23 Reductions (April 22)

23.1 Introduction

A common technique for deriving algorithms is *reduction* — Instead of solving a problem directly, we use an algorithm for some other related problem as a subroutine or black box.

For example, when we talked about nuts and bolts, I argued that once the nuts and bolts are sorted, we can match each nut to its bolt in linear time. Thus, since we can sort nuts and bolts in $O(n \log n)$ expected time, then we can also match them in $O(n \log n)$ expected time:

$$T_{\text{match}}(n) \leq T_{\text{sort}}(n) + O(n) = O(n \log n) + O(n) = O(n \log n).$$

Let's consider (as we did in lecture 3) a decision tree model of computation, where every query is a comparison between a nut and a bolt — too big, too small, or just right? The output to the matching problem is a permutation π, where for all i, the ith nut matches the $\pi(i)$th bolt. Since there are $n!$ permutations of n items, any nut/bolt comparison tree that matches n nuts and bolts has at least $n!$ leaves, and thus has depth at least $\lceil \log_3(n!) \rceil = \Omega(n \log n)$.

Now the same reduction from matching to sorting can be used to prove a lower bound for *sorting* nuts and bolts, just by reversing the inequality:

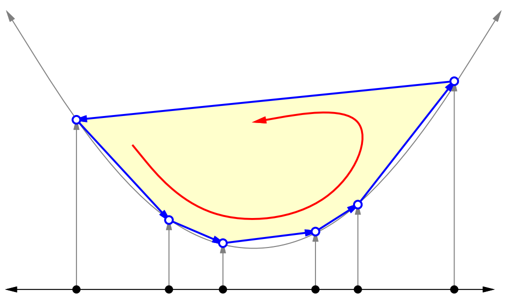
$$T_{\text{sort}}(n) \geq T_{\text{match}}(n) - O(n) = \Omega(n \log n) - O(n) = \Omega(n \log n).$$

Thus, any nut-bolt comparison tree that sorts n nuts and bolts has depth $\Omega(n \log n)$, and our randomized quicksort algorithm is optimal.[1]

23.2 Sorting to Convex Hulls

Here's a slightly less trivial example. Suppose we want to prove a lower bound for the problem of computing the convex hull of a set of n points in the plane. To do this, we demonstrate a reduction from sorting to convex hulls.

To sort a list of n numbers $\{a, b, c, \ldots\}$, we first transform it into a set of n points $\{(a, a^2), (b, b^2), (c, c^2), \ldots\}$. You can think of the original numbers as a set of points on a horizontal real number line, and the transformation as lifting those point up to the parabola $y = x^2$. Then we compute the convex hull of the parabola points. Finally, to get the final sorted list of numbers, we output the first coordinate of every convex vertex, starting from the leftmost vertex and going in counterclockwise order.



Reducing sorting to computing a convex hull.

[1]We could have proved this lower bound directly. The output to the sorting problem is *two* permutations, so there are $n!^2$ possible outputs, and we get a lower bound of $\lceil \log_3(n!^2) \rceil = \Omega(n \log n)$.

Transforming the numbers into points takes $O(n)$ time. Since the convex hull is output as a circular doubly-linked list of vertices, reading off the final sorted list of numbers also takes $O(n)$ time. Thus, given a black-box convex hull algorithm, we can sort in linear extra time. In this case, we say that *there is a linear time reduction from sorting to convex hulls.* We can visualize the reduction as follows:

$$\boxed{\begin{array}{rcl} \text{set of } n \text{ numbers} & \xrightarrow{O(n)} & \text{set of } n \text{ points} \\ & & \Downarrow \\ \text{sorted list of numbers} & \xleftarrow{O(n)} & \text{convex polygon} \end{array}}$$

The reduction gives us the following inequality.

$$T_{\text{sort}}(n) \leq T_{\text{convex hull}}(n) + O(n)$$

Since we can compute convex hulls in $O(n \log n)$ time, our reduction implies that we can also sort in $O(n \log n)$ time. More importantly, by reversing the inequality, we get a lower bound on the complexity of computing convex hulls.

$$T_{\text{convex hull}}(n) \geq T_{\text{sort}}(n) - O(n)$$

Since any binary decision tree requires $\Omega(n \log n)$ time to sort n numbers, it follows that any binary decision tree requires $\Omega(n \log n)$ time to compute the convex hull of n points.

23.3 Watch the Model!

This result about the complexity of computing convex hulls is often misquoted as follows:

> *Since we need $\Omega(n \log n)$ comparisons to sort, we also need $\Omega(n \log n)$ comparisons (between x-coordinates) to compute convex hulls.*

Although this statement is true, **it's completely trivial**, since it's impossible to compute convex hulls using *any* number of comparisons! In order to compute hulls, we *must* perform counterclockwise tests on triples of points.

The convex hull algorithms we've seen — Graham's scan, Jarvis's march, divide-and-conquer, Chan's shatter — can all be modeled as binary[2] decision trees, where every query is a counterclockwise test on three points. So our binary decision tree lower bound is meaningful, and several of those algorithms are optimal.

This is a subtle but important point about deriving lower bounds using reduction arguments. In order for any lower bound to be meaningful, it must hold in a model in which the problem can be solved! Often the problem we are reducing *from* is much simpler than the problem we are reducing *to*, and thus can be solved in a more restrictive model of computation.

23.4 Element Uniqueness (A Bad Example)

The *element uniqueness* problem asks, given a list of n numbers $x_1, x_2, \ldots, x_n$, whether any two of them are equal. There is an obvious and simple algorithm to solve this problem: sort the numbers,

[2] or ternary, if we allow colinear triples of points

and then scan for adjacent duplicates. Since we can sort in $O(n \log n)$ time, we can solve the element uniqueness problem in $O(n \log n)$ time.

We also have an $\Omega(n \log n)$ lower bound for sorting, but our reduction does *not* give us a lower bound for element uniqueness. The reduction goes the wrong way! Inscribe the following on the back of your hand[3]:

To prove that problem A is harder than problem B, reduce B to A.

There isn't (as far as I know) a reduction from sorting to the element uniqueness problem. However, using other techniques (which I won't talk about), it is possible to prove an $\Omega(n \log n)$ lower bound for the element uniqueness problem. The lower bound applies to so-called *algebraic decision trees*. An algebraic decision tree is a ternary decision tree, where each query asks for the sign of a constant-degree polynomial in the variables $x_1, x_2, \ldots, x_n$. A comparison tree is an example of an algebraic decision tree, using polynomials of the form $x_i - x_j$.

The reduction from sorting to element uniqueness implies that any algebraic decision tree requires $\Omega(n \log n)$ time to sort n numbers. But since algebraic decision trees are ternary decision trees, we already knew that.

23.5 Closest Pair

The simplest version of the *closest pair* problem asks, given a list of n numbers $x_1, x_2, \ldots, x_n$, to find the closest pair of elements, that is, the elements x_i and x_j that minimize $|x_i - x_j|$.

There is an obvious reduction from element uniqueness to closest pair, based on the observation that the elements of the input list are distinct if and only if the distance between the closest pair is bigger than zero. This reduction implies that the closest pair problem requires $\Omega(n \log n)$ time in the algebraic decision tree model.

There are also higher-dimensional closest pair problems; for example, given a set of n points in the plane, find the two points that closest together. Since the one-dimensional problem is a special case of the 2d problem — just put all n point son the x-axis — the $\Omega(n \log n)$ lower bound applies to the higher-dimensional problems as well.

23.6 3SUM to Colinearity ...

Unfortunately, lower bounds are relatively few and far between. There are thousands of computational problems for which we cannot prove any good lower bounds. We can still learn something useful about the complexity of such a problem by from reductions, namely, that it is harder than some other problem.

Here's an example. The problem 3SUM asks, given a sequence of n numbers $x_1, \ldots, x_n$, whether any three of them sum to zero. There is a fairly simple algorithm to solve this problem in $O(n^2)$ time **(hint, hint)**. This is widely believed to be the fastest algorithm possible. There is an $\Omega(n^2)$ lower bound for 3SUM, but only in a fairly weak model of computation.[4]

Now consider a second problem: given a set of n points in the plane, do any three of them lie on a common non-horizontal line? Again, there is an $O(n^2)$-time algorithm, and again, this is

[3]right under all those rules about logarithms, geometric series, and recurrences

[4]The $\Omega(n^2)$ lower bound holds in a decision tree model where every query asks for the sign of a linear combination of three of the input numbers. For example, "Is $5x_1 + x_{42} - 17x_5$ positive, negative, or zero?" See my paper "Lower bounds for linear satisfiability problems" (http://www.uiuc.edu/~jeffe/pubs/linsat.html) for the gory(!) details.

believed to be the best possible. The following reduction from 3SUM offers some support for this belief. Suppose we are given an array A of n numbers as input to 3SUM. Replace each element $a \in A$ with three points $(a, 0)$, $(-a/2, 1)$, and $(a, 2)$. Thus, we replace the n numbers with $3n$ points on three horizontal lines $y = 0$, $y = 1$, and $y = 2$.

If any three points in this set lie on a common non-horizontal line, they consist of one point on each of those three lines, say $(a, 0)$, $(-b/2, 1)$, and $(c, 2)$. The slope of the common line is equal to both $-b/2 - a$ and $c + b/2$; since these two expressions are equal, we must have $a + b + c = 0$. Similarly, is any three elements $a, b, c \in A$ sum to zero, then the resulting points $(a, 0)$, $(-b/2, 1)$, and $(c, 2)$ are colinear.

So we have a valid reduction from 3SUM to the colinear-points problem:

$$T_{\text{3SUM}}(n) \le T_{\text{colinear}}(3n) + O(n) \qquad \Longrightarrow \qquad T_{\text{colinear}}(n) \ge T_{\text{3SUM}}(n/3) - O(n).$$

Thus, if we could detect colinear points in $o(n^2)$ time, we could also solve 3SUM in $o(n^2)$ time, which seems unlikely. Conversely, if we could prove an $\Omega(n^2)$ lower bound for 3SUM in a sufficiently powerful model of computation, it would imply an $\Omega(n^2)$ lower bound for the colinear points problem as well.

The existing $\Omega(n^2)$ lower bound for 3SUM does *not* imply a lower bound for finding colinear points, because the model of computation is too weak. It is possible to prove an $\Omega(n^2)$ lower bound directly using an adversary argument, but only in a fairly weak decision-tree model of computation.

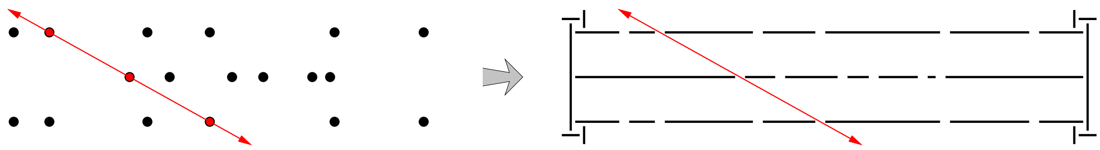
Note that in order to prove that the reduction is correct, we have to show that both yes answers and no answers are correct: the numbers sum to zero *if and only if* three points lie on a line. **Even though the reduction itself only goes one way, from the "easier" problem to the "harder" problem, the proof of correctness must go both ways.**

Anka Gajentaan and Mark Overmars[5] defined a whole class of computational geometry problems that are harder than 3SUM; they called these problems 3SUM-*hard*. A sub-quadratic algorithm for any 3SUM-hard problem would imply a subquadratic algorithm for 3SUM. I'll finish the lecture with two more examples of 3SUM-hard problems.

23.7 ... to Segment Splitting ...

Consider the following *segment splitting problem*: Given a collection of line segments in the plane, is there a line that does not hit any segment and splits the segments into two non-empty subsets?

To show that this problem is 3SUM-hard, we start with the collection of points produced by our last reduction. Replace each point by a "hole" between two horizontal line segments. To make sure that the only way to split the segments is by passing through three colinear holes, we build two "gadgets", each consisting of five segments, to cap off the left and right ends as shown in the figure below.



Top: $3n$ points, three on a non-horizontal line.
Bottom: $3n + 13$ segments separated by a line through three colinear holes.

[5]A. Gajentaan and M. Overmars, On a class of $O(n^2)$ problems in computational geometry, *Comput. Geom. Theory Appl.* 5:165–185, 1995. ftp://ftp.cs.ruu.nl/pub/RUU/CS/techreps/CS-1993/1993-15.ps.gz

This reduction could be performed in linear time if we could make the holes infinitely small, but computers can't really deal with infinitesimal numbers. On the other hand, if we make the holes too big, we might be able to thread a line through three holes that don't quite line up. I won't go into details, but it is possible to compute a working hole size in $O(n \log n)$ time by first computing the distance between the closest pair of points.

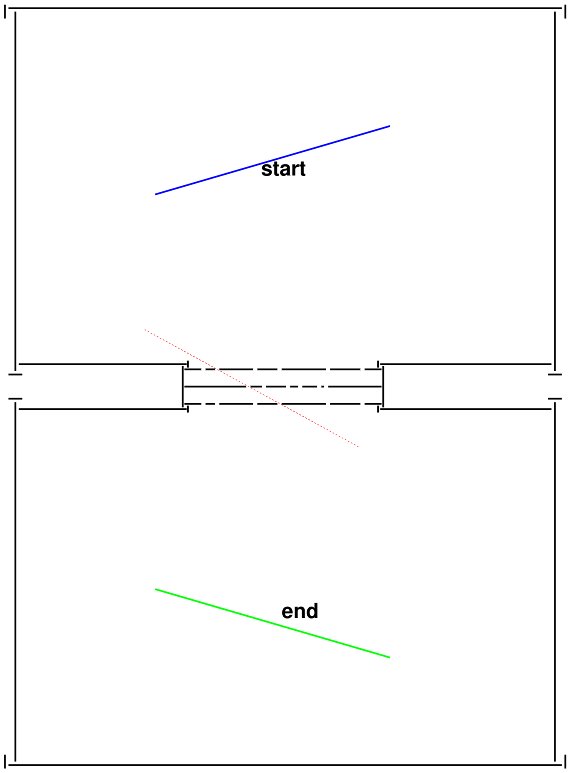
Thus, we have a valid reduction from 3SUM to segment splitting (by way of colinearity):

$$T_{3\text{SUM}}(n) \leq T_{\text{split}}(3n+13) + O(n \log n) \qquad \Longrightarrow \qquad T_{\text{split}}(n) \geq T_{3\text{SUM}}\left(\frac{n-13}{3}\right) - O(n \log n).$$

23.8 . . . to Motion Planning

Finally, suppose we want to know whether a robot can move from one position and location to another. To make things simple, we'll assume that the robot is just a line segment, and the environment in which the robot moves is also made up of non-intersecting line segments. Given an initial position and orientation and a final position and orientation, is there a sequence of translations and rotations that moves the robot from start to finish?

To show that this *motion planning* problem is 3SUM-hard, we do one more reduction, starting from the set of segments output by the previous reduction algorithm. Specifically, we use our earlier set of line segments as a "screen" between two large rooms. The rooms are constructed so that the robot can enter or leave each room only by passing through the screen. We make the robot long enough that the robot can pass from one room to the other if and only if it can pass through three colinear holes in the screen. (If the robot isn't long enough, it could get between the "layers" of the screen.) See the figure below:



The robot can move from one room to the other if and only if the screen between the rooms has three colinear holes.

Once we have the screen segments, we only need linear time to compute how big the rooms should be, and then $O(1)$ time to set up the 20 segments that make up the walls. So we have a fast reduction from 3SUM to motion planning (by way of colinearity and segment splitting):

$$T_{\text{3SUM}}(n) \le T_{\text{motion}}(3n+33) + O(n \log n) \qquad \Longrightarrow \qquad T_{\text{mtion}}(n) \ge T_{\text{3SUM}}\left(\frac{n-33}{3}\right) - O(n \log n).$$

Thus, a sufficiently powerful $\Omega(n^2)$ lower bound for 3SUM would imply an $\Omega(n^2)$ lower bound for motion planning as well. The existing $\Omega(n^2)$ lower bound for 3SUM does *not* imply a lower bound for this problem — the model of computation in which the lower bound holds is too weak to even solve the motion planning problem. In fact, the best lower bound anyone can prove for this motion planning problem is $\Omega(n \log n)$, using a (different) reduction from element uniqueness. But the reduction does give us *evidence* that motion planning "should" require quadratic time.