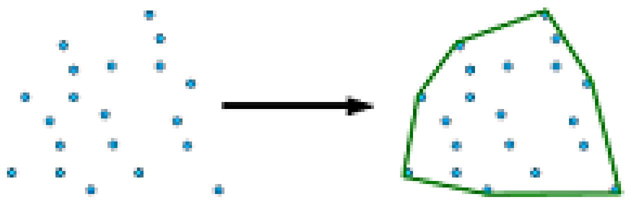
Convex Hull | Set 2 (Graham Scan)

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?

We have discussed Jarvis's Algorithm for Convex Hull. Worst case time complexity of Jarvis's Algorithm is O(n^2). Using Graham's scan algorithm, we can find Convex Hull in O(nLogn) time. Following is Graham's algorithm

Let points[0..n-1] be the input array.

1) Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered. Put the bottom-most point at first position.

2) Consider the remaining n-1 points and sort them by polor angle in counterclockwise order around points[0]. If polor angle of two points is same, then put the nearest point first.

3) Create an empty stack 'S' and push points[0], points[1] and points[2] to S.

4) Process remaining n-3 points one by one. Do following for every point 'points[i]'

4.1) Keep removing points from stack while orientation of following 3 points is not counterclockwise (or they don't make a left turn).

a) Point next to top in stack

b) Point at the top of stack

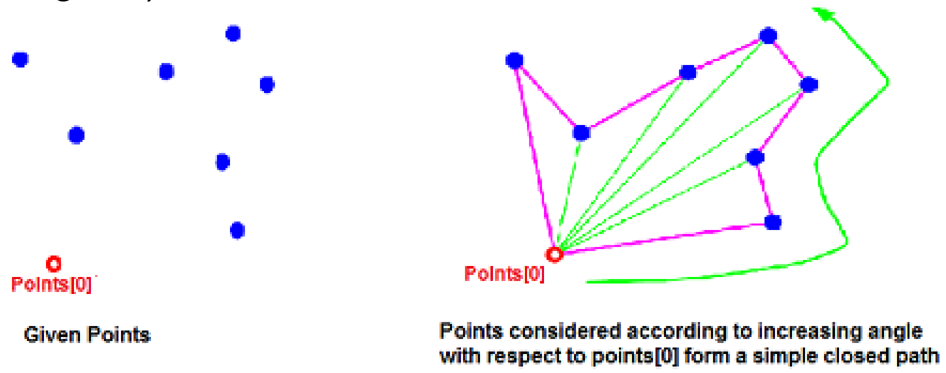
c) points[i]

4.2) Push points[i] to S

5) Print contents of S

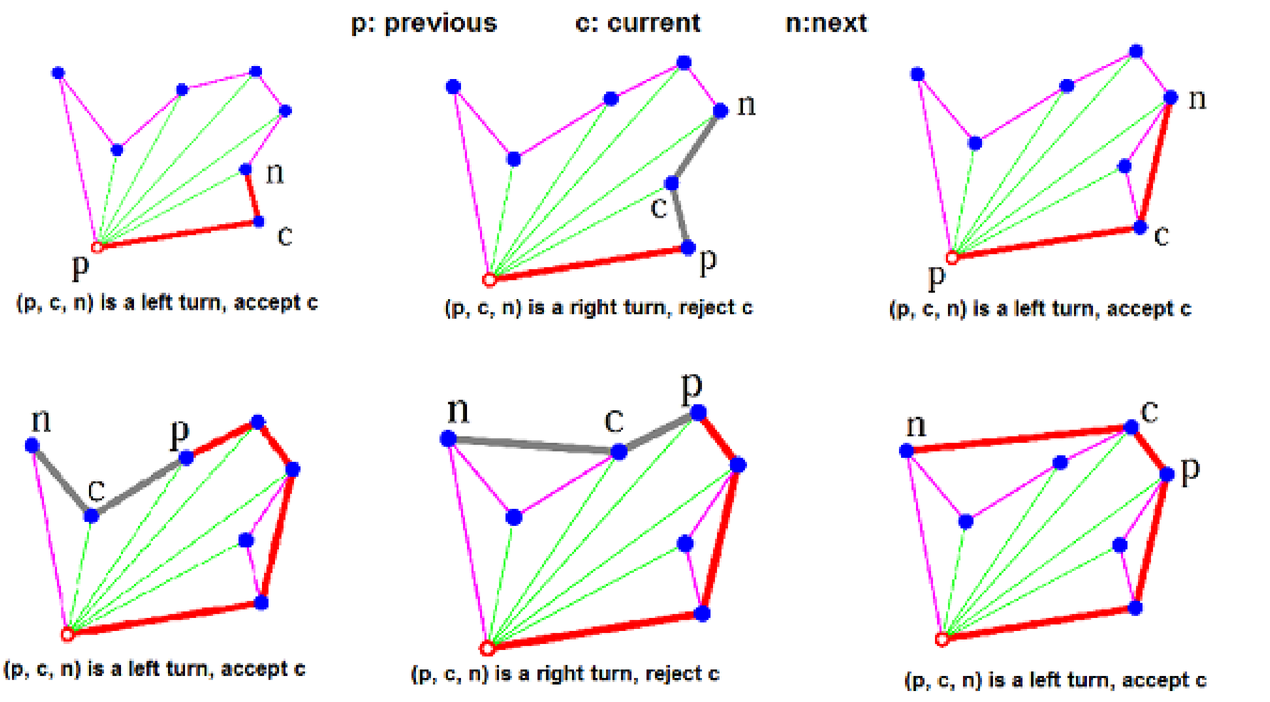
The above algorithm can be divided in two phases.

Phase 1 (Sort points): We first find the bottom-most point. The idea is to pre-process points be sorting them with respect to the bottom-most point. Once the points are sorted, they form a simple closed path (See following diagram).


Points[0]
Given Points
Points[0]
Points considered according to increasing angle with respect to points[0] form a simple closed path

What should be the sorting criteria? computation of actual angles would be inefficient since trigonometric functions are not simple to evaluate. The idea is to use the orientation to compare angles without actually computing them (See the compare() function below)

Phase 2 (Accept or Reject Points): Once we have the closed path, the next step is to traverse the path and remove concave points on this path. How to decide which point to remove and which to keep? Again, orientation helps here. The first two points in sorted array are always part of Convex Hull. For remaining points, we keep track of recent three points, and find the angle formed by them. Let the three points be prev(p), curr(c) and next(n). If orientation of these points (considering them in same order) is not counterclockwise, we discard c, otherwise we keep it. Following diagram shows step by step process of this phase (Source of these diagrams is Ref 2).


p: previous
c: current
n:next
(p, c, n) is a left turn, accept c
(p, c, n) is a right turn, reject c
(p, c, n) is a left turn, accept c
(p, c, n) is a left turn, accept c
(p, c, n) is a right turn, reject c
(p, c, n) is a left turn, accept c

In the above algorithm and below code, a stack of points is used to store convex hull points. With reference to the code, p is next-to-top in stack, c is top of stack and n is points[i].

Following is C++ implementation of the above algorithm.

```
// 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>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x;
    int y;
};

// A globle point needed for  sorting points with refere
// Used in compare function of qsort()
Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance betwe
int dist(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.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
}

// A function used by library function qsort() to sort a
// points with respect to the first point
int compare(const void *vp1, const void *vp2)
{
   Point *p1 = (Point *)vp1;
   Point *p2 = (Point *)vp2;

   // Find orientation
   int o = orientation(p0, *p1, *p2);
   if (o == 0)
     return (dist(p0, *p2) >= dist(p0, *p1))? -1 : 1;

   return (o == 2)? -1: 1;
}
```

```
// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
   // Find the bottommost point
   int ymin = points[0].y, min = 0;
   for (int i = 1; i < n; i++)
   {
     int y = points[i].y;

     // Pick the bottom-most or chose the left most poin
     if ((y < ymin) || (ymin == y && points[i].x < point
        ymin = points[i].y, min = i;
   }

   // Place the bottom-most point at first position
   swap(points[0], points[min]);

   // Sort n-1 points with respect to the first point.
   // before p2 in sorted ouput if p2 has larger polar a
   // counterclockwise direction) than p1
   p0 = points[0];
   qsort(&points[1], n-1, sizeof(Point), compare);

   // Create an empty stack and push first three points
   stack<Point> S;
   S.push(points[0]);
   S.push(points[1]);
   S.push(points[2]);

   // Process remaining n-3 points
   for (int i = 3; i < n; i++)
   {
      // Keep removing top while the angle formed by poi
      // top, and points[i] makes a non-left turn
      while (orientation(nextToTop(S), S.top(), points[i
         S.pop();
      S.push(points[i]);
   }

   // Now stack has the output points, print contents of
   while (!S.empty())
   {
       Point p = S.top();
       cout << "(" << p.x << ", " << p.y <<")" << endl;
       S.pop();
   }
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                      {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}
```

Output:

```
(0, 3)
(4, 4)
(3, 1)
(0, 0)
```

Time Complexity: Let n be the number of input points. The algorithm takes O(nLogn) time if we use a O(nLogn) sorting algorithm.
The first step (finding the bottom-most point) takes O(n) time. The second step (sorting points) takes O(nLogn) time. In third step, every element is pushed and popped at most one time. So the third step to process points one by one takes O(n) time, assuming that the stack operations take O(1) time. Overall complexity is O(n) + O(nLogn) + O(n) which is O(nLogn)

References:

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf