```
QuickHull (A):
 // A[0..n-1] set of points of size n
 // all points lie in positive X-Y axis
  Sort A in ascending order of x co-ordinatines, break ties with ascending y co-ordinates
  hull ← {A[0], A[n-1]}
  leftMost = A[0], rightMost = A[n-1]
  remove leftMost and rightMost points from A

  // Divide A into two subarrays with points on top half and points of bottom half of
  // the line formed by A[0] and A[n-1]
  topPoints ← {}, bottomPoints ← {}
  for each i ← 0 to n-2 do
      if orientation of A[i] on top side of leftMost and rightMost points
          topPoints ← topPoints ∪ {A[i]}
      else if orientation of A[i] on bottom side of leftMost and rightMost points
          bottomPoints ← bottomPoints ∪ {A[i]}
      else : skip if collinear orientation

  partition (topPoints, leftMost, rightMost, hull)
  partition (bottomPoints, leftMost, rightMost, hull)

partition(points, left, right, hull)
  farthest_pt = NIL, farthest_dist = 0
  for i ← left to right do
      if distance of points[i] from line formed by points[left] and points[right]
          farthest_pt = points[i], farthest_dist = dist of points[i] to line(points[left], points[right]

  hull ← hull ∪ {furthest_pt}
  remove all points inside and on the edges of Δ(points[left], points[right], furthest_pt)
  left_to_farthest_points ← {}, farthest_to_right_points ← {}
  for i ← 0 to # of remaining points
      if points[i] is between line(points[left], farthest_point)
          left_to_farthest_points ← left_to_farthest_points ∪ {points[i]}
      else
          farthest_to_right_points ← farthest_to_right_points ∪ {points[i]}

  partition(left_to_farthest_points, left, farthest_pt, hull)
  partition(farthest_to_right_points, farthest_pt, right, hull)
```