CS 4102: Algorithms

Lecture 26: Convex Hull

David Wu

Fall 2019

Today's Keywords

Reductions and lower bounds

Convex hull

Graham's algorithm (Graham scan)

Jarvis' algorithm (Jarvis march)

Chan's algorithm

CLRS Readings: Chapter 33.3

Homework

HW9, **HW10C** due Thursday, December 5, 11pm

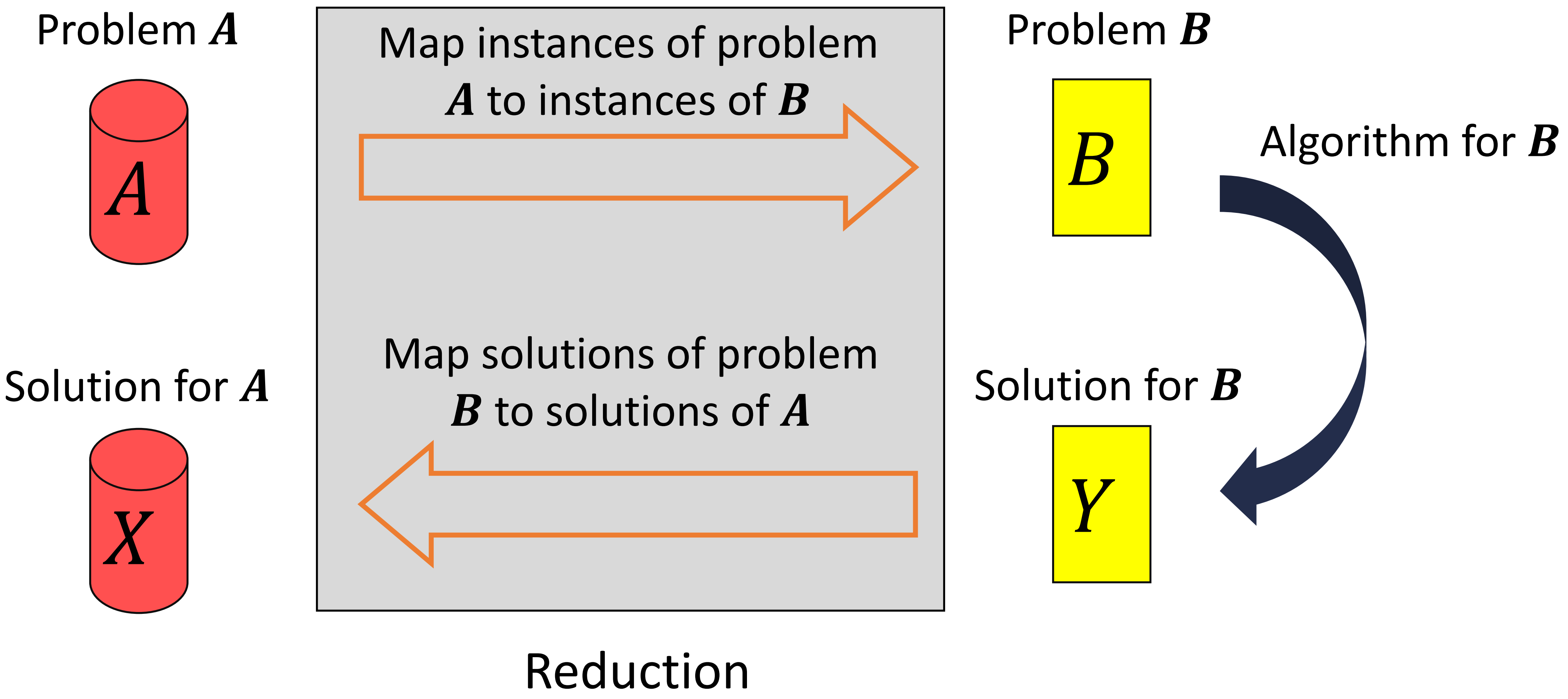
- Graphs, Reductions
- Written (LaTeX)

Final Exam

Monday, December 9, 7pm in Olsson 120

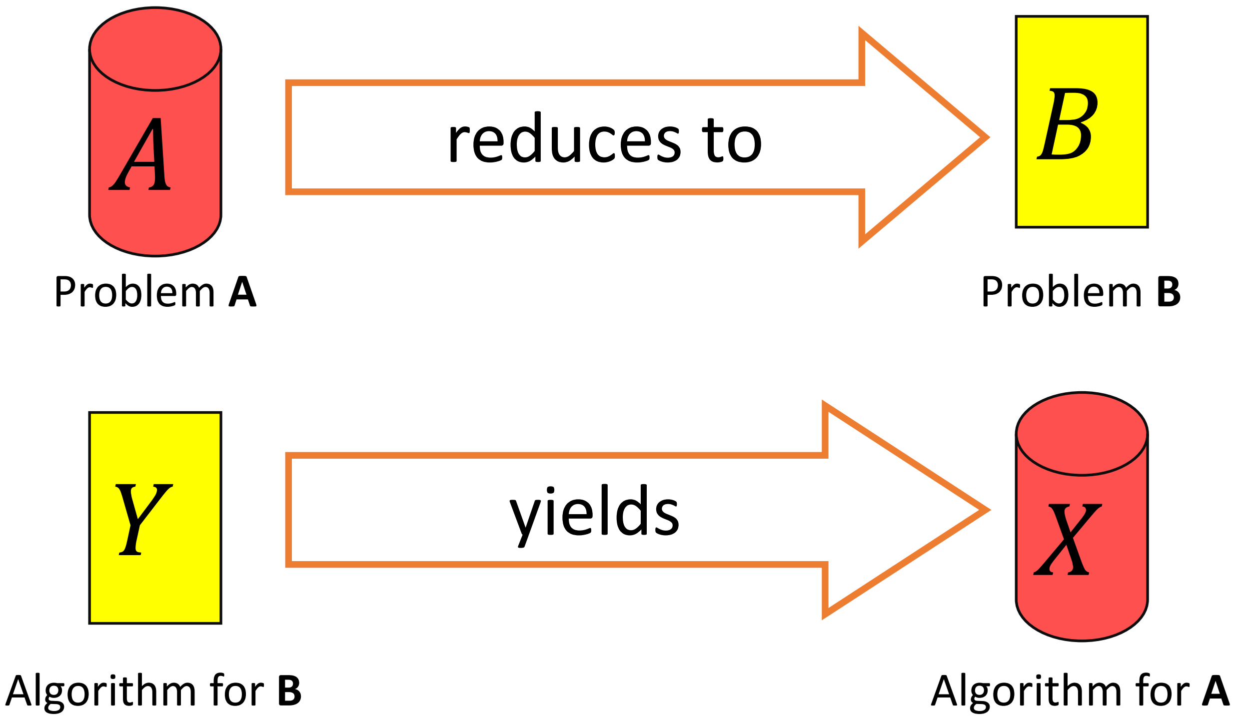
- Practice exam coming soon
- Review session likely the weekend before
- SDAC: Please sign-up for a time on December 9

Reductions



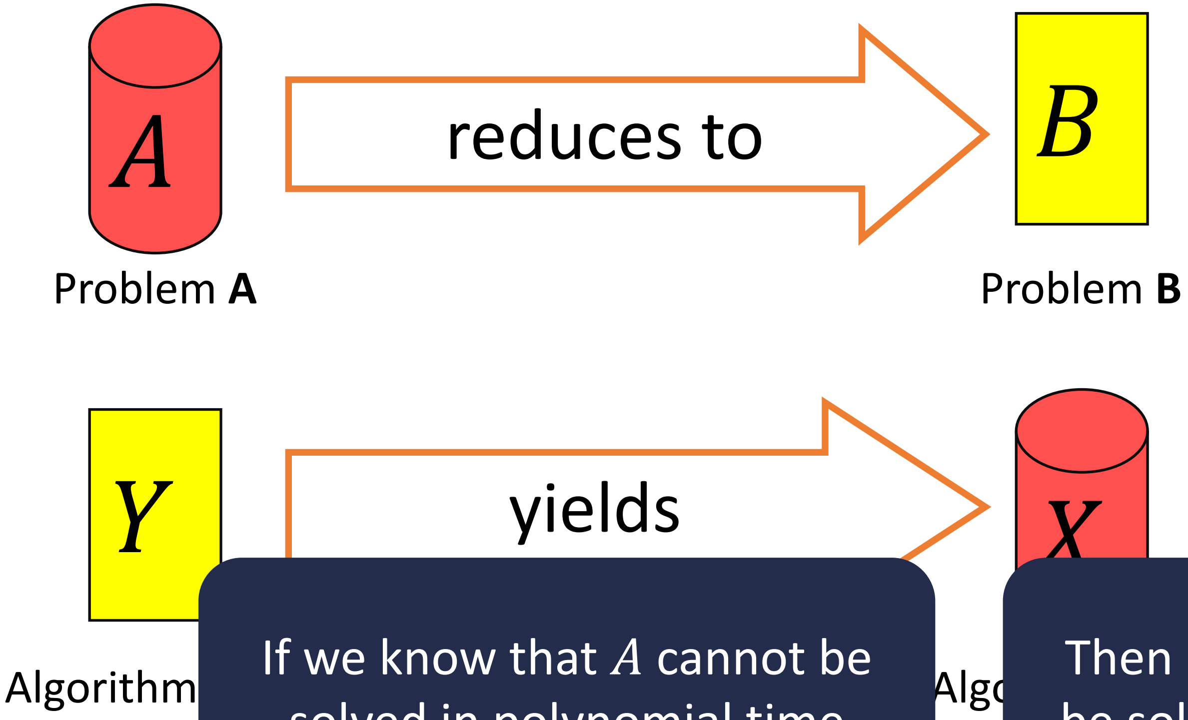
$\boldsymbol{A} \leq \boldsymbol{B}$: there is a reduction from $\boldsymbol{A}$ to $\boldsymbol{B}$

Understanding Reductions



Implication: A is no more difficult than B
(denoted $A \leq B$)

Worst-Case Lower Bounds via Reductions

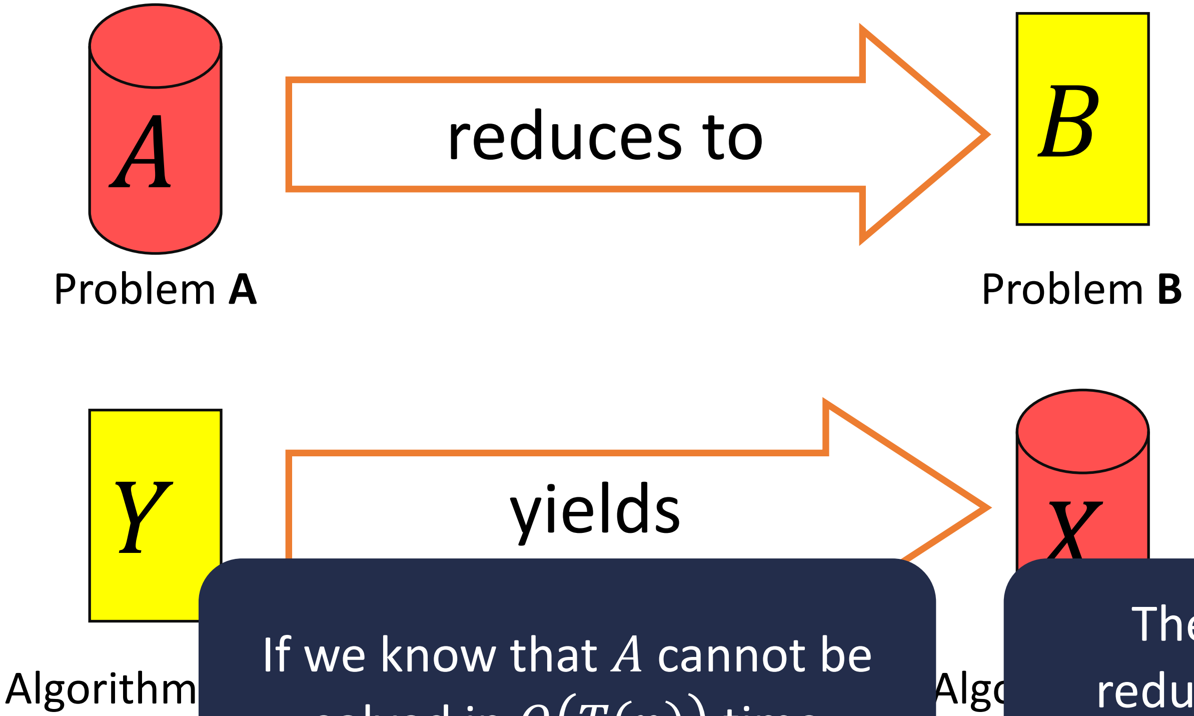


If we know that A cannot be solved in polynomial time

Then problem B also cannot be solved in polynomial time

Implication: A is <u>no more difficult</u> than B
(denoted $A \leq B$)

Worst-Case Lower Bounds via Reductions



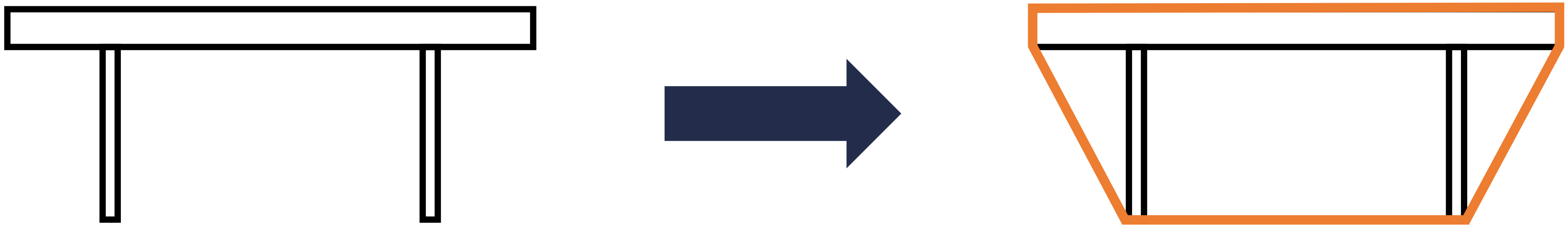
If we know that A cannot be solved in $O(T(n))$ time

Then, if there is a $O(T(n))$ reduction from A to B, then B cannot be solved in O$(T(n))$ time

Implication: A is no more difficult than B

(denoted $A \leq B$)

The Convex Hull Problem



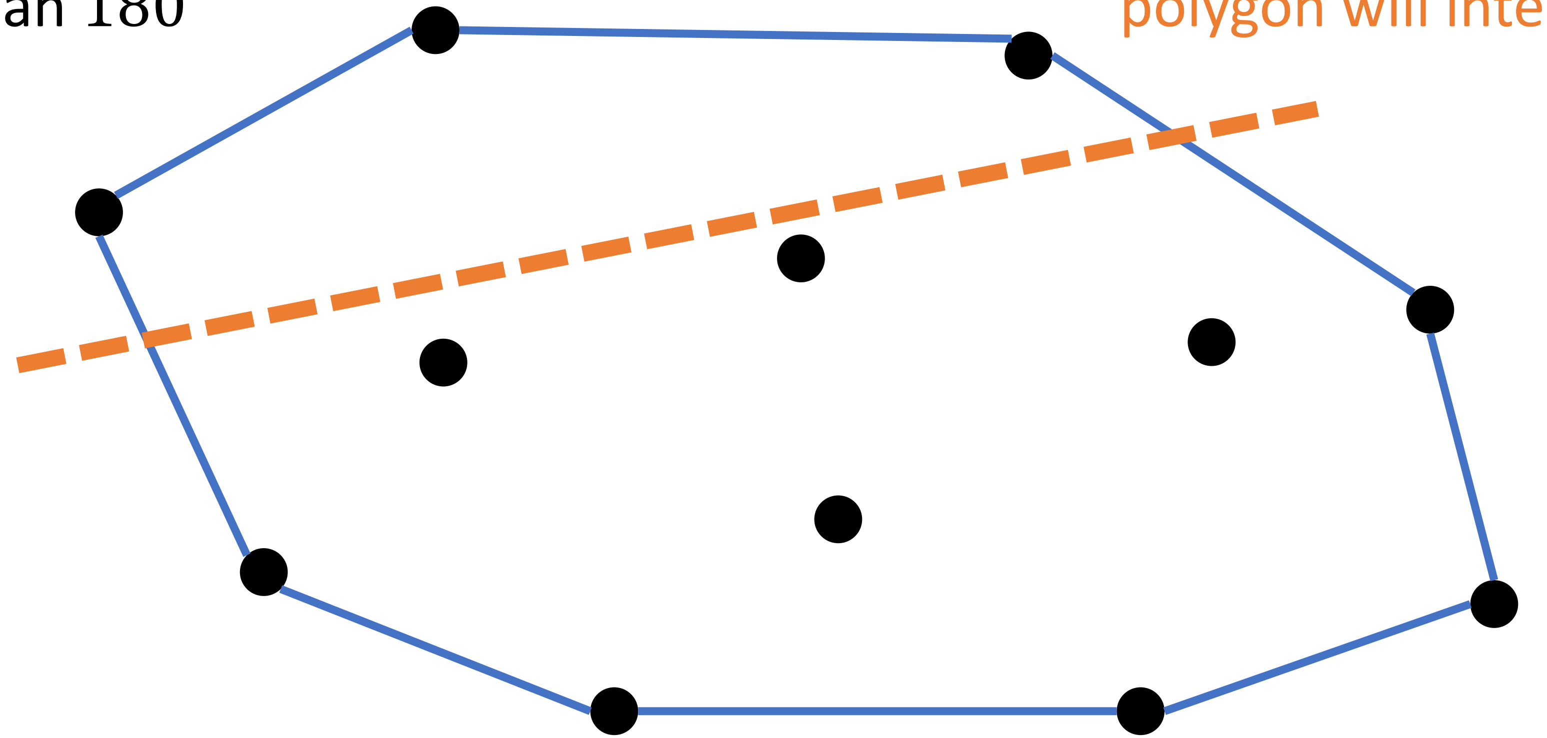
Problem: find the smallest convex polygon that bounds a shape (or more generally, a collection of points)

Example application: collision detection in computer graphics; 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

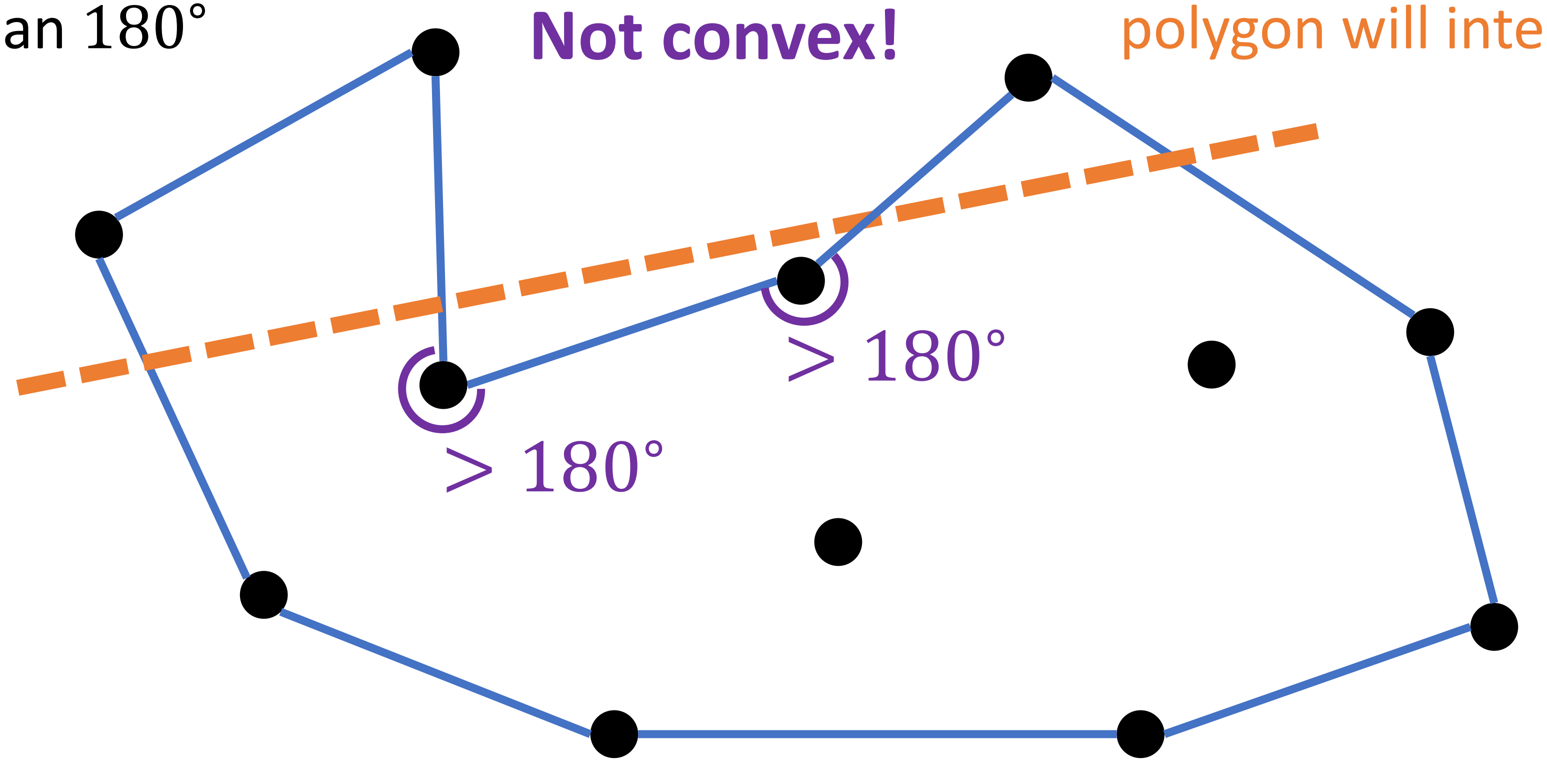


Problem: given a set of n points, find the smallest convex polygon such that every point is either on the boundary or the interior of the polygon

The Convex Hull Problem

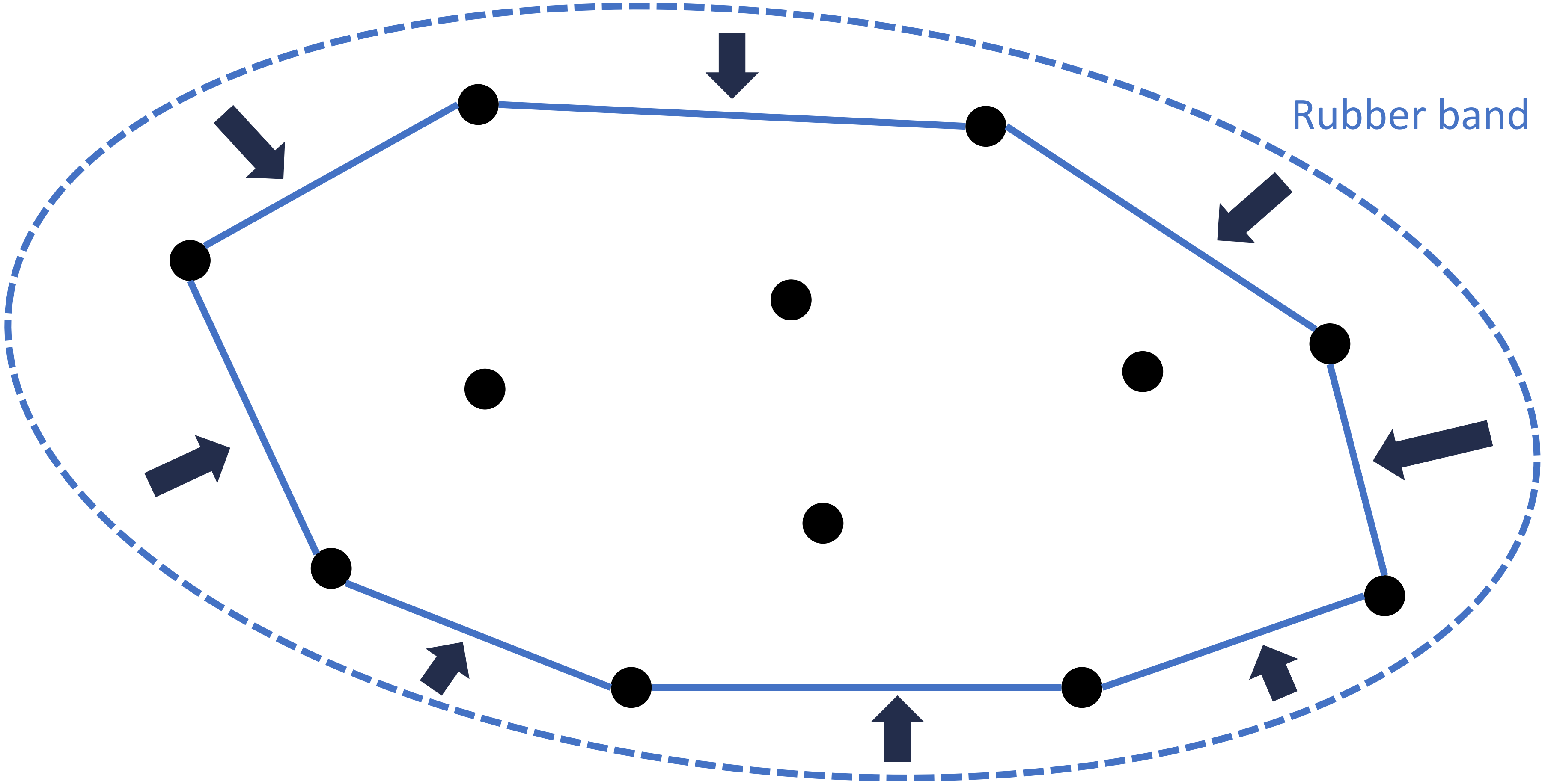
Convex polygon: all interior angles are less than $180°$

Equivalently: line drawn through polygon will intersect exactly twice



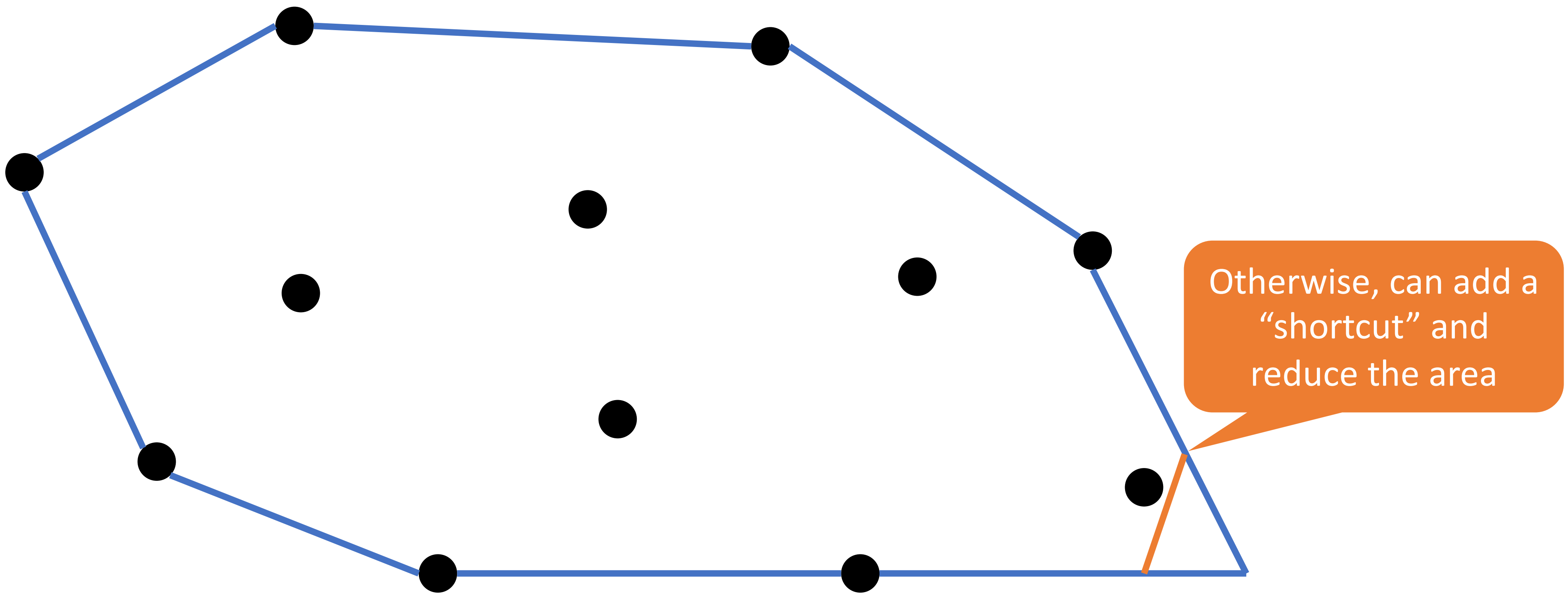
Problem: given a set of n points, find the smallest convex polygon such that every point is either on the boundary or the interior of the polygon

The Convex Hull Problem



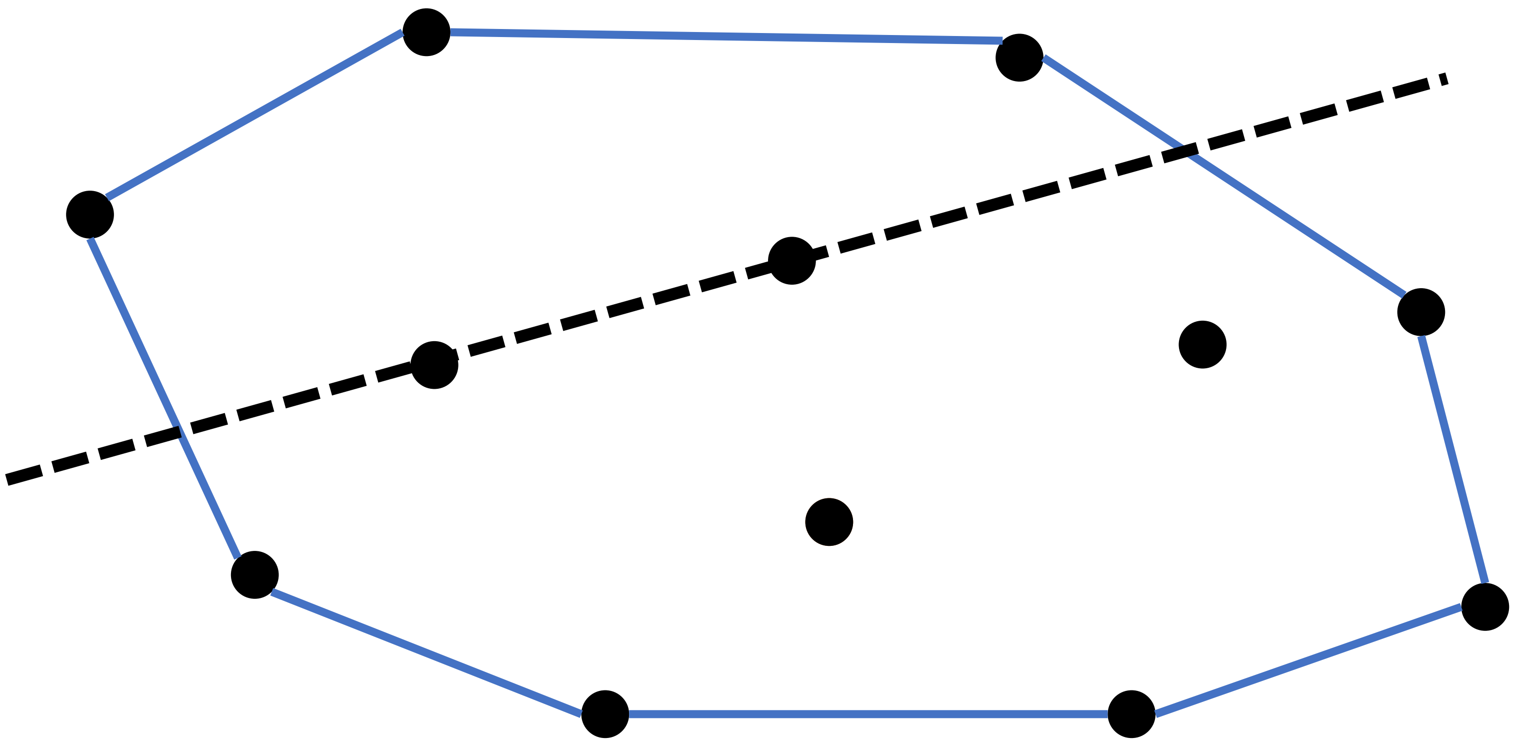
Rubber band analogy: imagine the points are nails sticking out of a board and wrapping a rubber band to encompass the nails; convex hull is resulting shape

The Convex Hull Problem



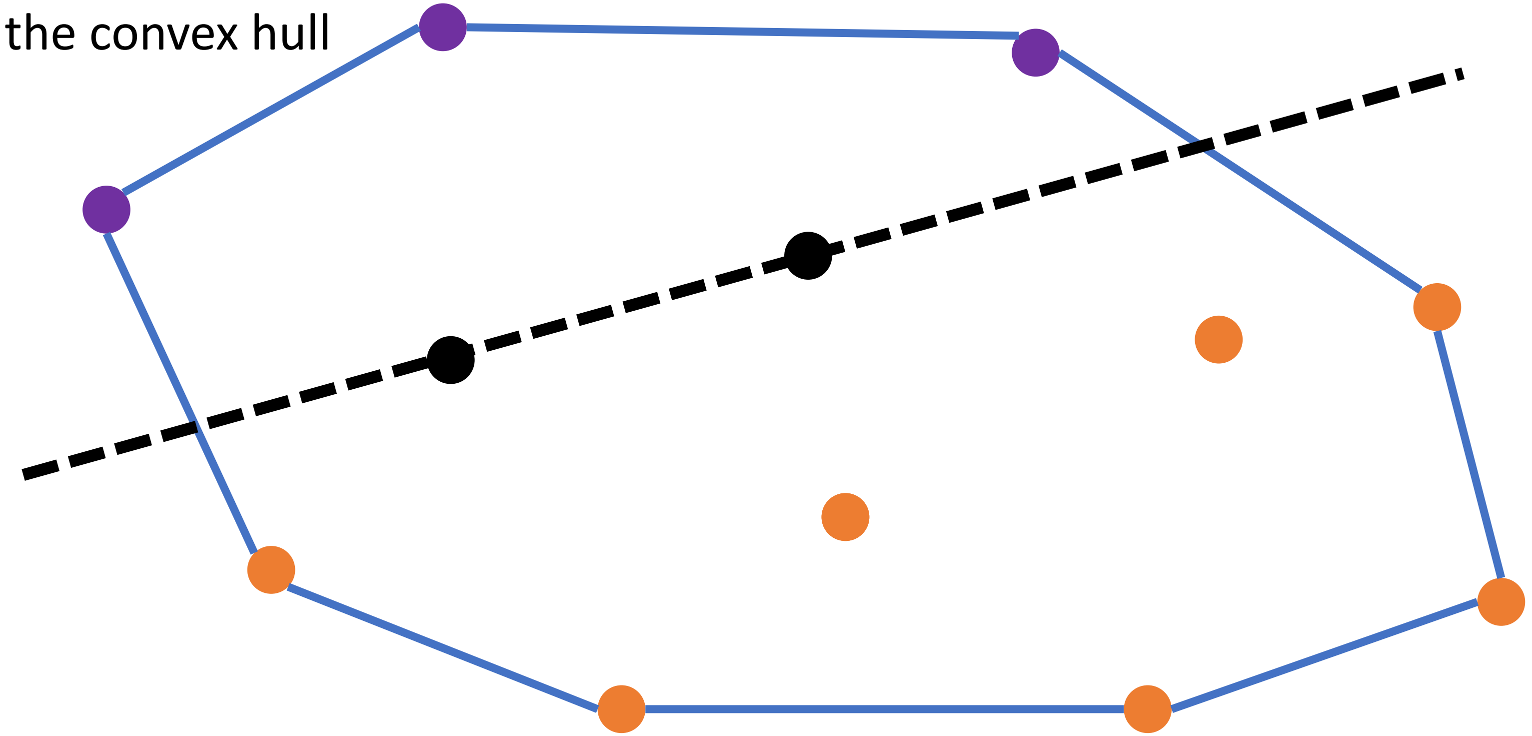
Observation: every point on the convex hull is one of the input points

A Brute Force Approach



A Brute Force Approach

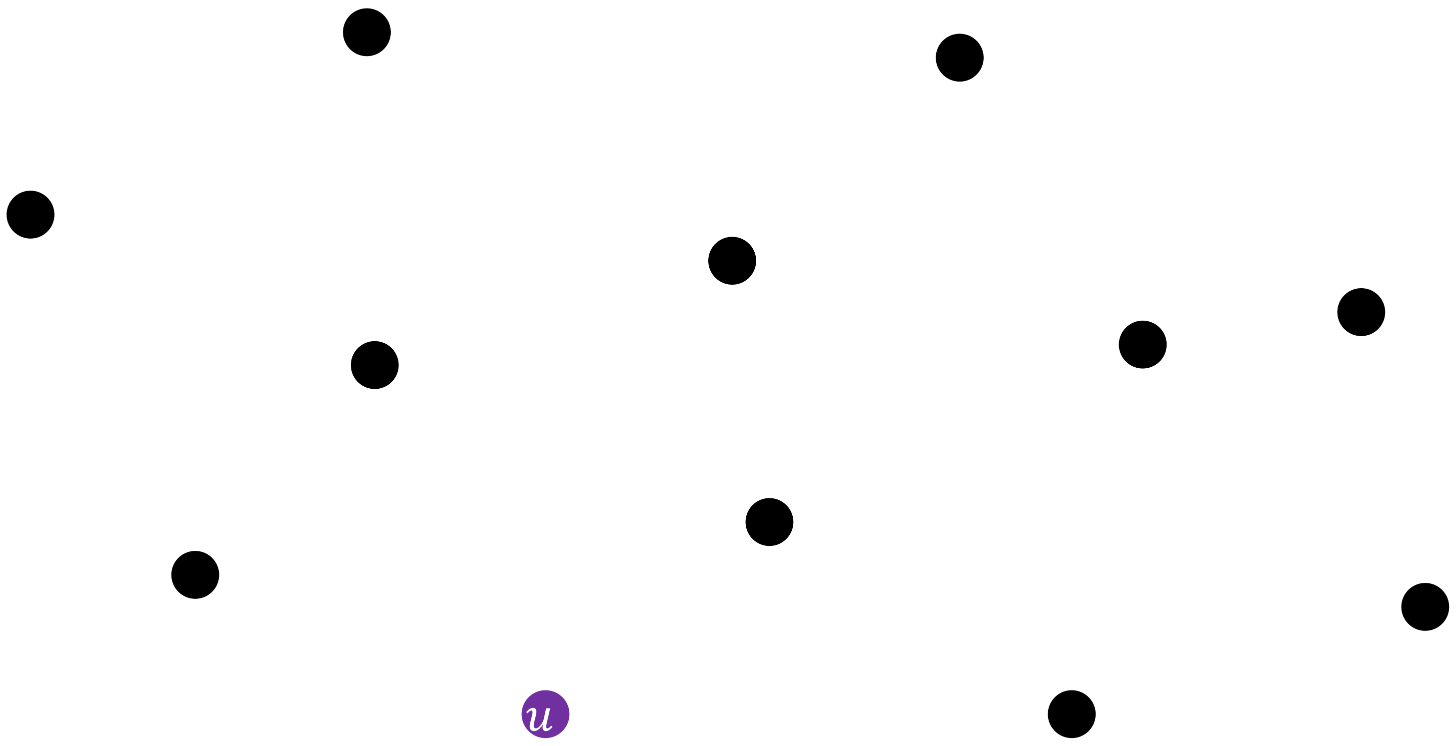
Observation: if there are points on <u>both</u> sides of the line, then the pair cannot be an edge in the convex hull



Run-time: $O(n^3)$

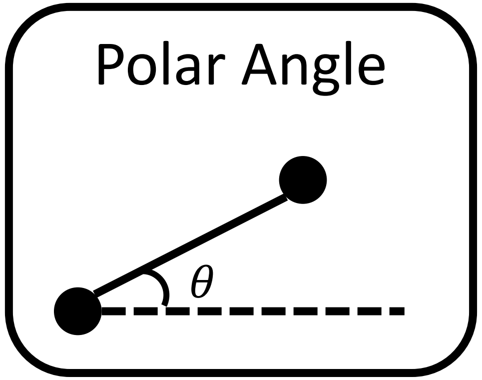
Brute force approach: for every pair of points, check if all other points are on the same side of the line

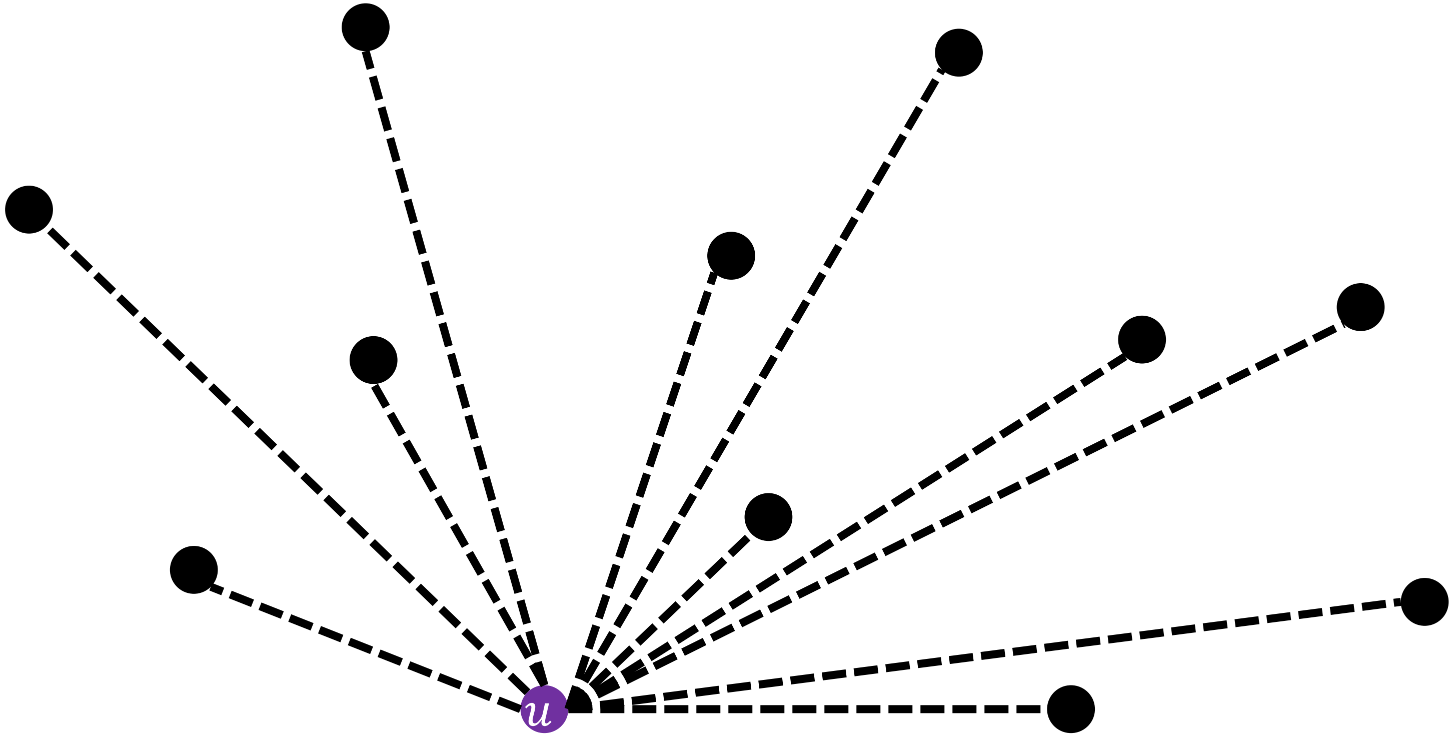
Graham's Algorithm



Observation: Extremal points must be part of the convex hull (e.g., bottom-most point, left-most point, etc.)

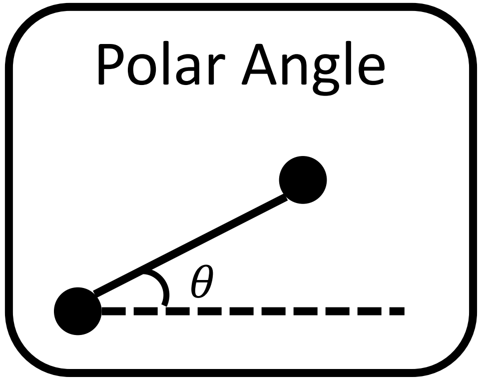
Graham's Algorithm

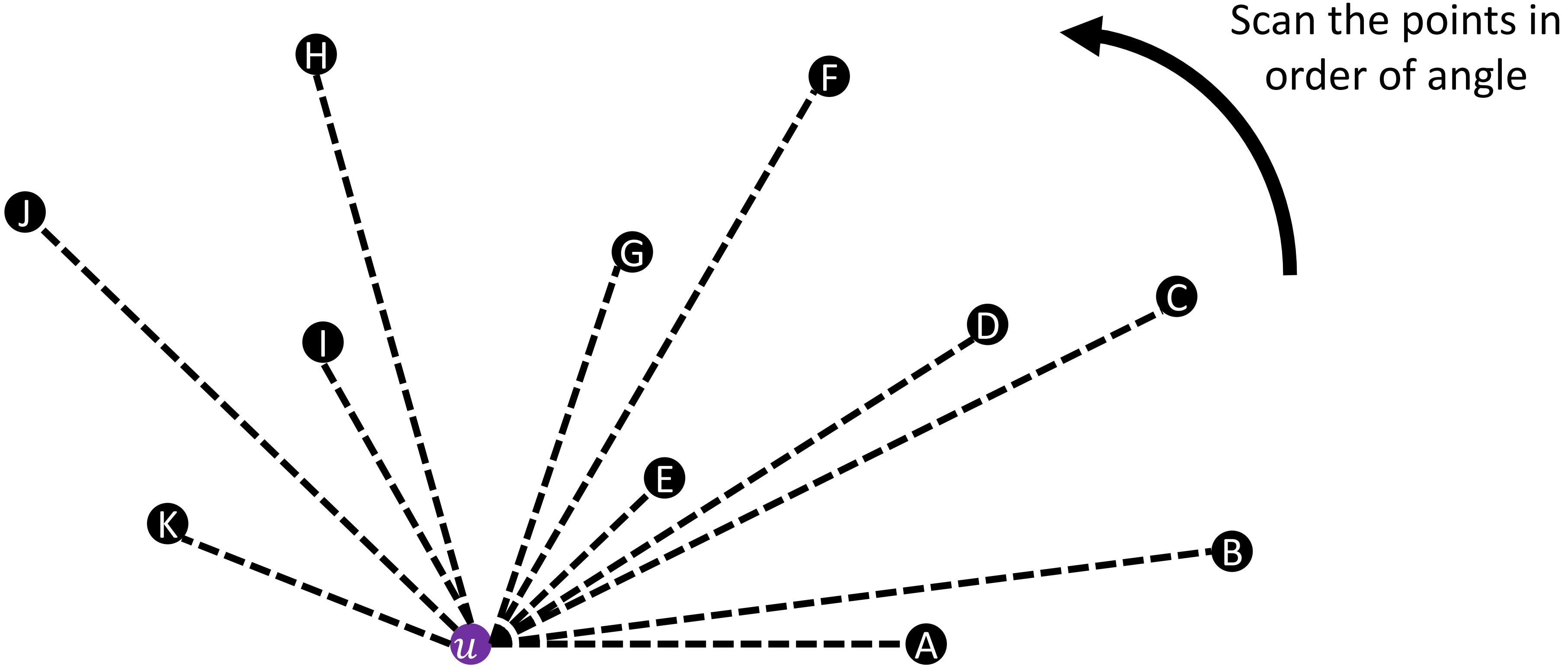




Consider the (polar) angle formed between base point u and every other point

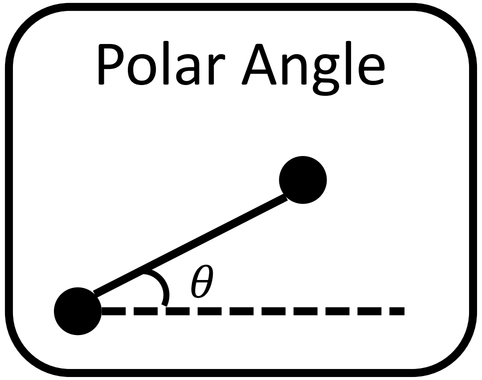
Graham’s Algorithm

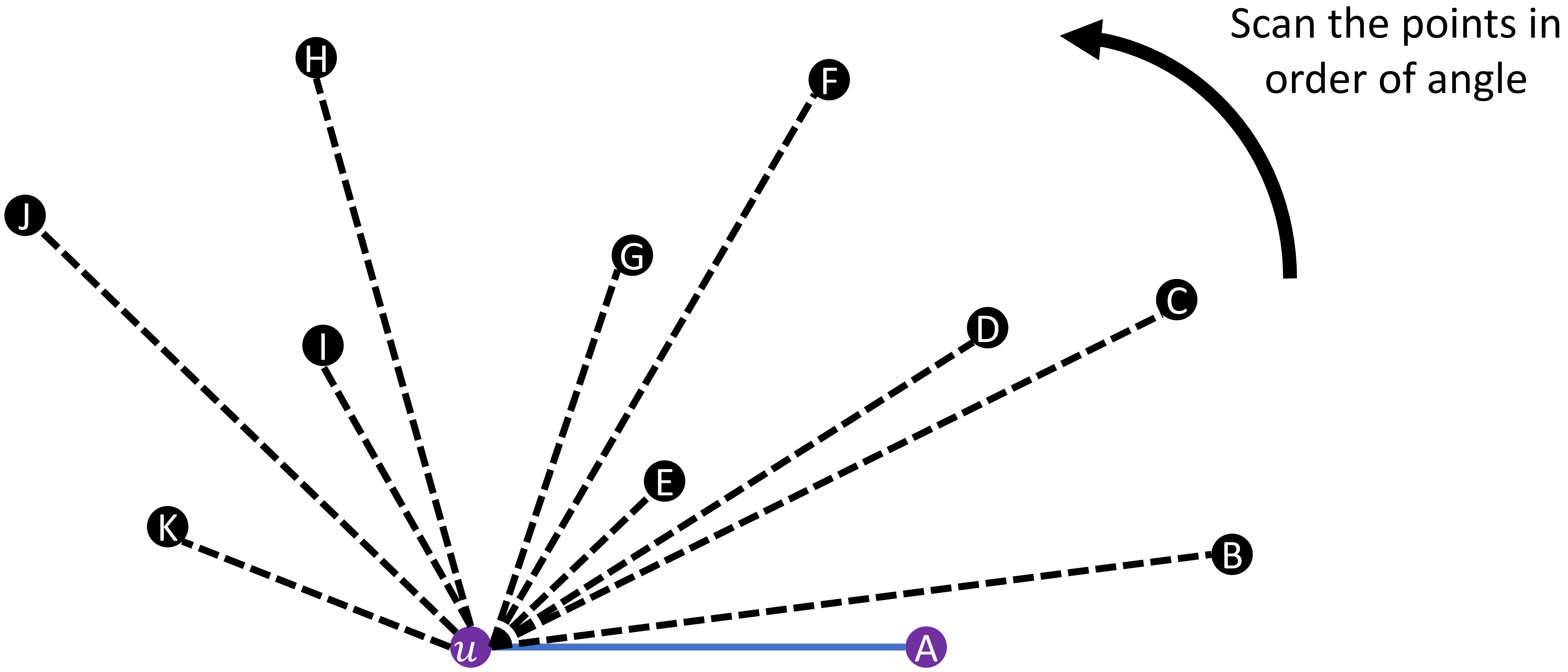




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

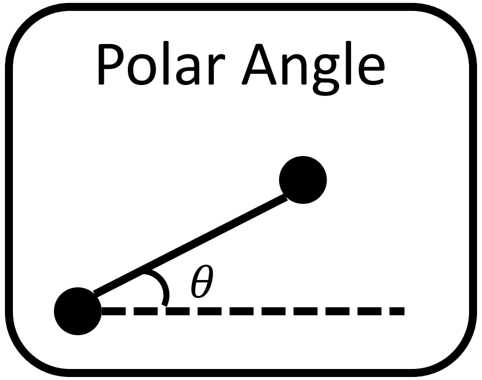
Graham's Algorithm

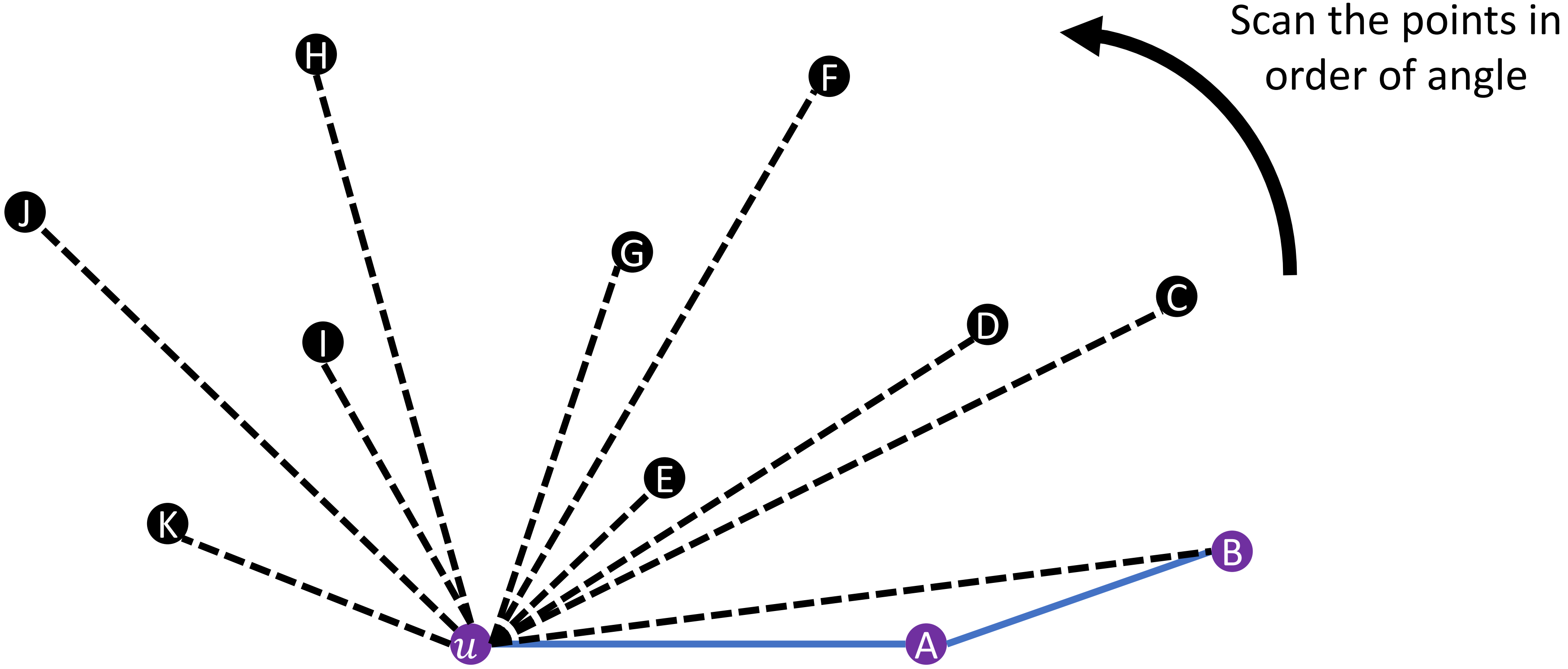




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm

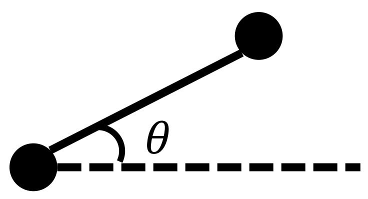


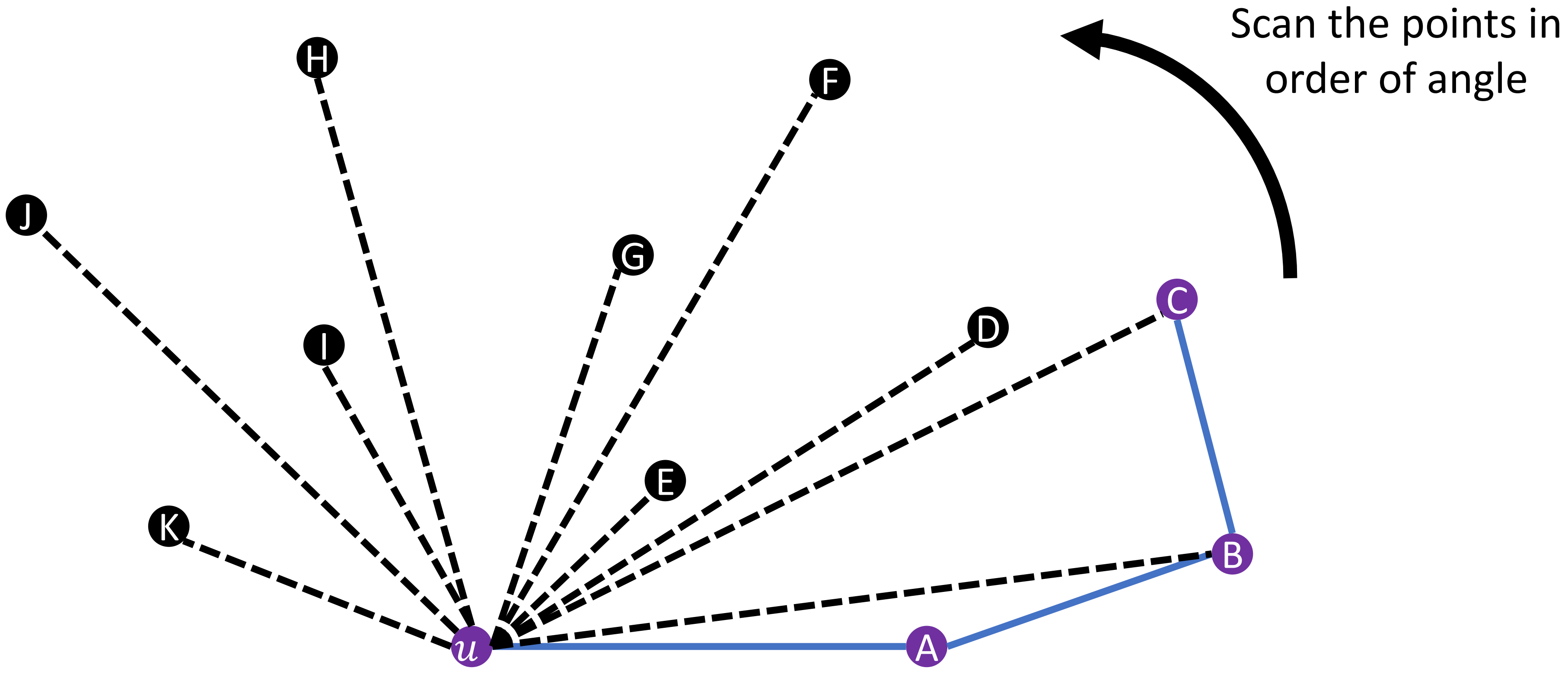


Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham’s Algorithm

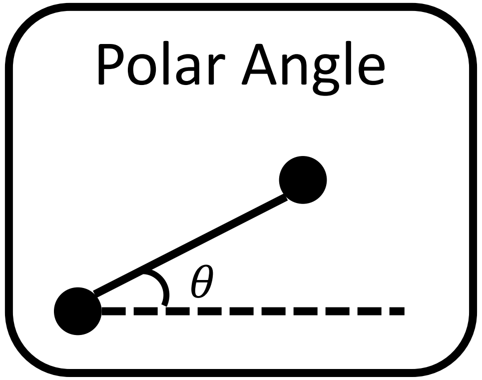
Polar Angle

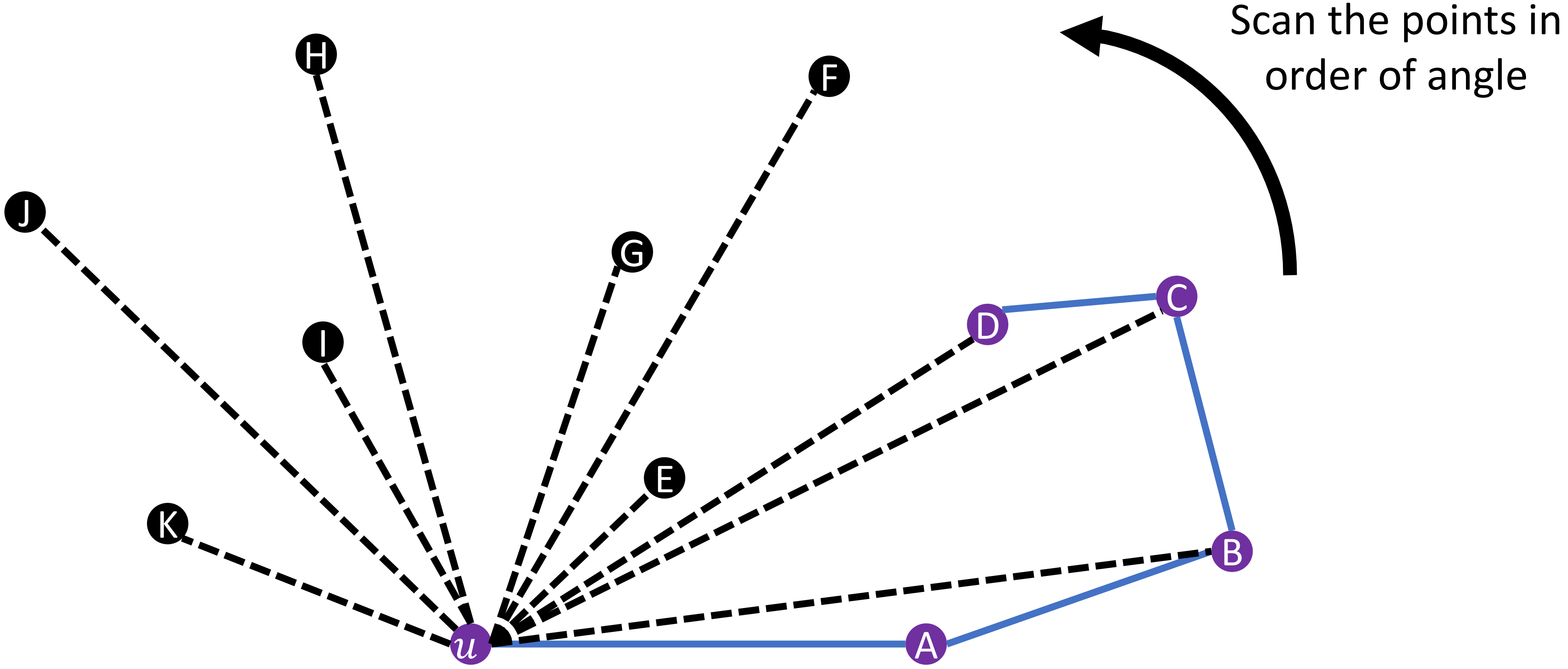




Idea: In order of angle, add points to the convex hull as long as it <u>preserves</u> convexity

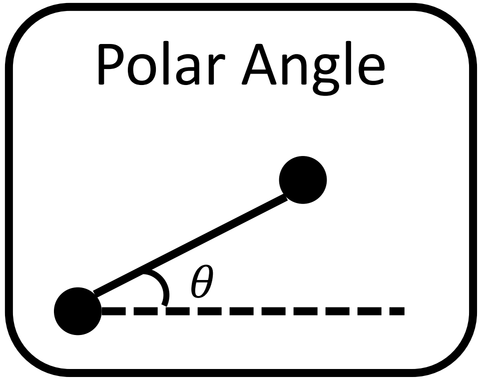
Graham's Algorithm

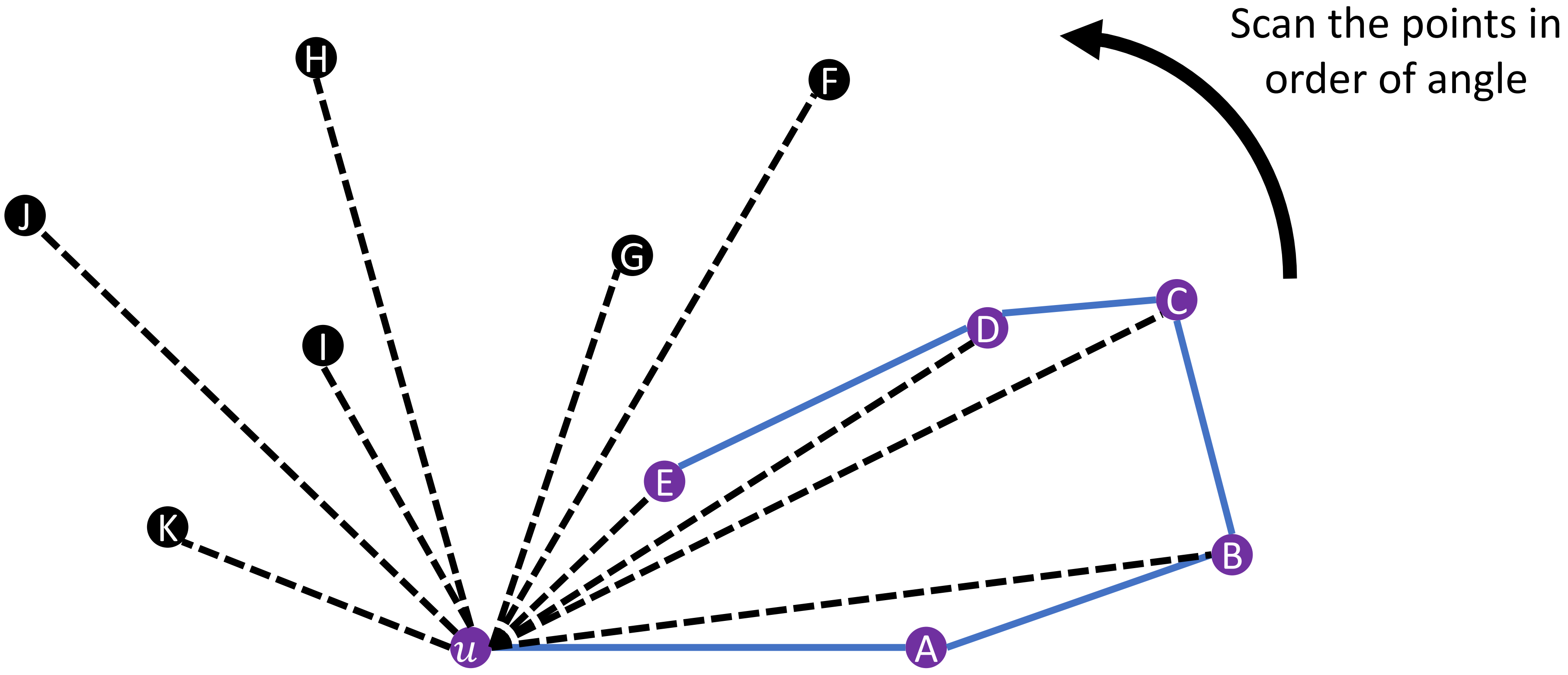




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

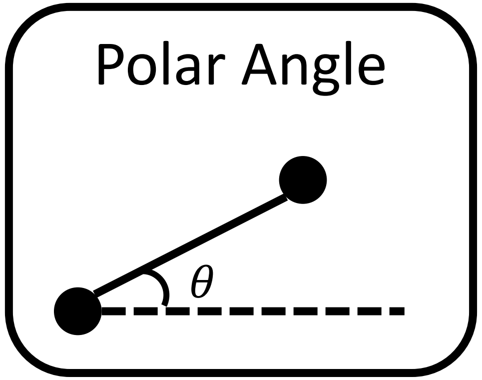
Graham's Algorithm



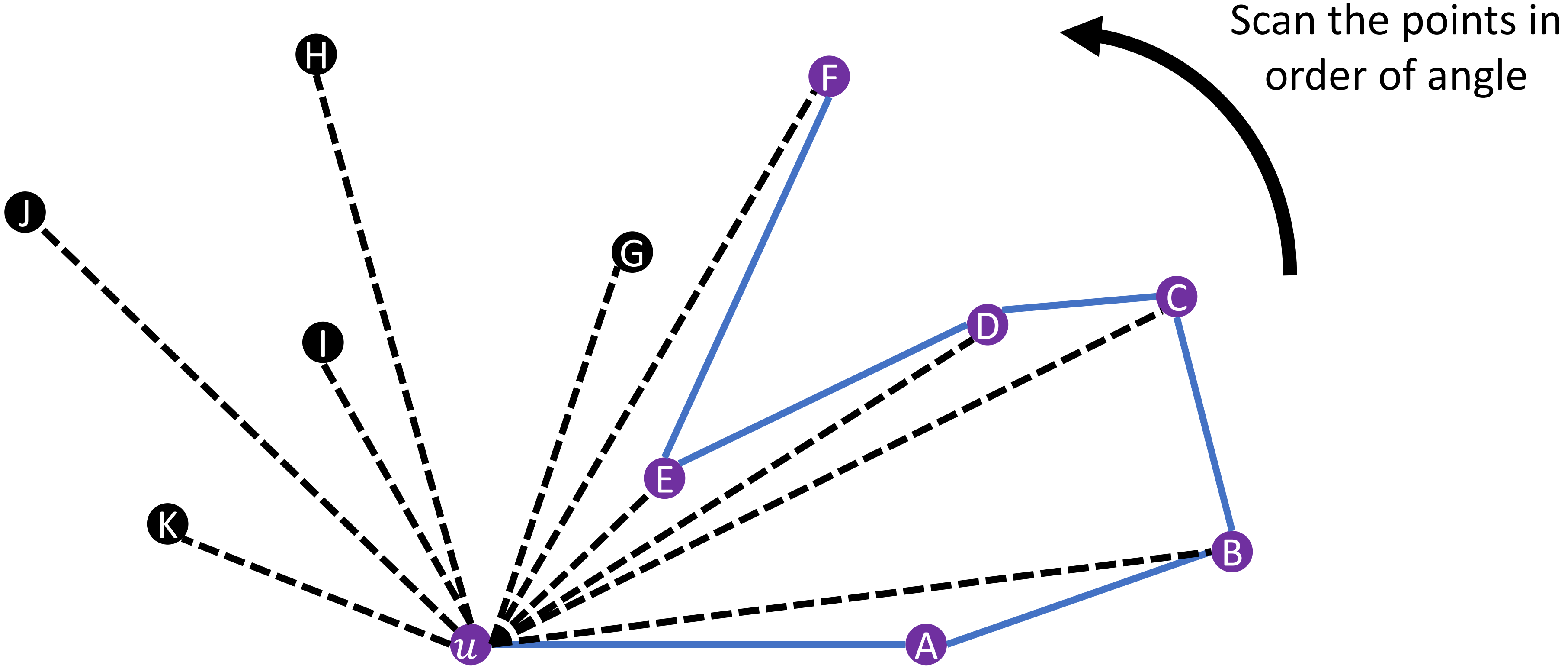


Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm

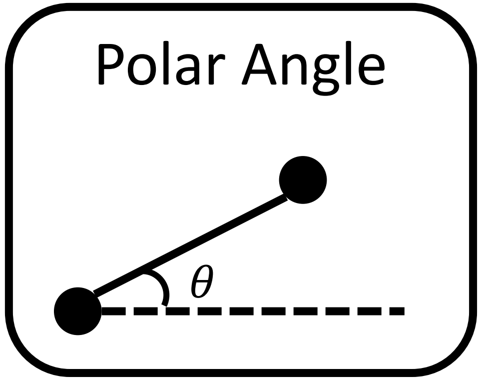


Not convex anymore!

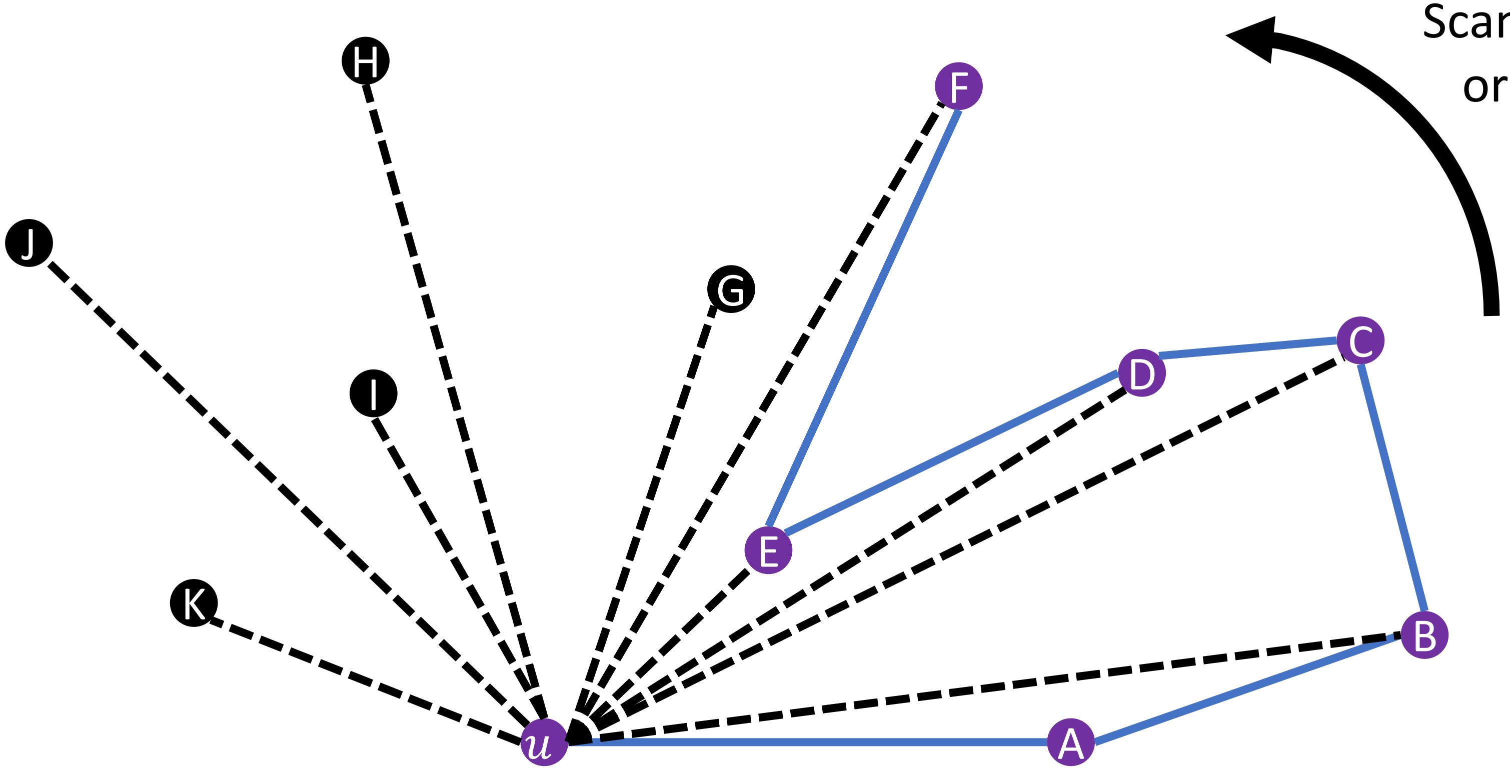


Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm



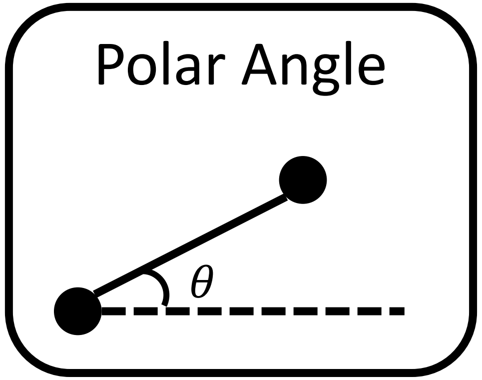
Not convex anymore!

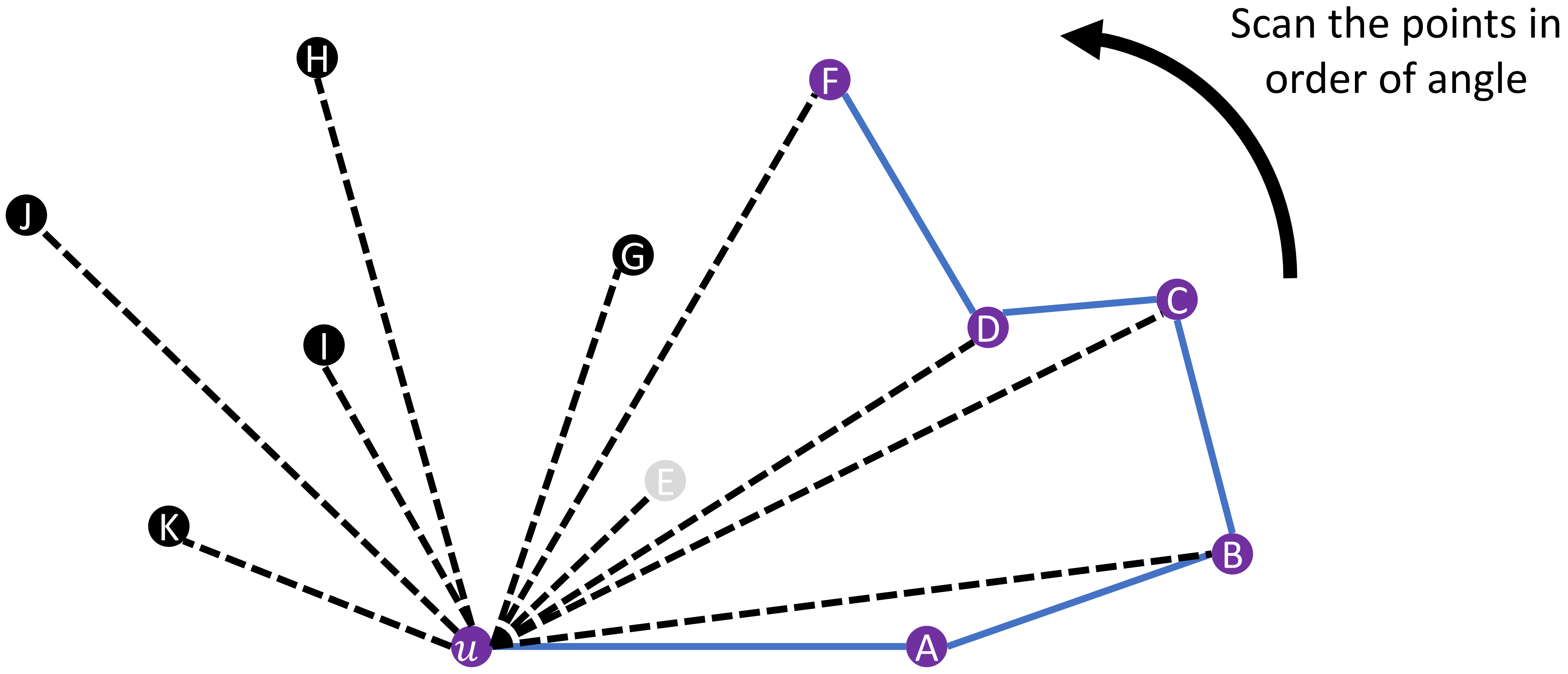


Scan the points in order of angle

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

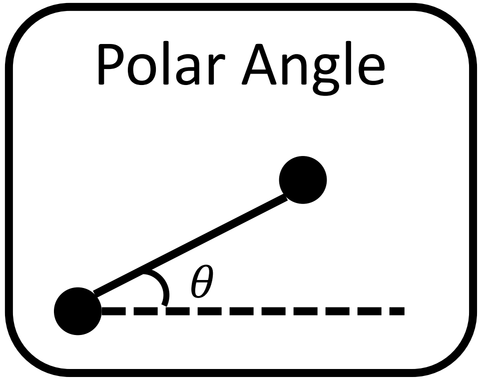
Graham's Algorithm



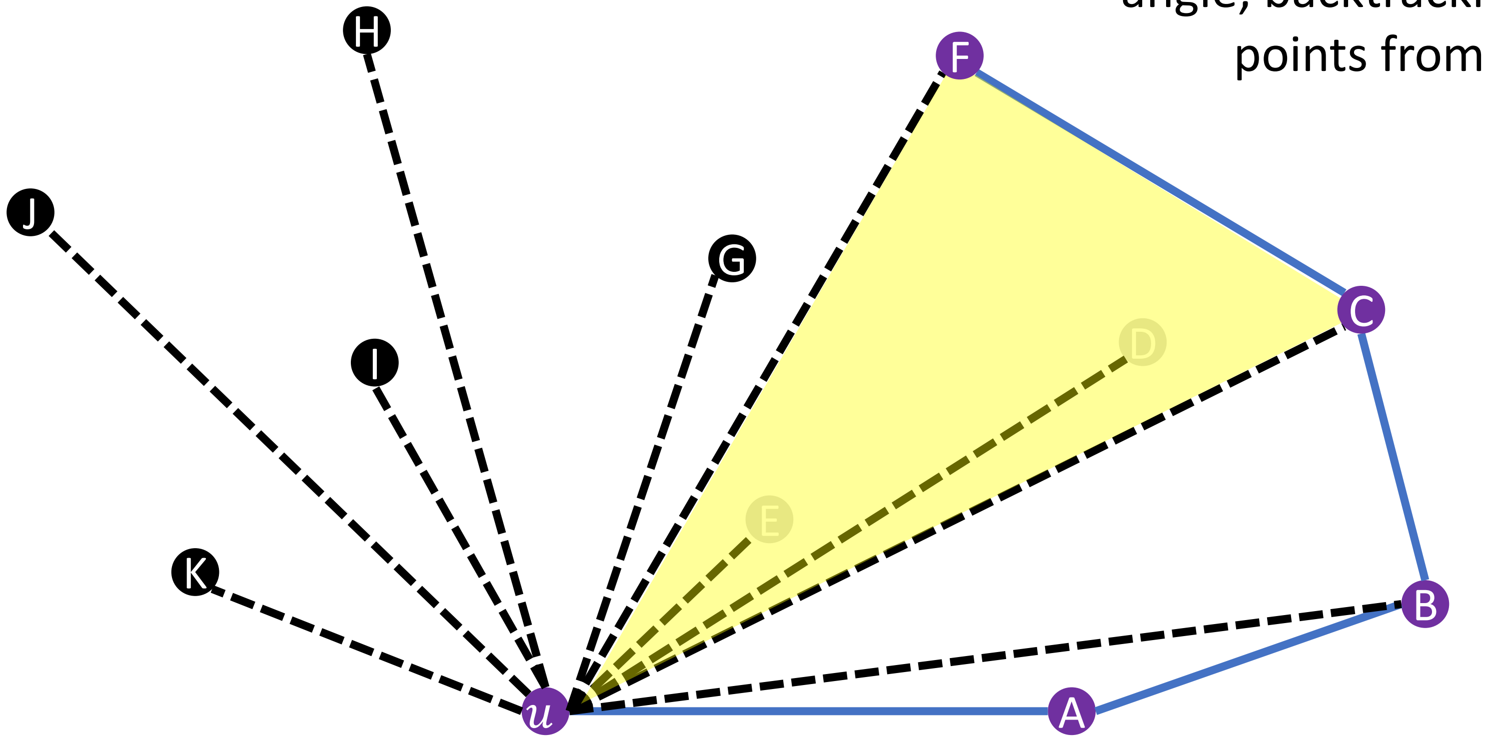


Idea: Try extending the convex hull from the previous vertex if we are unable to extend from the current one

Graham’s Algorithm

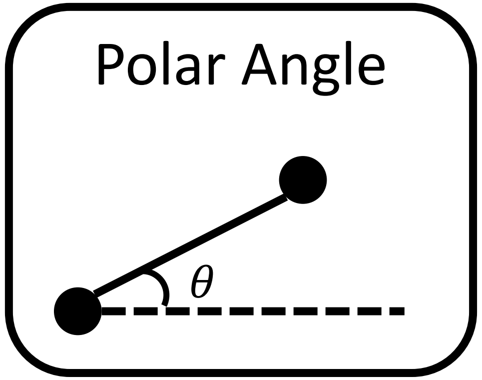


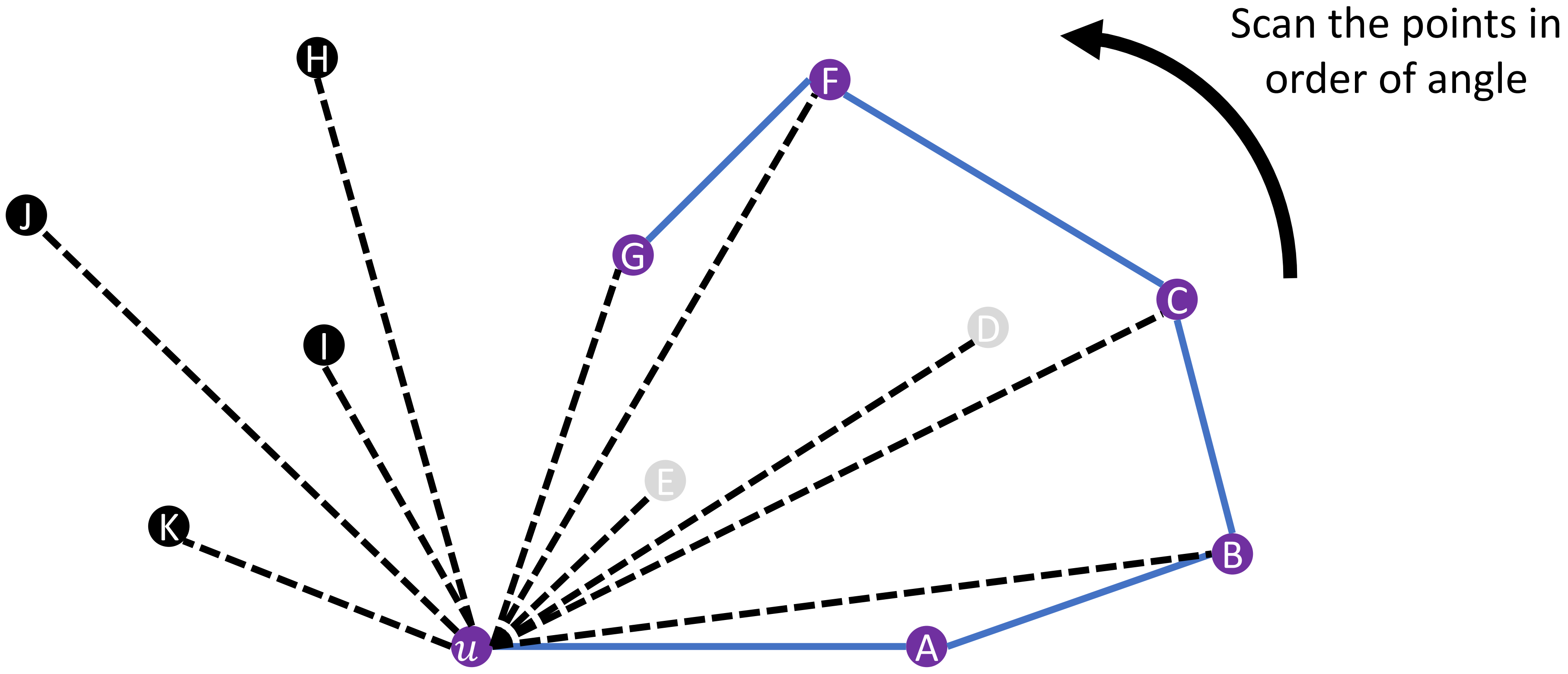
Observe: since points are sorted by angle, backtracking will <u>never</u> remove points from the convex hull



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

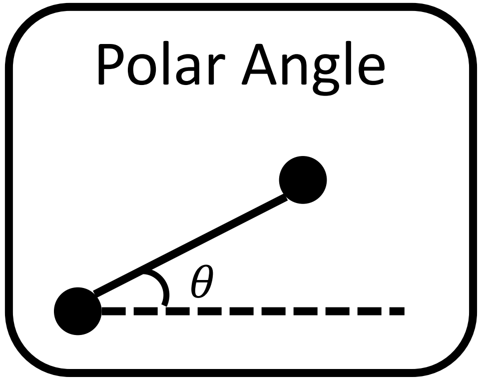
Graham's Algorithm



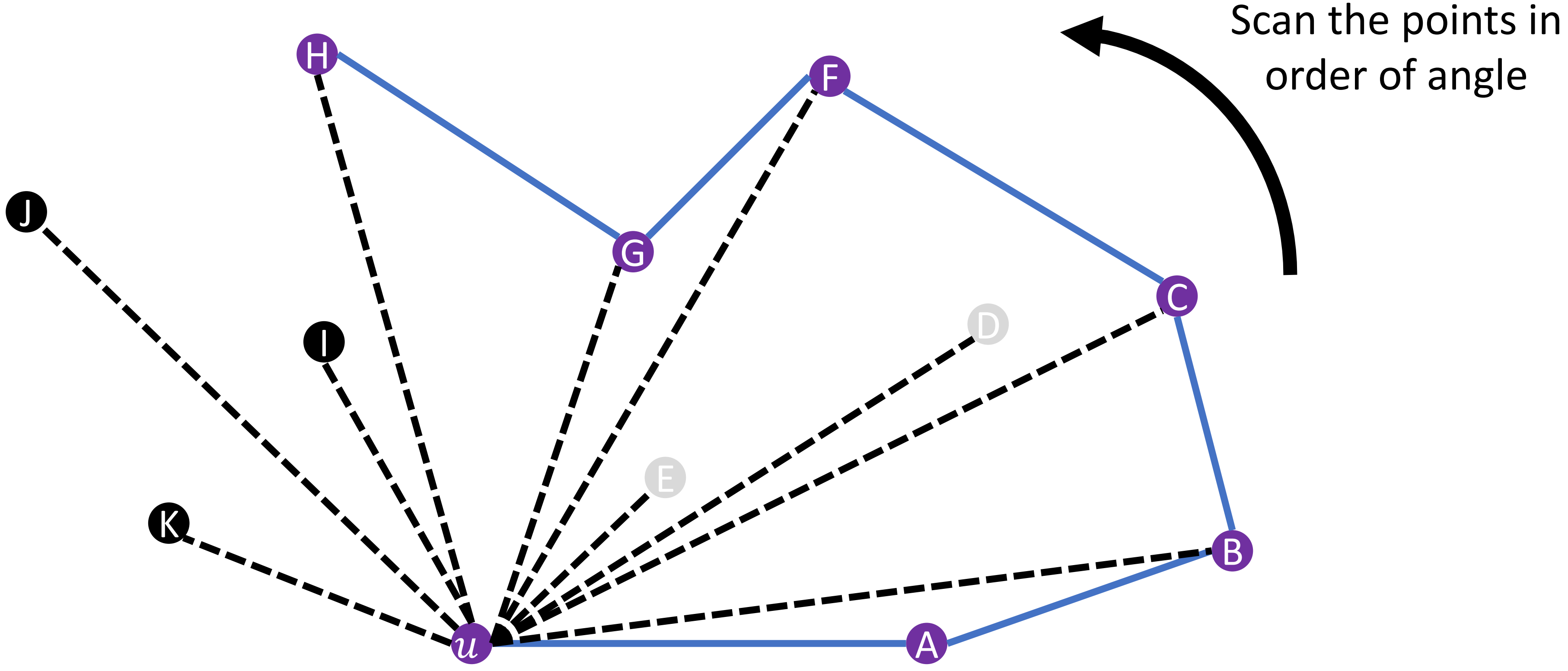


Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm

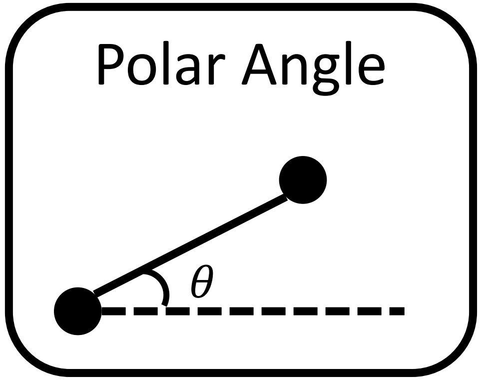


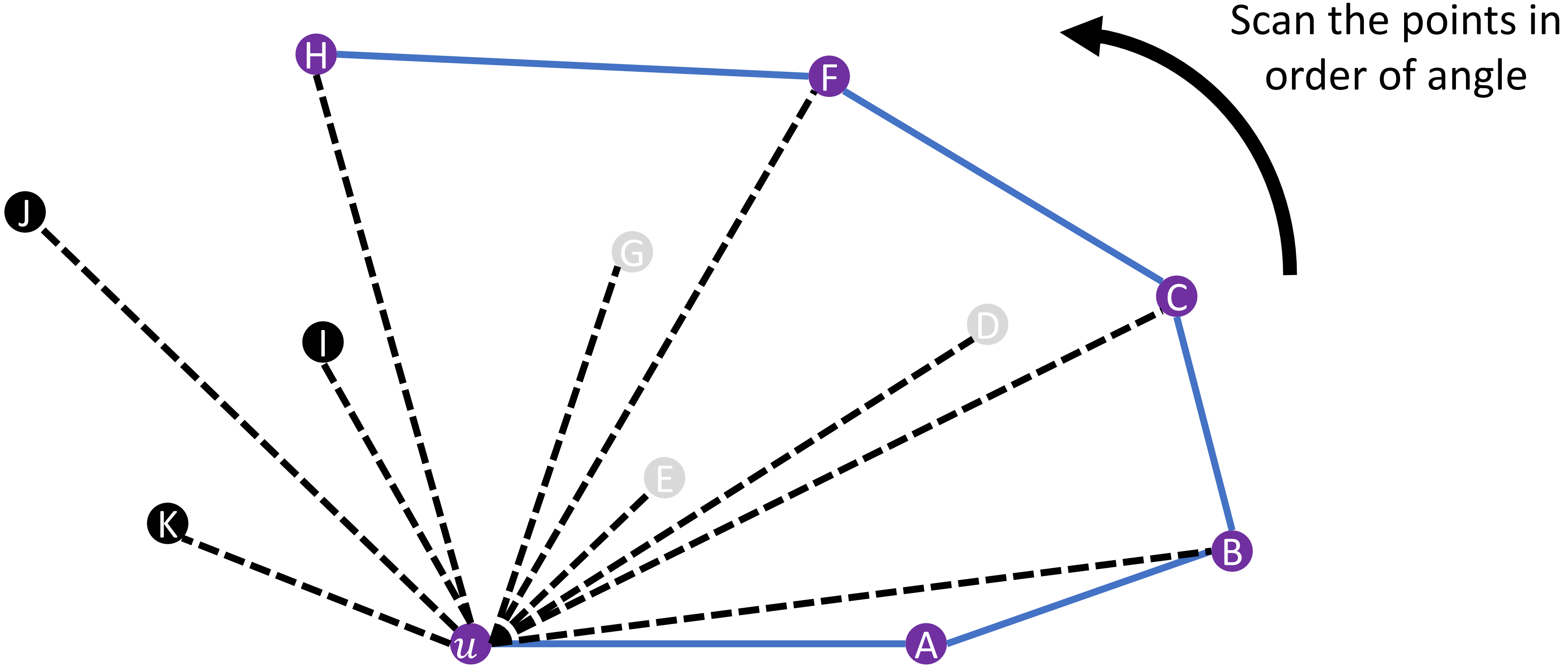
Not convex anymore!



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

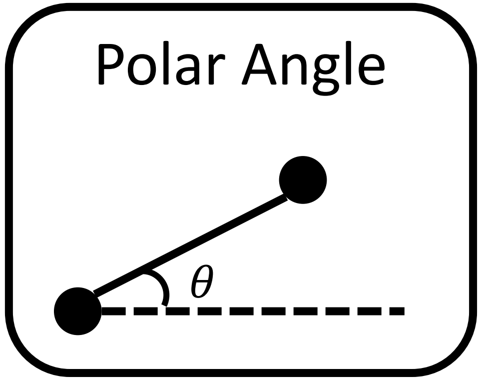
Graham’s Algorithm

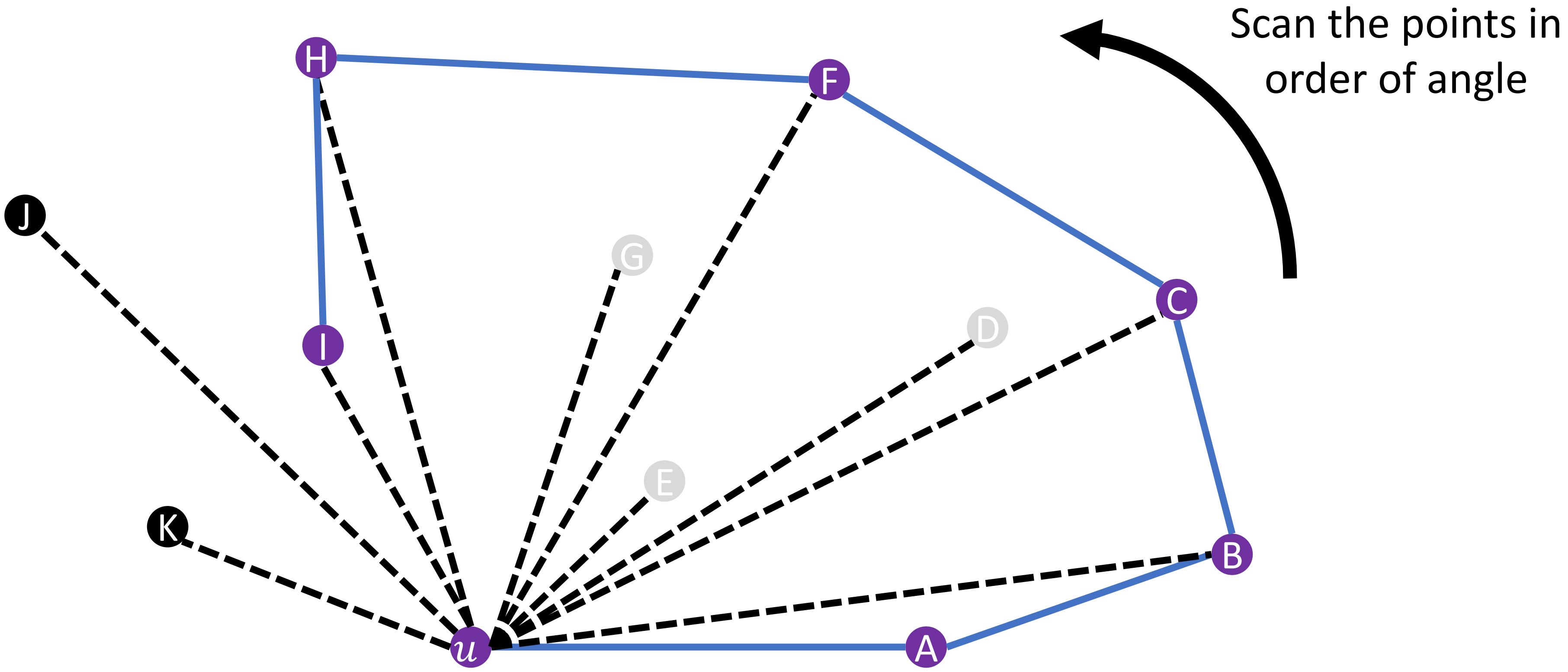




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm

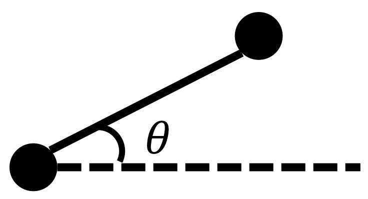




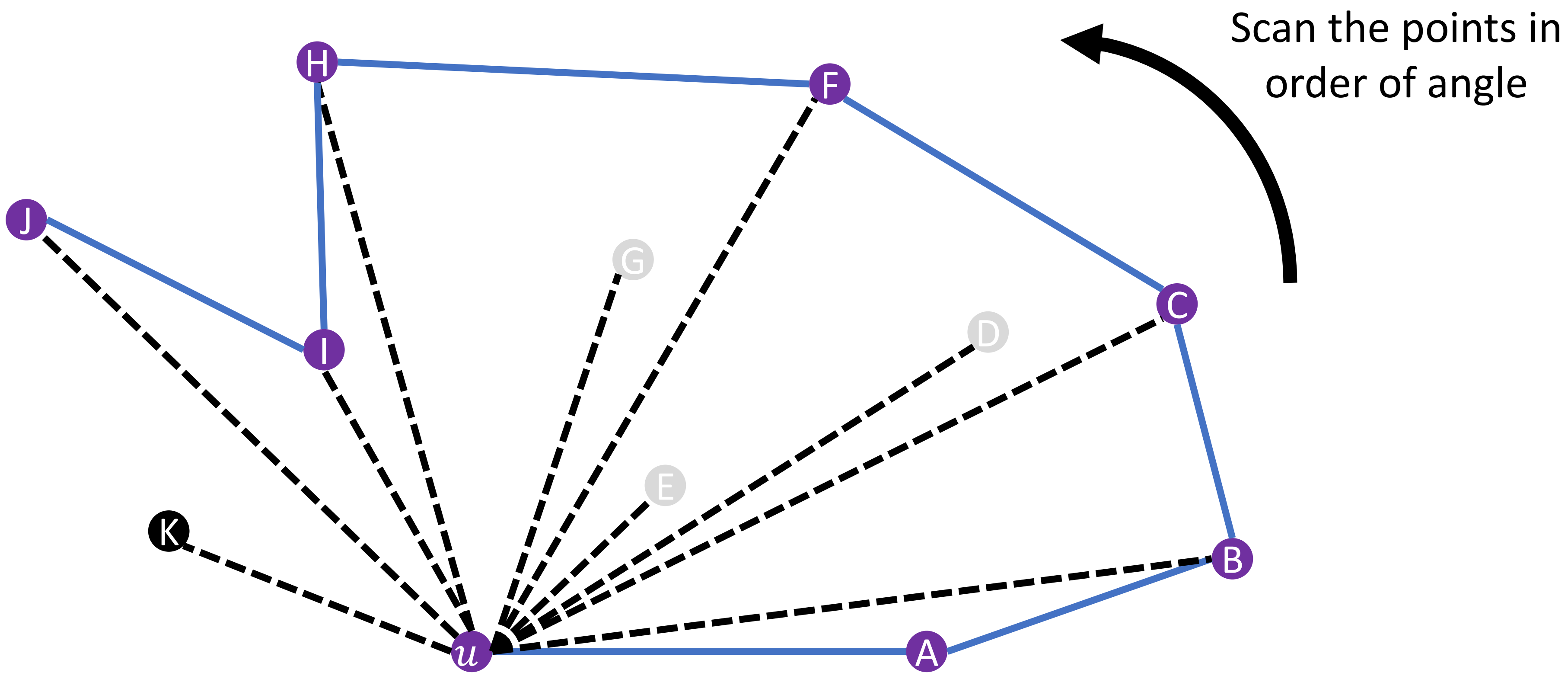
Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm

Polar Angle

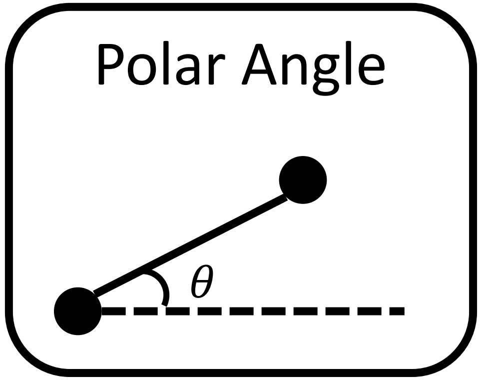


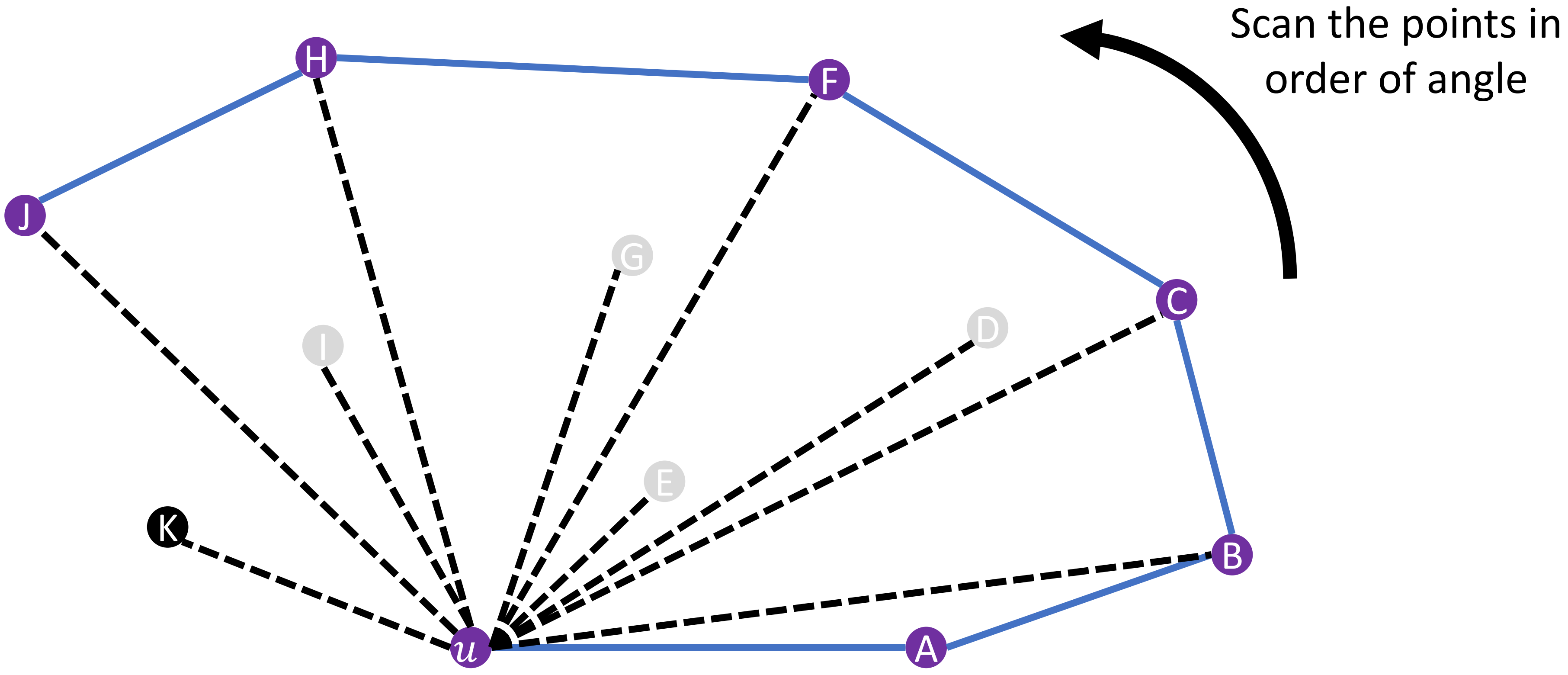
Not convex anymore!



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

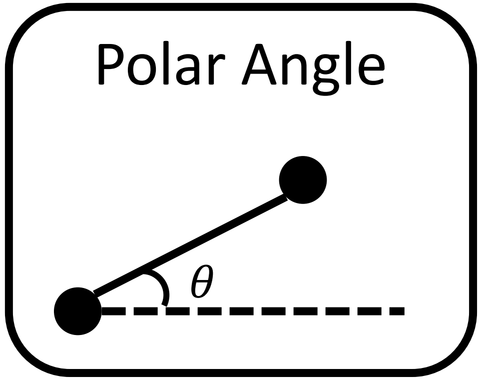
Graham’s Algorithm

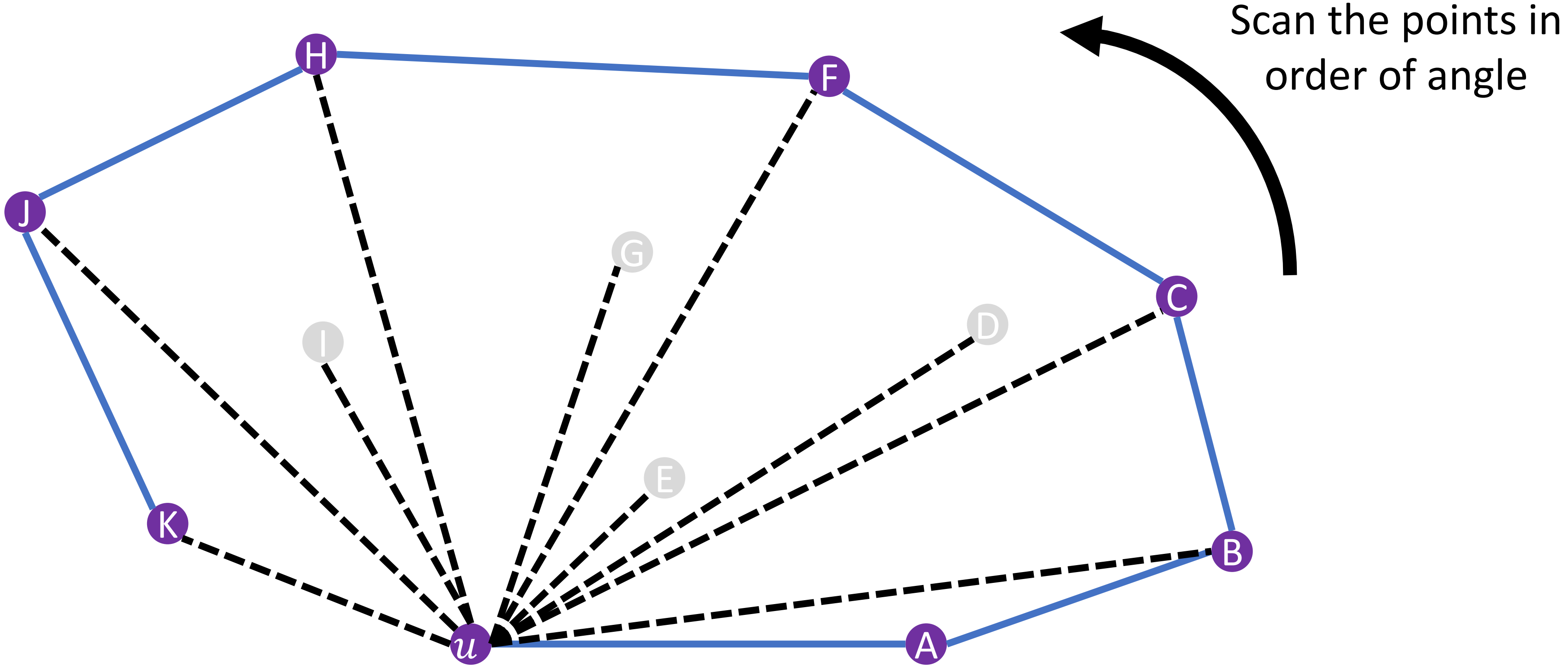




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

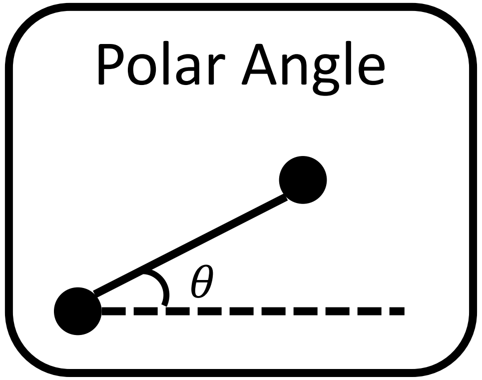
Graham's Algorithm

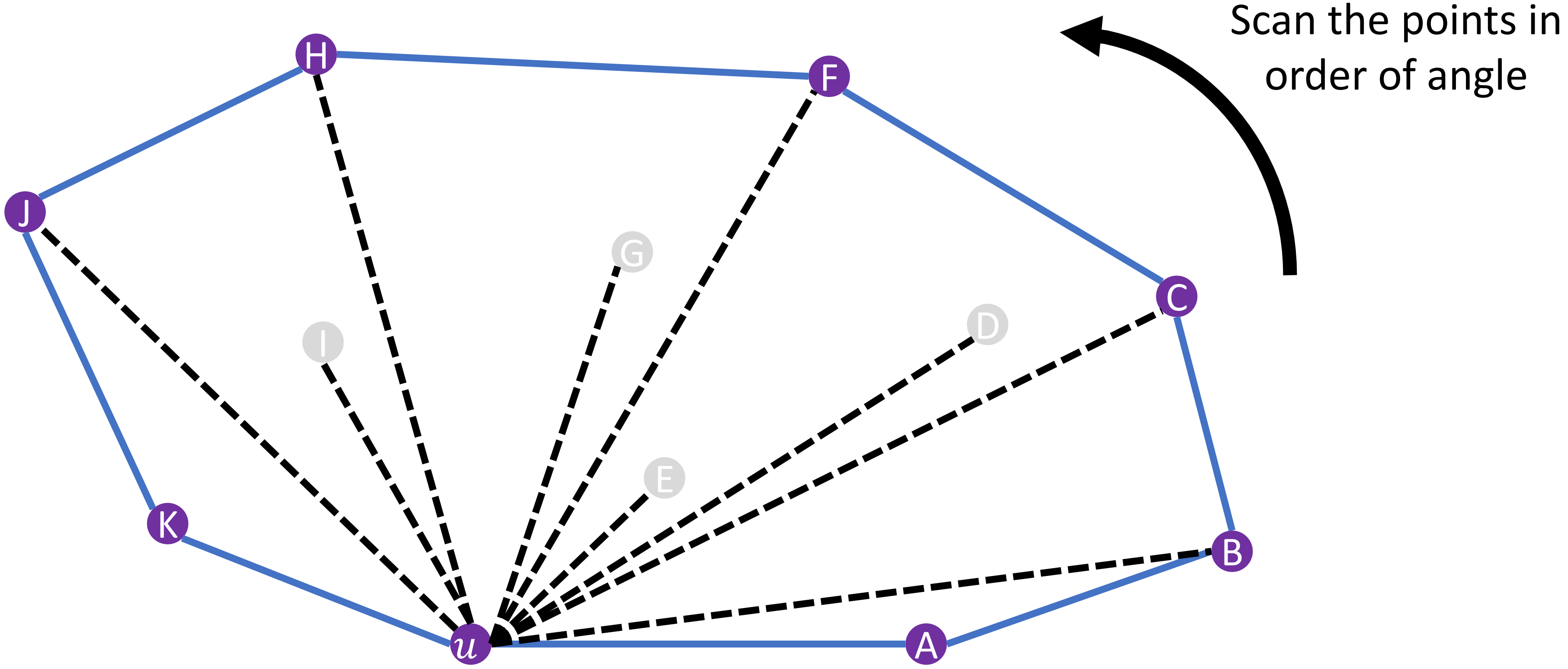




Idea: In order of angle, add points to the convex hull as long as it preserves convexity

Graham's Algorithm



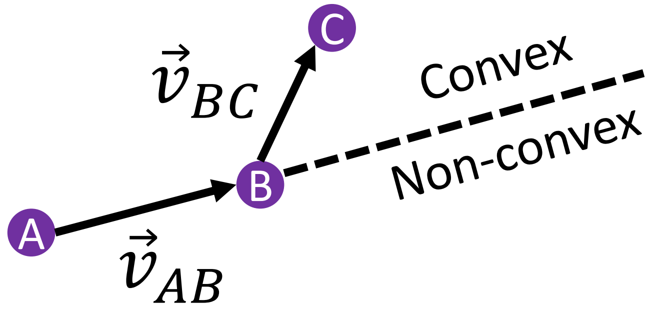


Idea: In order of angle, add points to the convex hull as long as it preserves convexity

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

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 <u>sign</u> 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

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

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 $O(1)$
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

Running time: $O(n \log n)$

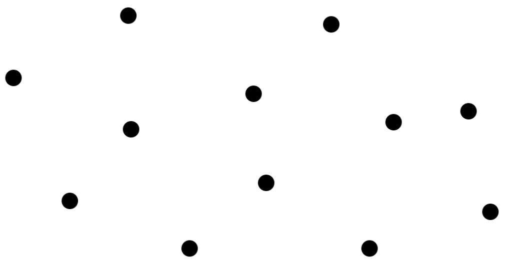
Graham's Algorithm

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 $O(1)$
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check

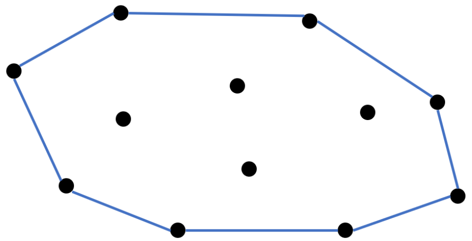
We have essentially reduced the problem of computing a convex hull to the problem of sorting!

Convex Hull to Sorting Reduction

convex hull



convex hull



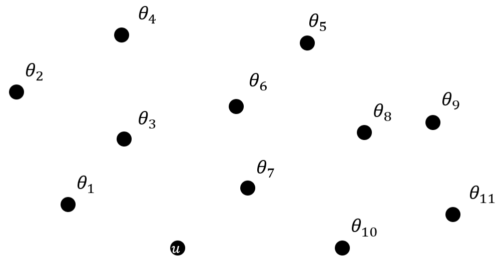
Map instances of problem
$\boldsymbol{A}$ to instances of $\boldsymbol{B}$

$O(n)$

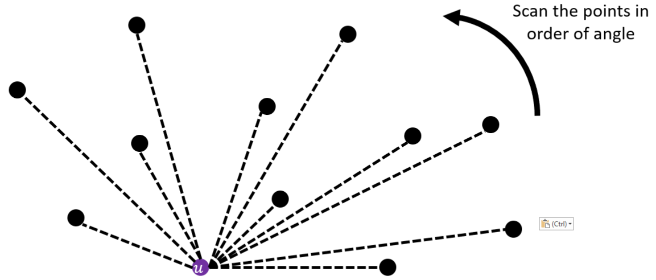
Map solutions of problem
$\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

$O(n)$

sorting



points sorted by angle



convex hull $\leq$ sorting

convex hull can be reduced to sorting in $O(n)$ time

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
   - If adding p_i makes C non-convex, then remove the last component of C and repeat this check — $O(1)$

$O(n \log n)$

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

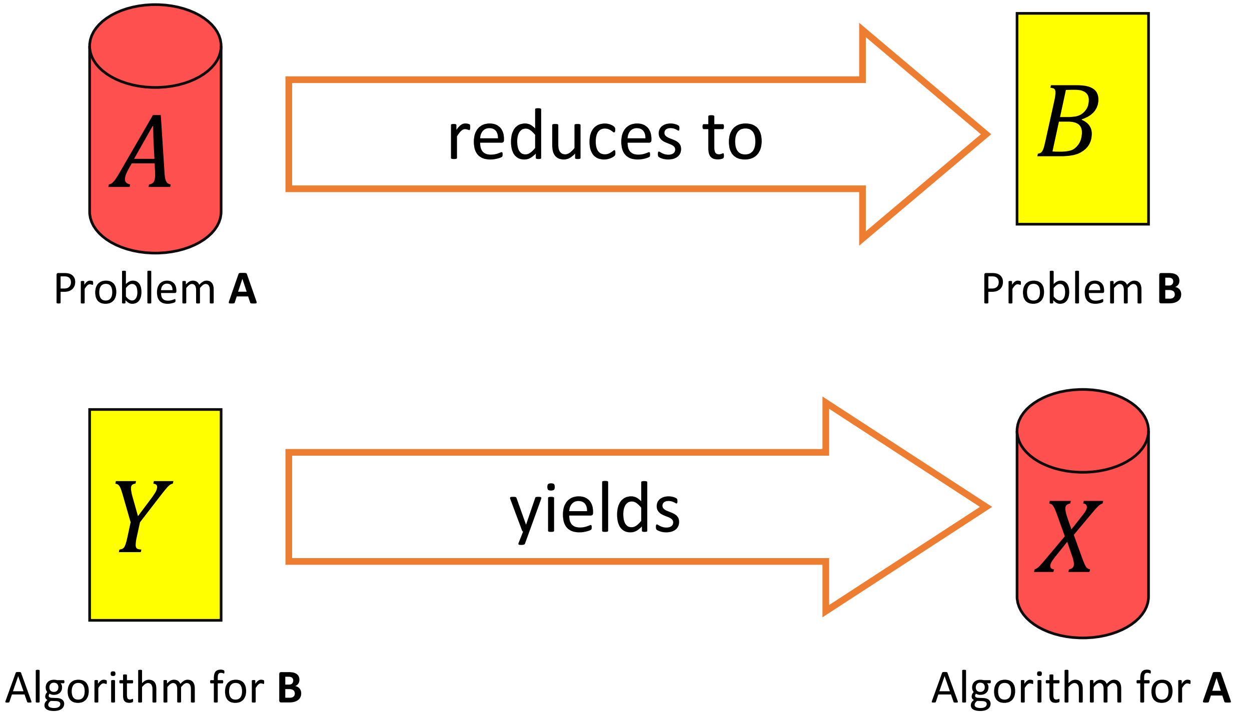
1. Let p_1 be the point with the smallest y-coordinate (and smallest x-coordinate if multiple points have the same minimum-y coordinate) $O(n)$
2. Add p_1 to the convex hull C (represented as a stack) $O(1)$
3. Sort all of the points based on their angle relative to p_1 $O(n \log n)$
4. For each of the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

Trivial lower bound: $\Omega(n)$

…me as best sorting algorithm

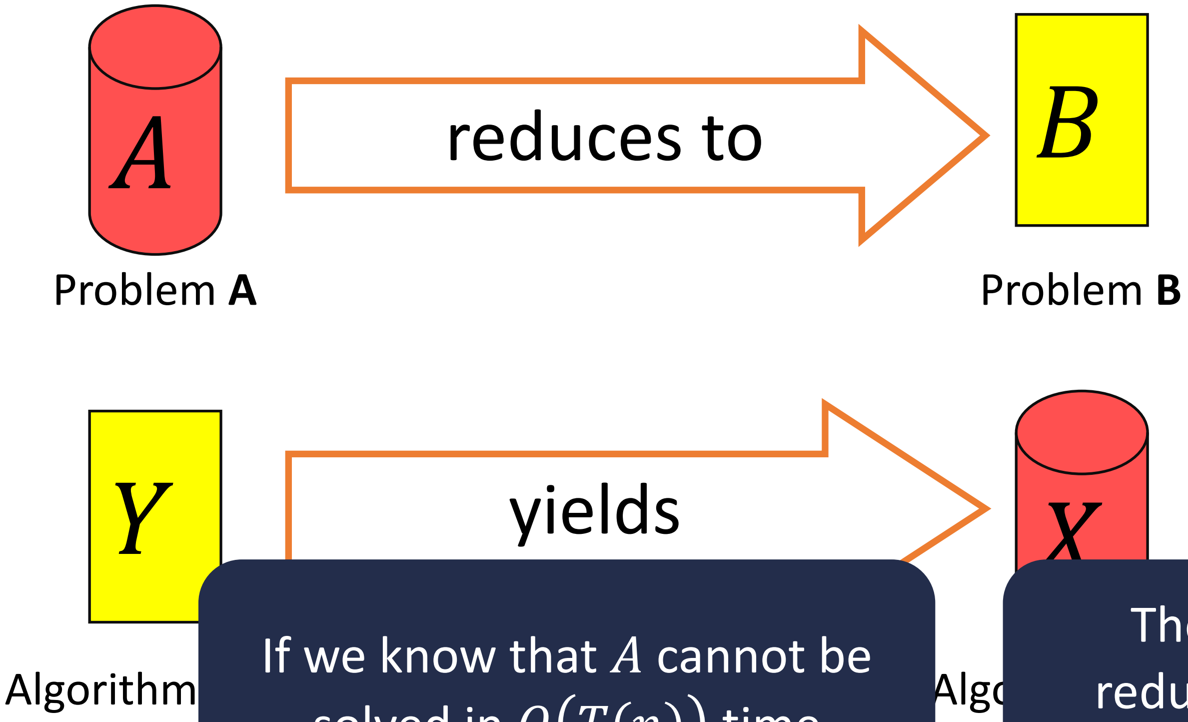
Can we do better (without going through sorting)?

Worst-Case Lower Bounds via Reductions



Implication: A is no more difficult than B
(denoted $A \leq B$)

Worst-Case Lower Bounds via Reductions

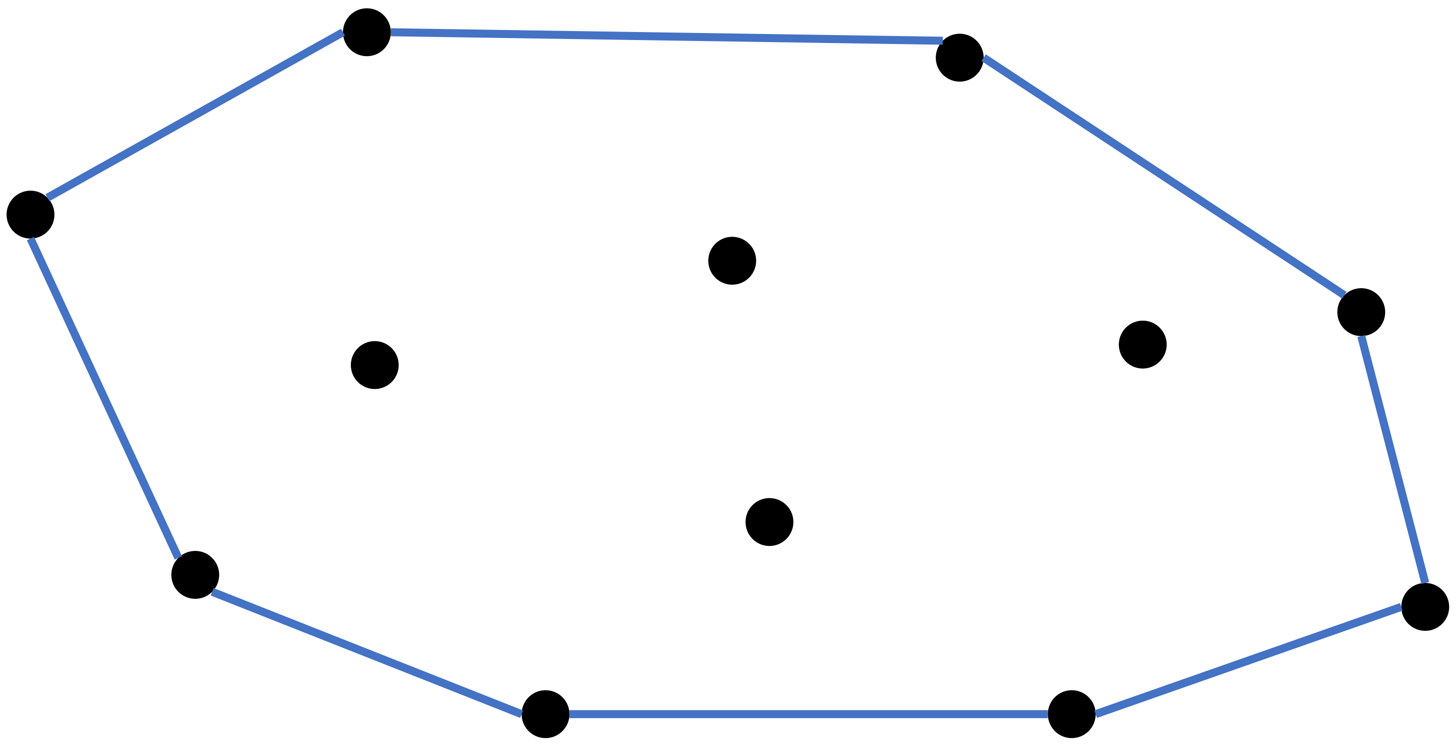


If we know that A cannot be solved in $O(T(n))$ time

Then, if there is a $O(T(n))$ reduction from A to B, then B <u>cannot</u> be solved in $O(T(n))$ time

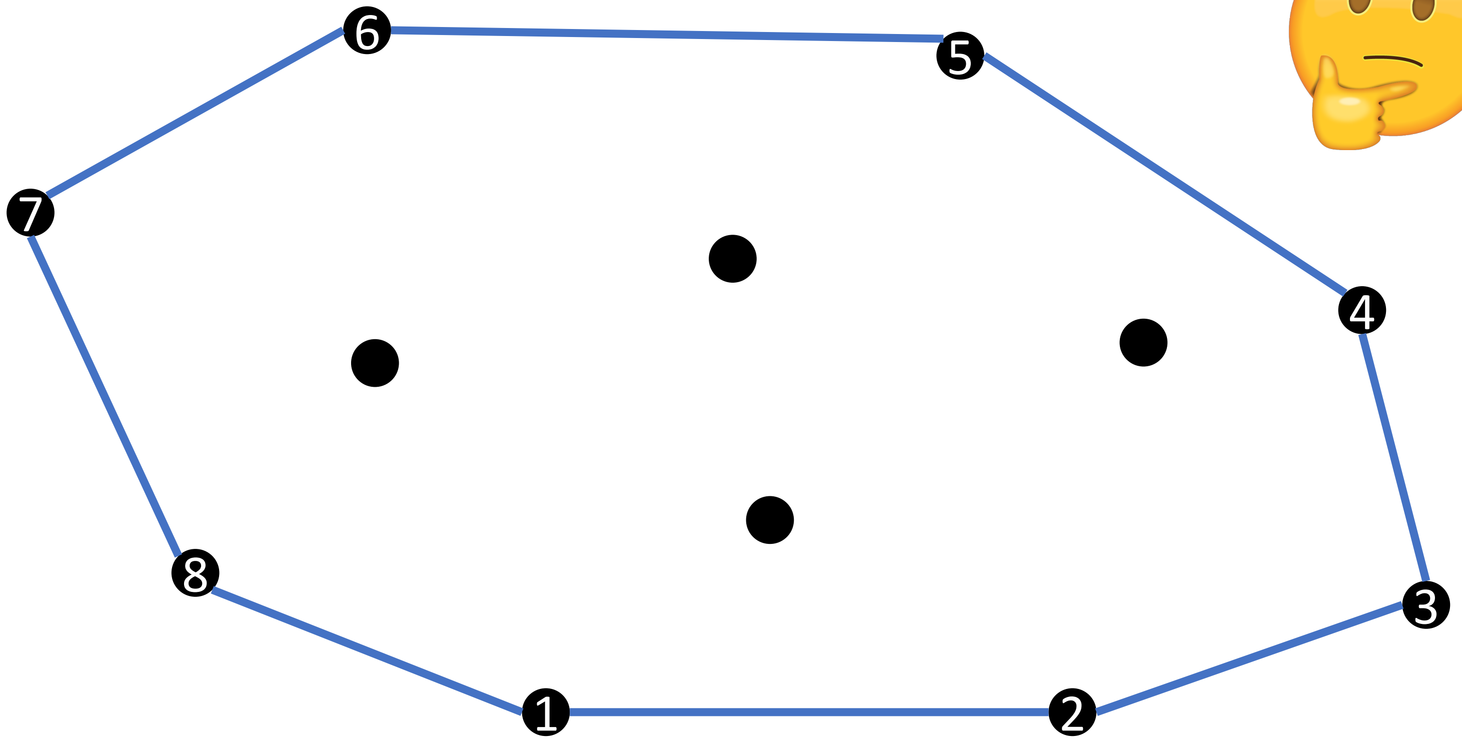
Implication: A is <u>no more difficult</u> than B
(denoted $A \leq B$)

Sorting to Convex Hull Reduction



Observe: convex hull consists of a subset of points in a prescribed order

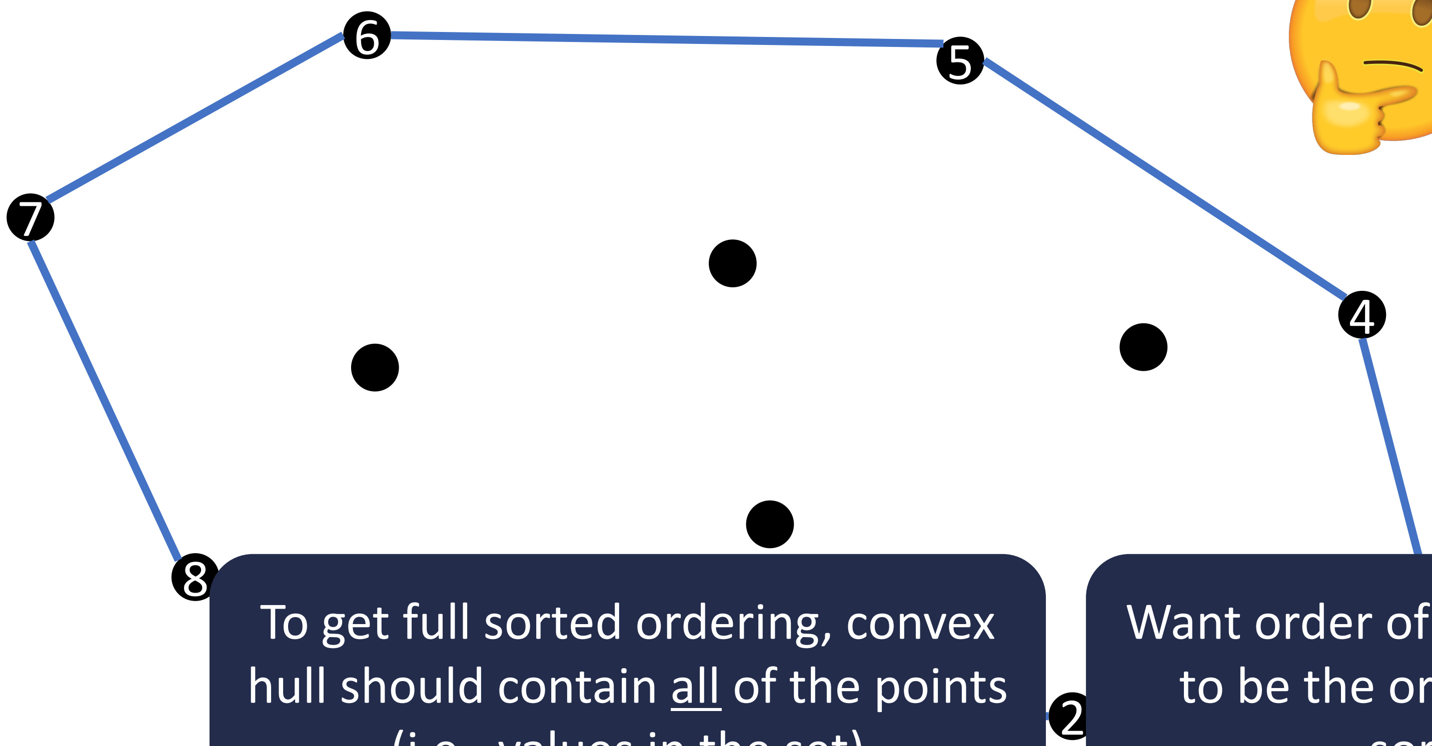
Sorting to Convex Hull Reduction



Can we use this to sort?

Observe: convex hull consists of a subset of points in a prescribed order

Sorting to Convex Hull Reduction



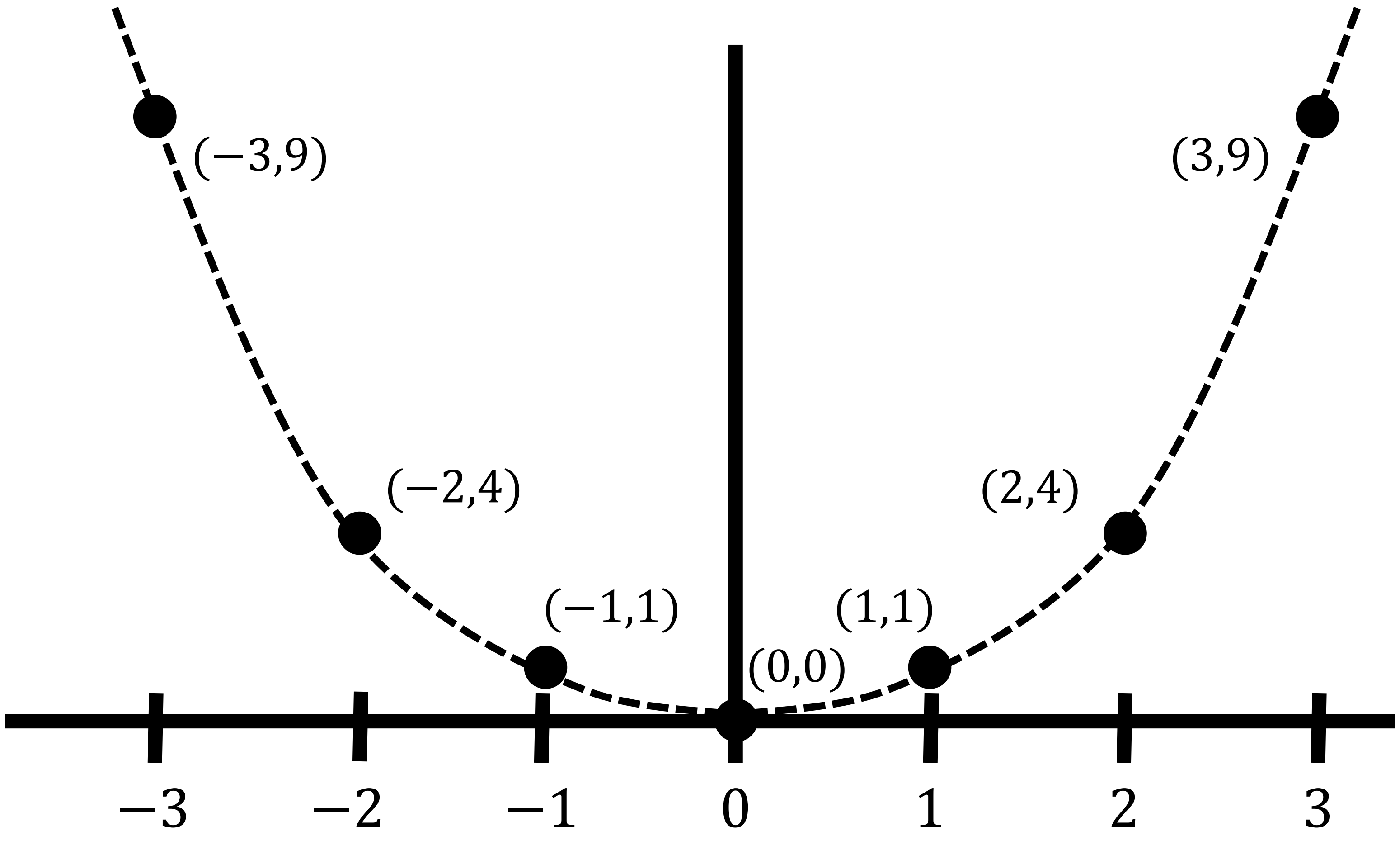
Can we use this to sort?

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

Sorting to Convex Hull Reduction



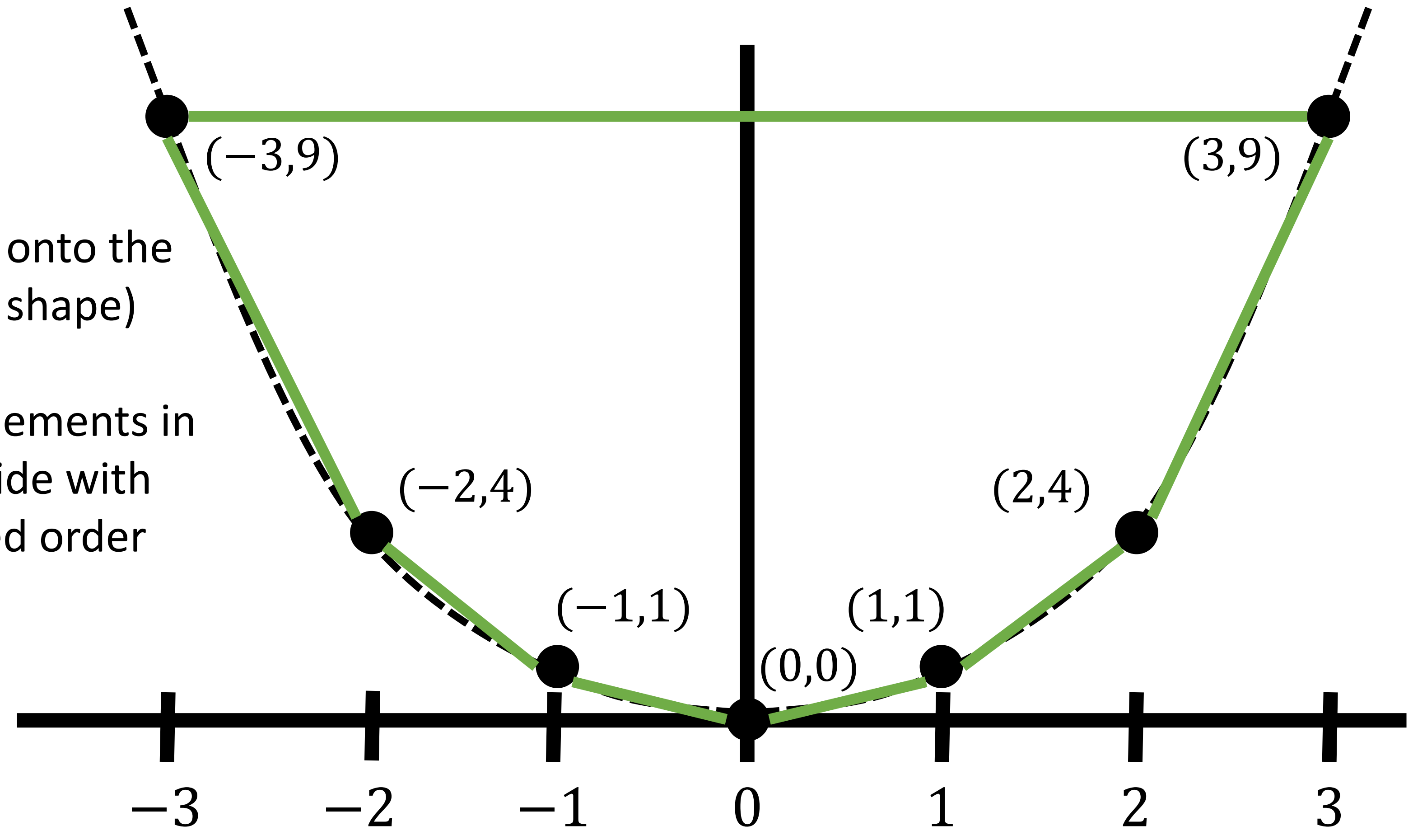
$x \mapsto (x, x^2)$

Goal: need a way to map list of (numeric) values onto a convex hull instance

Sorting to Convex Hull Reduction

Idea: Map points onto the <u>parabola</u> (convex shape)

Claim: order of elements in convex hull coincide with elements in sorted order



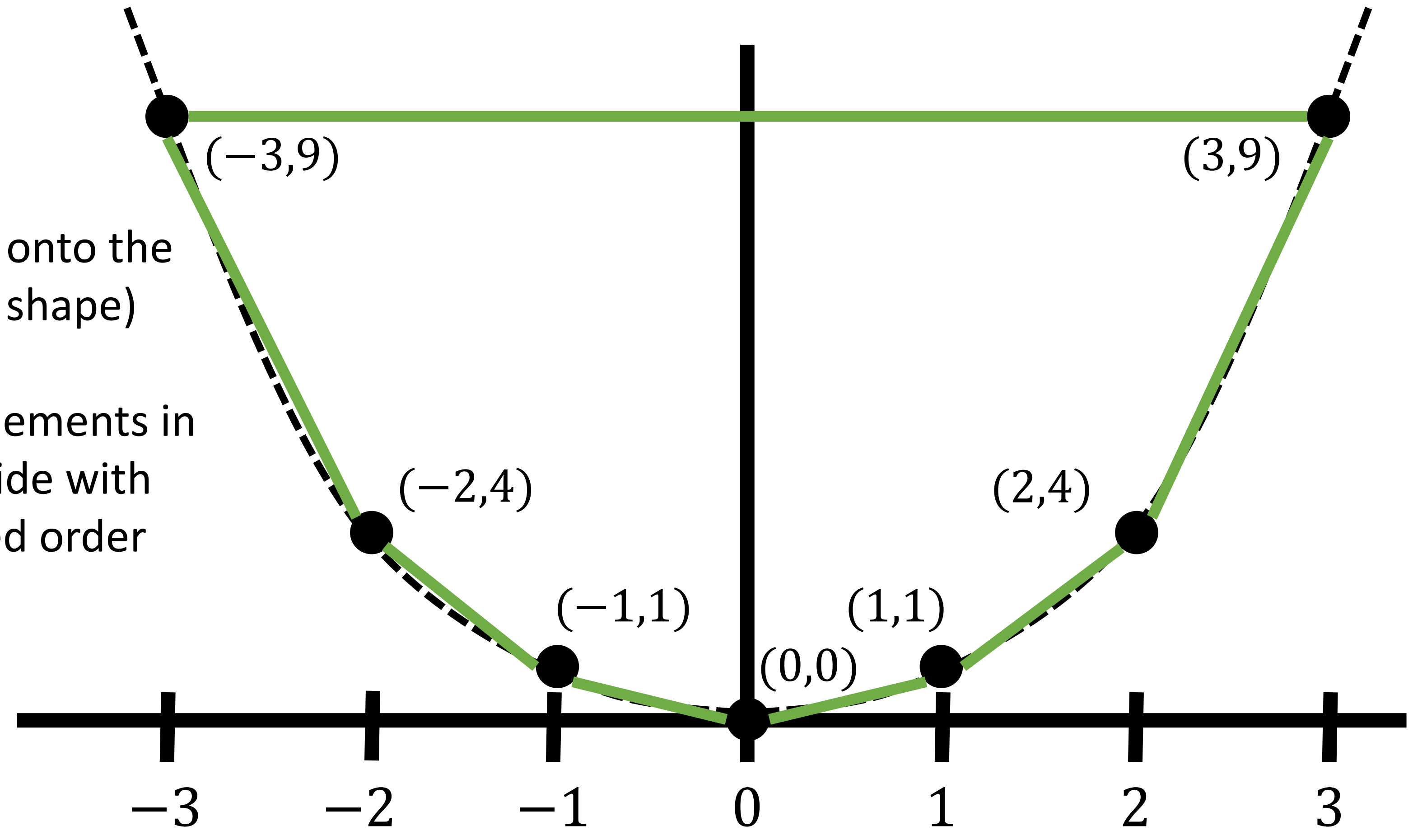
$x \mapsto (x, x^2)$

Goal: need a way to map list of (numeric) values onto a convex hull instance

Sorting to Convex Hull Reduction

Idea: Map points onto the <u>parabola</u> (convex shape)

Claim: order of elements in convex hull coincide with elements in sorted order



Conclusion: If we can solve convex hull, then we can sort numeric values

Convex Hull to Sorting Reduction

sorting

$-2 \quad 1 \quad -3 \quad 0 \quad 2 \quad 3 \quad -1$

Map instances of problem $\boldsymbol{A}$ to instances of $\boldsymbol{B}$

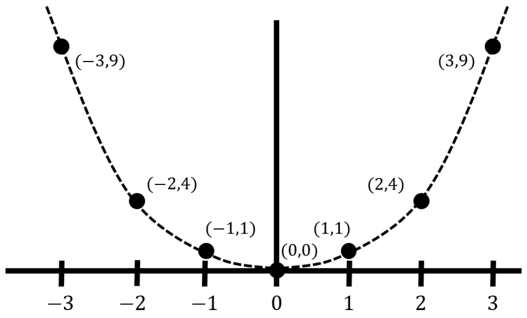
$O(n)$

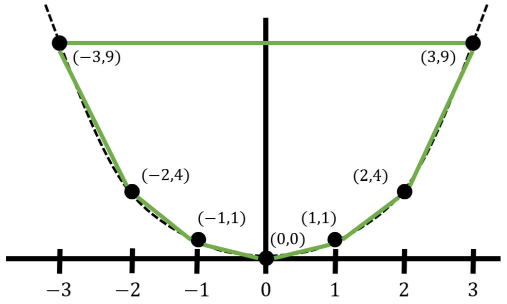
Map solutions of problem $\boldsymbol{B}$ to solutions of $\boldsymbol{A}$

$O(n)$

$-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3$

convex hull

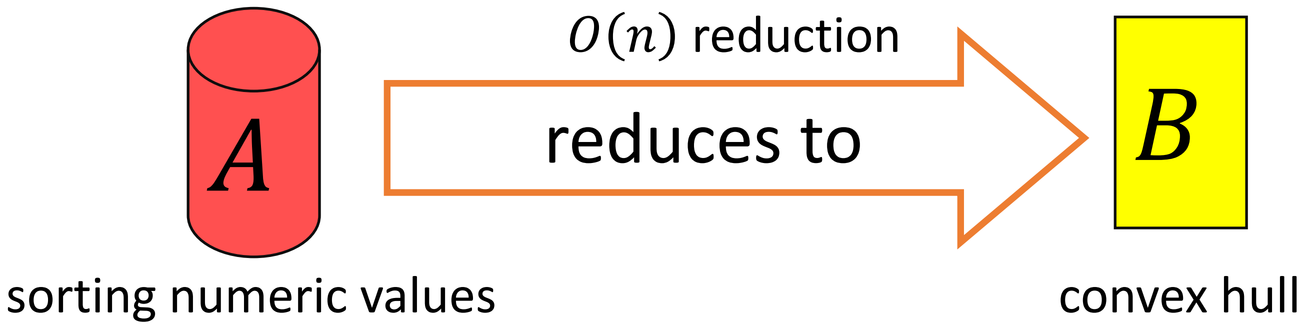




sorting numeric values $\leq$ convex hull

sorting numeric values can be reduced to convex hull in $O(n)$ time

Lower Bound for Convex Hull



Conclusion: a lower bound for sorting translates into one for convex hull

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

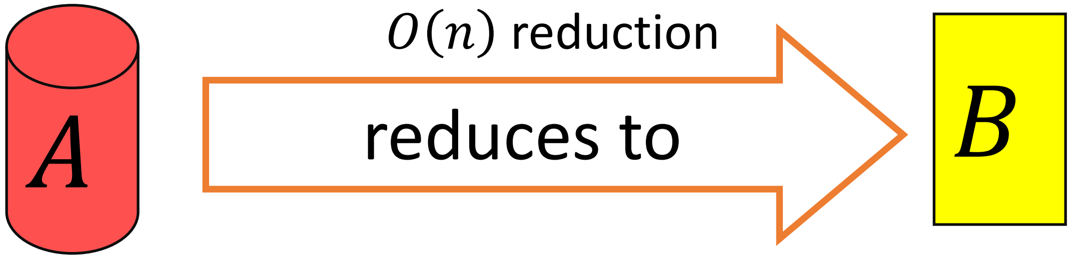
Our reduction is not a comparison sort algorithm, so cannot directly appeal to it

$\Omega(n \log n)$ 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

Lower Bound for Convex Hull



sorting numeric values

convex hull

Conclusion: a lower bound for sorting translates into one for convex hull

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

Our reduction is not a c

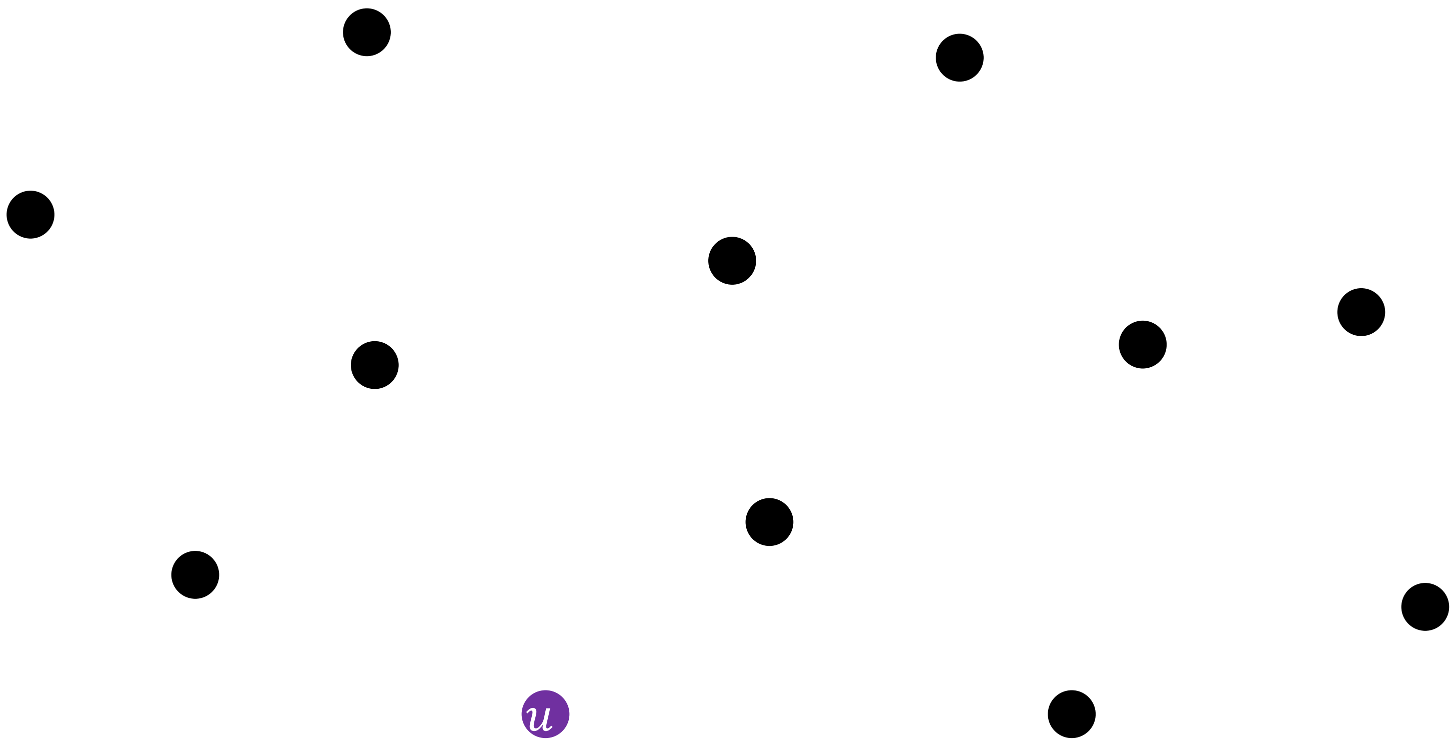
In fact, this lower bound holds even for algorithms that just identify the set of points on the convex hull (and not necessarily their order)!

$\Omega(n \log n)$ lower bound for s … e model"

(i.e., decisions can be

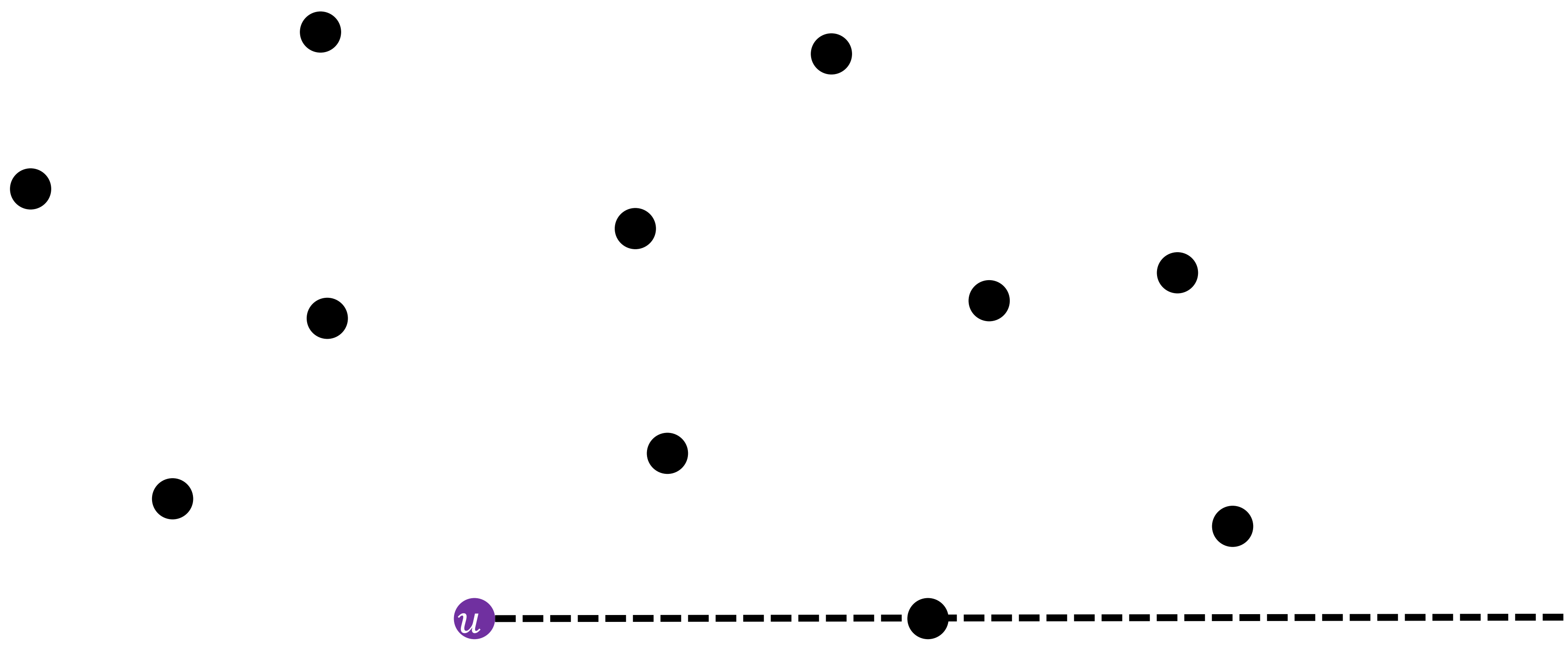
Implies $\Omega(n \log n)$ lower bound for computing convex hull in this model

Jarvis' Algorithm (Gift Wrapping Method)



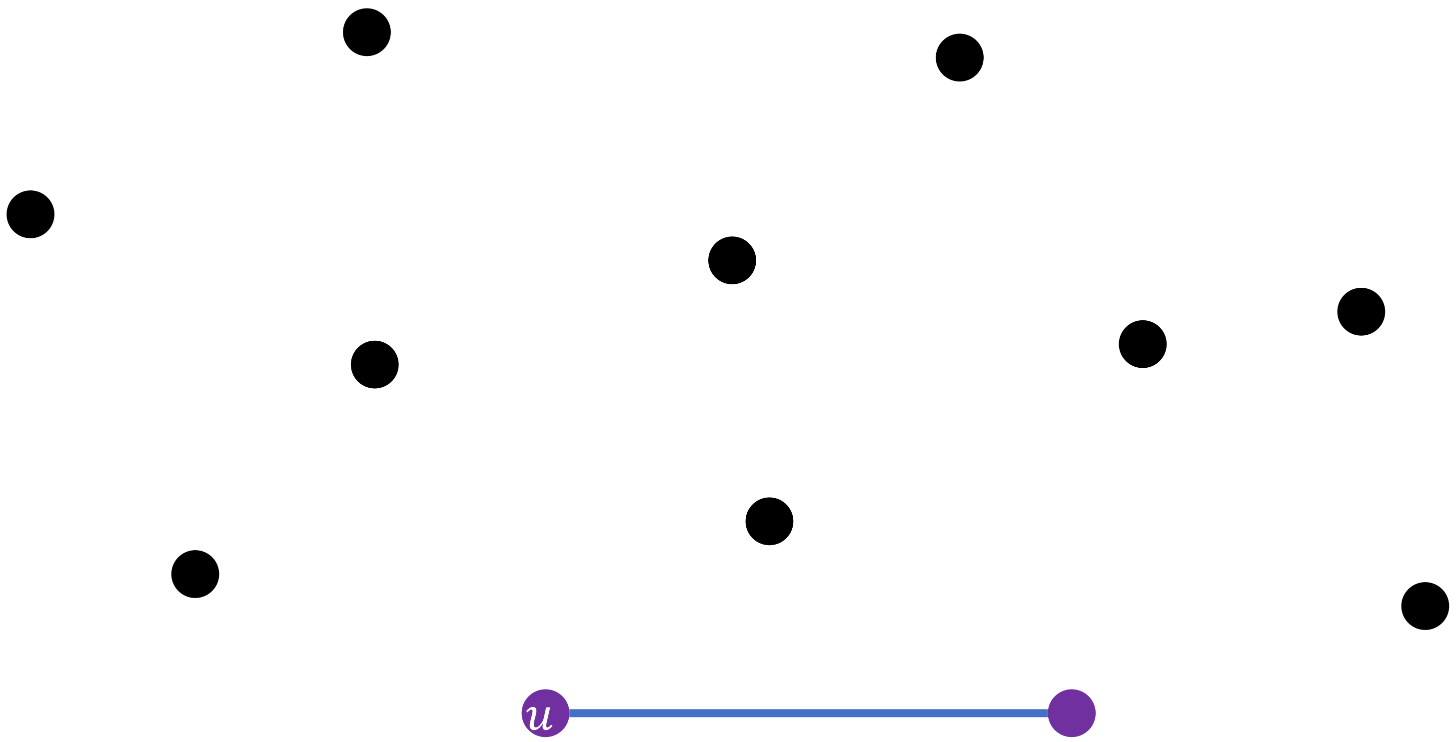
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



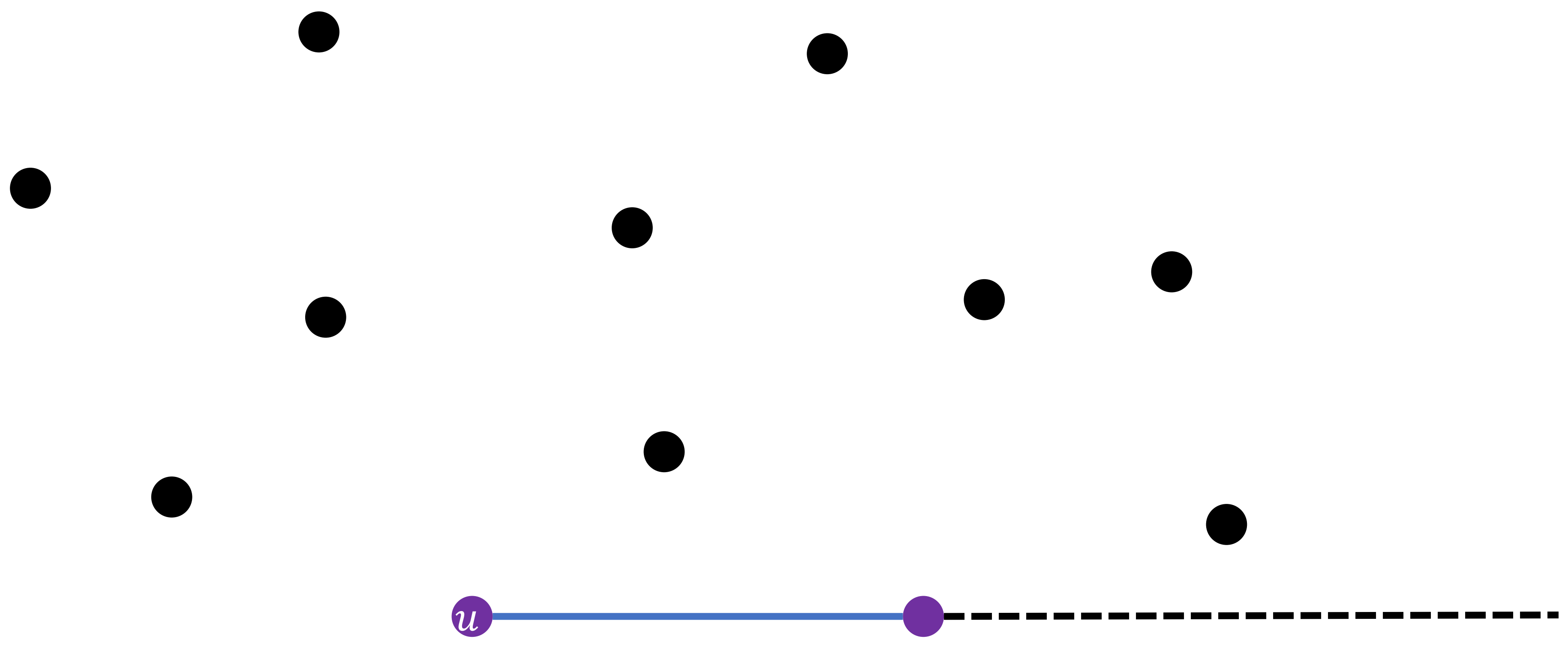
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis’ Algorithm (Gift Wrapping Method)



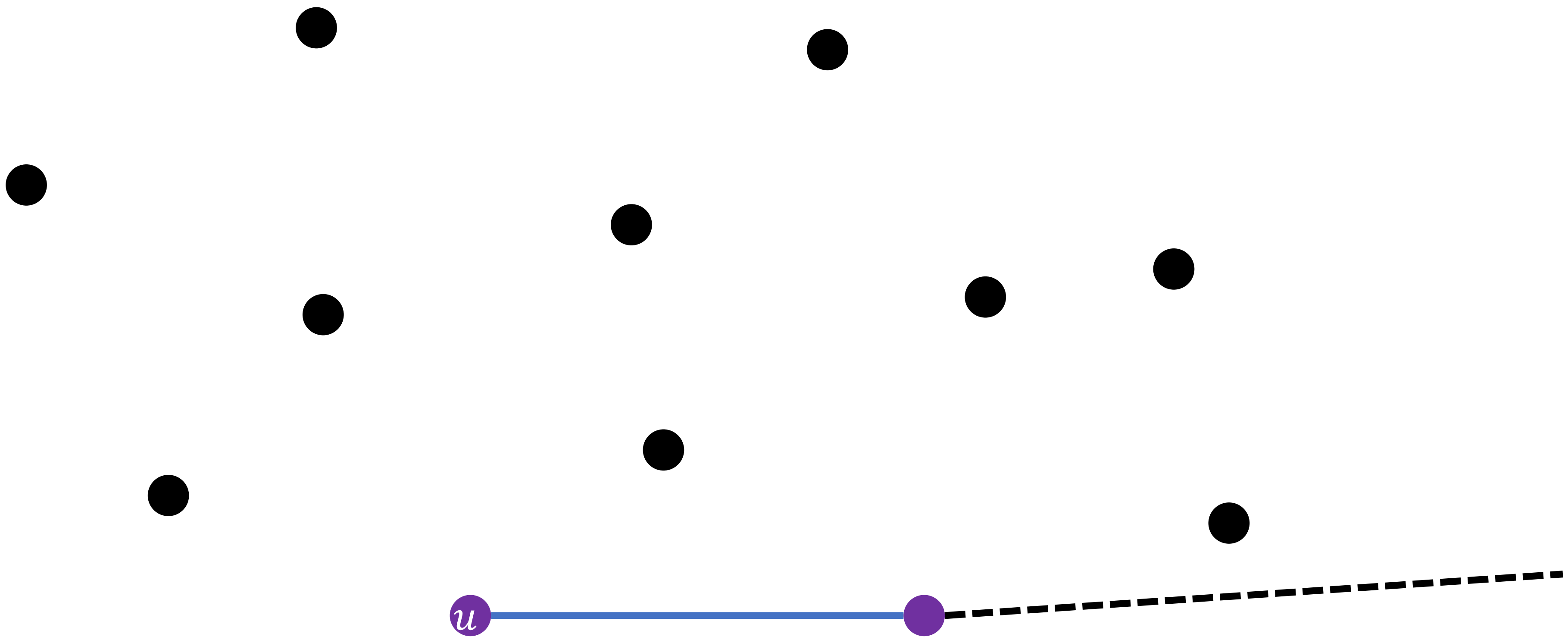
Idea: Start with extremal point and “wrap” points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



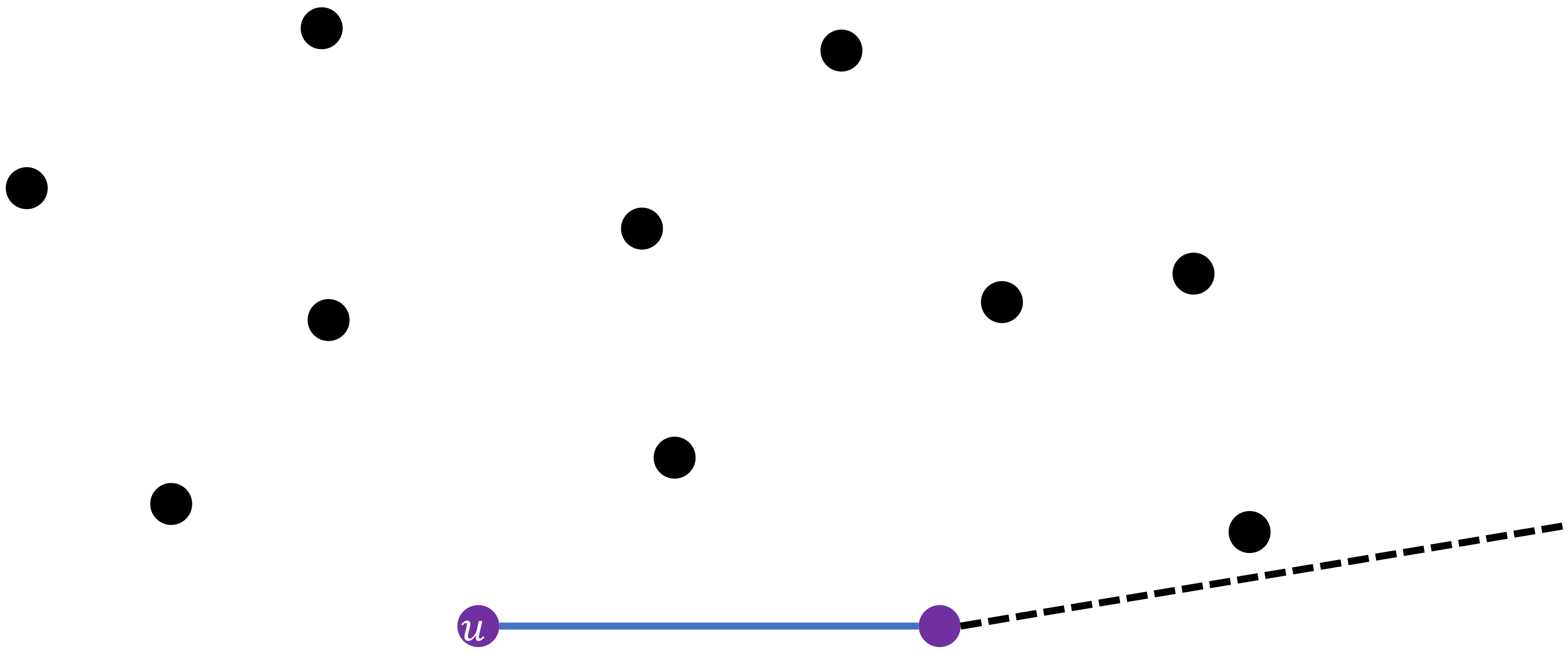
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



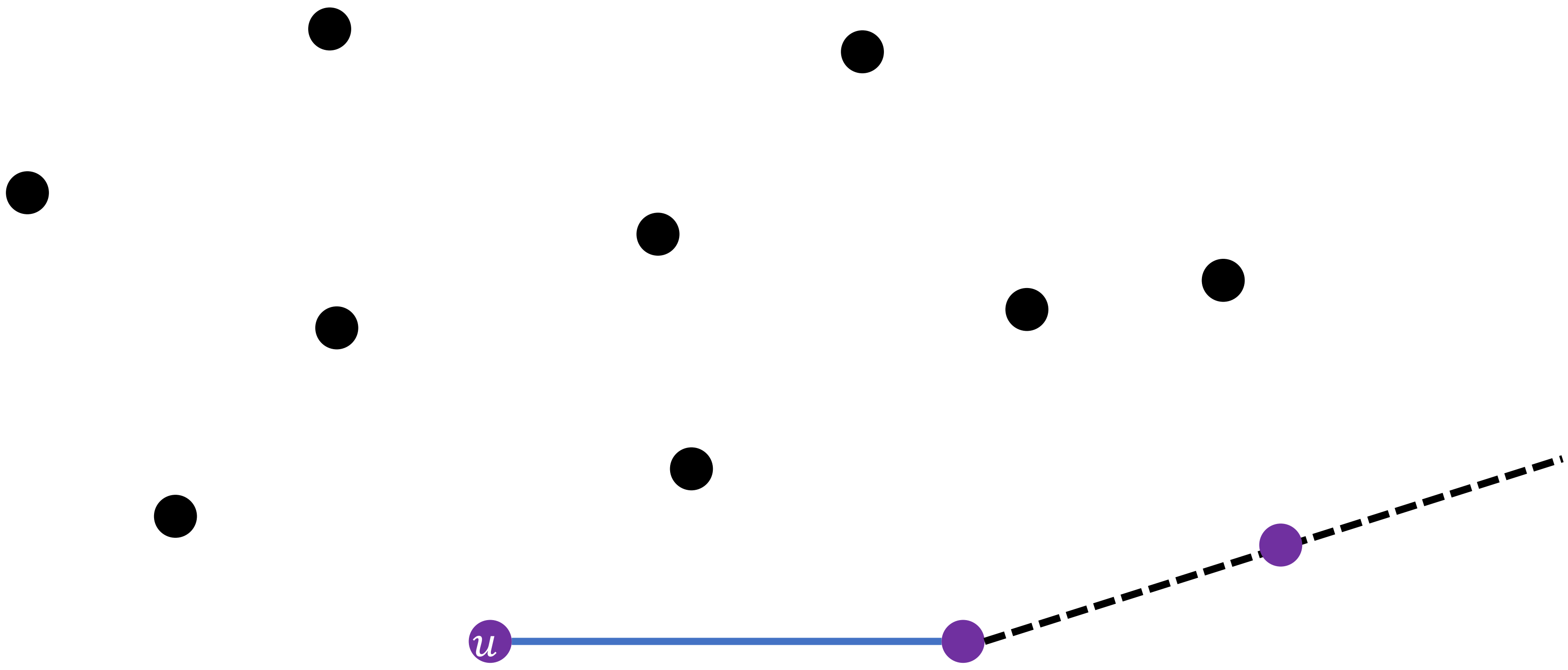
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



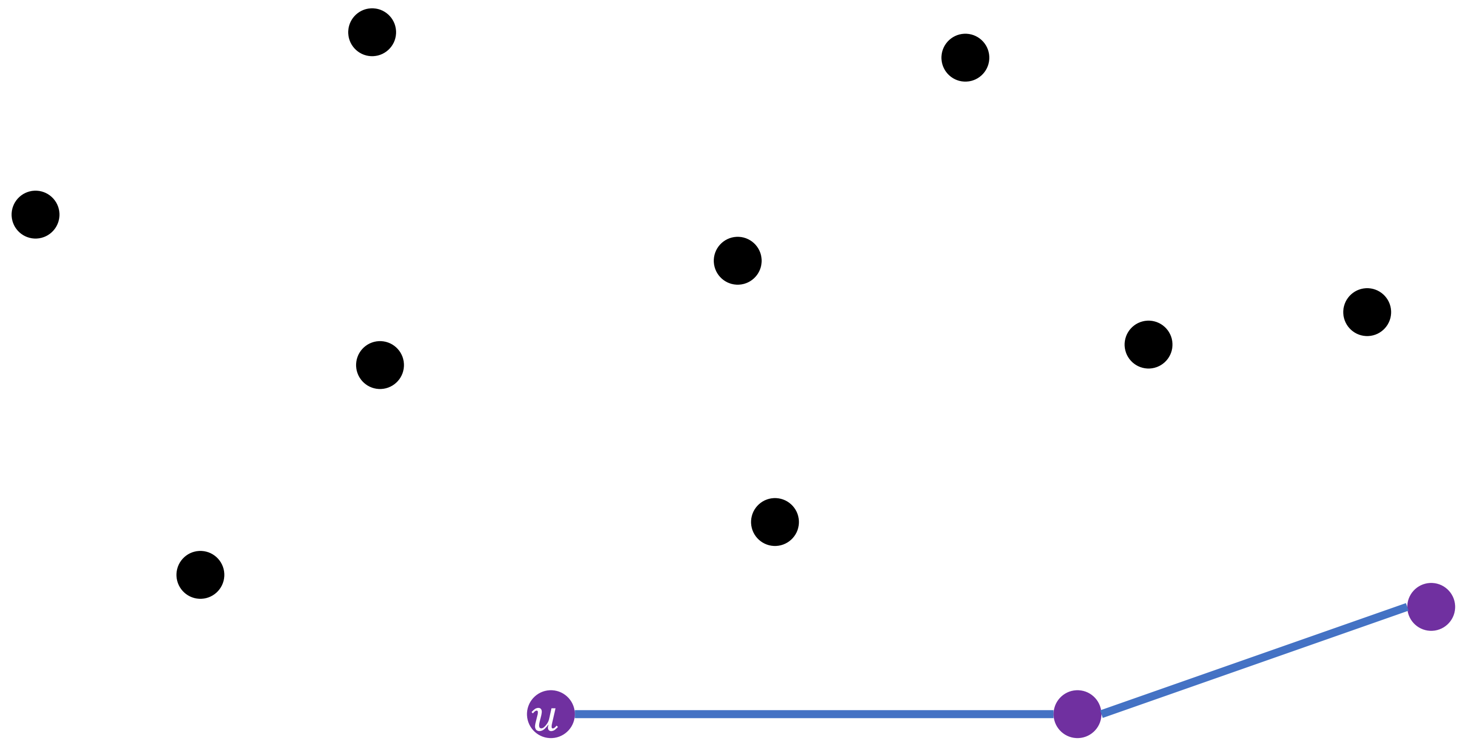
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



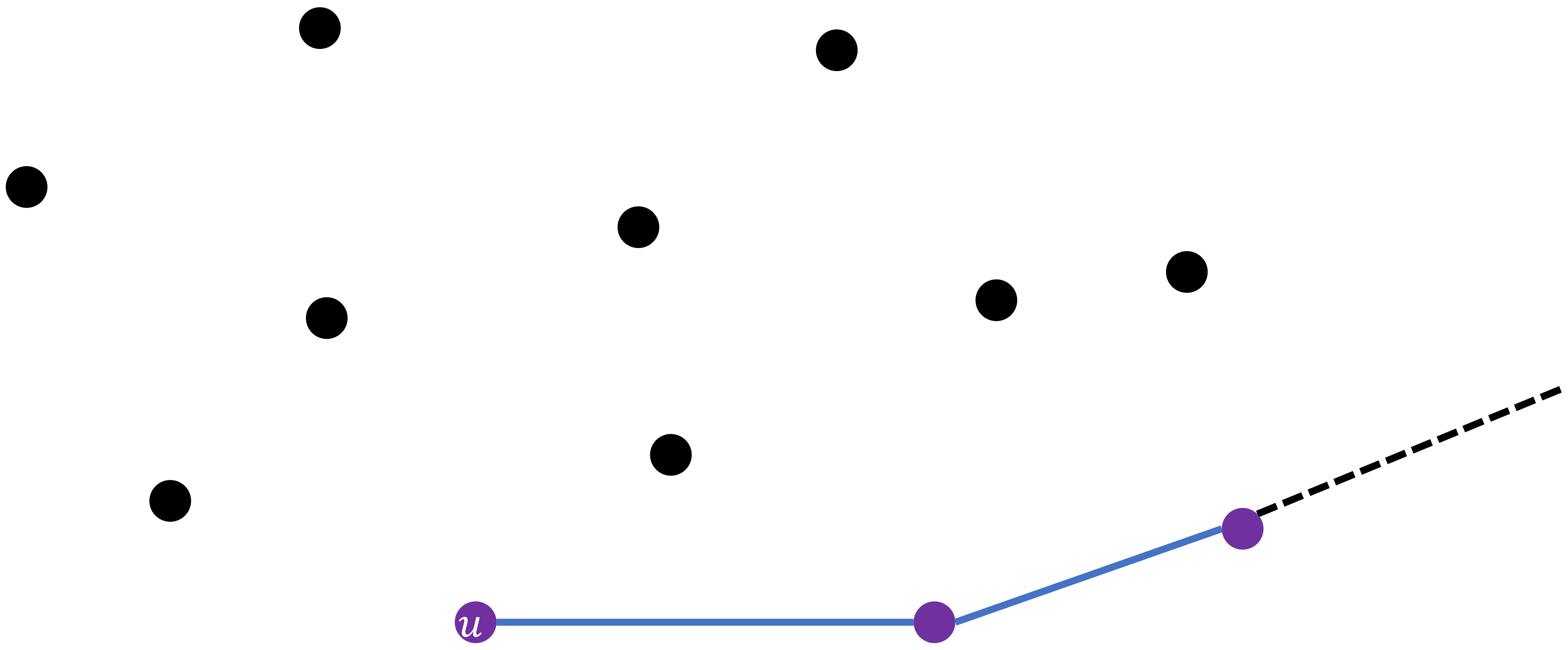
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



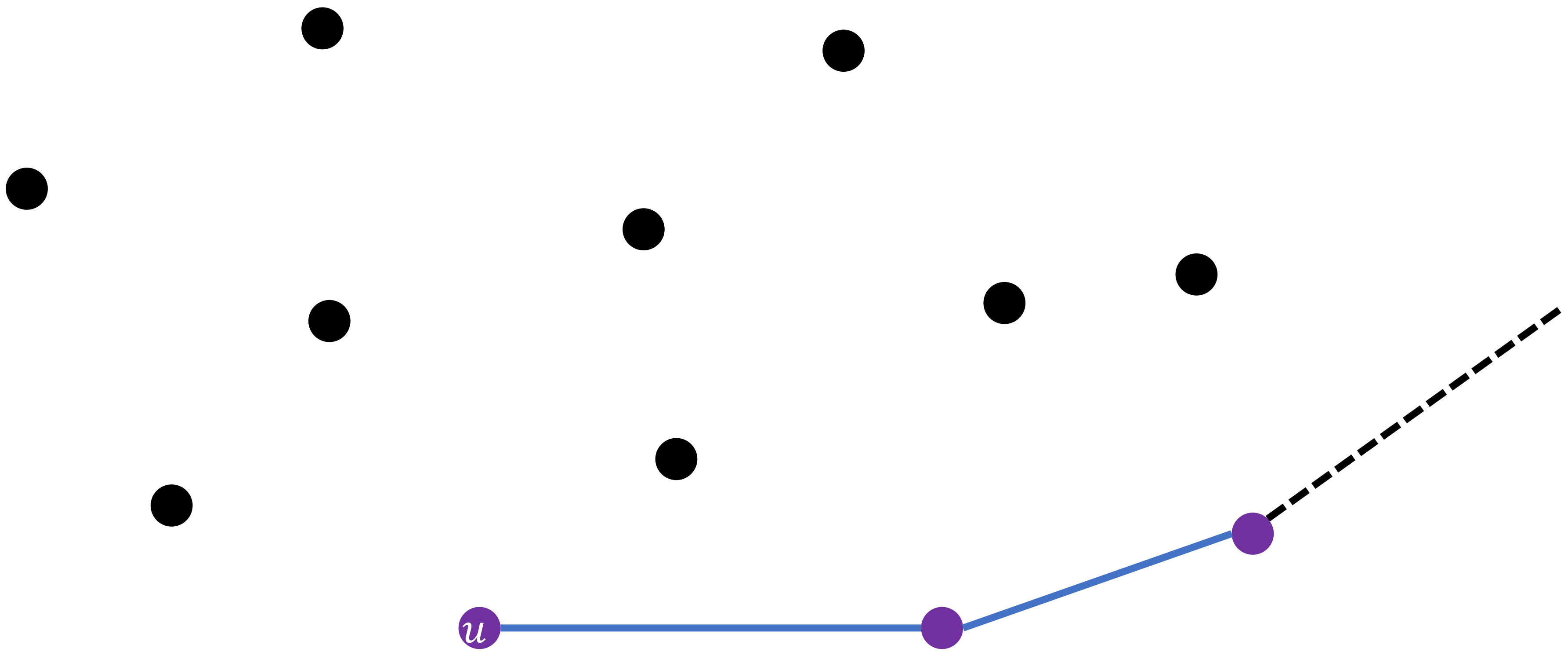
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



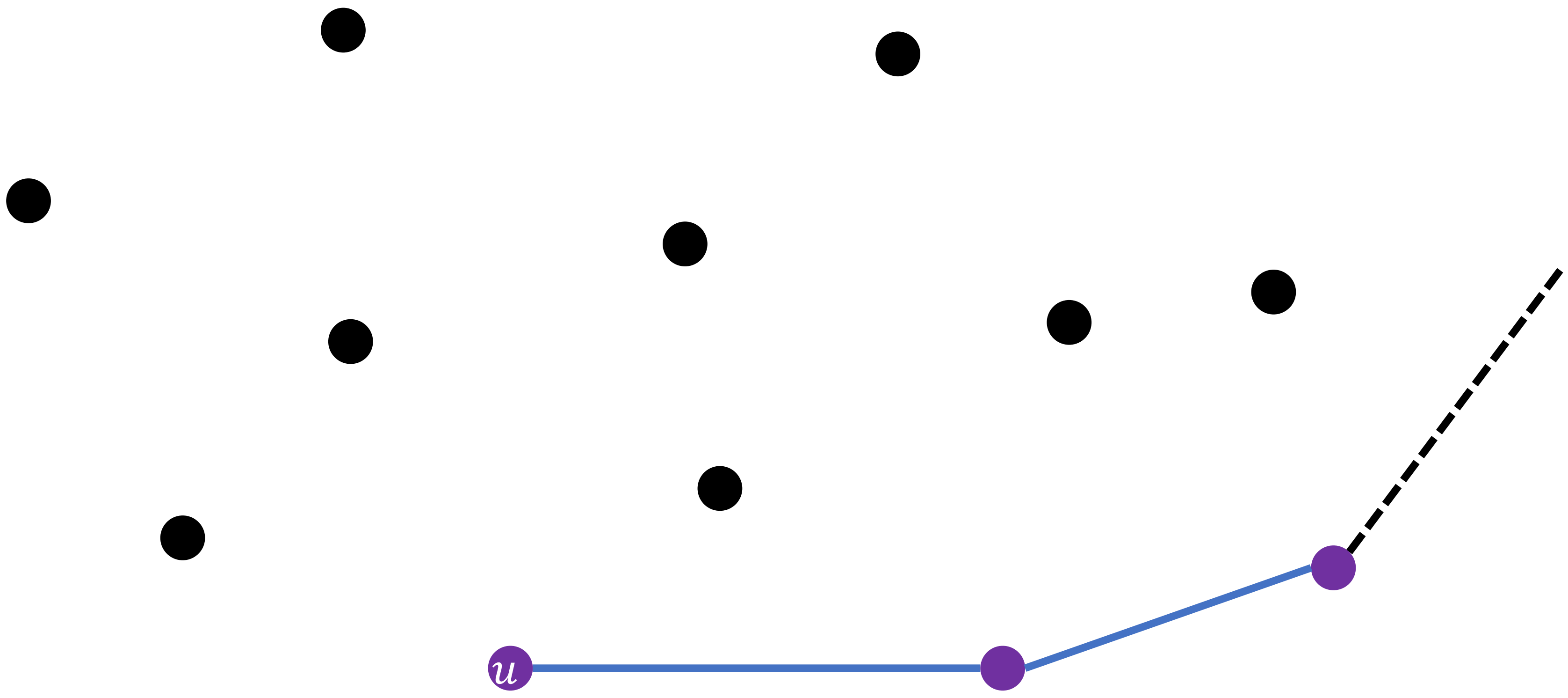
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



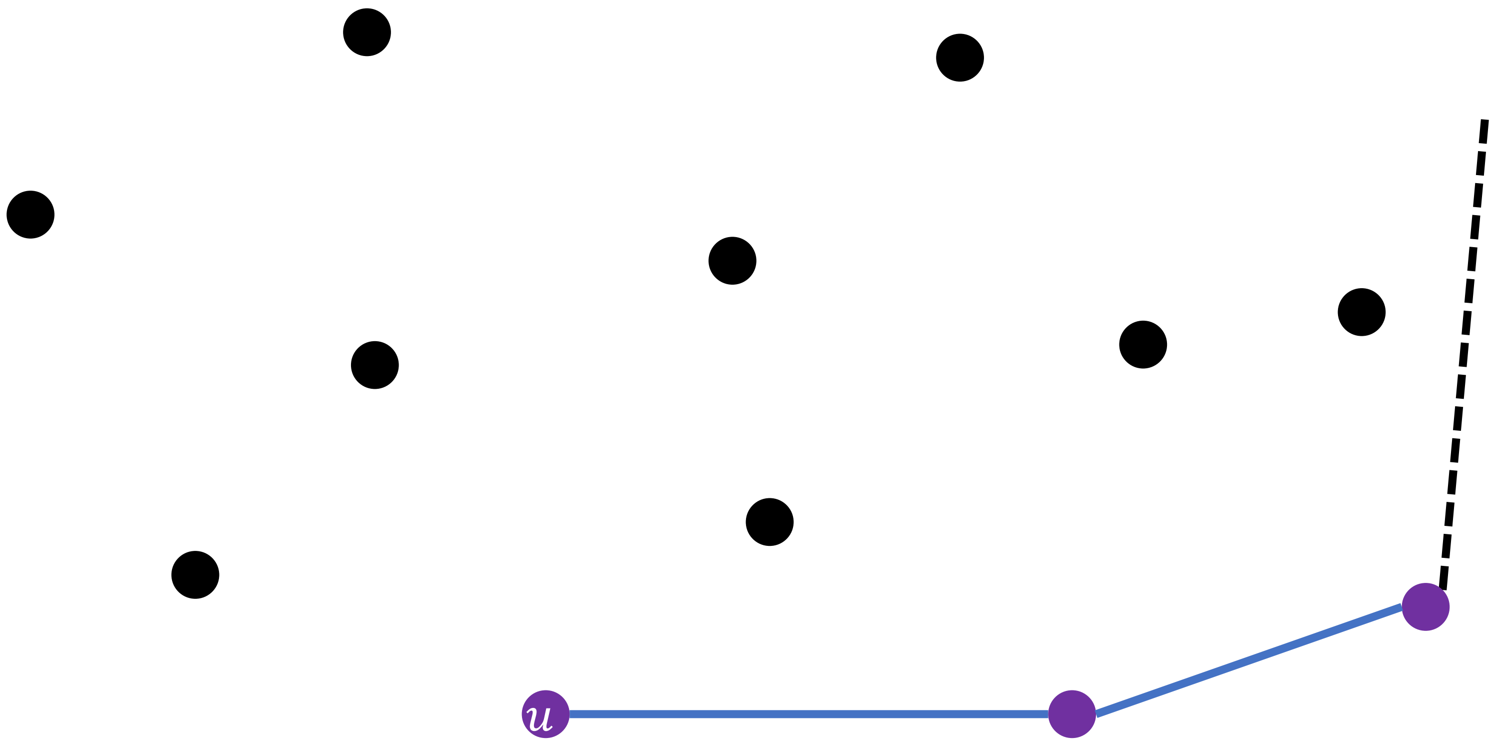
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



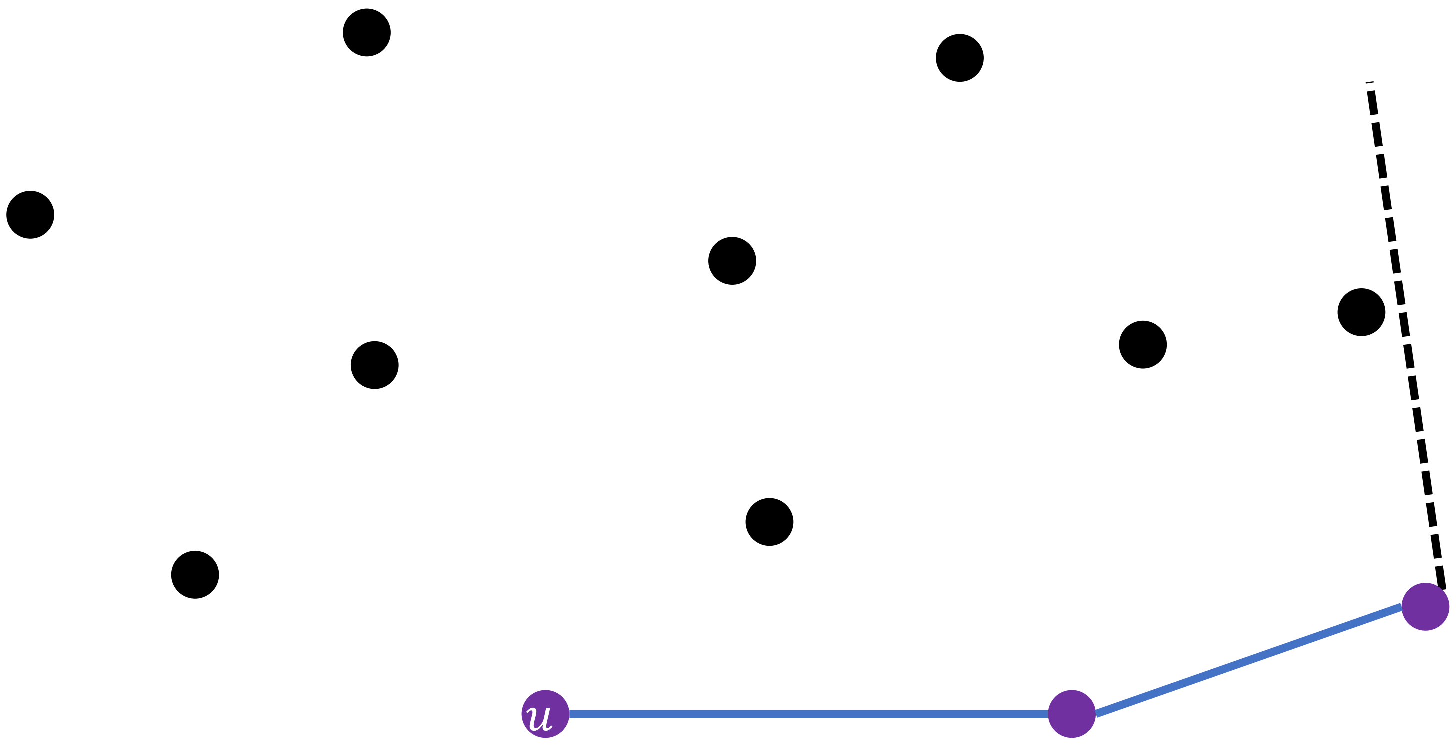
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



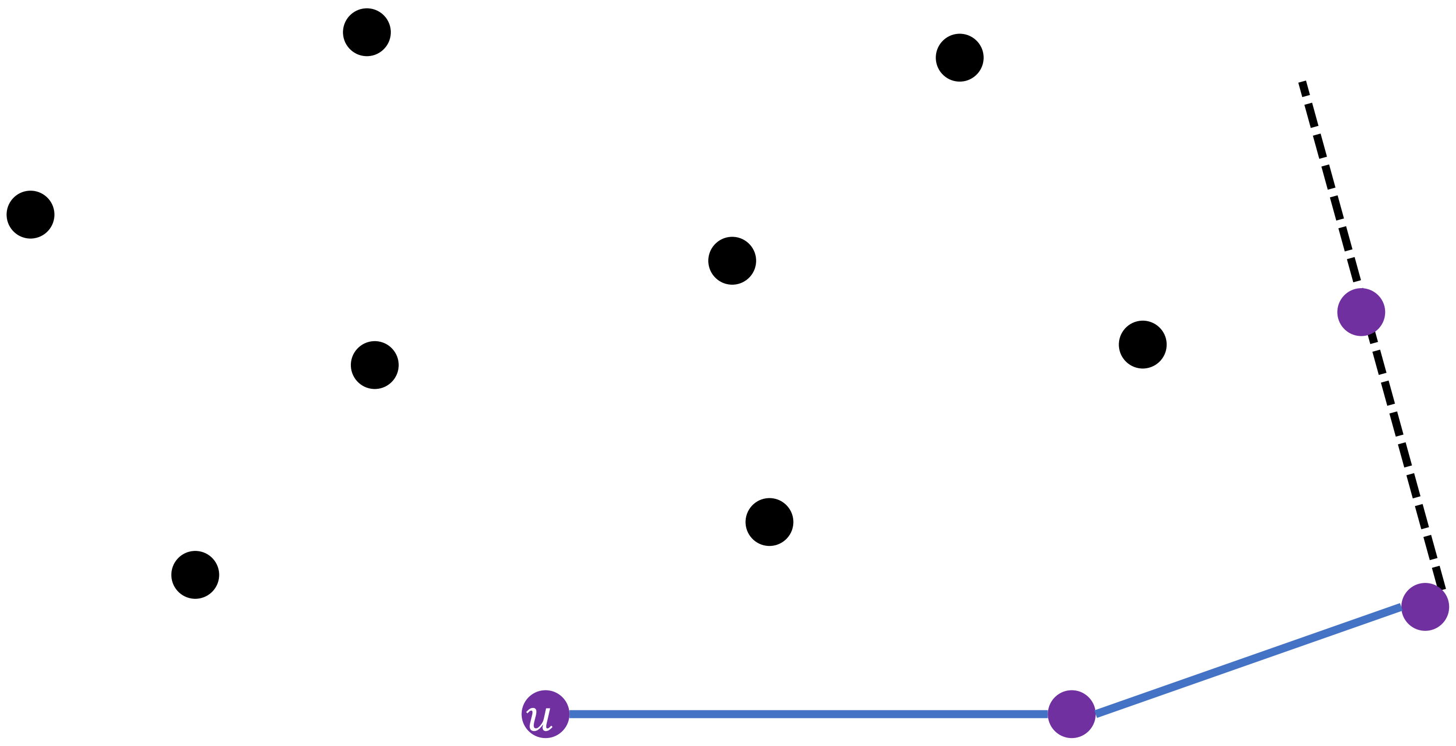
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis’ Algorithm (Gift Wrapping Method)



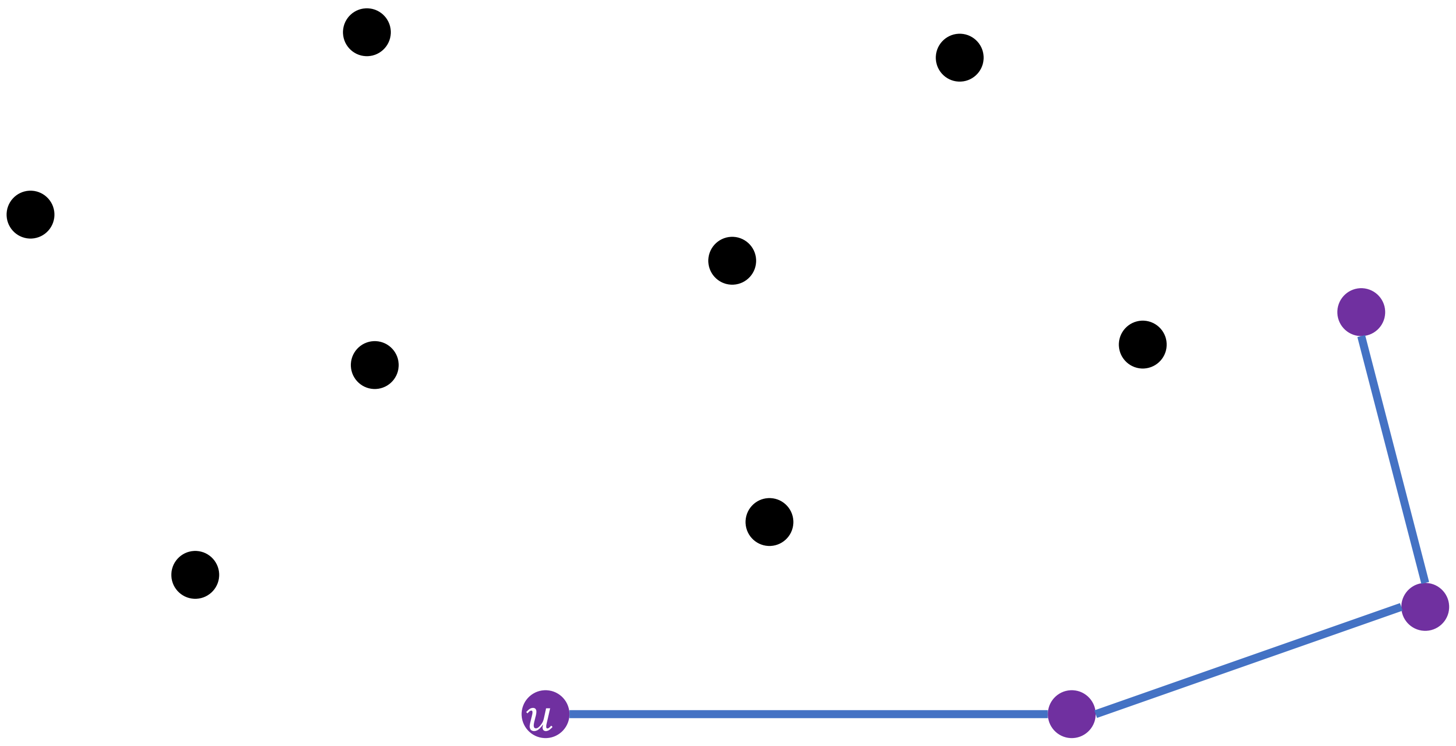
Idea: Start with extremal point and “wrap” points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



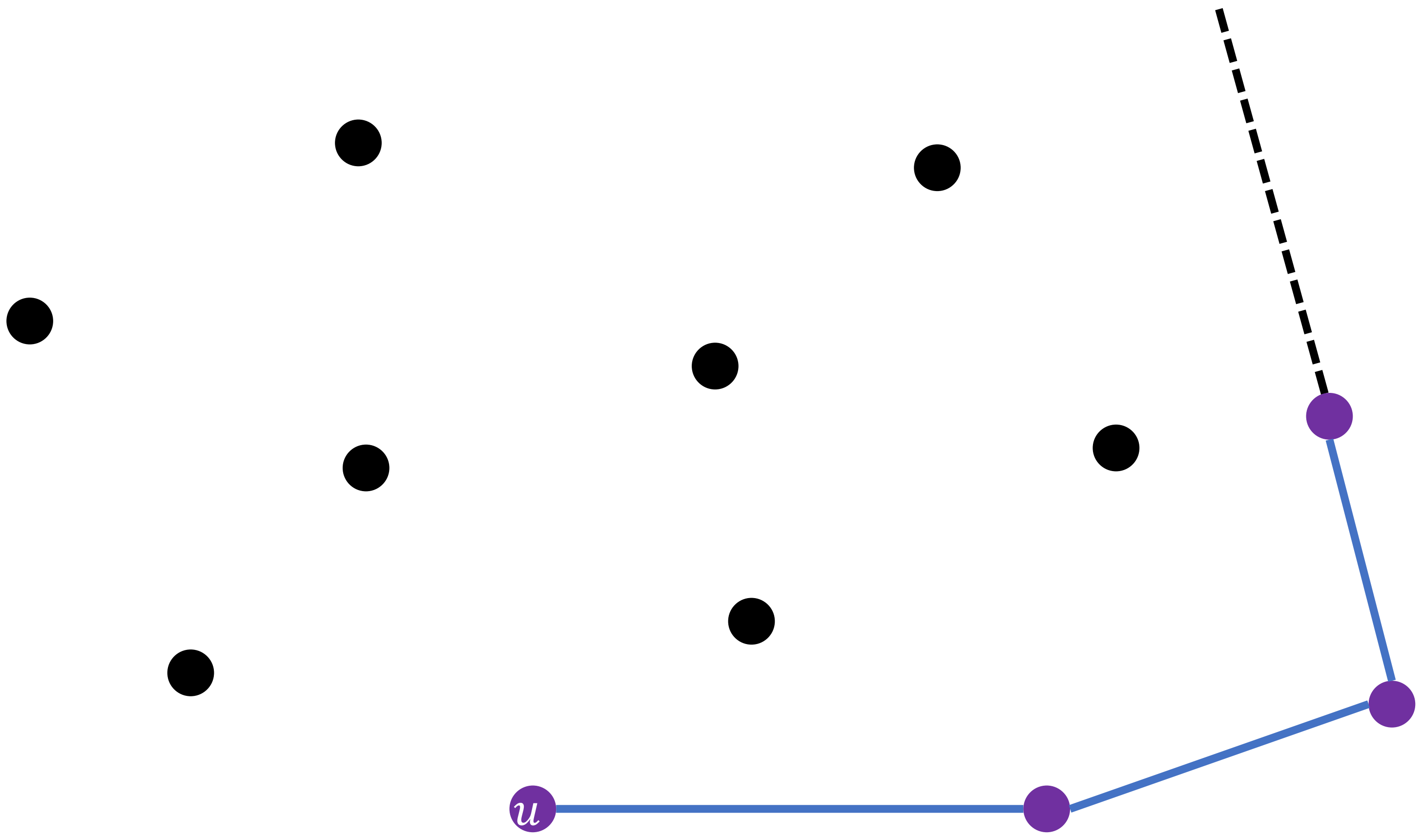
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



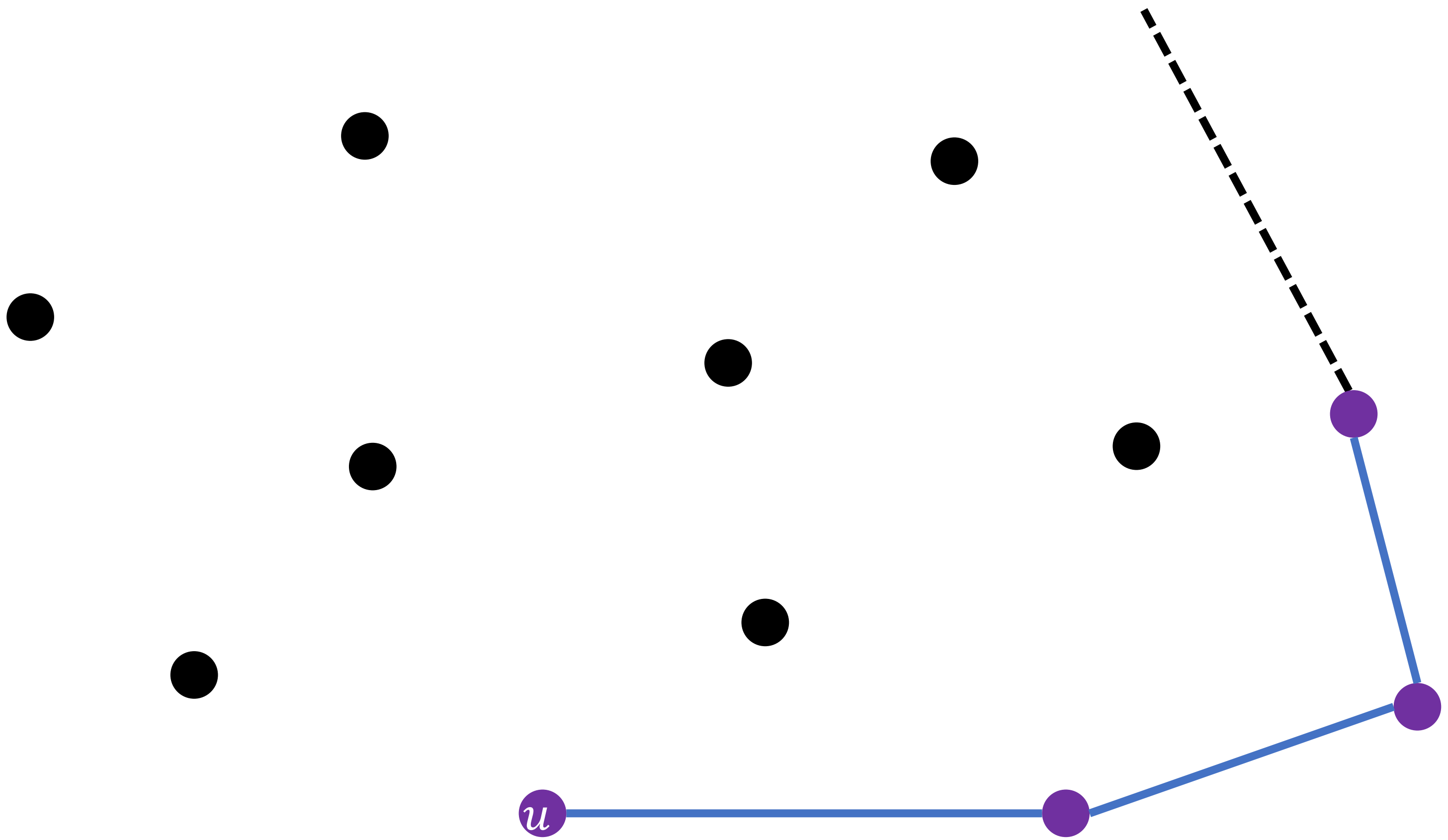
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



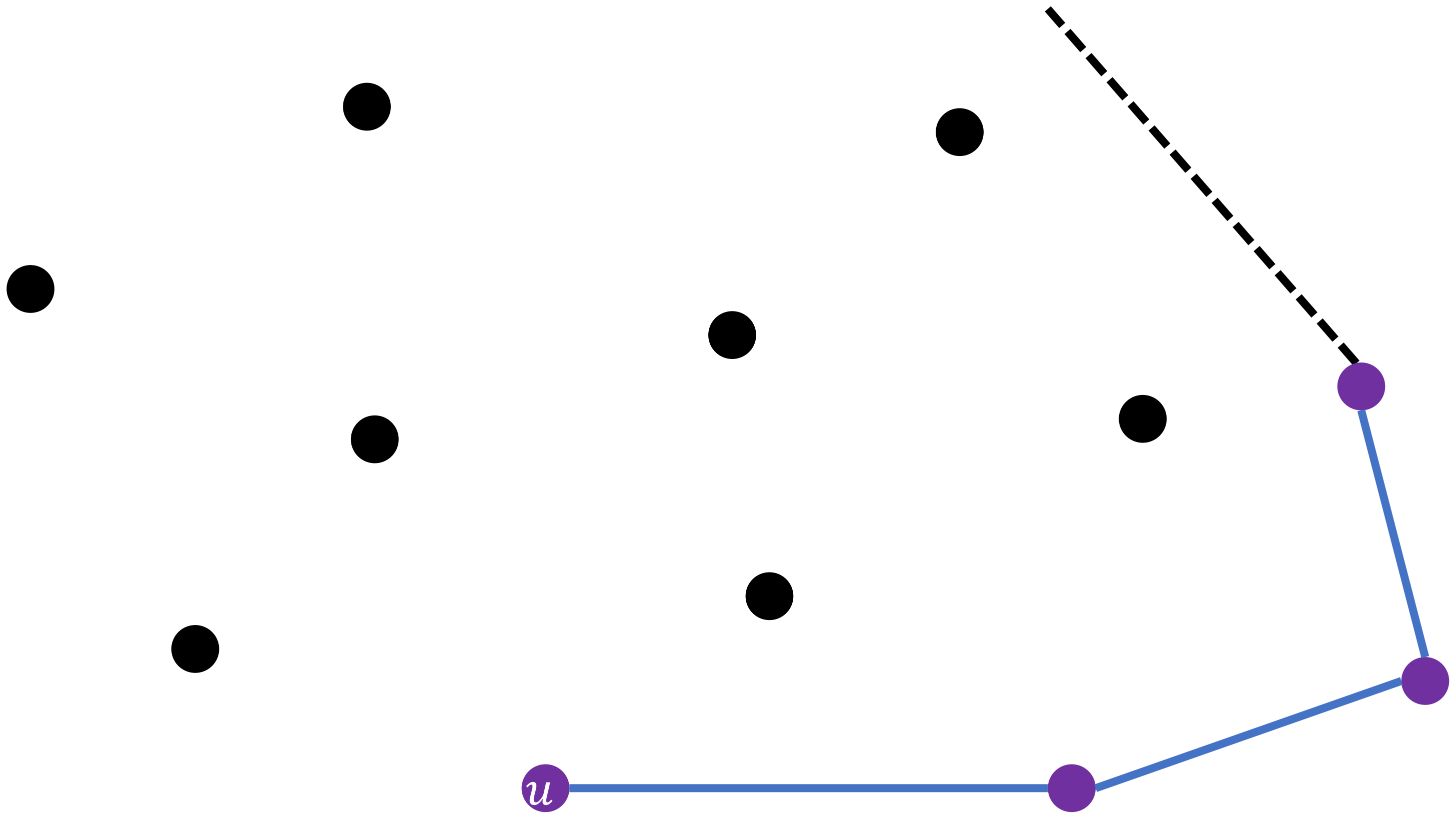
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



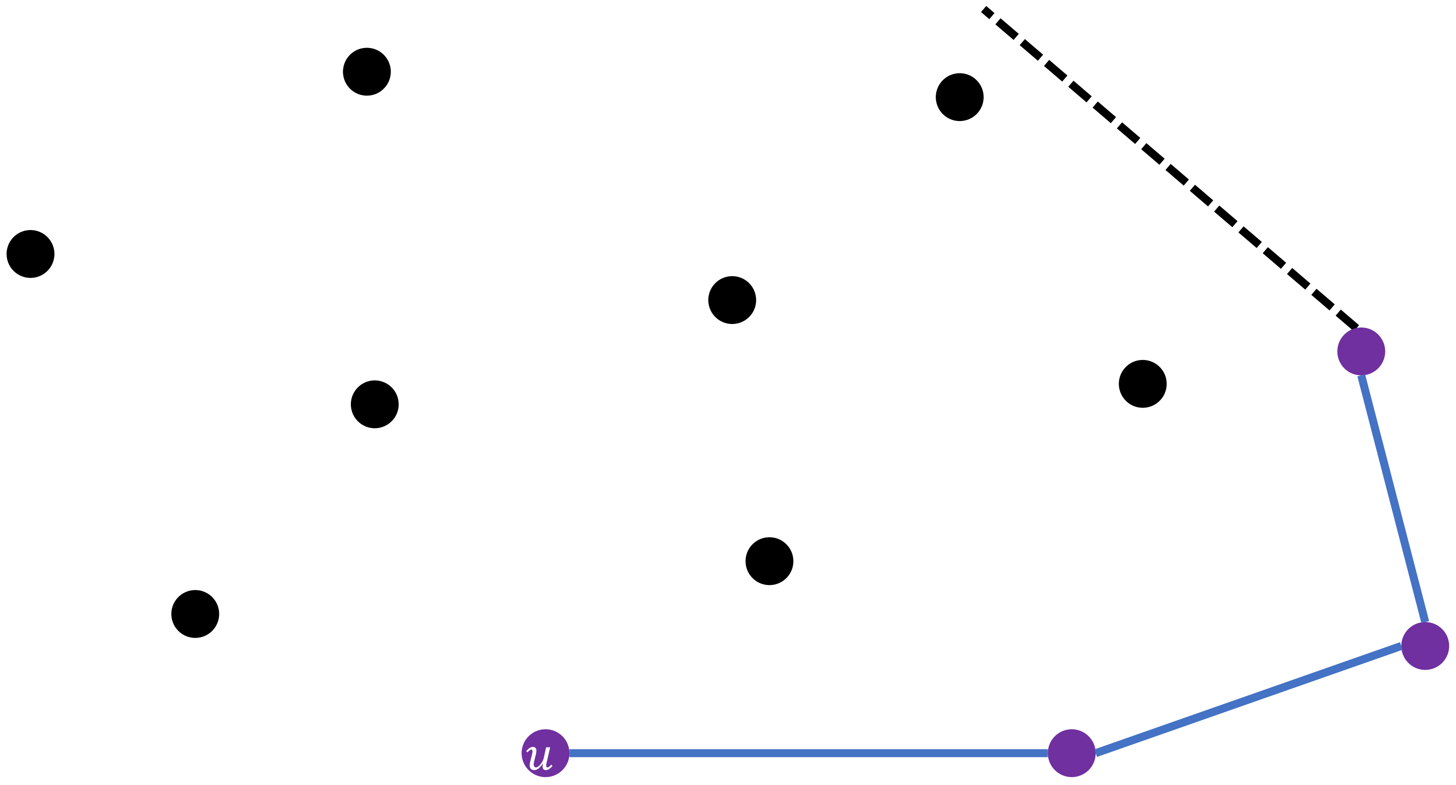
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



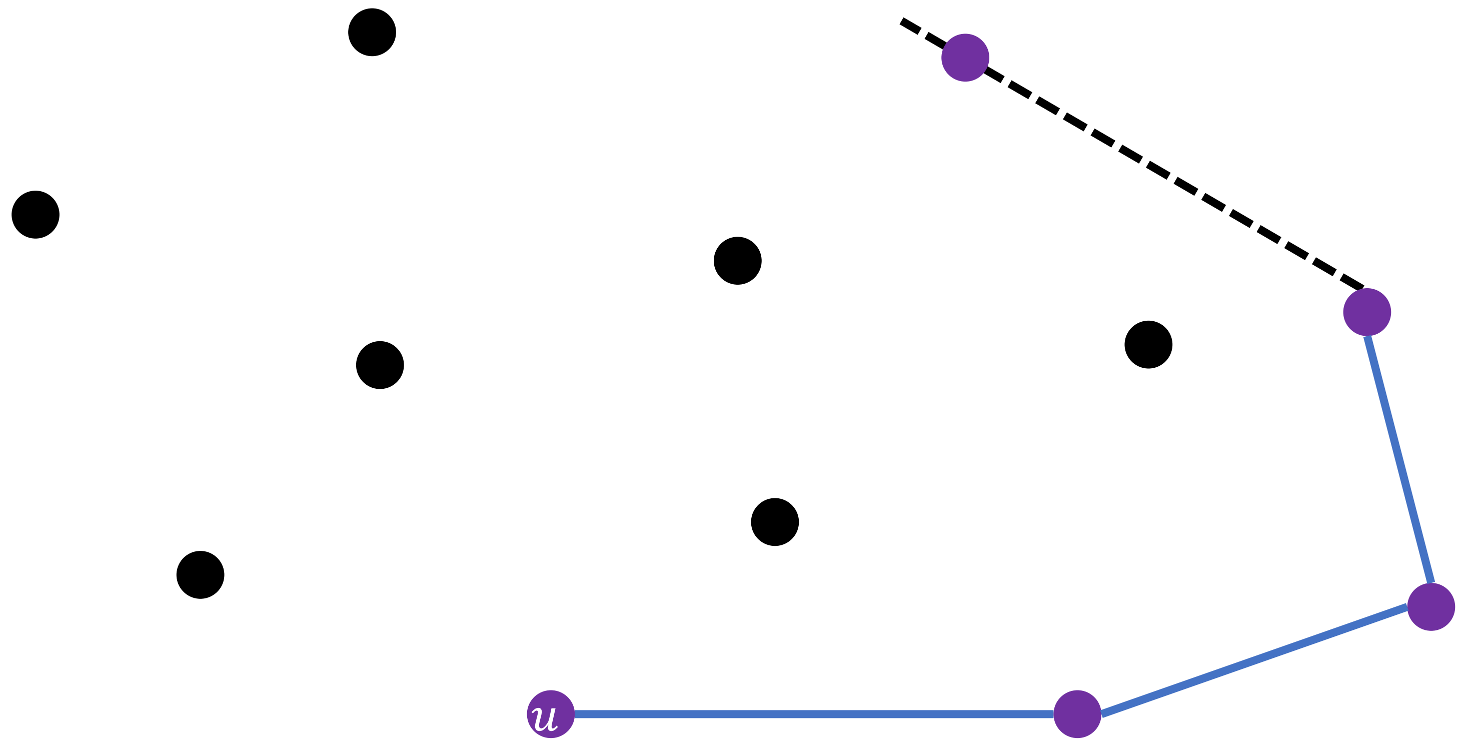
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



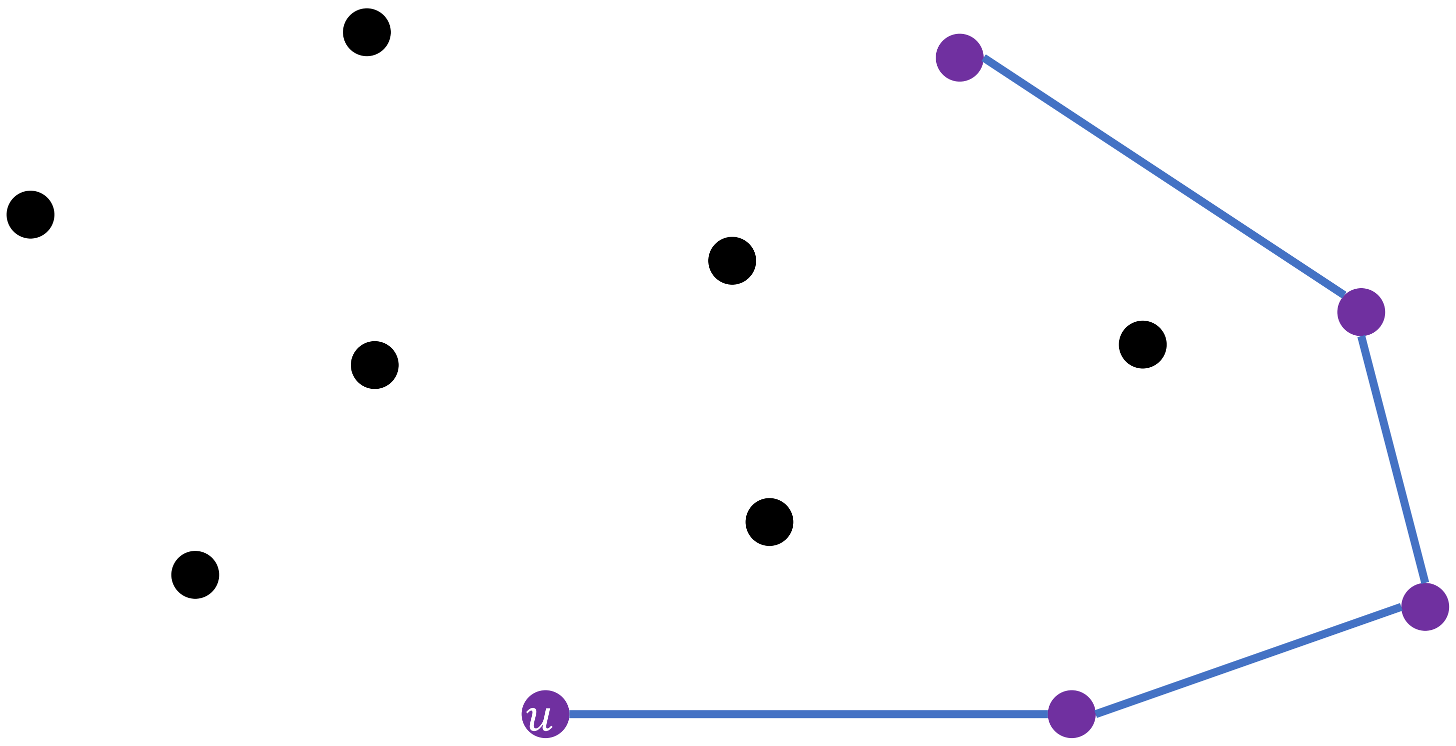
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



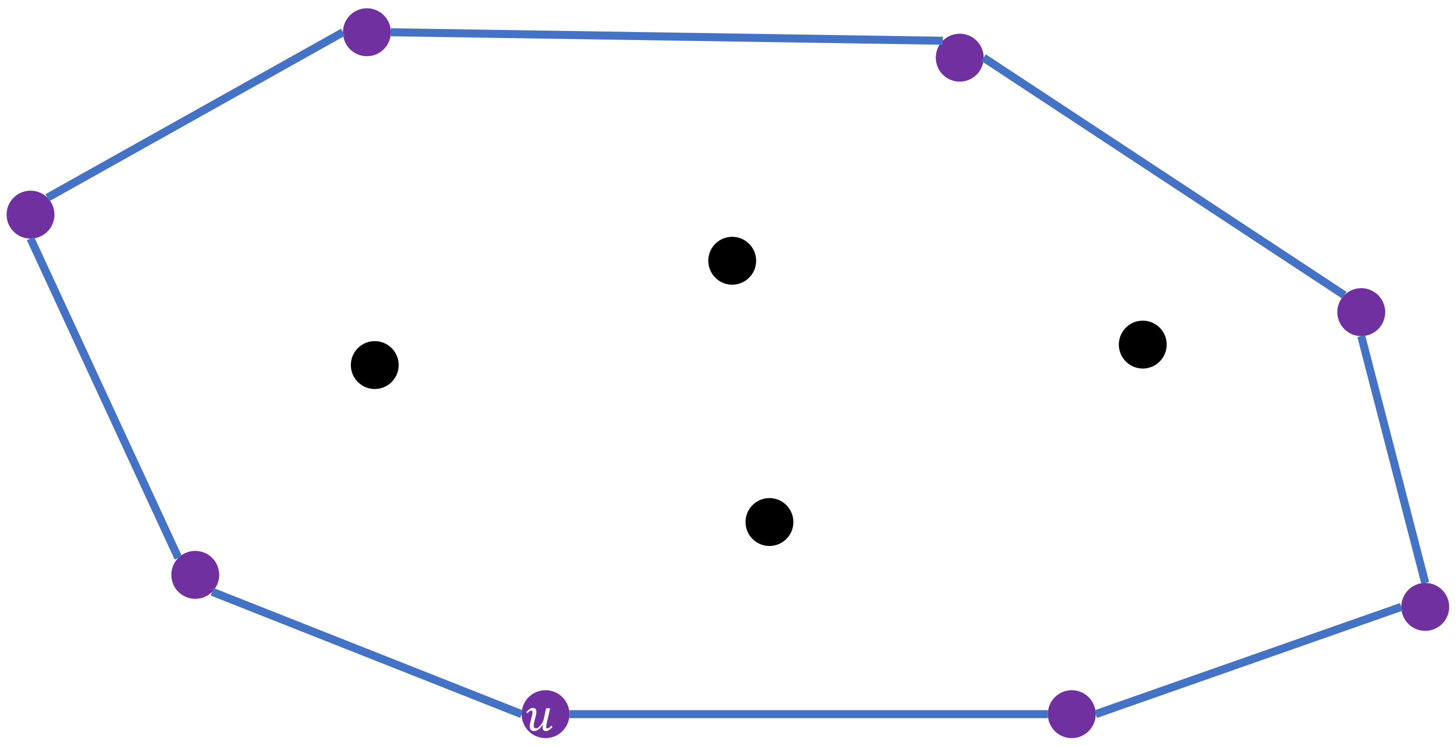
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



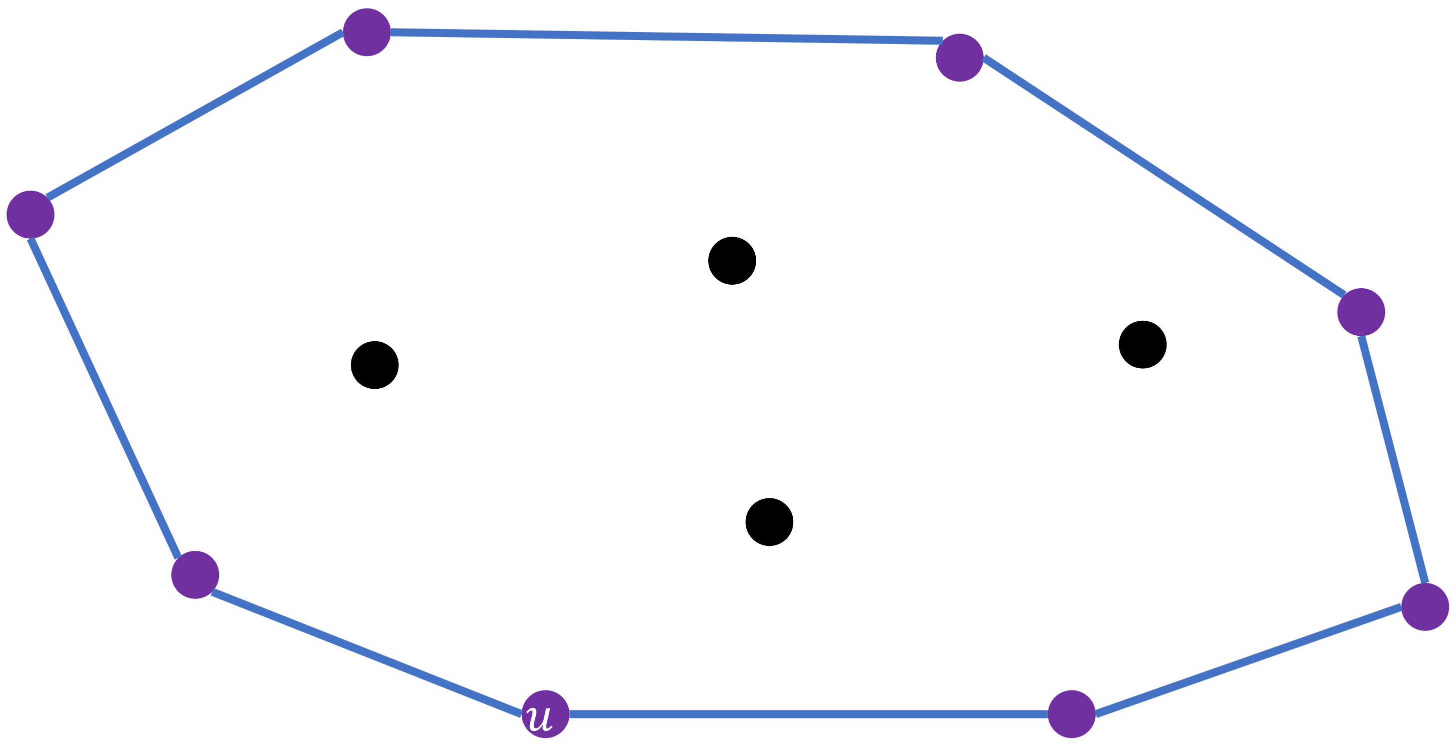
Idea: Start with extremal point and "wrap" points in counter-clockwise fashion

Jarvis' Algorithm (Gift Wrapping Method)



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

Jarvis' Algorithm (Gift Wrapping Method)

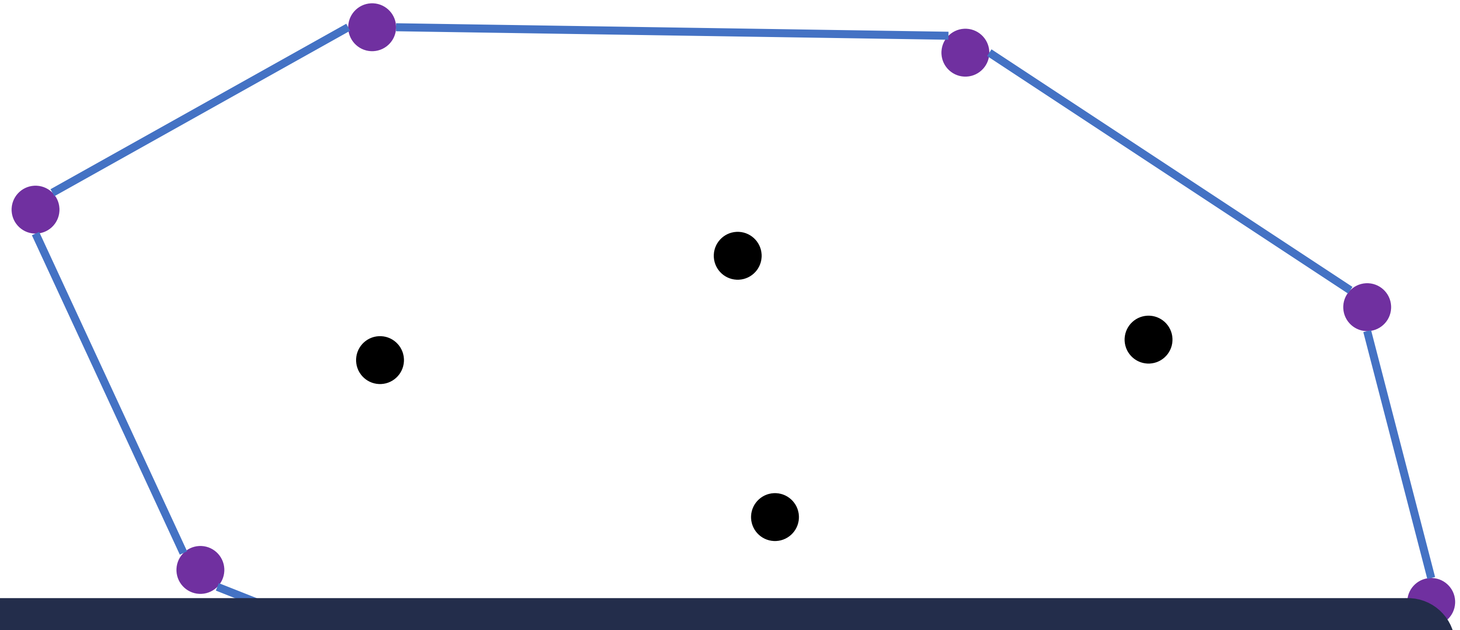


Can find the "next" point using a linear scan (i.e., point with largest angle)

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)



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 $O(n^2)$

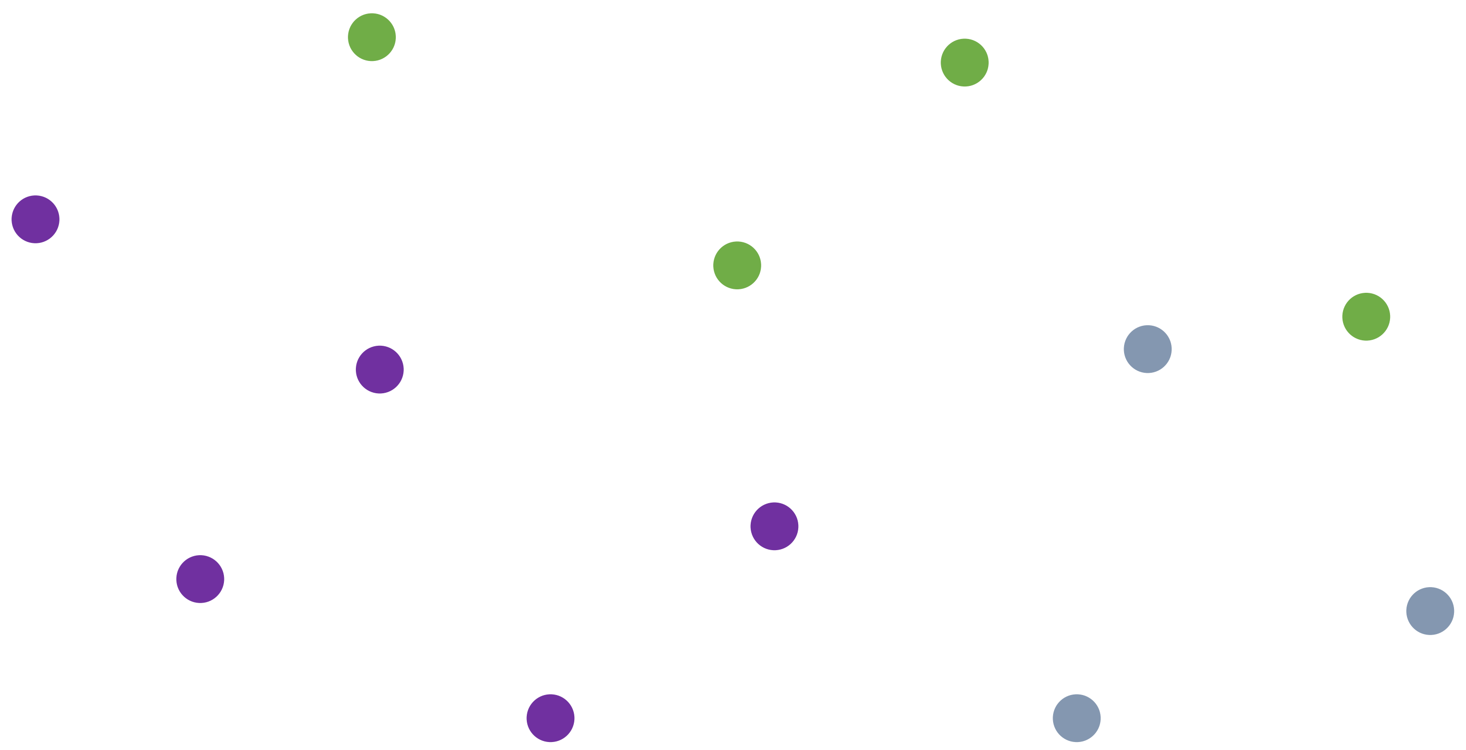
Ca

Number of iterat mber of points on convex hull

Run time: $O(nh)$ where h is the number of points on the convex hull

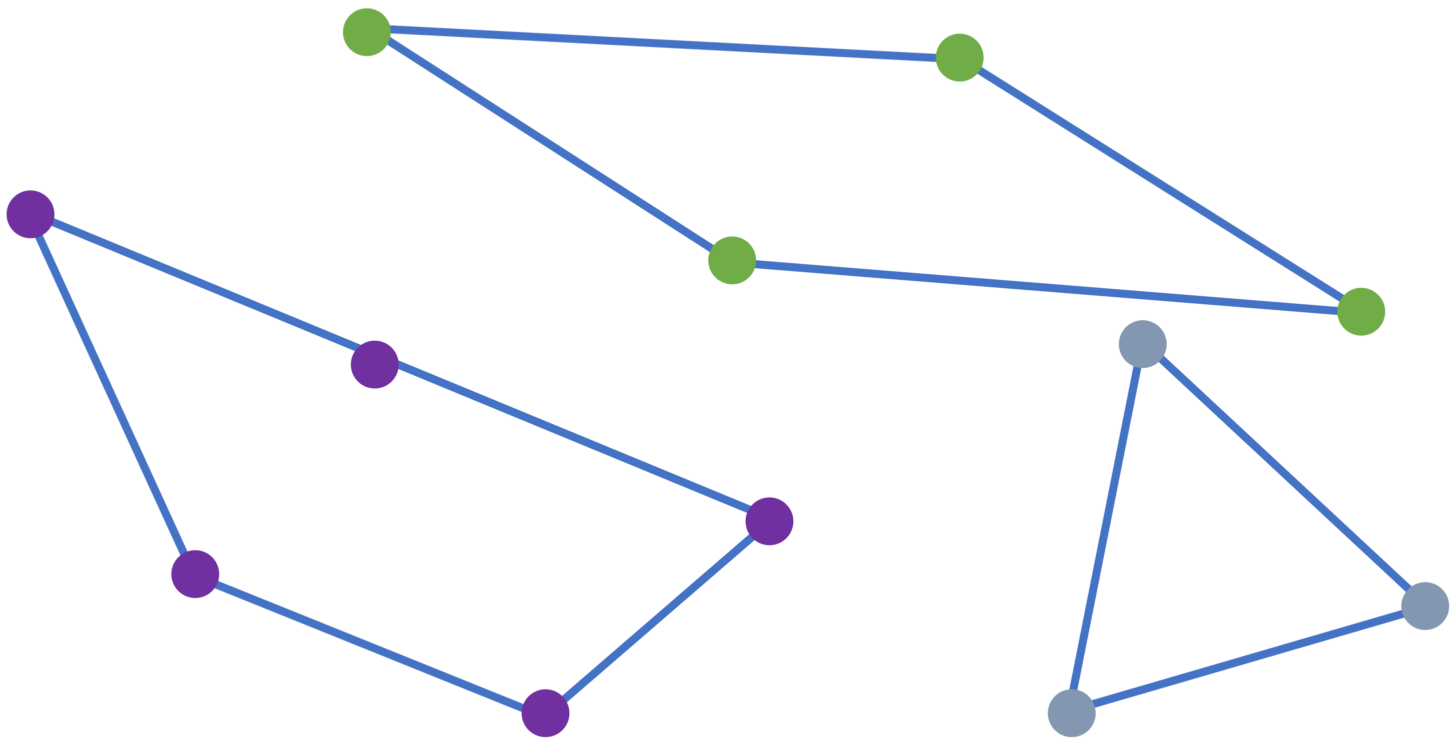
Chan’s Algorithm

Chan’s Algorithm



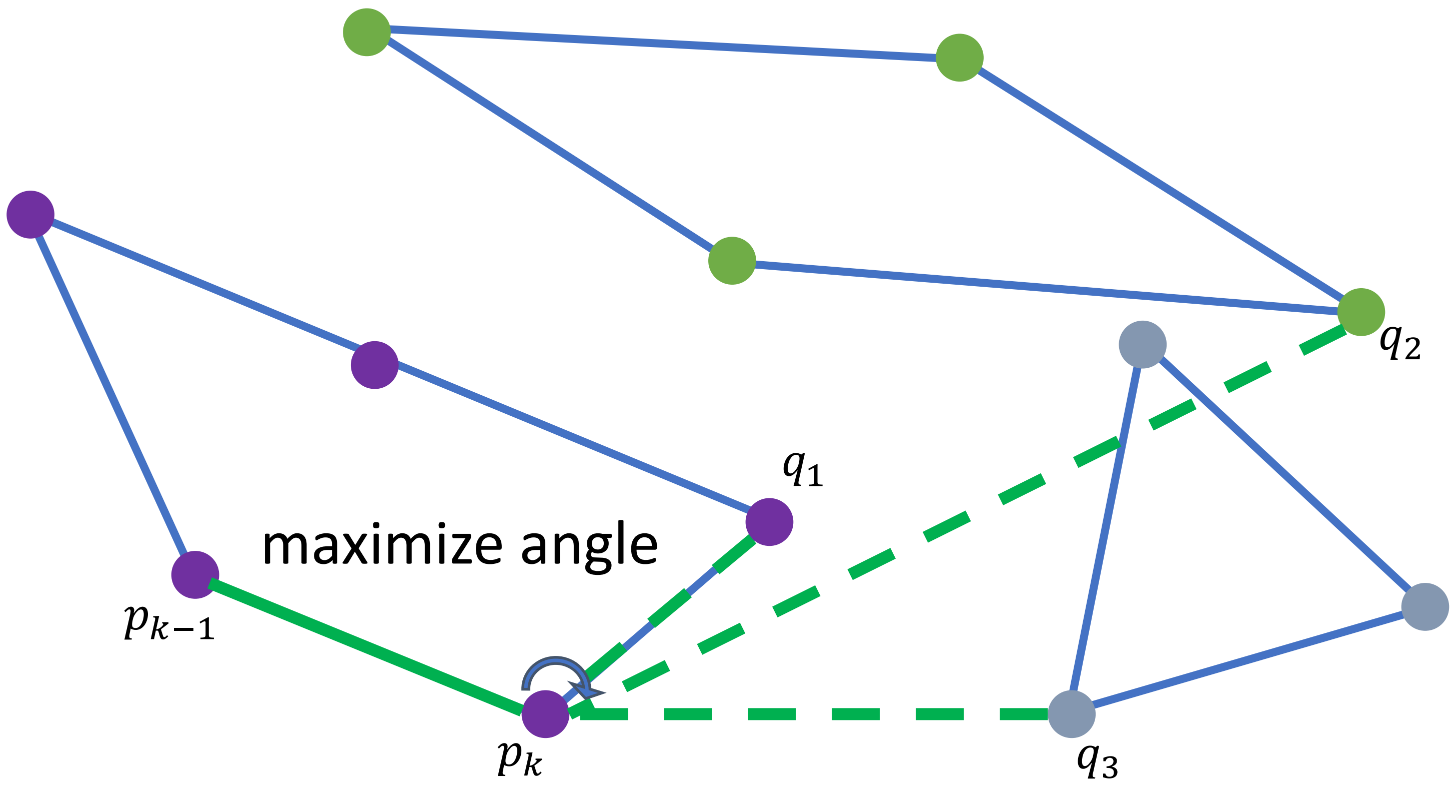
Divide into smaller subsets

Chan’s Algorithm



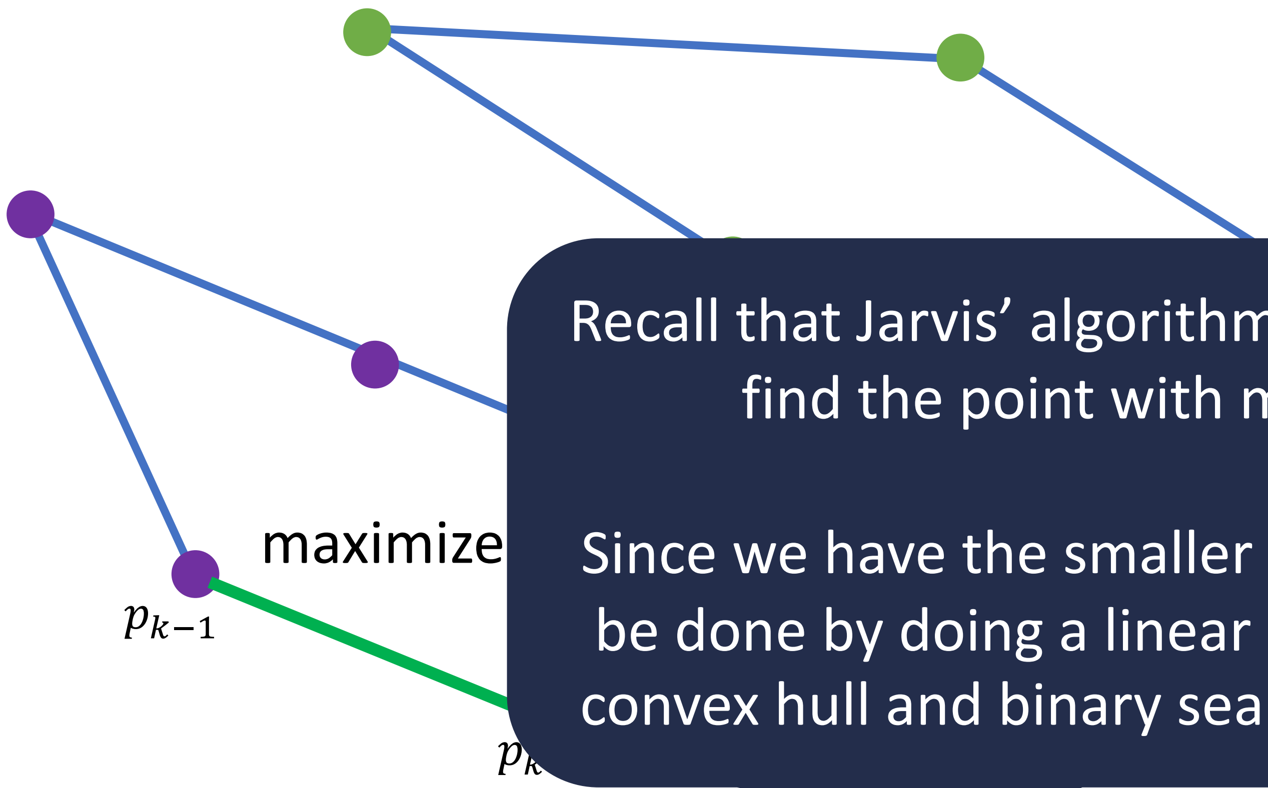
Use Graham’s Algorithm to **conquer** the smaller subsets

Chan's Algorithm



Use Jarvis' Algorithm to **combine** the solutions to the smaller subsets

Chan's Algorithm

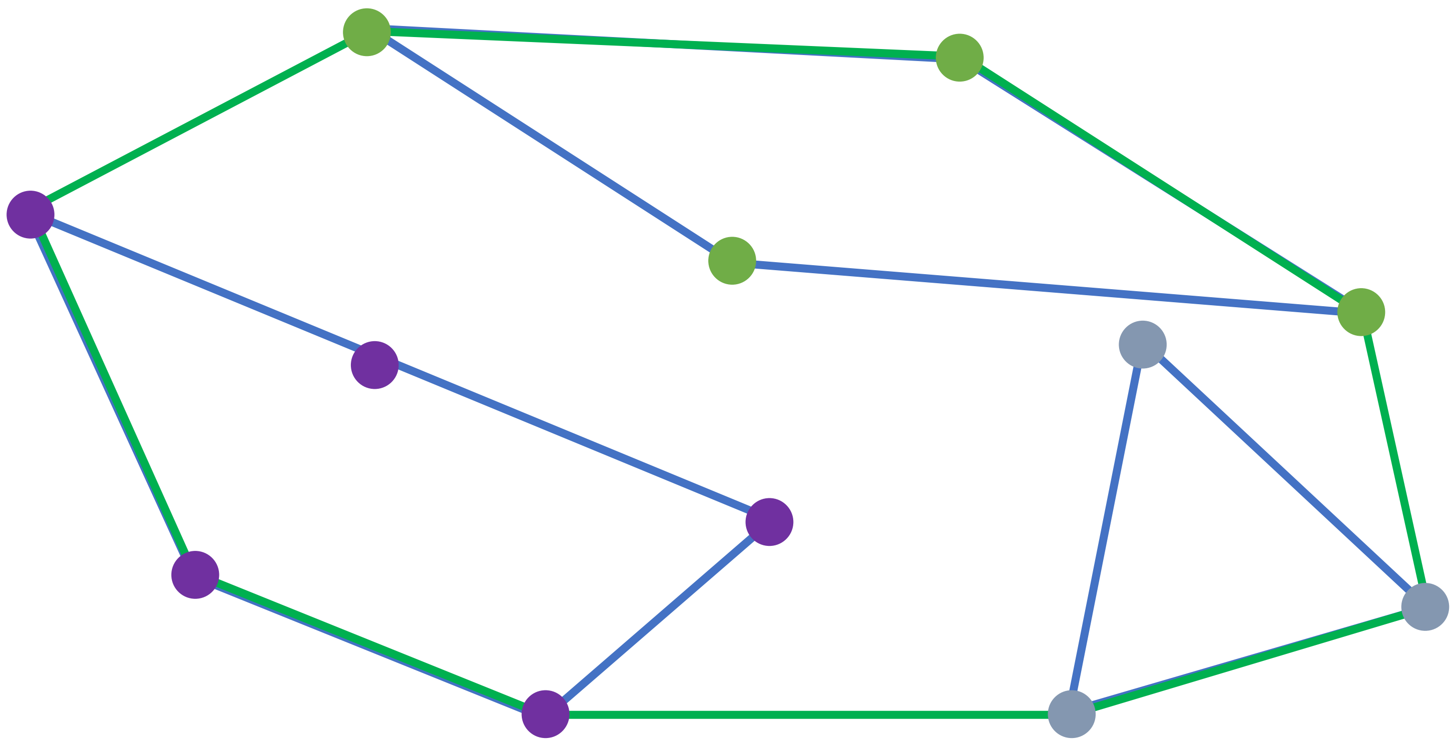


Recall that Jarvis' algorithm does a linear scan to find the point with maximum angle

Since we have the smaller convex hulls, this can be done by doing a linear scan over each small convex hull and binary searching within the hull

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

Running time: $O(n \log h)$ – optimal!