Convex Hulls (2D)

O'Rourke, Chapter 3

[Preparata and Hong, 1977]

Outline

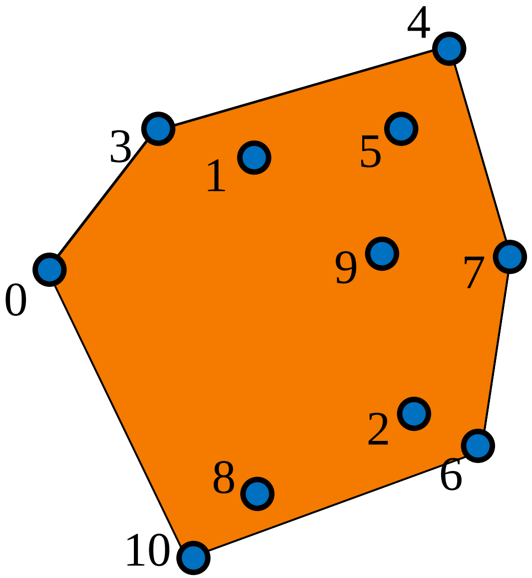
- Incremental Algorithm
- Divide-and-Conquer

Incremental Algorithm

Approach:

Grow the hull by iteratively adding points:

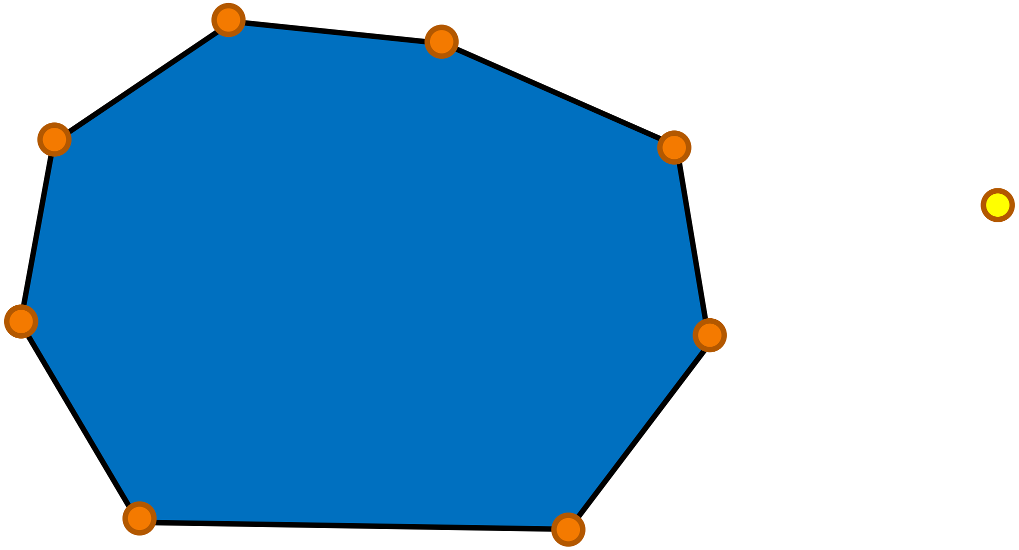
- If the point is in the hull, do nothing.
- Otherwise, grow the hull.



Incremental Algorithm

Note:

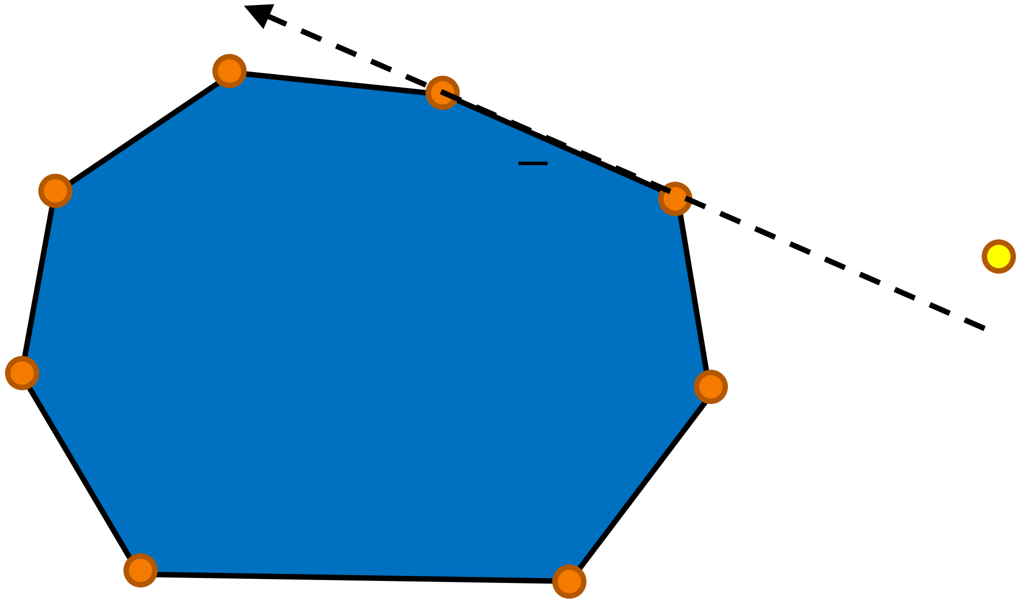
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

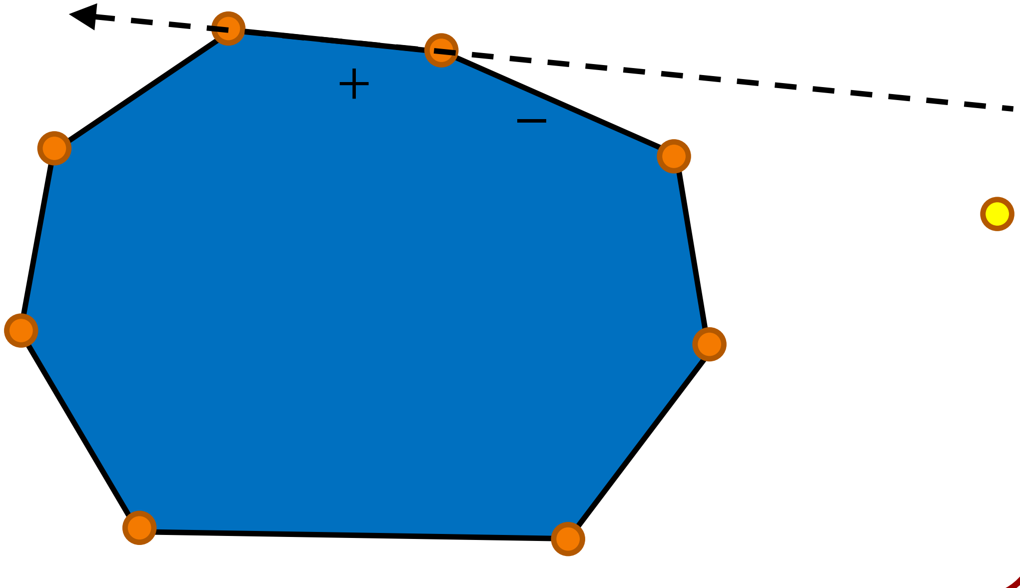
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

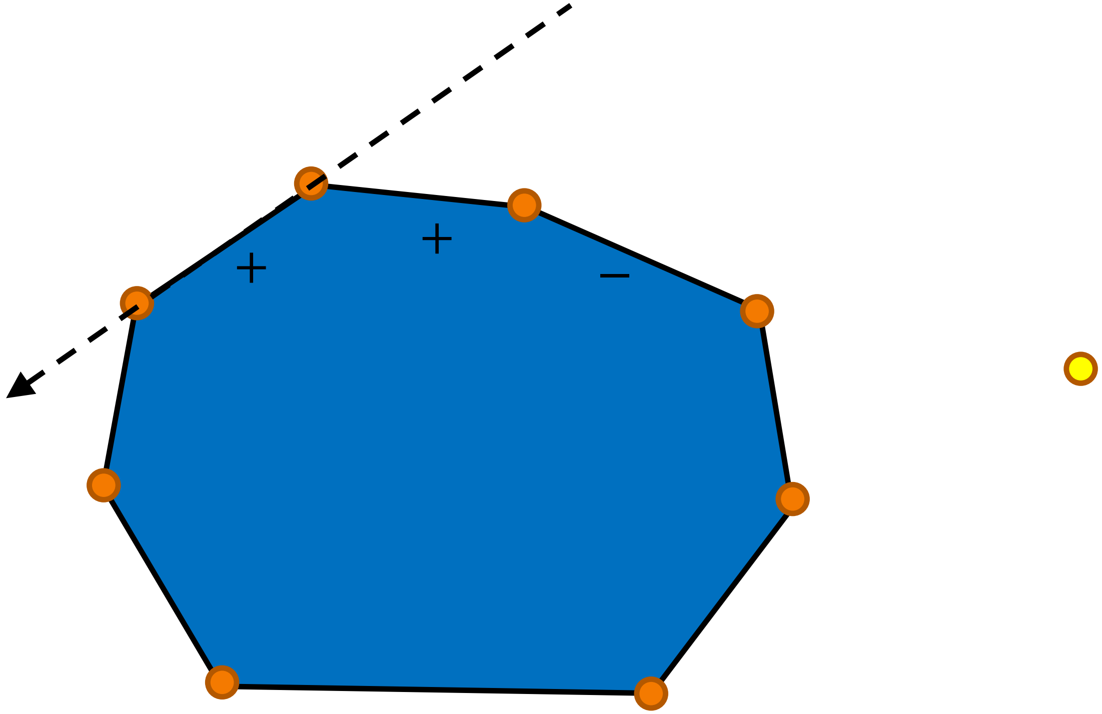
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

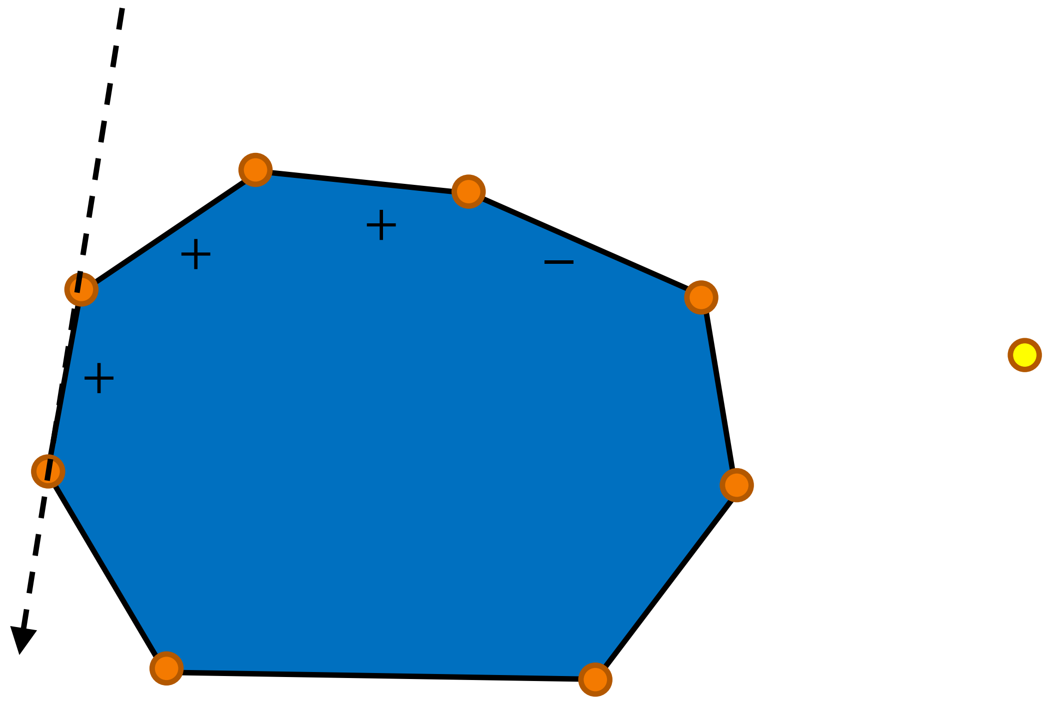
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

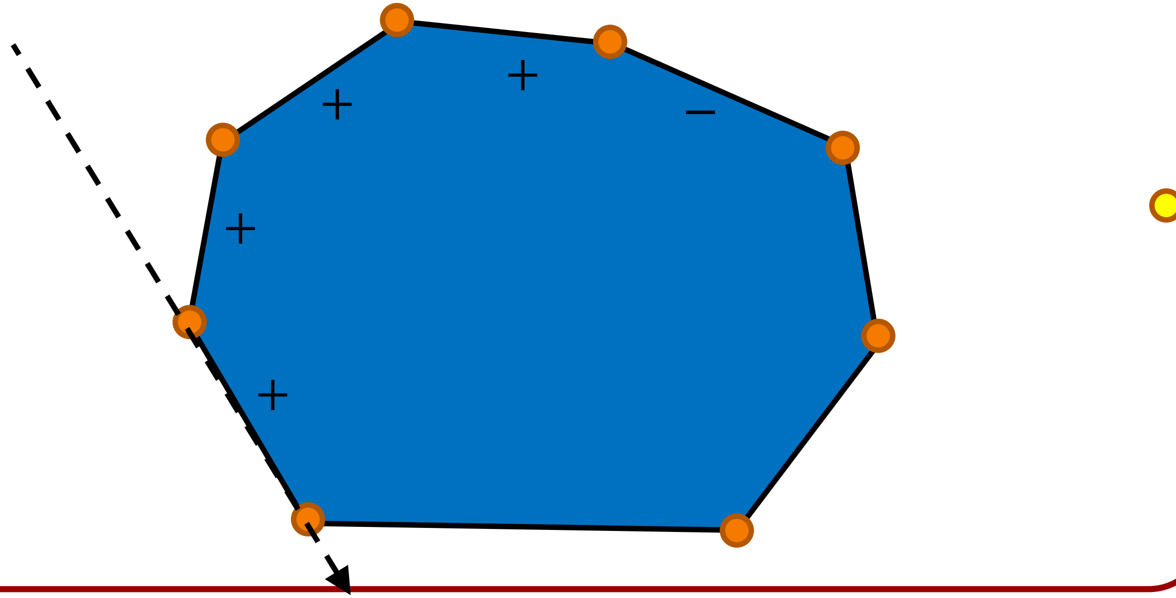
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

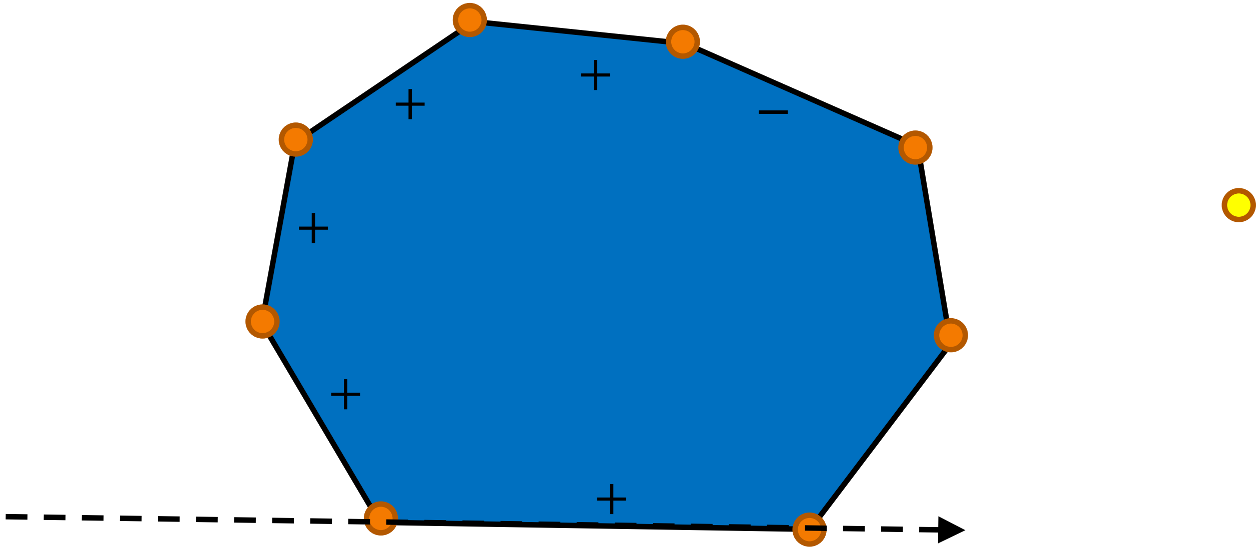
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

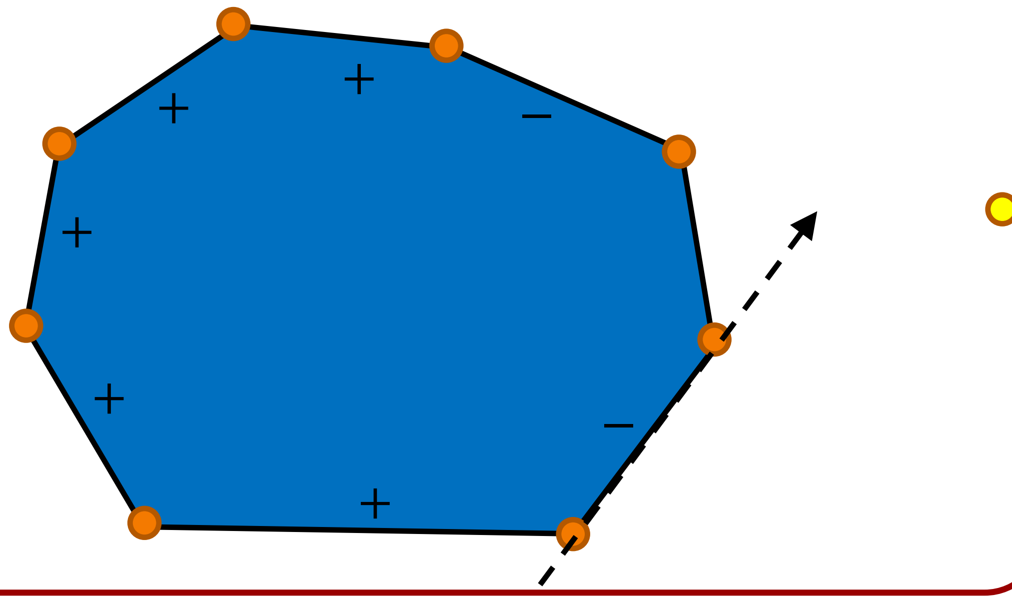
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

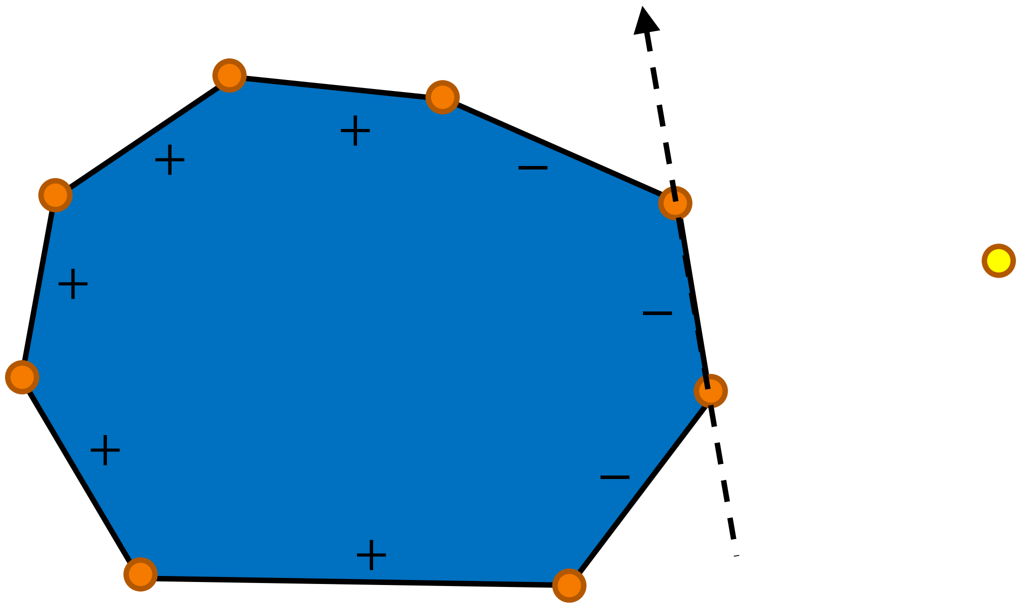
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

<u>Note</u>:

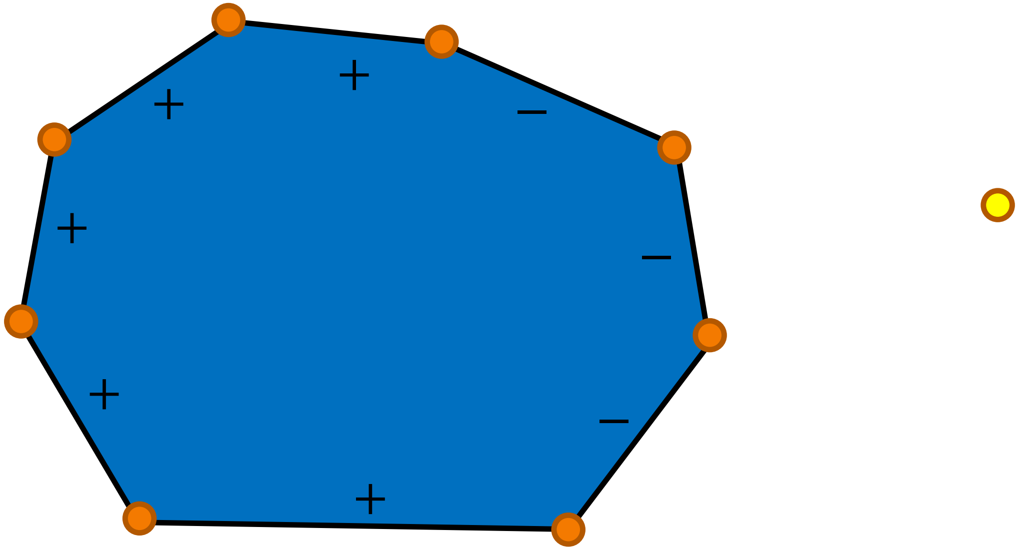
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

If a point is outside the hull, we can label the hull edges as left/right relative to the new point.
⇒ We get two vertex chains.



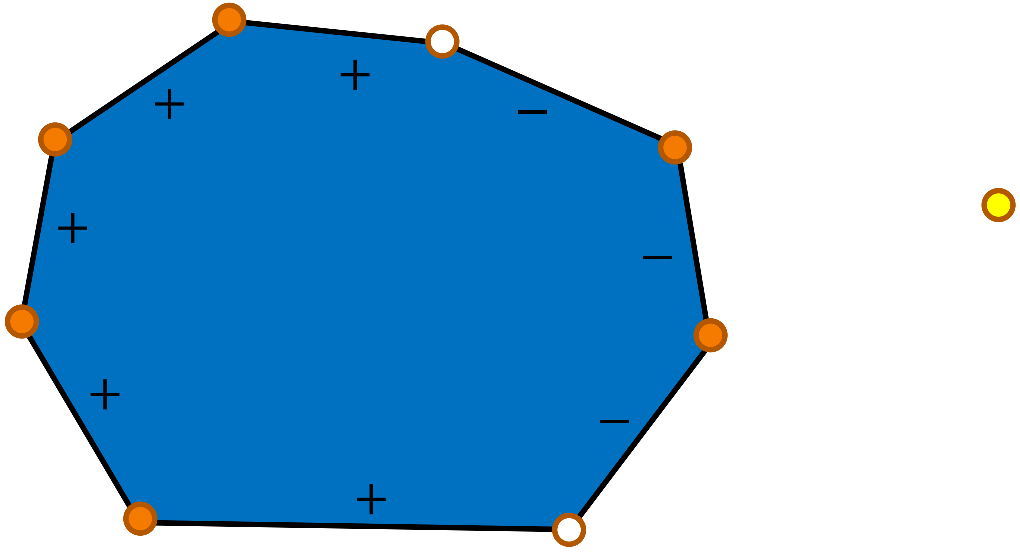
Incremental Algorithm

Note:

If a point is outside the hull, we can label the hull edges as left/right relative to the new point.

⇒ We get two vertex chains.

⇒ We get two transition vertices.

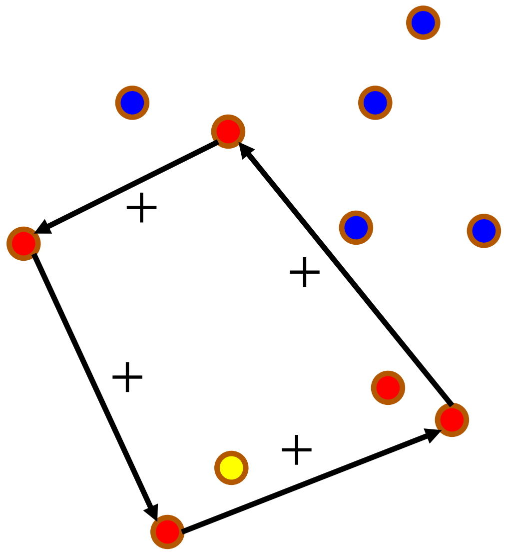


Incremental Algorithm

Naïve:

To add to a point to the hull, mark each edge, indicating if the points is to the left or right:

- If it is left of all edges, it is interior.

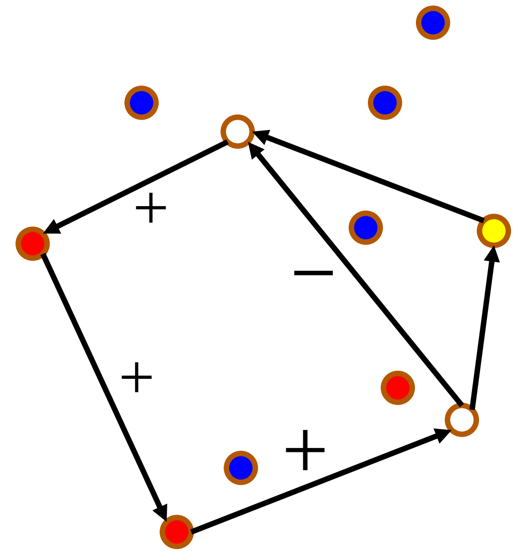


Incremental Algorithm

Naïve:

To add to a point to the hull, mark each edge, indicating if the points is to the left or right:

- If it is left of all edges, it is interior.
- Otherwise, there are two transition vertices.
  - » Connect the new point to those vertices.

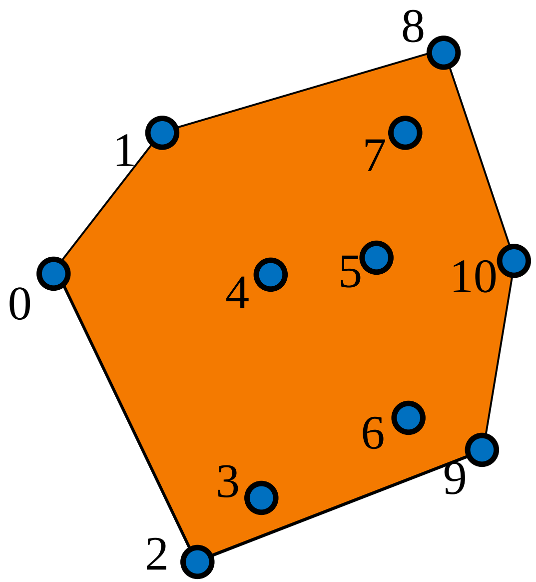


Complexity: $O(n^2)$

Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.



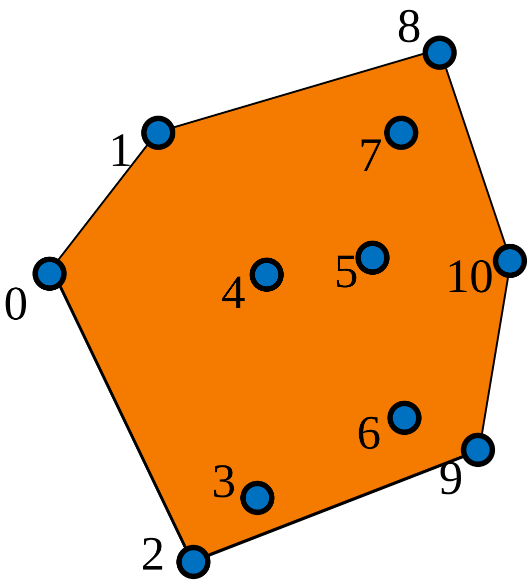
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

Since the points are sorted, each new point considered must be outside the current hull.



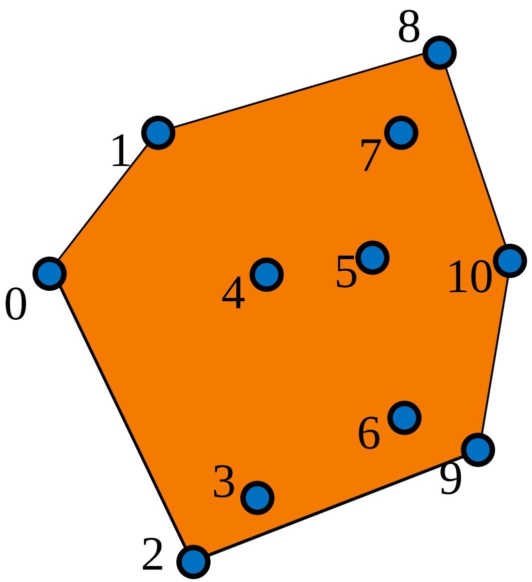
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

Since the points are sorted, each new point considered must see the previously added point.



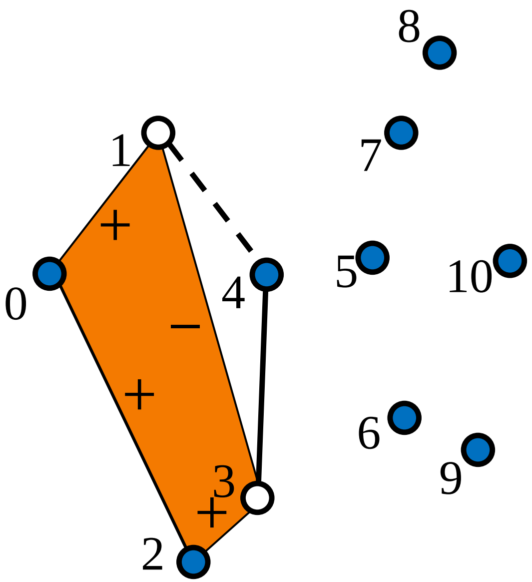
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

The edge between the new point and the previous one is between the transition vertices.



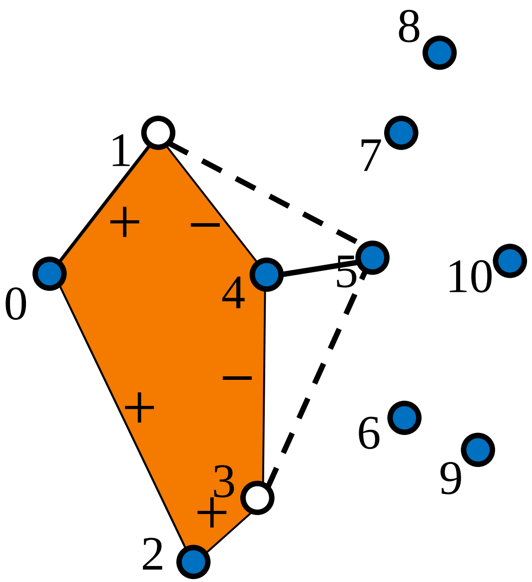
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

The edge between the new point and the previous one is between the transition vertices.



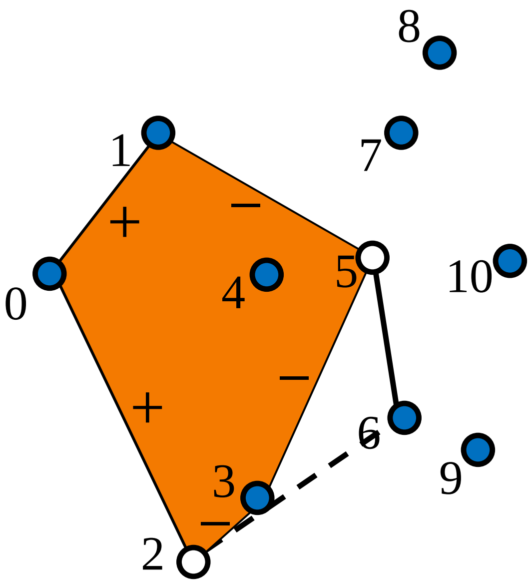
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

The edge between the new point and the previous one is between the transition vertices.



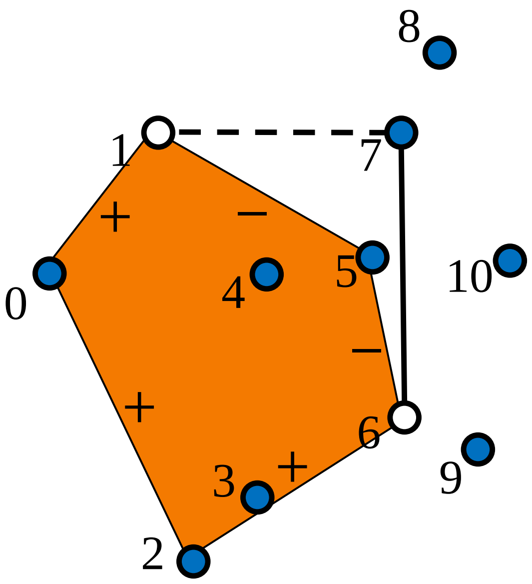
Incremental Algorithm

<u>Edelsbrunner (1987)</u>:

Sort the points lexicographically and then grow the hull by iteratively adding points.

<u>Note</u>:

The edge between the new point and the previous one is between the transition vertices.



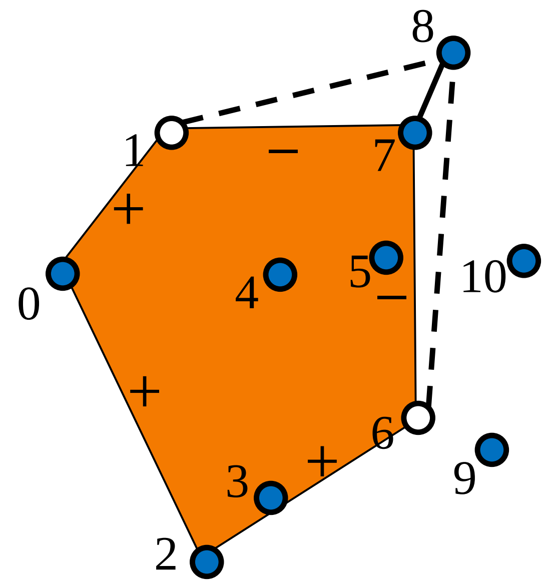
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

The edge between the new point and the previous one is between the transition vertices.



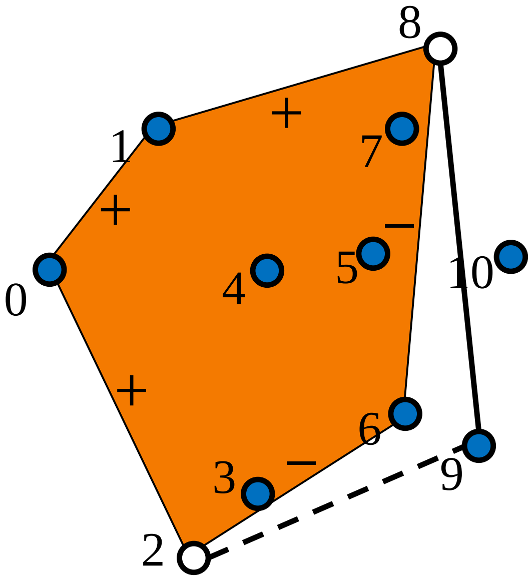
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

The edge between the new point and the previous one is between the transition vertices.



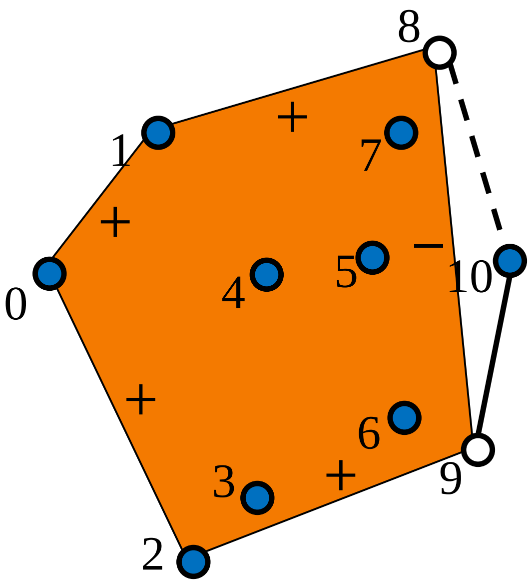
Incremental Algorithm

Edelsbrunner (1987):

Sort the points lexicographically and then grow the hull by iteratively adding points.

Note:

The edge between the new point and the previous one is between the transition vertices.



Convex Hull (2D)

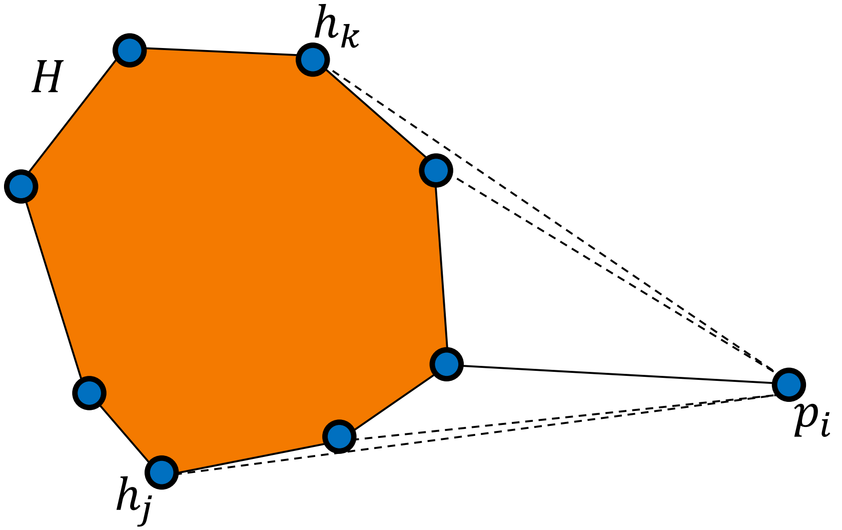
IncrementalAlgorithm(P)

- SortLexicographically(P)
- $H \leftarrow \{p_0, p_1, p_2\}$
- for $i \in [3, n)$:
  - $(h_j, h_k) \leftarrow$ TransitionVertices(H , p_i)
  - Replace(H , $\{h_j, \dots, h_k\}$, $\{h_j, p_i, h_k\}$)

Convex Hull (2D)

IncrementalAlgorithm(P)

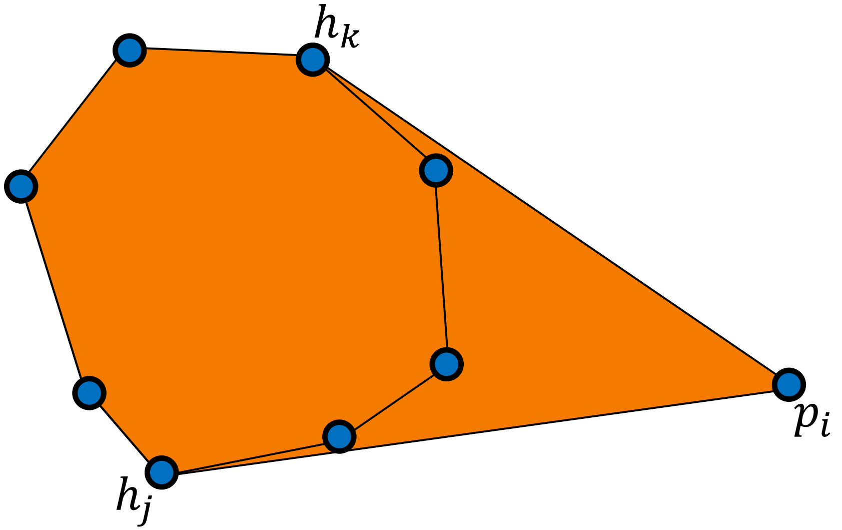
- SortLexicographically(P)
- $H \leftarrow \{p_0, p_1, p_2\}$
- for $i \in [3, n)$:
  - » $(\boldsymbol{h_j}, \boldsymbol{h_k}) \leftarrow$ **TransitionVertices(** $\boldsymbol{H}$ **,** $\boldsymbol{p_i}$ **)**
  - » Replace(H , $\{h_j, \ldots, h_k\}$, $\{h_j, p_i, h_k\}$)



Convex Hull (2D)

IncrementalAlgorithm(P)

- SortLexicographically(P)
- $H \leftarrow \{p_0, p_1, p_2\}$
- for $i \in [3, n)$:
  - » $(h_j, h_k) \leftarrow$ TransitionVertices(H , p_i)
  - » **Replace($\boldsymbol{H}$, $\{\boldsymbol{h_j}, \dots, \boldsymbol{h_k}\}$, $\{\boldsymbol{h_j}, \boldsymbol{p_i}, \boldsymbol{h_k}\}$)**



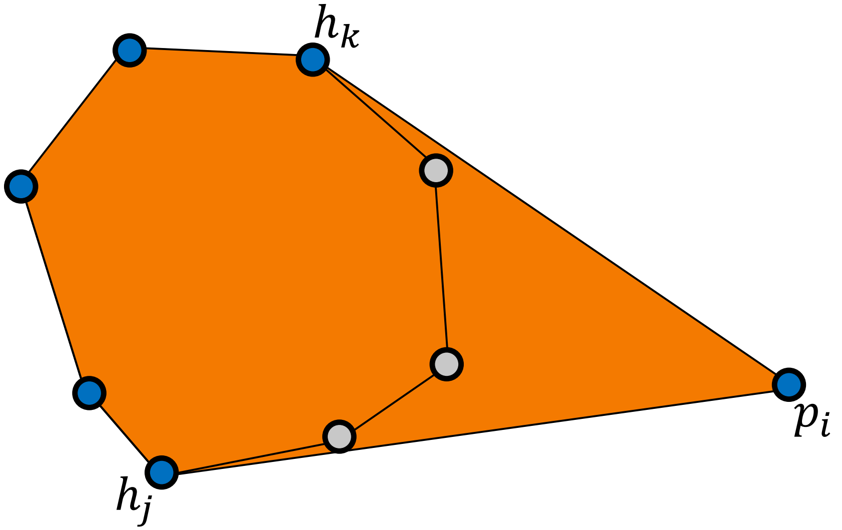
Convex Hull (2D)

IncrementalAlgorithm(P)

- SortLexicographically(P) ← $O(n \log n)$
- $H \leftarrow \{p_0, p_1, p_2\}$
- for $i \in [3, n)$:
  - » $(h_j, h_k) \leftarrow$ TransitionVertices(H , p_i)
  - » Replace(H , $\{h_j, \dots, h_k\}$, $\{h_j, p_i, h_k\}$)

  ← $O(?)$

<u>Note</u>:
Any vertex traversed to find the transition vertices is removed.



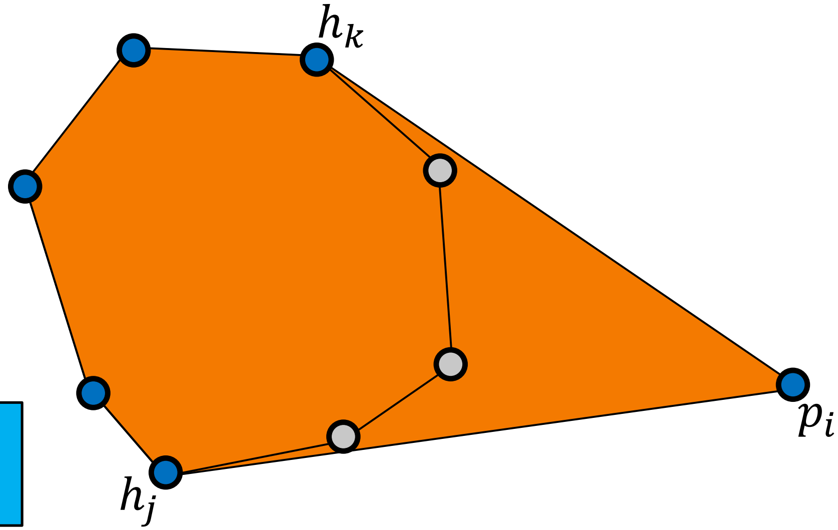
Convex Hull (2D)

IncrementalAlgorithm(P)

- SortLexicographically(P) ← $O(n \log n)$
- $H \leftarrow \{p_0, p_1, p_2\}$
- for $i \in [3, n)$:
  - » $(h_j, h_k) \leftarrow$ TransitionVertices(H , p_i)
  - » Replace(H , $\{h_j, \dots, h_k\}$, $\{h_j, p_i, h_k\}$)

  ← $O(n)$

Note:
Any vertex traversed to find the transition vertices is removed.



Complexity: $O(n \log n)$

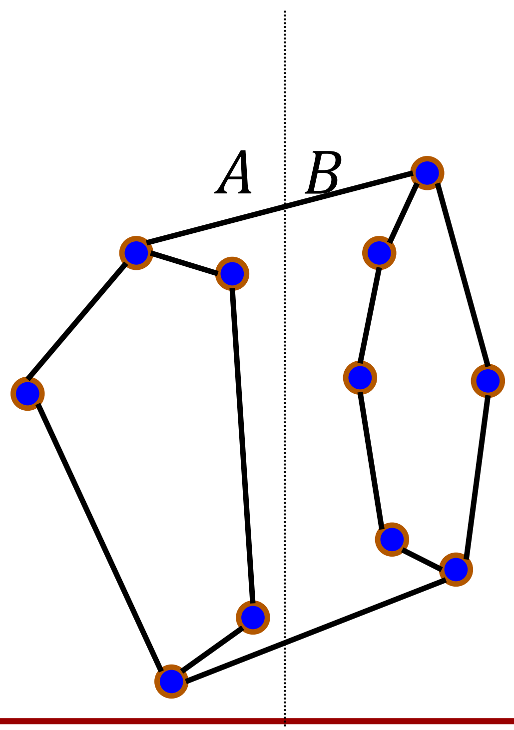
Outline

- Incremental Algorithm
- Divide-and-Conquer

Divide And Conquer

Recursively:

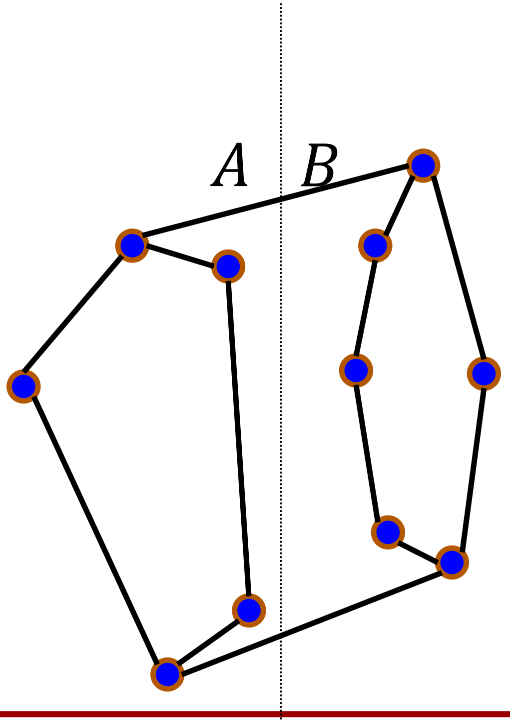
- Split the point-set in two.
- Compute the hull of both halves
- Merge the hulls



Divide And Conquer

<u>Efficiency</u>:

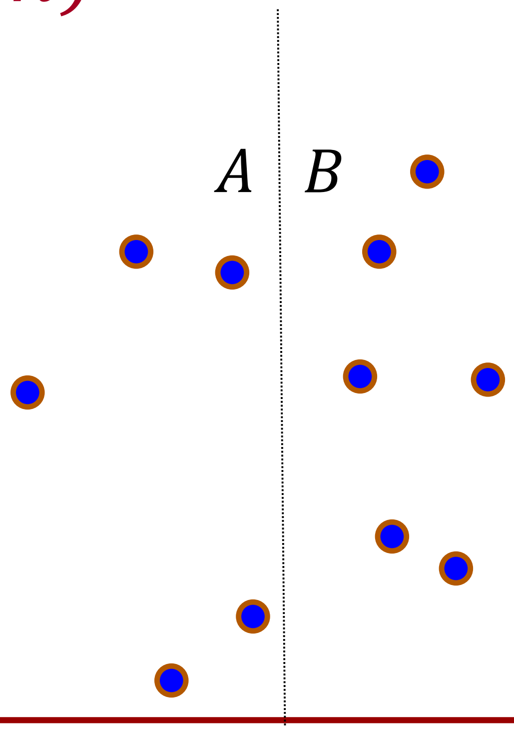
For this to be fast (log-linear), the splitting and the merging have to be fast (linear).



Divide And Conquer (Step 1)

<u>Split the point-set in two</u>:

- Sort the points along an axis and choose the $(n/2)$-th element.
  - »Pre-processing: $O(n \log n)$
  - »Run-time: $O(n)$
- Use fast median.
  - »Run-time: $O(n)$



Fast Median

Approach:

- To get the median of a set S, break up the set into subsets of size 5.*
- Compute the median of each subset.
- Compute the median of the medians. [Recursive]
- Use that to split S in two and find the biased median of the larger half. [Recursive]

*For simplicity, we will assume that $|S|$ is divisible by 5.

Fast Median

FastMedian($P = \{x_0, \ldots, x_{n-1}\}$, $s = |P|/2$):

- if($|P|$ ==1) return x_0
- $Q_i \leftarrow \{x_{5i+0}, \ldots, x_{5i+4}\}$
- for $i \in [0, |P|/5)$: $q_i \leftarrow$ SlowMedian(Q_i)
- $Q \leftarrow \{q_0, \ldots, q_{|P|/5-1}\}$
- (L , R) ← Split(P , FastMedian(Q , $|Q|/2$))
- if($|L| < s$) return FastMedian(R , $s - |L|$)
- else return FastMedian(L , s)

Fast Median

$O(n)$ Complexity:

To show that this has linear complexity, we show that every time we recurse on a subset $S' \subset S$, the size of the subset satisfies:

$$|S'| \leq |S| \cdot \varepsilon$$

for some fixed $\varepsilon < 1$.

Fast Median

FastMedian($P = \{x_0, \dots, x_{n-1}\}$, $s = |P|/2$):

- if($|P|$ ==1) return x_0
- $Q_i \leftarrow \{x_{5i+0}, \dots, x_{5i+4}\}$
- for $i \in [0, |P|/5)$: $q_i \leftarrow$ SlowMedian(Q_i)
- $Q \leftarrow \{q_0, \dots, q_{|P|/5-1}\}$
- (L , R) ← Split(P , FastMedian(Q , $|Q|/2$))
- if($|L| < s$) return FastMedian(R , $s - |L|$)
- else return FastMedian(L , s)

Claim:

- The subsets L and R defined by:

  (L , R) ← Split(P , FastMedian(Q , $|Q|/2$))

  have the property that $|L|, |R| \leq 4|P|/5$

Fast Median

Claim:

- The subsets L and R defined by:
  $(L, R) \leftarrow \text{Split}(P, \text{FastMedian}(Q, |Q|/5))$
  have the property that $|L|, |R| \leq 4|P|/5$

Proof:

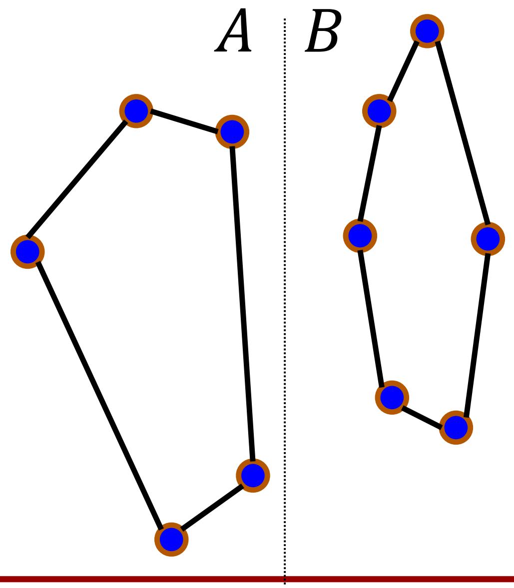
- Set $q = \text{FastMedian}(Q, |Q|/5)$
- For each $q_i \in Q$ with $q_i < q$ (50%)
  - » Each $p_i \in P_i$ with $p_i < q_i$ must be in L (40%)
- For each $q_i \in Q$ with $q_i \geq q$ (50%)
  - » Each $p_i \in P_i$ with $p_i \geq q_i$ must be in R (40%)

$\Rightarrow$ L and R both contain at least one fifth of the points in P.

Divide And Conquer (Step 2)

Compute the hull of the halves:

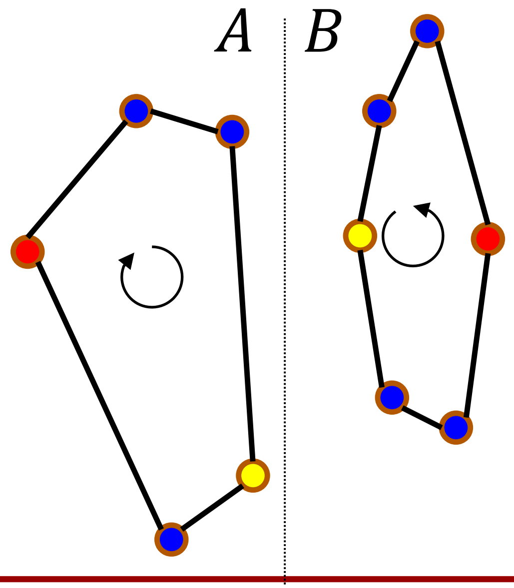
- If the subset has less than 6 points, apply the incremental algorithm,
- Otherwise recurse.



Divide And Conquer (Step 3)

Merging the hulls (lower tangent)*:

- Find the edge from A to B connecting the right-most point on A to the left-most point on B.
- Move CW on A and CCW on B, while A and B are not entirely above the edge.

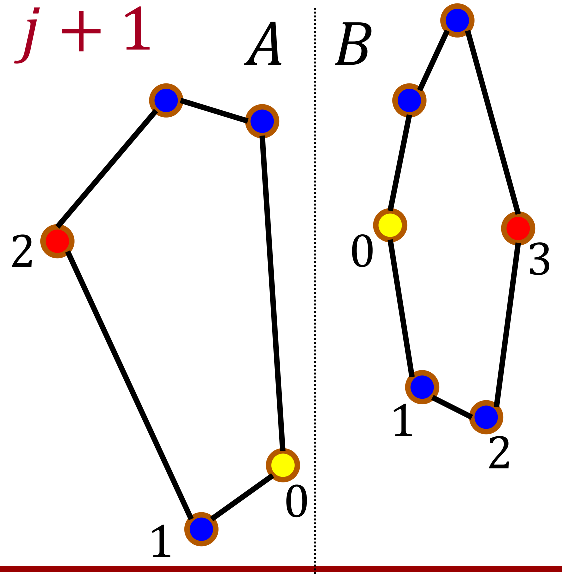


*Assuming general position

Merging the Hulls (lower tangent)

Merge (A , B):

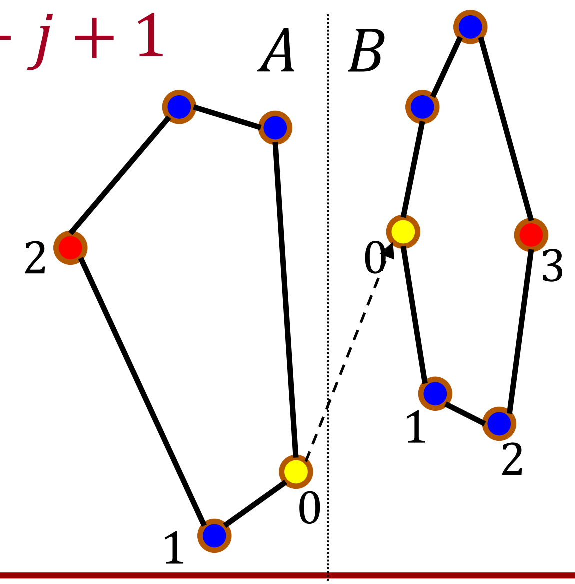
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i, j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i + 1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j + 1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

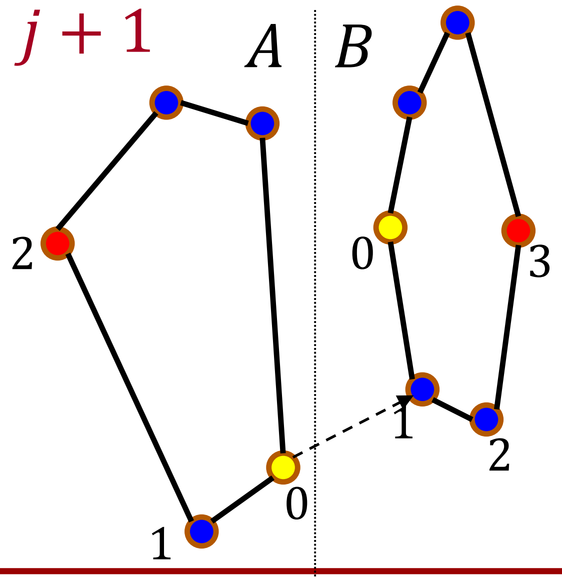
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - else: break



Merging the Hulls (lower tangent)

Merge (A , B):

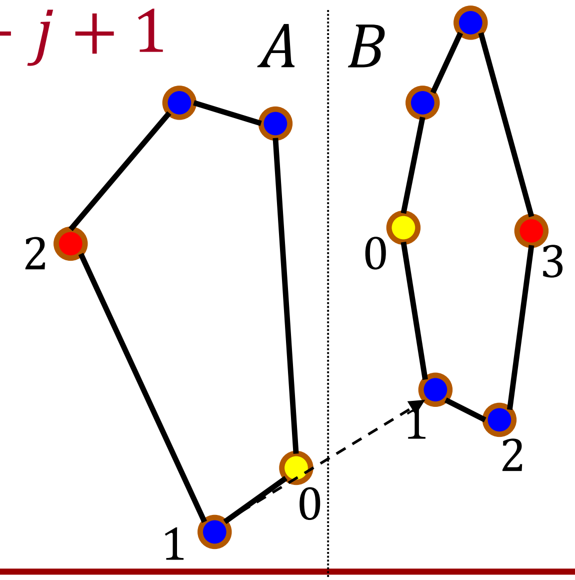
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

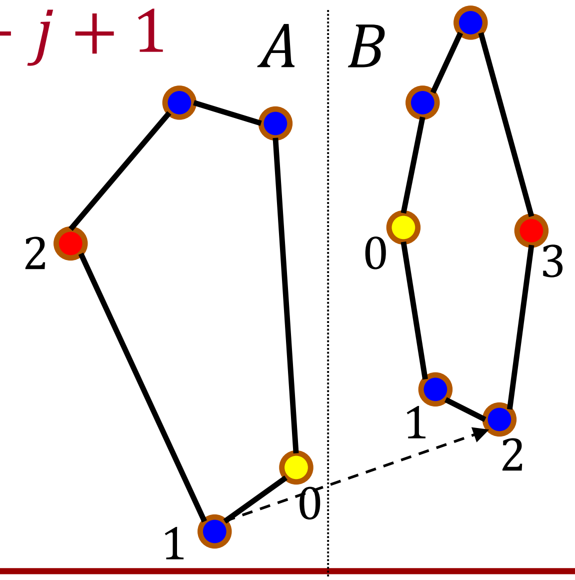
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i, j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i + 1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j + 1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i, j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i + 1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j + 1$
  - » else: break

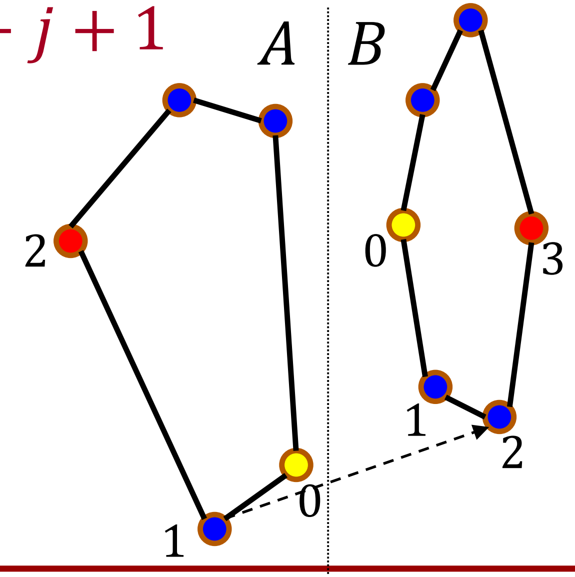


Merging the Hulls (lower tangent)

Merge (A , B):

- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break

Need to show that this terminates at the lower tangent in linear time.



Merging the Hulls (lower tangent)

Claim:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

We show that if this is true, then:

- The algorithm must terminate in linear time because:
  - i won't pass the left-most vertex of A.
  - j won't pass the right-most vertex of B.
- The algorithm terminates at the lower tangent.

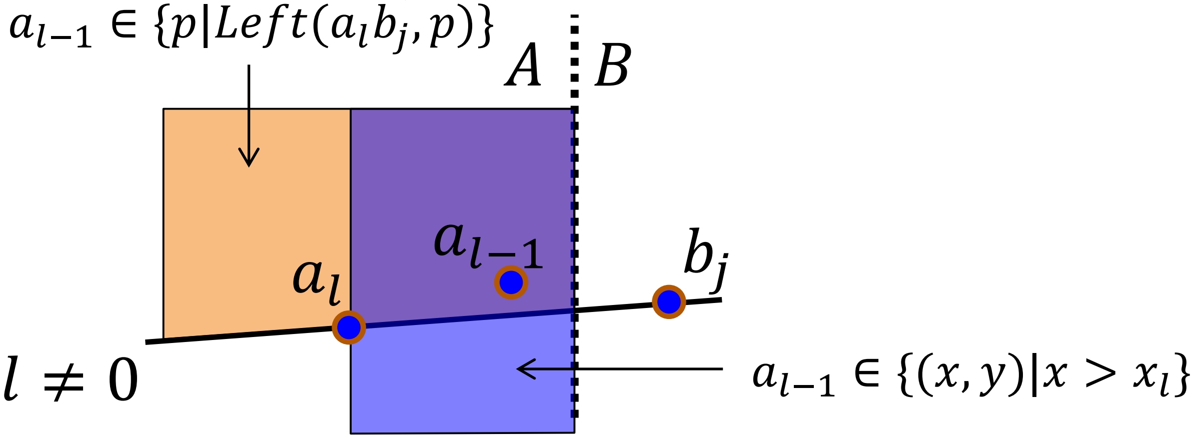
Merging the Hulls (lower tangent)

Claim:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - » Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

The algorithm must terminate because:

- » i doesn't pass the left-most vertex of A
- Right($\overrightarrow{a_l b_j}$, a_{l+1})==false



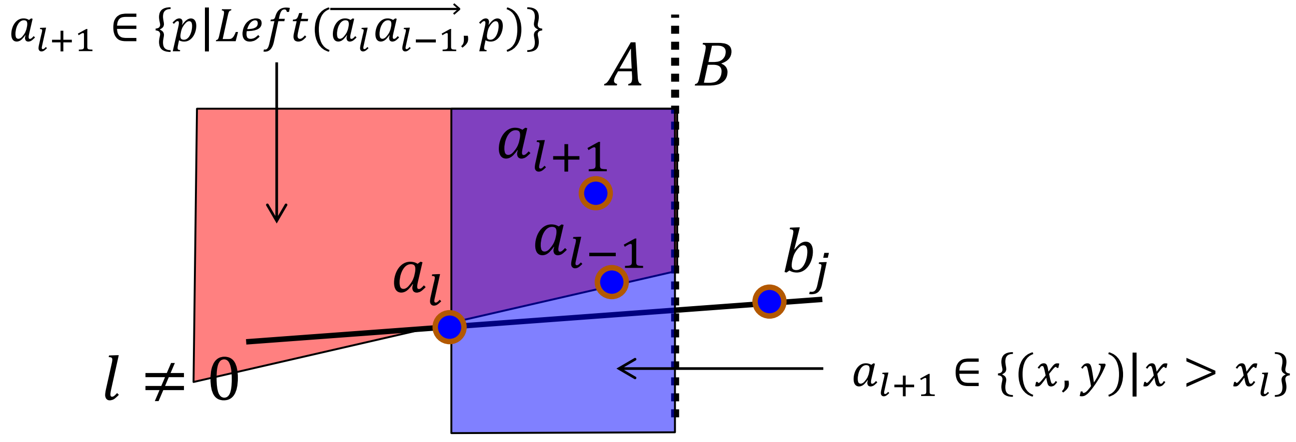
Merging the Hulls (lower tangent)

Claim:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - » Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

The algorithm must terminate because:

- » i doesn't pass the left-most vertex of A
- Right($\overrightarrow{a_l b_j}$, a_{l+1})==false



Merging the Hulls (lower tangent)

Claim:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

When the algorithm terminates the edge (i, j) is a lower tangent.

Merging the Hulls (lower tangent)

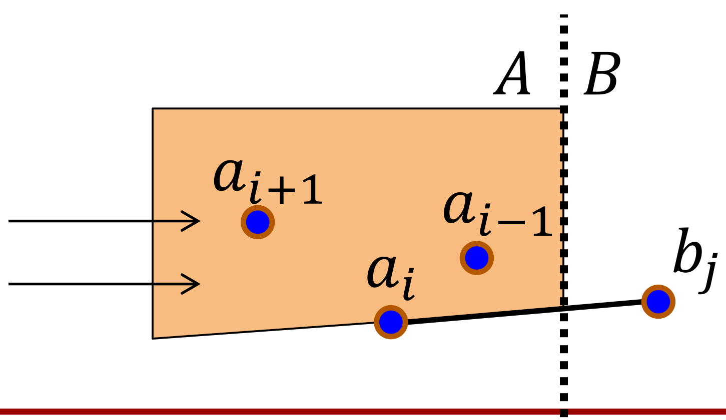
<u>Claim</u>:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - » Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - » Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

When the algorithm terminates the edge (i, j) is a lower tangent.

Case: $i \neq 0$

$$a_{i-1} \in \{p \mid Left(\overrightarrow{a_i b_j}, p)\}$$
$$a_{i+1} \in \{p \mid Left(\overrightarrow{a_i b_j}, p)\}$$



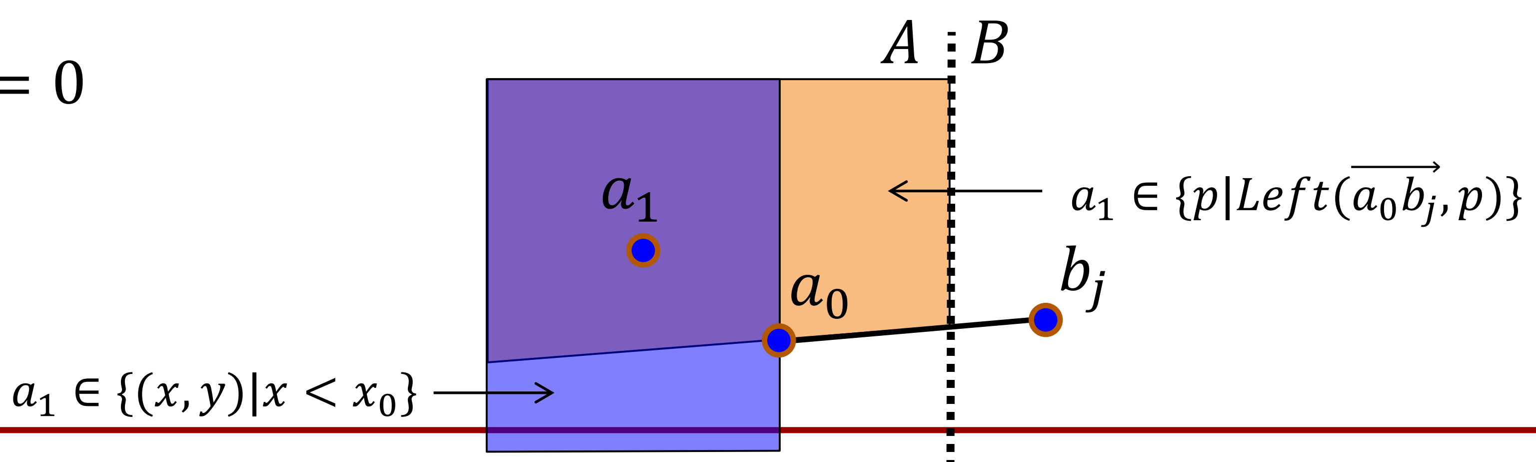
Merging the Hulls (lower tangent)

Claim:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

When the algorithm terminates the edge (i, j) is a lower tangent.

Case: $i = 0$



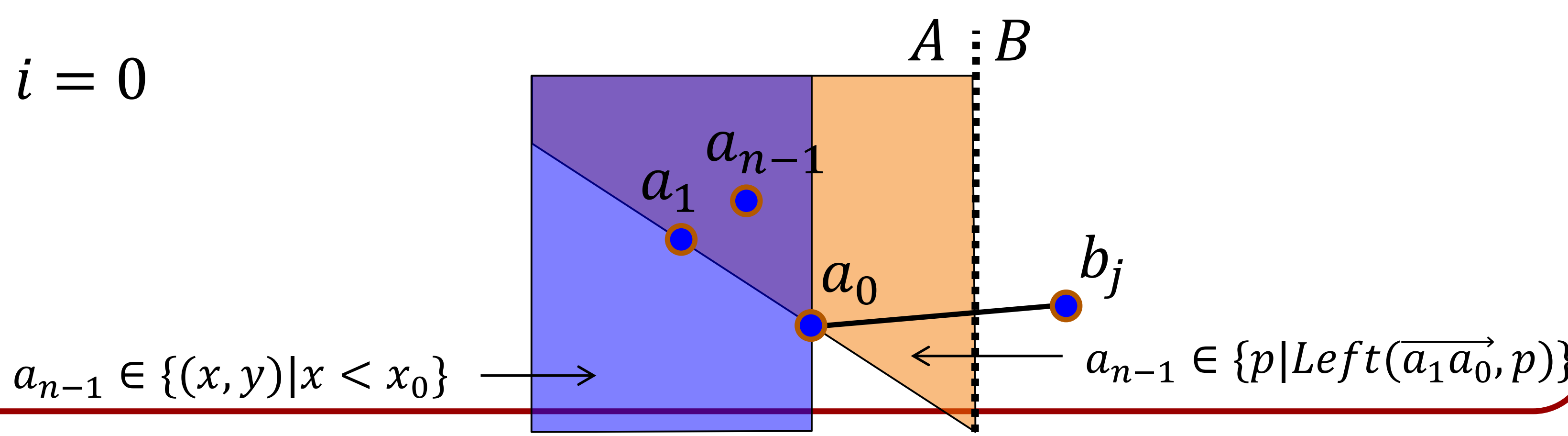
Merging the Hulls (lower tangent)

Claim:

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - » Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - » Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

When the algorithm terminates the edge (i, j) is a lower tangent.

Case: $i = 0$



Merging the Hulls (lower tangent)

- If edge $\overline{a_i b_j}$ connects A and B, then:
  - Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Proof by Induction:

Base case, $(i, j) = (0,0)$, is trivially satisfied.

Merging the Hulls (lower tangent)

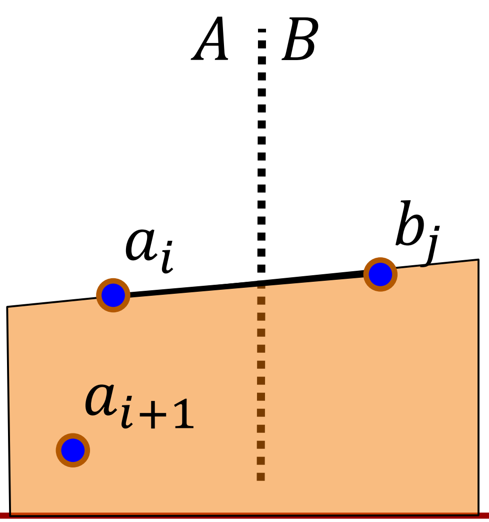
- If edge $\overline{a_i b_j}$ connects A and B, then:
  - Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

<u>Proof by Induction</u>:

Assume true for (i, j) and assume we transition $(i, j) \to (i + 1, j)$:

$\Rightarrow$ Right($\overrightarrow{a_i b_j}$, a_{i+1})

$\Rightarrow$ Left($\overrightarrow{a_{i+1} b_j}$, a_i)



Merging the Hulls (lower tangent)

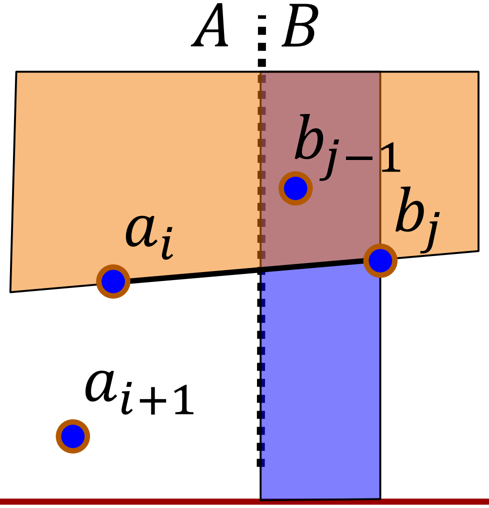
- If edge $\overline{a_i b_j}$ connects A and B, then:
  - Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
  - Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

<u>Proof by Induction</u>:

On the other hand:

- b_{j-1} must be left of b_j
- b_{j-1} must be left of edge $\overrightarrow{a_i b_j}$

$\Rightarrow b_{j-1}$ must be left of edge $\overrightarrow{a_{i+1} b_j}$



Merging the Hulls (lower tangent)

Complexity:

Both split and the merge run in $O(n)$.

$\Rightarrow$ The divide-and-conquer runs in $O(n \log n)$.