CONVEX HULL IMPLEMENTATION

a)Chan's Algorithm for Convex Hull

1 Introduction

This project implements **Chan's Algorithm** for computing the convex hull of a set of 2D points. The algorithm combines the advantages of Graham Scan and Jarvis March to achieve an optimal output-sensitive time complexity of $O(n \log h)$, where n is the number of input points and h is the number of points on the convex hull.

2 Overview of the Implementation

The implementation is done in Python and visualized using `matplotlib`. The main steps of the algorithm are as follows:

- Partition the input set of n points into groups of size at most m.
- Compute the convex hull of each group using Graham Scan.
- Merge the hulls using an enhanced version of Jarvis March by finding tangents via binary search.
- Repeat the process with an increased value of m (exponentially) until the complete convex hull is found.

3 Graham Scan for Sub-Hulls

The function `graham_scan` constructs the convex hull for a subset of points by maintaining the upper and lower hulls separately. It works in $O(m \log m)$ time per group, where m is the group size.

4 Binary Search for Tangent Finding

A critical step in Chan's Algorithm is efficiently finding the tangent from a point to a convex polygon. Instead of scanning the polygon linearly, this implementation uses a **binary search** approach to achieve $O(\log m)$ time per group.

Binary Search Procedure

- The polygon is assumed to be sorted in counter-clockwise (CCW) order.
- For a given external point p, we search for a point q on the polygon such that the line segment $\overline{pq}$ is tangent to the polygon.
- The function `is_tangent` checks whether a given vertex satisfies the tangent property by evaluating cross products with neighboring vertices.
- Binary search proceeds by comparing the cross product $\texttt{cross}(p, curr, next)$ to guide the search.

This approach reduces the tangent-finding step from $O(m)$ to $O(\log m)$, making the overall merging step efficient.

5 Main Algorithm: Chan's Hull Construction

- The `chan_hull` function orchestrates the process of computing group hulls and incrementally building the global convex hull.
- For each step, the best candidate point is selected based on the angle formed with the previous two points.
- If the hull closes (returns to the starting point) within m steps, the result is returned.
- Otherwise, m is increased exponentially and the process repeats in `full_chan_hull`.

6 Visualization

The program uses `matplotlib` to:

- Draw input points and label them.
- Show each group in a different color.
- Display group hulls and the final merged convex hull.

Output

The final convex hull is printed in clockwise (CW) order. The plot shows the hull with line segments and labeled points.

7 Key Functions and Logic

- `draw_points, draw_polygon`: For plotting.
- `find_angle, cross`: For geometric calculations.
- `binary_search, is_tangent`: Core of the tangent-finding mechanism.
- `graham_scan`: Computes convex hull for each group.
- `chan_hull, full_chan_hull`: Main algorithm functions.

Time Complexity

- Graham Scan per group: $O(m \log m)$.
- Binary Search for tangents: $O(\log m)$.
- Merging phase: $O(h \log m)$, repeated $\log \log n$ times.
- Overall: $O(n \log h)$.

8 Conclusion

This implementation of Chan's Algorithm uses binary search to optimize tangent finding, making the convex hull construction efficient for large datasets. The visualization helps in understanding the grouping, local hulls, and final merged result clearly.