Lab Convex Hull

3 December 2025

ASSUMPTIONS AND NOTES:

No three points in the input will be collinear.

At most one point on each side of the bounding box (so findWestmostPoint etc have a unique answer). Doesn't prevent points being in the corners!

A polygon should be a list of Points, in counter-clockwise order, which is a closed polygon if the list starts and ends on the same Point.

When writing big-O analysis: To represent total points in input, use N (e.g. O(N)). To represent number of points in correct result, which might be as big as N or might be much smaller, use H (e.g. O(H)). For the size of temporarily or partially constructed polygons, use P.

HELPER FUNCTIONS:

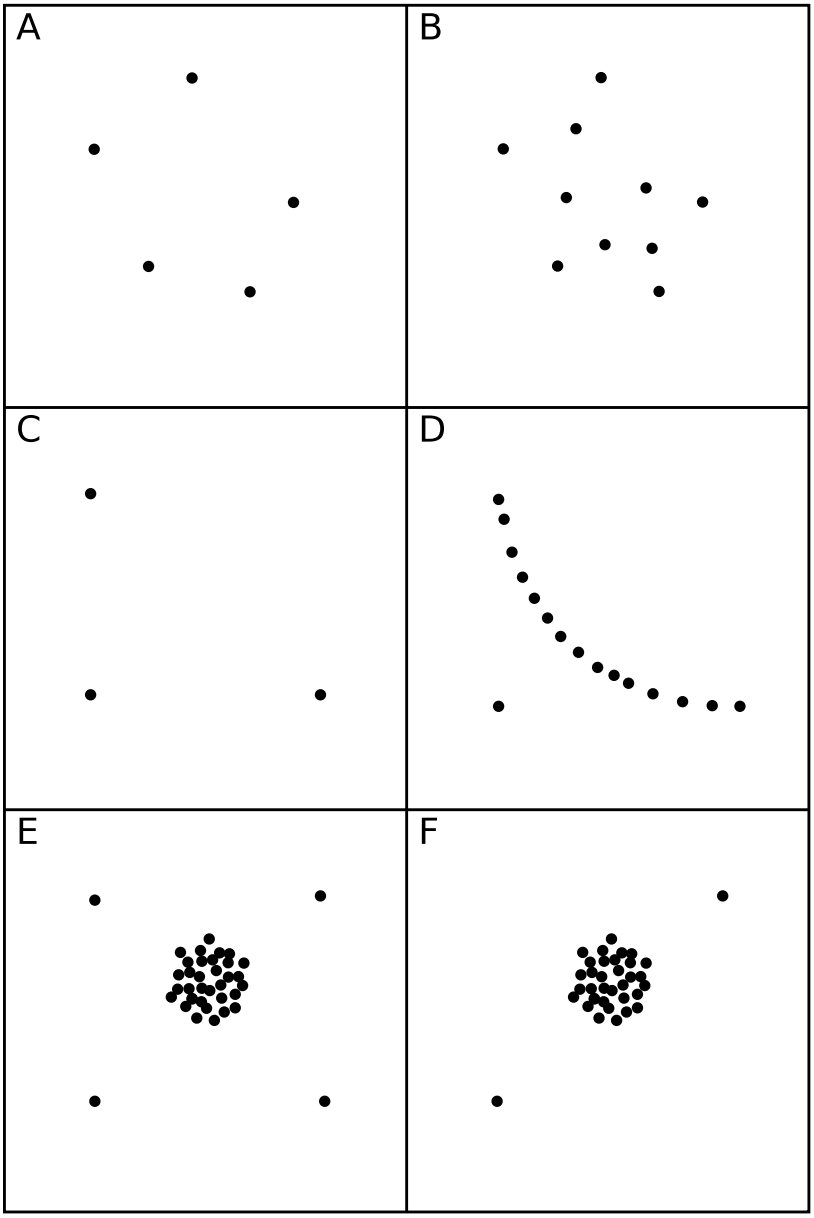
```
function findWestmostPoint (Set<Point> input) : Point
    (* definition not shown *)
end function

(* similarly findEastmostPoint, findNorthmostPoint, findSouthmostPoint *)

function ccw (Point p1, Point p2, Point p3) : enum(LEFT, COLLINEAR, RIGHT)
    ccwval := p1.x × p2.y + p2.x × p3.y + p3.x × p1.y
                          − p1.y × p2.x − p2.y × p3.x − p3.y × p1.x
    if ccwval < 0 return RIGHT
    else if ccwval = 0 return COLLINEAR
    else (* ccwval > 0 *) return LEFT
end function

function polygonContains (List<Point> polygon, Point point) : boolean
    for i := 0 to polygon.length − 2
        if polygon[i] = point
            return true
        if ccw(polygon[i], polygon[i+1], point) = RIGHT
            return false
    return true
end function
```

SOME USEFUL TEST INPUTS:



VERSION 1:

```
function convexHull (Set<Point> input) : List<Point>
    segList := new List<Pair<Point,Point>>

    foreach point1 in input
        foreach point2 in input
            found := true
            foreach point3 in input
                if ccw(point1, point2, point3) = RIGHT
                    found := false
            end for
            if found
                segList.add (pair(point1, point2));
            end if
        end for
    end for

    hull := new List<Point>

    start := segList[0].first
    hull.add(start)
    current := start

    repeat
        foreach segment in L
            if segment.first = current
                current := segment.second
                hull.add(current)
                break for
            end if
        end for
    until current = start

    return hull
end function
```

VERSION 2:

```
function convexHull (Set<Point> input) : List<Point>
    start := findNorthmostPoint(input) // or any extremum
    hull := new List<Point>
    hull.add(start)

    cur := start
    do
        foreach x in input
            bad := false
            next := x
            foreach y in input
                if ccw(cur, next, y) = RIGHT
                    bad := true
            end for
            if not bad
                hull.add(next)
                cur := next
            end if
        end for
    while hull.start ≠ hull.last

    return hull
end function
```

VERSION 3:

```
function convexHull (Set<Point> input) : List<Point>
    north, west, south, east := findNorthmostPoint(input), findWestmostPoint(input),
                                findSouthmostPoint(input), findEastmostPoint(input)
    c_hull := build_quadrant(north, west, input) )
    c_hull.removeLast()
    c_hull.addAll( build_quadrant(west, south, input) )
    c_hull.removeLast()
    c_hull.addAll( build_quadrant(south, east, input) )
    c_hull.removeLast()
    c_hull.addAll( build_quadrant(east, north, input) )
    return c_hull
end function

(* precondition: first, last are on the hull *)
function build_quadrant(Point first, Point last, Set<Point> input) : List<Point>
    c_hull = new List<Point>( one )
    if first = last
        return c_hull
    end if
    cur := first
    maybe_outside = find_outside(first, last, input)
    repeat
        cur := find_next (cur, last, maybe_outside)
        c_hull.add(cur)
    until cur = west
    return c_hull
end function

(* precondition: one is on the hull, one ≠ two, two is "ccw" of one *)
function find_next (Point one, Point two, Set<Point> pts) : Point
    outs = find_outside(one, two, pts)
    if outs.size = 0
        return two
    else
        return find_next(one, outs.grab_any(), outs)
end function

function find_outside (Point one, Point two, Set<Point> pts) : Set<Point>
    result = new Set
    foreach pt in pts
        if ccw(one, two, pt) = RIGHT
            result.add(pt)
    end foreach
    return result
end function
```

VERSION 4:

```
function convexHull (Set<Point> input) : List<Point>
    A := input.sort(from north to south)
    B := input.sort(from south to north)

    repeat
        sorted := innerEdit(A)
    until sorted
    repeat
        sorted := innerEdit(B)
    until sorted

    hull := copy(A)
    hull.removeLast()
    hull.addAll(B)
    return hull
end function


(* precondition: first and last element of P are on hull *)
function innerEdit (var List<Point> P) : List<Point>
    for i := 1 to P.size − 1
        prev := P[i−1]
        cur := P[i]
        next := P[i+1]

        direction := ccw(prev, cur, next)
        if direction ≠ LEFT
            A.remove(i)
            return false
        end if
    end for

    return true
end function
```

VERSION 5:

```
function convexHull (Set<Point> input) : List<Point>
    if (input size <= 3)
    then
        polygon := new List<Point>
        polygon.addAll (input)
        polygon.add (polygon.firstElement()) (* close the polygon *)
        if (input size = 3 and ccw (polygon[0], polygon[1], polygon[2]) = RIGHT)
            polygon.reverse() (* Polygon is now in ccw order *)
    else
        pt1, pt2 := any two elements of input
        set1, set2 := partition (pt1, pt2)
        poly1 := convexHull (set1)
        poly2 := convexHull (set2)
        polygon := merge (set1, List(pt1, pt2, pt1))
        polygon := merge (set2, polygon)
    end if
    return polygon

    function partition (Point pt1, Point pt2) : Set<Point>,Set<Point>
        left := new Set<Point>
        right := new Set<Point>
        foreach point in input
            if (point = pt1 or point = pt2)
            then continue
            else if (ccw (pt1, pt2, point) = LEFT)
                left.add(point)
            else
                right.add(point)
            end if
        end foreach
        return left, right
    end function

    function merge (List<Point> poly1, List<Point> poly2) : List<Point>
        leftPt := findWestmostPoint(poly1)
        rightPt := findEastmostPoint(poly2)
        if leftPt.x > rightPt.x
        then return merge(poly2, poy1)
        else
            botTanStart, botTanEnd := findTangent (leftPt, rightPt) (* this is O(n) *)
            topTanStart, topTanEnd := findTangent (rightPt, leftPt)
            return concatenate(poly1.sublist(topTanEnd, botTanStart)
                               poly2.sublist(botTanEnd, topTanStart)
                               poly1.sublist(topTanEnd))
        end if
    end function
end function
```

VERSION 6:

```
function convexHull (Set<Point> input) : List<Point>
    polygon := new List<Point>
    polygon.add (findWestmostPoint(input)) // or any extremum
    polygon.add (any two other elements of input)
    polygon.add (polygon.firstElement()) (* close the polygon *)
    if (ccw (polygon[0], polygon[1], polygon[2]) = RIGHT)
        polygon.reverse()
    (* Polygon is now triangle in ccw order *)

    procedure addToConvexPolygon (var List<Point> polygon, Point pointToAdd)
    (* Precondition: first/last elt of polygon is on convex hull of input *)
        for i := 0 to polygon.length − 1
            if ((i = 0 or ccw(polygon[i−1], polygon[i], pointToAdd) = LEFT)
               and ccw(polygon[i], pointToAdd, polygon[i+1]) = LEFT)
            then
                polygon.insert(pointToAdd, i+1)
                while i > 0
                      and ccw(polygon[i−1], polygon[i], polygon[i+1]) = RIGHT
                    polygon.remove(i)
                    i := i − 1
            end if
    end procedure

    foreach point in input
        if (not polygonContains (polygon, point))
            addToConvexPolygon(polygon, point)

    return polygon
end function
```

VERSION 7:

```
function convexHull (Set<Point> input) : List<Point>
    start := findWestmostPoint(input) // or any extremum

    allpts := new Deque<Point>
    allpts.addAll (input)
    allpts.remove (start)

    compare := new ccwcomp(start)
    allpts.sort(compare) // radial sort

    allpts.addFirst (start)
    allpts.addLast (start)

    hull := new LinkedList<Point> // permits O(1) iterator−based insertion/removal
    hull.addAll(allpts)

    current := hull.iterator()
    while current has at least two after it
        if ccw(current.value, current.next.value, current.next.next.value) = RIGHT
            current.next.remove()
            if current has at least one before it
                current := current.prev
            end if
        else
            current := current.next
        end if
    end while

    return hull
end function

class ccwcomp(Point pt1)
    function compare(Point pt2, Point pt3) : boolean
        return ccw(pt1, pt2, pt3)
    end function
end class
```