Algorithms

Outline

→ How to find the best algorithmic solutions to problems.

I. How to design algorithms

  • basic repertoire of algorithms
    – sorting (1st year), string algorithms (CS 240)
    – domain specific algorithms covered in other courses
      e.g., graph algorithms, linear programming (C&O); numerical algorithms (AM);
        algebraic algorithms in computer algebra (CS 487)
  • general paradigms: divide and conquer, greedy, dynamic programming, reductions

II. How to analyze algorithms → How good is this algorithm?

  • time, space, goodness of approximation
  • $O$-notation, worst/average case
  • models of computations

III. Lower bounds → Do we have the best algorithm?

  • basic lower bounds
  • NP-completeness and undecidability
Case Study: Convex Hull
Given \( n \) points in the plane, find their convex hull: the smallest convex set containing the points. (Like putting a rubber band around nails sticking out.) Why? Convex hull gives the “shape” of a set of points – better container than a minimal bounding box.

Equivalently (and more useful for thinking about algorithms) the convex hull is a polygon whose sides are formed by lines \( \ell \) that go through (at least) 2 points and have no points to one side of \( \ell \).
### A. Straightforward algorithm

For all pairs of points \( r, s \) \( \leftarrow O(n^2) \)

1. Find line through \( r, s \)
2. If all other points \( t \) lie on or to one side of \( \ell \) \( \otimes \)
   Then \( \ell \) forms part of convex hull

Time for \( n \) points: \( O(n^3) \)

Note: this is high-level pseudo-code. How do we test \( \otimes \)?

Following approach avoids division by 0, overflow: Only uses +, −, ×, <.

\[
\begin{align*}
\bullet \ r &= (x_1, y_1) \\
\bullet \ s &= (x_2, y_2) \\
\bullet \ t &= (x_3, y_3) \\
\bullet \ S &= (x_2-x_1)(y_3-y_1)-(y_2-y_1)(x_3-x_1)
\end{align*}
\]

Then \( S \)

\[
\begin{cases}
< 0 & \text{if path } (r, s, t) \text{ is moving clockwise} \\
= 0 & \text{if path } (r, s, t) \text{ forms a line} \\
> 0 & \text{if path } (r, s, t) \text{ is moving counterclockwise}
\end{cases}
\]

Can we do better? Yes — several possibilities.
B. Jarvis’ march

Observe that once we have found one line \( \ell \), there is a natural “next” line \( \ell' \). Rotate \( \ell \) through \( s \) until it hits the next point \( t \).

How can we find \( \ell' \)? Look at all lines through \( s \) and another point, and find the “extreme” one in the sense of minimizing angle \( \alpha \).

Finding extreme is like finding the minimum element of a set: \( O(n) \)

Whole algorithm is: \( O(n^2) \)

[This algorithm is good to use when the convex hull has few points. It actually takes time \( O(nh) \), where \( h \) is the number of convex hull points.]

Can we do better? Yes
C. Reduction
Repeatedly finding the minimum should remind you of sorting.
Sort points by $x$-coordinate.
Exercise: Find convex hull with $O(n)$ further work.
Hint: Find upper and lower convex hull separately.

A reduction uses an algorithm you know (sorting) to solve a new problem.
D. Use divide and conquer

Divide in half by vertical.
Recursively find convex hull on each side.
Combine by finding upper & lower bridge.

Details: initial $e = $ edge from max $x$ on left to min $x$ on right. “walk $e$ up” to get upper bridge, down to get lower bridge.

$O(n)$ to find median, upper and lower bridge

Get recurrence relation

$$T(n) = 2T(n/2) + O(n)$$

Like recurrence for merge sort, so $T(n) = O(n \log n)$. 
Can we do better?
In some sense, NO.
If we could find convex hull faster, we could sort faster.

This is not rigorous - what is the model of computation?

Challenge Look up Timothy Chan’s
“output sensitive convex hull algorithm” \(O(n \log h)\)

[Note: we saw \(O(n \log n)\) and \(O(nh)\). Which is better? Neither — hence Chan’s algorithm.]