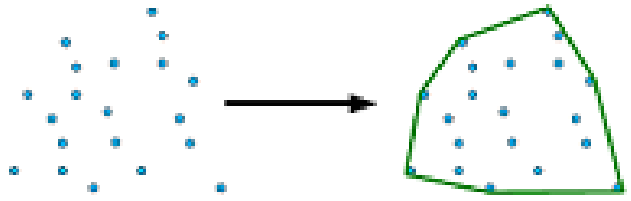
Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping)

Given a set of points in the plane. the convex hull of the set is the smallest convex polygon that contains all the points of it.



We strongly recommend to see the following post first.
How to check if two given line segments intersect?

The idea of Jarvis's Algorithm is simple, we start from the leftmost point (or point with minimum x coordinate value) and we keep wrapping points in counterclockwise direction. The big question is, given a point p as current point, how to find the next point in output? The idea is to useorientation() here. Next point is selected as the point that beats all other points at counterclockwise orientation, i.e., next point is q if for any other point r, we have "orientation(p, r, q) = counterclockwise". Following is the detailed algorithm.

1) Initialize p as leftmost point.
2) Do following while we don't come back to the first (or leftmost) point.
…..**a)** The next point q is the point such that the triplet (p, q, r) is counterclockwise for any other point r.
…..**b)** next[p] = q (Store q as next of p in the output convex hull).
…..**c)** p = q (Set p as q for next iteration).

```
// A C++ program to find convex hull of a set of points
// Refer http://www.geeksforgeeks.org/check-if-two-given
// for explanation of orientation()
#include <iostream>
using namespace std;

// Define Infinite (Using INT_MAX caused overflow proble
#define INF 10000
```

```
struct Point
{
    int x;
    int y;
};

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // colinear
    return (val > 0)? 1: 2; // clock or counterclock wis
}

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // There must be at least 3 points
    if (n < 3) return;

    // Initialize Result
    int next[n];
    for (int i = 0; i < n; i++)
        next[i] = -1;

    // Find the leftmost point
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Start from leftmost point, keep moving counterclo
    // until reach the start point again
    int p = l, q;
    do
    {
        // Search for a point 'q' such that orientation(
        // counterclockwise for all points 'i'
        q = (p+1)%n;
```

```
        for (int i = 0; i < n; i++)
          if (orientation(points[p], points[i], points[q
             q = i;

        next[p] = q;  // Add q to result as a next point
        p = q; // Set p as q for next iteration
    } while (p != l);

    // Print Result
    for (int i = 0; i < n; i++)
    {
        if (next[i] != -1)
           cout << "(" << points[i].x << ", " << points[
    }
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {2, 2}, {1, 1}, {2, 1},
                      {3, 0}, {0, 0}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}
```

Output: The output is points of the convex hull.

```
(0, 3)
(3, 0)
(0, 0)
(3, 3)
```

Time Complexity: For every point on the hull we examine all the other points to determine the next point. Time complexity is Θ(m * n) where n is number of input points and m is number of output or hull points (m <= n). In worst case, time complexity is $O(n^2)$. The worst case occurs when all the points are on the hull (m = n)

We will soon be discussing other algorithms for finding convex hulls.

Sources:

http://www.cs.uiuc.edu/~jeffe/teaching/373/notes/x05-convexhull.pdf

http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf