Task 1: Convex Hull Computation with Doubly Linked Lists

Part A

Implement the **Jarvis' March** algorithm to compute the convex hull. Find the leftmost point. Then iterate over the remaining points, selecting the most counterclockwise point relative to the current point until you come back to the starting point.

**Requirements**

* **Data Structure**: Implement a **doubly linked list** to store the points of the convex hull. Each node should store the x and y coordinates of a point and pointers to the next and previous nodes.
* **Input Format**: The input will be a CSV file where the first line indicates the total number of points. Subsequent lines represent points with their x and y coordinates separated by a space (e.g., x y).
* **Output Format**: Your program should output two sequences of points forming the convex hull: **one in clockwise order and the other in counterclockwise order.** This should be done by **traversing your doubly linked list to print the points in both clockwise and counterclockwise order**, in both cases starting from the first point added to the hull. This may require starting from a specific point and moving in one direction for clockwise and the opposite direction for counterclockwise. This should be output to the console (stdout).

JarvisMarch(points):

// Ensure there are at least 3 points

if length(points) < 3:

return empty convex hull

// Initialize an empty list to store convex hull points

convexHull <- empty list

// Find the leftmost point (pivot) among the given points

leftmost <- leftmost\_point(points)

// Start from the leftmost point

current <- leftmost

repeat:

// Add current point to the convex hull

add current to convexHull

// Find the next point 'nextPoint' such that it forms a counterclockwise turn

// with the current point and any other point in the set

nextPoint <- points[0]

for each point in points:

if nextPoint = current or orientation(nextPoint, current, point) = counterclockwise:

nextPoint <- point

// Set 'nextPoint' as the current point for the next iteration

current <- nextPoint

// Repeat until we return to the starting point (leftmost)

until current = leftmost

// Return the list of points in the convex hull

return convexHull

Part B

Implement the **Graham's Scan** algorithm to compute the convex hull. Find the **bottom-most point**, sort the points, and proceed with the scan, pushing and popping points from the stack (which will be your doubly linked list in this case) to ensure you maintain a convex hull throughout.

**Requirements**

* **Data Structure**: Implement a stack with doubly linked lists. Utilize this stack in Graham's Scan algorithm to store the convex hull points. Each node should store the x and y coordinates of a point and pointers to the next and previous nodes.
* **Input Format**: The input will be a CSV file where the first line indicates the total number of points. Subsequent lines represent points with their x and y coordinates separated by a space (e.g., x y).
* **Output Format**: Your program should output two sequences of points forming the convex hull: one in clockwise order and the other in counterclockwise order. This should be done by traversing your doubly linked list to print the points in both clockwise and counterclockwise order, in both cases starting from the first point added to the hull. This may require starting from a specific point and moving in one direction for clockwise and the opposite direction for counterclockwise. This should be output to the console (stdout).

Part C

Evaluate both algorithms through experimental analysis by quantifying the total basic operations across various input scales and configurations. Consider creating input sets of at least three distinct sizes, each under three differing distribution conditions: random, points on a circle, and random points contained within a set of points making up a simple hull.

You should use the following basic operations for each algorithm:

* Jarvis' March - A comparison between the angles of points.
* Graham's Scan - A comparison between the angles of points during the sort.

**Reporting**: Write a report including a discussion on the choice of data structure, the experimental evaluation, and conclusions drawn from the comparisons. Include any assumptions or simplifications made in your implementations.

**Submission Guidelines**

* Submit your C source code files with appropriate comments explaining the algorithms and data structures used.
* Your report should be in PDF format, including your findings from the experimental evaluation and any observations regarding the performance of the two algorithms.

**Grading Criteria**

* Correctness of the implemented algorithms and adherence to the requirements.
* Efficiency and proper use of the doubly linked list for storing the convex hull.
* Clarity of the report, including the depth of the experimental evaluation and the analysis of the results.
* Code readability, structure, and documentation.