Convex Hulls (2D)

O'Rourke, Chapter 3

[Preparata and Hong, 1977]

Outline

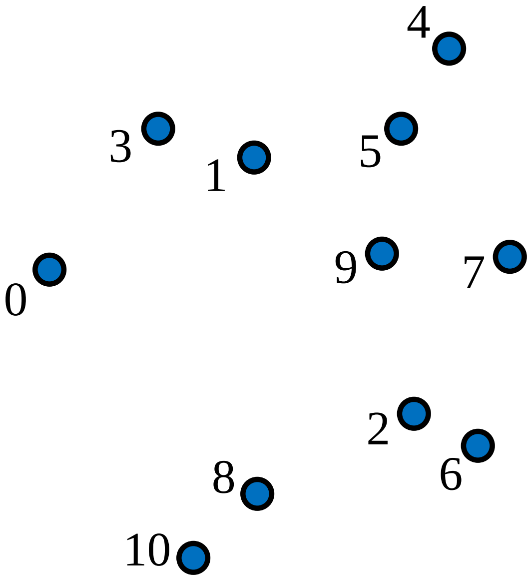
- Incremental Algorithm
- Divide-and-Conquer

Incremental Algorithm

<u>Approach</u>:

Grow the hull by iteratively adding points:

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

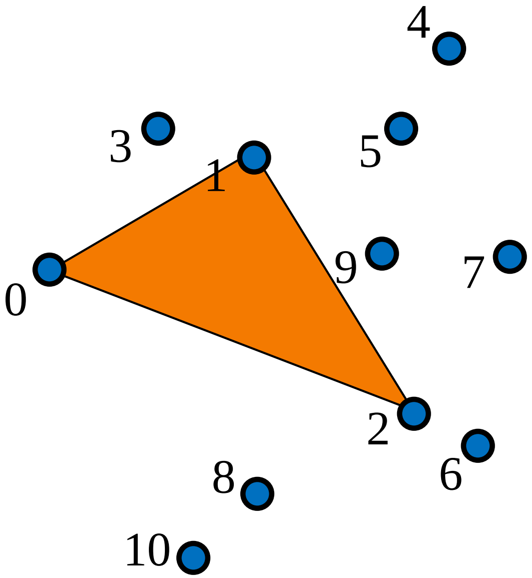


Incremental Algorithm

Approach:

Grow the hull by iteratively adding points:

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

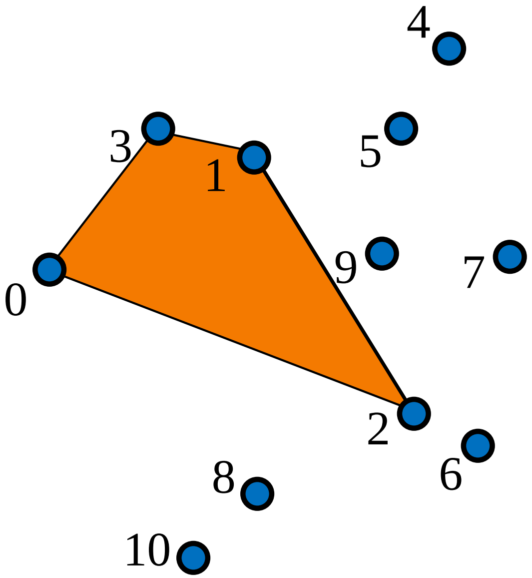


Incremental Algorithm

<u>Approach</u>:

Grow the hull by iteratively adding points:

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

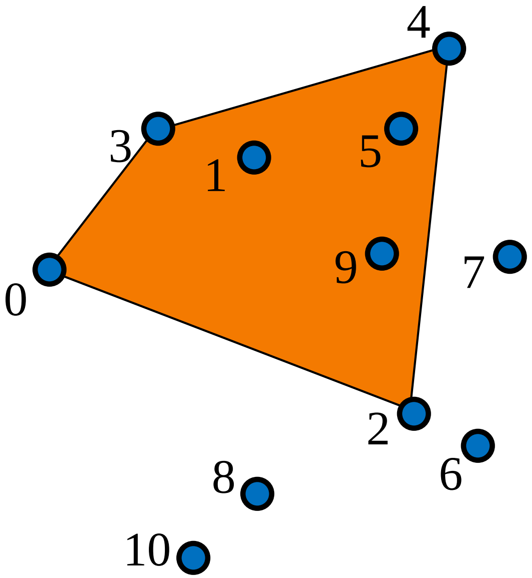


Incremental Algorithm

<u>Approach</u>:

Grow the hull by iteratively adding points:

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

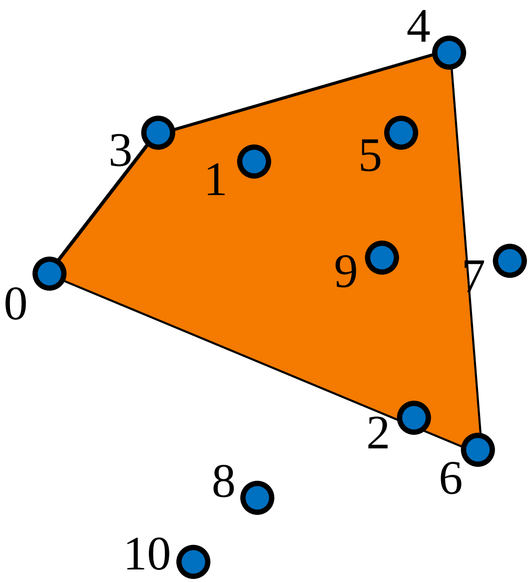


Incremental Algorithm

<u>Approach</u>:

Grow the hull by iteratively adding points:

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

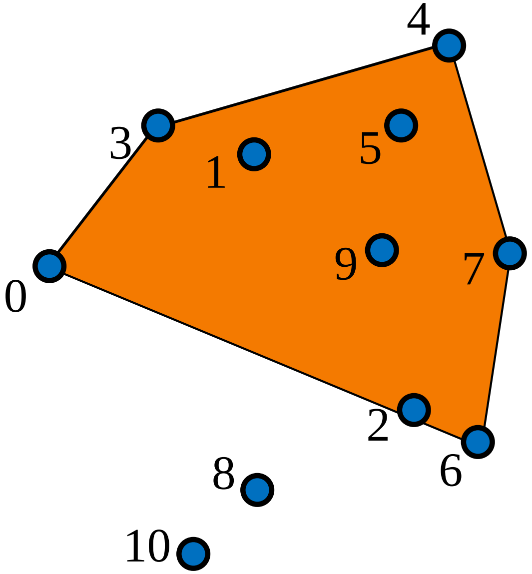


Incremental Algorithm

Approach:

Grow the hull by iteratively adding points:

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

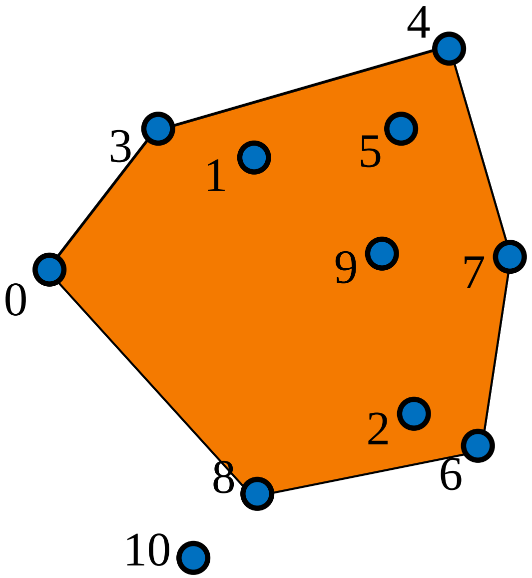


Incremental Algorithm

Approach:

Grow the hull by iteratively adding points:

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

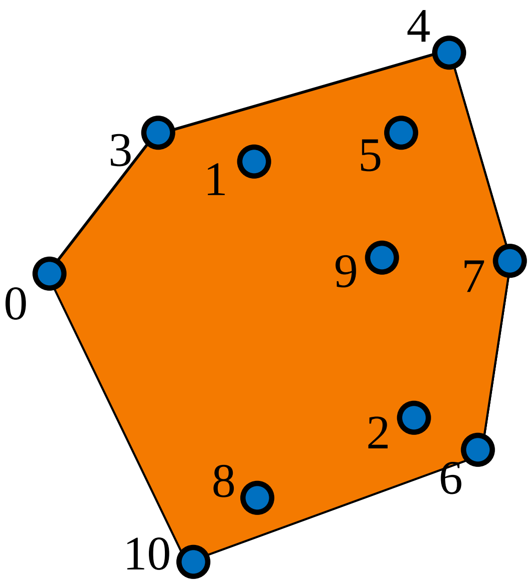


Incremental Algorithm

Approach:

Grow the hull by iteratively adding points:

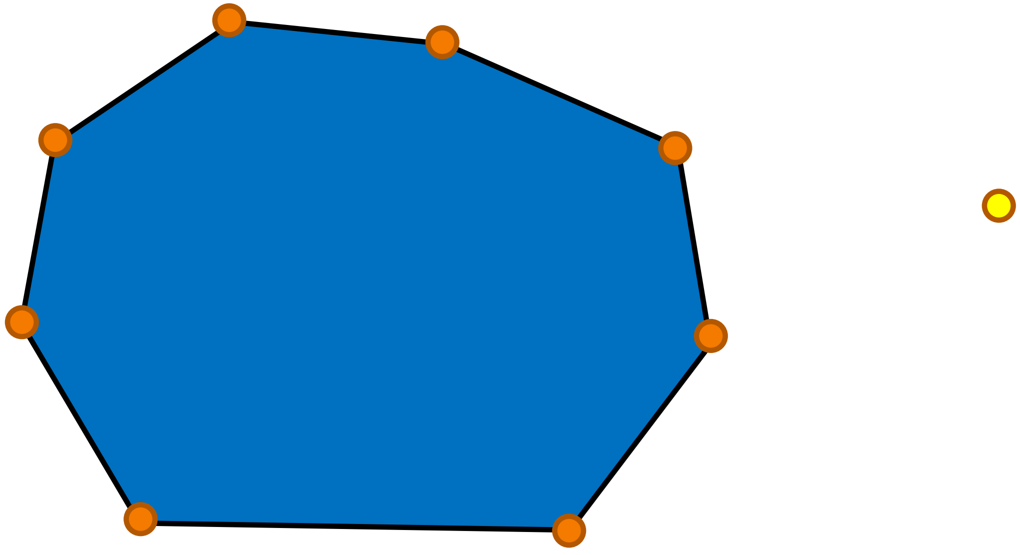
- If the point is in the hull, do nothing.
- Otherwise, grow the hull.



Incremental Algorithm

Note:

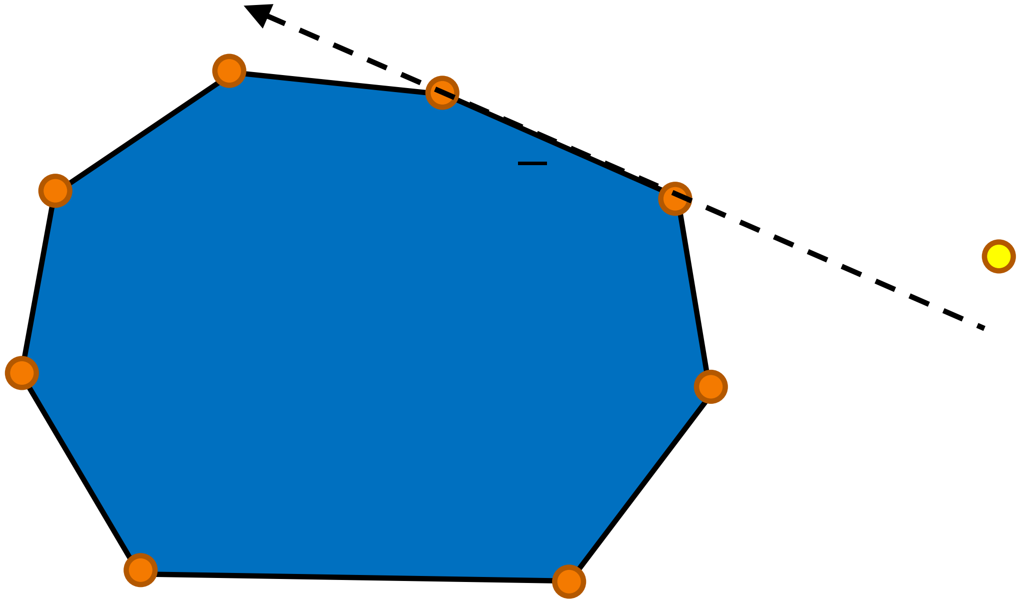
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

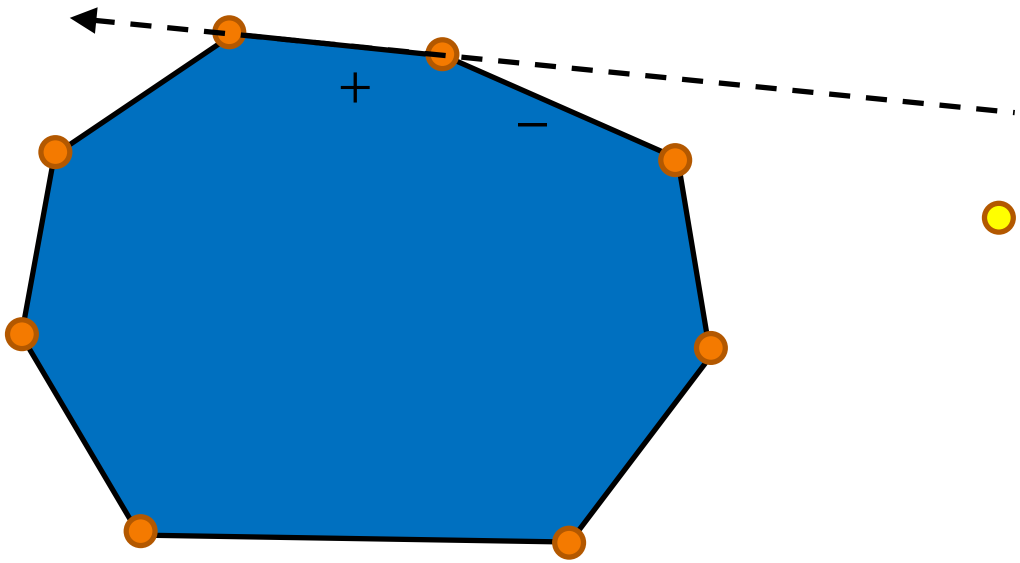
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

<u>Note</u>:

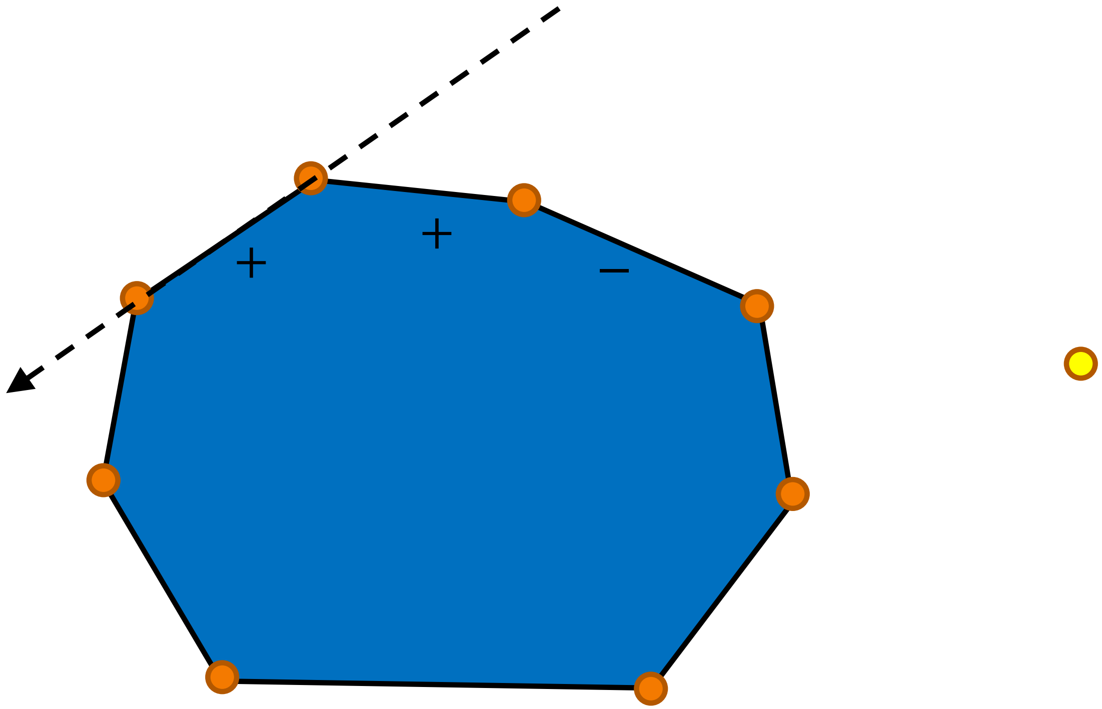
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

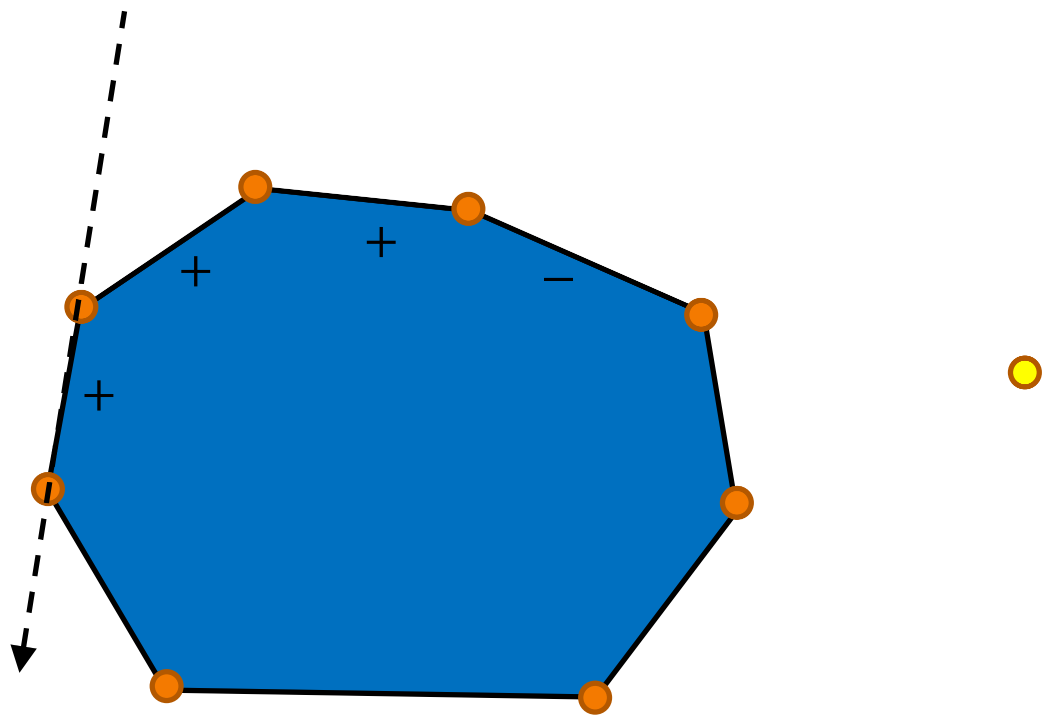
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

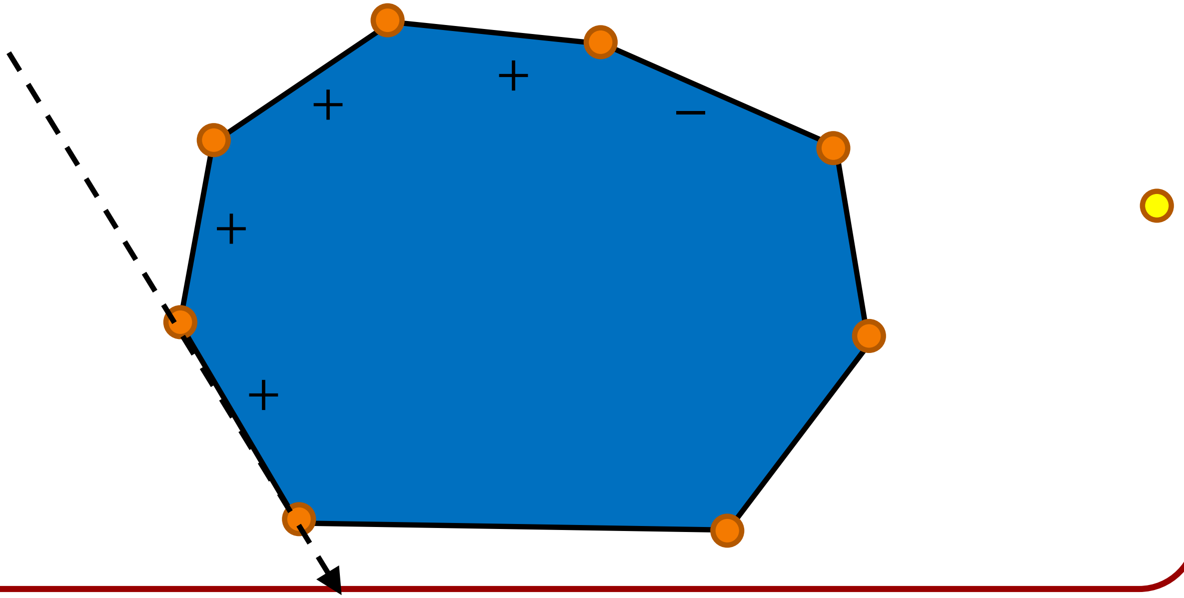
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

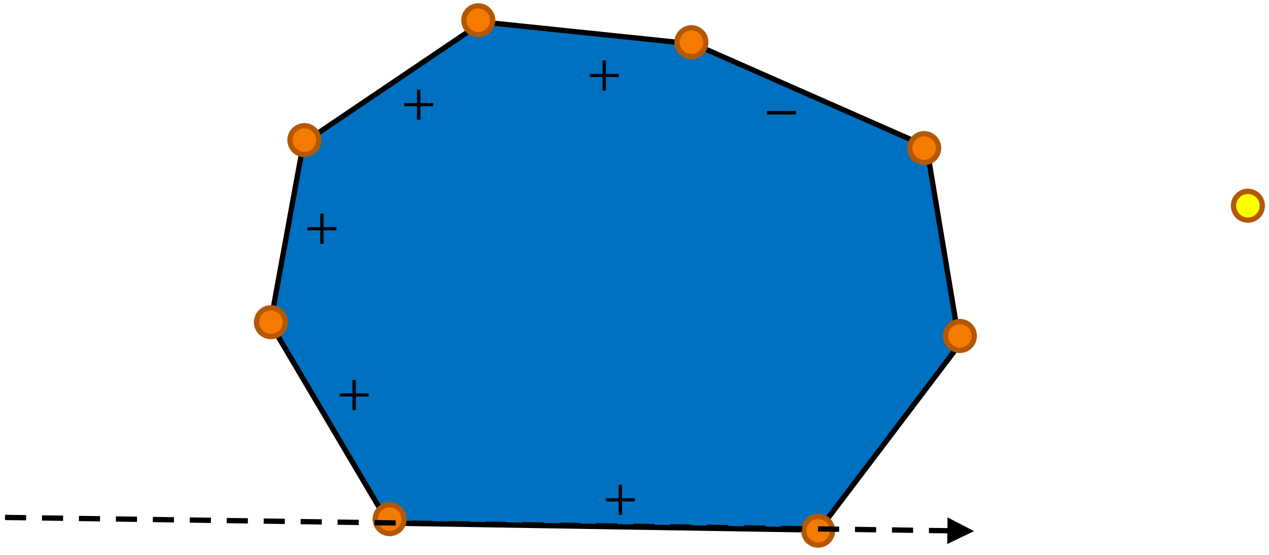
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

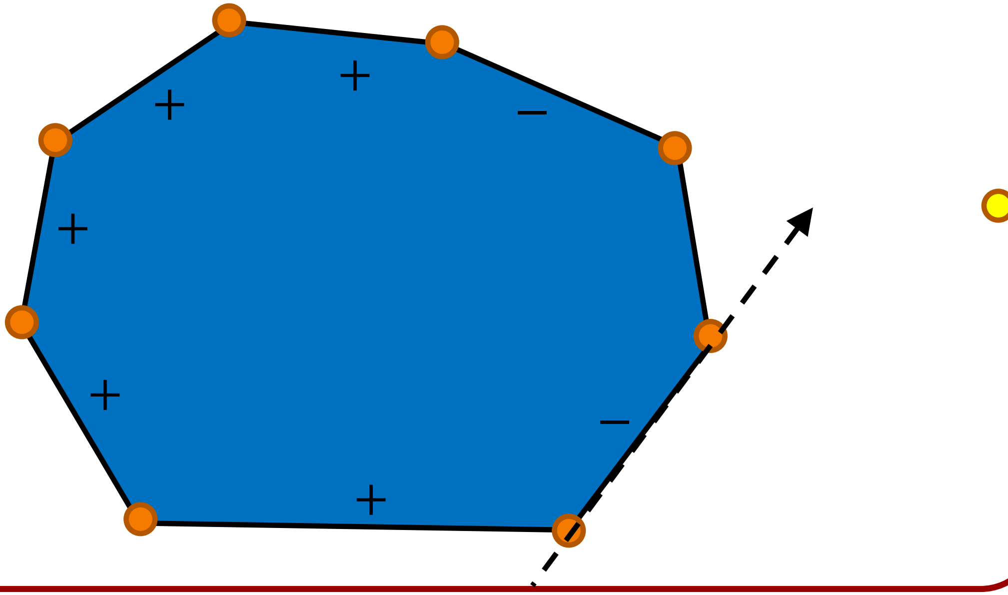
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

<u>Note</u>:

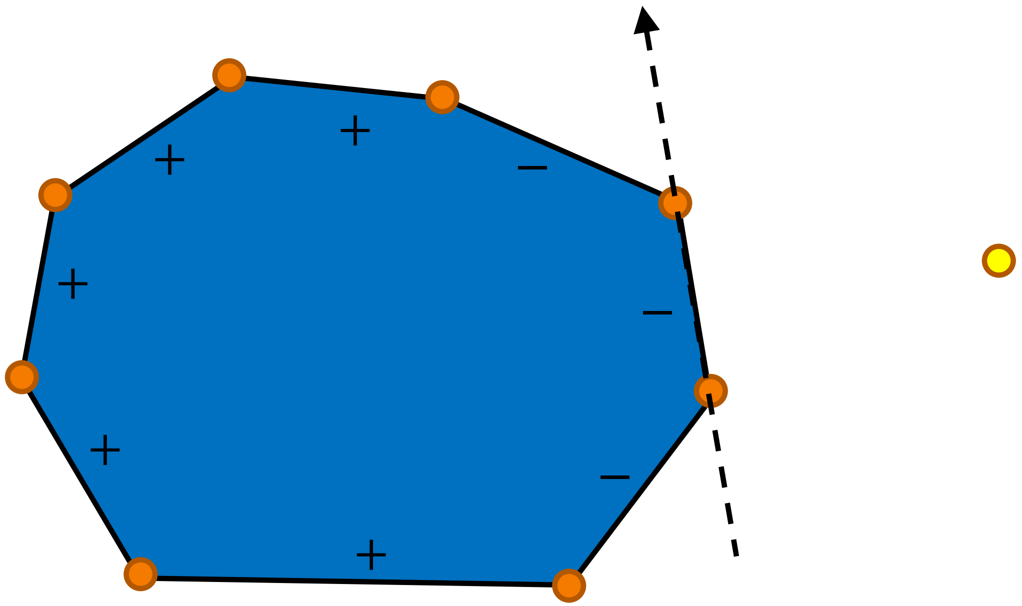
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



Incremental Algorithm

Note:

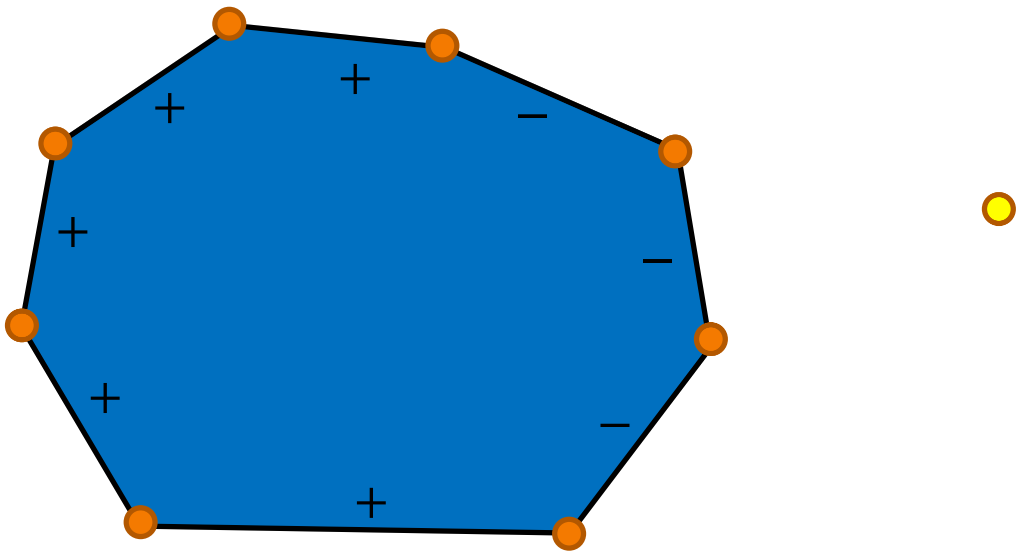
If a point is outside the hull, we can label the hull edges as left/right relative to the new point.



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.



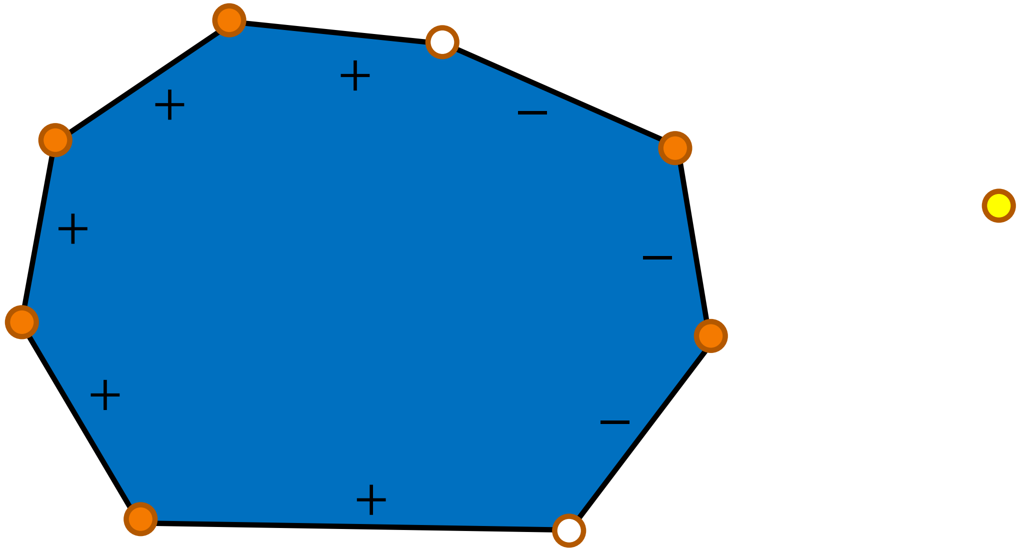
Incremental Algorithm

<u>Note</u>:

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.

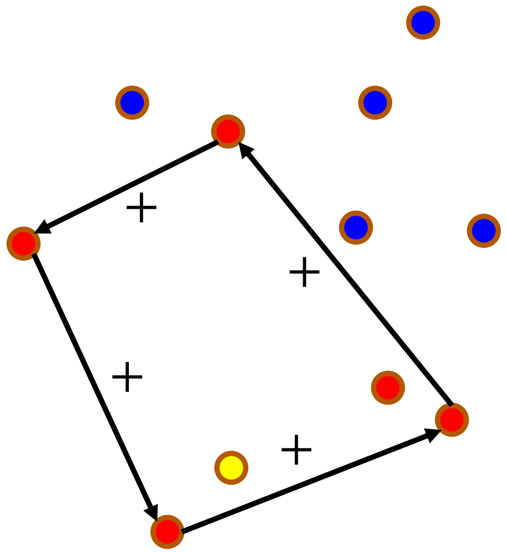


Incremental Algorithm

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.

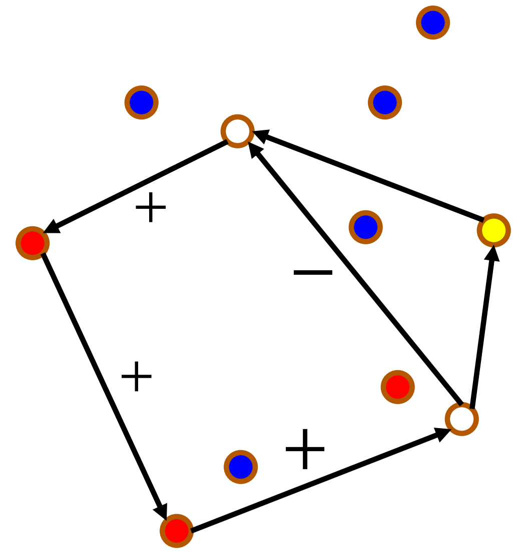


Incremental Algorithm

<u>Naïve</u>:

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.

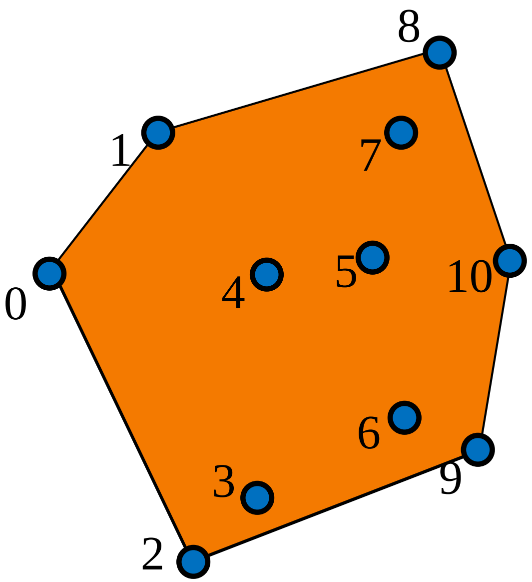


Complexity: $O(n^2)$

Incremental Algorithm

Edelsbrunner (1987):

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



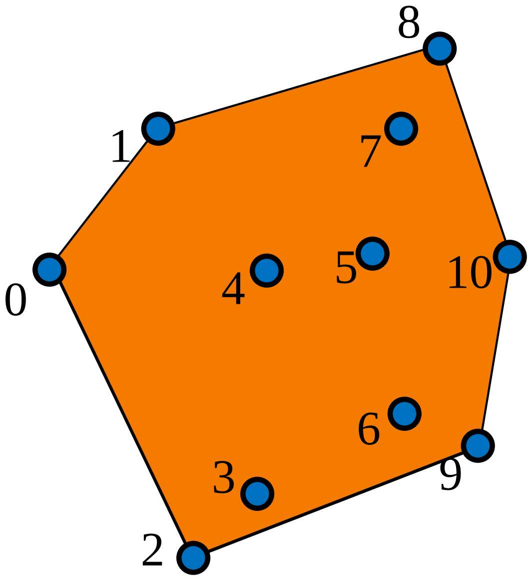
Incremental Algorithm

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.



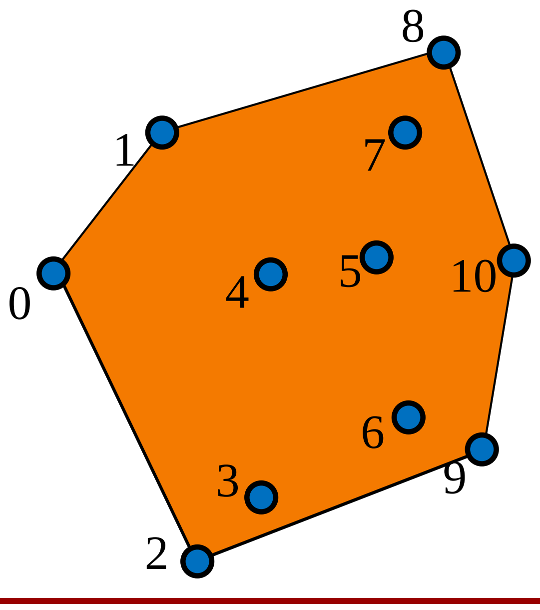
Incremental Algorithm

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.



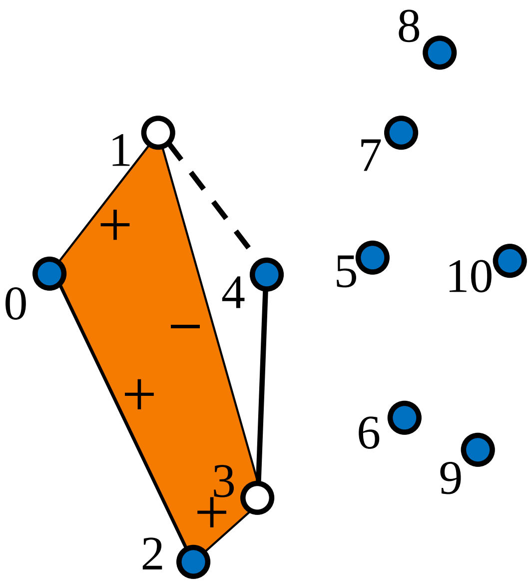
Incremental Algorithm

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.



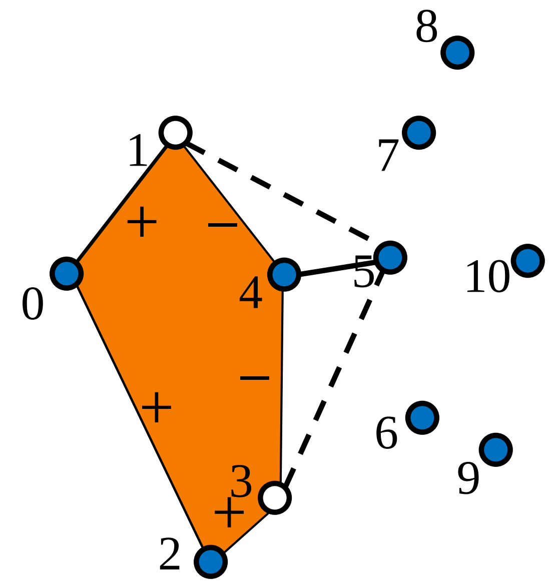
Incremental Algorithm

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.



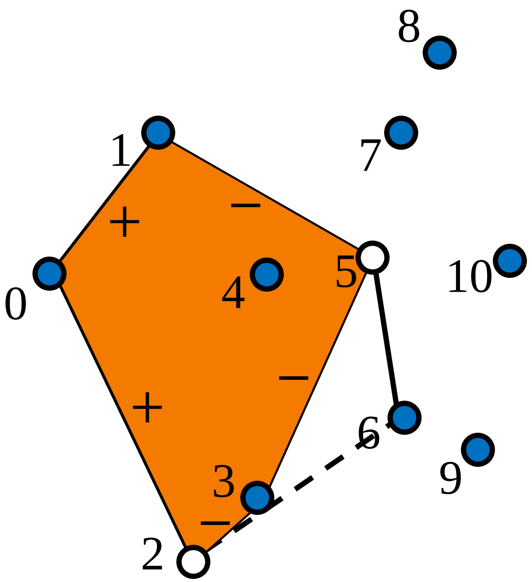
Incremental Algorithm

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.



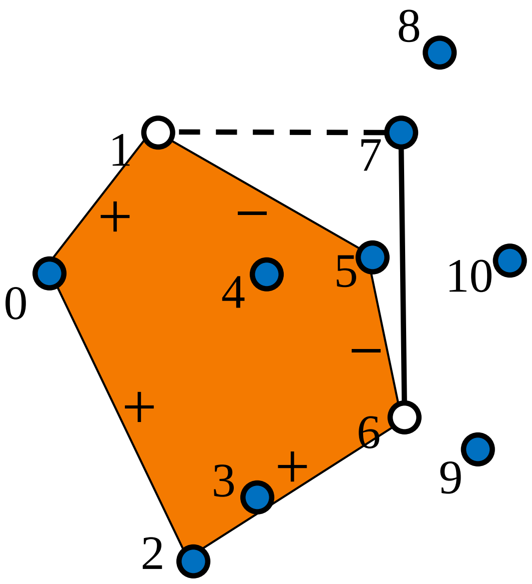
Incremental Algorithm

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.



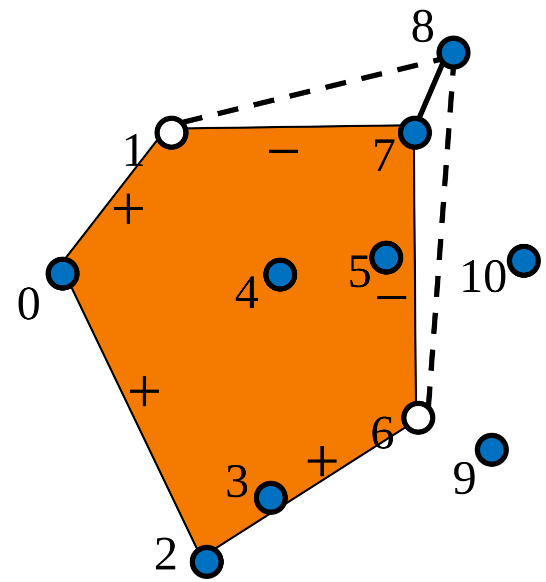
Incremental Algorithm

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.



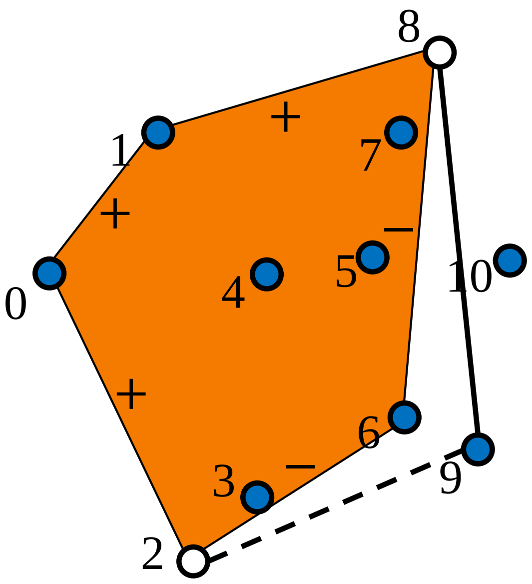
Incremental Algorithm

<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.



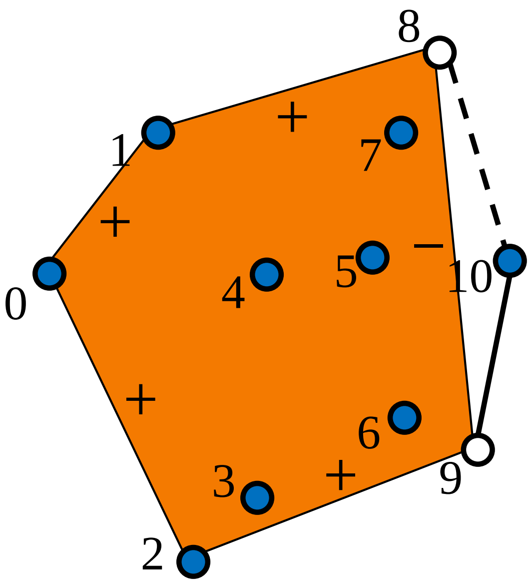
Incremental Algorithm

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.



Convex Hull (2D)

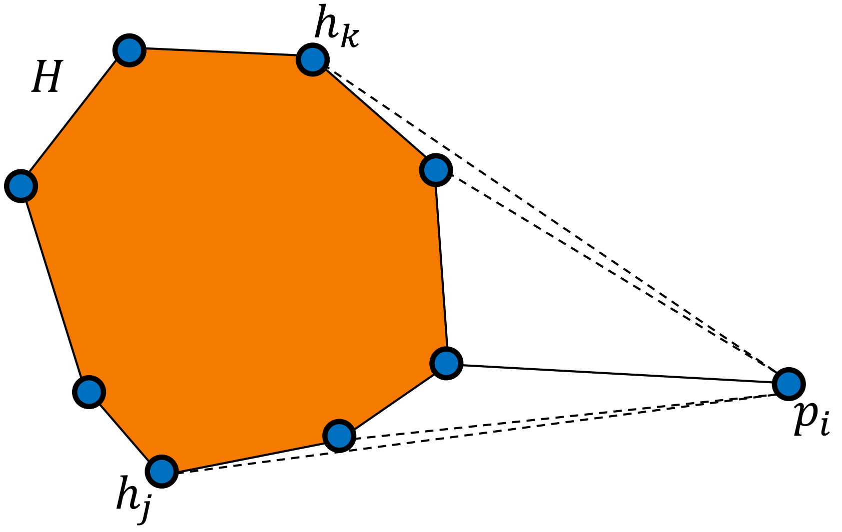
IncrementalAlgorithm(P)

- 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 , $\{h_j, \ldots, h_k\}$, $\{h_j, p_i, h_k\}$)

Convex Hull (2D)

IncrementalAlgorithm(P)

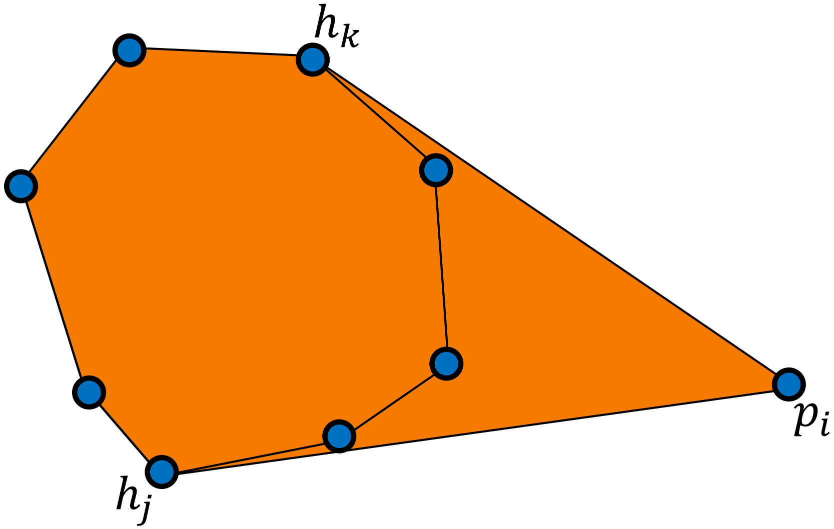
- SortLexicographically(P)
- $H \leftarrow \{p_0, p_1, p_2\}$
- for $i \in [3, n)$:
  - » $(\boldsymbol{h_j}, \boldsymbol{h_k}) \leftarrow$ **TransitionVertices(** $\boldsymbol{H}$ **,** $\boldsymbol{p_i}$ **)**
  - » Replace(H , $\{h_j, \ldots, h_k\}$, $\{h_j, p_i, h_k\}$)



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(** $\boldsymbol{H}$ **,** $\{\boldsymbol{h_j}, \dots, \boldsymbol{h_k}\}$ **,** $\{\boldsymbol{h_j}, \boldsymbol{p_i}, \boldsymbol{h_k}\}$ **)**



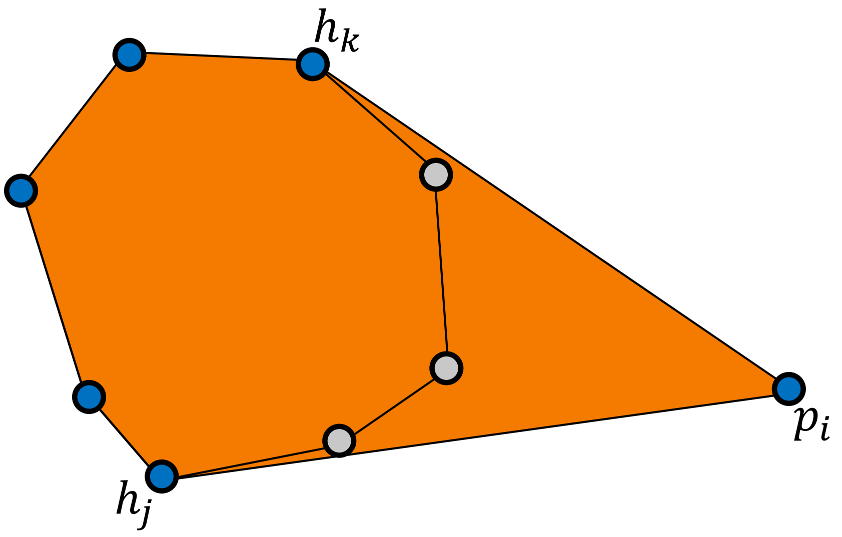
Convex Hull (2D)

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 , $\{h_j, \dots, h_k\}$, $\{h_j, p_i, h_k\}$)

← $O(?)$

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



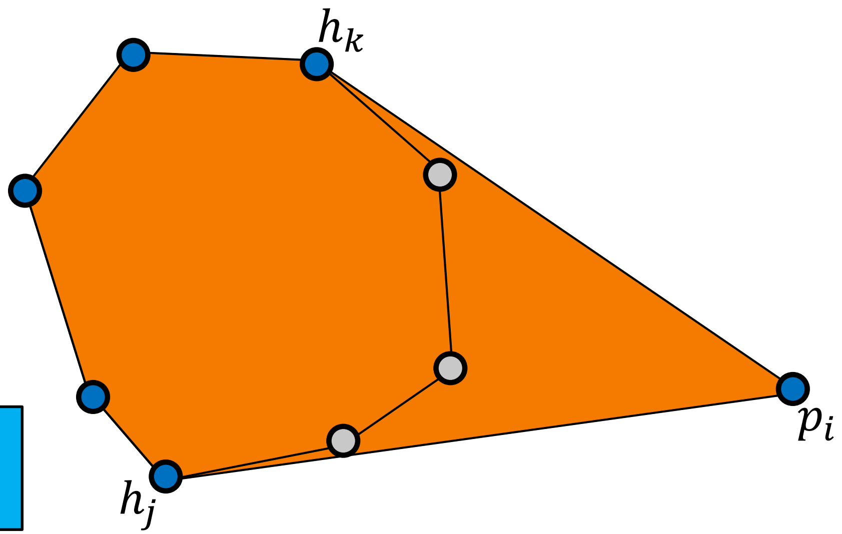
Convex Hull (2D)

IncrementalAlgorithm(P)

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

$\leftarrow O(n)$

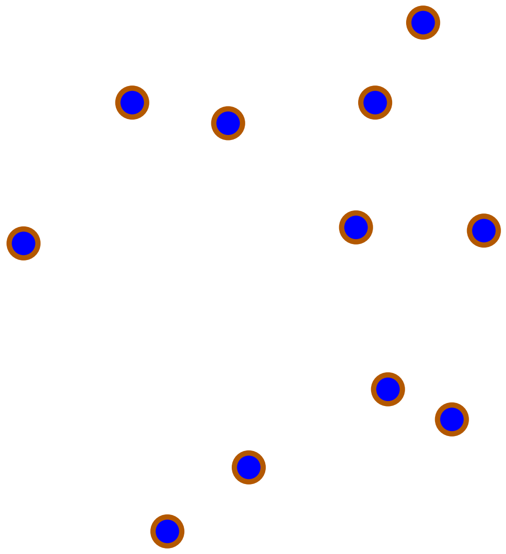
<u>Note</u>:
Any vertex traversed to find the transition vertices is removed.



Complexity: $O(n \log n)$

Outline

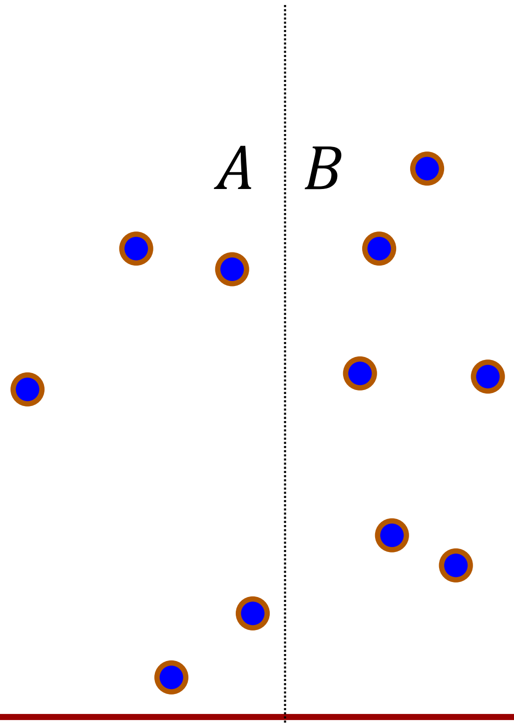
- Incremental Algorithm
- Divide-and-Conquer



Divide And Conquer

Recursively:

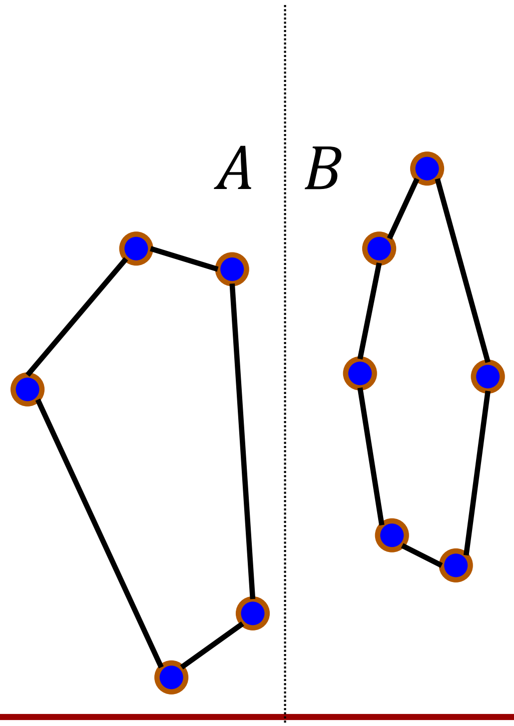
- Split the point-set in two.



Divide And Conquer

Recursively:

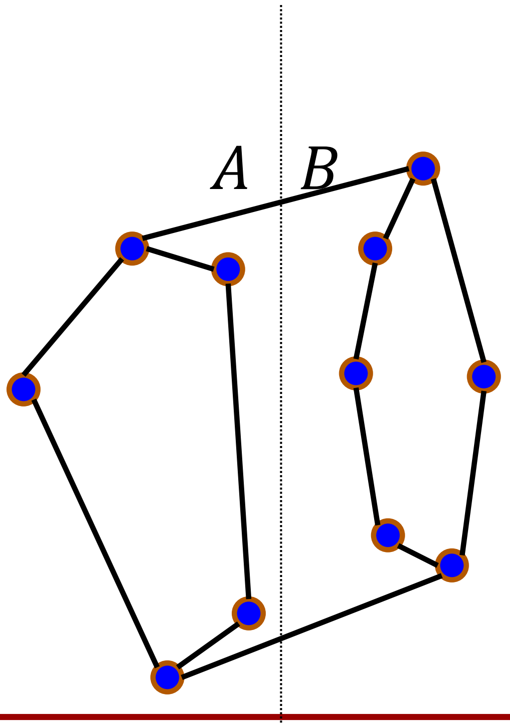
- Split the point-set in two.
- Compute the hull of both halves



Divide And Conquer

Recursively:

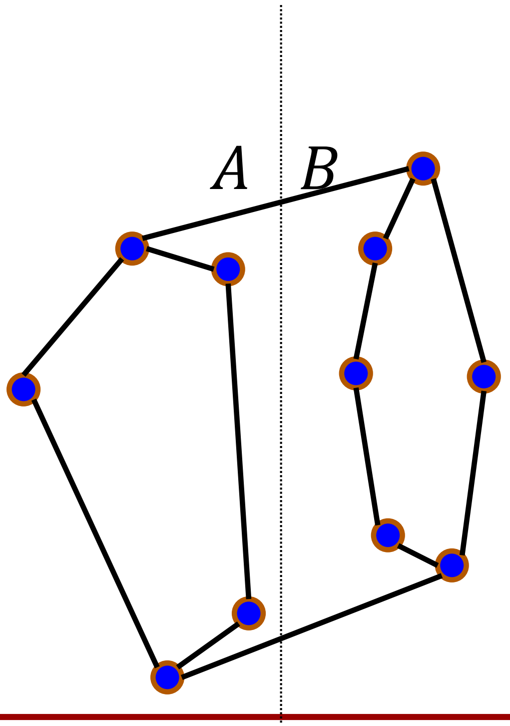
- Split the point-set in two.
- Compute the hull of both halves
- Merge the hulls



Divide And Conquer

Efficiency:

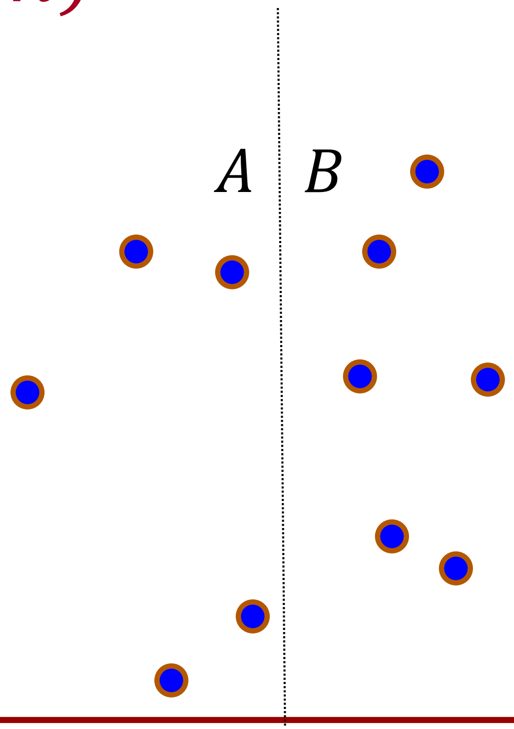
For this to be fast (log-linear), the splitting and merging have to be fast (linear).



Divide And Conquer (Step 1)

Split the point-set in two:

- 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($S = \{x_0, \dots, x_{n-1}\}$):

- return KthEntry(S , $|S|/2$)

KthEntry($S = \{x_0, \dots, x_{n-1}\}$, k):

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

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

KthEntry($S = \{x_0, \dots, x_{n-1}\}$, s):

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

Claim:

- The subsets L and R defined by:
  (L , R) ← Split(S , FastMedian(Q))
  have the property that $|L|, |R| \leq 4|S|/5$

Fast Median

Claim:

- The subsets L and R defined by:

  $(L , R) \leftarrow$ Split(S , KthEntry(Q))

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

Proof:

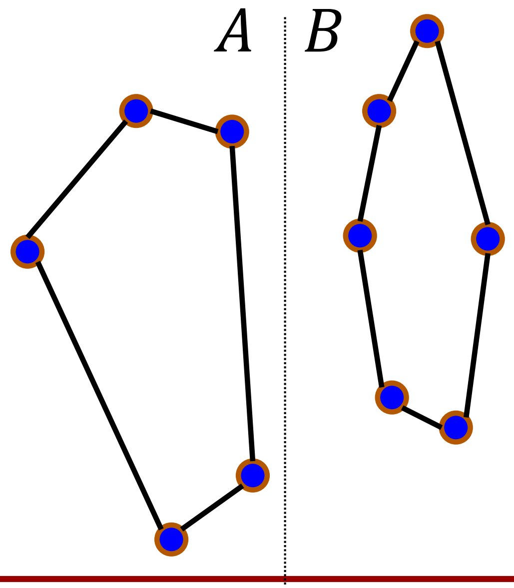
- Set $q =$ FastMedian(Q)
- The subset of $q_i \in Q$ with $q_i < q$ makes up 50% of Q.
  - » The subset of $p \in Q_i$ with $p < q_i$ makes up 40% of Q_i.

  $\Rightarrow$ Since the subset $\{p \in S | p < q_i < q\}$ is in L, the set L contains at least one fifth of the points in S.
- The subset of $q_i \in Q$ with $q_i \geq q$ makes up 50% of Q…

Divide And Conquer (Step 2)

Compute the hull of the halves:

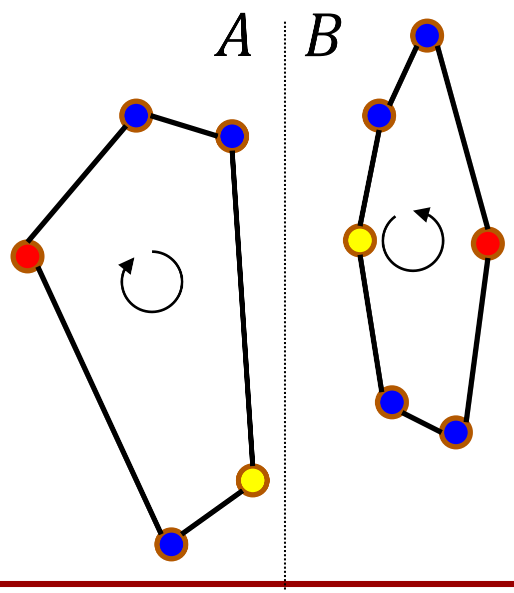
- If the subset has less than 6 points, apply the incremental algorithm,
- Otherwise recurse.



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.

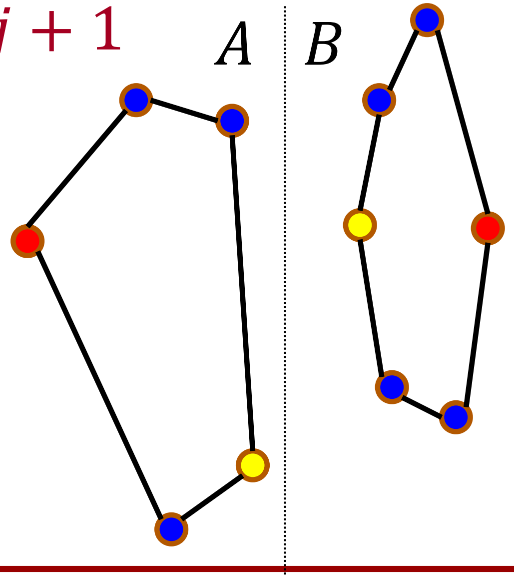


*Assuming general position

Merging the Hulls (lower tangent)

Merge (A , B):

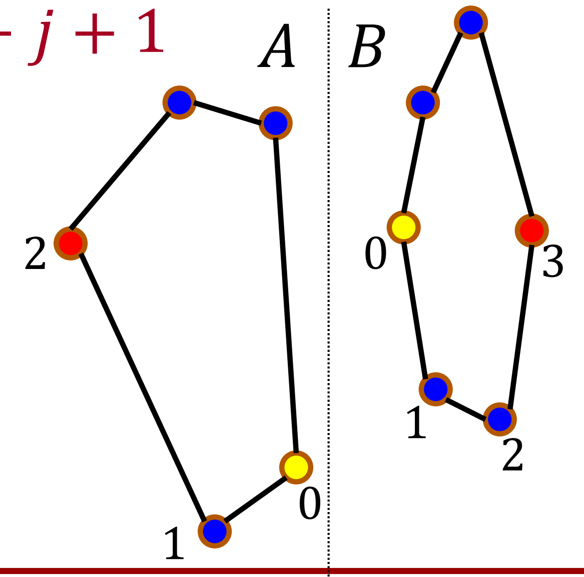
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

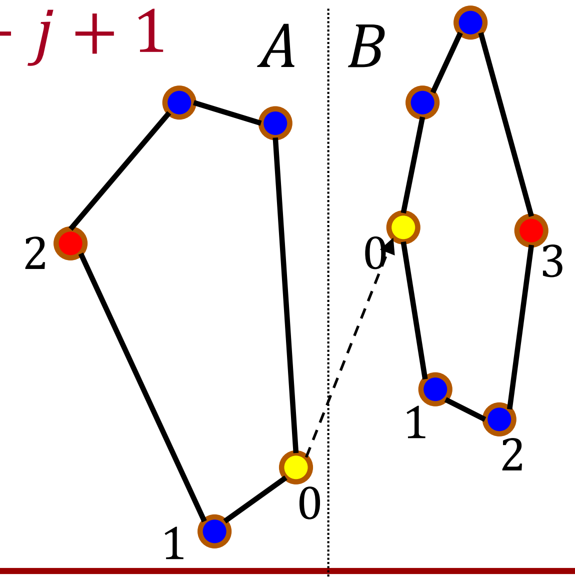
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

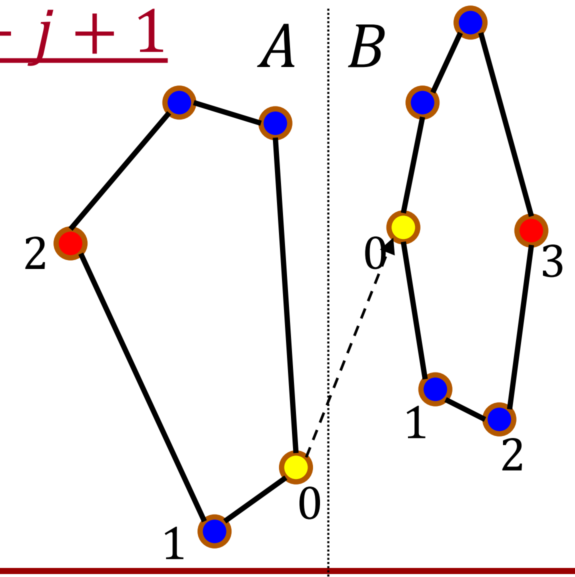
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

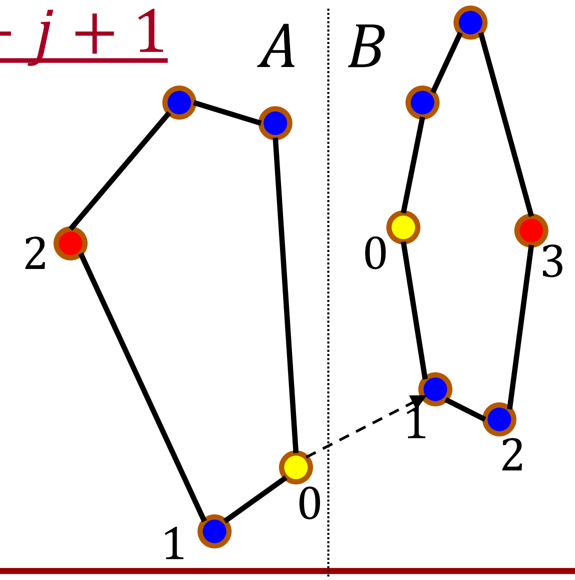
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i, j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

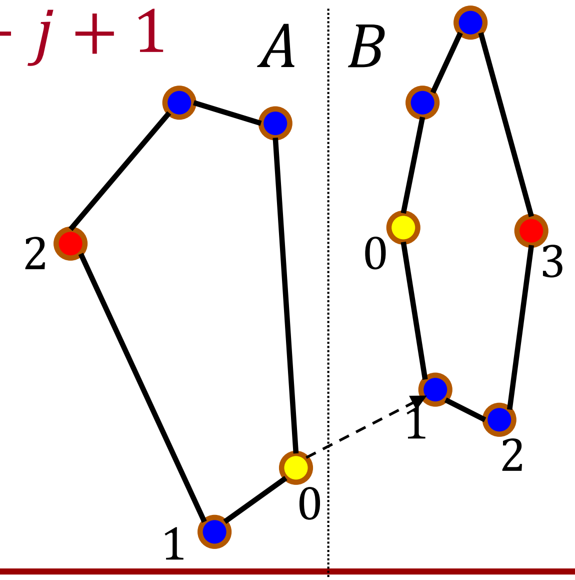
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - <u>else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$</u>
  - else: break



Merging the Hulls (lower tangent)

Merge (A , B):

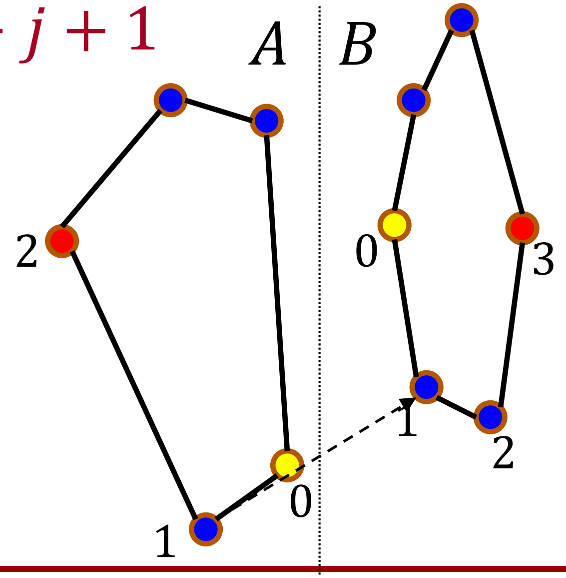
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

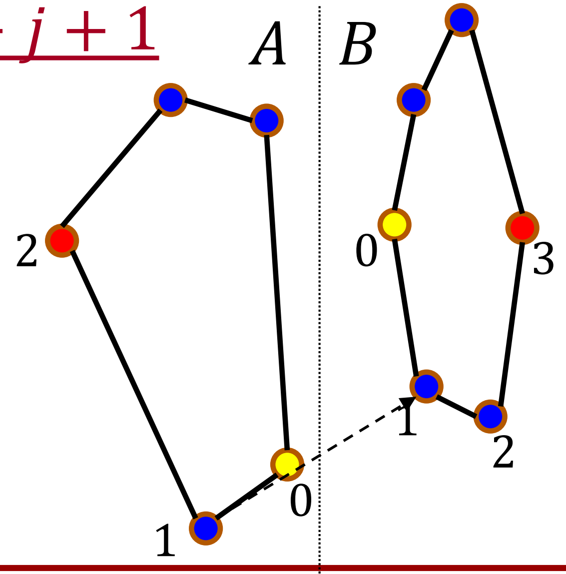
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

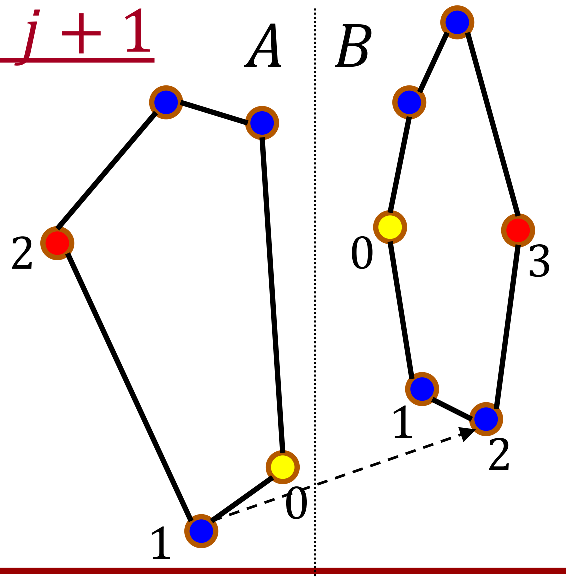
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i, j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i + 1$
  - » <u>else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j + 1$</u>
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

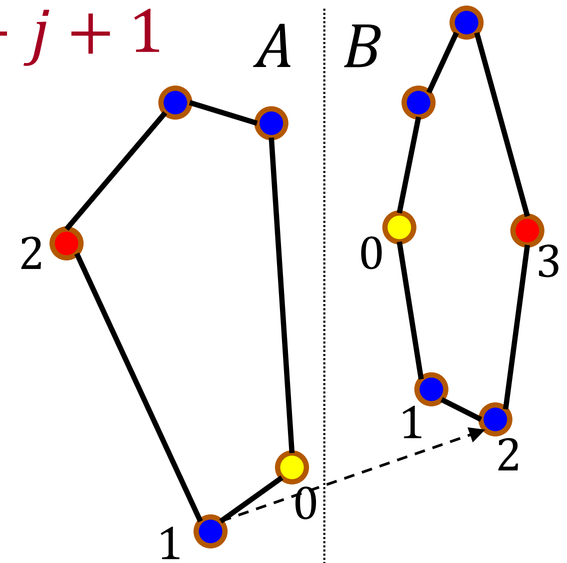
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i, j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i + 1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j + 1$
  - » else: break



Merging the Hulls (lower tangent)

Merge (A , B):

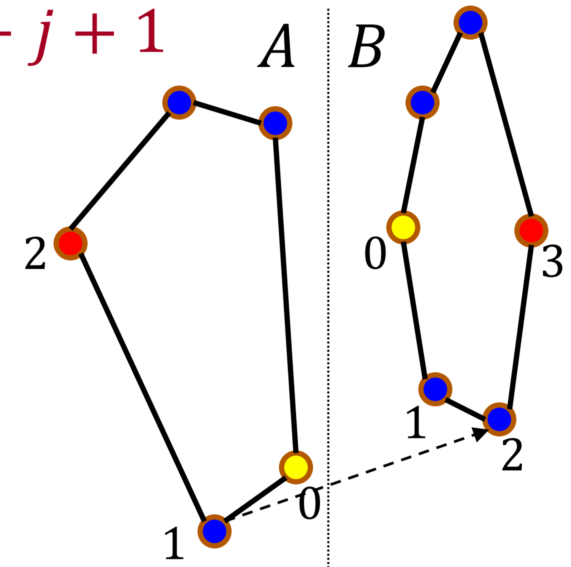
- $A \leftarrow$ SortCWFromRight(A)
- $B \leftarrow$ SortCCWFromLeft(B)
- $(i,j) \leftarrow (0,0)$
- while(true)
  - » if (Right($\overrightarrow{a_i b_j}$, a_{i+1})): $i \leftarrow i+1$
  - » else if(Right($\overrightarrow{a_i b_j}$, b_{j+1})): $j \leftarrow j+1$
  - » else: break



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 this terminates:

1. at the lower tangent
2. in linear time.

Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

First 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.

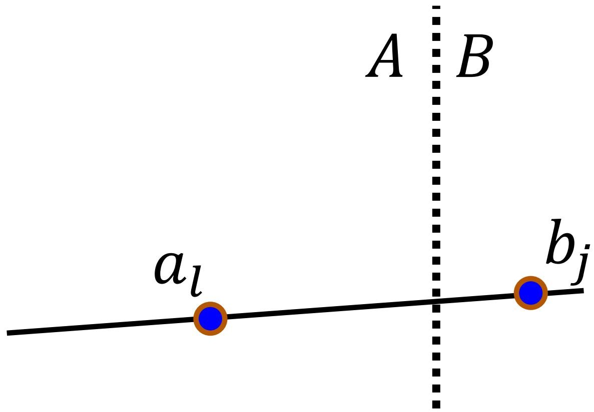
Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

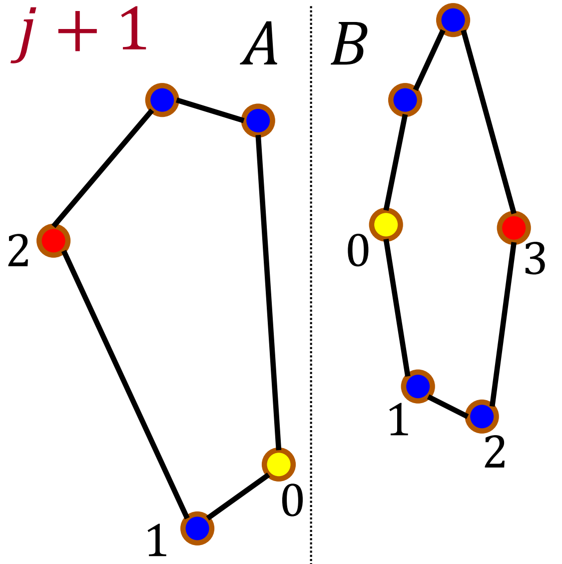
Will show that i won't pass the left-most vertex, a_l.



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)

Claim:

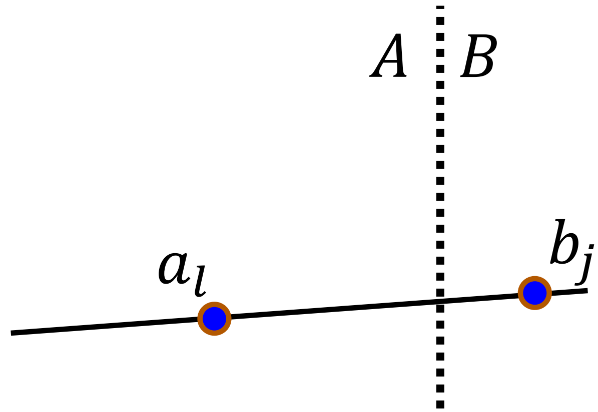
If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

Will show that i won't pass the left-most vertex, a_l.

$\Leftrightarrow$ Right($\overrightarrow{a_l b_j}$, a_{l+1})==false

Where can a_{l-1} be?



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

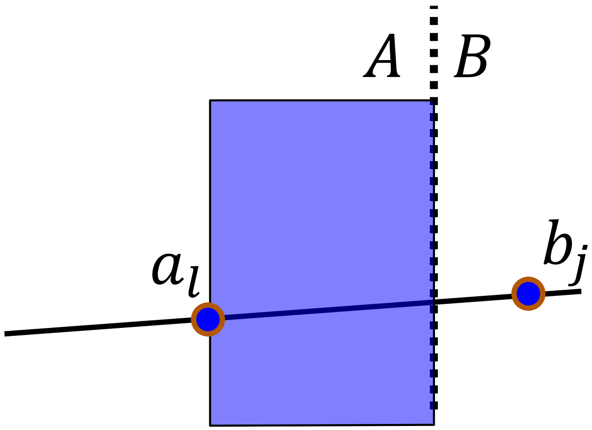
Will show that i won't pass the left-most vertex, a_l.

$\Leftrightarrow$ Right($\overrightarrow{a_l b_j}$, a_{l+1})==false

Where can a_{l-1} be?

Because a_l is left-most:

$a_{l-1} \in \{p | p^x > a_l^x\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

Will show that i won't pass the left-most vertex, a_l.

$\Leftrightarrow$ Right($\overrightarrow{a_l b_j}, a_{l+1}$)==false

Note that $l \neq 0$ because l indexes the left-most vertex in A while 0 indexes the right-most.

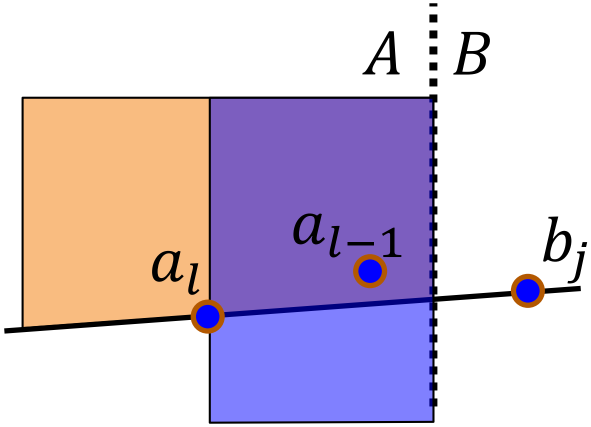
Where can a_{l-1} be?

Because a_l is left-most:

$a_{l-1} \in \{p | p^x > a_l^x\}$

Because the claim holds:

$a_{l-1} \in \{p | \text{Left}(\overrightarrow{a_l b_j}, p)\}$



Merging the Hulls (lower tangent)

Claim:

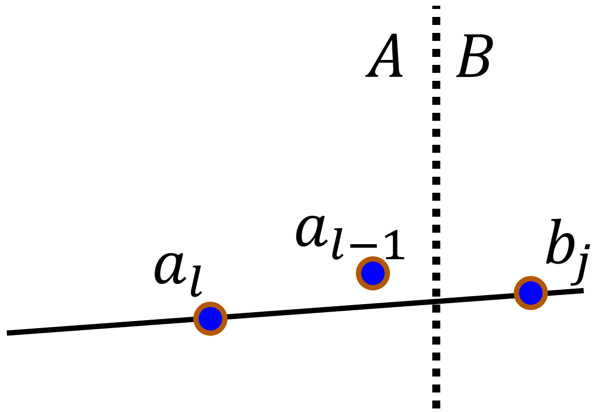
If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

Will show that i won't pass the left-most vertex, a_l.

$\Leftrightarrow$ Right($\overrightarrow{a_l b_j}$, a_{l+1})==false

Where can a_{l+1} be?



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

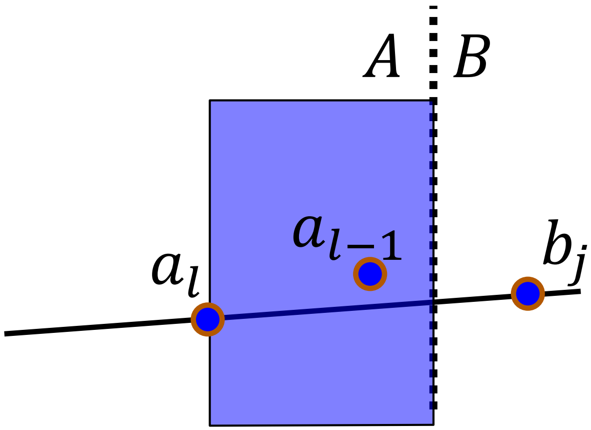
Will show that i won't pass the left-most vertex, a_l.

$\Leftrightarrow$ Right($\overrightarrow{a_l b_j}$, a_{l+1})==false

Where can a_{l+1} be?

Because a_l is left-most:

$a_{l+1} \in \{p | p^x > a_l^x\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.

Will show that i won't pass the left-most vertex, a_l.

$\Leftrightarrow$ Right($\overrightarrow{a_l b_j}$, a_{l+1})==false

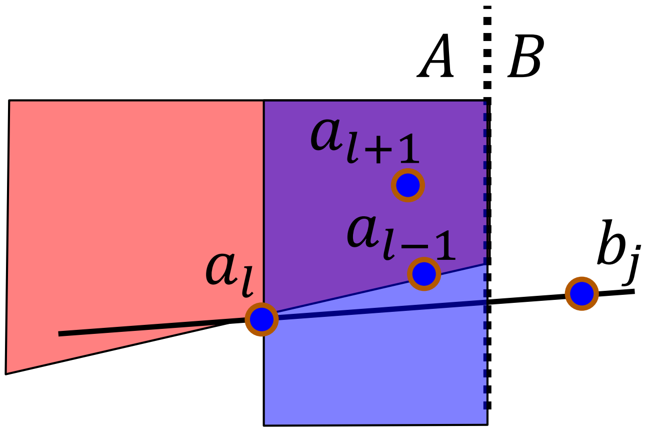
Where can a_{l+1} be?

Because a_l is left-most:

$a_{l+1} \in \{p | p^x > a_l^x\}$

Because a_l is convex:

$a_{l+1} \in \{p | \text{Left}(\overrightarrow{a_l a_{l-1}}, p)\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

Merging the Hulls (lower tangent)

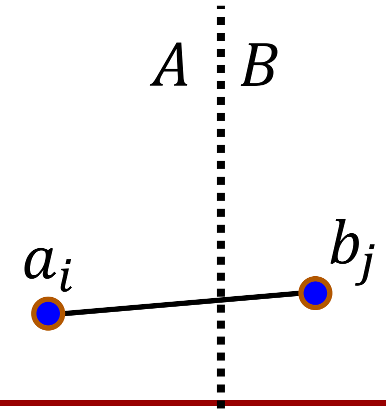
<u>Claim</u>:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

<u>Case $i \neq 0$</u>:



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

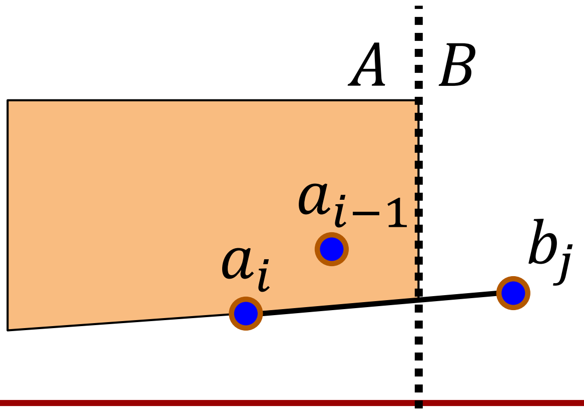
1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

Case $i \neq 0$:

By claim #1:

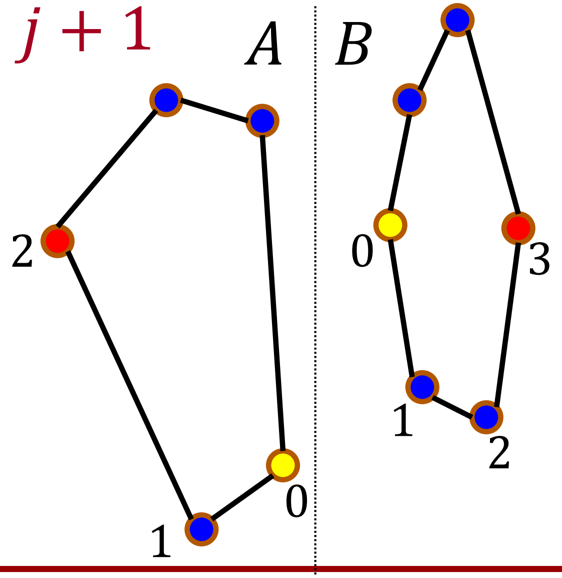
$a_{i-1} \in \{p \mid \mathrm{Left}(\overrightarrow{a_i b_j}, p)\}$



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)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

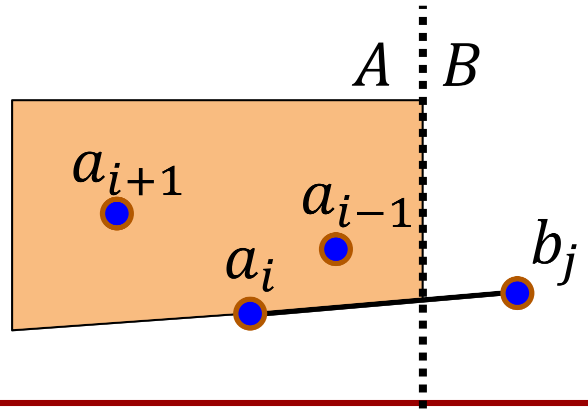
Case $i \neq 0$:

By claim #1:

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

Because we terminated:

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



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

Case $i \neq 0$:

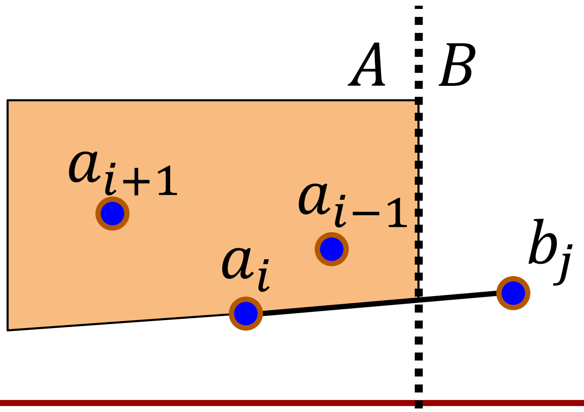
By claim #1:

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

Because we terminated:

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

$\Rightarrow \overrightarrow{a_i b_j}$ is a lower tangent of A.



Merging the Hulls (lower tangent)

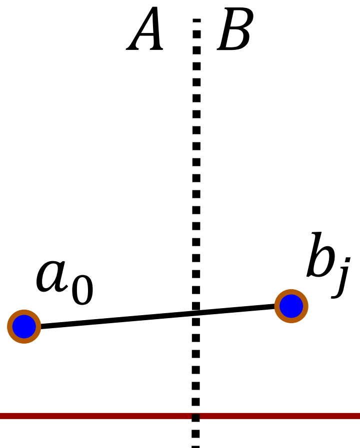
Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ 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:

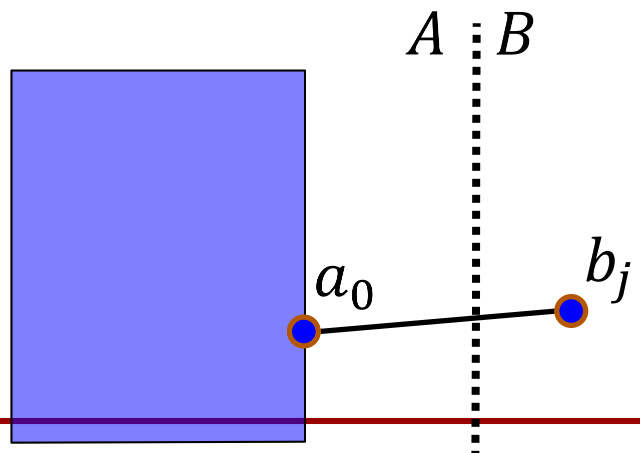
1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

Case $i = 0$:

Because a_0 is right-most:

$a_1 \in \{p | p^x < a_0^x\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

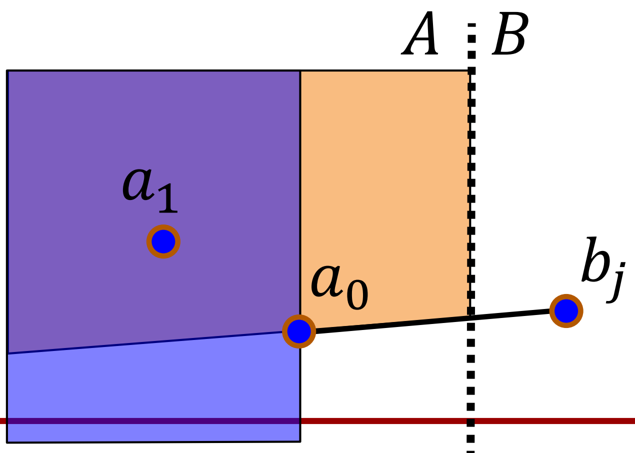
Case $i = 0$:

Because a_0 is right-most:

$a_1 \in \{p \mid p^x < a_0^x\}$

Because we terminated:

$a_1 \in \{p \mid \text{Left}(\overrightarrow{a_0 b_j}, p)\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

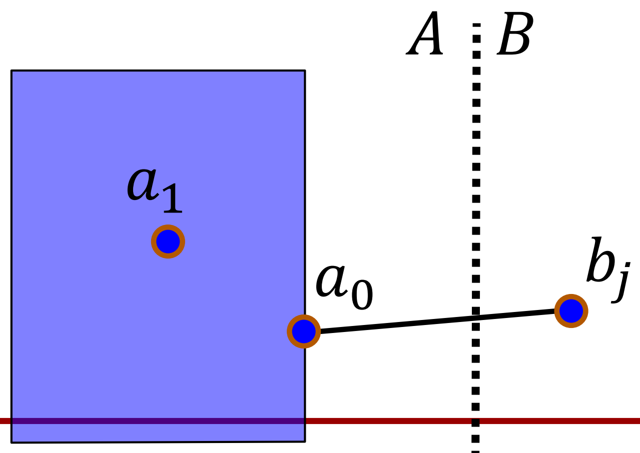
1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

Case $i = 0$:

Because a_0 is right-most:

$a_{n-1} \in \{p | p^x < a_0^x\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

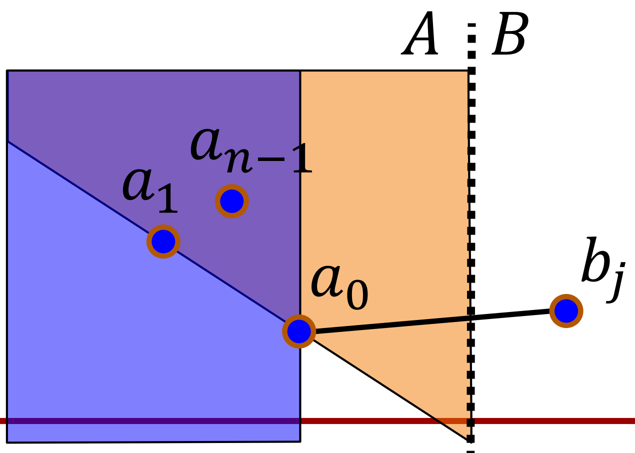
Case $i = 0$:

Because a_0 is right-most:

$a_{n-1} \in \{p \mid p^x < a_0^x\}$

Because A is convex:

$a_{n-1} \in \{p \mid \mathrm{Left}(\overrightarrow{a_1 a_0}, p)\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Will show that at termination, $\overrightarrow{a_i b_j}$ is a lower tangent.

Case $i = 0$:

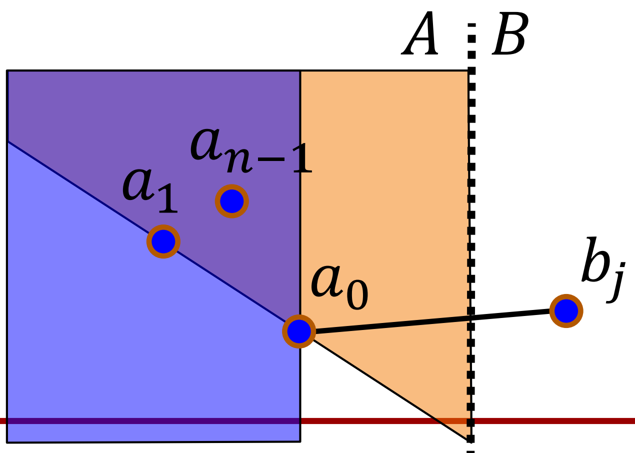
Because a_0 is right-most:

$a_{n-1} \in \{p | p^x < a_0^x\}$

Because A is convex:

$a_{n-1} \in \{p | \mathrm{Left}(\overrightarrow{a_1 a_0}, p)\}$

$\Rightarrow \overrightarrow{a_0 b_j}$ is a lower tangent of A.



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Proof by induction, $(i, j) = (0,0)$:

Both parts of the claim are trivially satisfied.

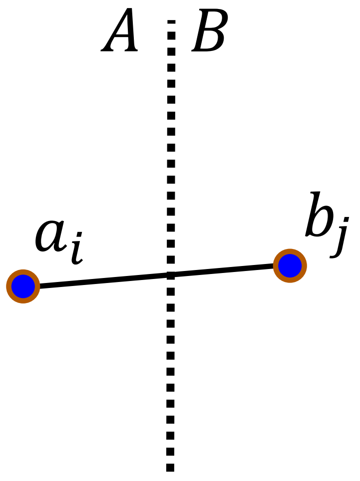
Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

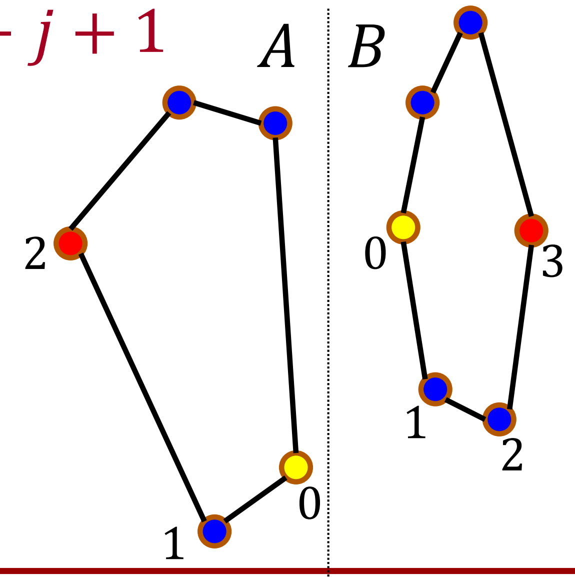
Proof by induction, $(i, j) \to (i + 1, j)$:



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)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Proof by induction #1, $(i, j) \to (i + 1, j)$:

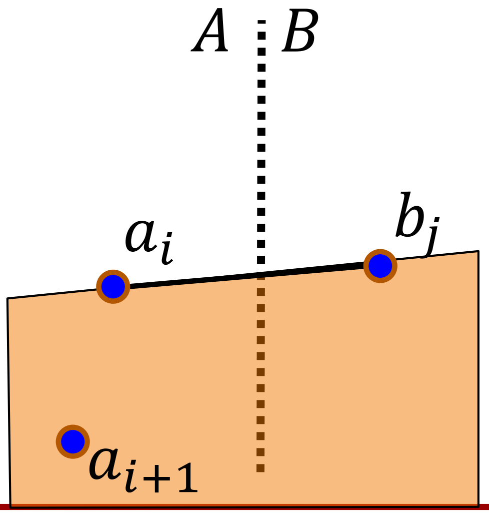
Since we advance on A:

$$a_{i+1} \in \left\{ p \middle| \mathrm{Right}\left(\overrightarrow{a_i b_j}, p\right) \right\}$$

Or, equivalently:

$$a_i \in \left\{ p \middle| \mathrm{Left}\left(\overrightarrow{a_{i+1} b_j}, p\right) \right\}$$

$\Rightarrow$ Claim #1 remains true.



Merging the Hulls (lower tangent)

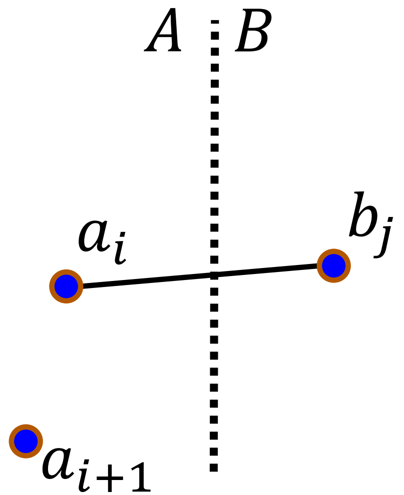
Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Proof by induction #2, $(i,j) \to (i+1,j), j = 0$:

- Claim #2 remains true.



Merging the Hulls (lower tangent)

Claim:

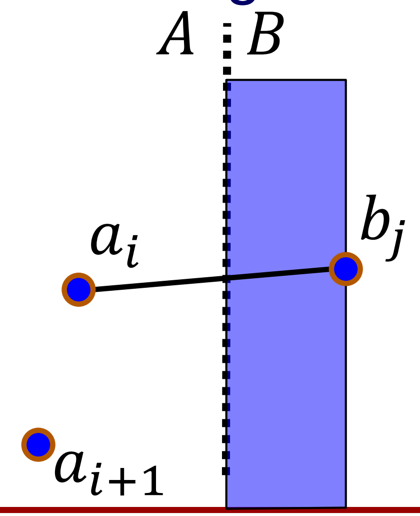
If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Proof by induction #2, $(i, j) \to (i+1, j), j \neq 0$:

As b_0 is left-most and we terminate before the right-most:

$b_{j-1} \in \{p \mid p^x < b_j^x\}$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

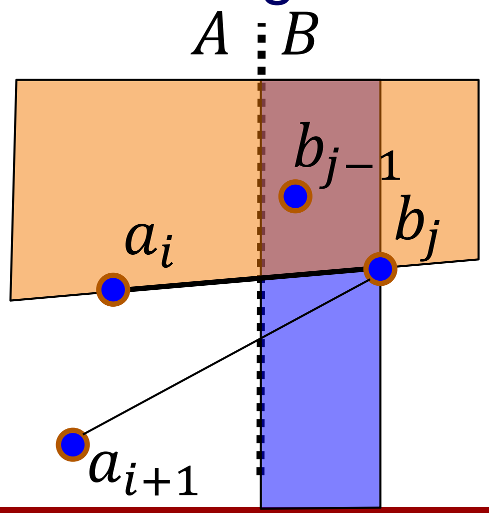
Proof by induction #2, $(i, j) \to (i + 1, j), j \neq 0$:

As b_0 is left-most and we terminate before the right-most:

$$b_{j-1} \in \{p \mid p^x < b_j^x\}$$

By the induction hypothesis:

$$b_{j-1} \in \left\{p \,\middle|\, \text{Left}\left(\overrightarrow{a_i b_j}, p\right)\right\}$$



Merging the Hulls (lower tangent)

Claim:

If edge $\overline{a_i b_j}$ connects A and B, then:

1. Either $i = 0$ or a_{i-1} is left of $\overrightarrow{a_i b_j}$.
2. Either $j = 0$ or b_{j-1} is left of $\overrightarrow{a_i b_j}$.

Proof by induction #2, $(i, j) \to (i + 1, j), j \neq 0$:

As b_0 is left-most and we terminate before the right-most:

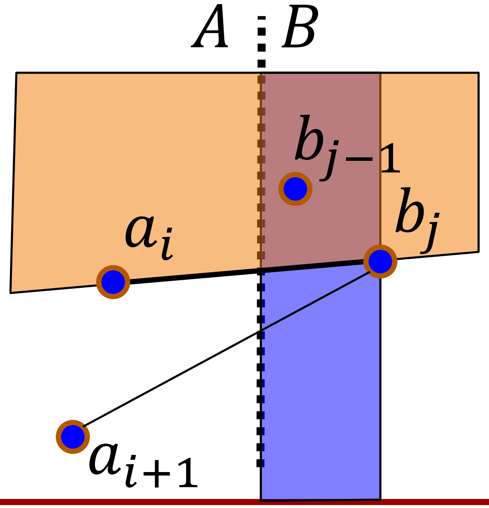
$$b_{j-1} \in \left\{p \middle| p^x < b_j^x\right\}$$

By the induction hypothesis:

$$b_{j-1} \in \left\{p \middle| \mathrm{Left}\left(\overrightarrow{a_i b_j}, p\right)\right\}$$

$$\Rightarrow b_{j-1} \in \left\{p \middle| \mathrm{Left}\left(\overrightarrow{a_{i+1} b_j}, p\right)\right\}$$

- Claim #2 remains true.



Merging the Hulls (lower tangent)

Complexity:

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

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