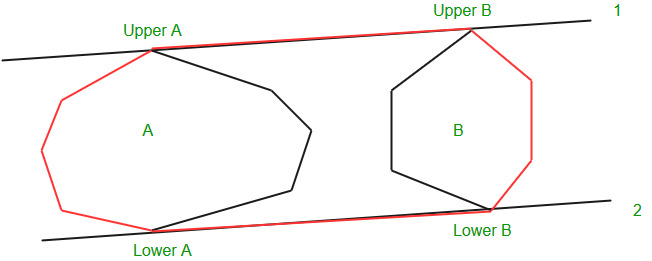
Prompt:

Design a divide-and-conquer algorithm for the convex hull problem by doing:

a) Let P1 and P2 be two disjoint convex polygons with n vertices in total. Give an O(n) time algorithm that computes the convex hull of P1 ∪ P2.

b) Use the algorithm from part (a) to develop an O(nlogn) time divide-and-conquer algorithm to compute the convex hull of a set of n points in the plane.



A.) Imagining that there are two disjoint convex poloygons that when connected can form the convex hull. Together they have N verticies. These polygons can be seen as the convex hull of the left and right points.

P1 = points on the left of the midpoint

P2 = points on the right of the midpoint

Pseudo Code:

LowerTangent()

L <- line joining the rightmost point of a and leftmost point of b

While(L crosses P1 or P2)

{

While (L crosses b)

L <- L’: point on B moves down

While(L crosses a)

L<-L’ : point on A moves down

UpperTangent()

U <- line joining the rightmost point of a and leftmost point of b

While(U crosses P1 or P2)

{

While (U crosses b)

U <- U’: point on B moves up

While(L crosses a)

U<-U’ : point on A moves up

makers(pt1,pt2,pt3)

               return (pt2.y-pt1.y)\*(pt3.x-pt2.X)-(pt3.y-pt2.y)\*(pt2.x-pt1.x)

// moveRightWhenUpperLeftWhenLower means that when we are looking for an upper tangent we move the point we // are looking at on the right hull up and we move the point on the left hull down WHEN we are looking

// for the lowerTangent

orientation(pt1, pt2,pt3)

               If makers(pt1,pt2,pt3) ç 0

                              Then moveRightWhenUpperLeftWhenLower ç True and

moveLeftWhenUpperRightWhenLower ç True

               Else If makers(pt1,pt2,pt3) is bigger than 0

                              Then moveLeftWhenUpperRightWhenLower ç True

                              Then moveRightWhenUpperLeftWhenLower ç False

               Else

                              Then moveRightWhenUpperLeftWhenLower ç True

                              And moveLeftWhenUpperRightWhenLower ç False

Return moveLeftWhenUpperRightWhenLower and moveRightWhenUpperLeftWhenLower

mergeTwoHulls(pt1,pt2,pt3)

               // get sizes of left and right hulls

              Size\_P1 ç size of P1

              Size\_P2 ç size of P2

P1\_starting\_point = 0

P2\_starting\_point = 0

for I from 0 to Size\_P1

if pt1[i].x is smaller than pt1[P1\_starting\_point].x

P1\_starting\_point = i

for I from 0 to Size\_P2

if pt2[i].x is smaller than pt2[P2\_starting\_point].x

P2\_starting\_point = i

// find upper tangent. That’s what these points are for.

indentP1 = P1\_starting\_point

indentP2 = P2\_starting\_point

// traverse the hulls looking for the points for the two uppers.

While orientation(pt2[indentP2],pt1[indentP1],pt1[rem((indentP1+1)/Size\_P1))] == moveLeftWhenUpperRightWhenLower

indentP1 = rem((indentP1+1)/Size\_P1))

While orientation(pt2[indentP2],pt1[indentP1],pt2[rem((Size\_P2 + indentP2-1)/Size\_P2))] == moveRightWhenUpperLeftWhenLower

indentP2 = rem((indentP1+1)/Size\_P1))

upperP1 = indentP1

upperP2 = indentP2

indentP1 = P1\_starting\_point

indentP2 = P2\_starting\_point

//Find the lower hull

While orientation(pt1[indentP1],pt2[indentP2],pt2[rem((indentP2+1)/Size\_P1))] == moveLeftWhenUpperRightWhenLower

indentP2 = rem((indentP2+1)/Size\_P2))

While orientation(pt2[indentP2],pt1[indentP1],pt1[rem((Size\_P1 + indentP1-1)/Size\_P2))] == moveRightWhenUpperLeftWhenLower

indentP1 = rem((indentP1-1)/Size\_P1))

lowerP1 = indentP1

lowerP2 = indentP2

indent = lowerP1

// binary search for the

while indent is not upperP1

add pt1[indent] to result

indent = floor of (indent+1)/Size\_P1

indent = lowerP2

while indent is not upperP2

add pt[indent] to result

indent = floor of (indent+1)/Size\_P2

return result