Convex Hull

Naresh Kota,Pranab Rai

Department of CSE,IIIT Bhubaneswar

January 5, 2026

Contents

Introduction

Convex Hull is a fundamental problem in **computational geometry**.

Definition: The convex hull of a set of points is the smallest convex polygon that encloses all points.

Applications:

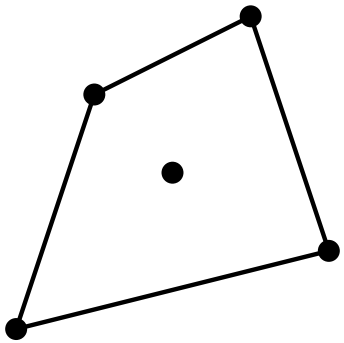
- Computer Graphics
- Collision Detection
- GIS

Concept Overview

Given a set of points in a 2D plane:

- Convex hull connects the outermost points
- All interior points lie inside the polygon
- Boundary points are called extreme points

Rubber Band Analogy



Imagine stretching a rubber band around the points. When released, it forms the convex hull.

Orientation Test

For points A(x_1, y_1), B(x_2, y_2), C(x_3, y_3):

$$val = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

Interpretation:

- $val = 0 \rightarrow$ Collinear
- $val > 0 \rightarrow$ Counter-Clockwise
- $val < 0 \rightarrow$ Clockwise

Orientation Test – Worked Example

Let:

$$A(1,1),\ B(3,1),\ C(2,2)$$

Compute:

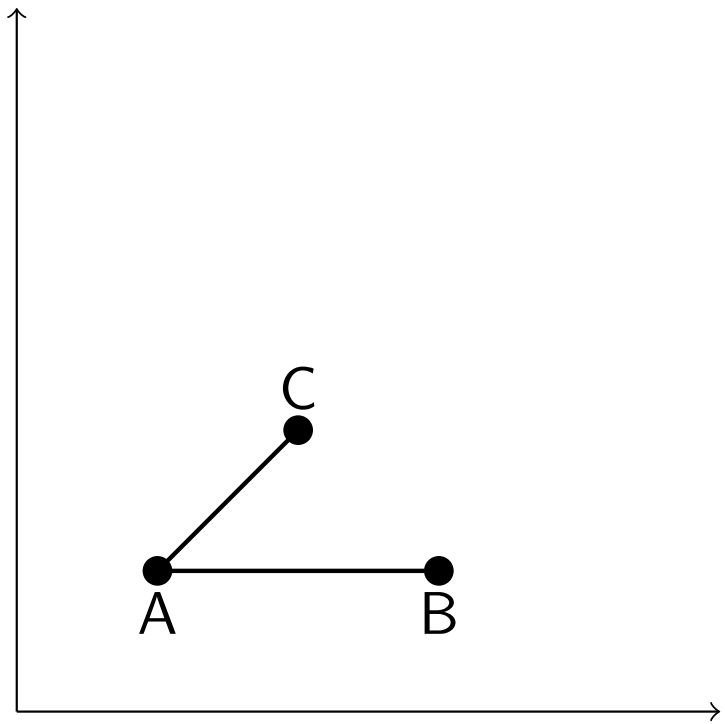
$$val = (x_B - x_A)(y_C - y_A) - (y_B - y_A)(x_C - x_A)$$

$$= (3-1)(2-1) - (1-1)(2-1)$$

$$= 2 \times 1 - 0 = 2$$

Since $val > 0$, the orientation is **Counter-Clockwise (Left Turn)**.

Orientation Test – Diagram



Since point C lies to the left of line AB, the orientation is **Counter-Clockwise**.

Naive (Brute Force) Algorithm

Idea:

- For every pair of points (P_i, P_j)
- Check if all other points lie on one side
- If yes $\rightarrow$ hull edge

Time Complexity: $O(n^3)$ **Space Complexity:** $O(1)$

Naive Algorithm – Numerical Example

Consider edge candidate:

$$P_1(1,1),\ P_2(3,1)$$

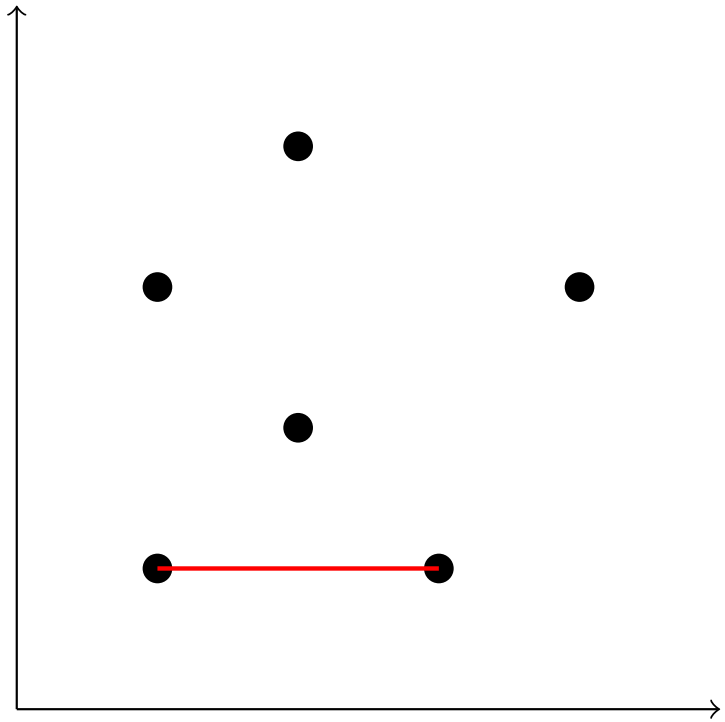
Check remaining points:

- $(2,2) \rightarrow$ above the line
- $(4,3) \rightarrow$ above the line
- $(2,4) \rightarrow$ above the line
- $(1,3) \rightarrow$ above the line

All points lie on the **same side**.

Conclusion: Edge $(1,1) - (3,1)$ belongs to the convex hull.

Naive Algorithm – Edge Verification



All other points lie above the red line.

Conclusion: Edge $(1, 1) - (3, 1)$ is part of the convex hull.

Naive Convex Hull – Time Complexity Analysis

Let n be the number of input points.

Step 1: Selecting Point Pairs

- Consider every pair of points (P_i, P_j) as a candidate hull edge
- Total number of pairs: $\binom{n}{2} = O(n^2)$

Step 2: Checking Side of Line

- For each pair (P_i, P_j), check all remaining points
- Use orientation / line equation to determine the side
- Number of remaining points: $n - 2 \approx O(n)$
- Time per pair: $O(n)$

Step 3: Hull Edge Decision

- If all points lie on the same side, the edge is part of the convex hull
- Otherwise, discard the edge

Total Time Complexity

$$T(n) = O(n^2) \times O(n) = \boxed{O(n^3)}$$

Space Complexity

$$\boxed{O(1)}$$

(Only constant extra variables are used)

Graham Scan – Core Idea

Graham Scan constructs the convex hull by:

- Sorting points by polar angle
- Removing points that cause right turns
- Using a stack to maintain convexity

Graham Scan – Algorithm Steps

1. Find the point with lowest y-coordinate (pivot P_0)
2. Sort all other points by polar angle with respect to P_0
3. Push first three points into a stack
4. For each remaining point:
   - While the top two stack points and current point make a clockwise turn, pop the stack
   - Push the current point
5. Stack contains convex hull

Graham Scan – Example (Step 1 & 2)

Given points:

$$(1,1),(2,2),(3,1),(4,3),(2,4),(1,3)$$

Step 1: Pivot Selection

Lowest y-coordinate point:

$$P_0 = (1,1)$$

Step 2: Polar Angle Sorting

Sorted order:

$$(1,1),(3,1),(4,3),(2,4),(1,3),(2,2)$$

Graham Scan – Example (Stack Operations)

Initial stack:

$$(1,1),(3,1),(4,3)$$

Check next point $(2,4)$:

$$\mathit{Orientation}((3,1),(4,3),(2,4)) = \mathit{CCW}$$

Push $(2,4)$

Next point $(2,2)$:

$$\mathit{Orientation}((4,3),(2,4),(2,2)) = \mathit{CW}$$

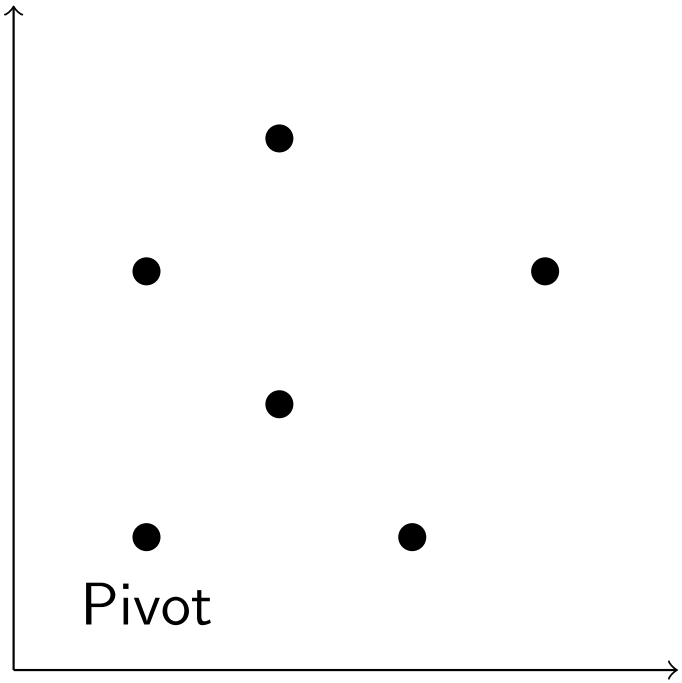
Pop $(2,4)$

Final Hull Points:

$$(1,1),(3,1),(4,3),(2,4),(1,3)$$

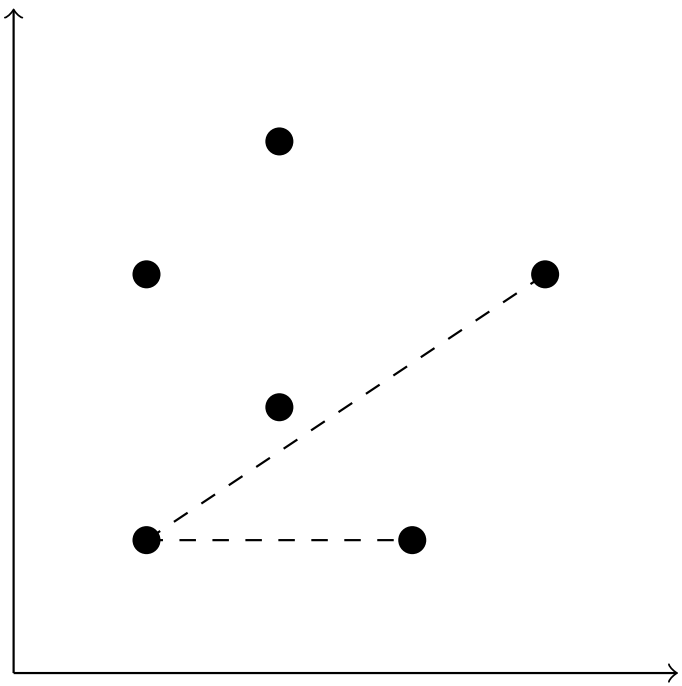
Graham Scan – Code and Geometry Sync

Step 1: Select Pivot P_0



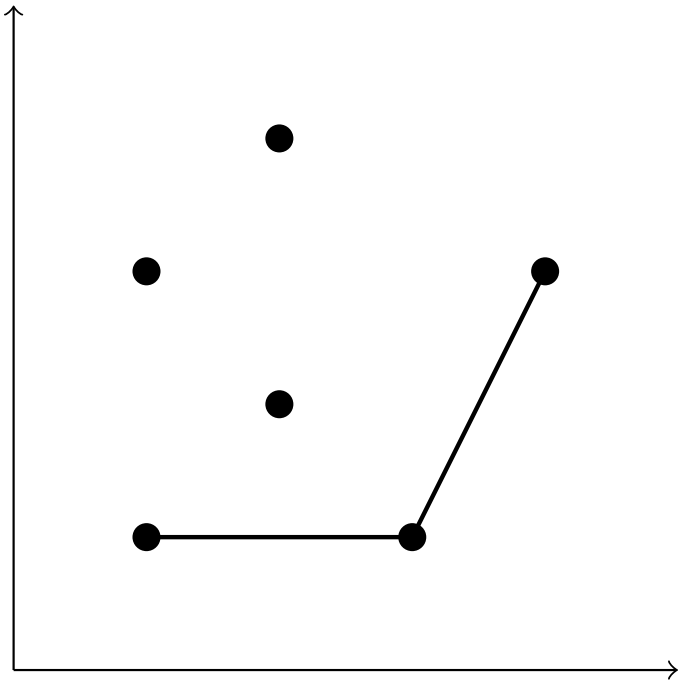
Graham Scan – Code and Geometry Sync

Step 2: Sort by polar angle



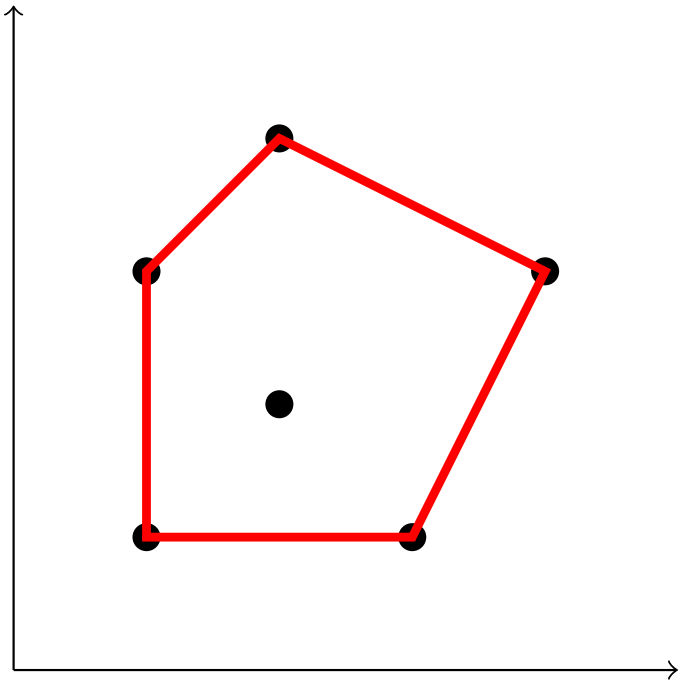
Graham Scan – Code and Geometry Sync

Step 3: Push points to stack



Graham Scan – Code and Geometry Sync

Step 4: Remove clockwise turns



Graham Scan – Time Complexity Analysis

Let n be the number of input points.

Step 1: Finding the Pivot Point

- Scan all points to find the lowest y-coordinate
- Time Complexity: $O(n)$

Step 2: Sorting by Polar Angle

- Sort $n-1$ points based on polar angle w.r.t pivot
- Each comparison uses orientation test ($O(1)$)
- Time Complexity: $O(n \log n)$

Step 3: Stack Initialization

- Push first three points into stack
- Time Complexity: $O(1)$

Step 4: Stack Processing (Scan Phase)

- Each point is pushed once and popped at most once
- Total stack operations $\leq 2n$
- Time Complexity: $O(n)$

Total Time Complexity

$$T(n) = O(n) + O(n \log n) + O(1) + O(n) = \boxed{O(n \log n)}$$

Jarvis March – Intuition

Jarvis March works like wrapping a gift:

- Start from an extreme point
- Repeatedly select the most counter-clockwise point
- Stop when the start point is reached

Jarvis March – Algorithm

1. Find the leftmost point
2. Set it as the current hull point
3. For every other point, find the one that makes the most CCW turn
4. Add it to hull and repeat

Jarvis March – Worked Example

Step 1: Start from leftmost point:

$$(1, 1)$$

Step 2: Choose most CCW point:

$$(3, 1)$$

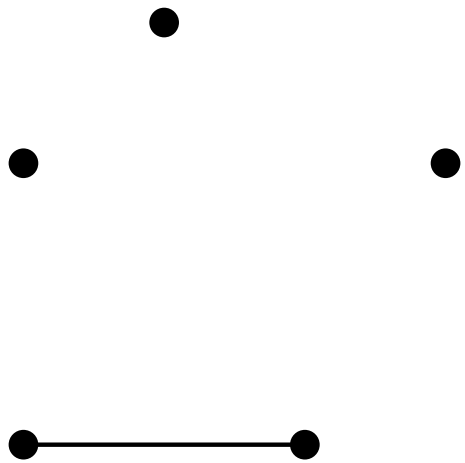
Repeat CCW selection:

$$(3, 1) \rightarrow (4, 3) \rightarrow (2, 4) \rightarrow (1, 3)$$

Return to starting point $(1, 1)$.

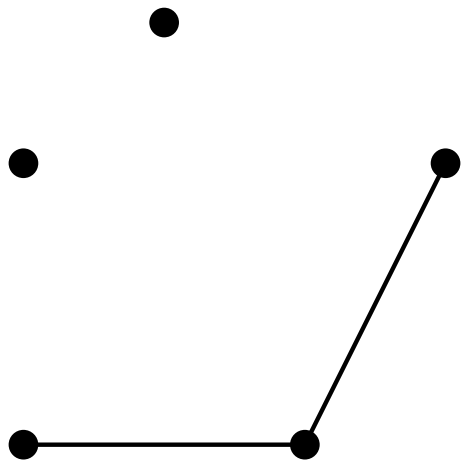
Hull completed.

Jarvis March – Gift Wrapping Animation



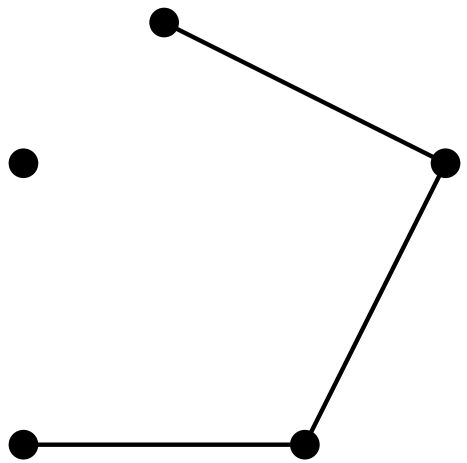
Wrapping proceeds by selecting the most CCW point.

Jarvis March – Gift Wrapping Animation



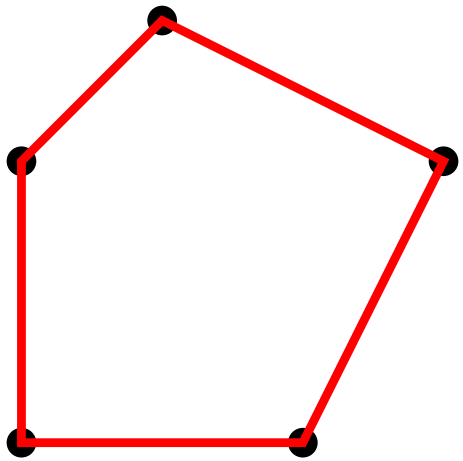
Wrapping proceeds by selecting the most CCW point.

Jarvis March – Gift Wrapping Animation



Wrapping proceeds by selecting the most CCW point.

Jarvis March – Gift Wrapping Animation



Wrapping proceeds by selecting the most CCW point.

Jarvis March (Gift Wrapping) – Time Complexity

Let:

- n = total number of input points
- h = number of points on the convex hull

Step 1: Find the Starting Point

- Find the leftmost (or lowest) point by scanning all points
- Time Complexity: $O(n)$

Step 2: Selecting the Next Hull Point

- For the current hull point, check orientation with all other points
- Select the most counter-clockwise point
- Time Complexity per hull point: $O(n)$

Step 3: Wrapping Until Return to Start

- The above step is repeated for each hull point
- Total number of hull points = h

Total Time Complexity

$$T(n) = O(n) + h \cdot O(n) = \boxed{O(nh)}$$

Best Case: h is small (e.g., triangle) $\Rightarrow O(n)$
Worst Case: All points on hull ($h = n$) $\Rightarrow O(n^2)$

Divide and Conquer – Strategy

This approach follows the merge sort paradigm:

- Divide the point set
- Solve subproblems recursively
- Merge partial hulls using tangents

Divide and Conquer – Steps

1. Sort points by x-coordinate
2. Divide into left and right halves
3. Recursively compute hulls
4. Merge hulls using upper and lower tangents

Divide and Conquer – Example

Sorted by x-coordinate:

$$(1,1),(1,3),(2,2),(2,4),(3,1),(4,3)$$

Divide into:

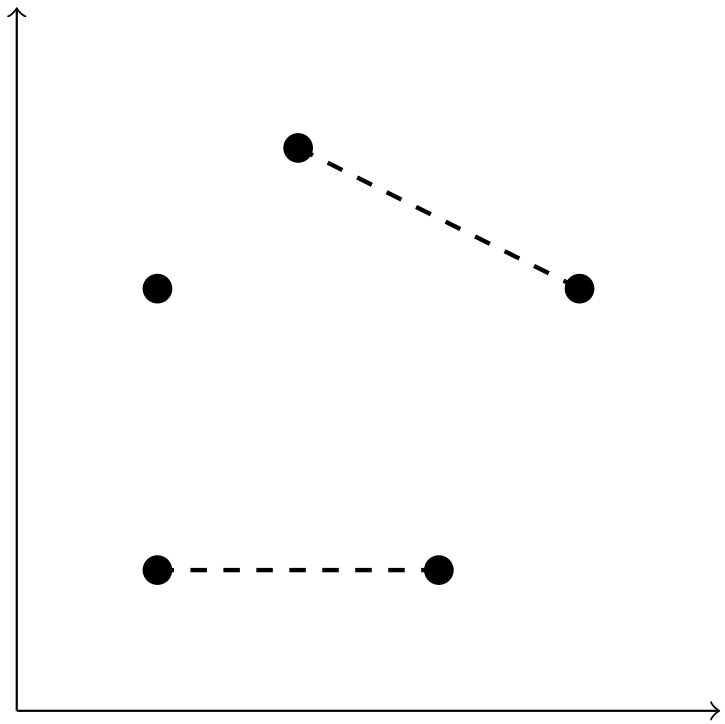
- Left Hull: $(1,1),(1,3),(2,4)$
- Right Hull: $(3,1),(4,3)$

Merge using upper and lower tangents.

Final Hull:

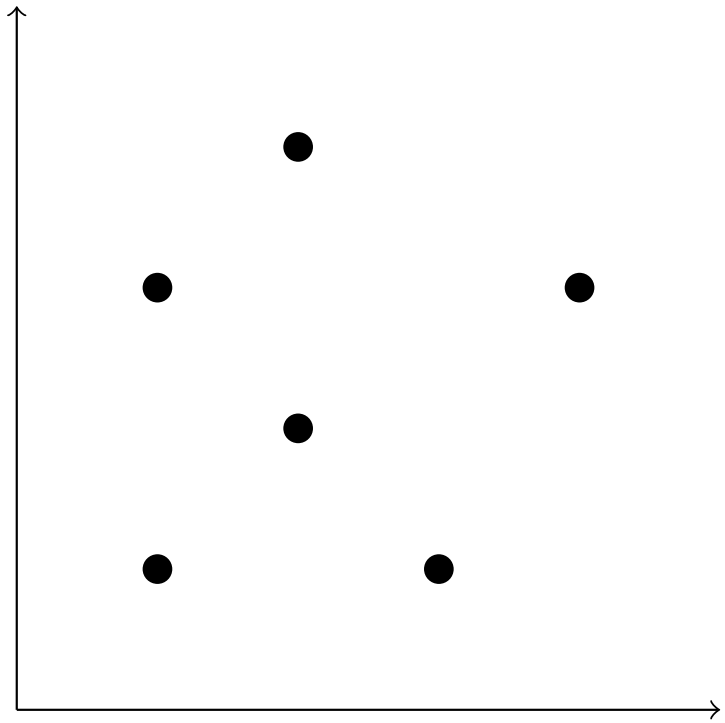
$$(1,1),(3,1),(4,3),(2,4),(1,3)$$

Divide and Conquer – Merging Hulls



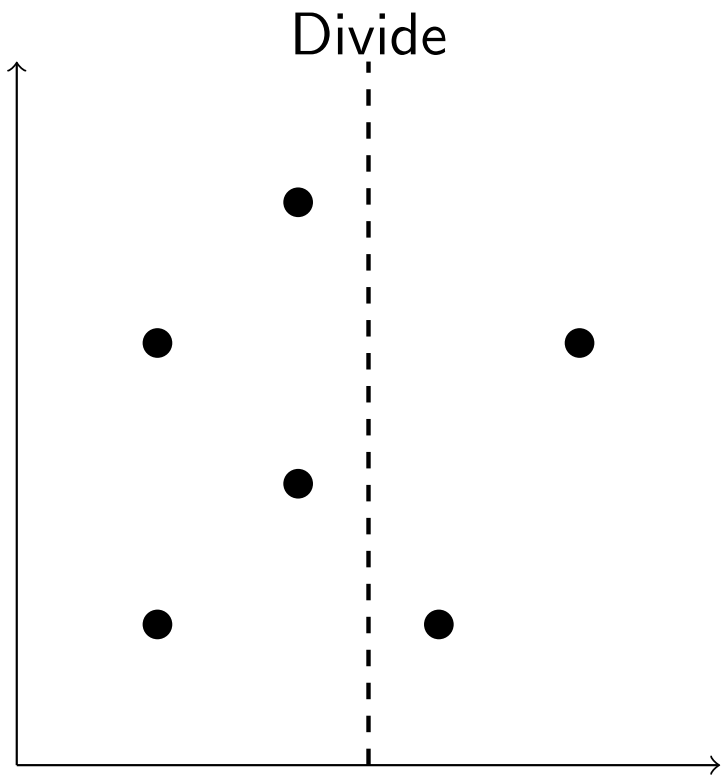
Upper and lower tangents merge the left and right hulls.

Divide and Conquer – Divide Step



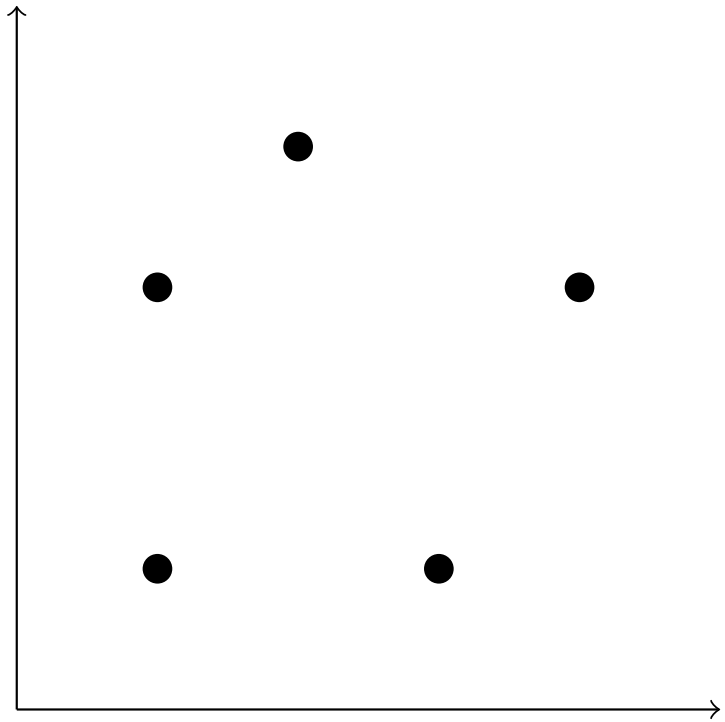
Initial set of points

Divide and Conquer – Divide Step



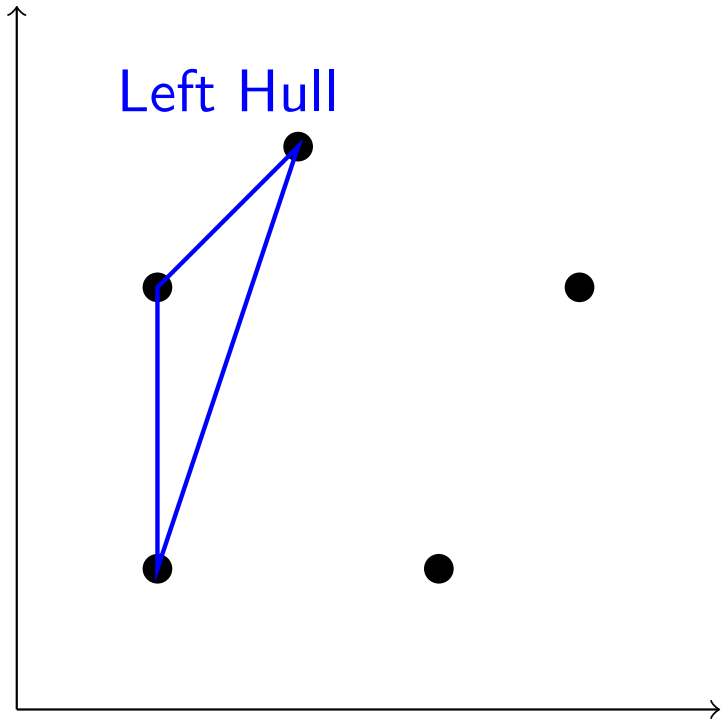
Points divided into left and right halves

Divide and Conquer – Recursive Hulls



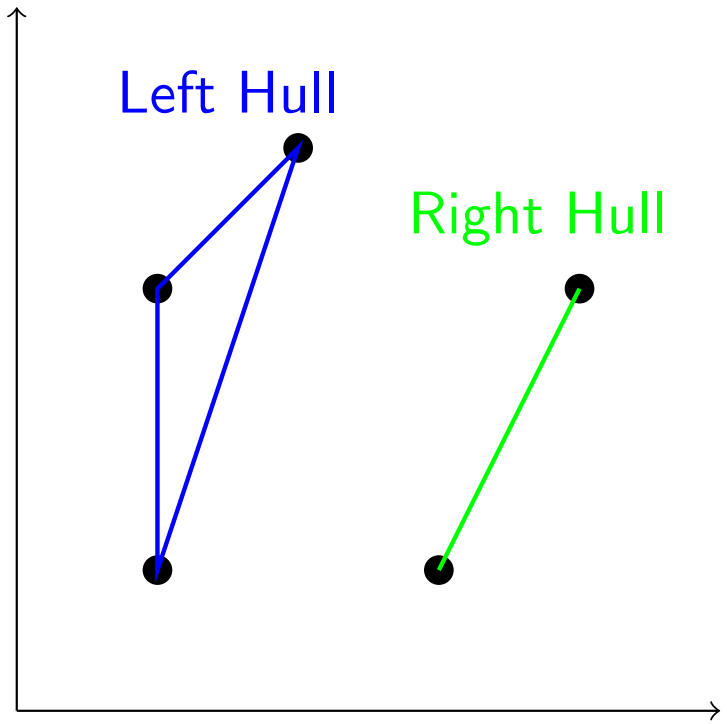
Recursive calls on each half

Divide and Conquer – Recursive Hulls



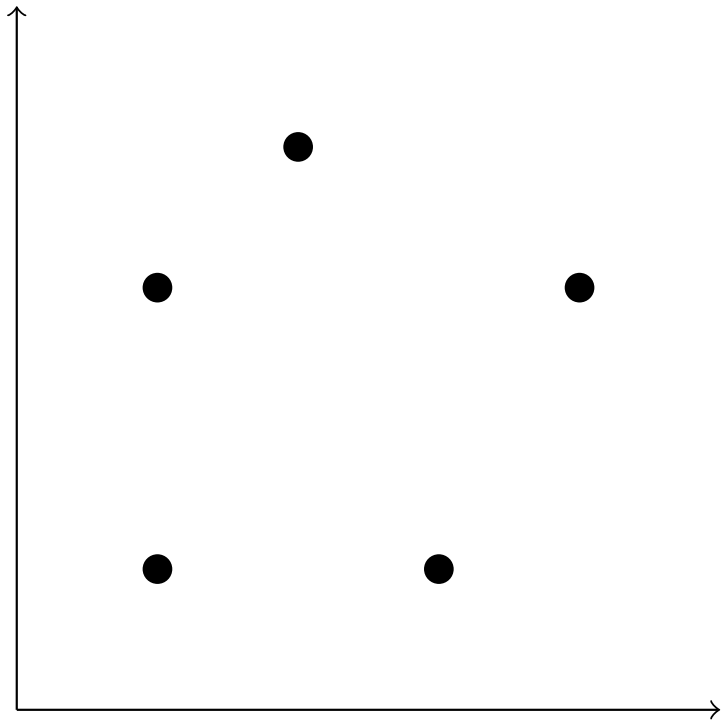
Convex hull of left subset

Divide and Conquer – Recursive Hulls



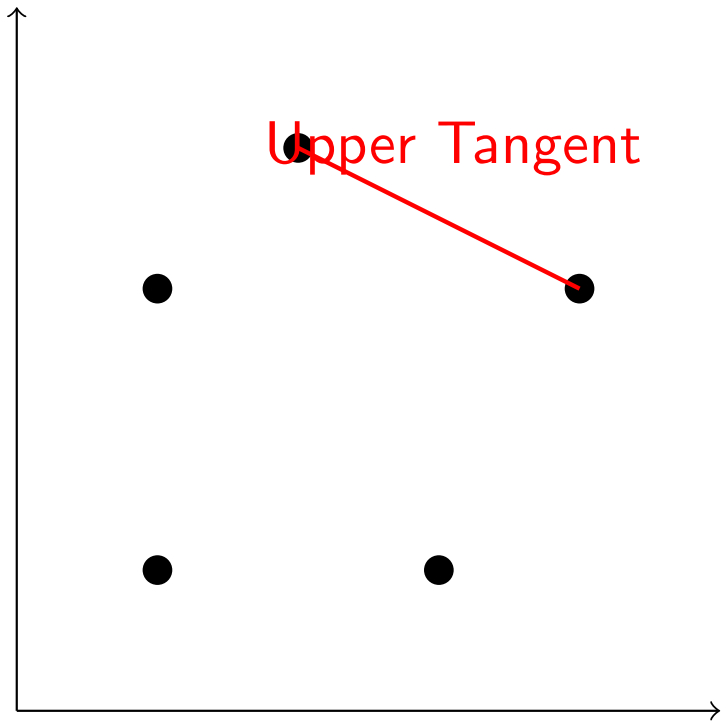
Convex hull of right subset

Divide and Conquer – Finding Upper Tangent



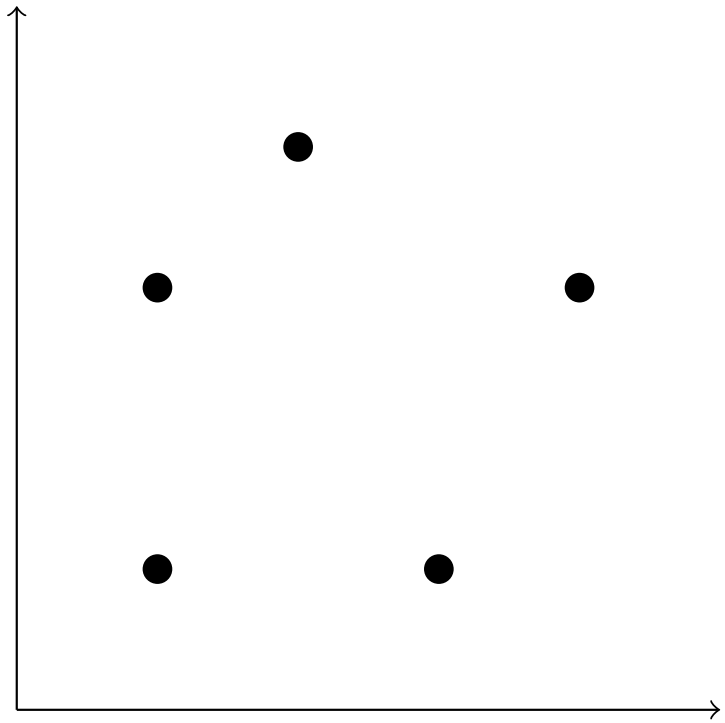
Searching for upper tangent

Divide and Conquer – Finding Upper Tangent



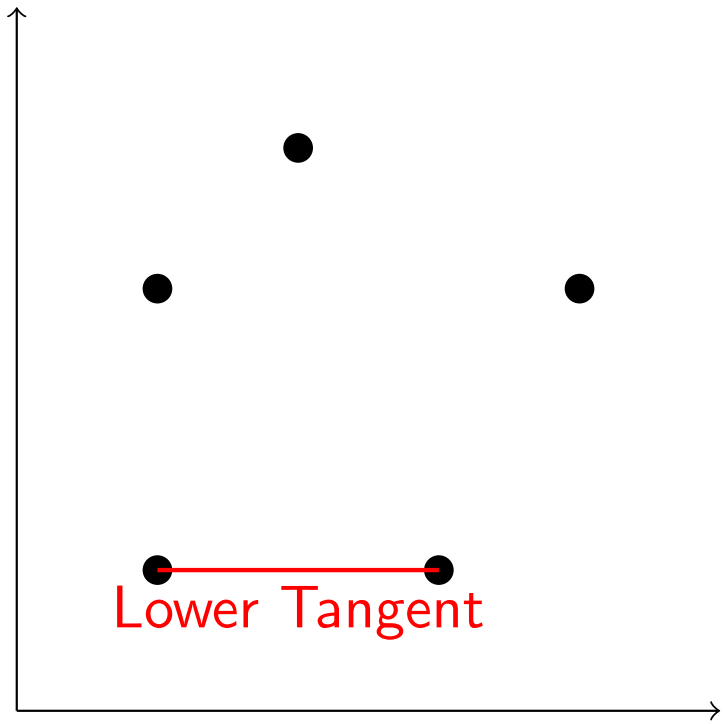
Upper tangent found

Divide and Conquer – Finding Lower Tangent



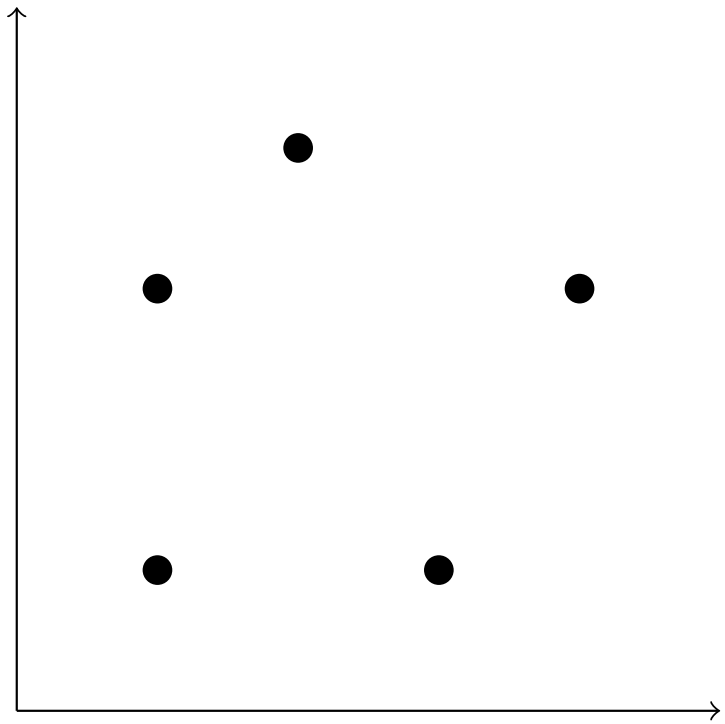
Searching for lower tangent

Divide and Conquer – Finding Lower Tangent



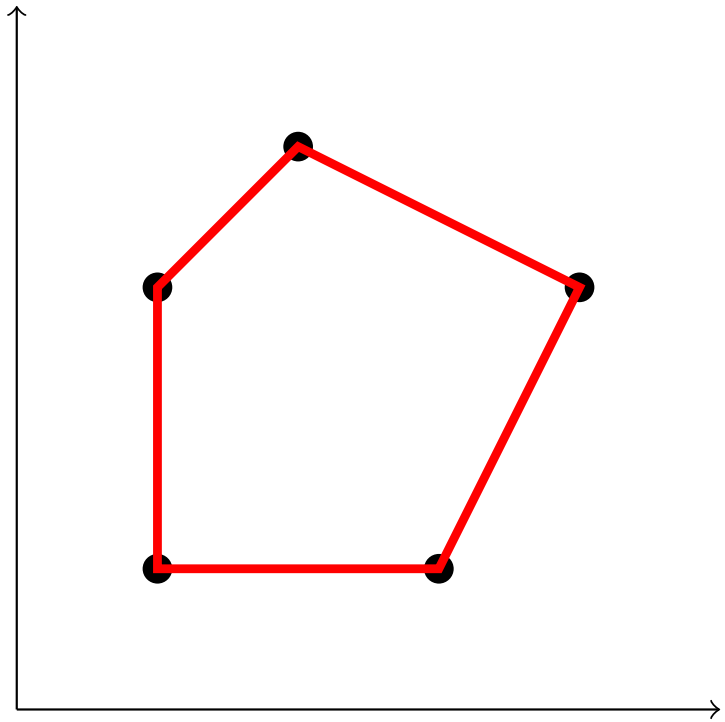
Lower tangent found

Divide and Conquer – Merge Step



Merging left and right hulls

Divide and Conquer – Merge Step



Final convex hull

Divide & Conquer Convex Hull – Recurrence Analysis

Let n be the number of input points.

Step 1: Divide

- Sort points by x-coordinate
- Divide the set into two halves of size $n/2$
- Cost of division (after sorting): $O(1)$

Step 2: Conquer

- Recursively compute convex hull for left half
- Recursively compute convex hull for right half
- This gives two subproblems of size $n/2$

Step 3: Merge

- Merge the two hulls by finding upper and lower tangents
- Tangent finding and merging takes linear time
- Merge Cost: $O(n)$

Recurrence Relation

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Using Master Theorem:

- $a = 2,\ b = 2,\ f(n) = O(n)$
- $n^{\log_b a} = n$

$$\boxed{T(n) = O(n \log n)}$$

QuickHull – Concept

QuickHull is inspired by QuickSort:

- Partition points using a baseline
- Select the farthest point
- Recursively process subsets

QuickHull – Algorithm Steps

1. Find leftmost and rightmost points A and B
2. Divide points into two sets (above and below AB)
3. Find point C farthest from AB
4. Discard points inside triangle ABC
5. Recursively apply QuickHull on remaining sets

QuickHull – Worked Example

Step 1: Leftmost and rightmost points:

$$A(1,1),\ B(4,3)$$

Step 2: Farthest point from line AB:

$$C(2,4)$$

Points inside triangle ABC are discarded.

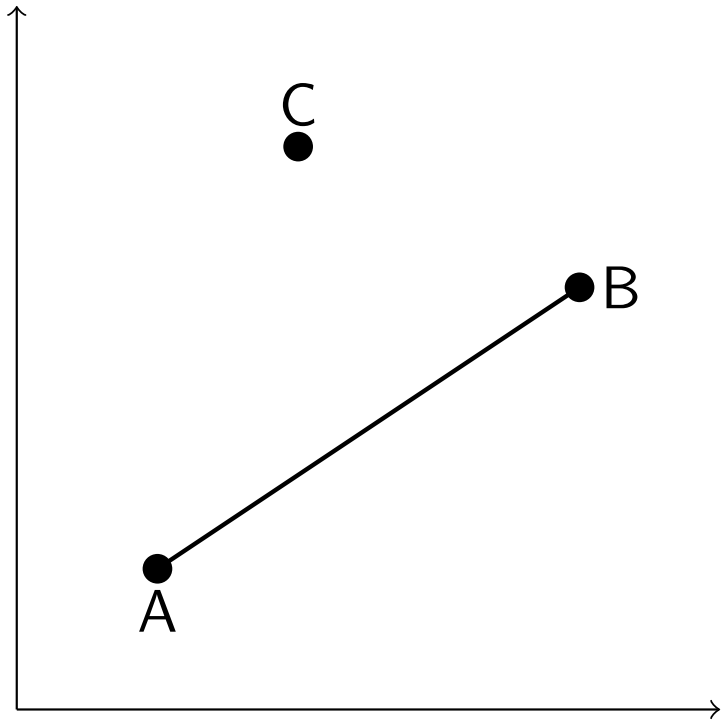
Recursive calls on:

$$A-C \quad \text{and} \quad C-B$$

Final Hull:

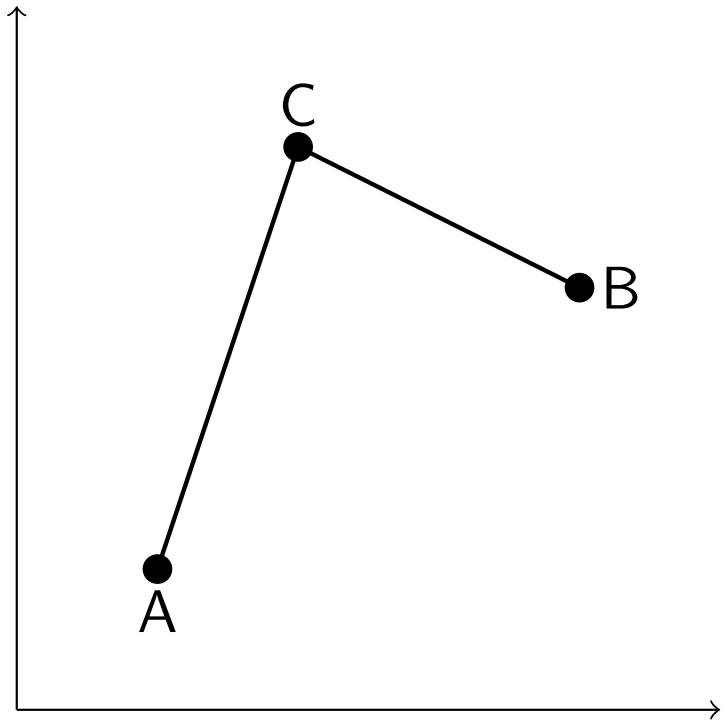
$$(1,1),(3,1),(4,3),(2,4),(1,3)$$

QuickHull – Animated Steps



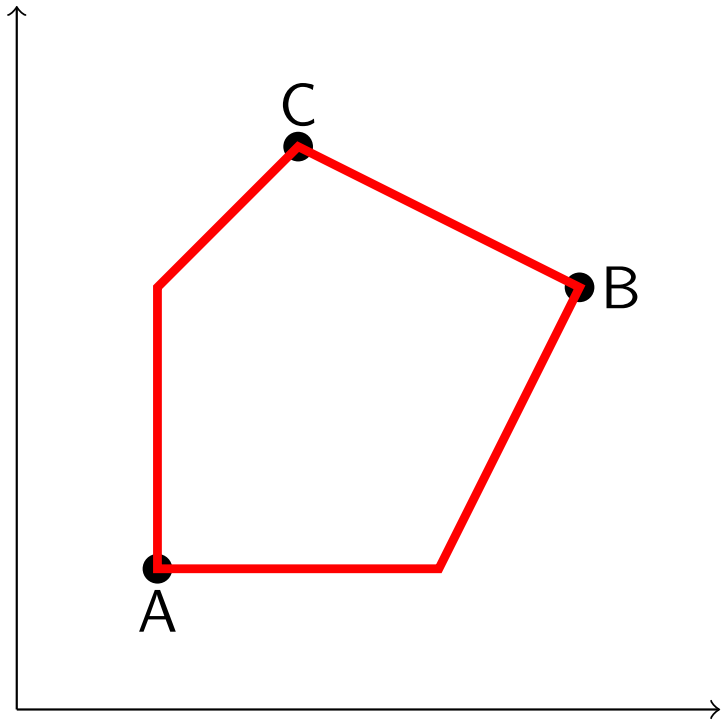
Initial baseline AB

QuickHull – Animated Steps



Farthest point selected

QuickHull – Animated Steps



Final convex hull

QuickHull – Time Complexity Analysis

Let n be the number of input points.

Step 1: Initial Partition

- Find leftmost and rightmost points A and B
- Partition remaining points into two sets (above and below line AB)
- Time Complexity: $O(n)$

Step 2: Farthest Point Selection

- Find the point farthest from the current line
- Distance computed using area (cross product)
- Time Complexity per recursion: $O(n)$

Step 3: Recursive Partitioning

- Recursively apply QuickHull on subsets formed by lines AC and CB
- Points inside the triangle are discarded

Average Case Analysis

- Points are fairly evenly distributed
- Each recursion splits the set roughly in half

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n) \Rightarrow \boxed{O(n \log n)}$$

Worst Case Analysis

- All points lie on the convex hull
- Only one point is eliminated per recursion

$$T(n) = T(n-1) + O(n) \Rightarrow \boxed{O(n^2)}$$

Algorithm Comparison

Algorithm	**Approach**	**Time Complexity**	**Space**
Naive (Brute Force)	Edge checking	$O(n^3)$	$O(1)$
Graham Scan	Sorting + Stack	$O(n \log n)$	$O(n)$
Jarvis March	Gift Wrapping	$O(nh)$	$O(h)$
Divide & Conquer	Recursive merge	$O(n \log n)$	$O(n)$
QuickHull	Partitioning	Avg: $O(n \log n)$Worst: $O(n^2)$	$O(n)$

Applications

- Computer Graphics & Image Processing
- Pattern Recognition & Machine Learning
- Geographic Information Systems (GIS) Mapping

Convex Hull in Computer Graphics & Image Processing

Problem

- Objects in images consist of thousands of pixels or vertices
- High computational cost for rendering and collision detection

Reasoning (Why Convex Hull?)

- Only extreme boundary points define the visible shape
- Interior points do not affect object interaction or visibility

How It Helps

- Represents objects using a minimal enclosing polygon
- Speeds up collision detection, rendering, and object tracking

Convex Hull in Pattern Recognition & Machine Learning

Problem

- Large, multi-dimensional datasets
- Difficulty in identifying class boundaries and anomalies

Reasoning (Why Convex Hull?)

- All valid data points of a class lie within the region formed by extreme samples
- Points outside the convex hull are statistically unlikely

How It Helps

- Defines class boundaries geometrically
- Detects outliers and visualizes cluster spread

Convex Hull in Geographic Information Systems (GIS)

Problem

- Geographic data (GPS points) are scattered and irregular
- Difficult to determine clear region boundaries

Reasoning (Why Convex Hull?)

- Convex hull provides the smallest region enclosing all spatial points
- Ensures complete spatial coverage without gaps

How It Helps

- Boundary mapping of cities, forests, and disaster zones
- Simplifies area and perimeter calculation

Conclusion

- Convex Hull finds the outer boundary of a set of points.
- All algorithms rely on the orientation test and cross product.
- Different strategies offer different time efficiencies.
- Algorithm choice depends on input size and hull complexity.
- Convex Hull is widely used in GIS, graphics, and collision detection.

Thank You