Divide and Conquer Approach for the Convex Hull Problem

1. Problem Statement

Given a set of n points in a two-dimensional plane, the **Convex Hull** problem is to find the smallest convex polygon that contains all the points.

Formally, for $P = \{p_1, p_2, \ldots, p_n\}$ with $p_i = (x_i, y_i)$, the convex hull $\mathrm{CH}(P)$ is defined as the minimal convex set containing all points of P.

2. Motivation

- Convex hull is a fundamental building block in computational geometry.
- Applications include pattern recognition, collision detection, image processing, GIS, and clustering.
- Naive algorithms run in $\mathcal{O}(n^3)$ or $\mathcal{O}(n^2)$ time.
- Divide and Conquer achieves $\mathcal{O}(n \log n)$ time, comparable to Graham Scan and Andrew's monotone chain.

3. Divide and Conquer Strategy

Step 1: Preprocessing

- Sort points by x-coordinate.

Step 2: Divide

- Split the point set P into two halves: Q (left) and R (right).

Step 3: Conquer

- Recursively compute convex hulls $CH(Q)$ and $CH(R)$.

Step 4: Combine (Merging Hulls)

- Merge $CH(Q)$ and $CH(R)$ into one convex hull.
- To do this, find the **upper tangent** and **lower tangent** that connect the two hulls without intersecting either polygon.
- Remove points between tangents and combine the hulls.

4. Pseudocode

```
function convexHullDivideAndConquer(P):
    sort P by x-coordinate
    return hullRecursive(P)

function hullRecursive(P):
    if |P| <= 5:
        return bruteForceHull(P)

    mid = |P|/2
    Q = left half of P
    R = right half of P

    hullQ = hullRecursive(Q)
    hullR = hullRecursive(R)

    return mergeHulls(hullQ, hullR)

function mergeHulls(hullQ, hullR):
    # Find tangents
    upper = findUpperTangent(hullQ, hullR)
    lower = findLowerTangent(hullQ, hullR)
    # Combine hulls along tangents
    return combine(hullQ, hullR, upper, lower)

function findUpperTangent(hullQ, hullR):
    i = rightmost point of hullQ
    j = leftmost point of hullR

    while True:
        # Move clockwise on hullQ if orientation is not correct
        while orientation(hullR[j], hullQ[i], hullQ[(i+1) mod |hullQ|]) >= 0:
            i = (i+1) mod |hullQ|
        # Move counter-clockwise on hullR if orientation is not correct
        while orientation(hullQ[i], hullR[j], hullR[(j-1) mod |hullR|]) <= 0:
            j = (j-1) mod |hullR|
        else:
            break
    return (i, j)
```

```
function findLowerTangent(hullQ, hullR):
    i = rightmost point of hullQ
    j = leftmost point of hullR

    while True:
        # Move counter-clockwise on hullQ if orientation is not correct
        while orientation(hullR[j], hullQ[i], hullQ[(i-1) mod |hullQ|]) <= 0:
            i = (i-1) mod |hullQ|
        # Move clockwise on hullR if orientation is not correct
        while orientation(hullQ[i], hullR[j], hullR[(j+1) mod |hullR|]) >= 0:
            j = (j+1) mod |hullR|
        else:
            break
    return (i, j)

function combine(hullQ, hullR, upper, lower):
    (i_upper, j_upper) = upper
    (i_lower, j_lower) = lower

    newHull = []

    # Traverse hullQ from i_upper to i_lower
    k = i_upper
    newHull.append(hullQ[k])
    while k != i_lower:
        k = (k+1) mod |hullQ|
        newHull.append(hullQ[k])

    # Traverse hullR from j_lower to j_upper
    k = j_lower
    newHull.append(hullR[k])
    while k != j_upper:
        k = (k+1) mod |hullR|
        newHull.append(hullR[k])

    return newHull
```

5. Finding Upper and Lower Tangents

We use an **orientation test** to decide rotations:

$$\text{orientation}(a, b, c) = (c - a) \times (b - a)$$

where

$$\text{orientation}(a, b, c) = \begin{cases} > 0 & \text{clockwise (CW)} \\ < 0 & \text{counter-clockwise (CCW)} \\ = 0 & \text{collinear} \end{cases}$$

Definitions of Notation

- $CH(Q)$: convex hull of the left subset Q (vertices stored in counterclockwise order).
- $CH(R)$: convex hull of the right subset R (also counterclockwise order).
- L: a candidate vertex on $CH(Q)$.

- R: a candidate vertex on $CH(R)$.
- L_{next}, L_{prev}: the next and previous vertices of L in the cyclic order of $CH(Q)$.
- R_{next}, R_{prev}: the next and previous vertices of R in the cyclic order of $CH(R)$.

Initially, L is chosen as the *rightmost point* of $CH(Q)$, and R as the *leftmost point* of $CH(R)$.

Upper Tangent

1. Start with L = rightmost point of $CH(Q)$ and R = leftmost point of $CH(R)$.
2. While orientation$(L, R, L_{\text{next}}) < 0$ (CCW), move L forward to L_{next}.
3. While orientation$(R, L, R_{\text{prev}}) > 0$ (CW), move R backward to R_{prev}.
4. Repeat until L and R stop changing. The final (L, R) defines the upper tangent.

Lower Tangent

1. Start again with L = rightmost point of $CH(Q)$ and R = leftmost point of $CH(R)$.
2. While orientation$(L, R, L_{\text{prev}}) > 0$ (CW), move L backward to L_{prev}.
3. While orientation$(R, L, R_{\text{next}}) < 0$ (CCW), move R forward to R_{next}.
4. Repeat until stable. The final (L, R) defines the lower tangent.

6. Worked Example

Consider the set of points:

$$P = \{(0,0), (1,1), (2,2), (3,1), (4,0), (2,-1)\}.$$

1. Sort by x: $[(0,0), (1,1), (2,2), (2,-1), (3,1), (4,0)]$.
2. Divide into $Q = \{(0,0), (1,1), (2,2)\}$ and $R = \{(2,-1), (3,1), (4,0)\}$.
3. Compute hulls recursively:
   - $CH(Q) = \{(0,0), (1,1), (2,2)\}$
   - $CH(R) = \{(2,-1), (3,1), (4,0)\}$
4. Find upper tangent: $(2,2)$ to $(3,1)$.
5. Find lower tangent: $(0,0)$ to $(2,-1)$.
6. Final hull: $\{(0,0), (2,-1), (4,0), (3,1), (2,2)\}$.

7. Correctness and Key Insight

- The recursive step ensures each subset is convex.
- The only possible missing edges after merging are those crossing between $CH(Q)$ and $CH(R)$.
- Upper and lower tangents guarantee the merged polygon is convex and includes all points.

8. Complexity Analysis

- Sorting: $\mathcal{O}(n \log n)$.
- Divide step: $\mathcal{O}(1)$.
- Conquer step: recurrence
$$T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n),$$
for merging hulls.
- By the Master Theorem: $T(n) = \mathcal{O}(n \log n)$.
- Space: $\mathcal{O}(n)$ for recursive calls and storing hulls.

9. Applications

- **Computer Graphics:** Collision detection and shape reconstruction.
- **Geographical Information Systems (GIS):** Determining boundaries of regions, territories, or clusters of points.
- **Robotics and Path Planning:** Obstacle avoidance using convex obstacle representations.
- **Machine Learning:** Used in clustering and support vector machines for boundary estimation.

10. Conclusion

The divide and conquer convex hull algorithm is an elegant alternative to Graham Scan and Andrew's algorithm. It achieves $\mathcal{O}(n \log n)$ time by recursively solving smaller hulls and merging them efficiently with tangent-based combination.