CS6161: Design and Analysis of Algorithms (Fall 2020)

Convex Hull

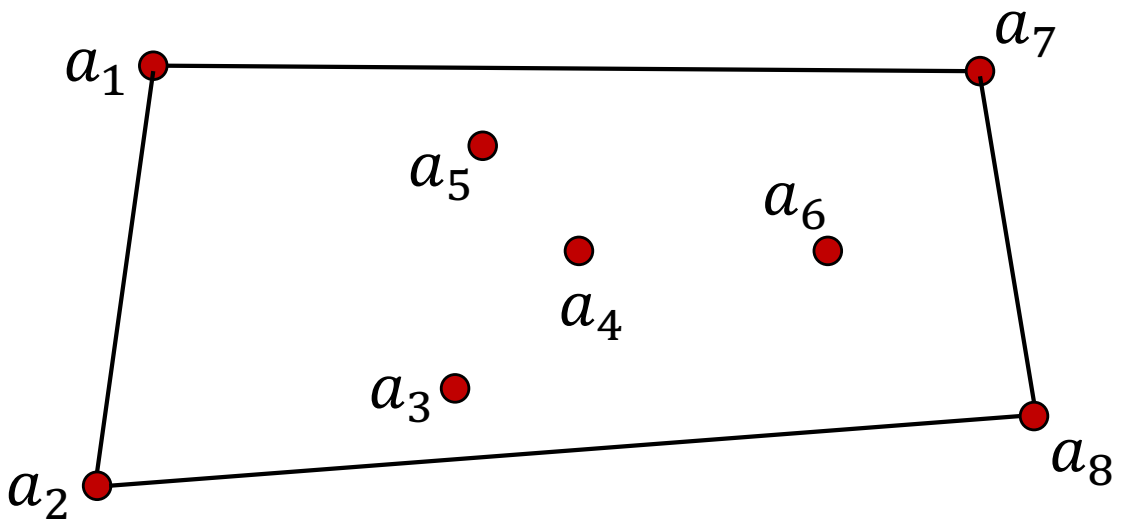
Instructor: Haifeng Xu

Outline

- The Convex Hull Problem
- Fast Algorithm for Convex Hull

The Convex Hull Problem in Plane

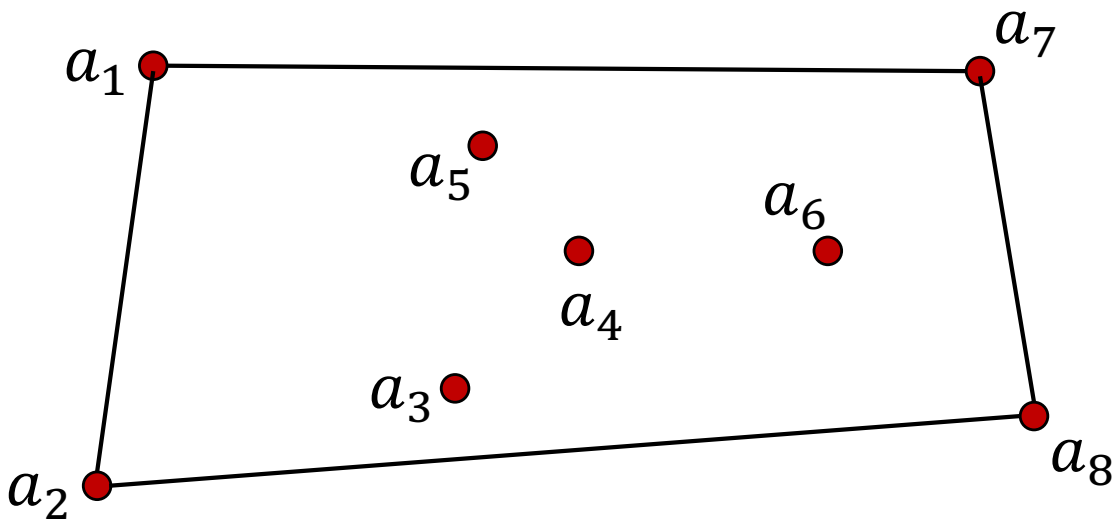
- Input: 2-D points $S = \{(x_1, y_1), \cdots, (x_n, y_n)\}$
- Output: their convex hull



That is, no input points are outside the polygon

The Convex Hull Problem in Plane

- Input: 2-D points $S = \{(x_1, y_1), \cdots, (x_n, y_n)\}$
- Output: their convex hull

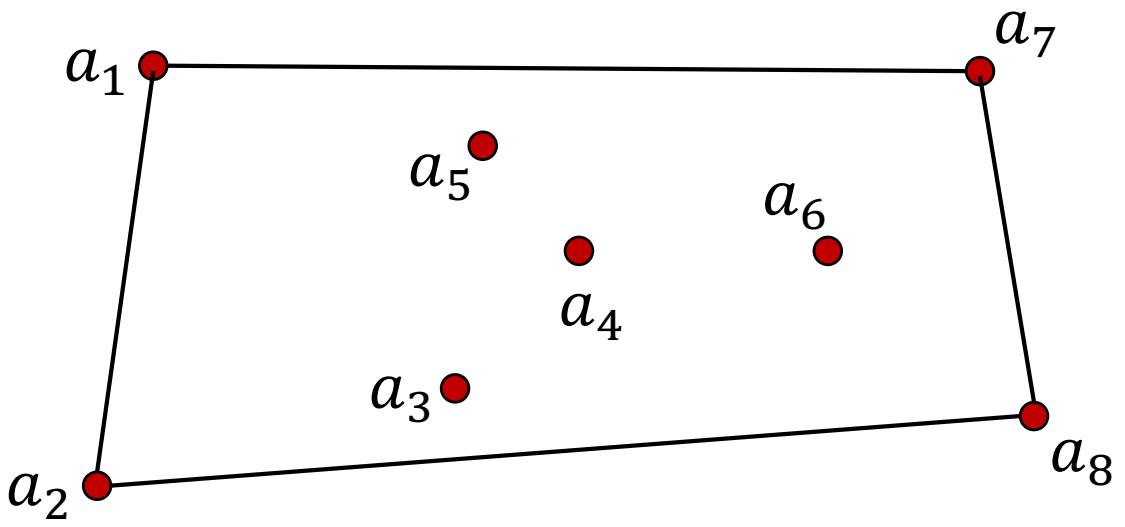


Q: How to represent a convex hull?

- As a set of points? Like $\{a_1, a_2, a_7, a_8\}$? ✗
- Cannot construct the convex hull from this set easily…

The Convex Hull Problem in Plane

- Input: 2-D points $S = \{(x_1, y_1), \cdots, (x_n, y_n)\}$
- Output: their convex hull $\{a_1, a_7, a_8, a_2\}$

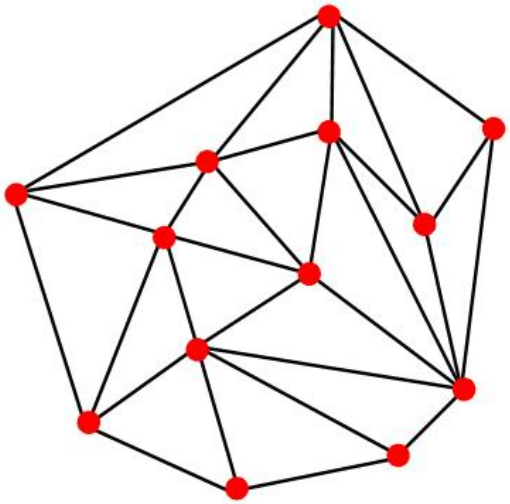


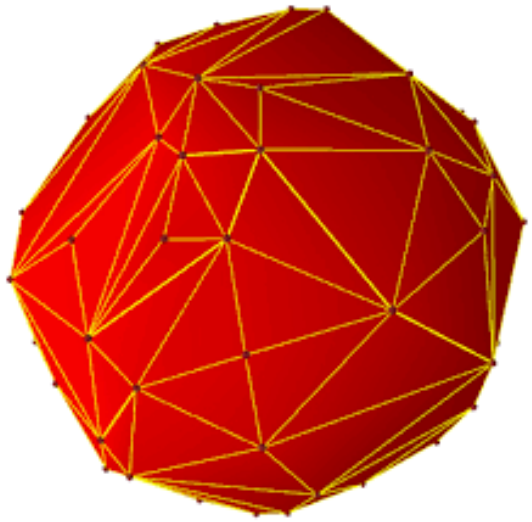
Q: How to represent a convex hull?

- Correct way: points in clockwise or anti-clockwise order

Applications of Convex Hull

- We consider 2-D in lecture, but the problem is well-defined in any dimension
  - The algorithm we will design generalizes to 3-D, but higher dimension is much more intricate





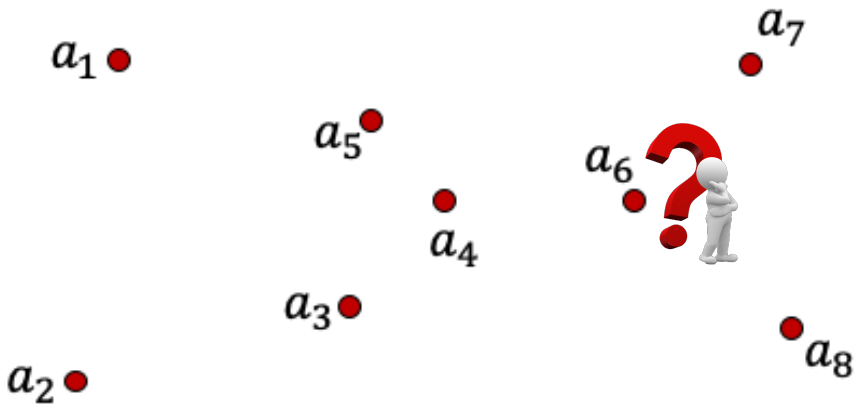
Applications of Convex Hull

- We consider 2-D in lecture, but the problem is well-defined in any dimension
  - The algorithm we will design generalizes to 3-D, but higher dimension is much more intricate
- A fundamental problem in computational geometry
- Convexity is a central concept in modern optimization, algorithm design, machine learning, game theory…

All in all: we need a fast algorithm for it!

First Attempt

- Even naïve enumeration is not easy to think

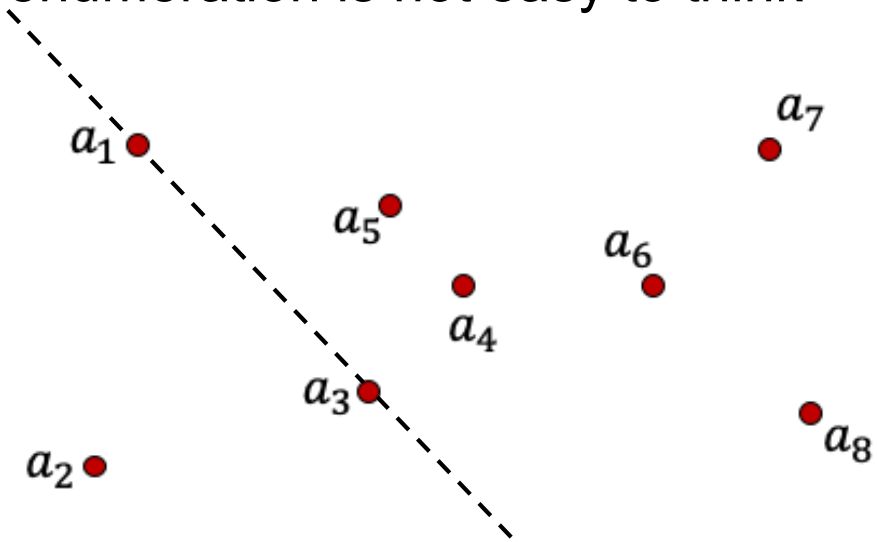


Q: how to decide whether a point is on the convex hull?

- With our eye it's easy, but computers do not have an eye
- Cannot easily do….but can algebraically decide whether two points are both on the hull

First Attempt

- Even naïve enumeration is not easy to think

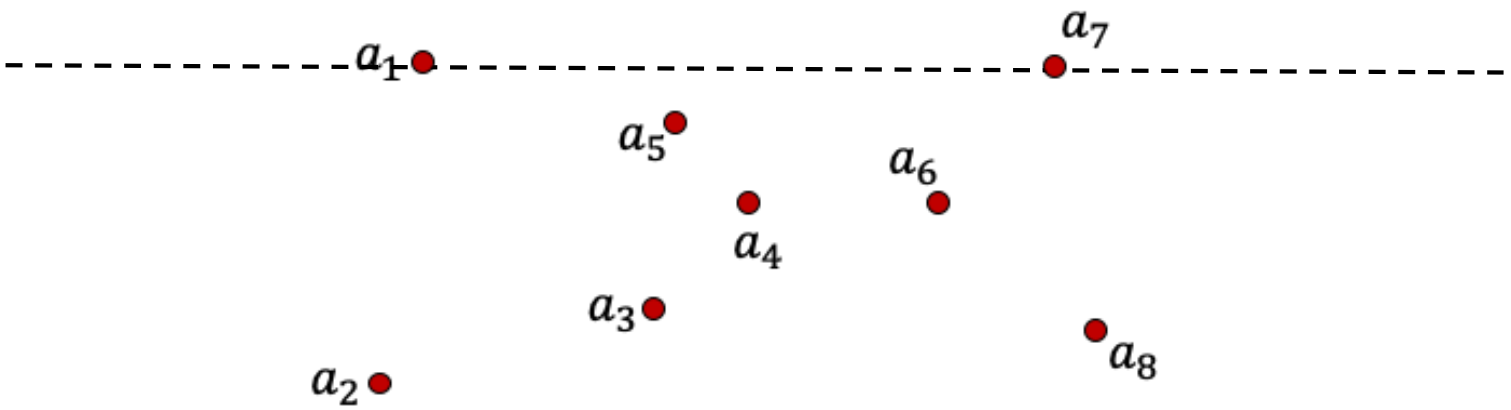


Q: how to decide whether a point is on the convex hull?

- Why cannot both a_1, a_3 be on the hull?

First Attempt

- Even naïve enumeration is not easy to think



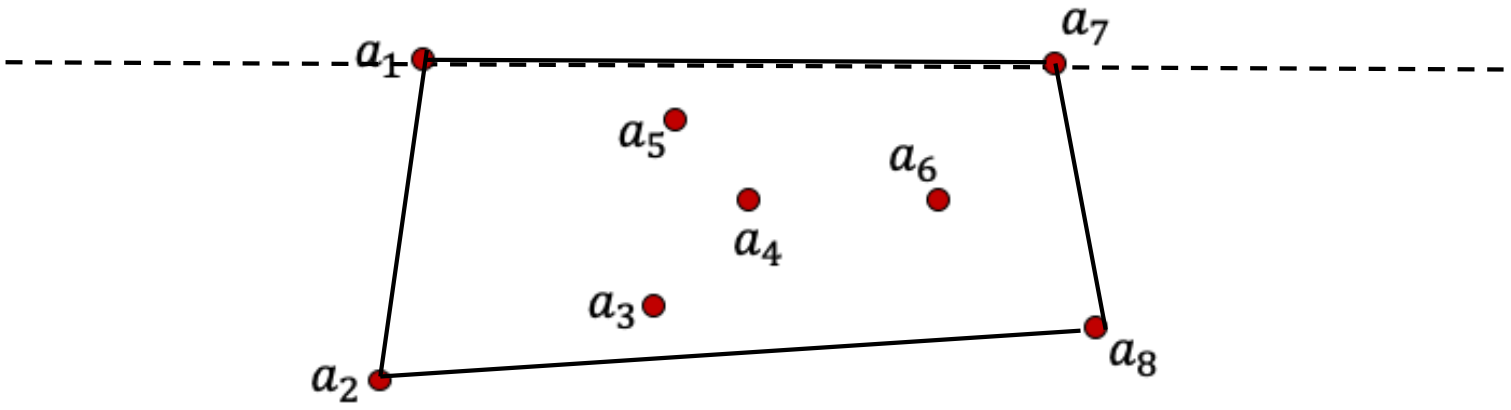
Q: how to decide whether a point is on the convex hull?

- Why cannot both a_1, a_3 on the hull?
- Why a_1, a_7 are both on the hull?

  → All other points are on one side of (a_1, a_7)

First Attempt

- Even naïve enumeration is not easy to think



Facts:

(1) If all other points are on one side of (a_i, a_j), then both a_i, a_j are on the convex hull.

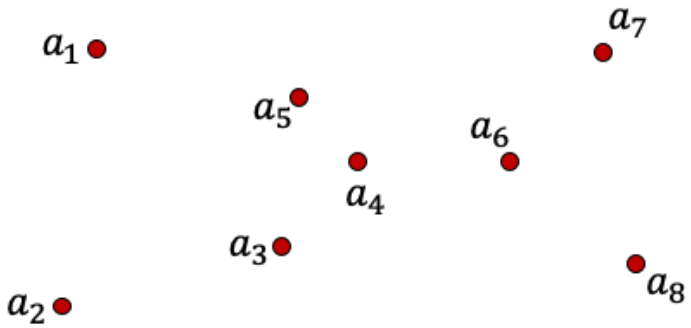
(2) Conversely, if a_i is on the hull, then there must exists a_j such that all other points are on one side of (a_i, a_j).

An Enumeration Algorithm

- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order



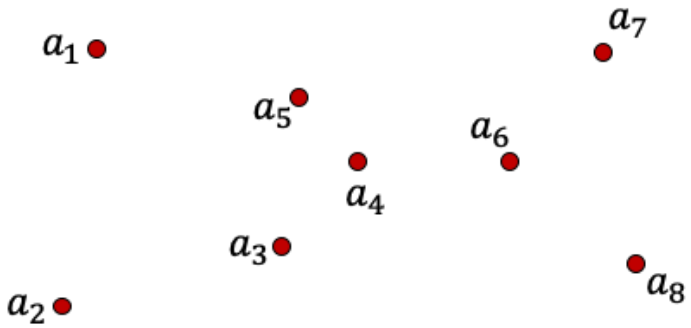
An Enumeration Algorithm

- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order
  - Start with any point, say a_1

a_1



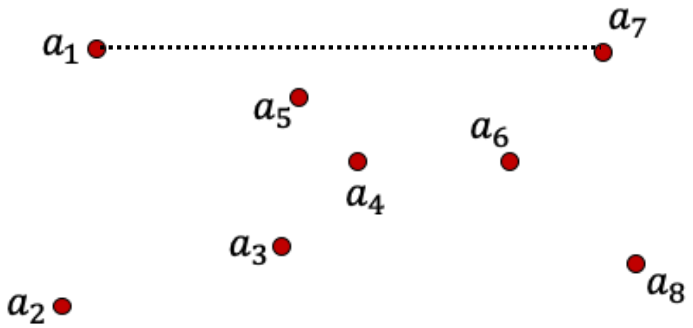
An Enumeration Algorithm

- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order
  - Start with any point, say a_1
  - Add one of a_1's neighbor, say a_7

$a_1 \rightarrow a_7$



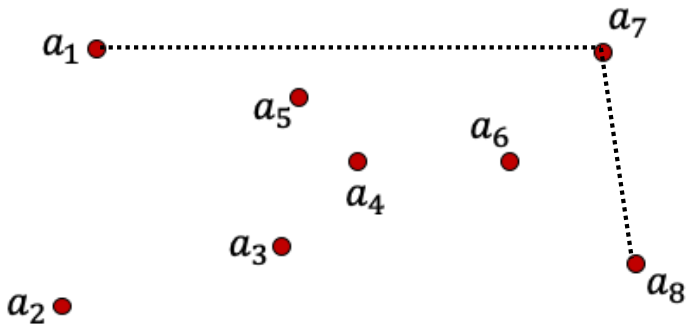
An Enumeration Algorithm

- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order
  - Start with any point, say a_1
  - Add one of a_1's neighbor, say a_7

$a_1 \rightarrow a_7 \rightarrow a_8$



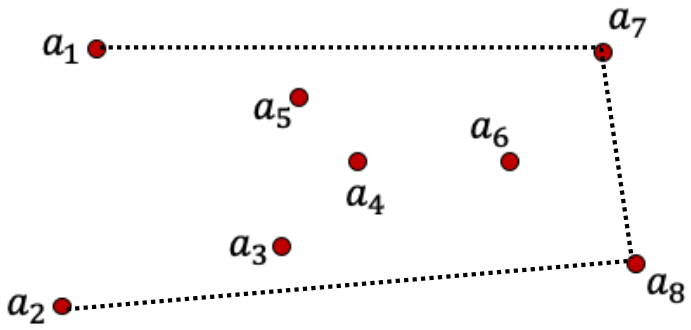
An Enumeration Algorithm

- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order
  - Start with any point, say a_1
  - Add one of a_1's neighbor, say a_7

$a_1 \rightarrow a_7 \rightarrow a_8 \rightarrow a_2$



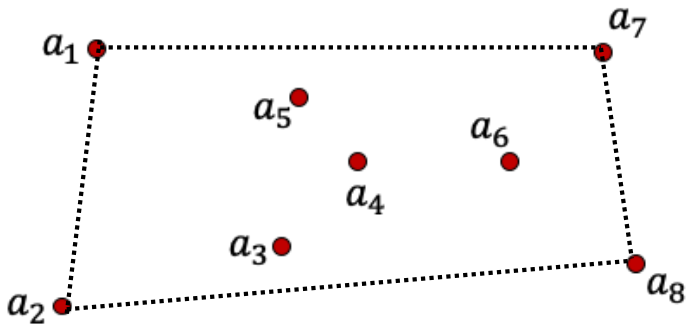
An Enumeration Algorithm

- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order
  - Start with any point, say a_1
  - Add one of a_1's neighbor, say a_7

$a_1 \rightarrow a_7 \rightarrow a_8 \rightarrow a_2 \rightarrow a_1$



An Enumeration Algorithm

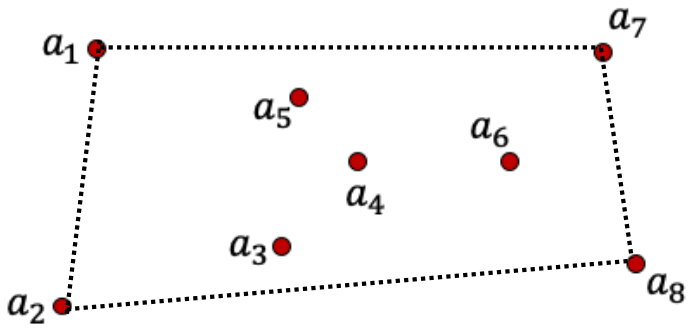
- Step 1: enumerating all point pairs (a_i, a_j) and check whether all other points are on one side of line (a_i, a_j)

  We get (a_1, a_7), (a_1, a_2), (a_2, a_8), (a_7, a_8)

- Step 2: output the convex hull, as a sequence of points in clockwise or anti-clockwise order
  - Start with any point, say a_1
  - Add one of a_1's neighbor, say a_7

$a_1 \rightarrow a_7 \rightarrow a_8 \rightarrow a_2 \rightarrow a_1$

Had we chosen a_2, we will get the anti-clockwise order



Pseudo-Code

NaïveConvexHull(S)

1

2

3

4

5

6

7

8

9

10

11

Pseudo-Code

NaïveConvexHull(S)

1 $n = \text{length}(S)$

2 Candidate pair set $P = \text{empty}()$

3

4

5

6

7

8

9

10

11

Pseudo-Code

NaïveConvexHull(S)

1 $n =$ length(S)

2 Candidate pair set $P =$ empty()

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6

7

8

9

10

11

Pseudo-Code

NaïveConvexHull(S)

1 $n = \text{length}(S)$

2 Candidate pair set $P = \text{empty}()$

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6 Indicator $I_{ij} = 1$

7 **For** $k = 1, \cdots, n$

8 Check whether a_k is on the side of l_{ij} as a_{k-1}.

9

10

11

Pseudo-Code

NaïveConvexHull(S)

1 $n =$ length(S)

2 Candidate pair set $P =$ empty()

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6 Indicator $I_{ij} = 1$

7 **For** $k = 1, \cdots, n$

8 Check whether a_k is on the side of l_{ij} as a_{k-1}.

9 **If** not, $I_{ij} = 0$, exist this for-loop

10

11

Pseudo-Code

NaïveConvexHull(S)

1 $n =$ length(S)

2 Candidate pair set $P =$ empty()

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6 Indicator $I_{ij} = 1$

7 **For** $k = 1, \cdots, n$

8 Check whether a_k is on the side of l_{ij} as a_{k-1}.

9 **If** not, $I_{ij} = 0$, exist this for-loop

10 **If** $I_{ij} = 1$, add (a_i, a_j) to set P

11

Pseudo-Code

NaïveConvexHull(S)

1 $n =$ length(S)

2 Candidate pair set $P =$ empty()

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6 Indicator $I_{ij} = 1$

7 **For** $k = 1, \cdots, n$

8 Check whether a_k is on the side of l_{ij} as a_{k-1}.

9 **If** not, $I_{ij} = 0$, exist this for-loop

10 **If** $I_{ij} = 1$, add (a_i, a_j) to set P

11 Construct a list from candidate pairs in P

Running Time Analysis

NaïveConvexHull(S)

1 $n =$ length(S)

2 Candidate pair set $P =$ empty()

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6 Indicator $I_{ij} = 1$

7 **For** $k = 1, \cdots, n$

8 Check whether a_k is on the side of l_{ij} as a_{k-1}.

9 **If** not, $I_{ij} = 0$, exist this for-loop

10 **If** $I_{ij} = 1$, add (a_i, a_j) to set P

11 Construct a list from candidate pairs in P

Constant time

Constant time

$O(n^3)$ time

Running Time Analysis

NaïveConvexHull(S)

1 $n =$ length(S)

2 Candidate pair set $P =$ empty()

3 **For** $i = 1, \cdots, n-1$

4 **For** $j = i+1, \cdots, n$

5 Construct line l_{ij} crossing a_i, a_j

6 Indicator $I_{ij} = 1$

7 **For** $k = 1, \cdots, n$

8 Check whether a_k is on the side of l_{ij} as a_{k-1}.

9

10 **If** $I_{ij} = 1$, add (a_i, a_j) to set P

11 Construct a list from candidate pairs in P

$O(n^3)$ time

Constant time

Constant time

Can we do better?

Outline

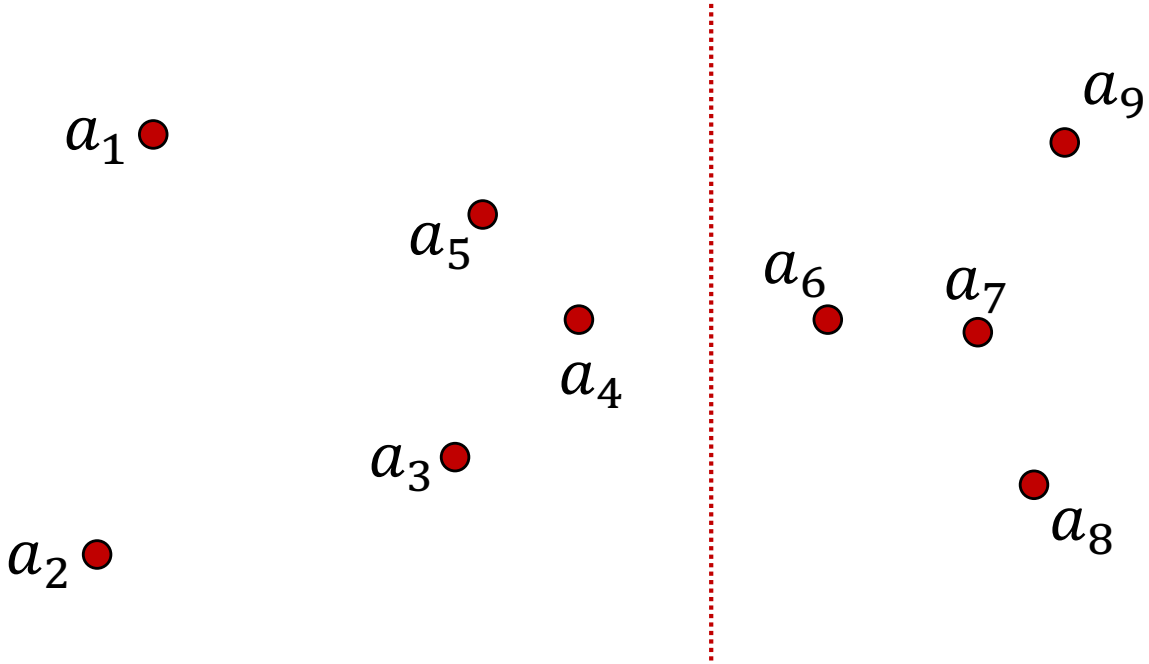
- Convex Hull Problem
- Fast Algorithm for Convex Hull

We want to do Divide and Conquer

Why?

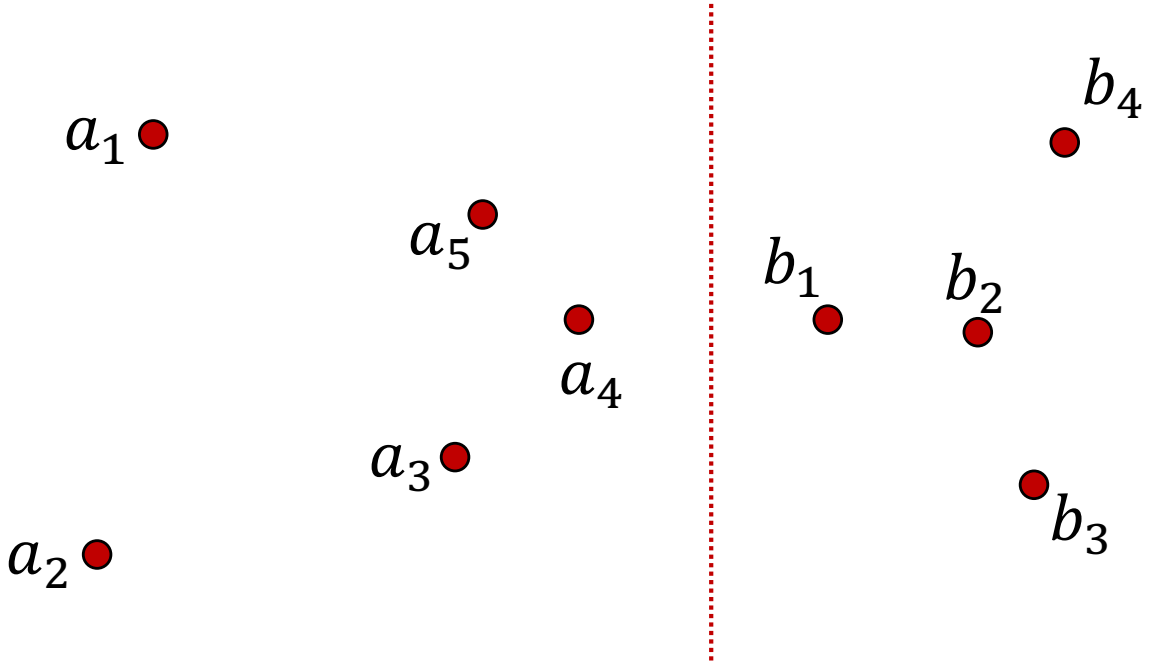
- Because people have tried different ways, and divide and conquer works
- For a new problem you may face, typically need to try different techniques in order to figure out the correct one

How to Divide?



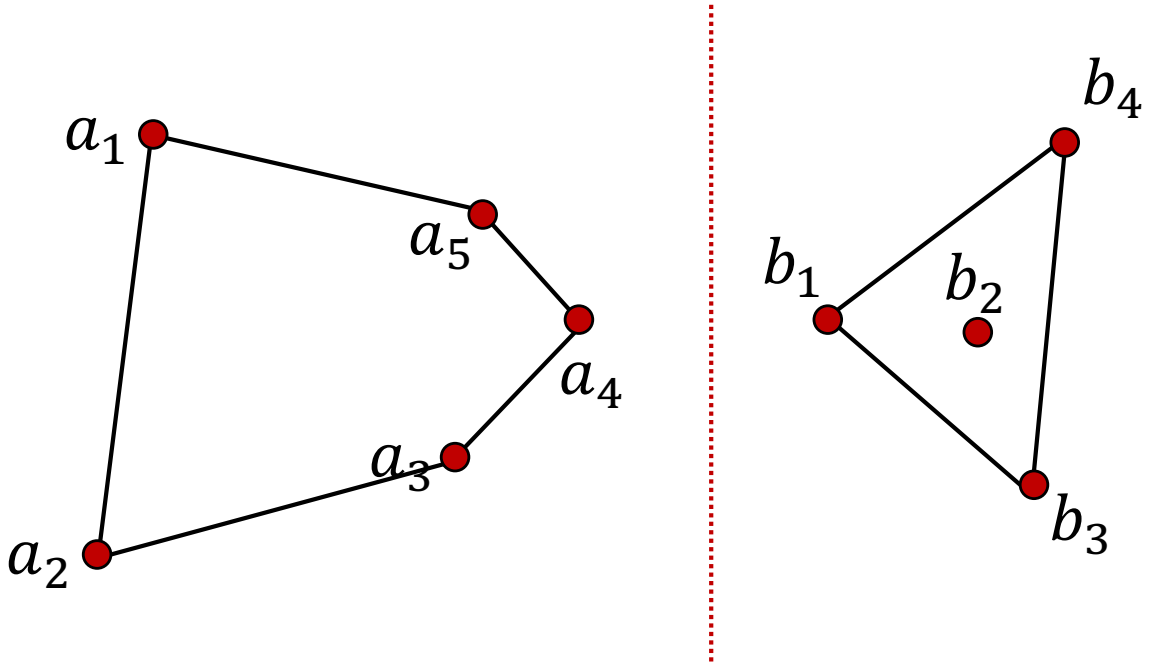
- Sort the points according to x-axis, and divide them at the middle

How to Divide?



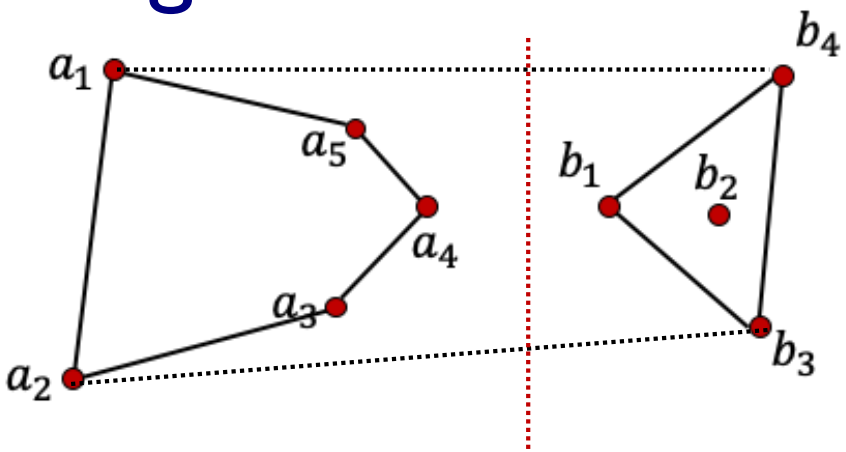
- Sort the points according to x-axis, and divide them at the middle

How to Divide?



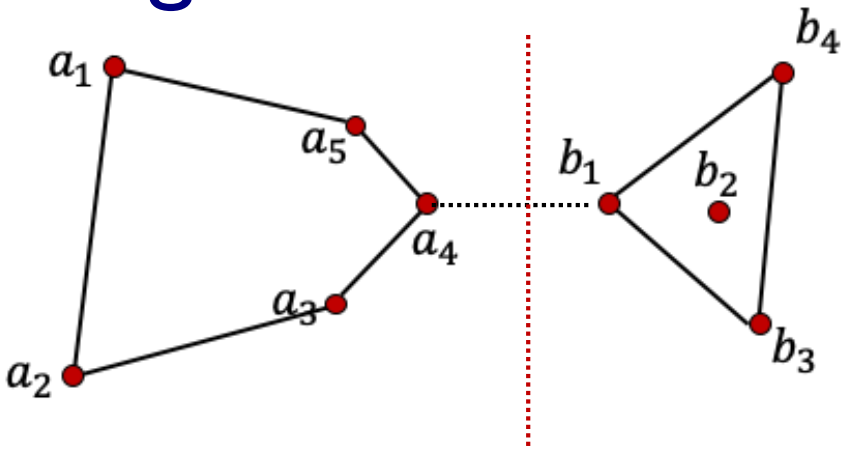
- Sort the points according to x-axis, and divide them at the middle
- Solve sub-problems

How to Merge?



Goal: want to algebraically find upper/lower tangent of the hull

How to Merge?



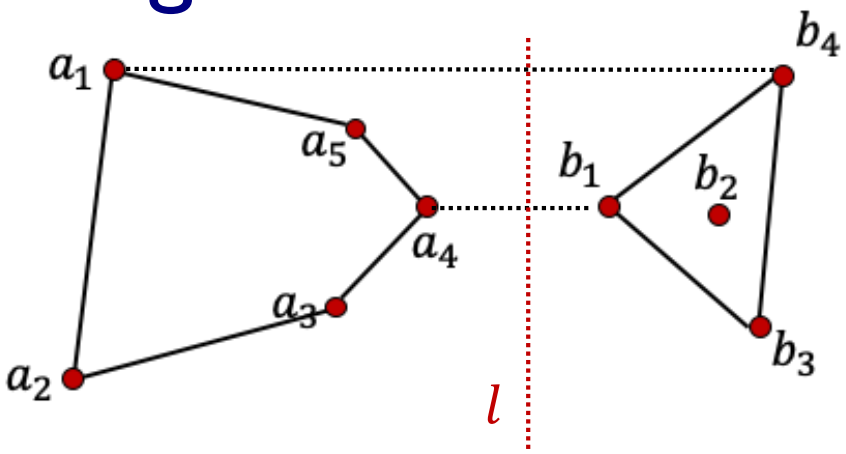
Goal: want to algebraically find upper/lower tangent of the hull

- Why not line (a_4, b_1)?
  - Points are on different sides of line (a_4, b_1)!

True, but this takes $O(n)$ time to check for each pair (a_i, b_j)

Need a faster approach

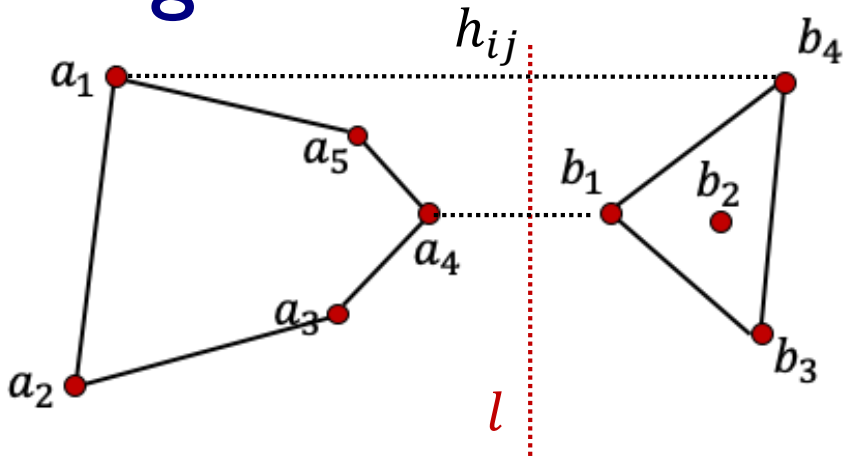
How to Merge?



Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 1: (a_i, b_j) is the upper tangent only if its intersection with cutting line l is the highest possible

How to Merge?



Goal: want to algebraically find upper/lower tangent of the hull

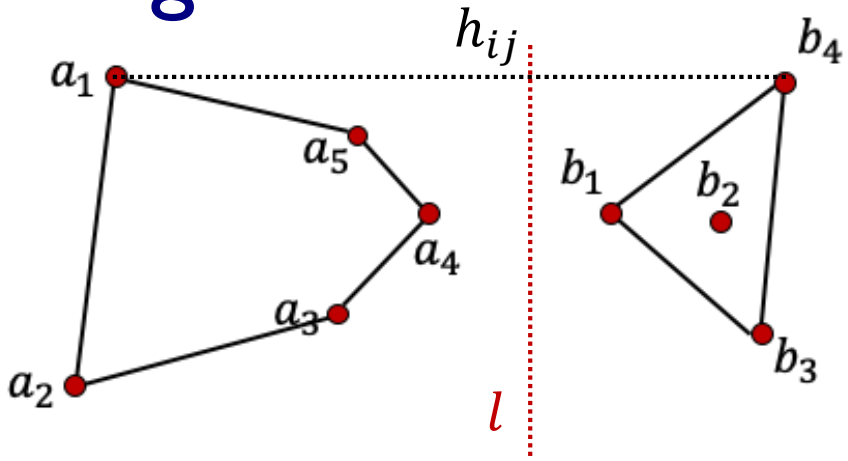
- Key idea 1: (a_i, b_j) is the upper tangent only if its intersection with cutting line l is the highest possible

Only need to compute h_{ij} for each (a_i, b_j), and then compare them

Enumeration of pairs lead to $O(n^2)$ time for Merge

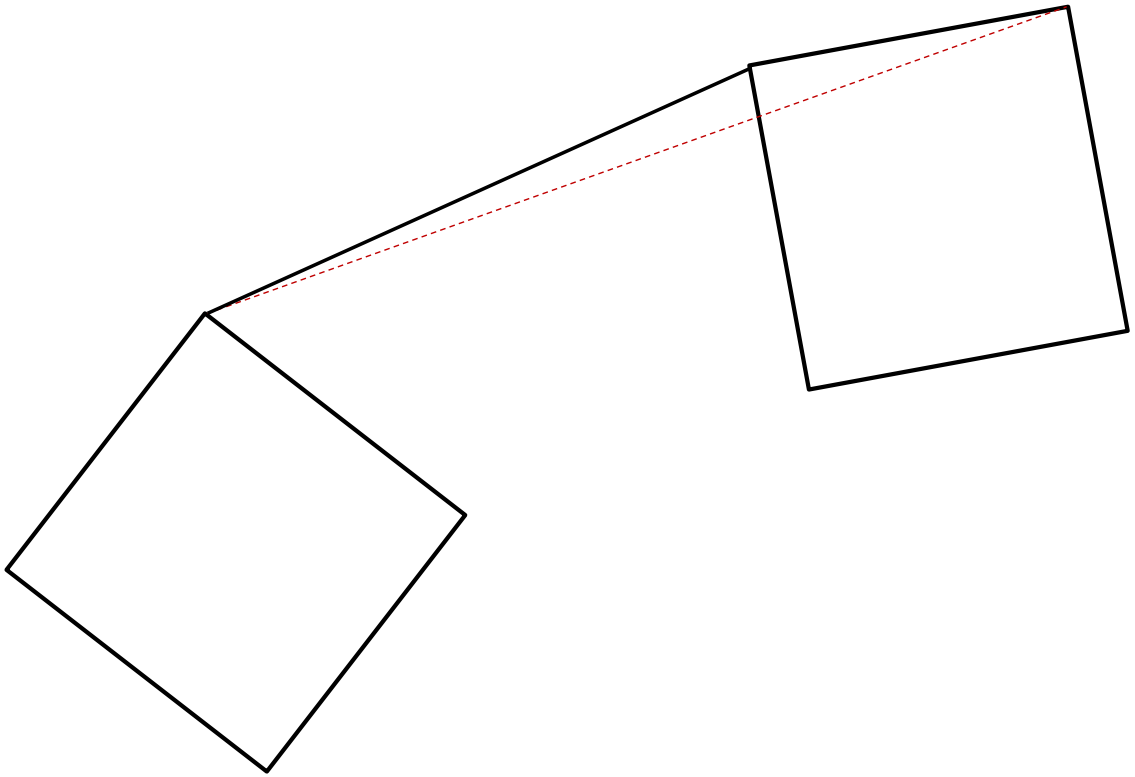
Q: Can we be faster?

How to Merge?



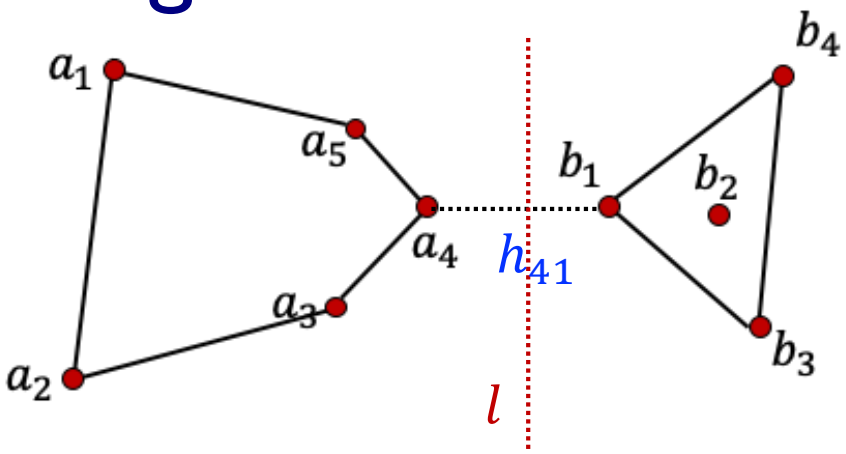
Goal: want to algebraically find upper/lower tangent of the hull

- What about picking the highest point of each subproblem?
  - Seems to work here?
  - Unfortunately, no always….



Connecting the two highest points do not work here!

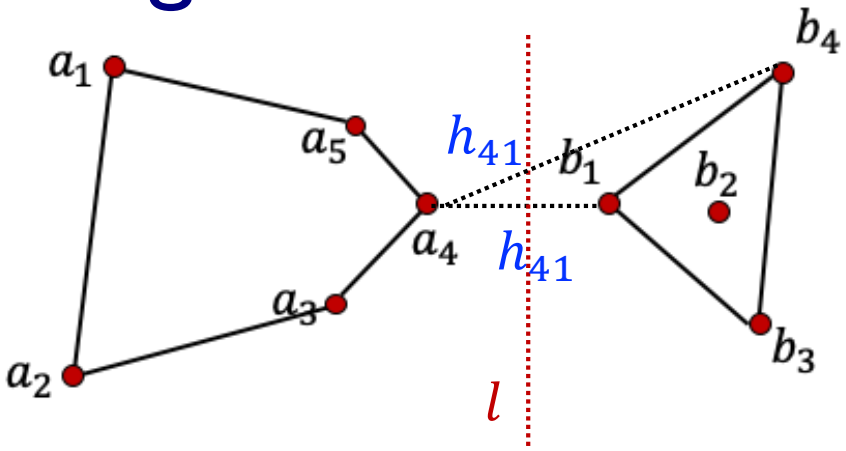
How to Merge?



Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

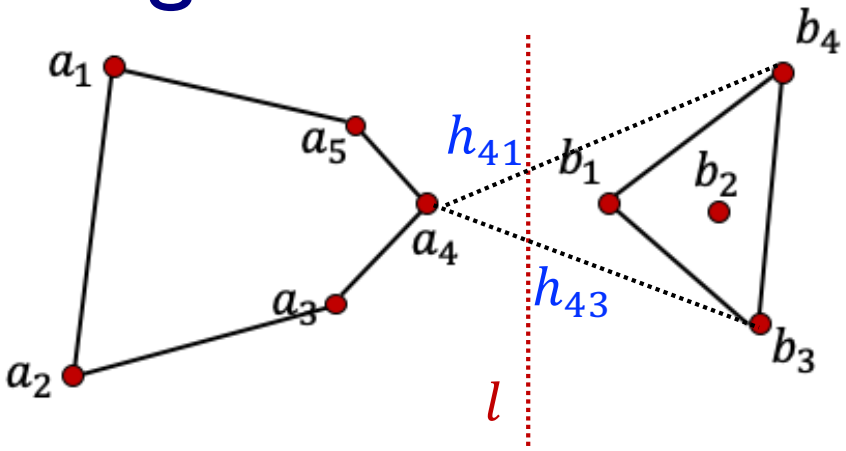


Good move 👍

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

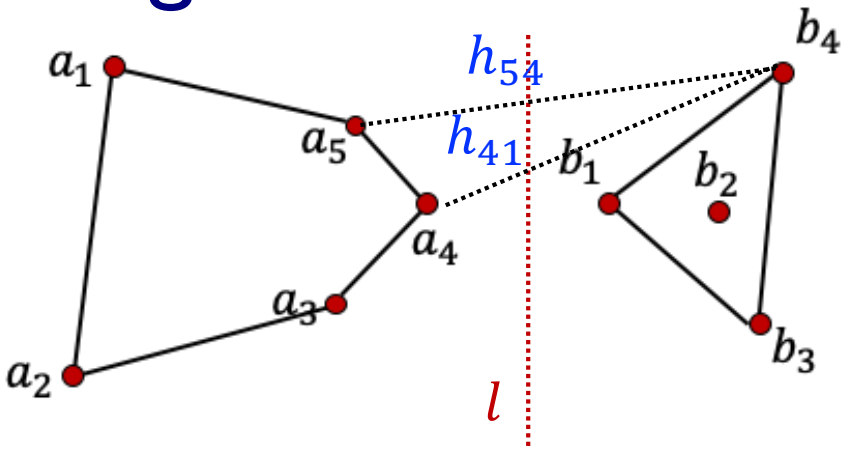


Bad remove; go back

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

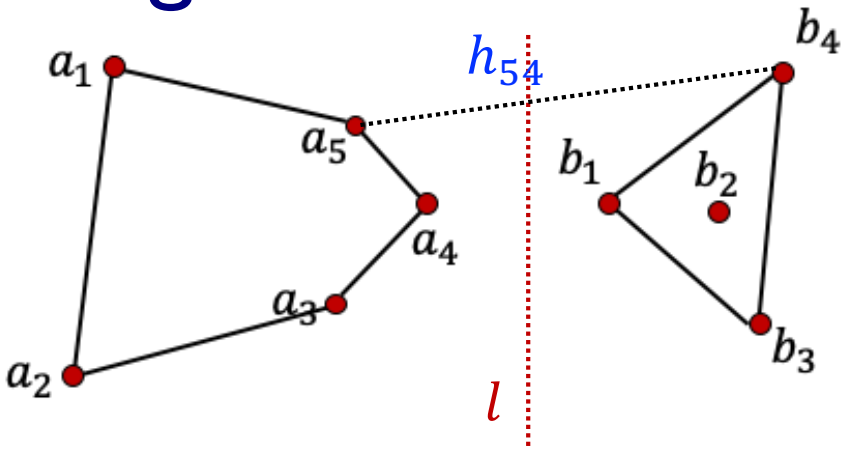


Good move 👍

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

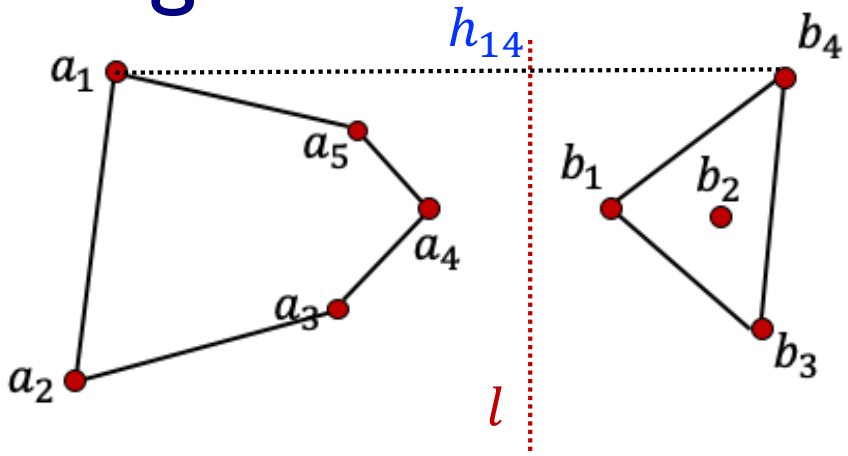


Good move 👍

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

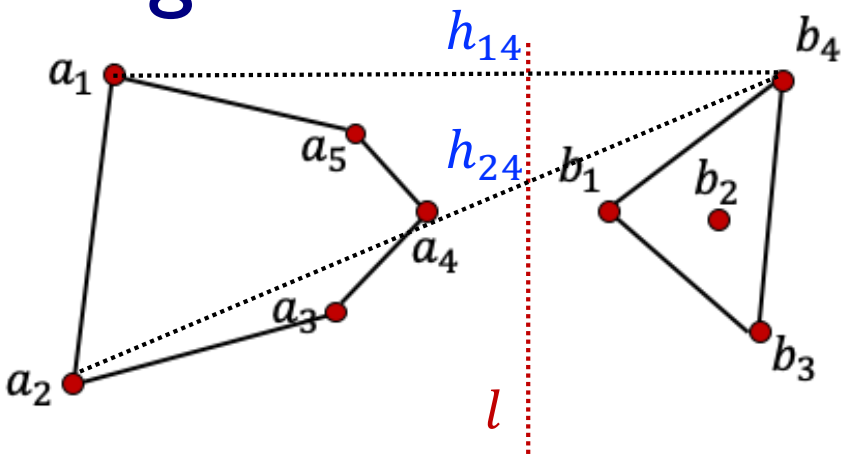


Good move 👍

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

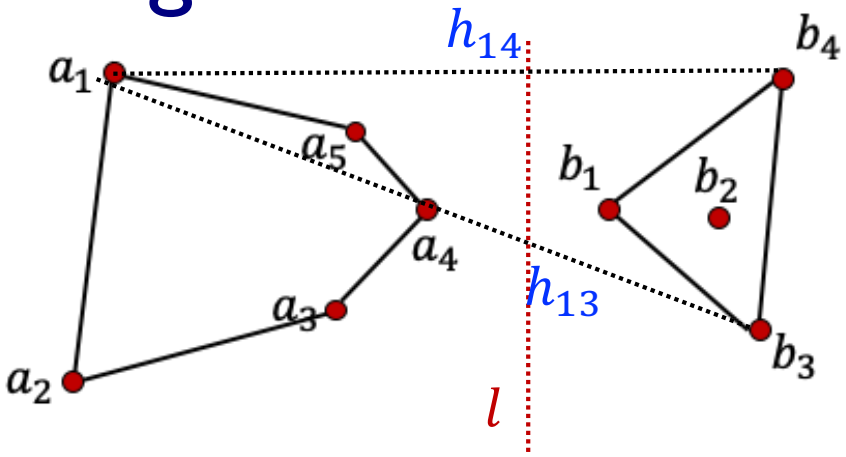


Bad move; go back

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

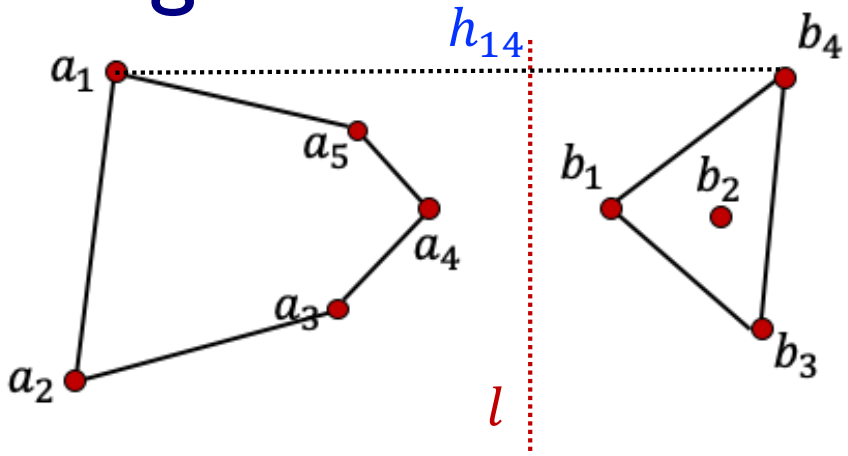


Bad move; go back

Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

How to Merge?

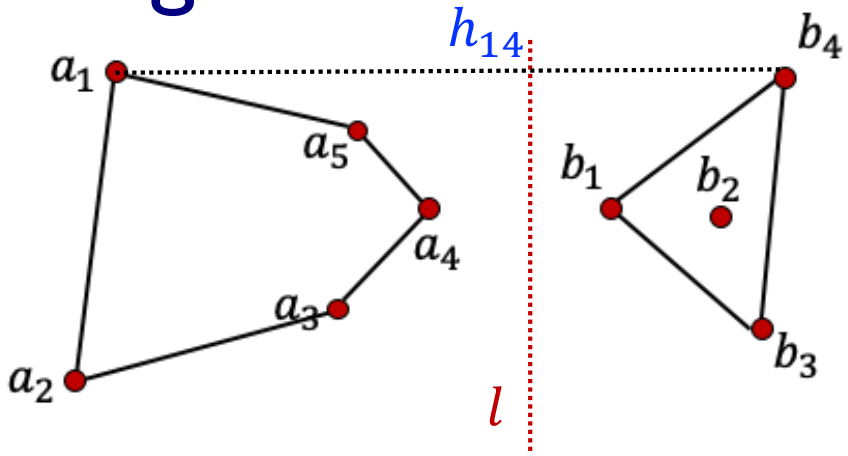


Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

We are sure now that (a_1, b_4) is the upper tangent

How to Merge?

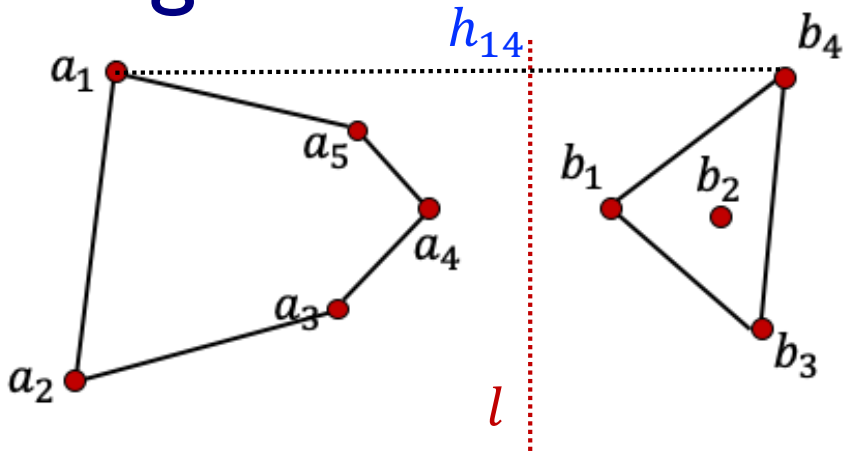


Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

Fact. Due to convexity, if Step 2 does not lead to higher intersection with l, then (a_i, b_j) must be the upper tangent line.

How to Merge?

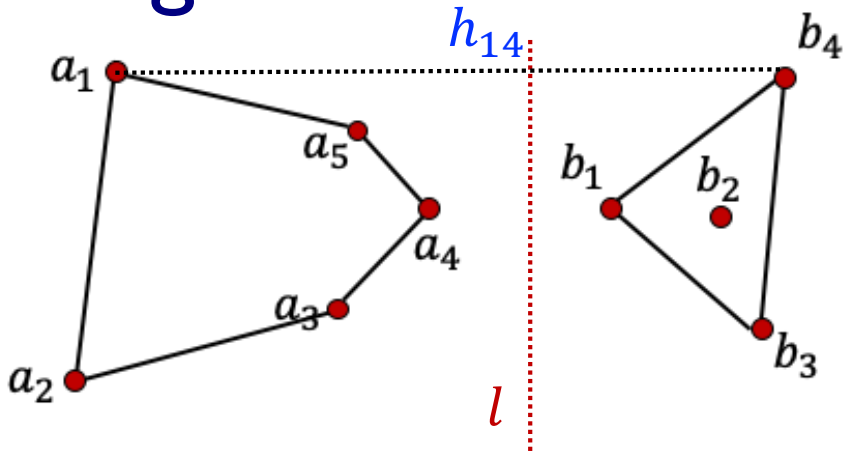


Goal: want to algebraically find upper/lower tangent of the hull

- Key idea 2: the TwoFinger algorithm for Merge
  1. Start with any (a_i, b_j)
  2. Try next $b_{j'}$ in clockwise order or next $a_{i'}$ in anti-clockwise order, and check whether $h_{ij'}$ or $h_{i'j}$ goes higher

TwoFinger algorithm takes $O(n)$ time!

How to Merge?



Merge: Now we find tangent (a_1, b_4) and (a_2, b_3), how to merge?

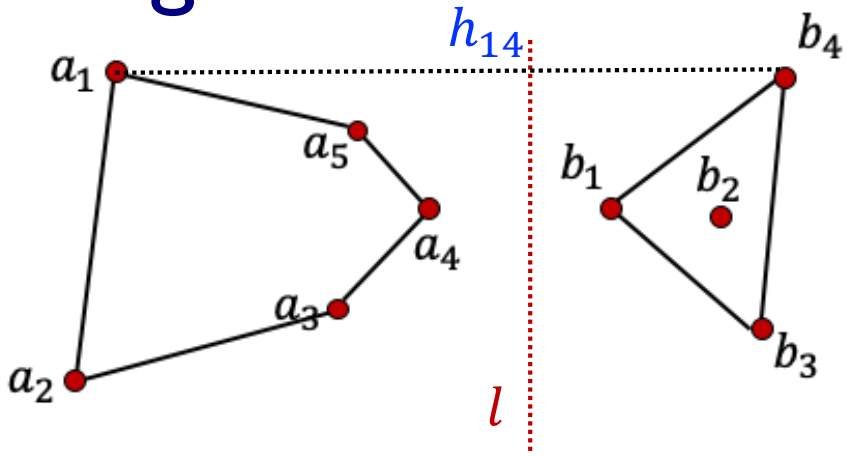
A: $a_1 \rightarrow a_5 \rightarrow a_4 \rightarrow a_3 \rightarrow a_2 \rightarrow a_1$

B: $b_1 \rightarrow b_4 \rightarrow b_3 \rightarrow b_1$

Tangent $(a_1, b_4), (a_2, b_3)$

Merge:

How to Merge?



Merge: Now we find tangent (a_1, b_4) and (a_2, b_3), how to merge?

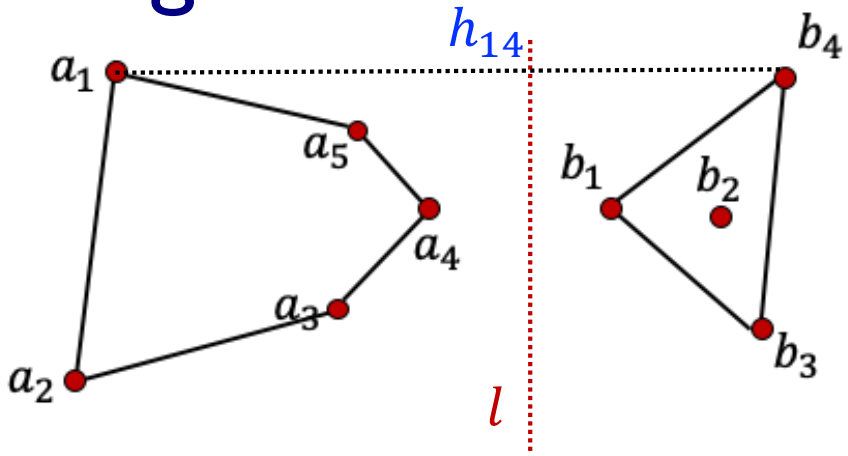
A: $a_1 \to a_5 \to a_4 \to a_3 \to a_2 \to a_1$

B: $b_1 \to b_4 \to b_3 \to b_1$

Tangent $(a_1, b_4), (a_2, b_3)$

Merge $a_1 \to b_4$

How to Merge?



Merge: Now we find tangent (a_1, b_4) and (a_2, b_3), how to merge?

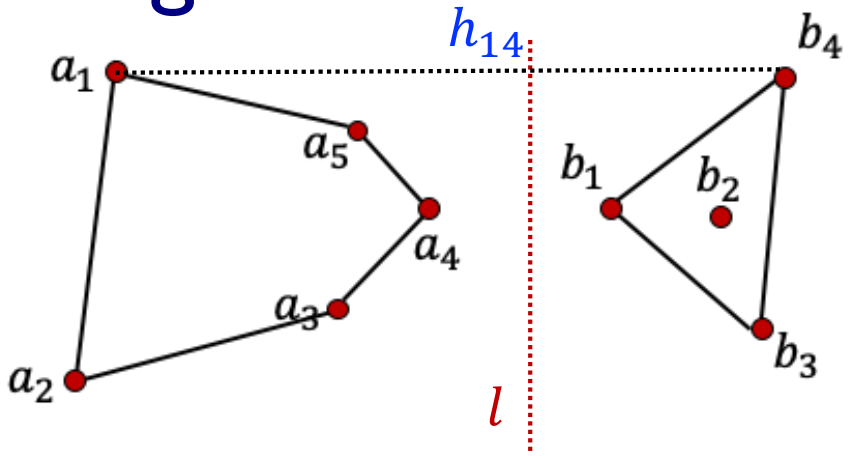
A: $a_1 \to a_5 \to a_4 \to a_3 \to a_2 \to a_1$

B: $b_1 \to b_4 \to b_3 \to b_1$

Tangent $(a_1, b_4), (a_2, b_3)$

Merge $a_1 \to b_4$

How to Merge?



Merge: Now we find tangent (a_1, b_4) and (a_2, b_3), how to merge?

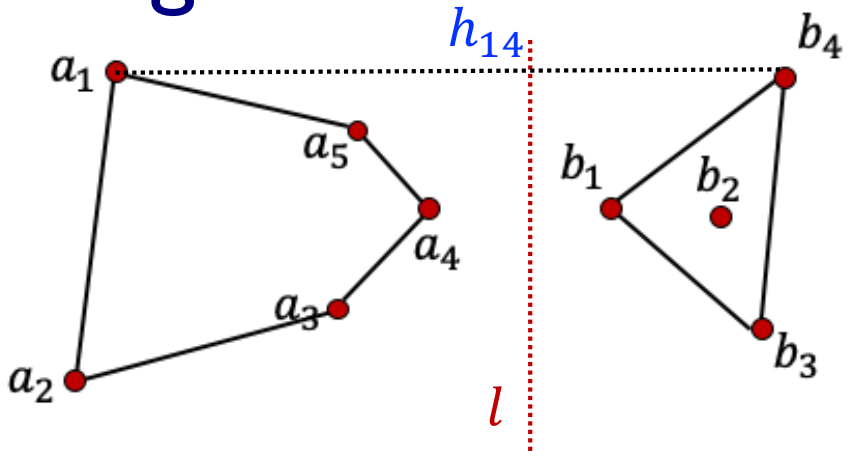
A: $a_1 \to a_5 \to a_4 \to a_3 \to a_2 \to a_1$

B: $b_1 \to b_4 \to b_3 \to b_1$

Tangent $(a_1, b_4), (a_2, b_3)$

Merge $a_1 \to b_4 \to b_3$

How to Merge?



Merge: Now we find tangent (a_1, b_4) and (a_2, b_3), how to merge?

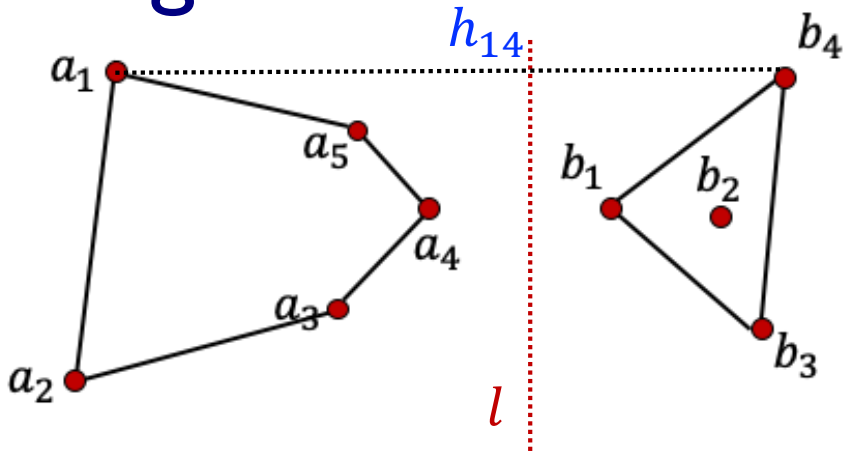
A: $a_1 \rightarrow a_5 \rightarrow a_4 \rightarrow a_3 \rightarrow a_2 \rightarrow a_1$

B: $b_1 \rightarrow b_4 \rightarrow b_3 \rightarrow b_1$

Tangent $(a_1, b_4), (a_2, b_3)$

Merge $a_1 \rightarrow b_4 \rightarrow b_3 \rightarrow a_2$

How to Merge?



Merge: Now we find tangent (a_1, b_4) and (a_2, b_3), how to merge?

A: $a_1 \to a_5 \to a_4 \to a_3 \to a_2 \to a_1$

B: $b_1 \to b_4 \to b_3 \to b_1$

Tangent $(a_1, b_4), (a_2, b_3)$

Merge $a_1 \to b_4 \to b_3 \to a_2 \to a_1$

Takes $O(n)$ time a well

Pseudo-Code

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1

2

3

4

5

6

7

Pseudo-Code

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1 $n = \text{length}(S)$

2 Divide points at the middle into set A, B

3

4

5

6

7

Pseudo-Code

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1 n = length(S)

2 Divide points at the middle into set A, B

3 L_A = ConvexHull(A)

4 L_B =ConvexHull(B)

5

6

7

Pseudo-Code

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1 $n = \text{length}(S)$

2 Divide points at the middle into set A, B

3 $L_A = \text{ConvexHull}(A)$

4 $L_B = \text{ConvexHull}(B)$

5 Find upper/lower tangent $(a_i, b_j), (a_{i'}, b_{j'})$ from L_A, L_B

6

7

Pseudo-Code

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1 $n = \text{length}(S)$

2 Divide points at the middle into set A, B

3 $L_A = \text{ConvexHull}(A)$

4 $L_B = \text{ConvexHull}(B)$

5 Find upper/lower tangent $(a_i, b_j), (a_{i'}, b_{j'})$ from L_A, L_B

6 Merge L_A, L_B using $(a_i, b_j), (a_{i'}, b_{j'})$

7

Pseudo-Code

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1 $n = \text{length}(S)$

2 Divide points at the middle into set A, B

3 $L_A = \text{ConvexHull}(A)$

4 $L_B = \text{ConvexHull}(B)$

5 Find upper/lower tangent $(a_i, b_j), (a_{i'}, b_{j'})$ from L_A, L_B

6 Merge L_A, L_B using $(a_i, b_j), (a_{i'}, b_{j'})$

7 Return the merged solution

Running Time Analysis

$O(n \log n)$

Preprocessing: Sort points according to x-axis

ConvexHull(S)

1 $n =$ length(S)

2 Divide points at the middle into set A, B

3 $L_A =$ ConvexHull(A) → $T\left(\frac{n}{2}\right)$

4 $L_B =$ConvexHull(B)

$O(n)$

5 Find upper/lower tangent $(a_i, b_j), (a_{i'}, b_{j'})$ from L_A, L_B

6 Merge L_A, L_B using $(a_i, b_j), (a_{i'}, b_{j'})$ → $O(n)$

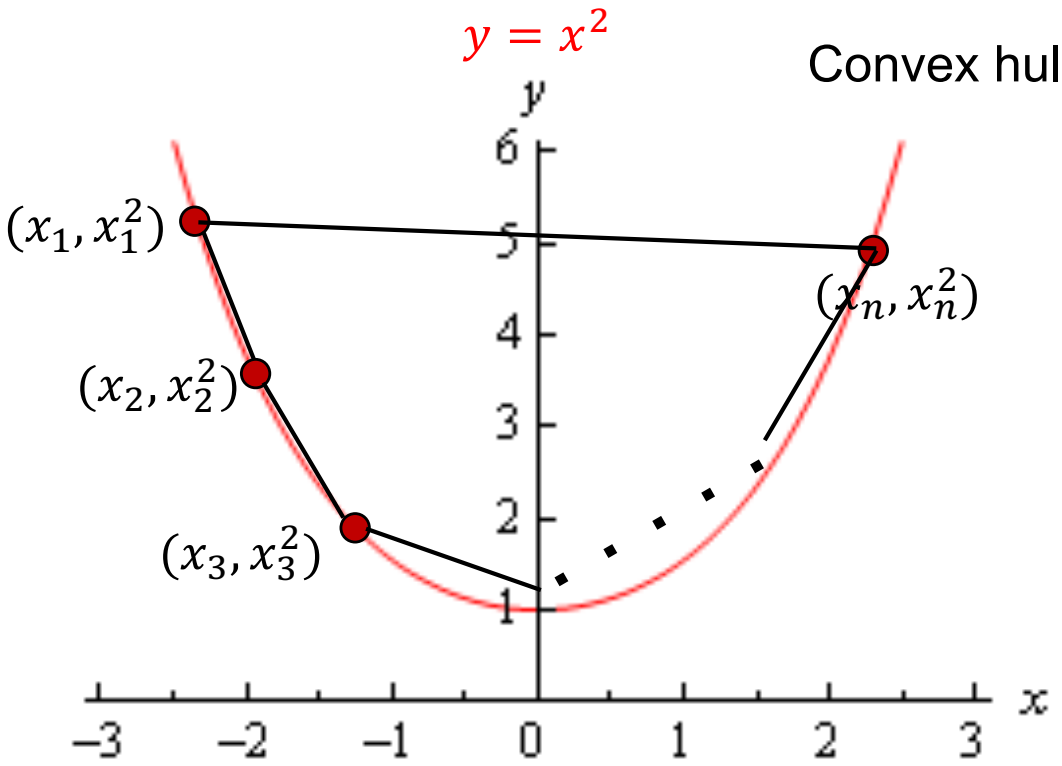
7 Return the merged solution

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \longrightarrow T(n) = \Theta(n \log n)$$

Is $n \log n$ the Best?

Thm. No algorithms can find the convex hull in 2-D faster than $\Theta(n \log n)$.

- Proof: carefully construct a convex hull instance such that its answer directly implies sorting

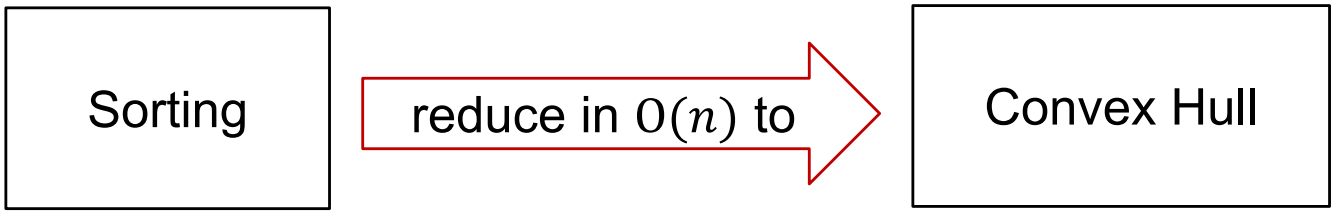


Convex hull implies a sorting of $x_1, x_2, \cdots, x_n$

Is $n \log n$ the Best?

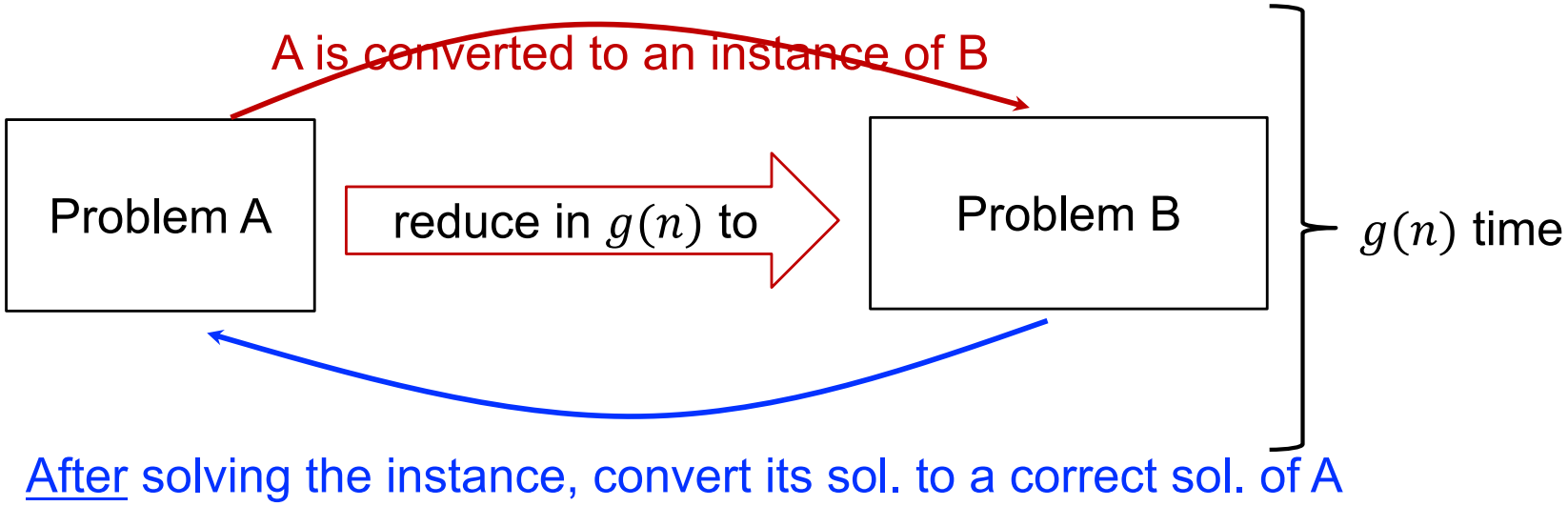
Thm. No algorithms can find the convex hull in 2-D faster than $\Theta(n \log n)$.

- Proof: carefully construct a convex hull instance such that its answer directly implies sorting
- That is, any algorithm solving the convex hull problem can also solve the sorting problem, with addition $O(n)$ work
  - This is called reduction



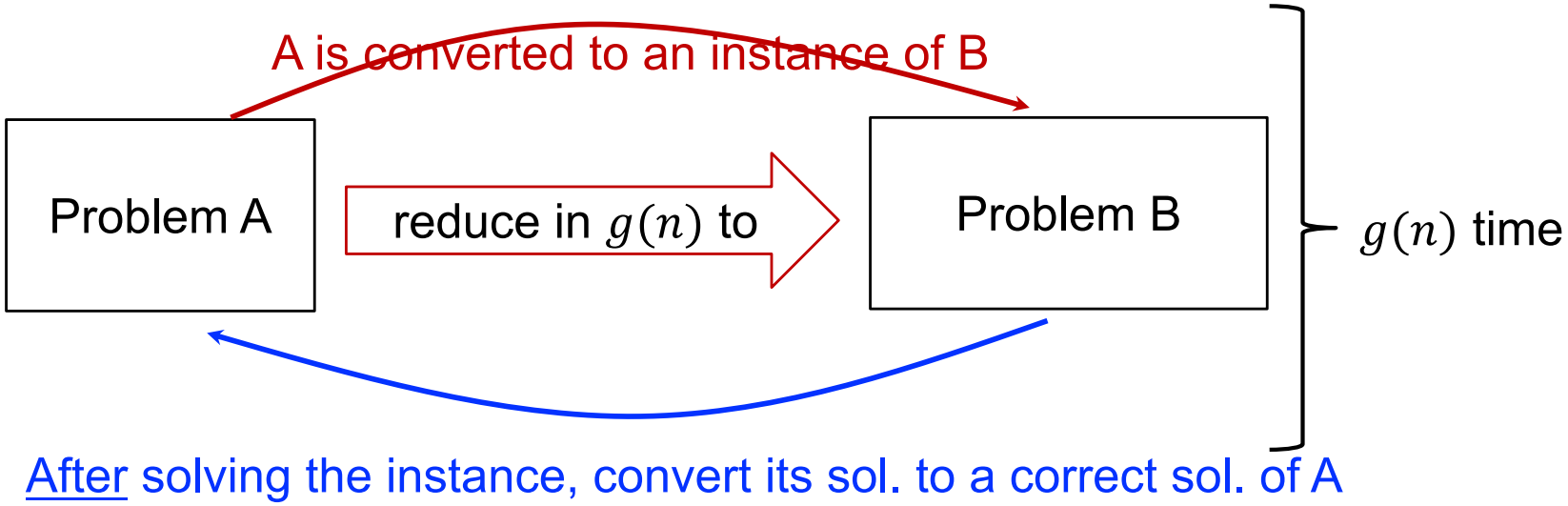
That is, sorting can be converted to a convex hull instance, whose solution can be converted back to a solution of sorting, in $O(n)$ time

Standard two-step procedure of reductions



Note: $g(n)$ does not include time of solving the instance of B

Standard two-step procedure of reductions



If $g(n) = O(T_B(n))$ where $T_B(n)$ is the fastest time for solving Problem B, we say Problem B is no easier than Problem A

- Because Problem B cannot be solved in a faster way than solving A
- So, convex hull is no easier than sorting, and requires at least $\Theta(n \log n)$ time as well

Summary

- Convex Hull is an important problem
- For points in 2-D, there is a $\Theta(n \log n)$ algorithm
- This is the best possible algorithm
- This problem is well-defined for d-dimension as well
  - Best possible algorithm is $\Theta(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$
  - For $d = 2,3$, the best is $\Theta(n \log n)$
  - For $d = 4,5$, the best is $\Theta(n^2)$

See, e.g., the paper
An Optimal Convex Hull Algorithm in Any Fixed Dimension

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu