Today’s Keywords

- Reductions
- NP Hard, NP Completeness
- k-Clique
- Convex Hull
- Graham Scan
- Jarvis’ March
- Chan’s Algorithm
• Chapter 34
• HW9 due Thursday at 11pm
  – Reductions, Graphs
  – Written (LaTeX)
• HW10C due Thursday at 11pm
  – Implement a problem from HW9
  – No late submissions
Final Exam

• Monday, December 9, 7pm in Maury 209 (our section)
  – Practice exam out! Solutions later this week
  – Review session this weekend (look for an email)
  – SDAC: please schedule for some time on Monday 12/9
Reductions

- Algorithm technique of supreme ultimate power
- Convert instance of problem A to an instance of Problem B
- Convert solution of problem B back to a solution of problem A
Possible uses

- **Use solver for B to solve A**

  \[ f(n) \text{-reduces to} \]

  - Don’t know how to solve
  - Algorithm for B
  - Can be used to make
  - With \( O(f(n)) \) overhead
  - Algorithm for A
  - Problem A
  - Problem B

- **Prove lower bound for B by showing it’s as hard as A**

  \[ f(n) \text{-reduces to} \]

  - Problem we know is slow to solve (proved)
  - Algorithm for B
  - Problem A
  - Problem B
  - With \( O(f(n)) \) overhead
  - Can be used to make
  - B is no faster than A
  - A is no harder than B
  - Algorithm for A
  - Problem B
  - Problem A

Don’t know how to solve

Do know how to solve
Reduction Proof Notation

Problem A \( f(n) \)-reduces to Problem B

Algorithm for B \( Y \)

Algorithm for A \( X \)

A is not a harder problem than B  \( A \leq B \)

If \( A \) requires time \( \Omega(f(n)) \) time then \( B \) also requires \( \Omega(f(n)) \) time  \( A \leq f(n) B \)

Or we could have solved \( A \) faster using B’s solver!
Proof of Lower Bound by Reduction

To Show: $Y$ is slow

1. We know $X$ is slow (by a proof) (e.g., $X =$ some way to open the door)

2. Assume $Y$ is quick [toward contradiction] ($Y =$ some way to light a fire)

3. Show how to use $Y$ to perform $X$ quickly

4. $X$ is slow, but $Y$ could be used to perform $X$ quickly
   conclusion: $Y$ must not actually be quick
P vs NP

• P
  – Deterministic Polynomial Time
  – Problems solvable in polynomial time
    • $O(n^p)$ for some number $p$

• NP
  – Non-Deterministic Polynomial Time
  – Problems verifiable in polynomial time
    • $O(n^p)$ for some number $p$

• Open Problem: Does P=NP?
  – Certainly $P \subseteq NP$
• How can we try to figure out if $P=NP$?
• Identify problems at least as “hard” as $NP$
  – If any of these “hard” problems can be solved in polynomial time, then all NP problems can be solved in polynomial time.
• Definition: NP-Hard:
  – $B$ is NP-Hard if $\forall A \in NP, A \leq_p B$
  – $A \leq_p B$ means $A$ reduces to $B$ in polynomial time
NP-Hardness Reduction

Any NP-Hard Problem

Then this could be done in polynomial time

Solution for \( A \)

If This could be done in Polynomial time

Reduction

Problem to show is NP-Hard

Solution for \( B \)

If This could be done in Polynomial time

Then this could be done in polynomial time

Solution for \( A \)

Reduction

Problem to show is NP-Hard

Solution for \( B \)
“Together they stand, together they fall”

Problems solvable in polynomial time iff ALL NP problems are

NP-Complete = NP \cap NP-Hard

**How to show a problem is NP-Complete?**

- Show it belongs to NP
  - Give a polynomial time verifier
- Show it is NP-Hard
  - Give a reduction from another NP-H problem

We now just need a FIRST NP-Hard problem
3-SAT

• Shown to be NP-Hard by Cook and Levin (independently)
• Given a 3-CNF formula (logical AND of clauses, each an OR of 3 variables), Is there an assignment of true/false to each variable to make the formula true?

\[(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land (z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})\]

Clause

\[x = true\]
\[y = false\]
\[z = false\]
\[u = true\]
Given a graph $G$ and a number $k$, is there a clique of size $k$?

- Clique: A complete subgraph
1. Show that it belongs to NP
   – Give a polynomial time verifier
2. Show it is NP-Hard
   – Give a reduction from a known NP-Hard problem
   – We will show $3SAT \leq_p kClique$
Given a Graph, $k$, and a potential solution
1. Check that the solution has $k$ nodes
2. Check that every pair of nodes share an edge
Instance of 3SAT to Instance of $k$-Clique

For each clause, produce a node for each of its three variables

Connect each node to all non-contradictory nodes in the other clauses (i.e., anything that’s not its negation)

Let $k =$ number of clauses

There is a $k$-Clique in this graph iff there is a satisfying assignment
There are $k$ triplets in the graph, and no two nodes in the same triplet are adjacent.

To have a $k$-Clique, must have one node from each triplet.

Cannot select a node for both a variable and its negation.

Therefore selection of nodes is a satisfying assignment.
Select one node for a true variable from each clause

There will be $k$ nodes selected
We can’t select both a node and its negation
All nodes will be non-contradictory, so they will be pairwise adjacent
3SAT \leq_p kClique

Solution for 3SAT

Solution for kClique

3SAT

\( A \)

\( X \)

\( 0(n^p) \)

Make a triplet per clause, connect non-contradictory nodes among clauses

Assign each variable selected to True

kClique

\( B \)

\( Y \)

Reduction
NP-Complete Problems

• We’ve now seen 4 NP-Complete problems
  – 3SAT
  – k-Independent Set
  – k-Vertex Cover
  – k-Clique

• You’ve seen 2 more in HW9
  – Backpacking
  – Subset Sum
One More Reduction
The Convex Hull Problem

**Problem:** find the smallest convex polygon that bounds a shape (or more generally, a collection of points)

**Example application:** collision detection in computer graphics, vision, robotics; also useful for solving other problems, especially in computational geometry (e.g., furthest pair of points)
The Convex Hull Problem

Convex polygon: all interior angles are less than 180°

Equivalently: line drawn through polygon will intersect exactly twice

Given a set of $n$ points, find the smallest convex polygon such that every point is either on the boundary or in the interior of the polygon.
The Convex Hull Problem

**Convex polygon:** all interior angles are less than 180°

Given a set of \( n \) points, find the smallest convex polygon such that every point is either on the boundary or in the interior of the polygon.
Rubber band analogy: imagine the points are nails sticking out of a board and wrapping a rubber band to encompass the nails; convex hull is resulting shape.
Observation: every point on the convex hull is one of the input points.

Otherwise, can add a “shortcut” and reduce the area.
A Brute Force Approach

If there are points on both sides of the line, then the pair cannot be an edge in the convex hull.
A Brute Force Approach
Brute force approach: for every pair of points, check if all other points are on the same side of the line $O(n^3)$
Observation: Extremal points must be part of the convex hull (e.g., bottom-most point, left-most point, etc.)
Consider the (polar) angle formed between base point $u$ and every other point.
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity
Idea: In order of angle, add points to the convex hull as long as it preserves convexity.
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity
Graham’s Algorithm

**Idea:** In order of angle, add points to the convex hull as long as it preserves convexity.
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity.
**Idea:** In order of angle, add points to the convex hull as long as it preserves convexity.
**Graham’s Algorithm**

**Idea:** In order of angle, add points to the convex hull as long as it preserves convexity.

Not convex anymore!

Scan the points in order of angle.
Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one.
Graham’s Algorithm

Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one.

Scan the points in order of angle.
Graham’s Algorithm

Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one

Observe: since points are sorted by angle, backtracking will never remove points from the convex hull
Idea: In order of angle, add points to the convex hull as long as it preserves convexity.
Idea: In order of angle, add points to the convex hull as long as it preserves convexity
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity.

Polar Angle

Scan the points in order of angle.
**Graham’s Algorithm**

**Idea:** In order of angle, add points to the convex hull as long as it preserves convexity.
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity.
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity.
Graham’s Algorithm

Idea: In order of angle, add points to the convex hull as long as it preserves convexity.
Idea: In order of angle, add points to the convex hull as long as it preserves convexity
1. Let $p_1$ be the point with the smallest $y$-coordinate (and smallest $x$-coordinate if multiple points have the same minimum-$y$ coordinate)
2. Add $p_1$ to the convex hull $C$ (represented as an ordered list)
3. Sort all of the points based on their angle relative to $p_1$
4. For each of the points $p_i$ in sorted order:
   • Try adding $p_i$ to the convex hull $C$
   • If adding $p_i$ makes $C$ non-convex, then remove the last component of $C$ and repeat this check

How to implement this?

Imagine driving from $A \rightarrow B$
- $B \rightarrow C$ is convex if need to take a “left turn” to reach $C$
- $B \rightarrow C$ is non-convex if need to take a “non-left turn”

Decide “left turn” vs. “right turn” by computing the sign of the (vector) cross product between $\vec{v}_{AB}$ and $\vec{v}_{BC}$
Graham’s Algorithm

1. Let $p_1$ be the point with the smallest $y$-coordinate (and smallest $x$-coordinate if multiple points have the same minimum $y$ coordinate)

2. Add $p_1$ to the convex hull $C$ (represented as an ordered list)

3. Sort all of the points based on their angle relative to $p_1$

4. For each of the points $p_i$ in sorted order:
   • Try adding $p_i$ to the convex hull $C$
   • If adding $p_i$ makes $C$ non-convex, then remove the last component of $C$ and repeat this check

Which data structure to use?

Need to be able to insert elements and remove in order of most-recent insertion

Can implement both operations in constant-time using a stack
1. Let $p_1$ be the point with the smallest $y$-coordinate (and smallest $x$-coordinate if multiple points have the same minimum $y$-coordinate)
2. Add $p_1$ to the convex hull $C$ (represented as a stack)
3. Sort all of the points based on their angle relative to $p_1$
4. For each of the points $p_i$ in sorted order:
   - Try adding $p_i$ to the convex hull $C$
   - If adding $p_i$ makes $C$ non-convex, then remove the last component of $C$ and repeat this check

Correctness?

See CLRS 33.3
Running Time of Graham’s Algorithm

1. Let $p_1$ be the point with the smallest $y$-coordinate (and smallest $x$-coordinate if multiple points have the same minimum-$y$ coordinate)  \(O(n)\)
2. Add $p_1$ to the convex hull $C$ (represented as a stack)  \(O(1)\)
3. Sort all of the points based on their angle relative to $p_1$  \(O(n \log n)\)
4. For each of the points $p_i$ in sorted order:
   • Try adding $p_i$ to the convex hull $C$  \(O(n)\)
   • If adding $p_i$ makes $C$ non-convex, then remove the last component of $C$ and repeat this check  \(O(n \log n)\)
We have essentially **reduced** the problem of computing a convex hull to the problem of sorting!  

Running Time of Graham’s Algorithm

1. Let $p_1$ be the point with the smallest $y$-coordinate (and smallest $x$-coordinate if multiple points have the same minimum-$y$ coordinate) \(O(n)\)

2. Add $p_1$ to the convex hull $C$ (represented as a stack) \(O(1)\)

3. Sort all of the points based on their angle relative to $p_1$ \(O(n \log n)\)

4. For each of the points $p_i$ in sorted order:
   - Try adding $p_i$ to the convex hull $C$ \(O(n)\)
   - If adding $p_i$ makes $C$ non-convex, then remove the last component of $C$ and repeat this check
Convex Hull to Sorting Reduction

Map instances of problem $A$ to instances of $B$

$O(n)$

Map solutions of problem $B$ to solutions of $A$

$O(n)$

Reduction

convex hull $\leq$ sorting

convex hull can be reduced to sorting in $O(n)$ time
1. Let $p_1$ be the point with the smallest $y$-coordinate (and smallest $x$-coordinate if multiple points have the same minimum-$y$ coordinate) \(O(n)\)
2. Add $p_1$ to the convex hull $C$ (represented as a stack) \(O(1)\)
3. Sort all of the points based on their angle relative to $p_1$ \(O(n \log n)\)
4. For each of the points $p_i$ in sorted order:
   - Try adding $p_i$ to the convex hull $C$ \(O(n)\)
   - If adding $p_i$ makes $C$ non-convex, then remove the last component of $C$ and repeat this check

Running time of Graham’s algorithm: same as best sorting algorithm

Can we do better (without going through sorting)?
Running Time of Graham’s Algorithm

1. Let \( p_1 \) be the point with the smallest \( y \)-coordinate (and smallest \( x \)-coordinate if multiple points have the same minimum-\( y \) coordinate) \[ O(n) \]
2. Add \( p_1 \) to the convex hull \( C \) (represented as a stack) \[ O(1) \]
3. Sort all of the points based on their angle relative to \( p_1 \) \[ O(n \log n) \]
4. For each of the points \( p_i \) in sorted order:
   - Try adding \( p_i \) to the convex hull \( C \) \[ O(n) \]
   - If adding \( p_i \) makes \( C \) non-convex, then remove the last component of \( C \) and repeat this check

**Trivial lower bound:** \( \Omega(n) \)

Can we do better (without going through sorting)?
Worst Case Lower Bound Proofs

If we know that $A$ cannot be solved in $O(f(n))$ time, then $A$ is not a harder problem than $B$.

If there is a $O(f(n))$ reduction from $A$ to $B$, then $B$ cannot be solved in $O(f(n))$ time.

$A \leq B$
Sorting to Convex Hull Reduction

If this is $\Omega(n \log n)$

Then this is $\Omega(n \log n)$
Observe: convex hull consists of a subset of points in a prescribed order
Can we use this to sort?

Observe: convex hull consists of a subset of points in a prescribed order
Can we use this to sort?

To get full sorted ordering, convex hull should contain all of the points (i.e., values in the set)

Want order of points in convex hull to be the order of elements in sorted order

Observe: convex hull consists of a subset of points in a prescribed order
• **Goal:** need a way to map list of (numeric) values onto a convex hull instance
  – Given: -2 1 -3 0 2 3 -1
  – Create some convex hull instance where all points on the convex hull
Given: \(-2 \quad 1 \quad -3 \quad 0 \quad 2 \quad 3 \quad -1\)

\(x \Rightarrow (x, x^2)\)

sorting CH

Creates a parabola!
Claim: order of elements in convex hull coincide with elements in sorted order

Given:  \(-2\ 1\ -3\ 0\ 2\ 3\ -1\)
• Reduction Construction
  – Convert each element to a 2D point, \( x \Rightarrow (x, x^2) \) \( O(n) \)
  – Run Convex Hull algorithm
  – Find minimum \( x \)-coordinate in convex hull points \( O(n) \)
  – List convex hull points’ \( x \)-coordinate in prescribed order \( O(n) \)

Reduction cost: \( O(n) \)
Convex Hull to Sorting Reduction

Create 2D points

\[ O(n) \]

Find minimum \( x \)

Read CH solution in order

\[ O(n) \]

sorting numeric values \( \leq \) convex hull

sorting numeric values can be reduced to convex hull in \( O(n) \) time
Convex Hull to Sorting Reduction

Create 2D points

\[ O(n) \]

Find minimum \( x \)

Read CH solution in order

\[ O(n) \]

If this is \( \Omega(n \log n) \)

Then this is \( \Omega(n \log n) \)

Then this is \( \Omega(n \log n) \)

If this is \( \Omega(n \log n) \)

-2 1 -3 0 2 3 -1

-3 -2 -1 0 1 2 3

sorting numeric values \( \leq \) convex hull

sorting numeric values can be reduced to convex hull in \( O(n) \) time
Conclusion: a lower bound for sorting translates into one for convex hull

Our lower bound for sorting: $\Omega(n \log n)$ for comparison-based sorts

Our reduction is not a comparison sort algorithm

$\Omega(n \log n)$ lower bound for sorting also holds in an “algebraic decision tree model” (i.e., decisions can be an algebraic function of inputs)

Implies $\Omega(n \log n)$ lower bound for computing convex hull in this model
Conclusion: a lower bound for sorting translates into one for convex hull

Our lower bound for sorting: $\Omega(n \log n)$ for comparison-based sorts

Our reduction is not a comparison sort algorithm

$\Omega(n \log n)$ lower bound for sorting is also valid in an "algebraic decision tree model" (i.e., decisions can be an algebraic function of inputs)

Implies $\Omega(n \log n)$ lower bound for computing convex hull in this model

In fact, this lower bound holds even for algorithms that just identify the set of points on the convex hull (and not necessarily their order)!
Jarvis’ Algorithm (Gift Wrapping Method)

**Idea:** Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion.
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

**Idea:** Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

**Idea:** Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion

Diagram shows a set of points in a two-dimensional space, with an arrow indicating the counter-clockwise wrapping.
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

**Idea:** Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Idea: Start with extremal point and “wrap” points in counter-clockwise fashion
Jarvis’ Algorithm (Gift Wrapping Method)

Can find the “next” point using a linear scan

**Number of iterations**: number of points on convex hull

**Run time**: $O(nh)$ where $h$ is the number of points on the convex hull
Jarvis’ Algorithm (Gift Wrapping Method)

- Can find the “next” point using a linear scan
- Number of iterations: number of points on convex hull
- Run time: $O(nh)$ where $h$ is the number of points on the convex hull
- Output-dependent running time (similar to Ford-Fulkerson)
  - Can be better than Graham’s Algorithm when $h \ll \log n$
  - **Worst case:** $h = n$, so $O(n^2)$
GRAHAM SCAN: $O(n \log n)$, OR JARVIS MARCH $O(nh)$?

Why don't we have both?

CHAN'S ALGORITHM: $O(n \log h)$
Chan’s Algorithm
Chan’s Algorithm

Divide into smaller subsets
Chan’s Algorithm

Use Graham’s Algorithm to **conquer** the smaller subsets
Chan’s Algorithm

Use Jarvis’ Algorithm to **combine** the solutions to the smaller subsets
Chan’s Algorithm

Use Jarvis’ Algorithm to **combine** the solutions to the smaller subsets
Chan’s Algorithm

Combines Graham’s Algorithm and Jarvis’ Algorithm
Given points $P$, size of subsets $m$, guess of number of hull points $H$

Partition $P$ into subsets $P_1, P_2, ..., P_{\lfloor n/m \rfloor}$ of size at most $m$
for $i = 1, ..., \lfloor n/m \rfloor$
    Compute $\text{conv}(P_i)$ using Graham’s Algorithm, store in counter-clockwise order
$p_0 \leftarrow (0, -\infty)$
$p_1 \leftarrow$ rightmost point of $P$
for $k = 1, ..., H$ (each hull point)
    for $i = 1, ..., \lfloor n/m \rfloor$ (each subset)
        Use binary search on $\text{conv}(P_i)$ to find point $q_i$
        that maximizes the angle $\angle p_{k-1} p_k q_i$
        $p_{k+1} \leftarrow q \in \{q_1, q_2, ..., q_{\lfloor n/m \rfloor}\}$ that maximizes the angle $\angle p_{k-1} p_k q$ (Jarvis’ Algorithm)
        if $p_{k+1} = p_1$, then return $\text{conv}(P) = \{p_1, p_2, ..., p_k\}$

$O(n \log h)$
Where $h$ is the number of hull points

https://link.springer.com/content/pdf/10.1007%2FBF02712873.pdf