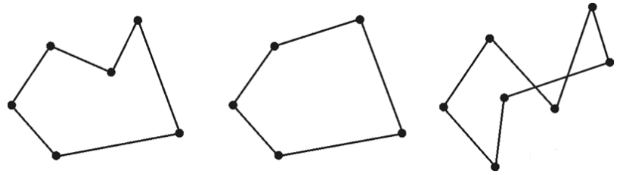
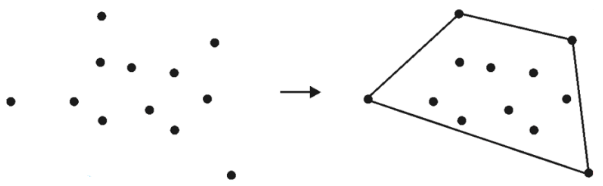
Convex Hull using Divide and Conquer

Convex hull is the smallest region covering given set of points. Polygon is called **convex polygon** if the angle between any of its two adjacent edges is always less than 1800. Otherwise, it is called a concave polygon. Complex polygons are self-intersecting polygons.



(a)Concave polygon (b) Convex polygon (c) Complex polygon

The convex hull of the set of points Q is the convex polygon P that encompasses all of the points given. The problem of finding the smallest polygon P such that all the points of set Q are either on the boundary of P or inside P is known as the convex hull problem. The convex hull of the points in question is seen in following figure. The vertex of a polygon is a point shared by two neighboring edges.



Convex hull of given points

**Divide and Conquer Approach**

* Sort all of the points by their X coordinates. The tie is broken by ranking points according to their Y coordinate.
* Determine two extreme points A and B, where A represents the leftmost point and B represents the rightmost point. A and B would be the convex hull’s vertices. Lines AB and BA should be added to the solution set.
* Find the point C that is the farthest away from line AB.
* Calculate the convex hull of the points on the line AC’s right and left sides. Remove line AB from the original solution set and replace it with AC and CB.
* Process the points on the right side of line BA in the same way.
* Find the convex hull of the points on the left and right of the line connecting the two farthest points of that specific convex hull recursively.

**Algorithm of Convex Hull**

Algorithm for finding convex hull using divide and conquer strategy is provided below:

Algorithm ConvexHull(P)

// P is a set of input points

Sort all the points in P and find two extreme points A and B

S1 ← Set of points right to the line AB

S2 ← Set of points right to the line BA

Solution ← AB followed by BA

Call FindHull(S1, A, B)

Call FindHull(S2, B, A)

**Algorithm for subroutine FindHull is described below :**

Algorithm FindHull(P, A, B)

if isEmpty(P) then

return

else

C ← Orthogonally farthest point from AB

Solution ← Replace AB by AC followed by CB

Partition P – { C } in X0, X1 and X2

Discard X0 in side triangle

Call FindHull(X1, A, C)

Call FindHull(X2, C, B)

end

**Complexity analysis**

Pre-processing step is to sort the points according to increasing order of their X coordinate. Sorting can be done in O(nlog2n) time. Finding two farthest points from the sorted list takes O(1) time. Dividing points into two halves S1 and S2 take O(1) time by joining A and B. In the average case, S1 and S2 contain half of the points. So, recursively computing the convex hull of A and B takes T(n/2) each. Merging of two convex hulls is done in linear time O(n), by finding the orthogonally farthest point. Total running time after preprocessing the points is given by,

T(n) = 2T(n/2) + O(n) + O(1)

= 2T(n/2) + n … (1)

Solving original recurrence for n/2,

T(n/2) = 2T(n/4) + n/2

Substituting this in equation (1),

T(n) = 2[ 2T(n/4) + n/2 ] + n

= 22 T(n/22) + 2n

.

.

.

After k substitutions,

T(n) = 2k T(n/2k) + k.n … (2)

Division of array creates binary tree, which has height log2n, so let us consider that k grows up to log2n,

k = log2n ⇒ n = 2k

Substitute these values in equation (2)

T(n) = n.T(n/n) + log2n . n

T(n) = O(n.log2n)

**Example of Convex Hull**

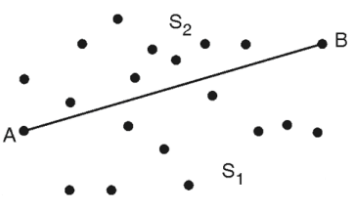
**Problem: Find the convex hull for a given set of points using divide and conquer approach.**



Find convex hull for these points

**Solution:**

**Step 1:** According to the algorithm, find left most and rightmost points from the set P and label them as A and B. Label all the points on the right of AB as S1 and all the points on the right of BA as S2.

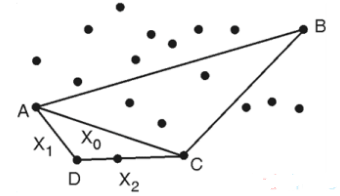


Solution = {AB, BA}

Make a recursive call to FindHull (S1, A, B) and FindHull(S2 ,B, A)

**Step 2 :** FindHull(S1 ,A, B)

Find point C orthogonally farthest from line AB



Solution = Solution – { AB } ∪ {AC, CB}

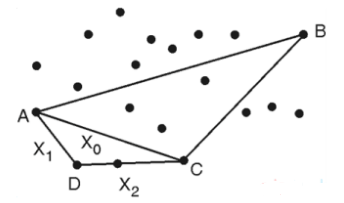
= {AC, CB, BA}

Label regions X0, X1 and X2 as shown in above figure

Make recursive calls: FindHull (X1, A, C) and FindHull (X2, C, B)

**Step 3 :** FindHull(X1, A, C)

Find point D orthogonally farthest from line AC



Solution = Solution – {AC} ∪ {AD, DC}

= {AD, DC, CB, BA}

Label regions X0, X1 and X2 as shown in above figure

Make recursive calls: FindHull (X1, A, D) and FindHull (X2, D, C)

But X1 and X2 sets are empty, so algorithm returns

**Step 4 :** FindHull(X2, C, B)

Find point E orthogonally farthest from line CB

Solution = Solution – {CB} ∪ {CE, EB}

= {AD, DC, CE, EB, BA}

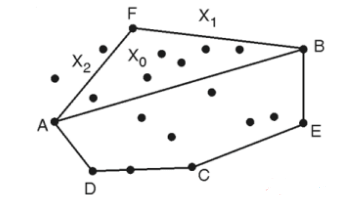
Label regions X0, X1 and X2 as shown in Fig. P.3.6.1(d).

Make recursive calls: FindHull (X1, C, E) and FindHull (X2, E, B).

But X1 and X2 sets are empty, so algorithm returns Now we will explore the points in S2, on the right-hand side of the line BA

**Step 5 :** FindHull(S2 ,B, A)

Find point F orthogonally farthest from line BA



Solution = Solution – {BA} ∪ {BF, FA}

= {AD, DC, CE, EB, BF, FA}

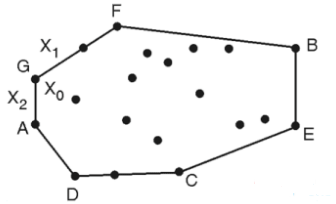
Label regions X0, X1 and X2 as shown in above figure.

Make recursive calls : FindHull (X1, B, F) and FindHull (X2, F, A)

But X1 set is empty, so call to FindHull (X1, B, F) returns

**Step 6 :** FindHull (X2, F, A)

Find point G orthogonally farthest from line FA



Solution = Solution – {FA} ∪ {FG, GA}

= {AD, DC, CE, EB, BF, FG, GA}

Label regions X0,

Collision avoidance: If an automobile’s convex hull avoids collisions with obstructions, so does the car. Because calculating collision-free routes is considerably easier with a convex vehicle, it is frequently used to design paths.

The lowest area rectangle that encloses a polygon has at least one side flush with the polygon’s convex hull, and hence the hull is computed in the first step of minimum rectangle methods. Finding the smallest three-dimensional box enclosing an item is also dependent on the 3D-convex hull.

X1 and X2 as shown in last figure

Make recursive calls: FindHull (X1, F, G) and FindHull (X2, G, A).

But X1 and X2 sets are empty, so algorithm returns. And no more recursive calls are left. So polygon with edges (AD, DC, CE, EB, BF, FG, GA) is the convex hull of given points.

**Applications of Convex Hull**

Collision avoidance: If the convex hull of car avoids collisions with obstructions, so does the car. Because calculating collision-free routes is considerably easier with a convex hull of vehicle, it is frequently used to design paths.

Smallest Box: Convex hull helps to determine the minimum size required to put object in in. It is useful to determine box size for the object. Finding the smallest three-dimensional box enclosing an item is also dependent on the 3D-convex hull.

Shape Analysis: Convex hull of object is useful to analyze the shape of object.

Pattern recognition, image processing, statistics, geographic information systems, game theory, phase diagram creation, and static code analysis via abstract interpretation are some more practical scenarios where convex hull plays important role.

It is also a building component for a variety of other computational-geometric methods, such as the rotating calipers technique for calculating the breadth and diameter of given set of points.