

Lab Convex Hull

4 December 2024

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
```

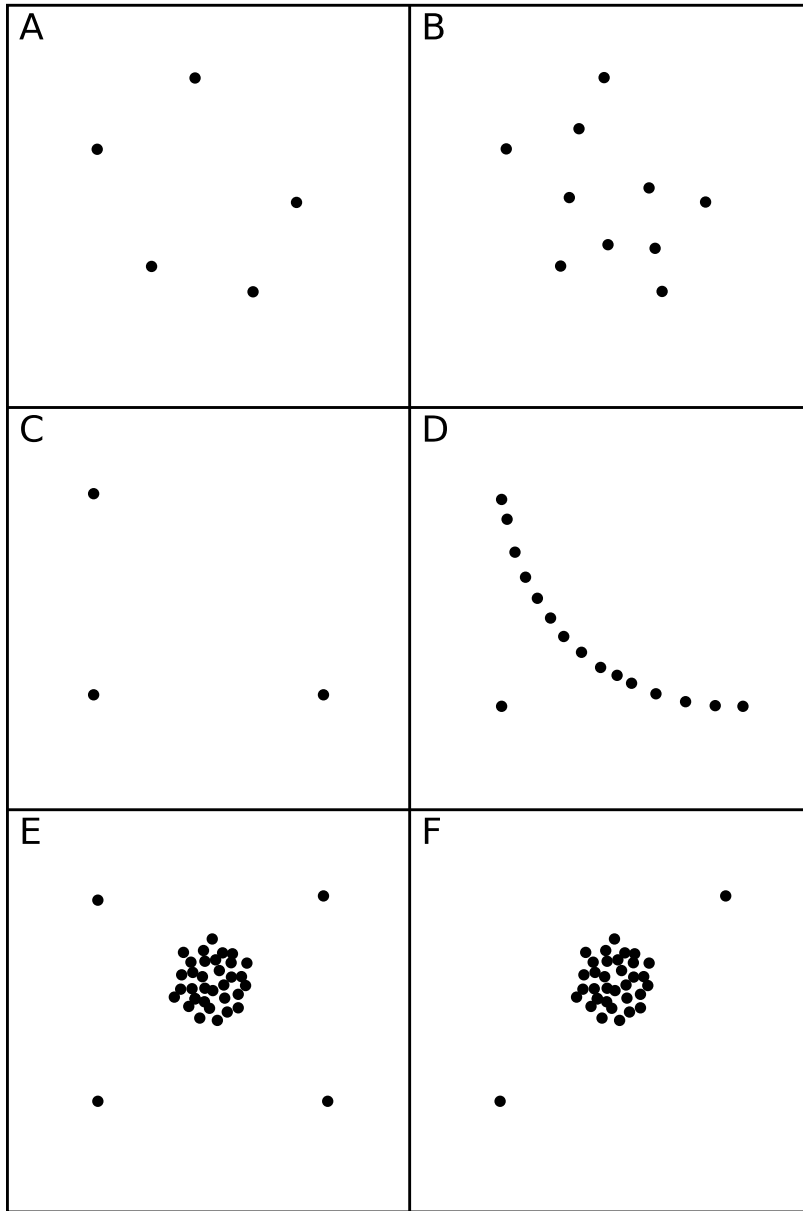
```
        if ccw(polygon[i], polygon[i+1], point) = RIGHT
```

```
            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 := findSouthmostPoint(input) // or any extremum
  polygon := new List<Point>
  polygon.add(start)

  current := start
  do
    foreach p2 in input
      checkHull := true
      foreach p3 in input
        if ccw(current, p2, p3) = RIGHT
          checkHull := false
        end for
      if checkHull
        polygon.add(p2)
        current := p2
      end if
    end for
  until current = start

  return polygon
end function
```

VERSION 3:

```

function convexHull (Set<Point> input) : List<Point>
  hull := new List<Point>
  hull.add(findWestmostPoint(input))
  hull.add(findSouthmostPoint(input))
  hull.add(findEastmostPoint(input))
  hull.add(findNorthmostPoint(input))
  hull.removeDups()
  (* hull has 2-4 items *)
  hull.add(hull[0]) // close the polygon

  maybeOutside = new Set<Point>
  foreach pt in input // possible optimisation [[[ O(N) times) ]]]
    if not polygonContains(hull, pt)
      maybeOutside.add(pt)
  end foreach

  i := 1

  while i < hull.size - 1
    added := false
    foreach pt in maybeOutside
      if (ccw (hull[i-1], pt, hull[i]) = LEFT)
        hull.insert(pt, i)
        (* but adjacent points may no longer be on hull *)
        while (i + 2 < hull.length
          and ccw(E, hull[i+1], hull[i+2]) = RIGHT)
          hull.remove(i+1)
        while (i - 2 >= 0
          and ccw(hull[i-2], hull[i-1], E) = RIGHT)
          hull.remove(i-1)
          i := i - 1
        added := true
        break foreach
    end foreach
    if not added
      i := i + 1
    end if
  end while

  return hull
end function

```

VERSION 4:

```
function convexHull (Set<Point> input) : List<Point>
  west := findWestmostPoint(input)
  east := findEastmostPoint(input)

  tophull := bothull := [ west, east ]

  tophull.sort(westward order)
  bothull.sort(eastward order)

  bothull.removeFirst()
  hull := new LinkedList<Point> // permits O(1) iterator-based insertion/removal
  hull.addAll(tophull)
  hull.addAll(bothull)

  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
```

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, poly1)
  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

```