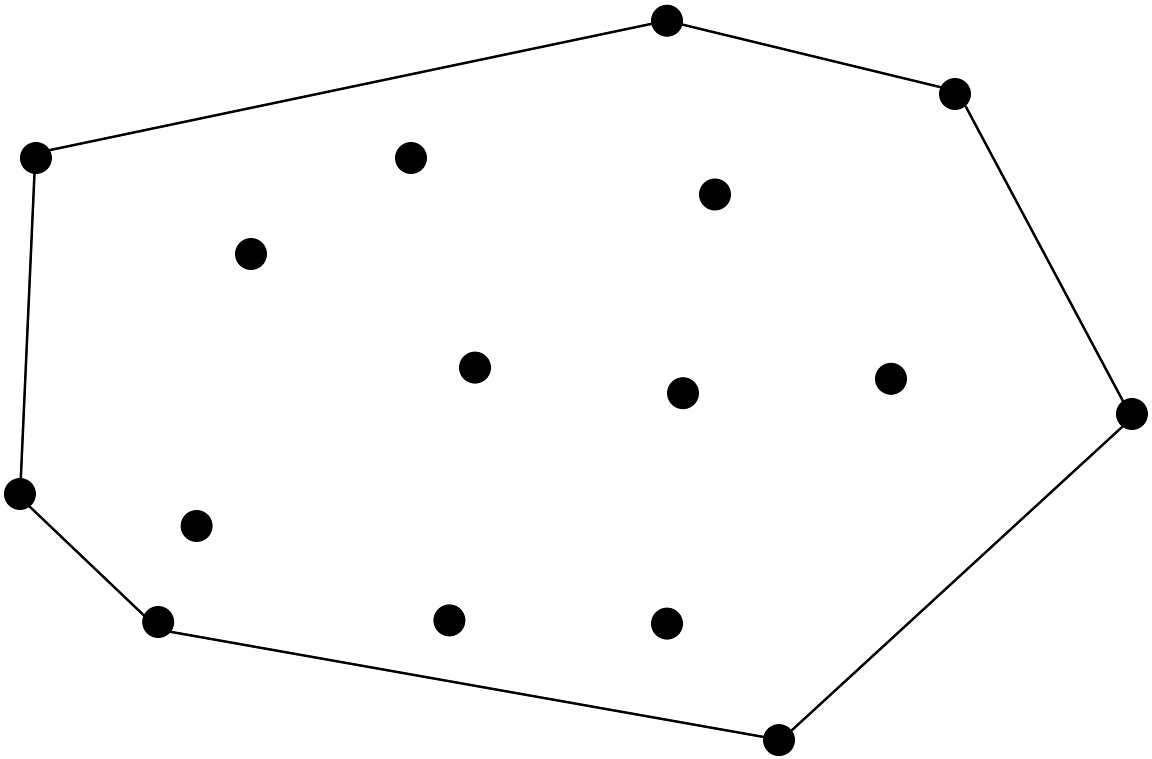
CS133
Computational Geometry

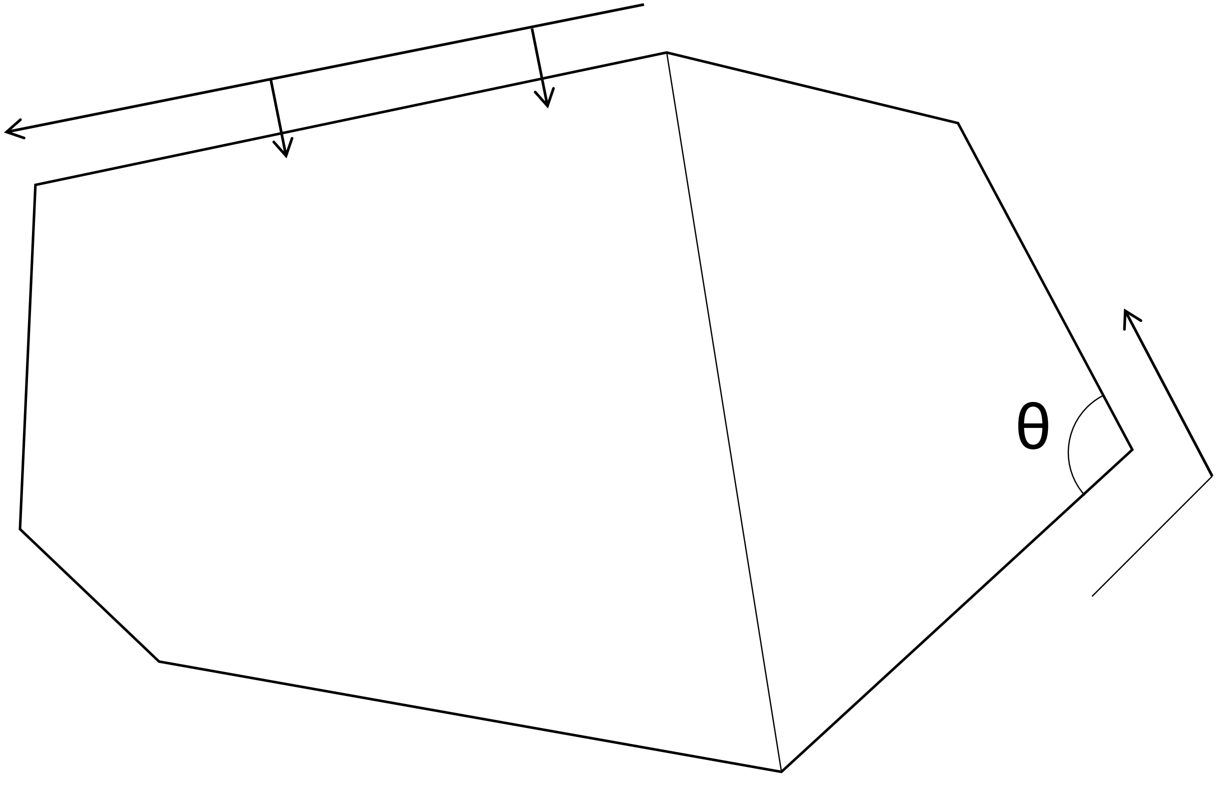
Convex Hull

Convex Hull

- Given a set of n points, find the minimal convex polygon that contains all the points

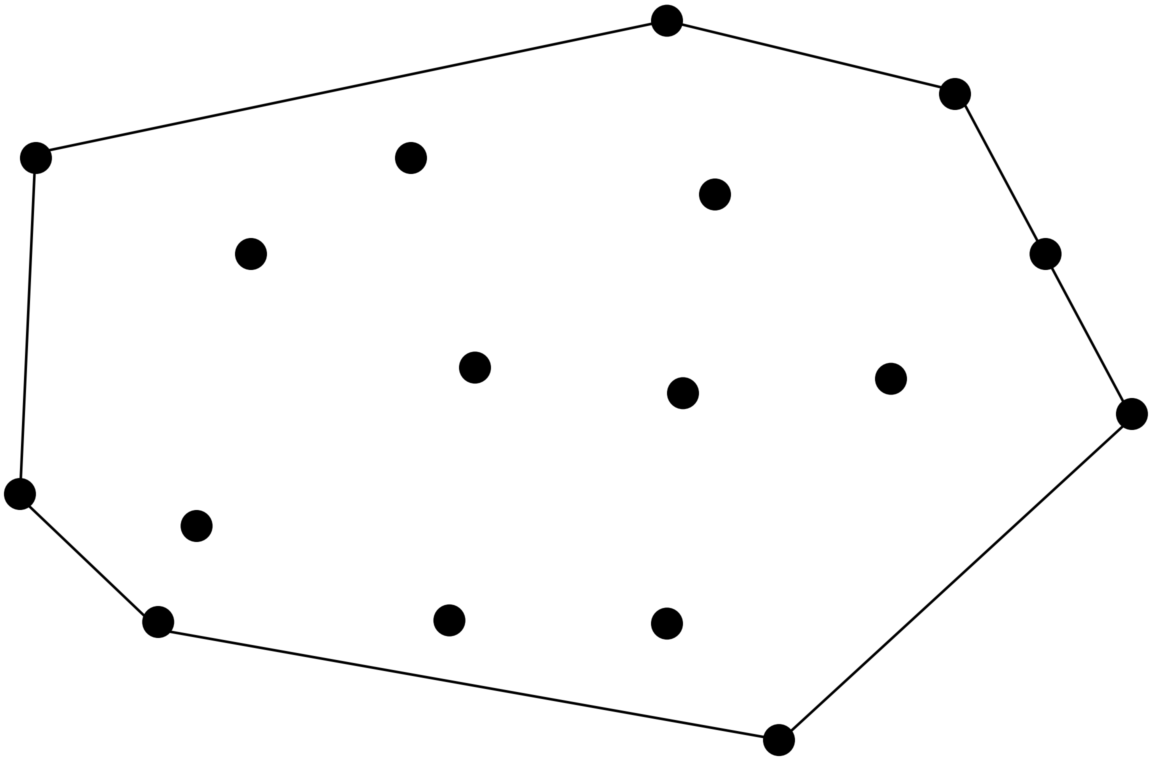


Convex Hull Properties



Convex Hull Representation

- The convex hull is represented by all its points sorted in CW/CCW order
- Special case: Three collinear points

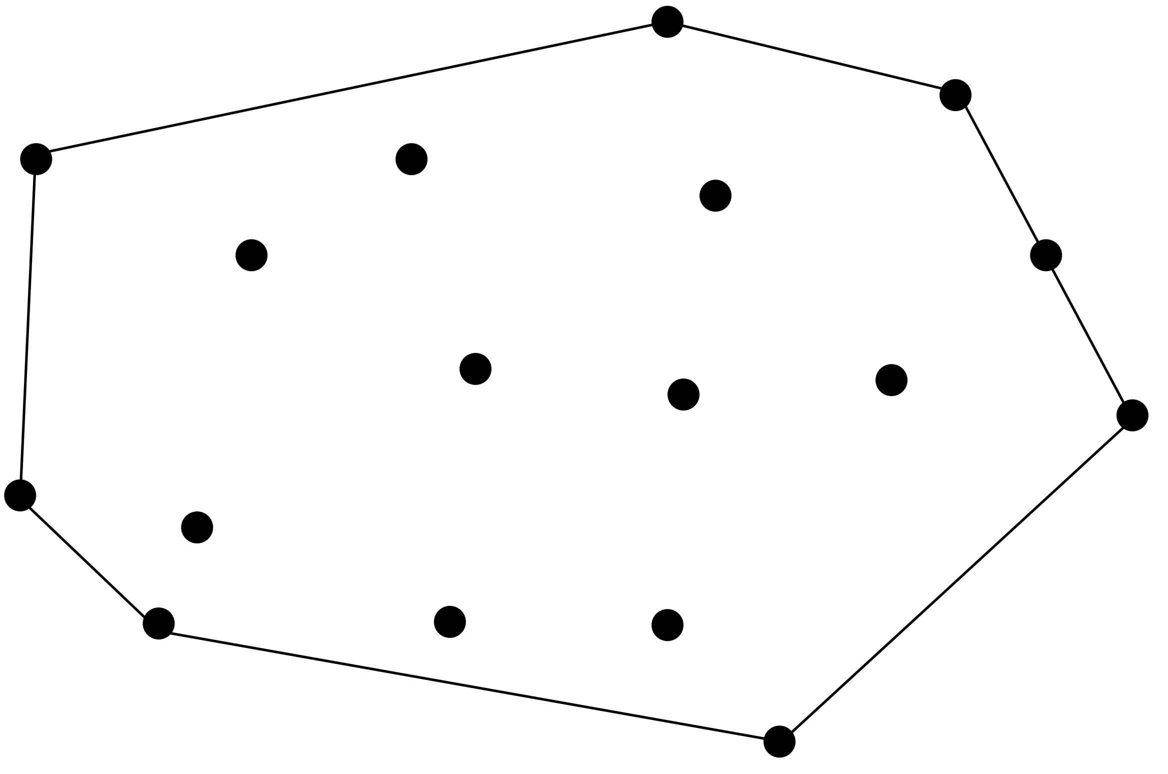


Naïve Convex Hull Algorithm

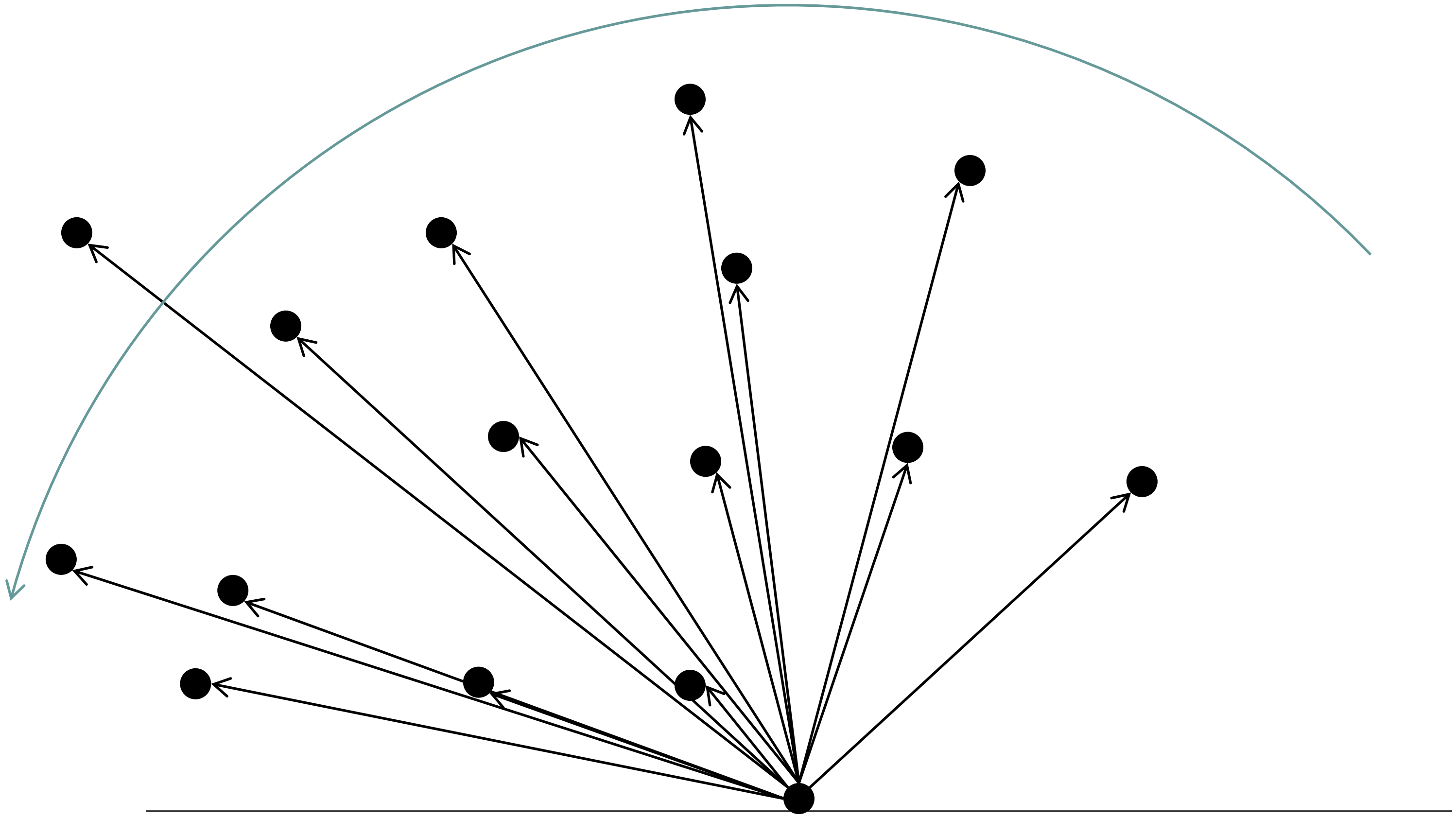
- Iterate over all possible line segments
- A line segment is part of the convex hull if all other points are to its left
- Emit all segments in a CCW order

- Running time $O(n^3)$

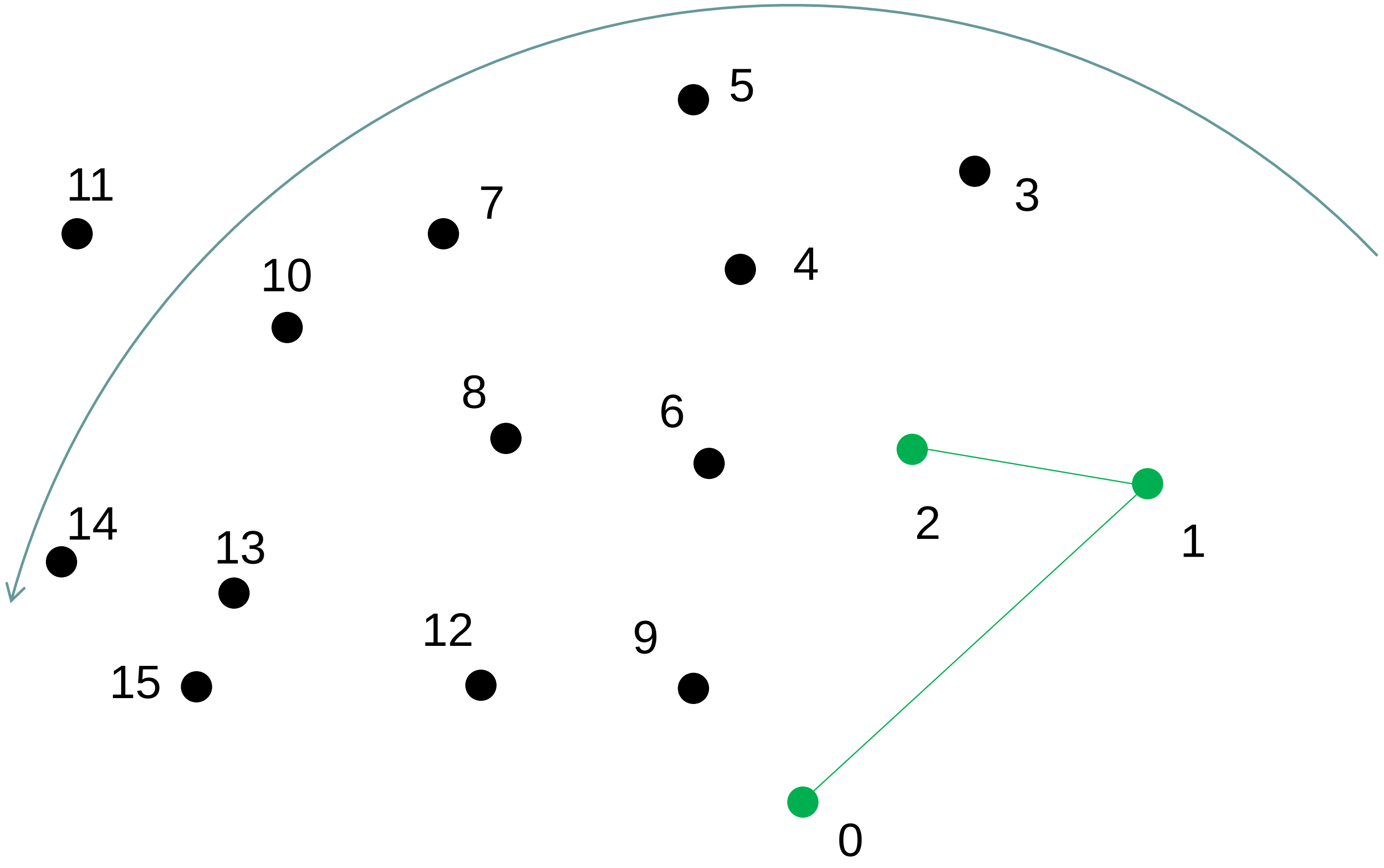
Naïve Convex Hull Algorithm



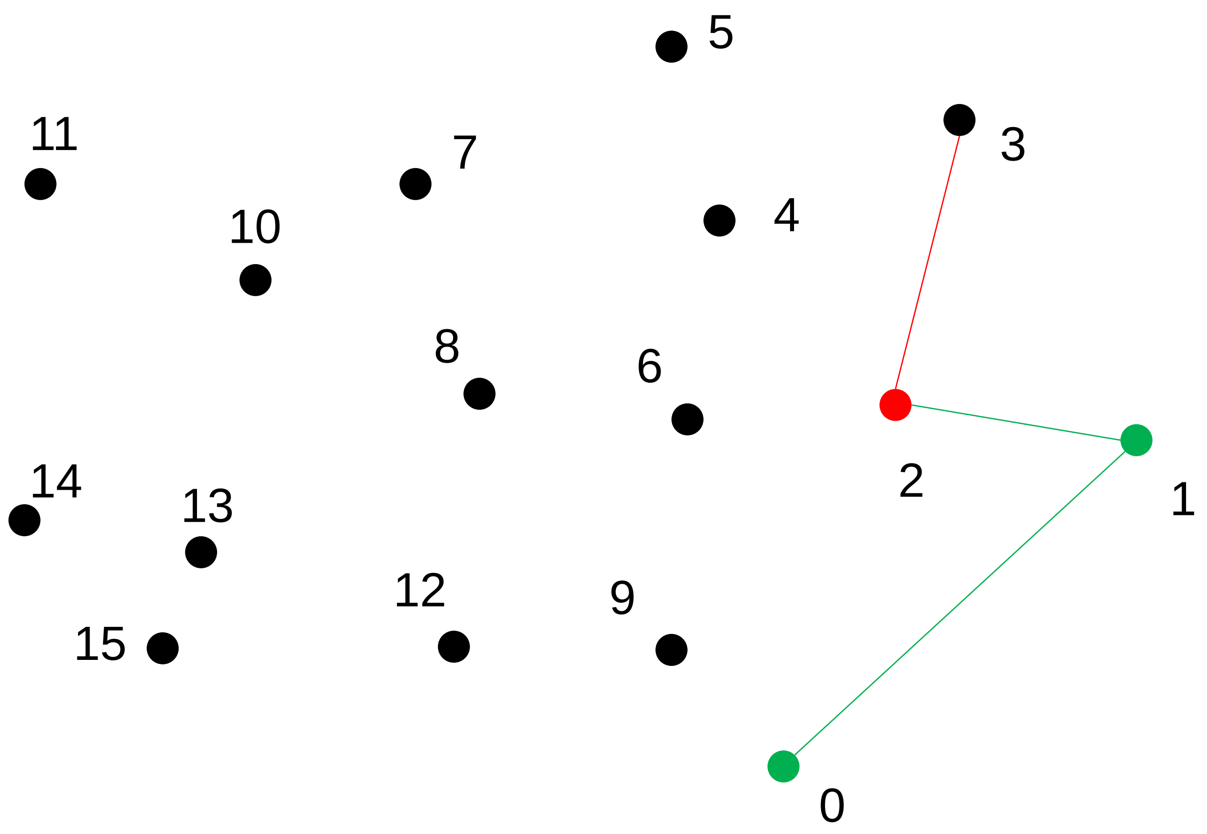
Graham Scan Algorithm



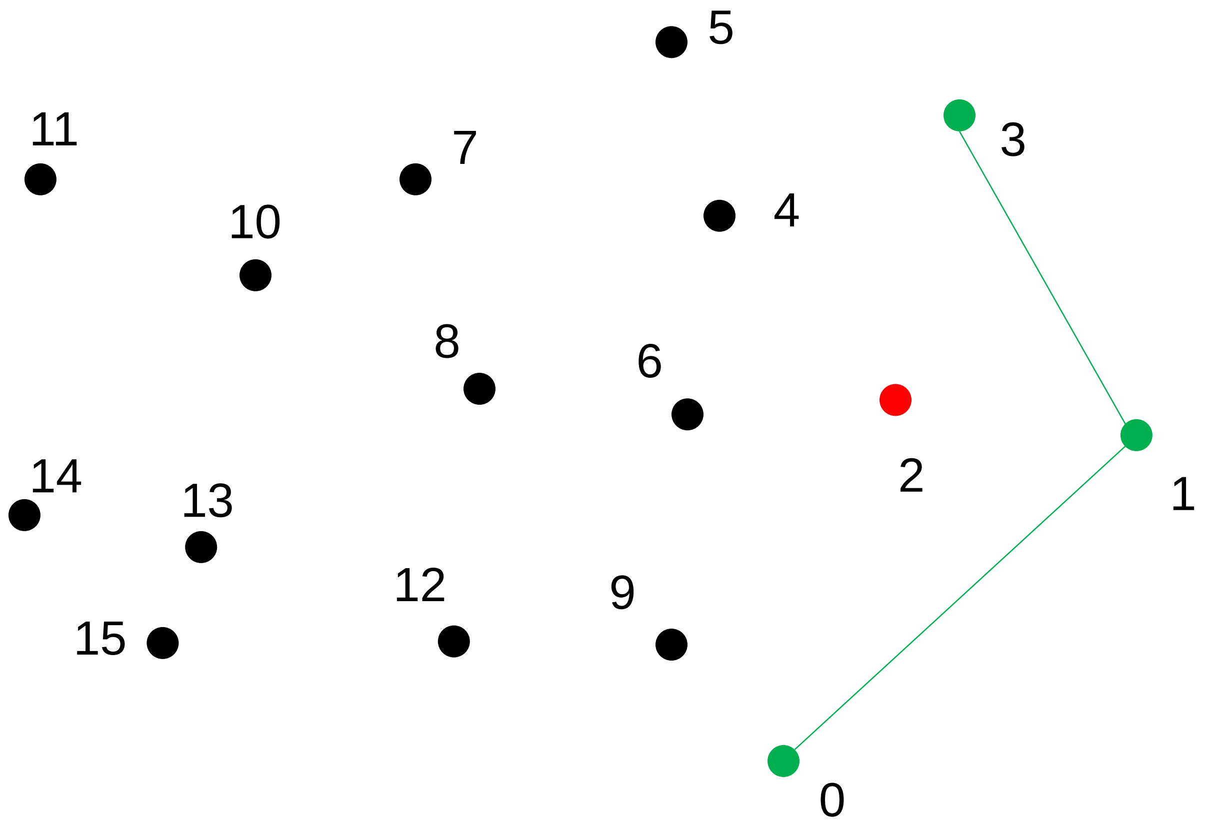
Graham Scan Algorithm



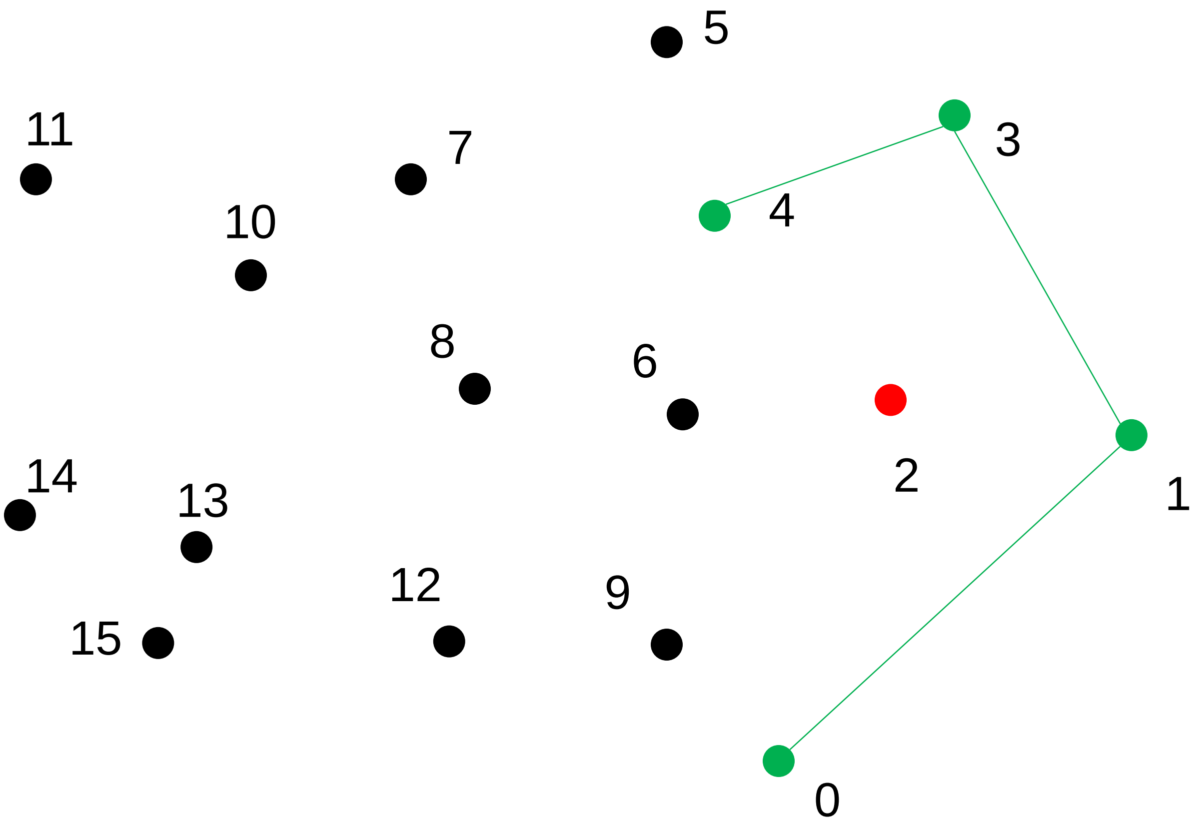
Graham Scan Algorithm



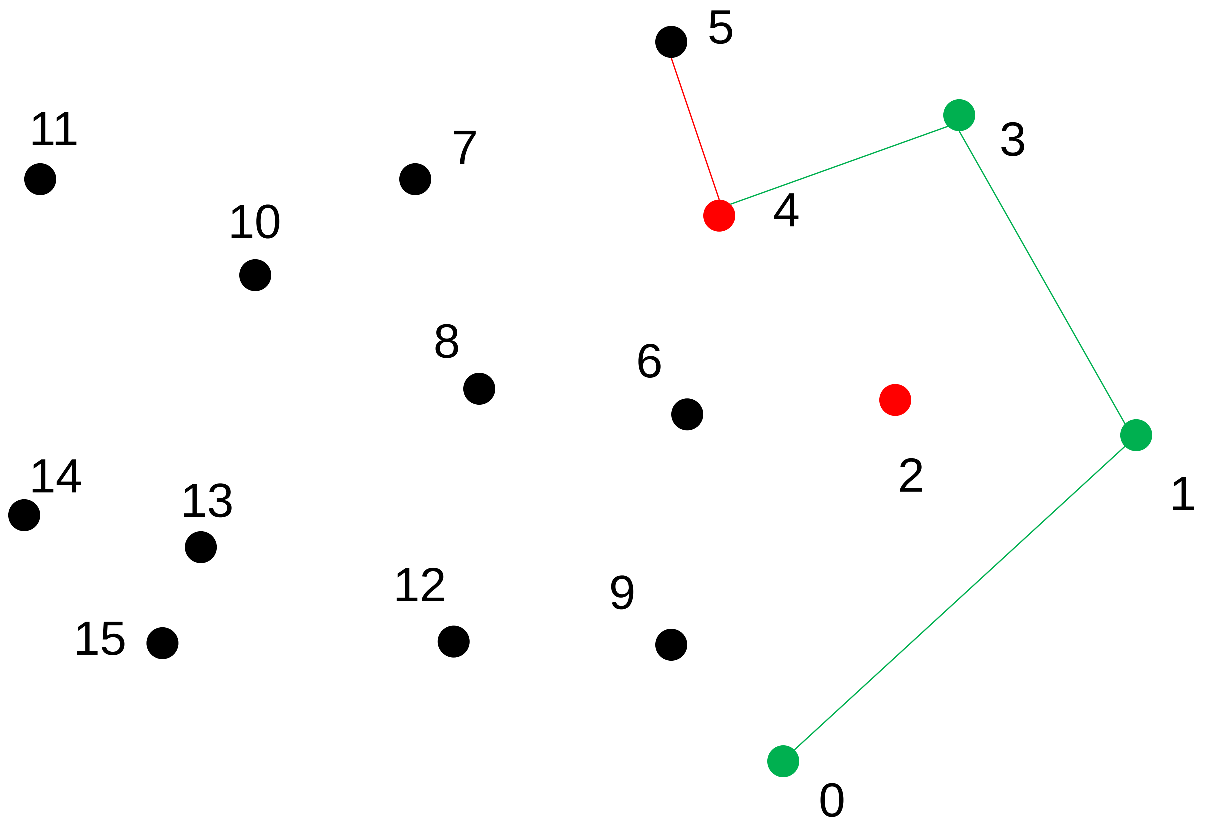
Graham Scan Algorithm



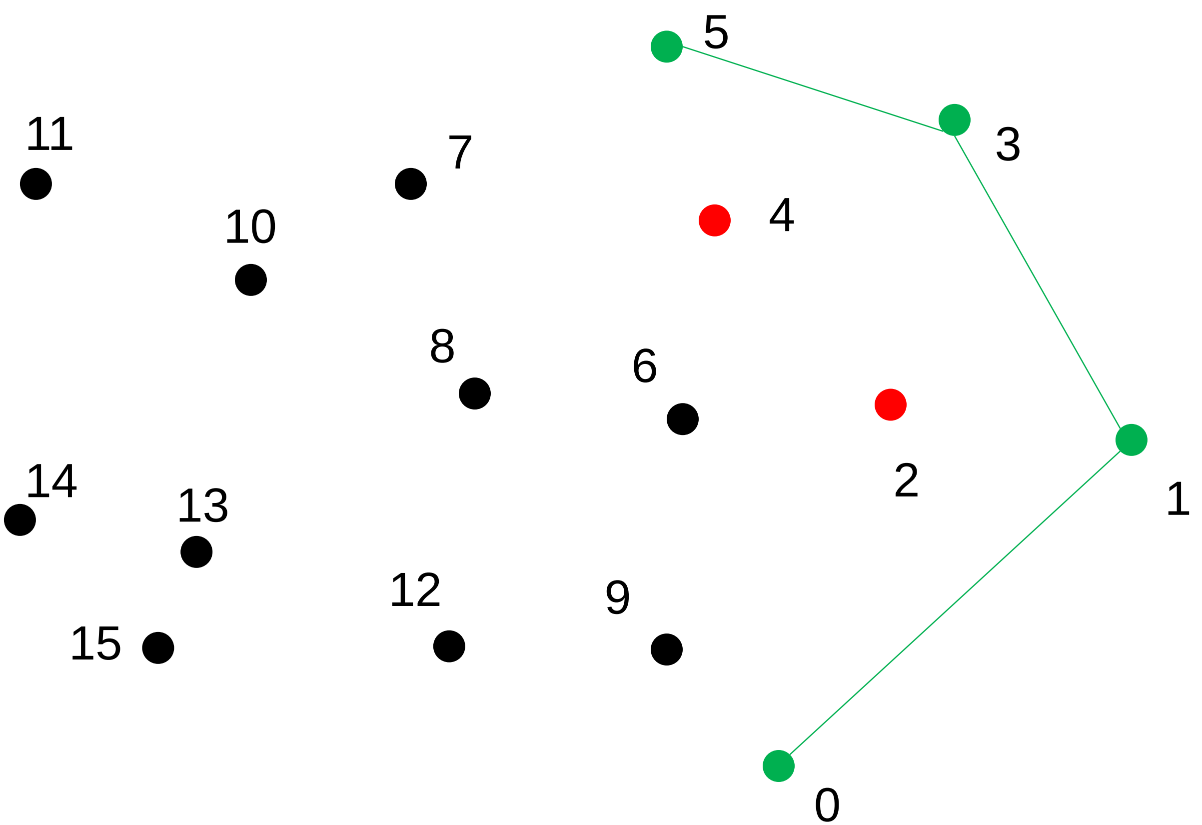
Graham Scan Algorithm



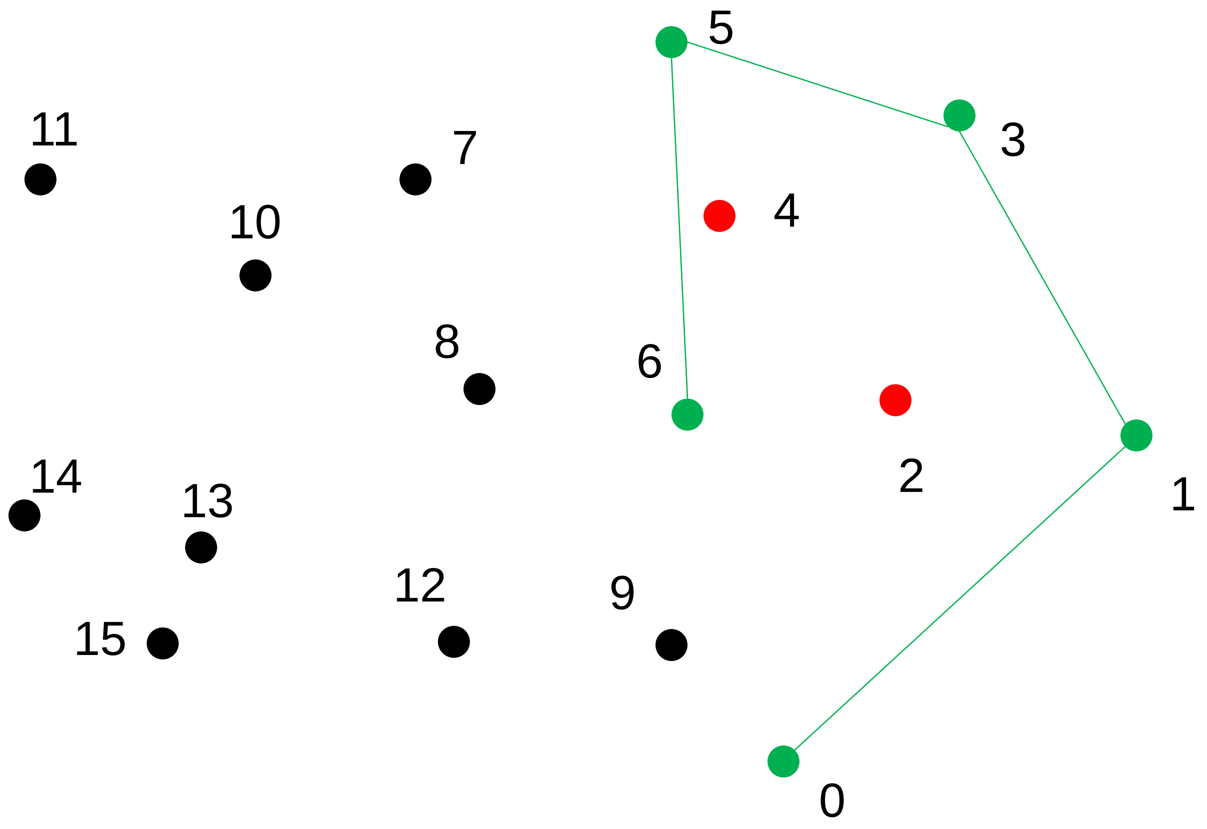
Graham Scan Algorithm



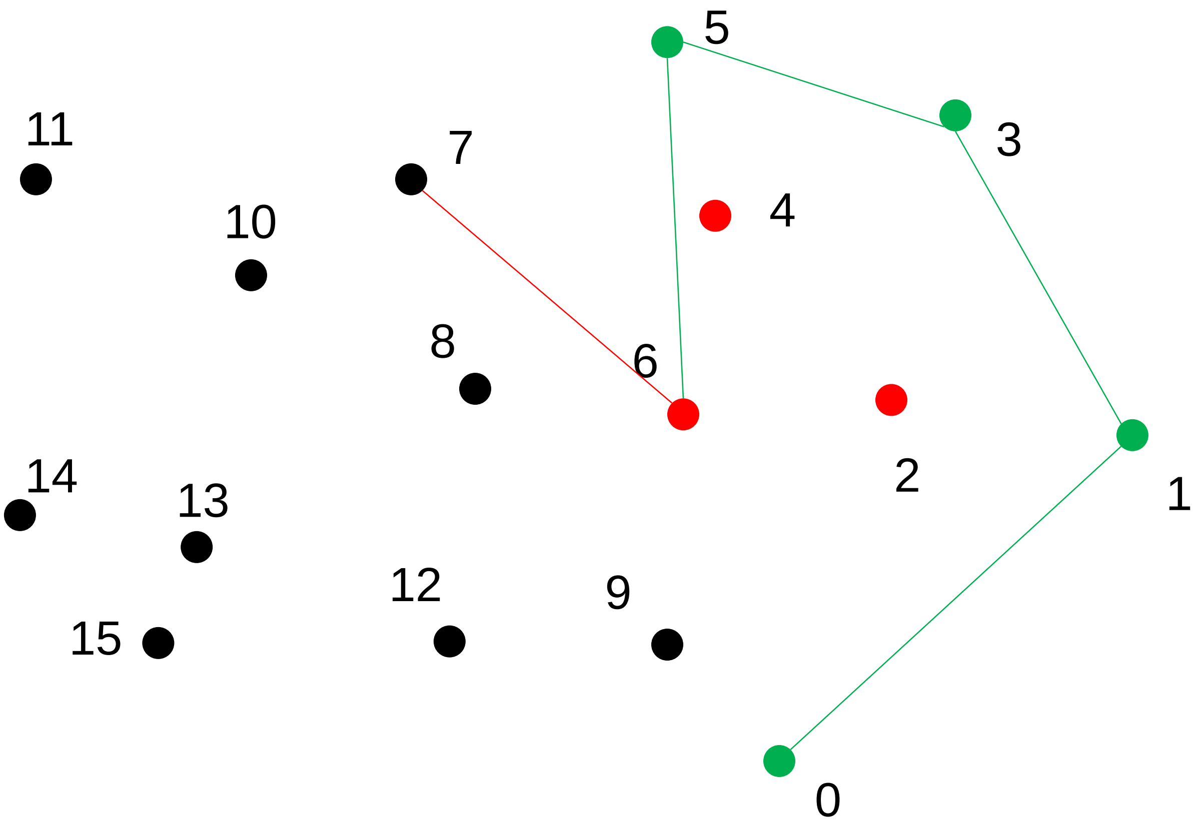
Graham Scan Algorithm



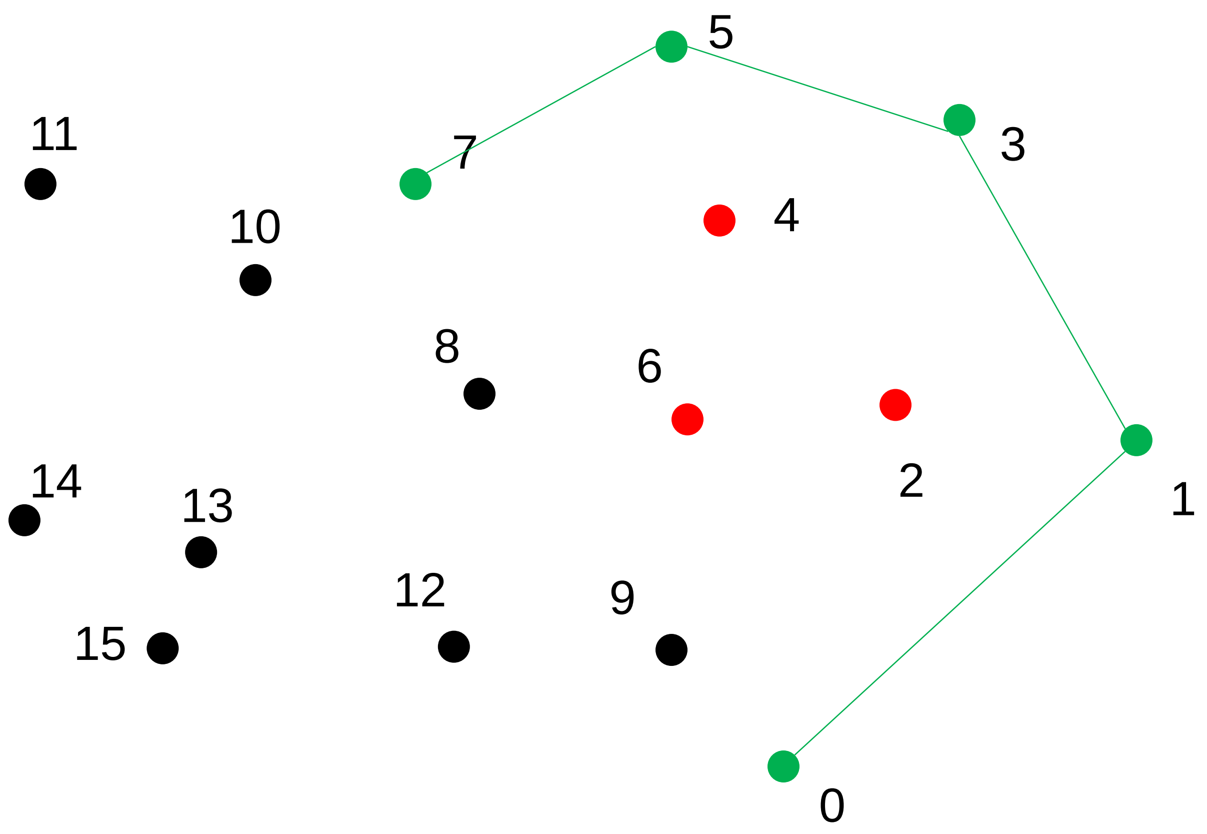
Graham Scan Algorithm



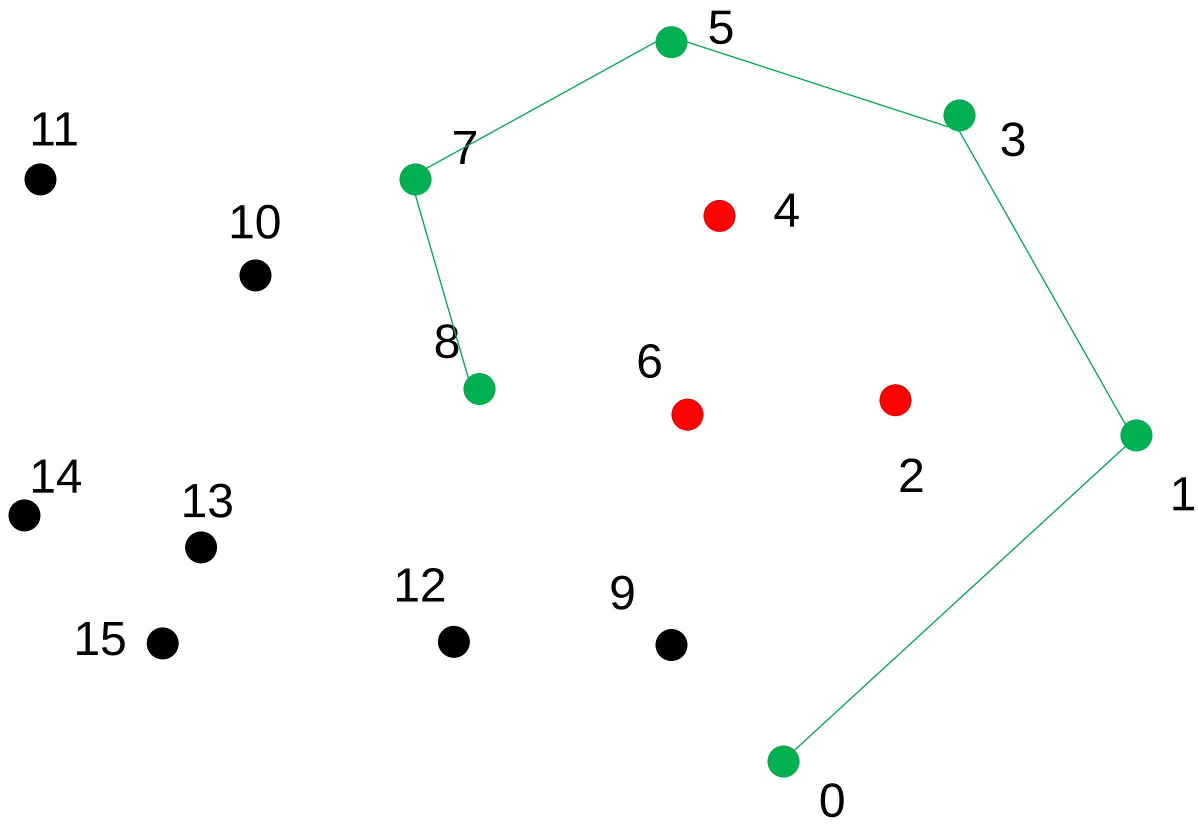
Graham Scan Algorithm



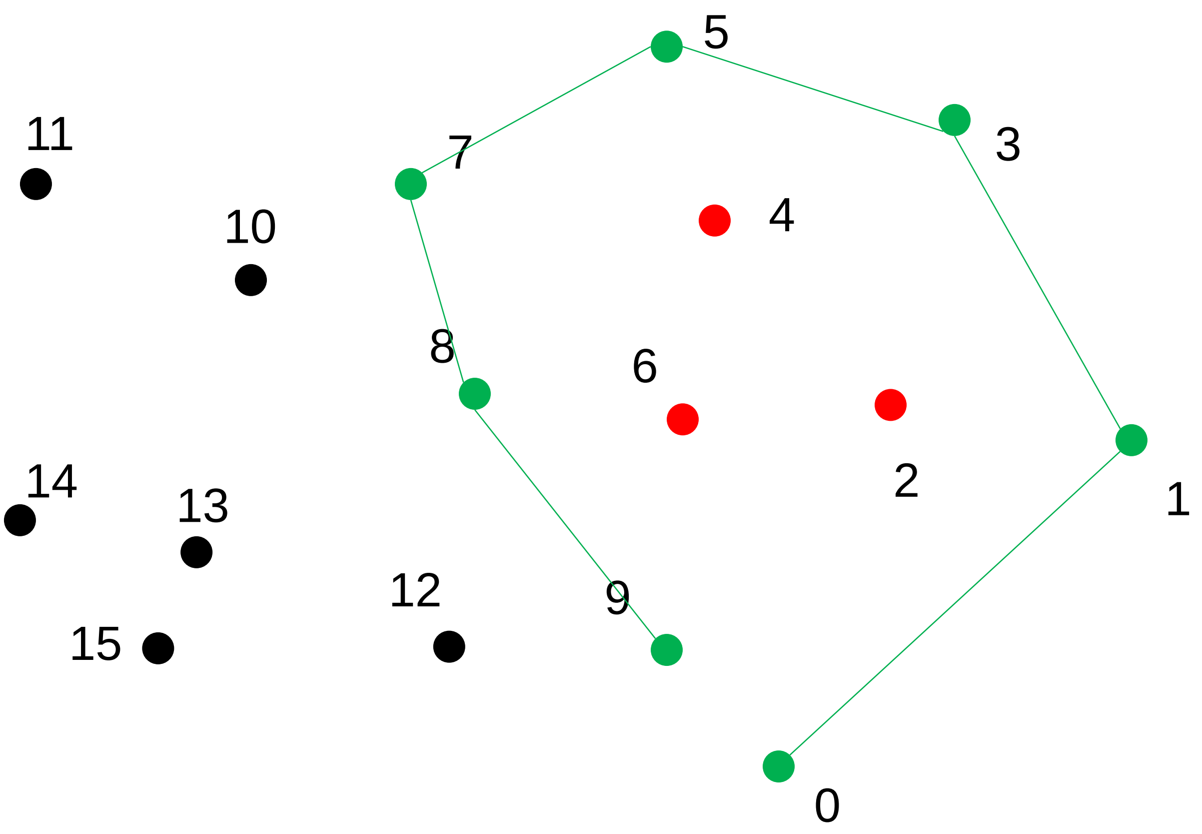
Graham Scan Algorithm



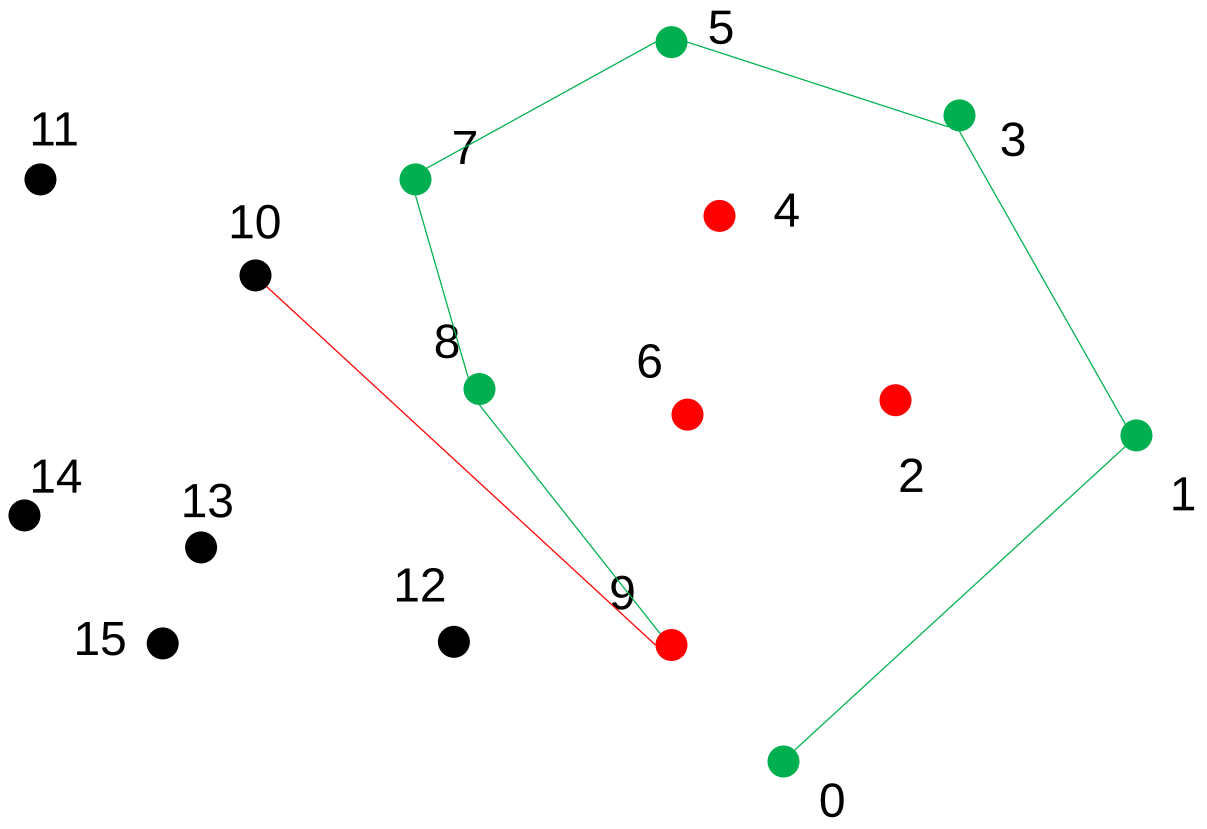
Graham Scan Algorithm



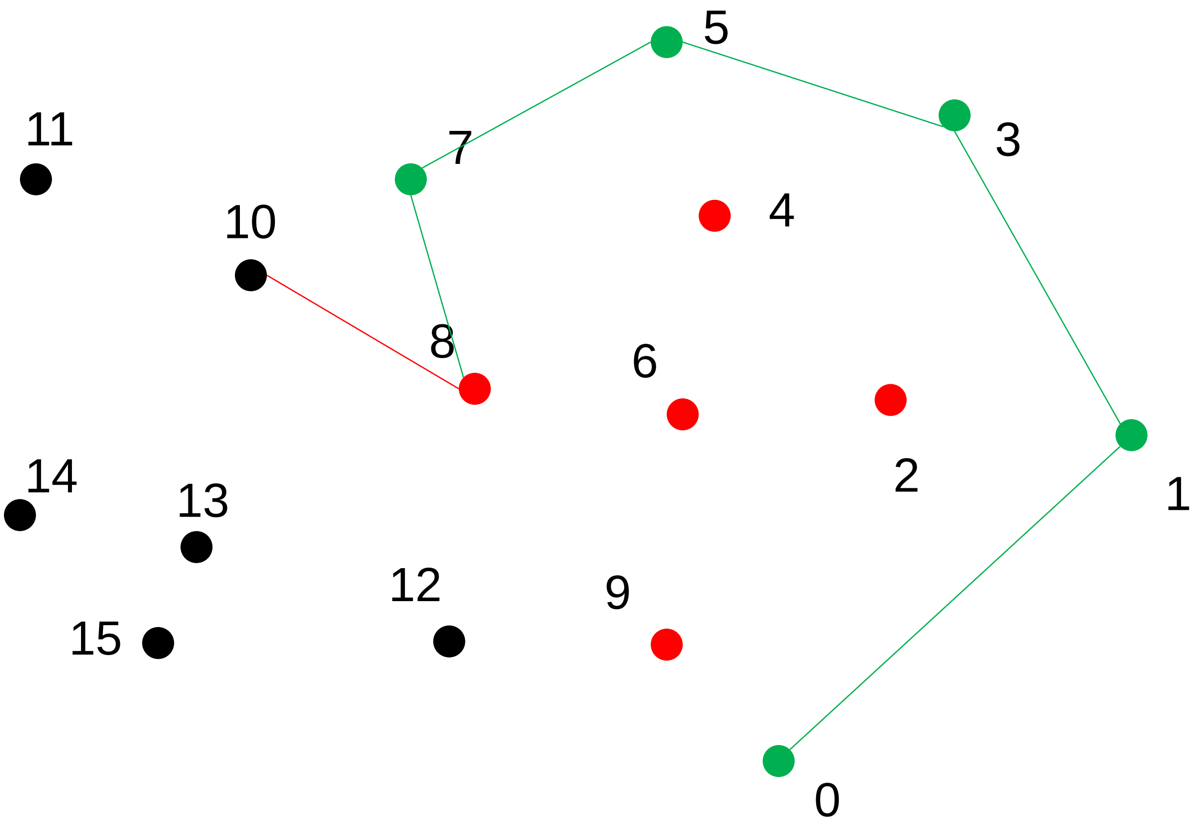
Graham Scan Algorithm



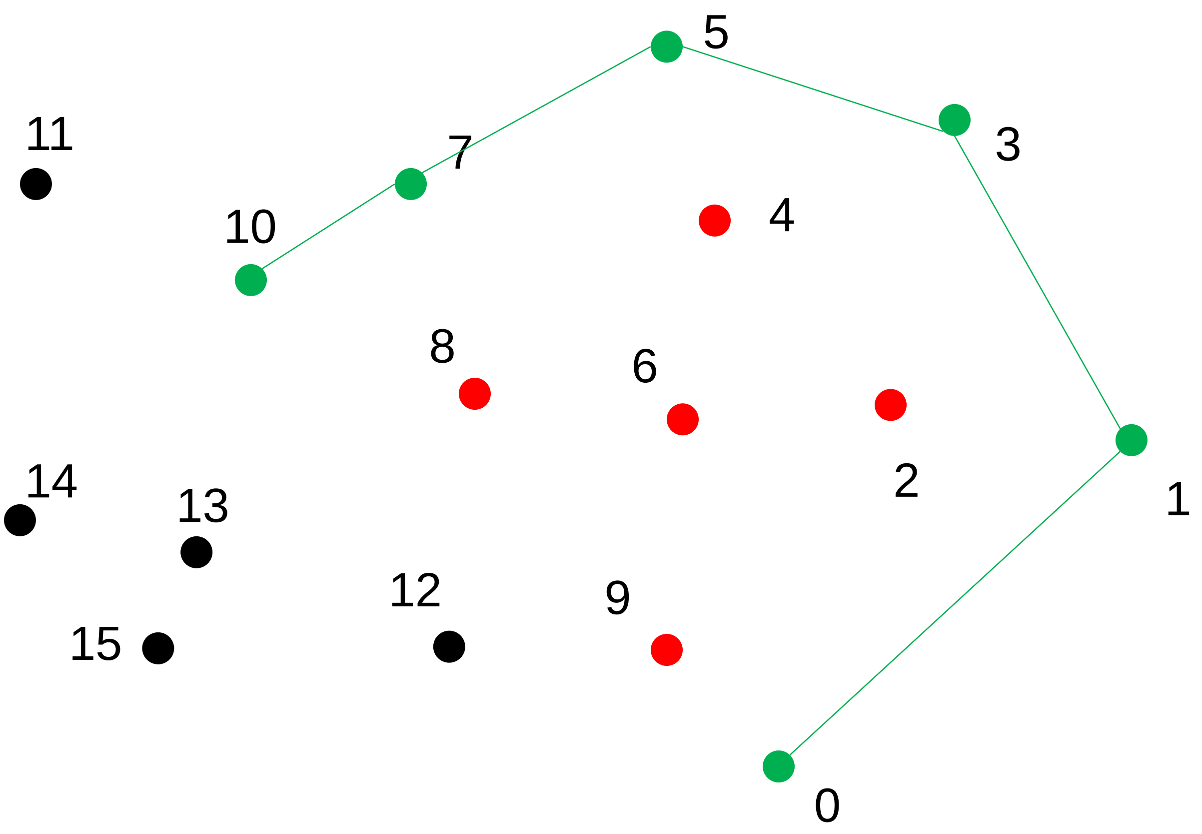
Graham Scan Algorithm



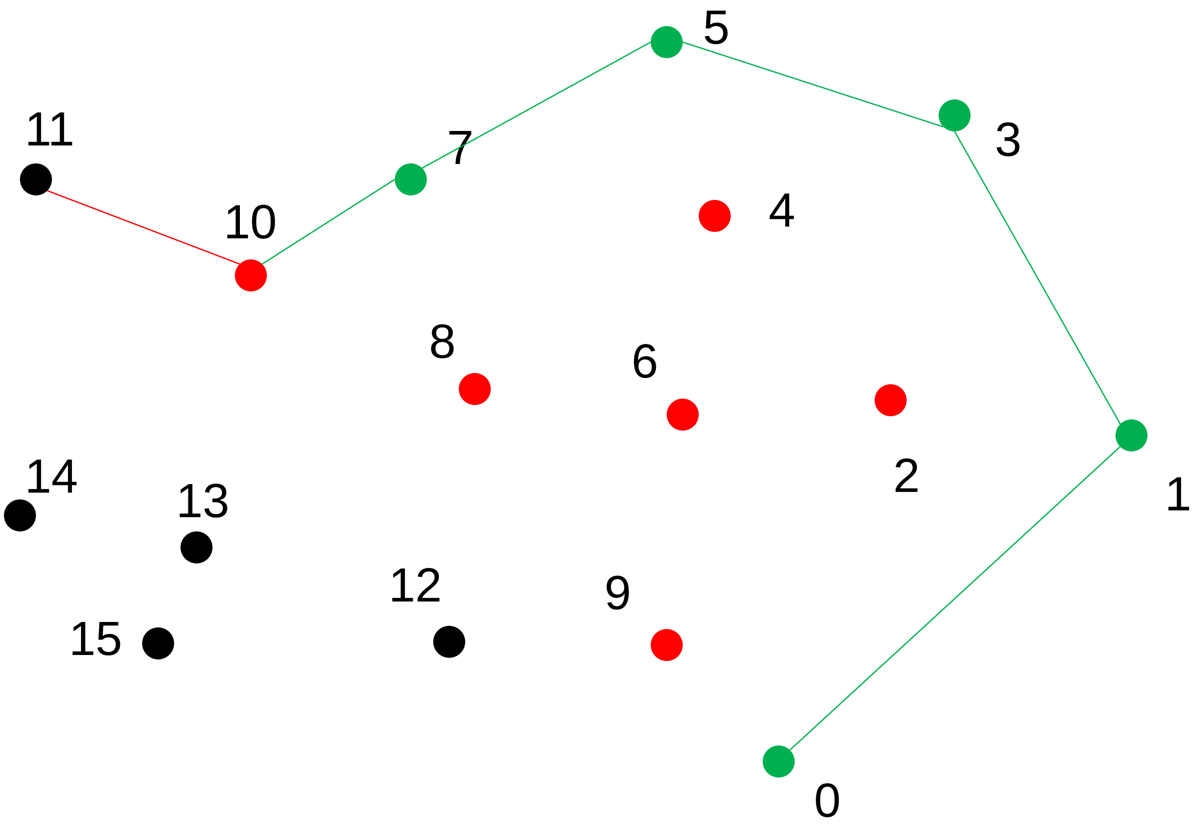
Graham Scan Algorithm



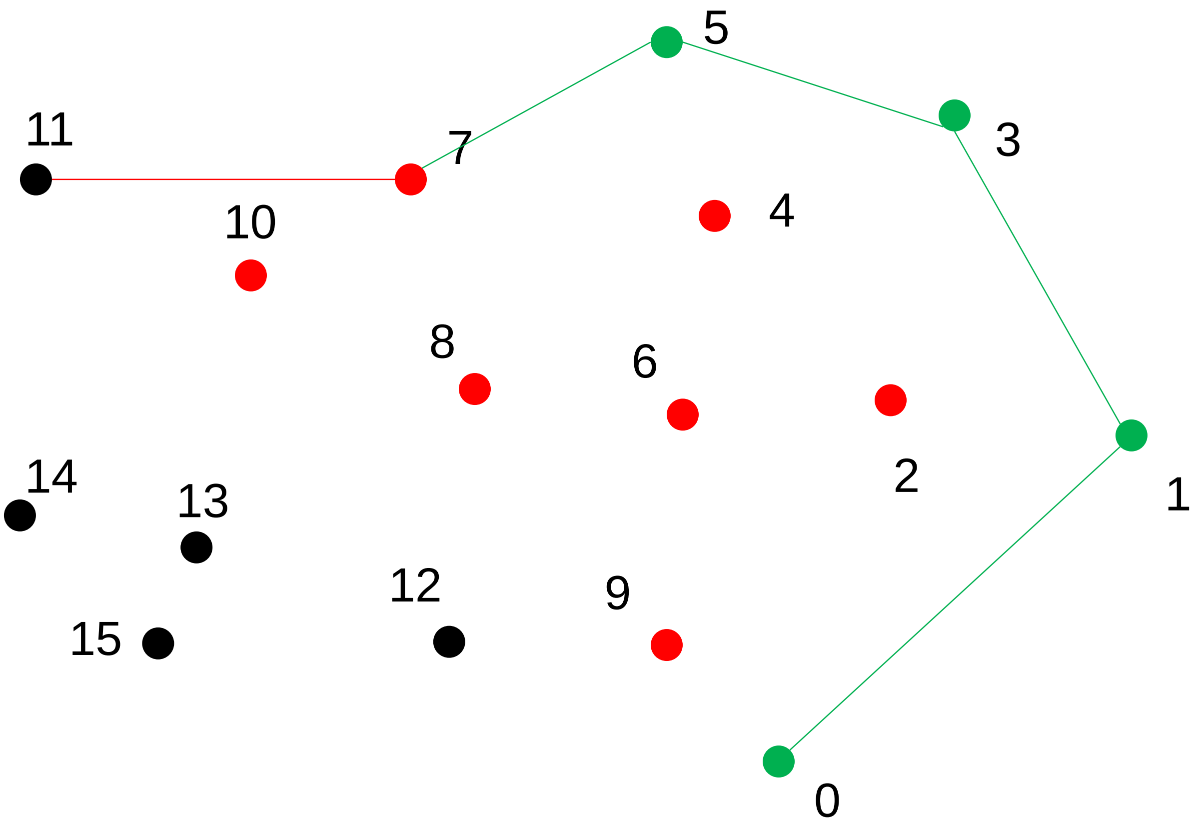
Graham Scan Algorithm



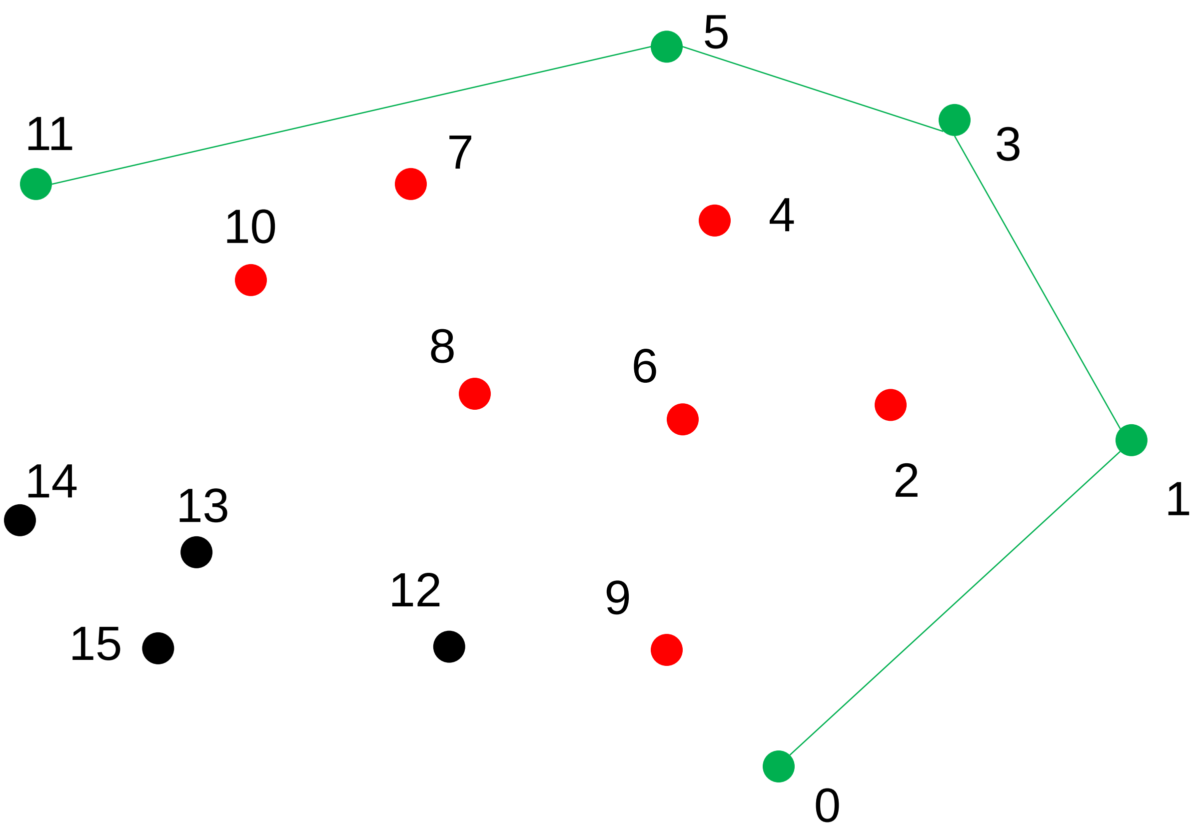
Graham Scan Algorithm



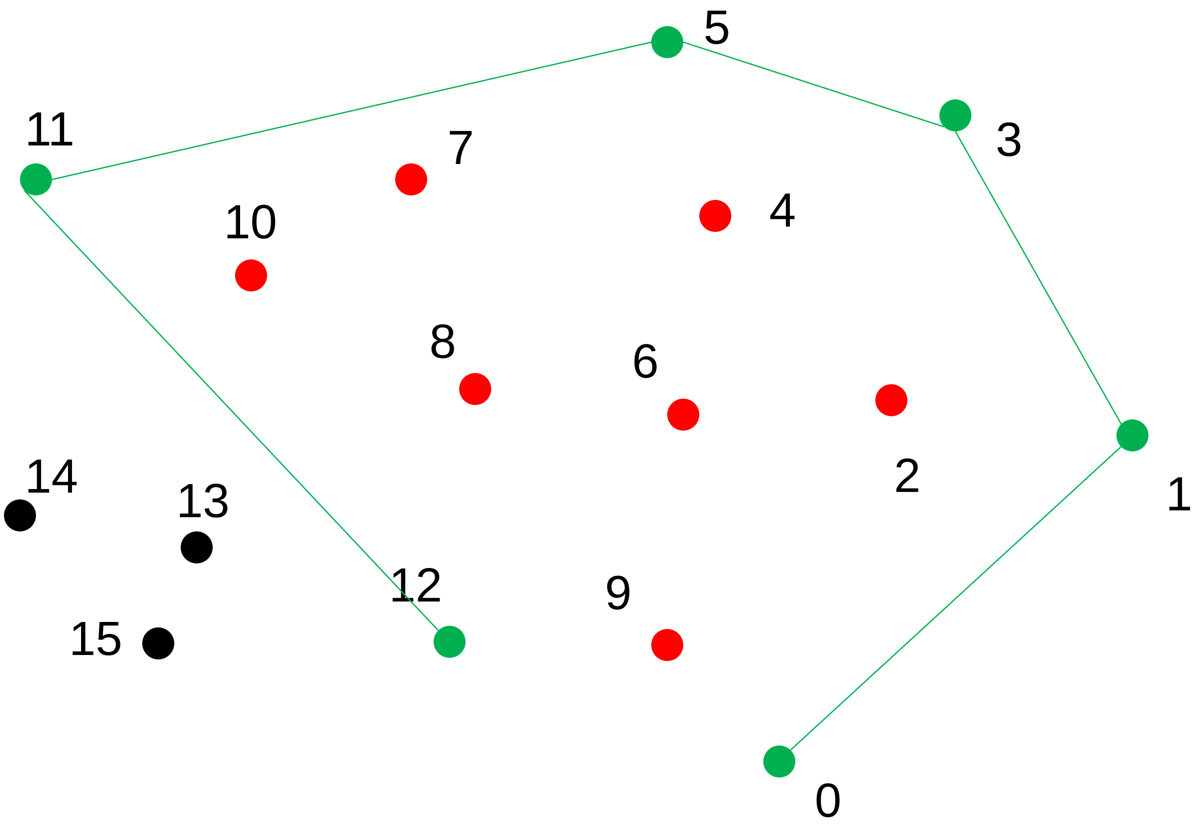
Graham Scan Algorithm



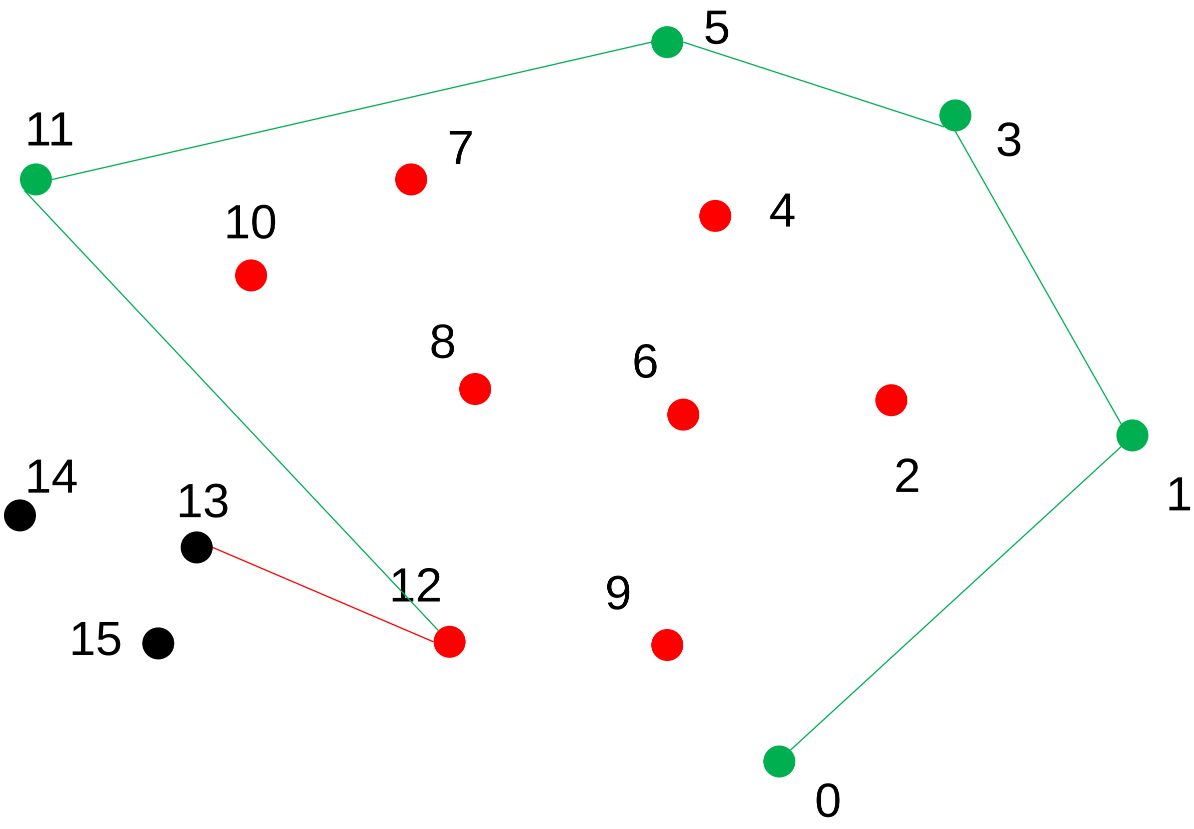
Graham Scan Algorithm



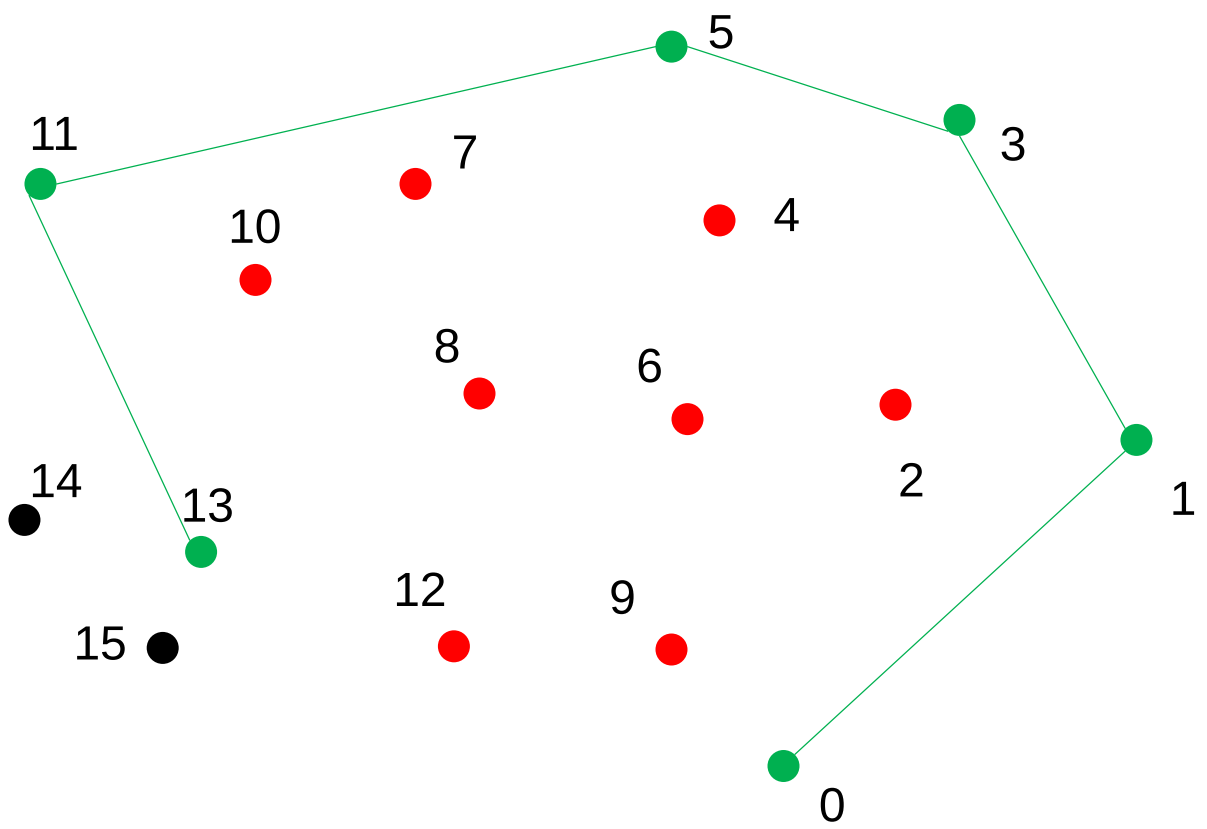
Graham Scan Algorithm



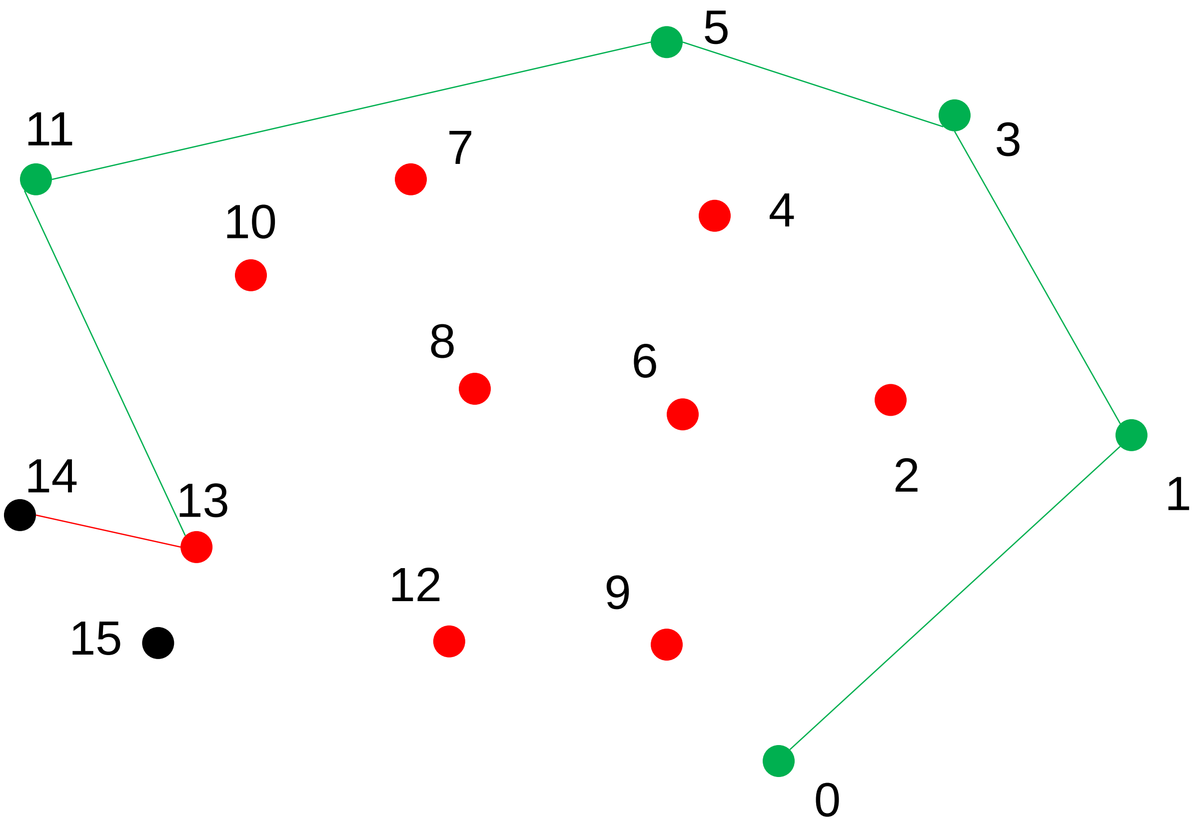
Graham Scan Algorithm



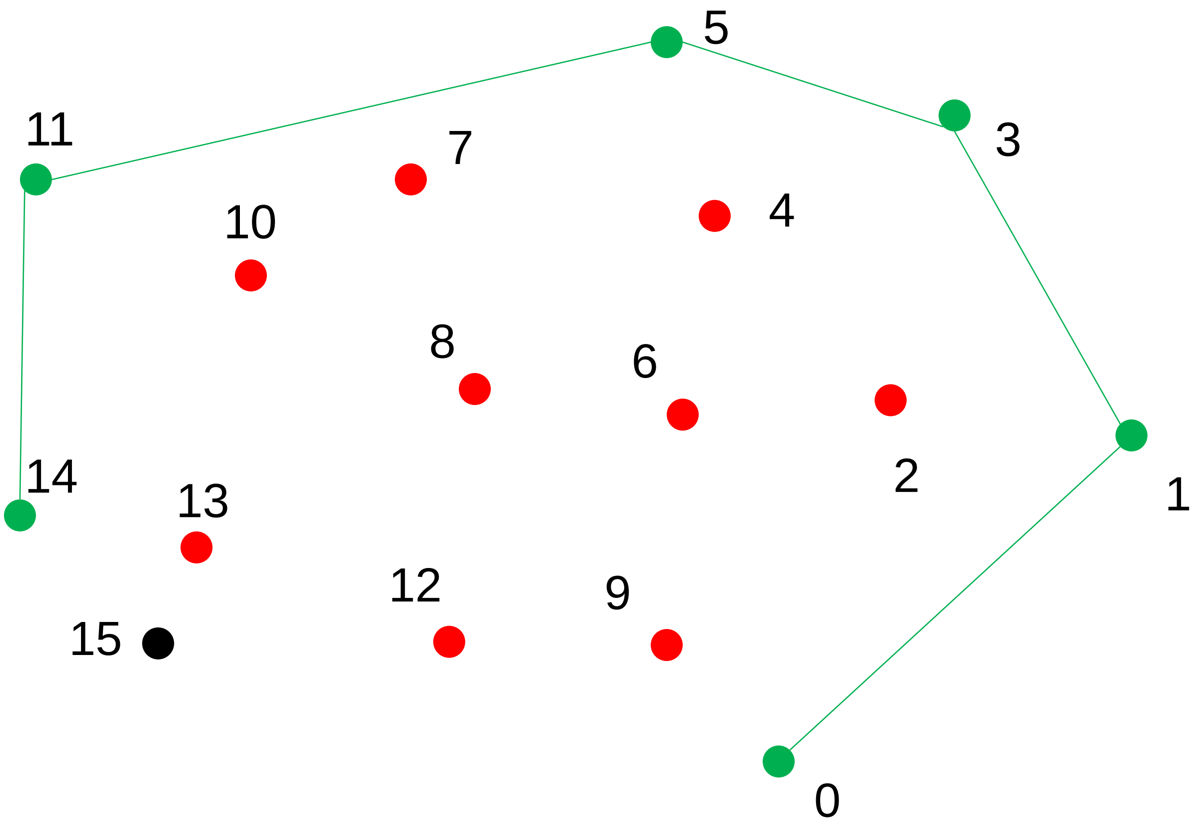
Graham Scan Algorithm



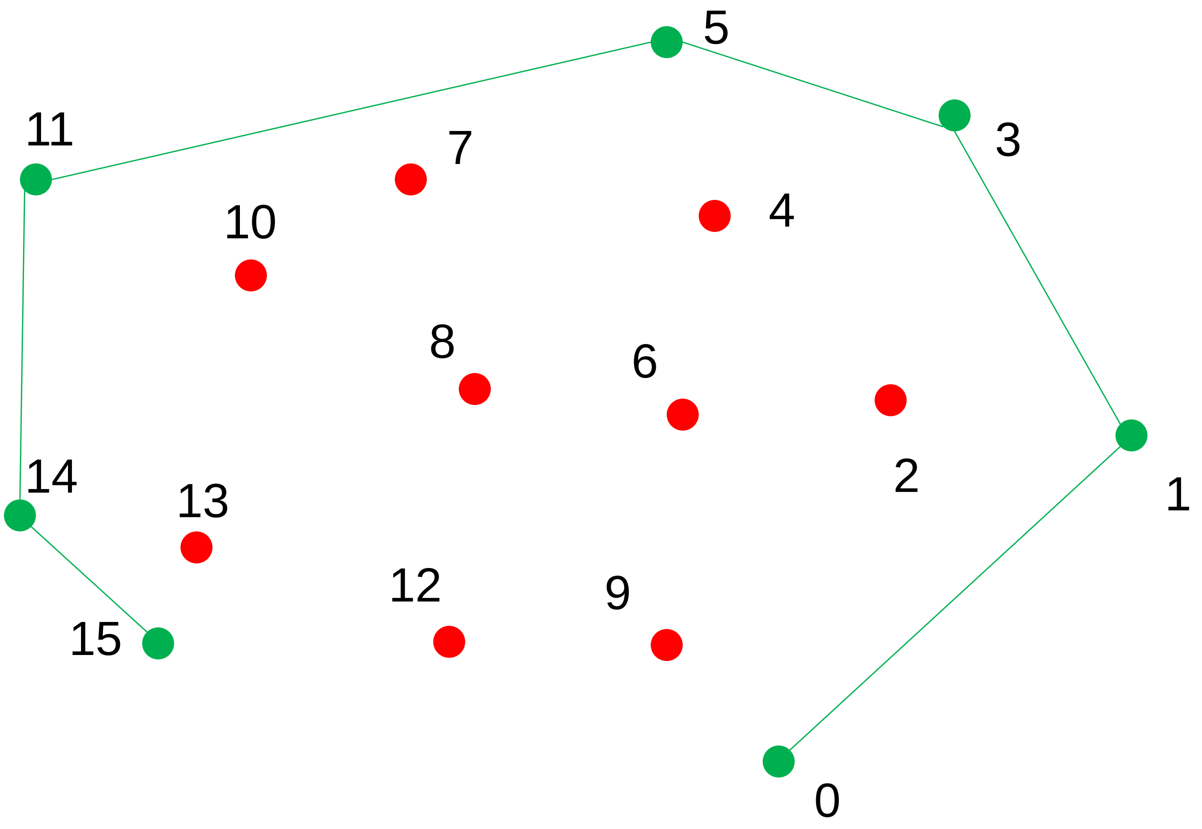
Graham Scan Algorithm



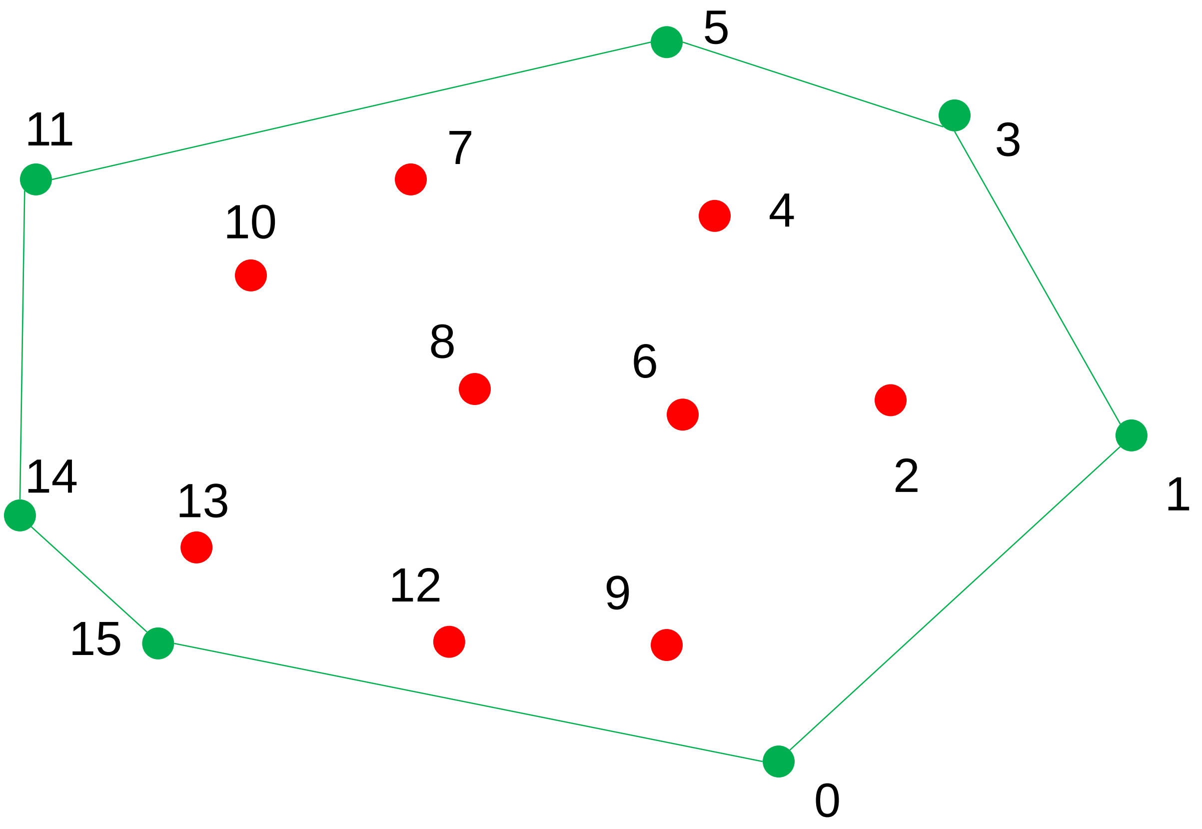
Graham Scan Algorithm



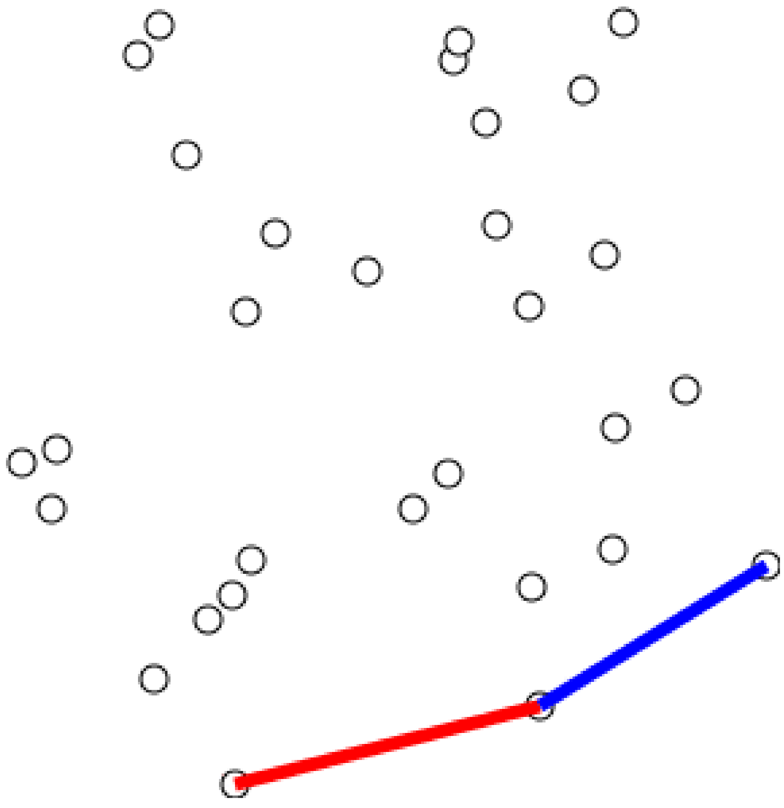
Graham Scan Algorithm



Graham Scan Algorithm



Example

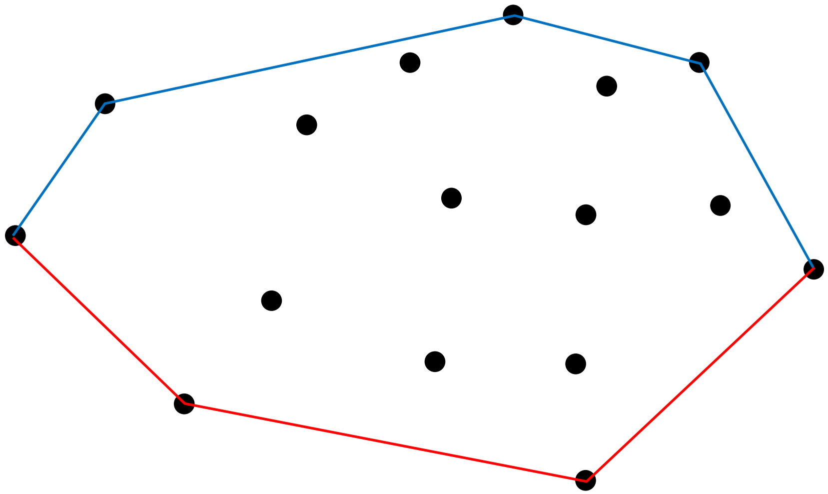


Graham Scan Pseudo Code

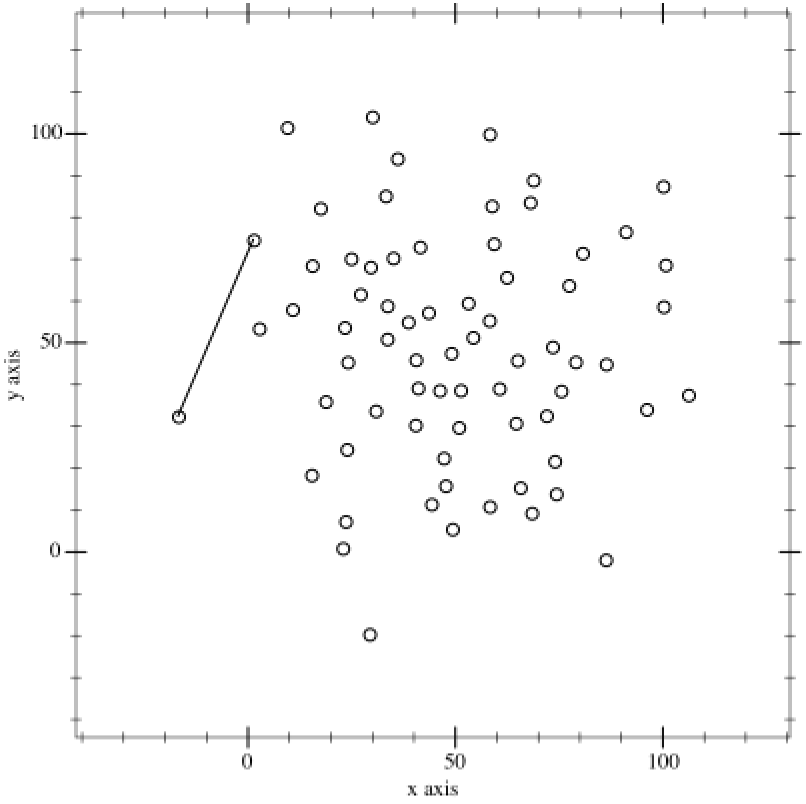
- Select the point with minimum y
- Sort all points in CCW order $\{p_0, p_1, \dots, p_n\}$
- $S = \{p_0, p_1\}$
- For $i = 2$ to n
  - While $|S| > 2$ && p_i is to the right of S_{-2}, S_{-1}
    - S.pop
  - S.push(p_i)

Monotone Chain Algorithm

- Has some similarities with Graham scan algorithm
- Instead of sorting in CCW order, it sorts by one coordinate (e.g., x-coordinates)



Example

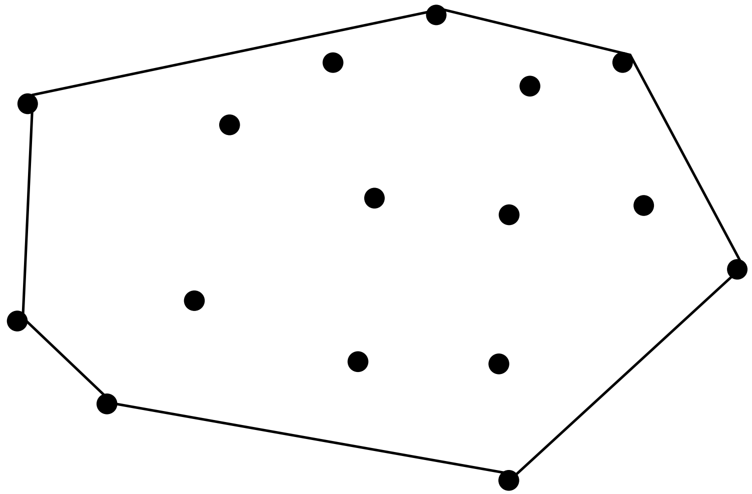


Pseudo Code

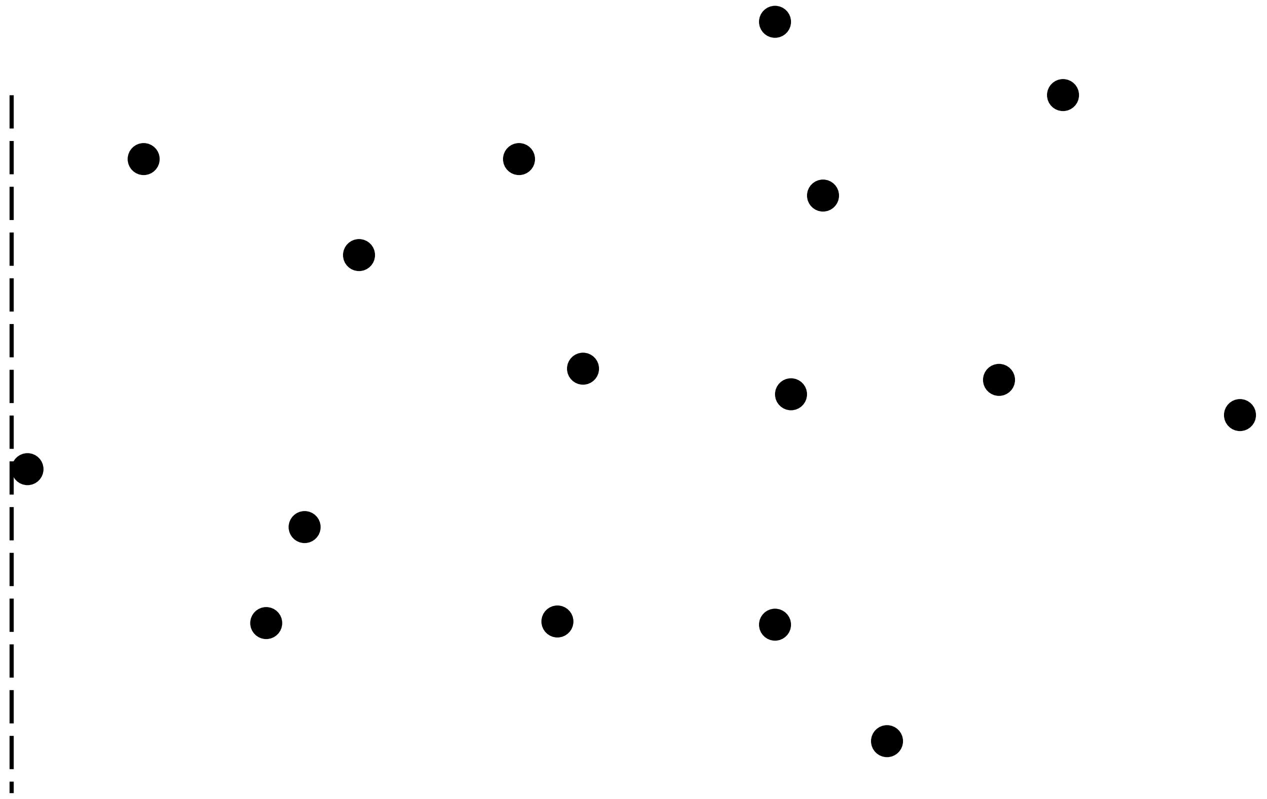
- Sort S by x
- $U = \{S_0\}$
- For $i = 1$ to n
  - while $|U| > 1$ && S_i is to the left of $\overrightarrow{U_{-2}U_{-1}}$
    - U.pop
  - U.push(S_i)
- $L = \{S_0\}$
  - While $|L| > 1$ && S_i is to the right of $\overrightarrow{L_{-2}L_{-1}}$
    - L.pop
  - L.push(S_i)

Gift Wrapping Algorithm

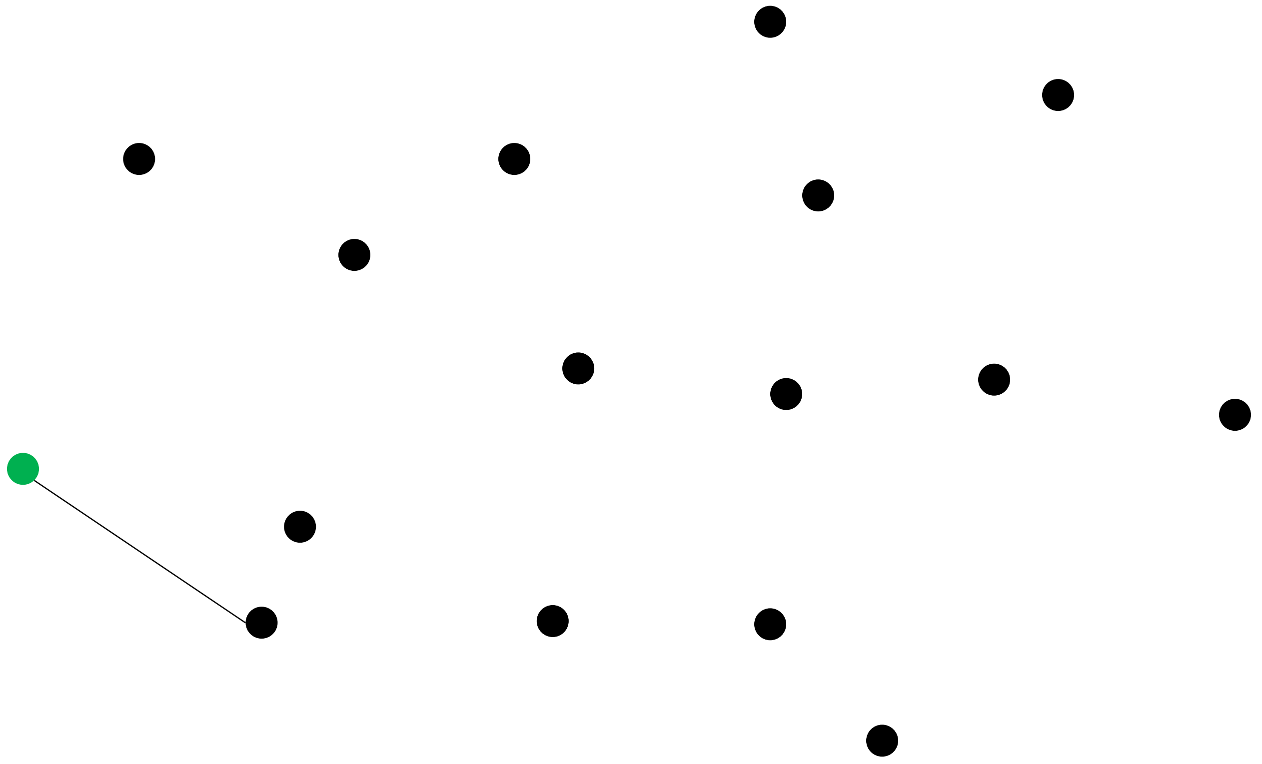
- Start with a point on the convex hull
- Find more points on the hull one at a time
- Terminate when the first point is reached back
- Also knows as Jarvi's March Algorithm



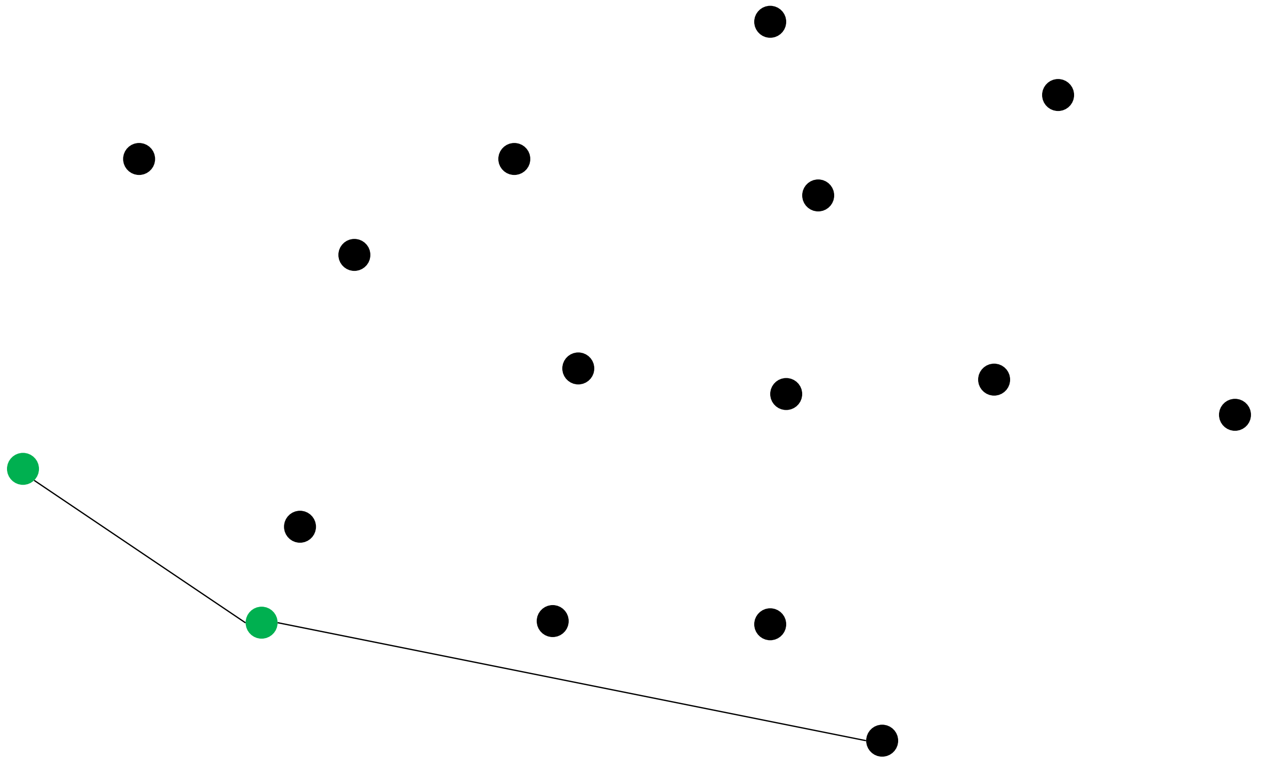
Gift Wrapping Example



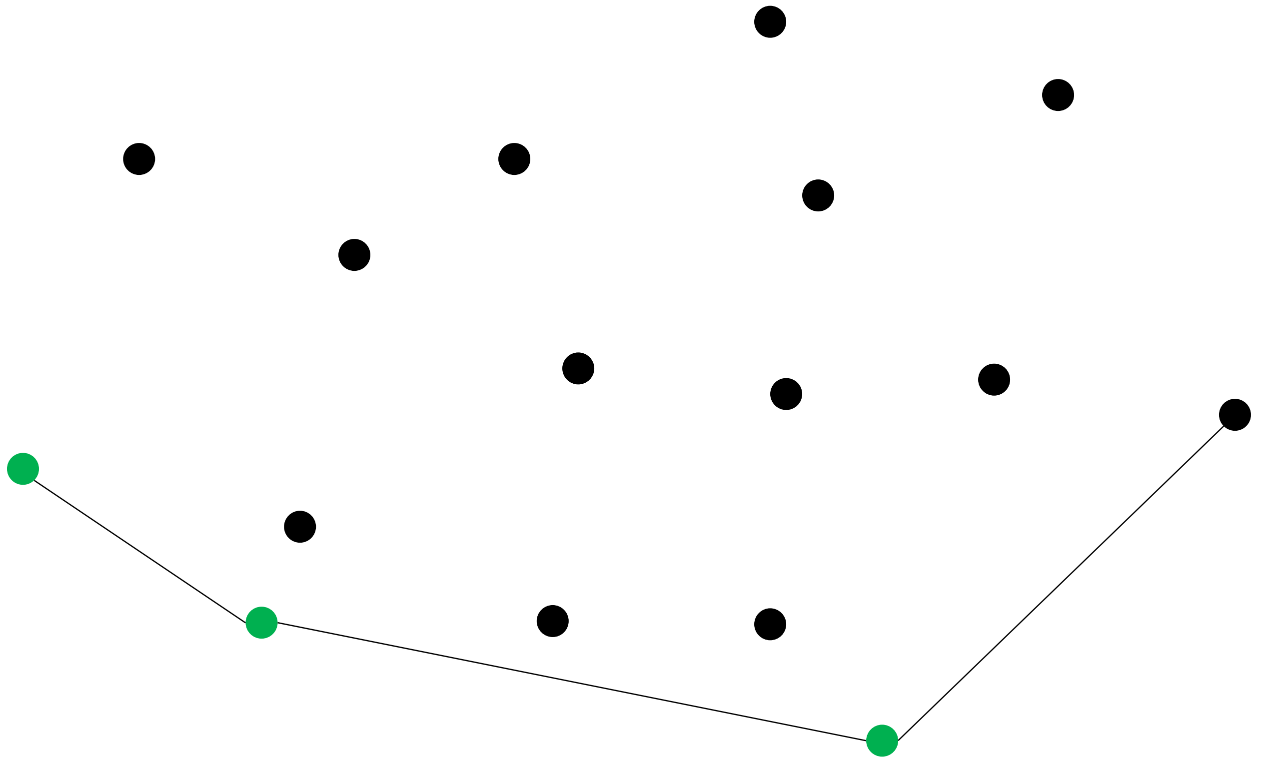
Gift Wrapping Example



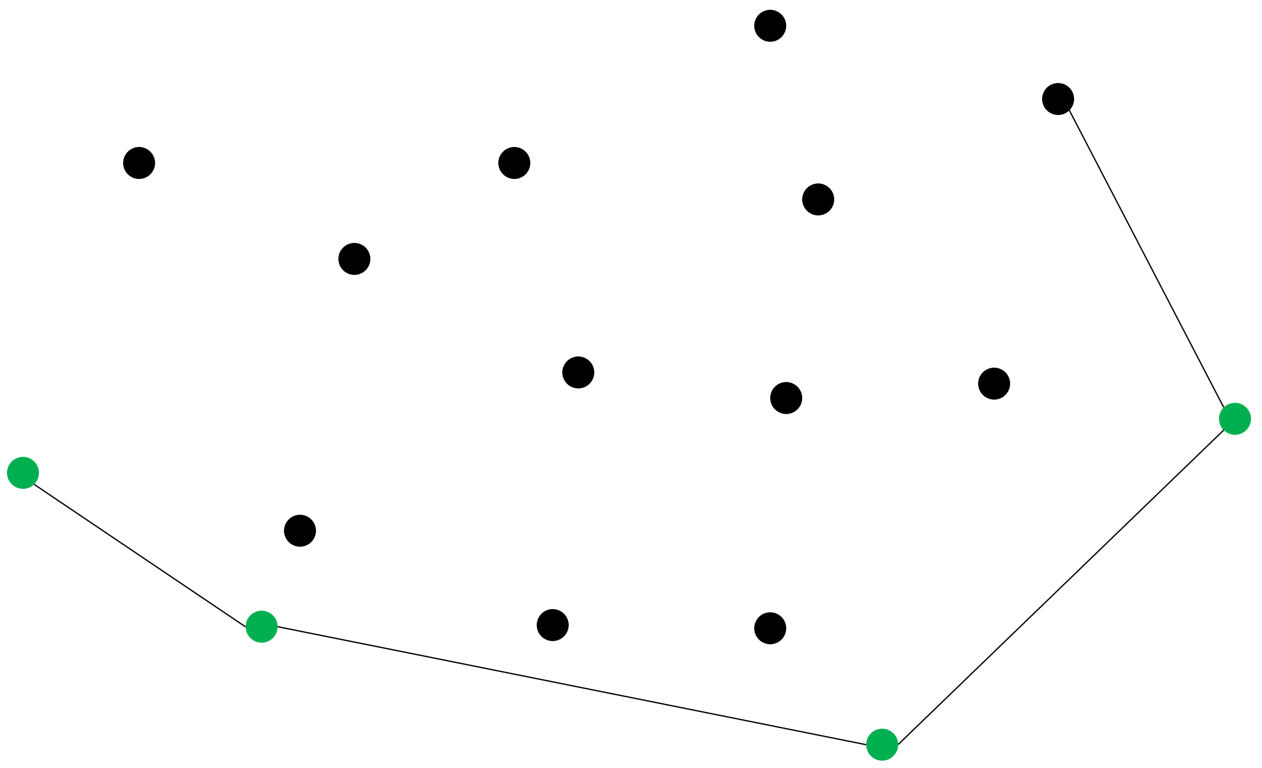
Gift Wrapping Example



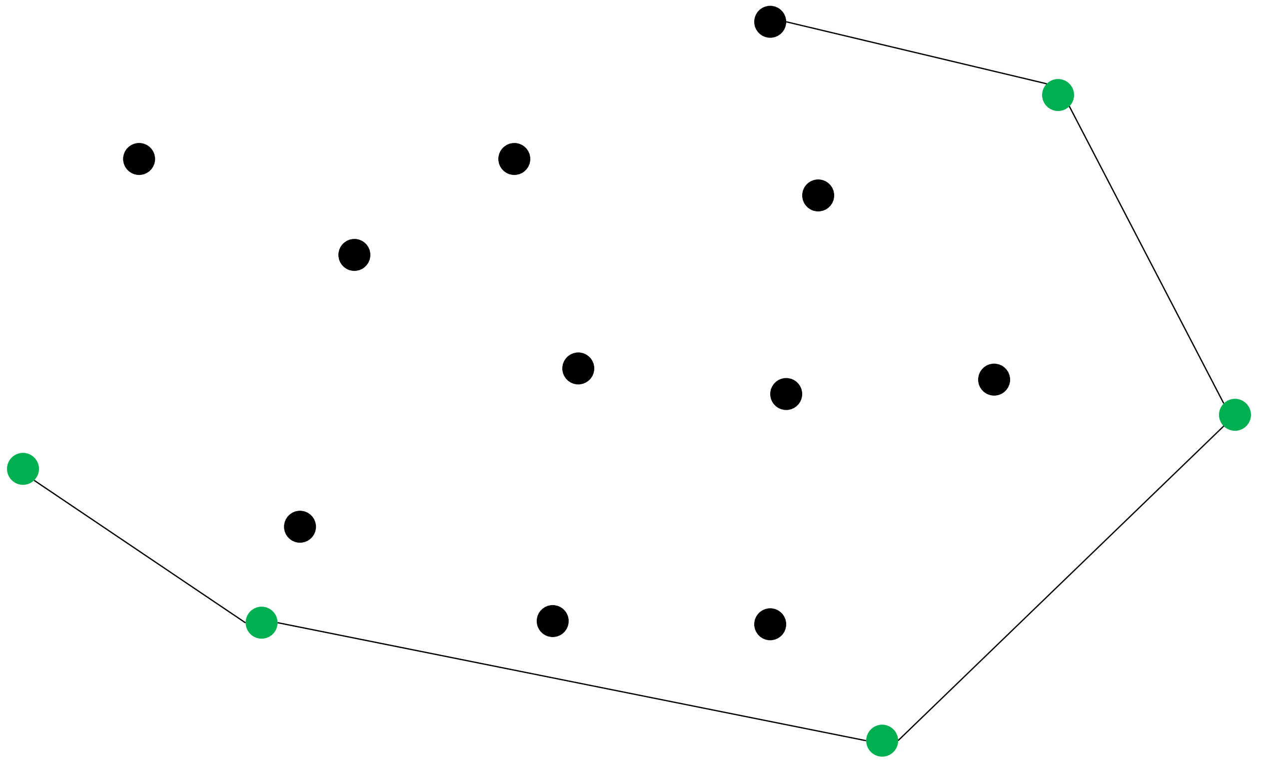
Gift Wrapping Example



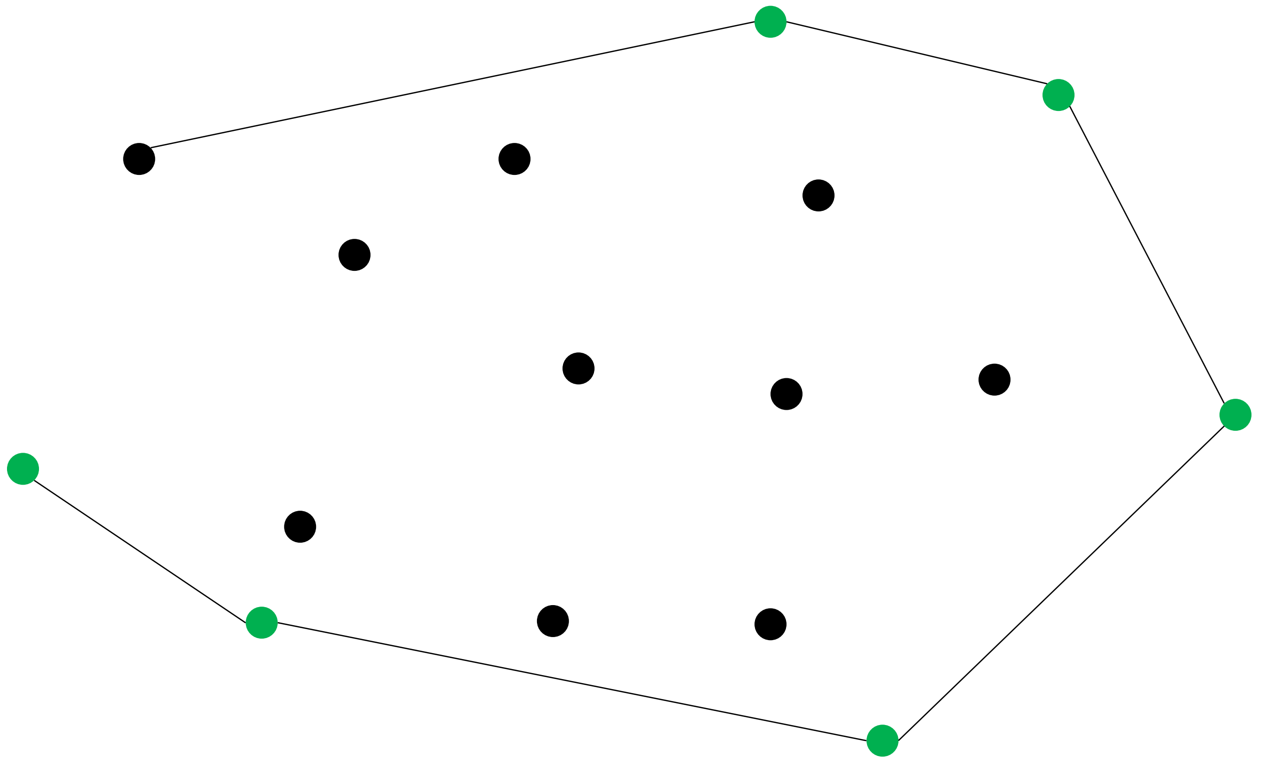
Gift Wrapping Example



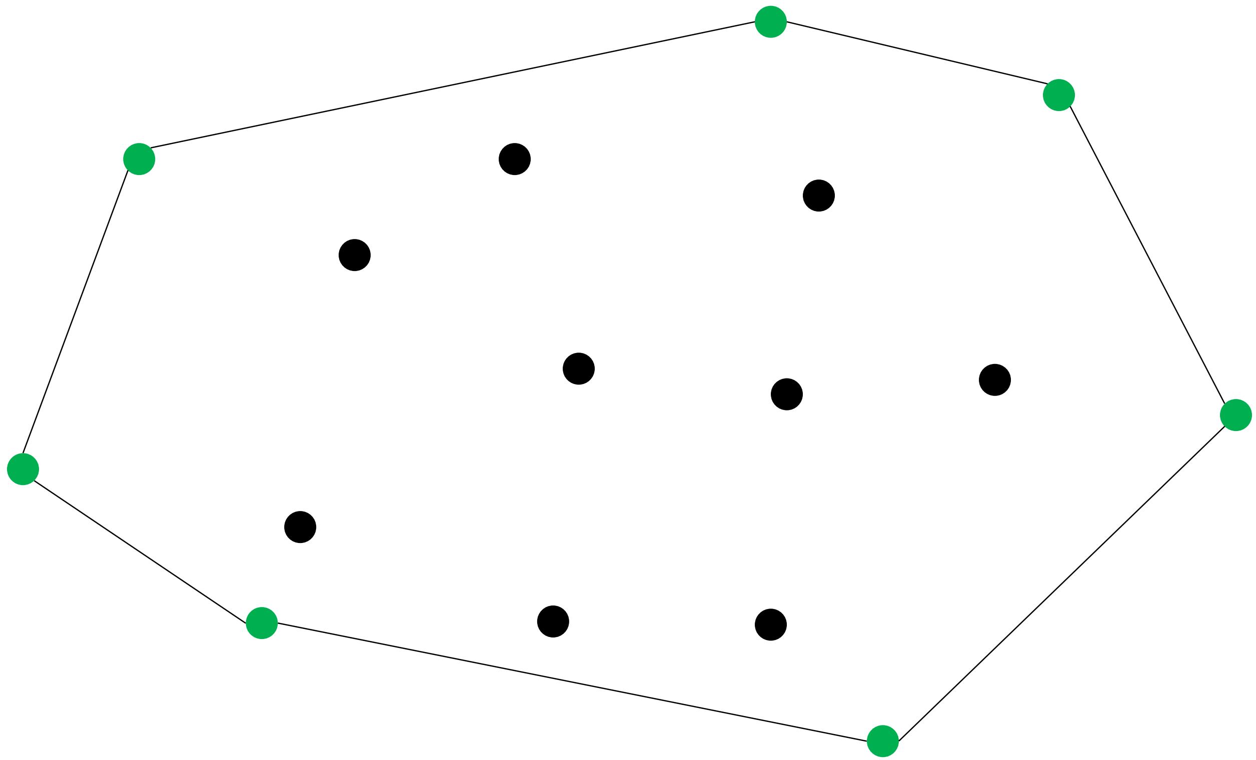
Gift Wrapping Example



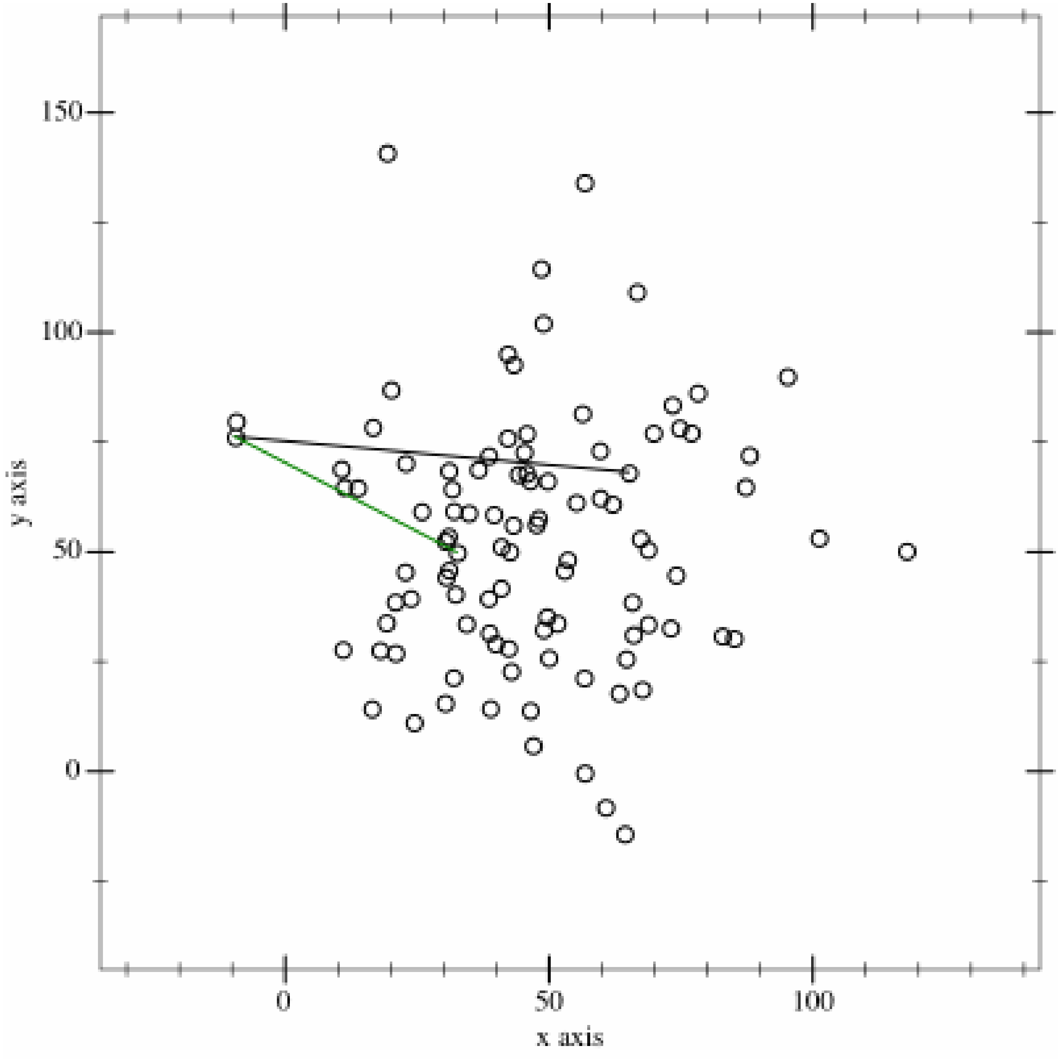
Gift Wrapping Example



Gift Wrapping Example



Gift Wrapping Example

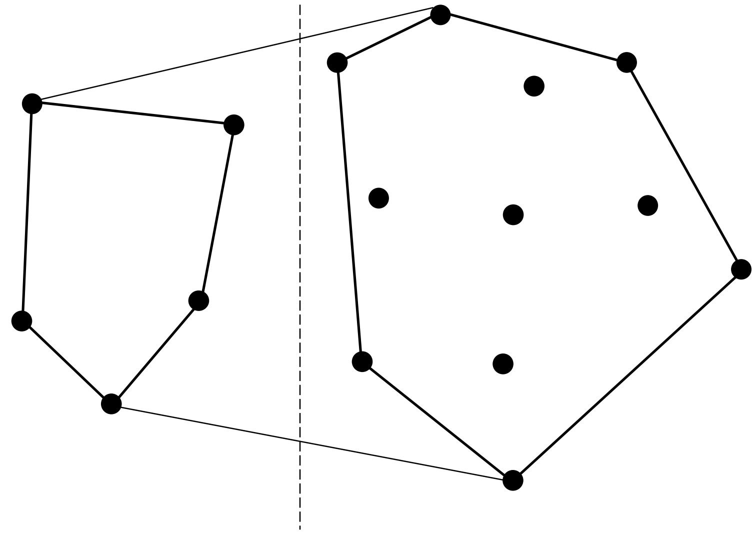


Gift Wrapping Pseudo Code

- Gift Wrapping(S)
  - CH= {}
  - CH << Left most point
  - do
    - Start point = CH.last
    - End point = CH[0]
    - For each point s ∈ S
      - If Start point = End Point OR
        s is to the left of $\overrightarrow{Start\ point, End\ point}$
      - End point = s
    - CH << End point
- Running time $O(n \cdot k)$

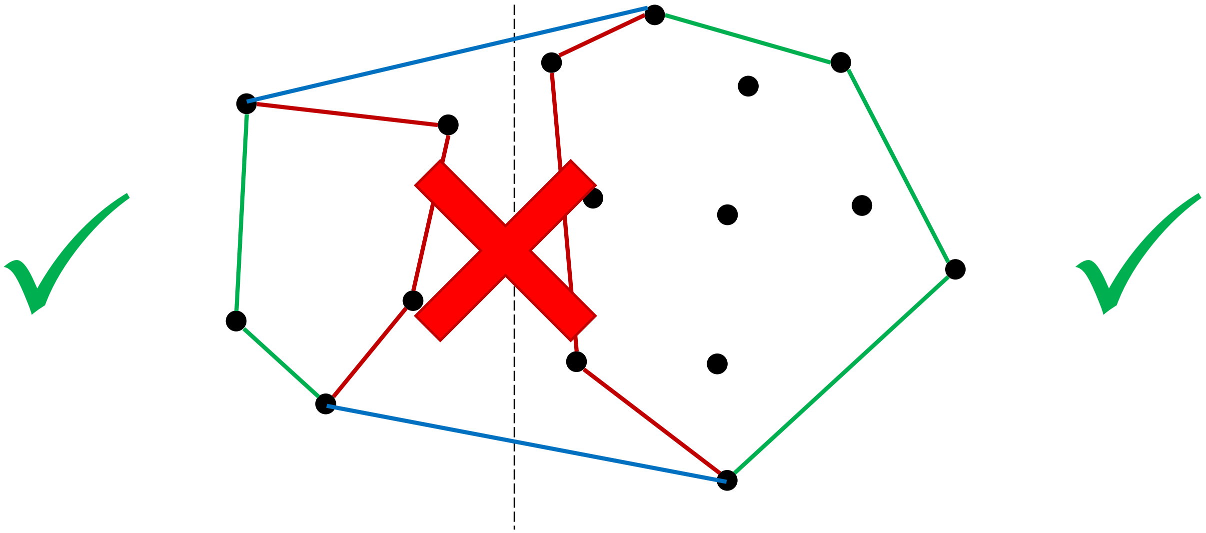
Divide & Conquer Convex Hull

- ConvexHull(S)
  - Splits S into two subsets S1 and S2
  - Ch1 = ConvexHull(S1)
  - Ch2 = ConvexHull(S2)
  - Return Merge(Ch1, Ch2)

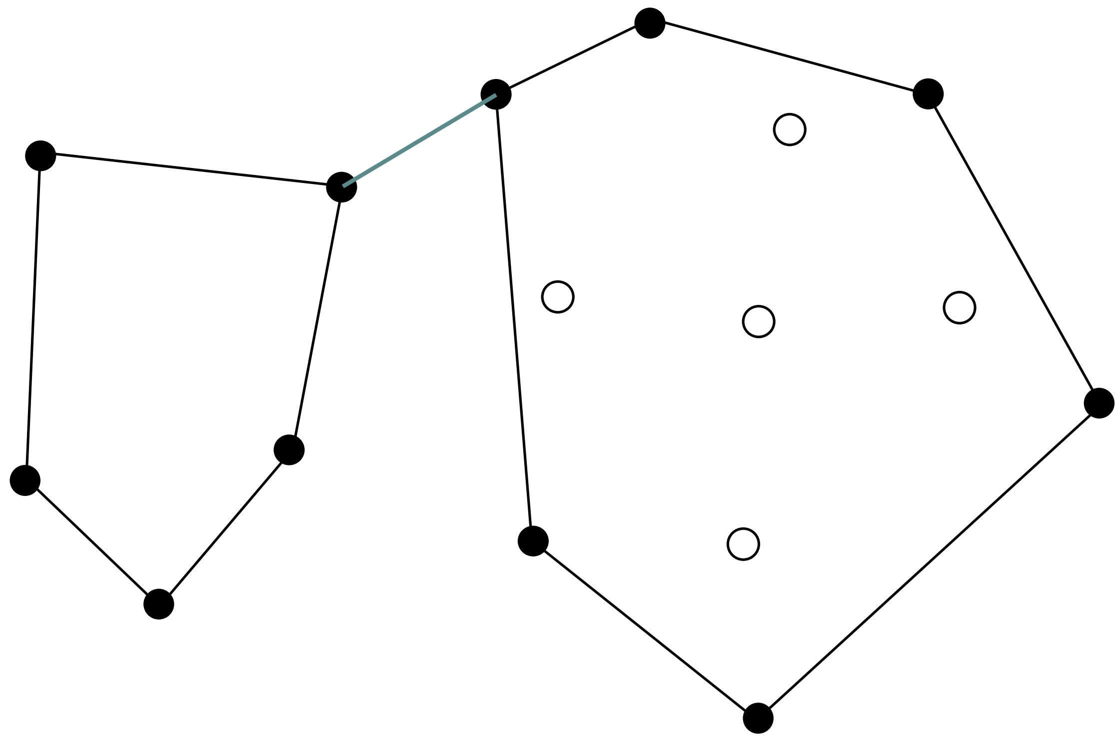


Divide & Conquer Convex Hull

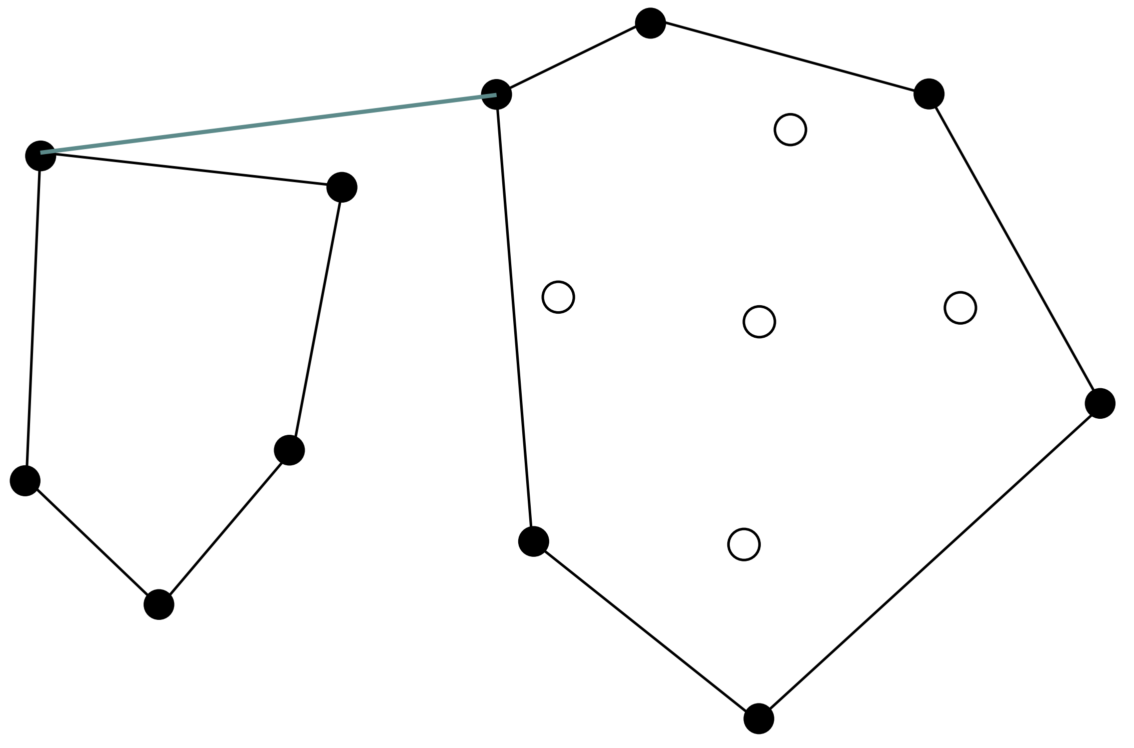
- ConvexHull(S)
  - Splits S into two subsets S1 and S2
  - Ch1 = ConvexHull(S1)
  - Ch2 = ConvexHull(S2)
  - Return Merge(Ch1, Ch2)



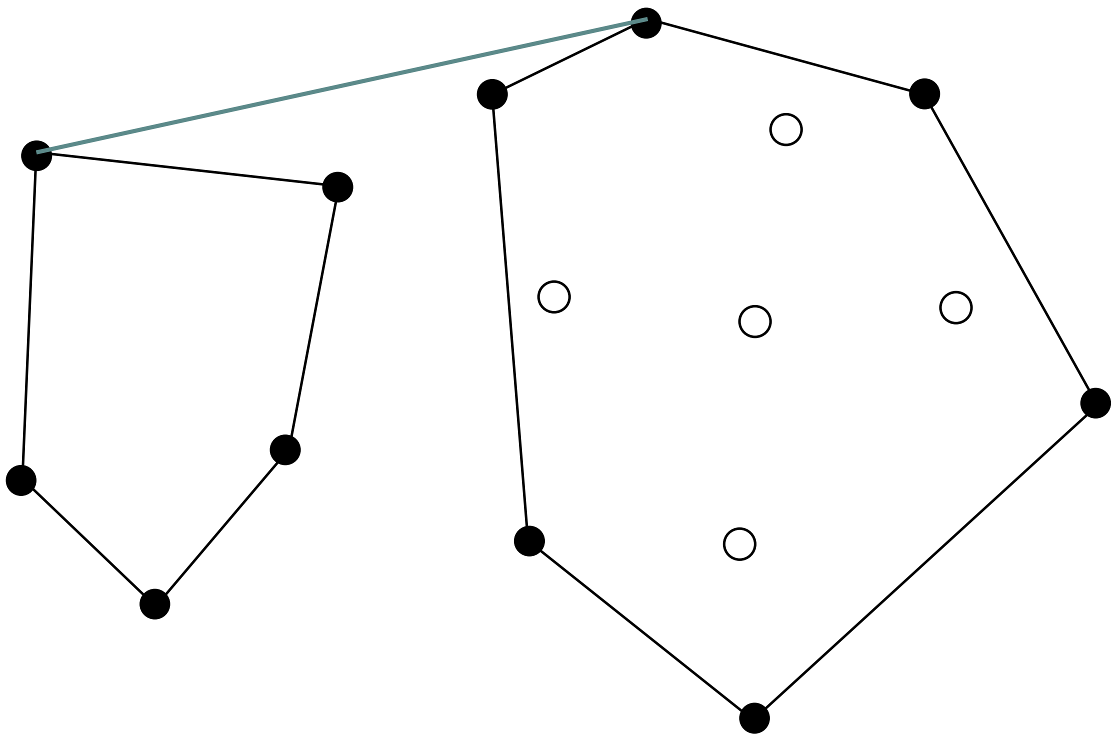
Merge: Upper Tangent



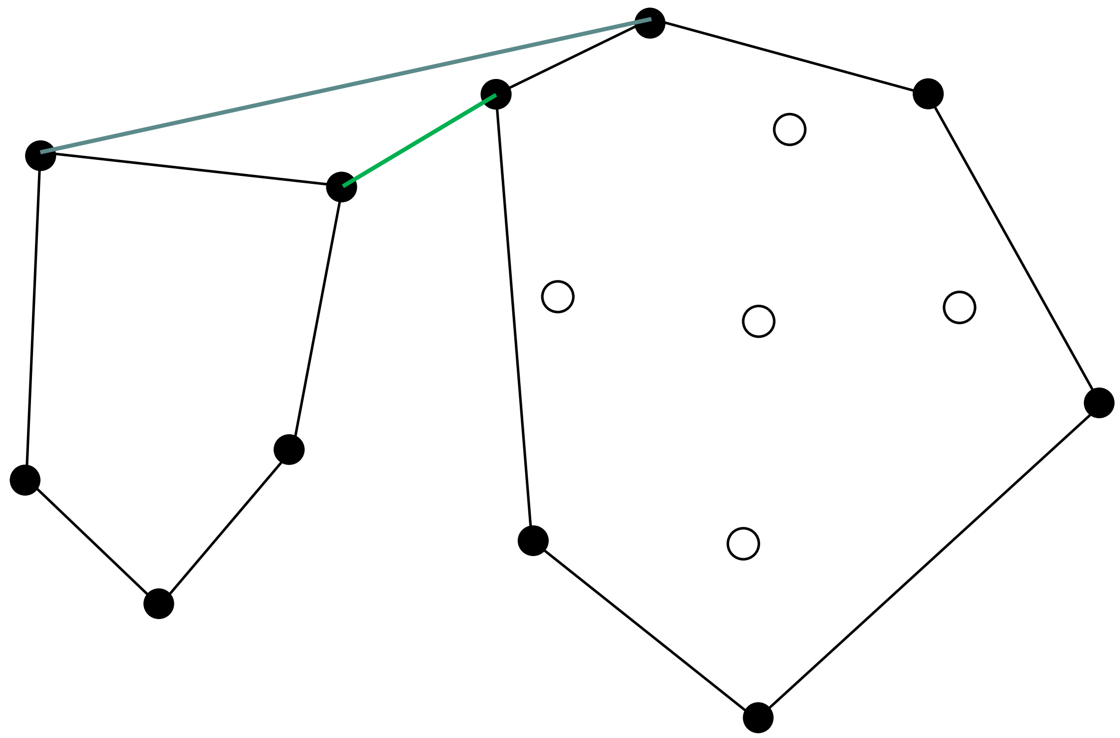
Merge: Upper Tangent



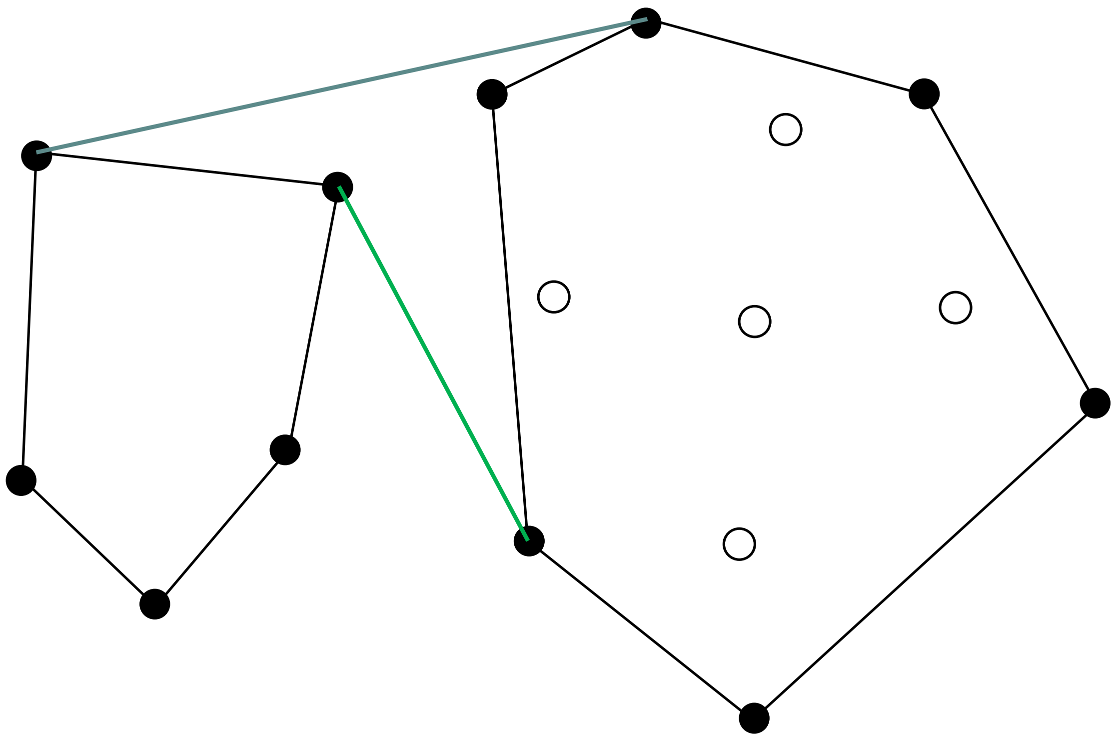
Merge: Upper Tangent



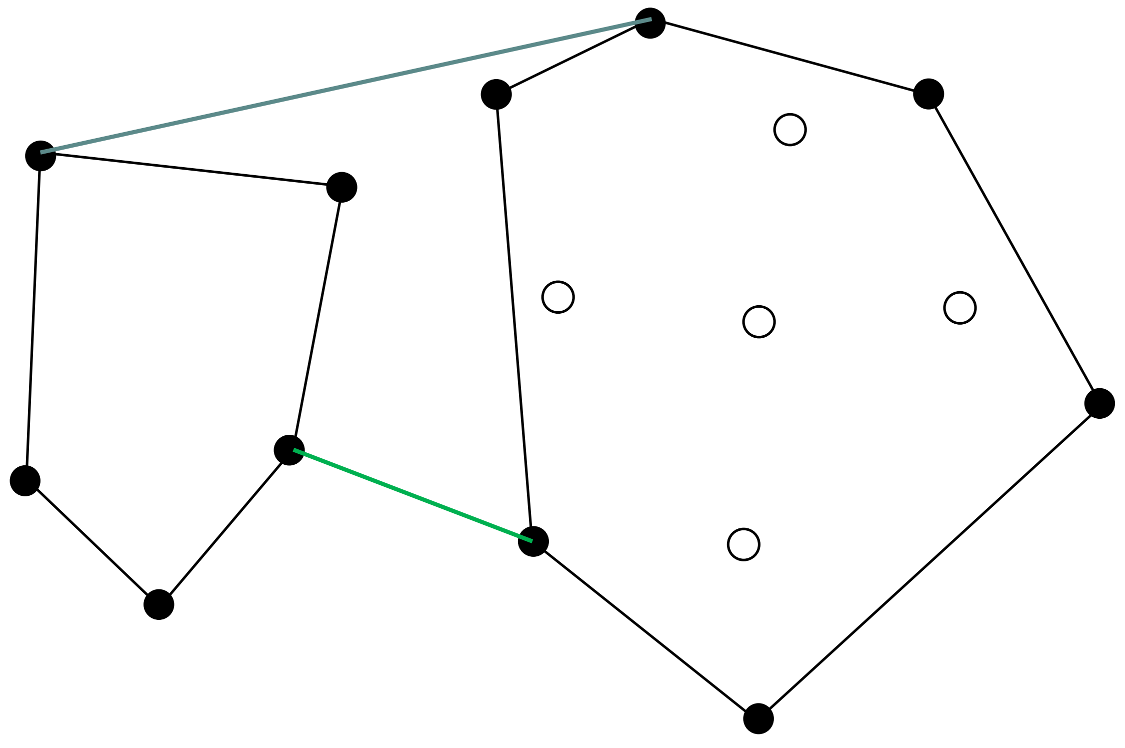
Merge: Lower Tangent



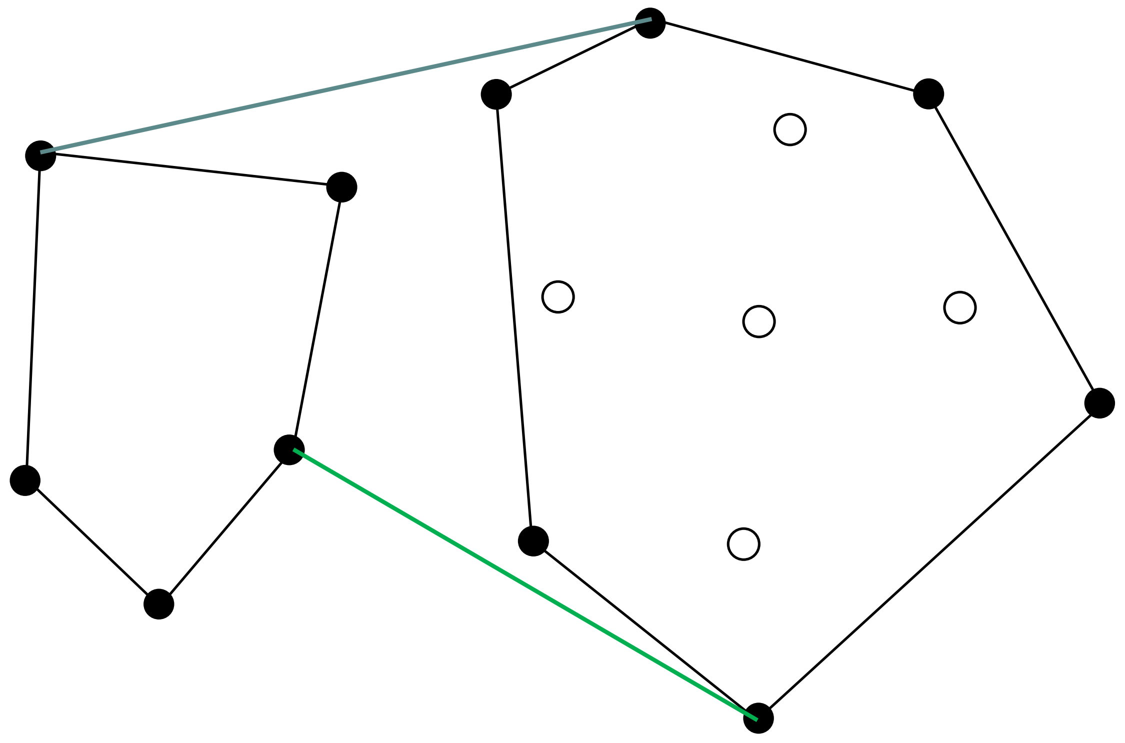
Merge: Lower Tangent



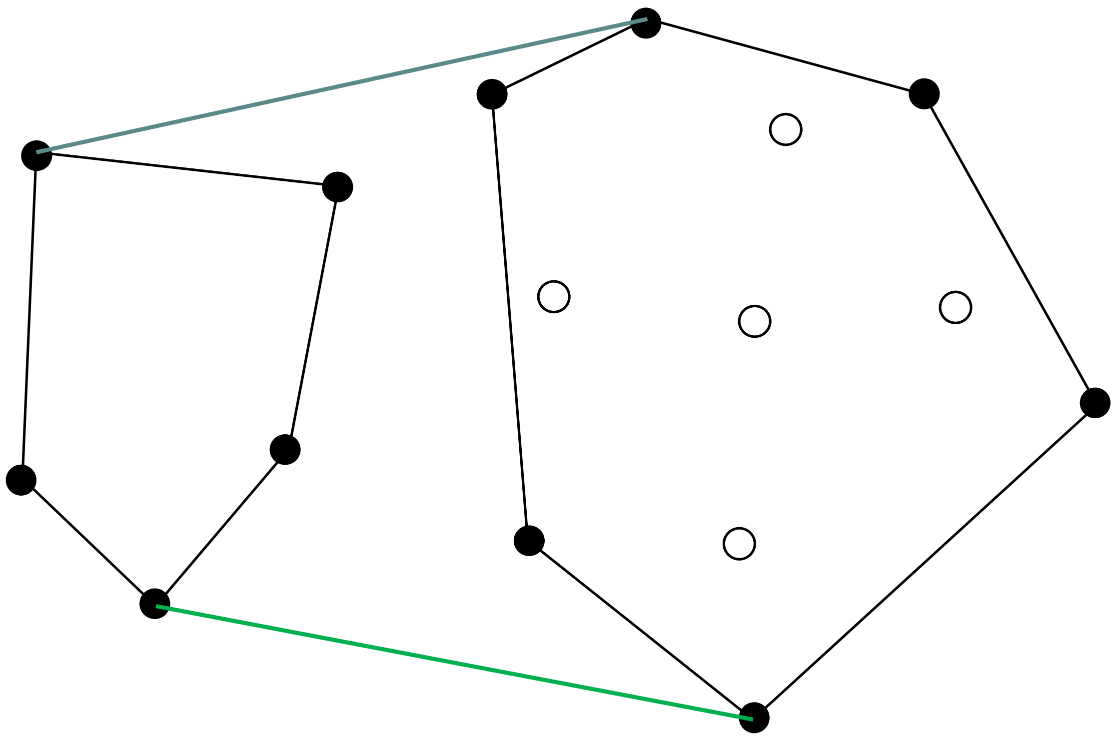
Merge: Lower Tangent



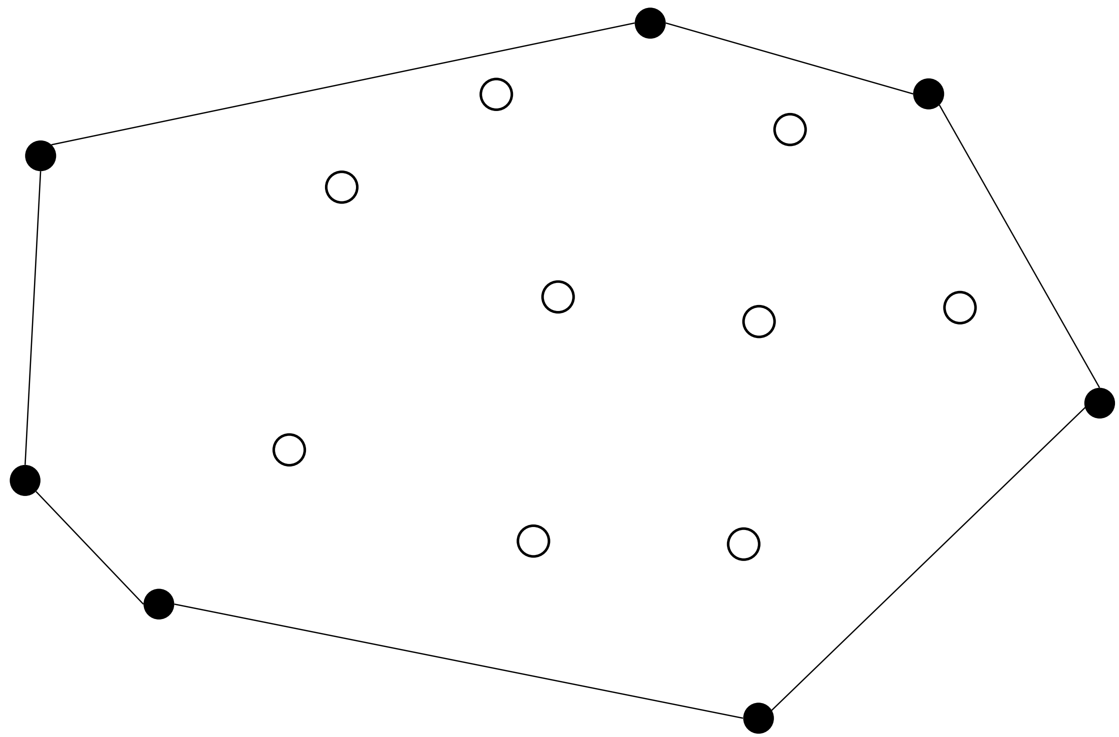
Merge: Lower Tangent



Merge: Lower Tangent



Merge: Lower Tangent



Merge Step

- Upper Tangent(L, R)
  - P_i = Right most point in L
  - P_j = Left most point in R
  - Do
    - Done = true
    - While P_{i+1} is to the right of $\overrightarrow{p_j p_i}$
      - $i++$; done = false
    - While P_{j-1} is to the left of $\overrightarrow{p_i p_j}$
      - $j--$; done = false

Analysis

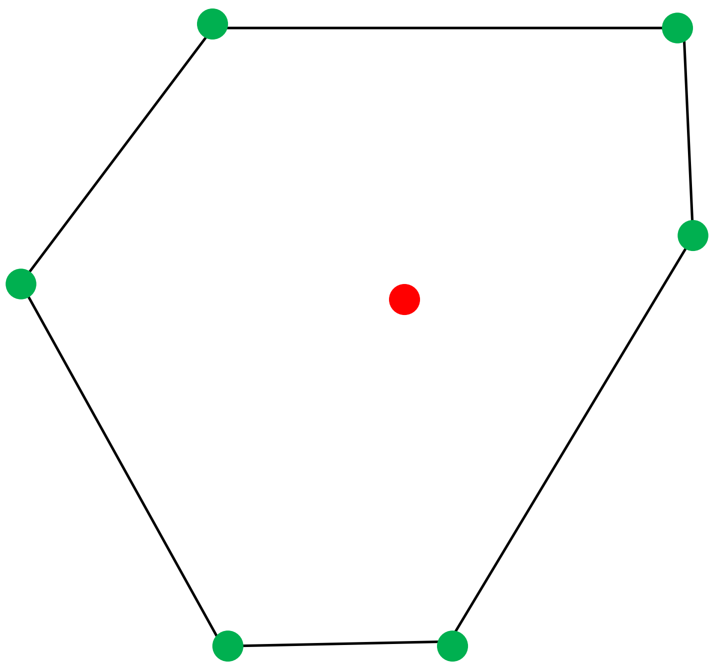
- Sort step: $O(n \cdot \log n)$
- Merge step: $O(n)$
- Recursive part
  - $T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n$
  - $T(n) = O(n \cdot \log n)$
- Overall running time $O(n \cdot \log n)$

Incremental Convex Hull

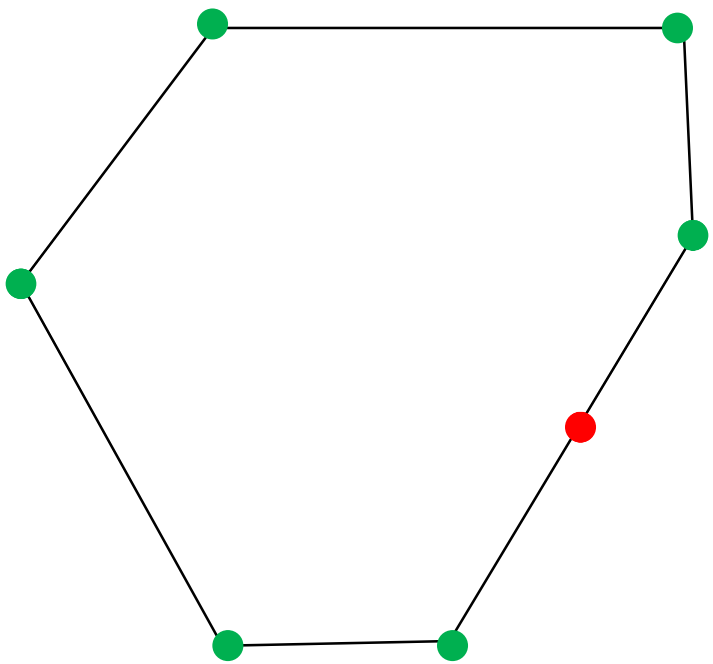
- Start with an initial convex hull
- Add one additional point to the convex hull
- Given a convex hull CH and a point p, how to compute the convex hull of {CH, p}?

- Think: Insert an element into a sorted list

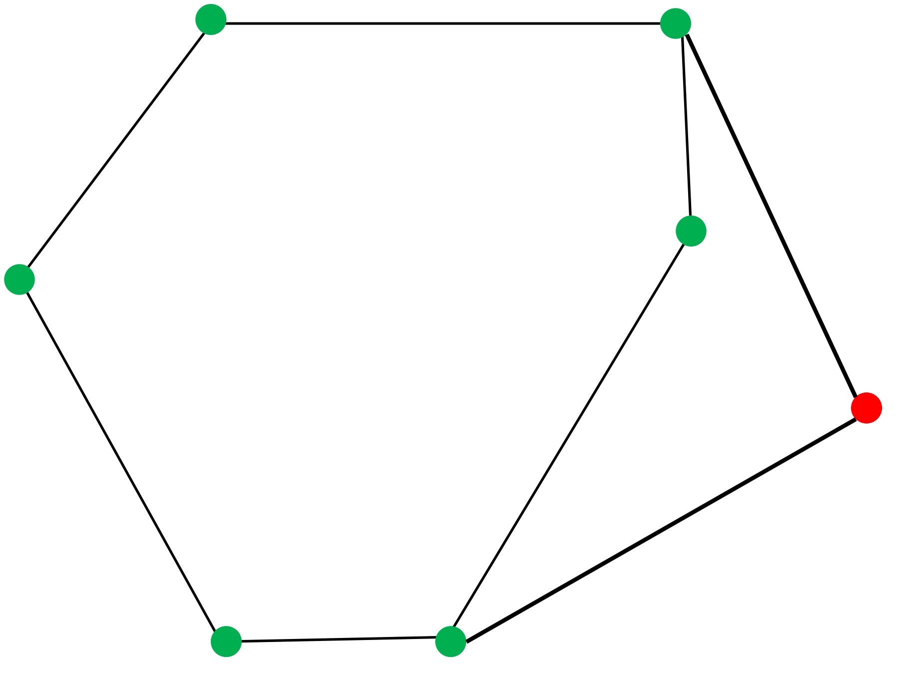
Case 1: p inside CH



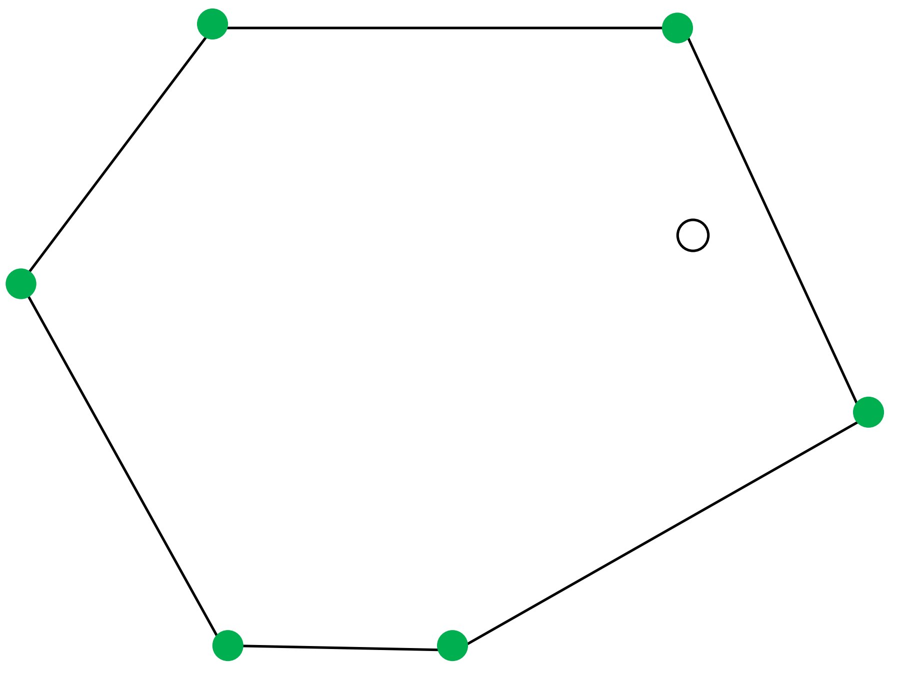
Case 2: p on CH



Case 3: p outside CH



Case 3: p outside CH



Analysis of the Insert Function

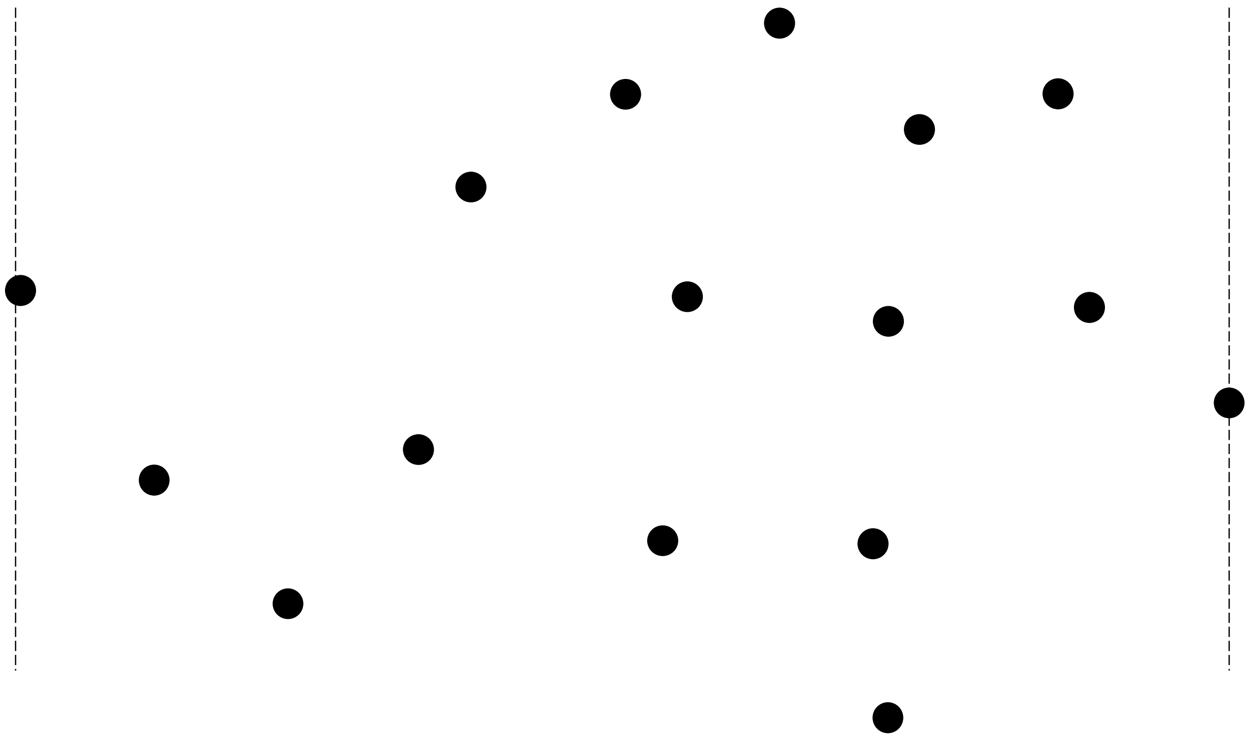
- Test whether the point is inside, outside, or on the polygon O(n)
- Find the two tangents O(n)

- A more efficient algorithm can have an amortized running time of O(log n)

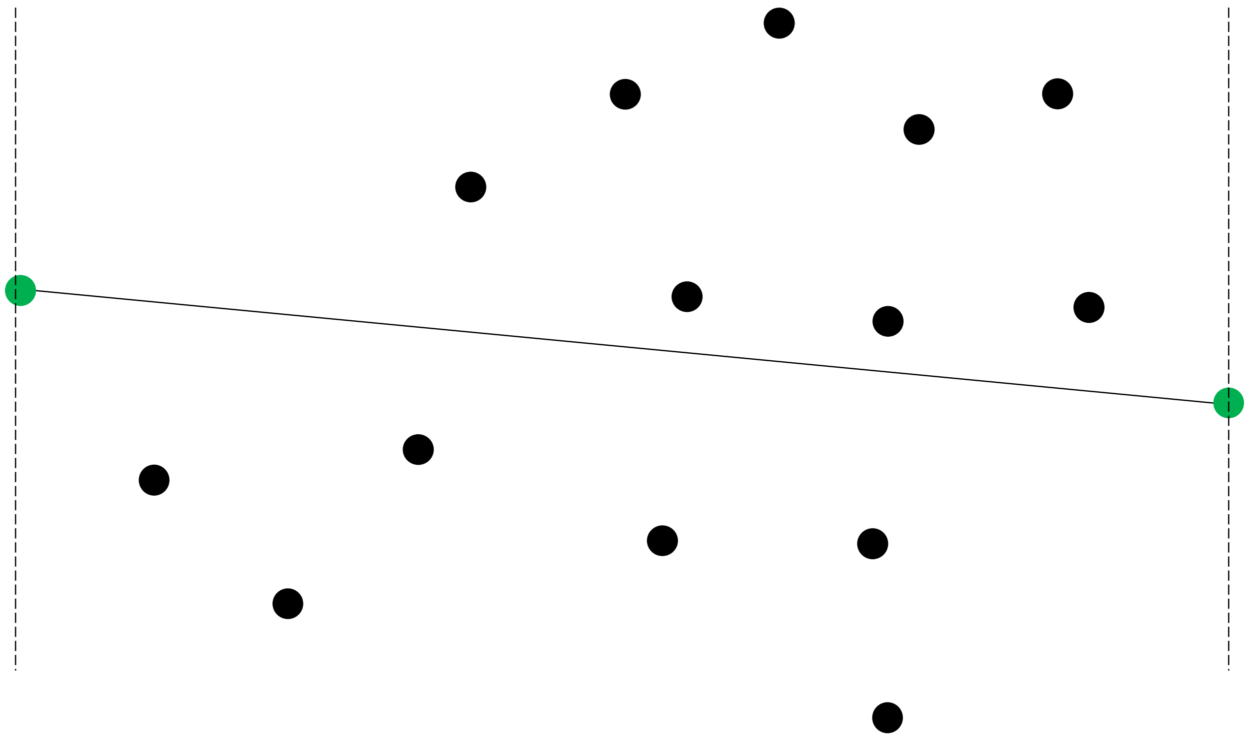
Quick Hull

- If we can have a divide-and-conquer algorithm similar to merge sort …
- why not having an algorithm similar to quick sort?
- Sketch
  - Find a pivot
  - Split the points along the pivot
  - Recursively process each side

Quick Hull

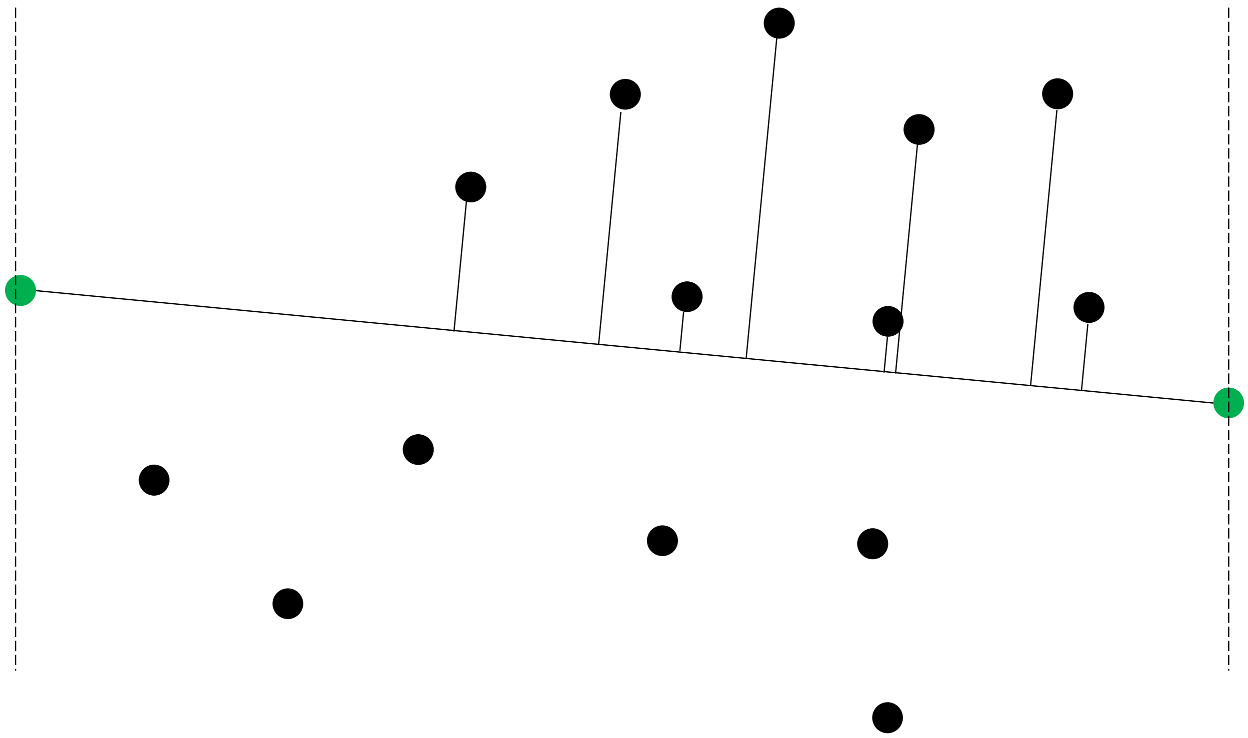


Quick Hull



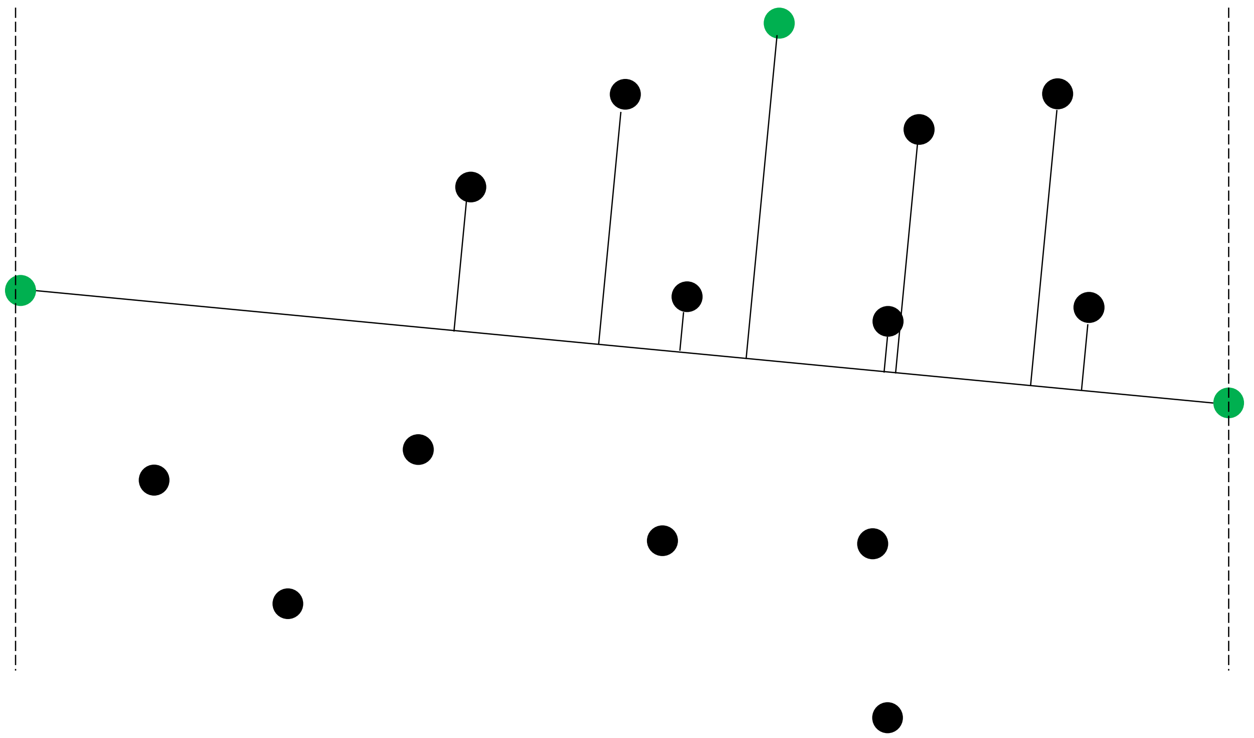
How to split the points across the line segment?

Quick Hull

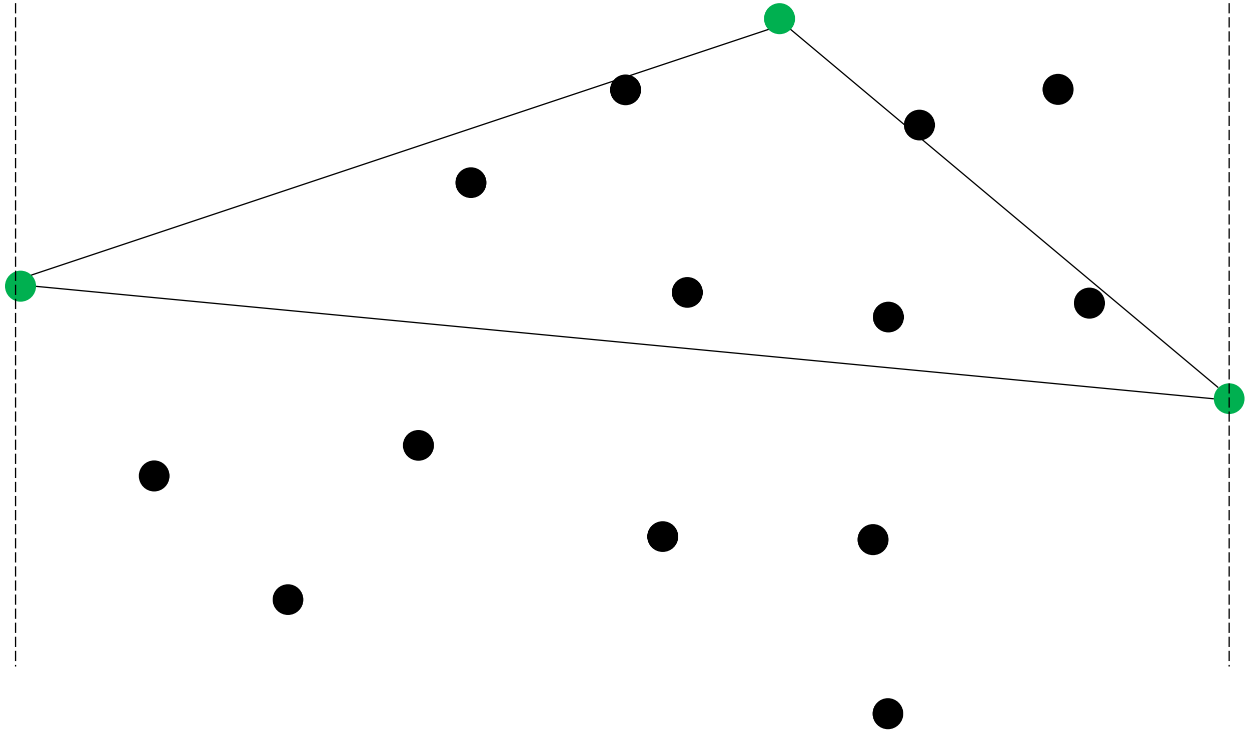


How to select the farthest point?

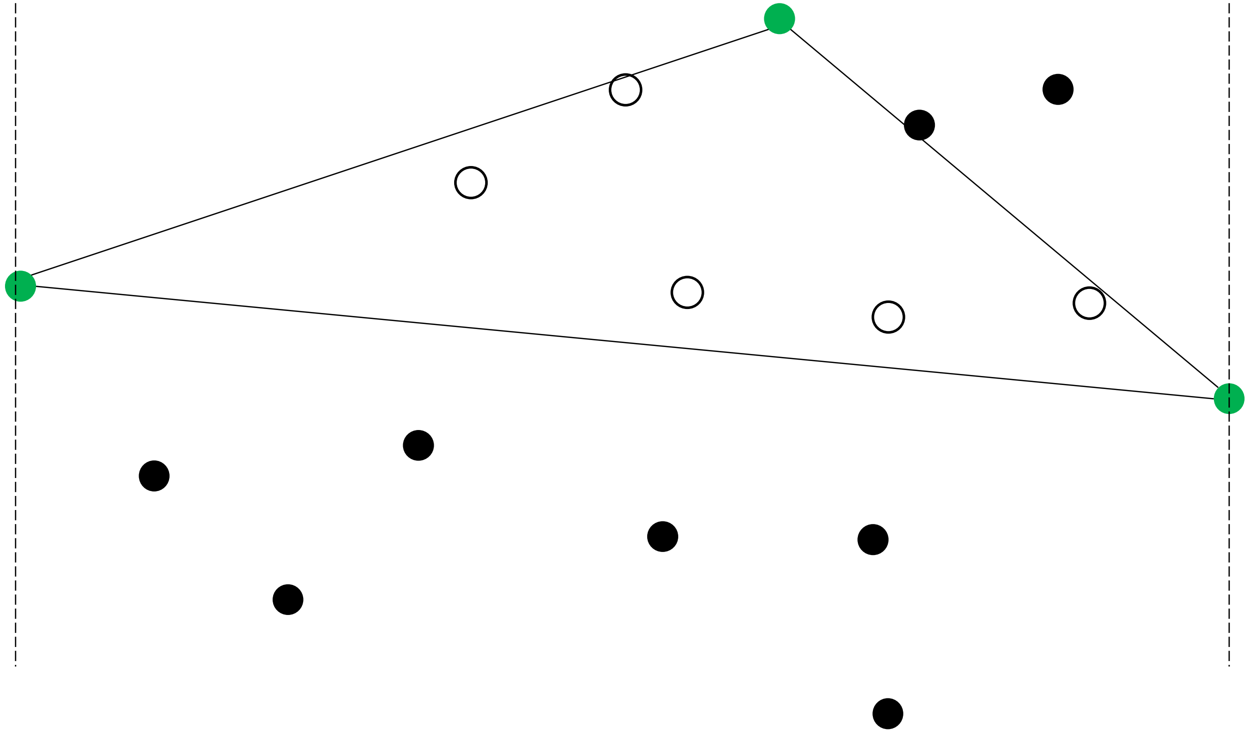
Quick Hull



Quick Hull

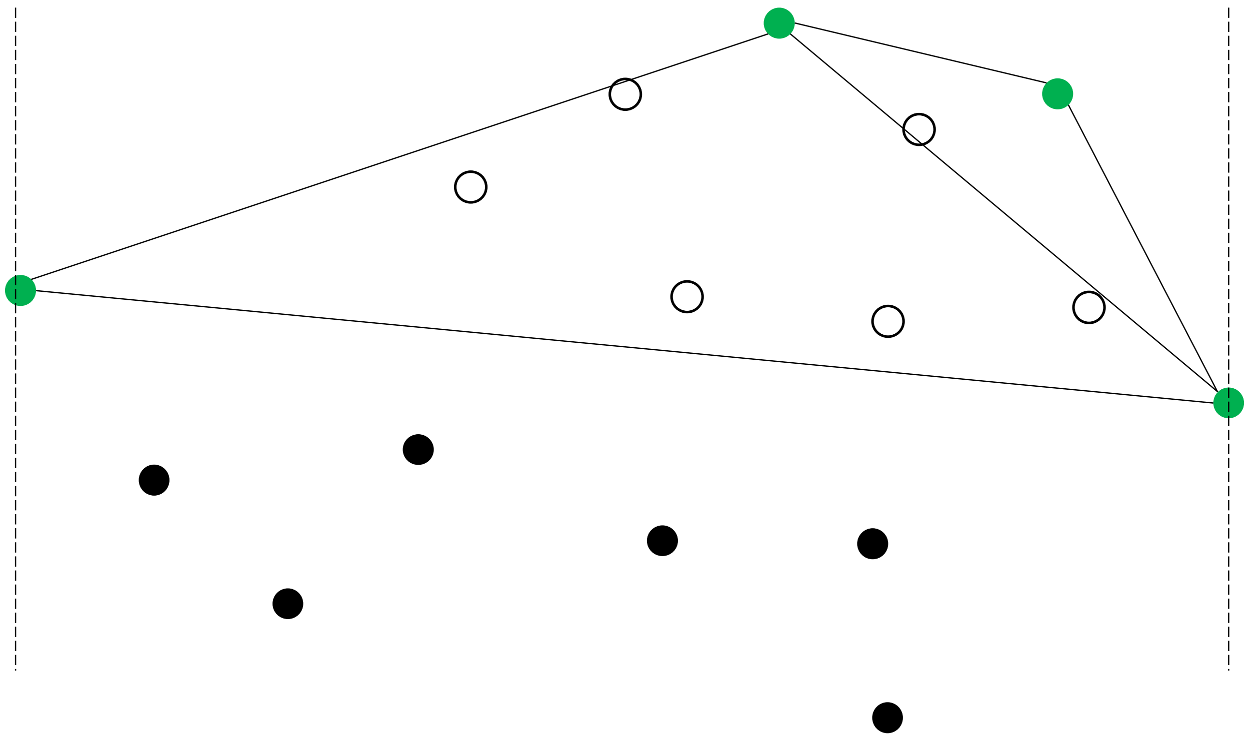


Quick Hull

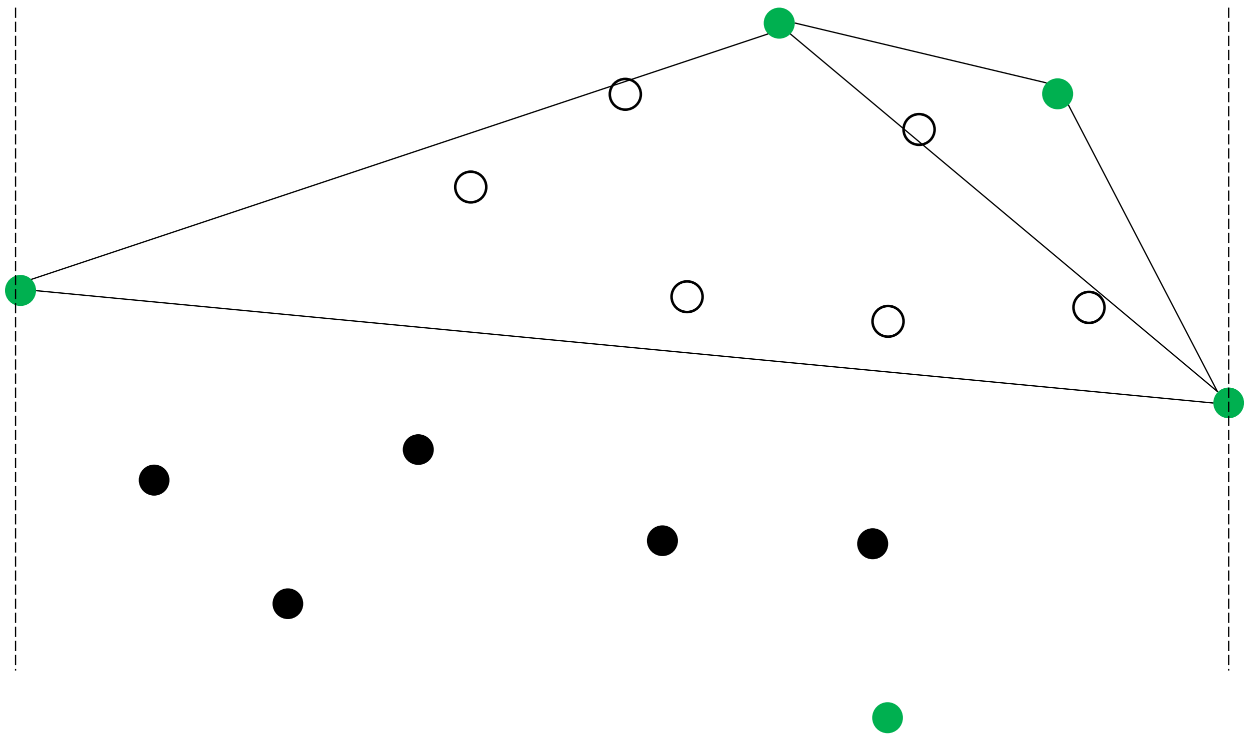


How to split the points into three subsets?

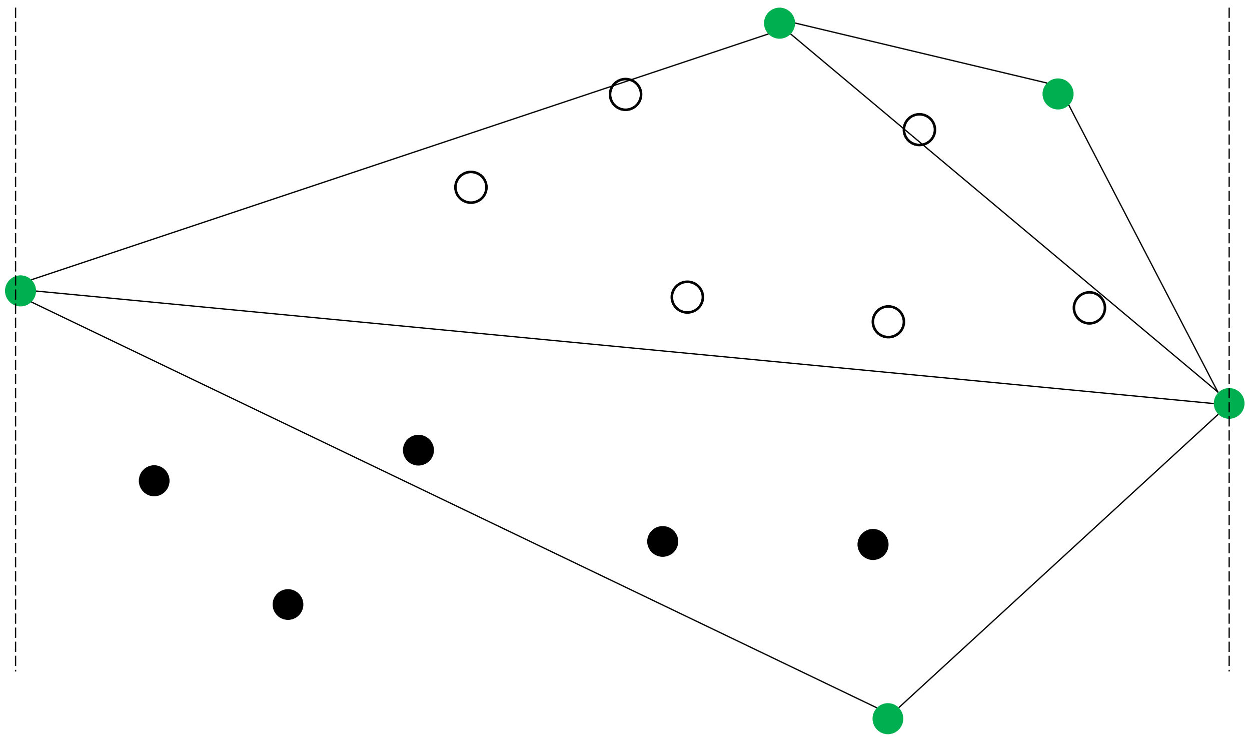
Quick Hull



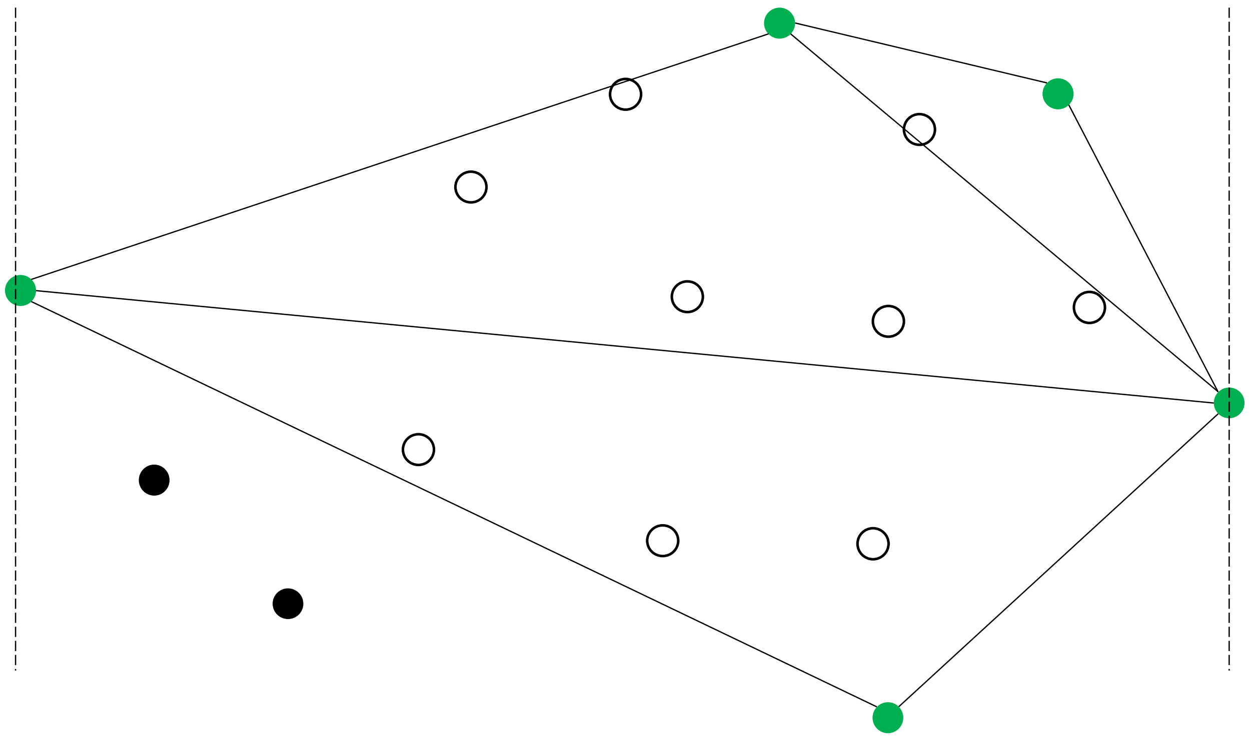
Quick Hull



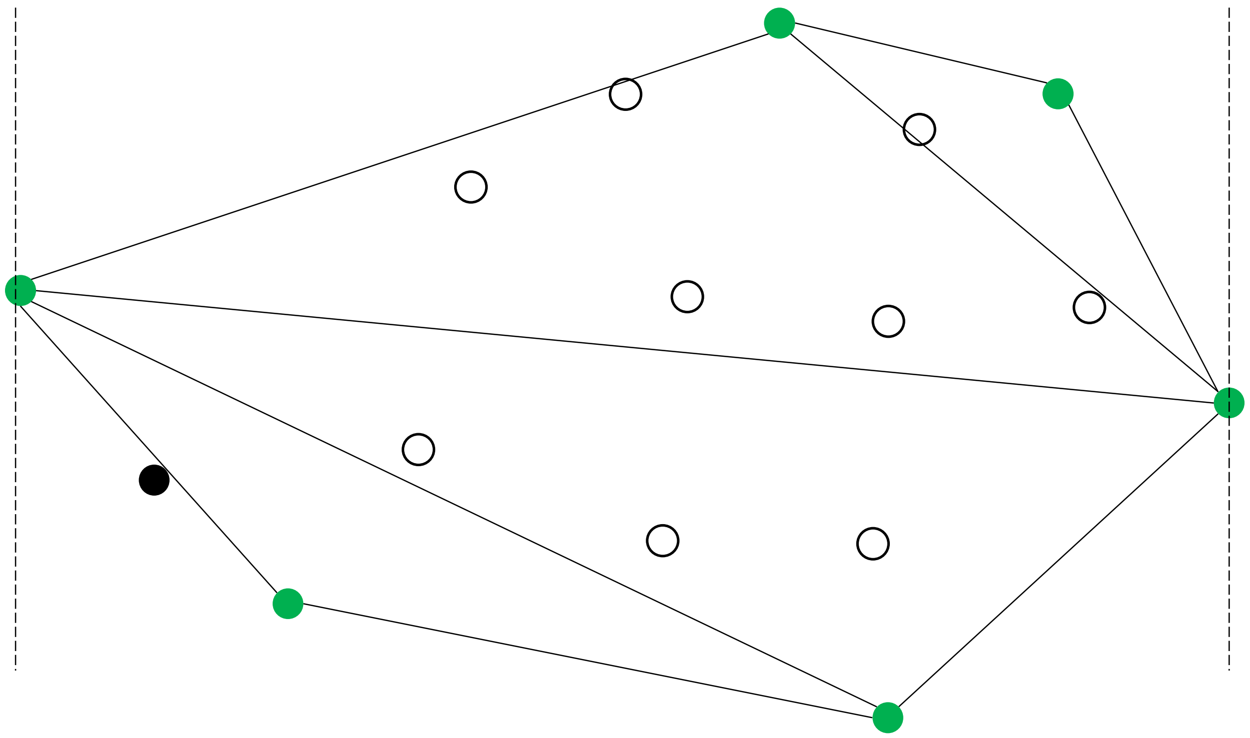
Quick Hull



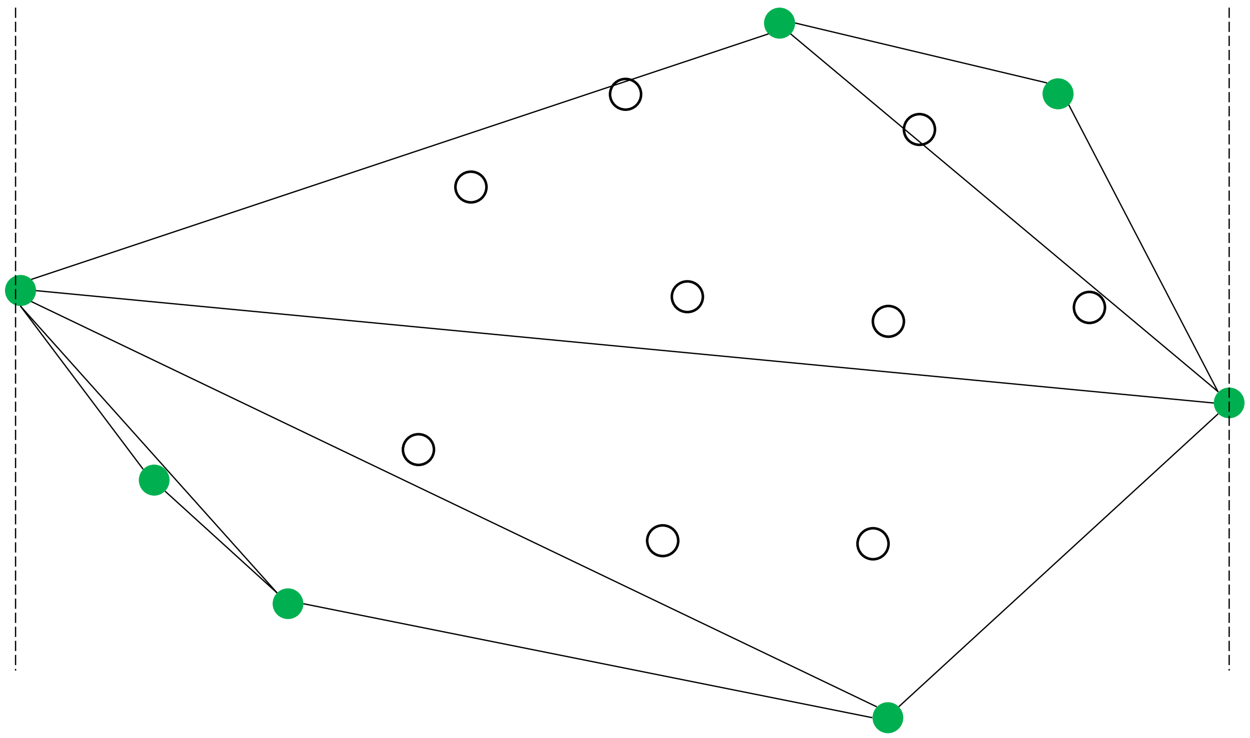
Quick Hull



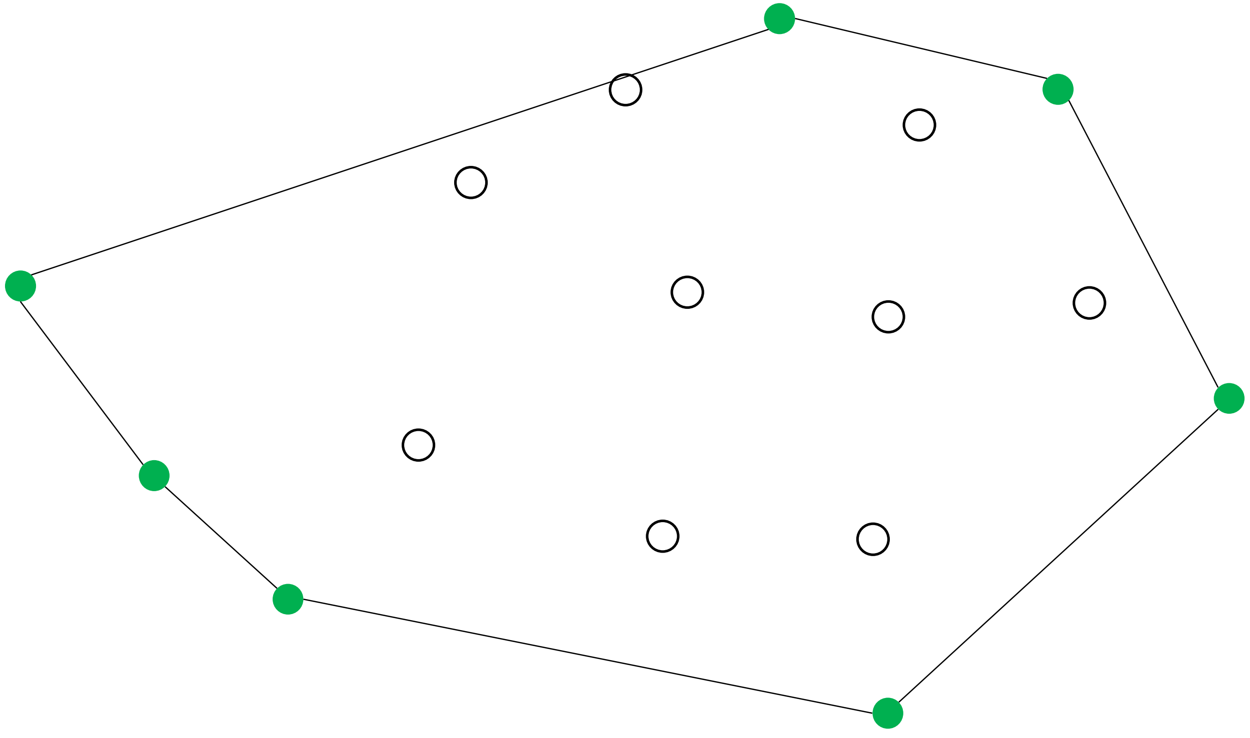
Quick Hull



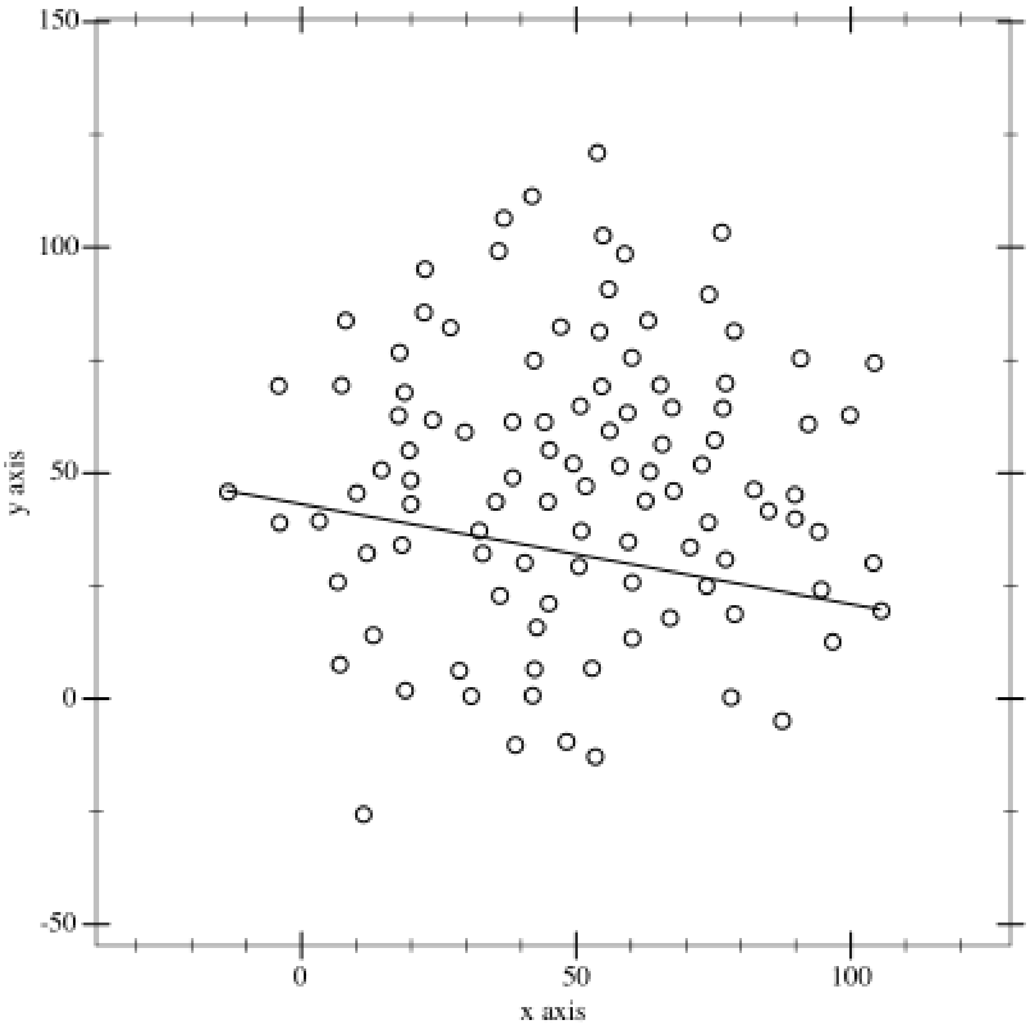
Quick Hull



Quick Hull



Example



Running Time Analysis

- $T(n) = T(n_1) + T(n_2) + O(n)$
- Worst case $n_1 = n - k$ or $n_2 = n - k$, where k is a small constant (e.g., k=1)
  - $T(n) = O(n^2)$
- Best case $n_1 = k$ and $n_2 = k$, where k is a small constant
  - In this case, most of the points are pruned
  - $T(n) = O(n)$
- Average case, $n_1 = \alpha n$ and $n_2 = \beta n$, where $\alpha < 1$ and $\beta < 1$
  - $T(n) = O(n \log n)$