as possible
- when the problem is intractable (by showing that best possible algorithm is BAD)

methods for establishing lower bounds:

- brute force
- information-theoretic arguments (decision trees)
- adversary arguments
- problem reduction

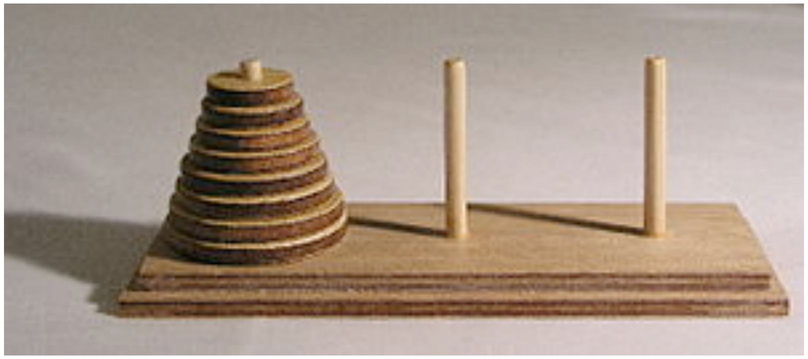
Brute force arguments

sometimes, a problem-specific approach works

example: polynomial evaluation $p(N) = a_N x^N + a_{N-1} x^{N-1} + \ldots + a_0$

- evaluating this polynomial requires $\Omega(N)$ steps, since each coefficient must be processed

example: Towers of Hanoi puzzle



- can prove, by induction, that moving a tower of size N requires $\Omega(2^N)$ steps

Information-theoretic arguments

can sometimes establish a lower bound based on the amount of information the solution must produce

example: guess a randomly selected number between 1 and N

- with possible responses of "correct", "too low", or "too high"

- the amount of uncertainty is $\lceil \log_2 N \rceil$, the number of bits needed to specify the selected largest number
  e.g., N = 127 → 7 bits

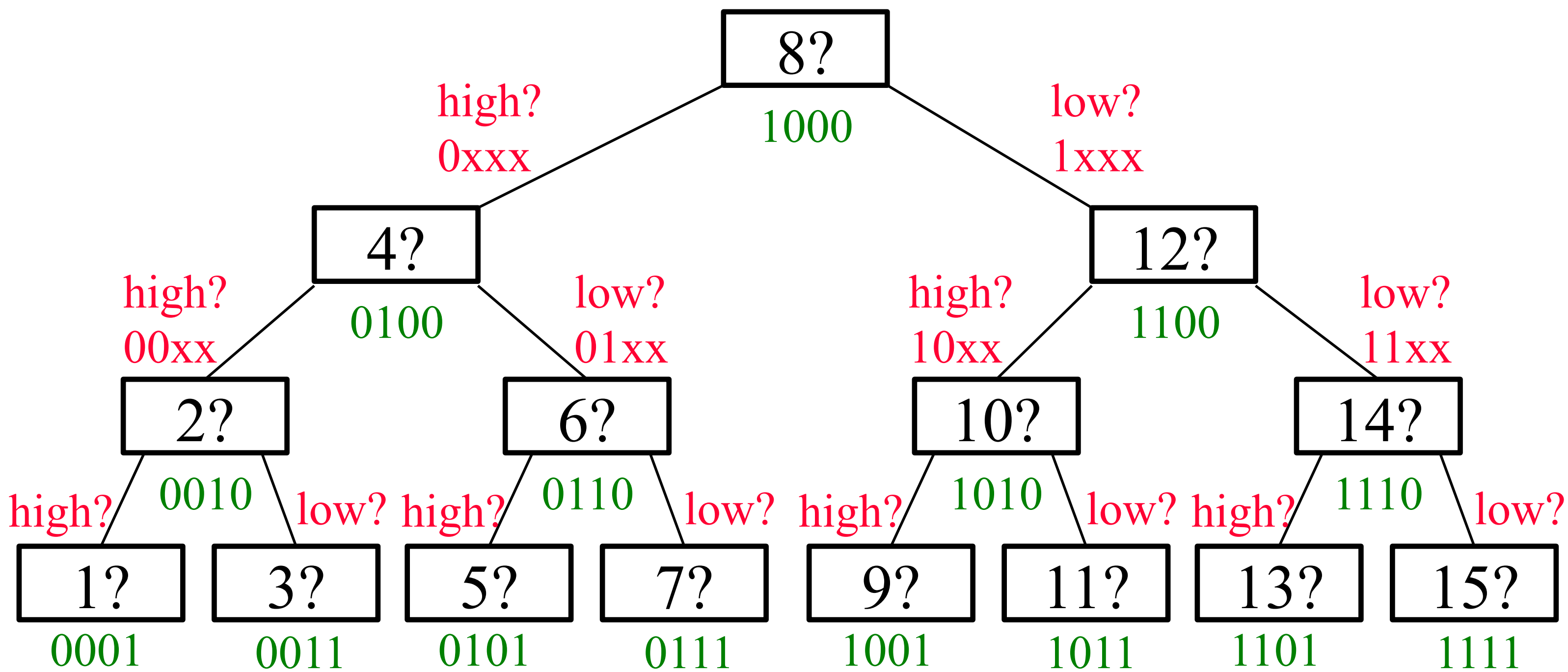
- each answer to a question yields at most 1 bit of information
  if guess of 64 yields “too high,” then 1st bit must be a 0 → 0xxxxxx
  if next guess of 32 yields “too low,”, then 2nd bit must be 1 → 01xxxxx
  if next guess of 48 yields “too low,” then 3rd bit must be 1 → 011xxxx
  . . .

- thus, $\lceil \log_2 N \rceil$ is a lower bound on the number of questions

Decision trees

a useful structure for information-theoretic arguments is a *decision tree*

example: guessing a number between 1 and 15



- min # of nodes in the decision tree?
- min height of binary tree with that many nodes?

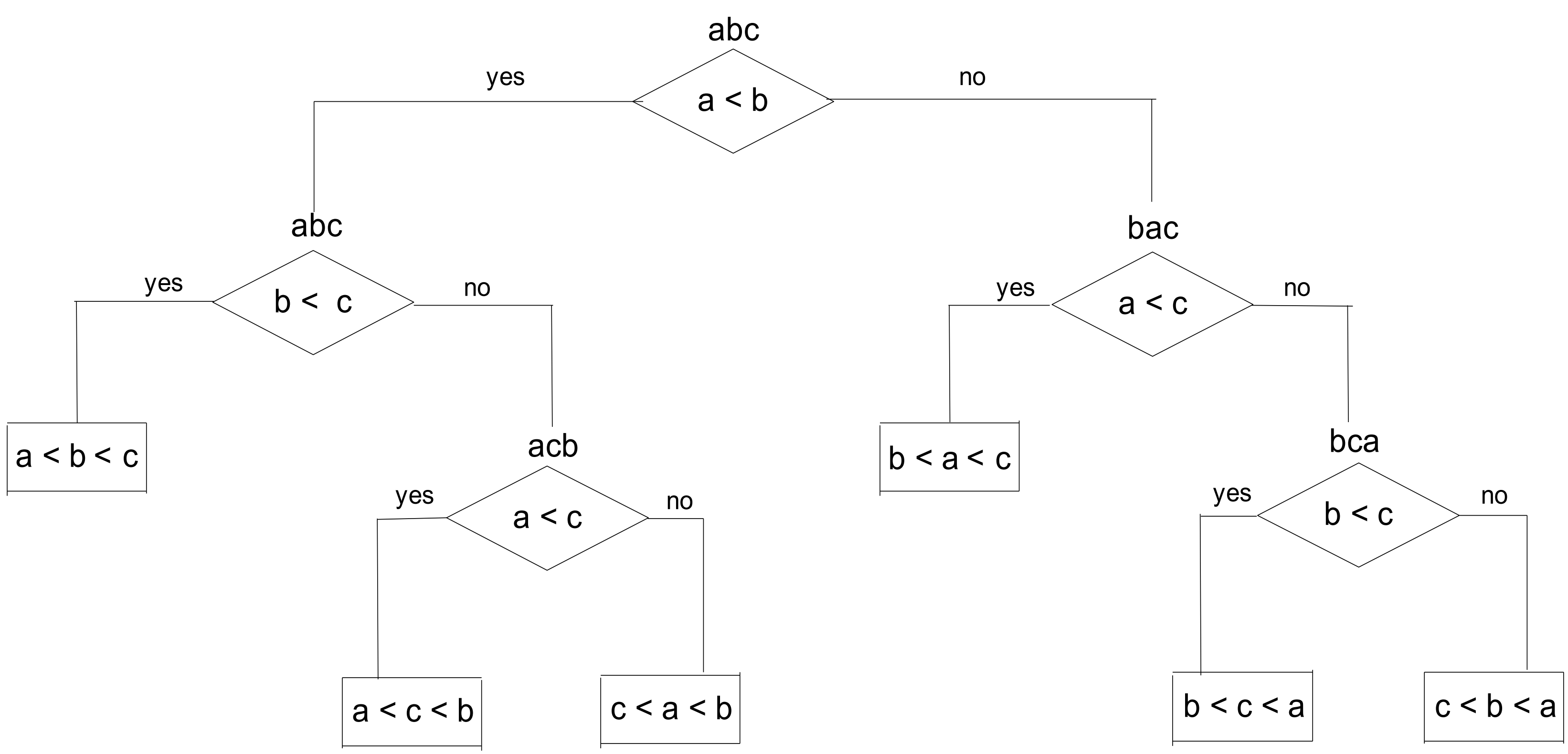
note that this problem is Ω(minimal decision tree height)

Decision trees

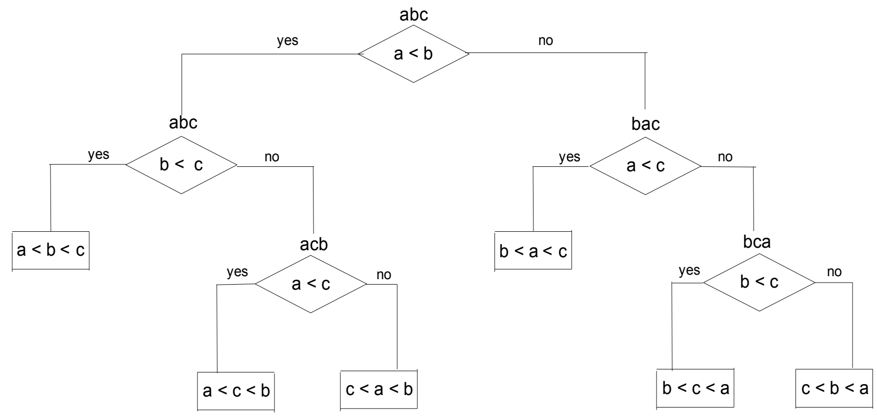
in general, a *decision tree* is a model of algorithms involving comparisons

- internal nodes represent comparisons
- leaves represent outcomes

e.g., decision tree for 3-element (comparison-based) sort:



Decision trees & sorting



note that any comparison-based sorting algorithm can be represented by a decision tree

- number of leaves (outcomes) $\geq$ N!
- height of binary tree with N! leaves $\geq \lceil \log_2 N! \rceil$
- therefore, the minimum number of comparisons required by any comparison-based sorting algorithm $\geq \lceil \log_2 N! \rceil$
- since $\lceil \log_2 N! \rceil \approx N \log_2 N$ (*proof not shown*), $\Omega(N \log N)$ steps are required

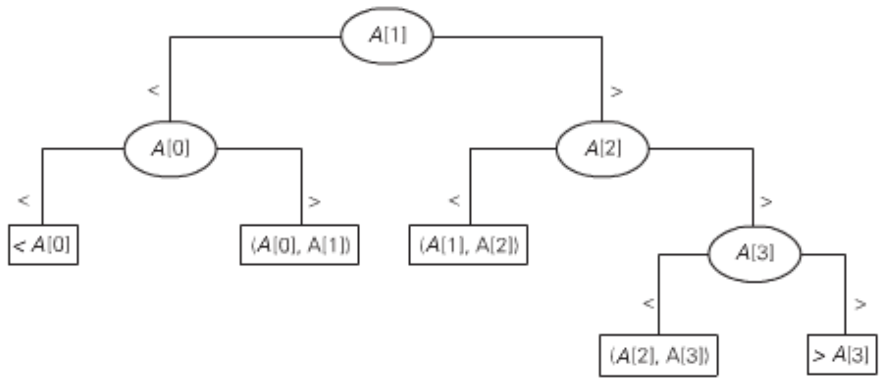
thus, merge/quick/heap sorts are as good as it gets

Decision trees & searching

similarly, we can use a decision tree to show that binary search is as good as it gets (assuming the list is sorted)

decision tree for binary search of 4-element list:

- internal nodes are found elements
- leaves are ranges if not found



- number of leaves (ranges where not found) = N + 1
- height of binary tree with N+1 leaves ≥ $\lceil \log_2 (N+1) \rceil$
- therefore, the minimum number of comparisons required by any comparison-based searching algorithm ≥ $\lceil \log_2 (N+1) \rceil$
- $\Omega(\log N)$ steps are required

Adversary arguments

using an *adversary argument*, you repeatedly adjust the input to make an algorithm work the hardest

example: dishonest hangman

- adversary always puts the word in a larger of the subset generated by last guess
- for a given dictionary, can determine a lower bound on guesses

example: merging two sorted lists of size N (as in merge sort)

- adversary makes it so that no list "runs out" of values (e.g., $a_i < b_j$ iff $i < j$)
- forces 2N-1 comparisons to produce $b_1 < a_1 < b_2 < a_2 < \ldots < b_N < a_N$

Problem reduction

problem reduction uses a transform & conquer approach

- if we can show that problem *P* is at least as hard as problem *Q*, then a lower bound for *Q* is also a lower bound for *P.*

  i.e., hard(P) ≥ hard(Q) ➔ if Q is Ω(X), so is P

in general, to prove lower bound for P:

1. find problem Q with a known lower bound
2. reduce that problem to problem P

   i.e., show that can solve Q by solving an instance of P
3. then P is at least as hard as Q, so same lower bound applies

example: prove that multiplication (of N-bit numbers) is Ω(N)

1. squaring an N-bit number is known to be Ω(N)
2. can reduce squaring to multiplication: $x^2 = x * x$
3. then multiplication is at least as hard as squaring, so also Ω(N)

REASONING: if multiplication could be solved in O(X) where X < N,

*then could do x^2 by doing x*x → O(X) < O(N)* *CONTRADICTION OF SQUARE'S Ω(N)*

Problem reduction example

CLOSEST NUMBERS (CN) PROBLEM: given N numbers, find the two closest numbers

1. consider the ELEMENT UNIQUENESS (EU) problem
   - given a list of N numbers, determine if all are unique (no dupes)
   - this problem has been shown to have a lower bound of $\Omega(N \log N)$

2. can reduce EU to CN

   consider an instance of EU: given numbers $e_1, \ldots, e_N$, determine if all are unique
   - find the two closest numbers (this is an instance of CN)
   - if the distance between them is > 0, then $e_1, \ldots, e_N$ are unique

3. this shows that CN is at least as hard as EU
   - can solve an instance of EU by performing a transformation & solving CN
   - since transformation is O(N), CN must also have a lower-bound of $\Omega(N \log N)$

   REASONING: if CN could be solved in O(X) where X < N log N,

   then could solve EU by transforming & solving CN → O(N) +O(X) < O(N log N)

   CONTRADICTION OF EU's $\Omega(N \log N)$

Another example

CLOSEST POINTS (CP) PROBLEM: given N points in the plane, find the two closest points

1. consider the CLOSEST NUMBER (CN) problem
   - we just showed that CN has a lower bound of $\Omega(N \log N)$

2. can reduce CN to CP

   consider an instance of CN: given numbers $e_1, \ldots, e_N$, determine closest numbers
   - from these N numbers, construct N points: $(e_1, 0), \ldots, (e_N, 0)$
   - find the two closest points (this is an instance of CP)
   - if $(e_i, 0)$ and $(e_j, 0)$ are closest points, then e_i and e_j are closest numbers

3. this shows that CP is at least as hard as CN
   - can solve an instance of CN by performing a transformation & solving CP
   - since transformation is O(N), CP must also have a lower-bound of $\Omega(N \log N)$

   REASONING: if CP could be solved in O(X) where X < N log N,

   then could solve CN by transforming & solving CP → O(N) +O(X) < O(N log N)

   CONTRADICTION OF CN's $\Omega(N \log N)$

Tightness

note: if an algorithm is $\Omega(N \log N)$, then it is also $\Omega(N)$

are the $\Omega(N \log N)$ lower bounds tight for CLOSEST NUMBERS and CLOSEST POINTS problems?

- can you devise $O(N \log N)$ algorithm for CLOSEST NUMBERS?
- can you devise $O(N \log N)$ algorithm for CLOSEST POINTS?