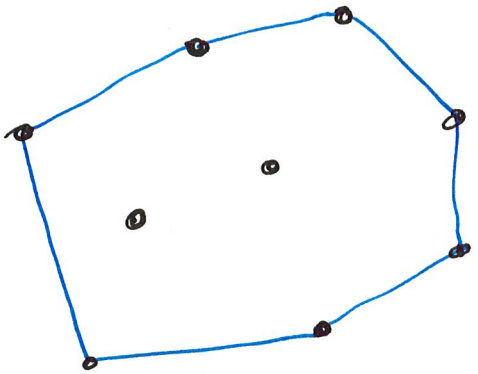
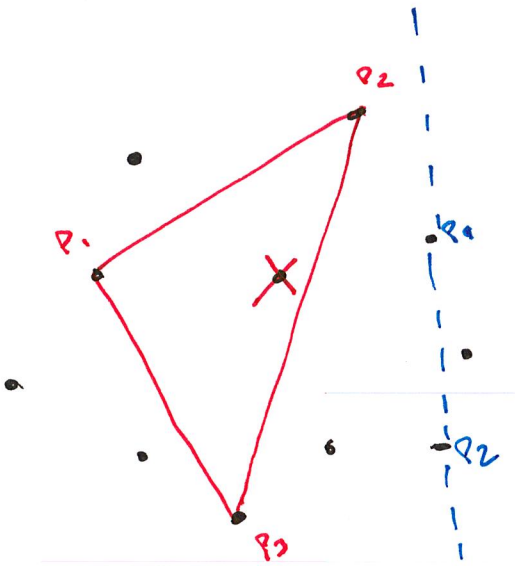
Convex Hull



- Given a set of n points S, find the smallest convex polygon containing all the points
- we'll present the convex polygon as the ordered sequence of vertices

Brute Force #1 $O(n^4)$

- Any point inside any triangle is not part of the hull
- sort by polar angle around a centroid

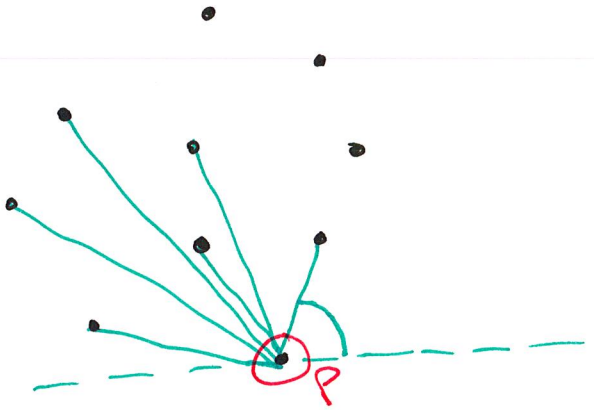


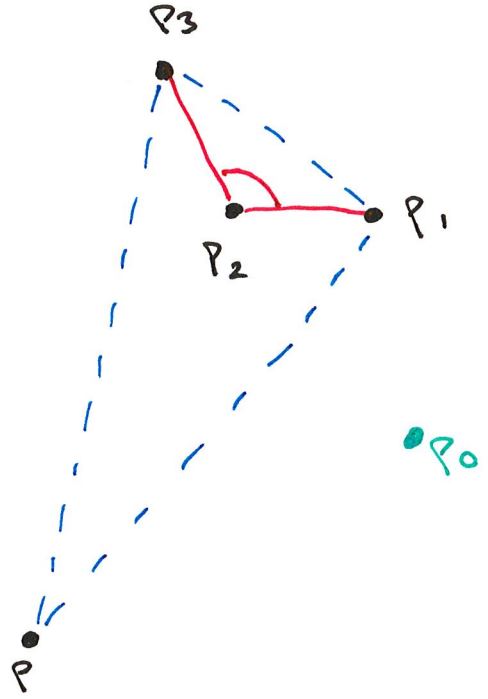
Brute Force #2 $O(n^3)$

- For each line segment (pair of points) if all other points are on the same side of that line, the segment is part of the hull.

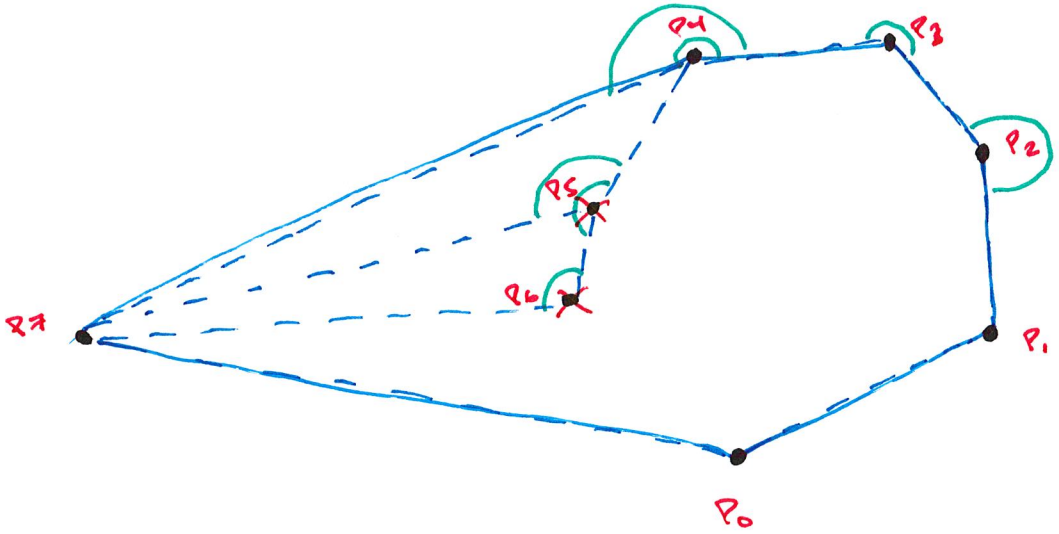
Graham's Scan

1. Find a known point on the hull p
2. Sort the remaining points by polar angle in counter-clockwise order around p
3. Process the points in consecutive triples $p_1 p_2 p_3$
   - If $\angle p_1 p_2 p_3 < \pi$ then p_2 is in the interior of the triangle $p p_1 p_3$ therefore not part of the hull





- If $\angle p_1 p_2 p_3 < \pi$ discard p_2 and try $p_0 p_1 p_3$
- otherwise advance to $p_2 p_3 p_4$

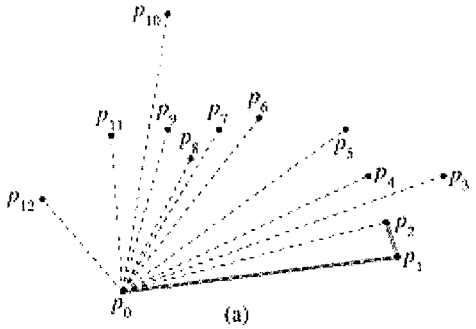

P4
P3
P2
P5
P6
P7
P1
P0

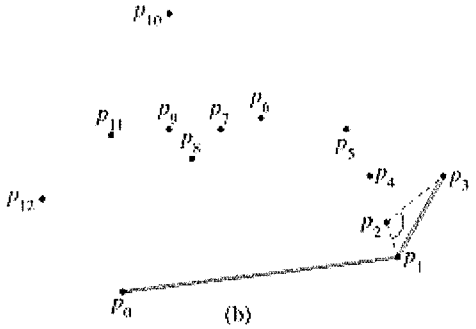
```
GrahamScan(Q)
1  let p_0 be the point in Q with the lowest y-coordinate
2  let ⟨p_1, p_2, ..., p_m⟩ be the remaining points in Q
   sorted by polar angle around p_0
3  if m < 2
4        return "convex hull is empty"
5  else let S be an empty stack
6        Push(p_0, S)
7        Push(p_1, S)
8        Push(p_2, S)
9        for i = 3 to m
10             while the angle formed by p_i and the top 2 elements
               of S makes a non-left turn
11                   Pop(S)
12             Push(p_i, S)
13 return S
```

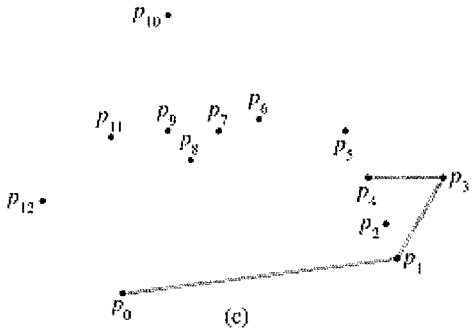
Θ(n)

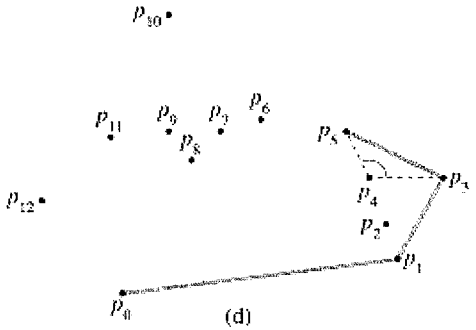
Θ(n log n)

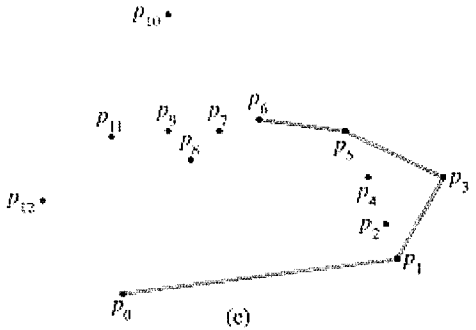
Θ(n)

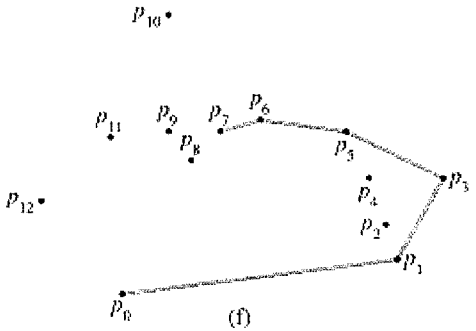

p_{10}
p_{11}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(a)

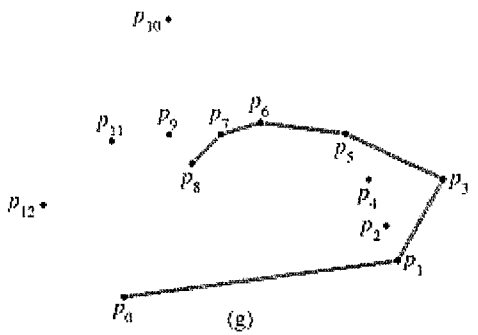

p_{10}
p_{11}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(b)

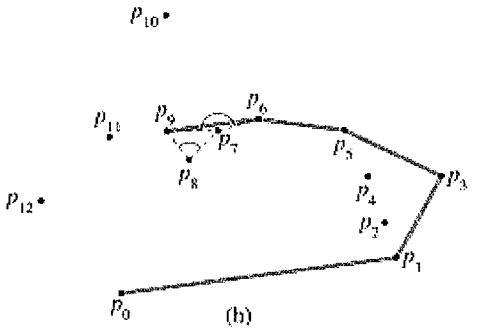

p_{10}
p_{11}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(c)

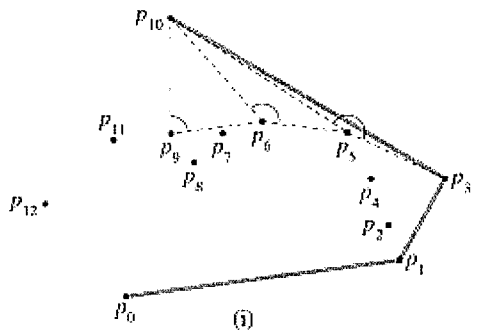

p_{10}
p_{11}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(d)

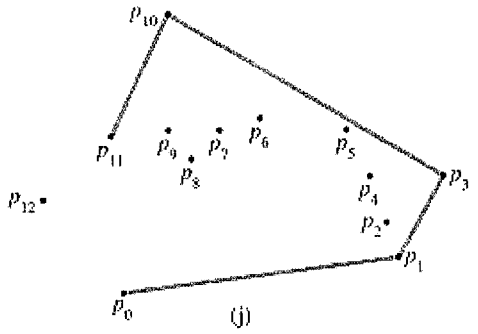

p_{10}
p_{11}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(e)

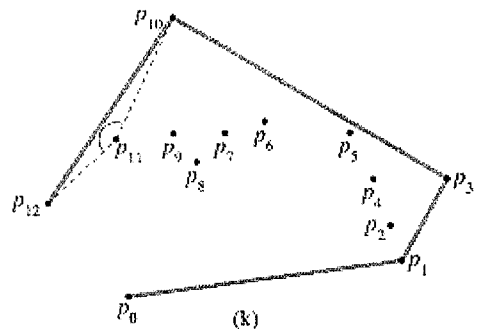

p_{10}
p_{11}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0
(f)

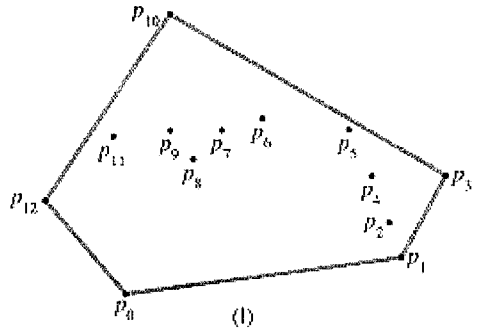

p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_3
p_{12}
p_4
p_2
p_1
p_0
(g)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_3
p_{12}
p_4
p_2
p_1
p_0
(h)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_3
p_{12}
p_4
p_2
p_1
p_0
(i)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_3
p_{12}
p_4
p_2
p_1
p_0
(j)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_3
p_{12}
p_4
p_2
p_1
p_0
(k)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_3
p_{12}
p_4
p_2
p_1
p_0
(l)

PotD: Singleton Finder

- Given a sorted array in which each element appears twice and one element appears only once. Give a divide and conquer algorithm to find that element with complexity *O(log n)*
- Example

  input: [1, 1, 3, 3, 4, 5, 5, 7, 7, 8, 8]

  output: 4