Divide and Conquer

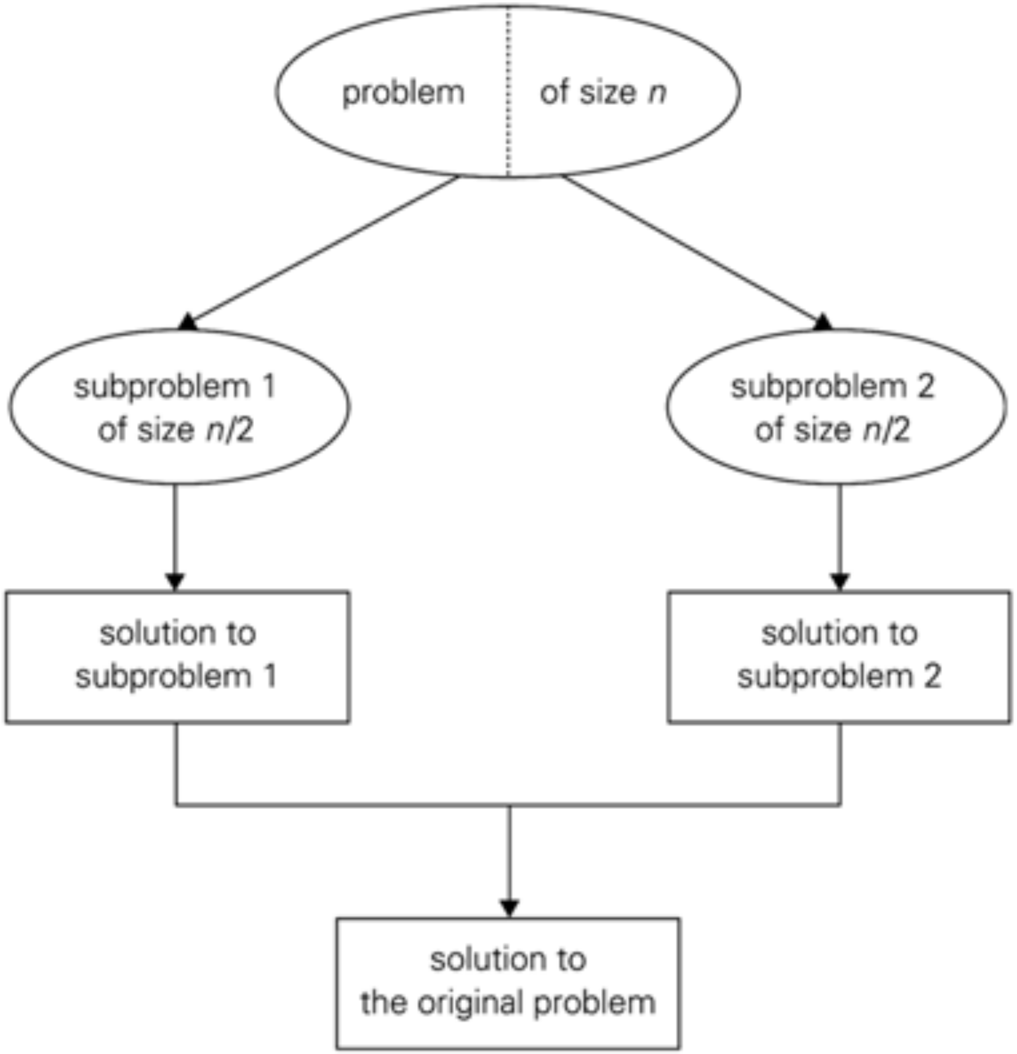
Chapter 4.1 – 4.2

Mei-Chen Yeh

General plan

1. **Divide** into several smaller instances of the same problem
2. **Solve** the smaller instances
3. **Combine** the solutions

Example: a typical case



Example 1: binary search

- Find an element in a *sorted* array:

1. **Divide**: Check middle element.
2. **Conquer**: Recursively search 1 sub-array.
3. **Combine**: Trivial.

Example: Find 9

3	5	7	8	9	12	15

Recurrence for binary search

$$T(n) = 1T(n/2) + \Theta(1)$$

 # sub-problems

sub-problem size

work dividing and combining

$$T(n) = \Theta(\lg n)$$

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow case\ 2$$

<u>side information</u>:
$\lg n = \log_e n$

Example 2: powering a number

- Compute a^n, where $n \in \mathbb{N}$.
- Naïve algorithm: $\Theta(n)$
- Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) = \Theta(\lg n)$$

Example 3: matrix multiplication

- Input: A = $[a_{ij}]$, B = $[b_{ij}]$
- Output: C = $[c_{ij}]$ = AB

$i, j = 1, 2, \ldots, n$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Naïve algorithm

for $i \leftarrow 1$ **to** n **do**

  for $j \leftarrow 1$ **to** n **do**

    $c_{ij} = 0$

    for $k \leftarrow 1$ **to** n **do**

      $c_{ij} = c_{ij} + a_{ik} b_{kj}$

$$T(n) = \Theta(n^3)$$

Divide-and-conquer algorithm

- *nxn* matrix = 2x2 matrix of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8 mults of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices

4 adds of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices

n是邊長

A divide-and-conquer algorithm

T(n)

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
1. $n = A.rows$
2. Let C be a new n x n matrix
3. **if** $n == 1$
4. $c_{11} = a_{11} \cdot b_{11}$
5. **else** partition A, B, and C
6. $r =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(a, e) T(n/2)
   \+ SQUARE-MATRIX-MULTIPLY-RECURSIVE(b, g) T(n/2) $n^2/4$
7. $s =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(a, f) T(n/2)
   \+ SQUARE-MATRIX-MULTIPLY-RECURSIVE(b, h) T(n/2) $n^2/4$
8. $t =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(c, e) T(n/2)
   \+ SQUARE-MATRIX-MULTIPLY-RECURSIVE(d, g) T(n/2) $n^2/4$
9. $u =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(c, f) T(n/2)
   \+ SQUARE-MATRIX-MULTIPLY-RECURSIVE(d, h) T(n/2) $n^2/4$
10. **return** C

$$\mathbf{T(n) = 8T(n/2) + \Theta(n^2)}$$

Analysis

$$T(n) = 8T(n/2) + \Theta(n^2)$$

 # sub-matrices

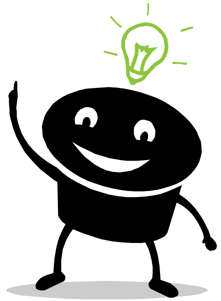
sub-matrix size

work adding sub-matrices

$$n^{\log_b a} \quad = n^{\log_2 8} = n^3 \Rightarrow case\,1$$

$$T(n) = \Theta(n^3)$$

no better than the naïve algorithm ☹



Strassen’s algorithm

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C \quad = \quad A \quad \cdot \quad B$$

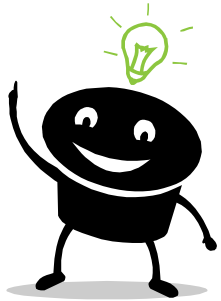
Multiply 2x2 matrices with only **7** recursive mults

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$\boxed{r = P_5 + P_4 - P_2 + P_6}$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 mults

18 adds/subs



Strassen’s algorithm

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$
$$C = A \cdot B$$

Multiply 2x2 matrices with only **7** recursive mults

$$\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}$$

$$\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}$$

Strassen's algorithm

- 1. Divide: Partition A and B into $\frac{n}{2} \times \frac{n}{2}$ sub-matrices. Form terms to be multiplied using + and –.
  2. Conquer: Perform **7** multiplications of $\frac{n}{2} \times \frac{n}{2}$ sub-matrices recursively.
  3. Combine: Form C using + and – on $\frac{n}{2} \times \frac{n}{2}$ sub-matrices.

$$T(n) = 7T(n/2) + \Theta(n^2)$$

Analysis

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow case\,1 \quad T(n) = \Theta(n^{2.81})$$

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant.

In fact, Strassen's algorithm beats the naïve algorithm for $n \geq 32$ or so.

Best to date (or theoretical interest): $\Theta(n^{2.376\ldots})$

Two more examples

Closest-Pair Problem

- Find two closest points in a set of *n* points
- Assumptions
  - Points are in a plane. $P_i = (x_i, y_i)$
  - The standard Euclidean distance is used to measure distances between points.

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

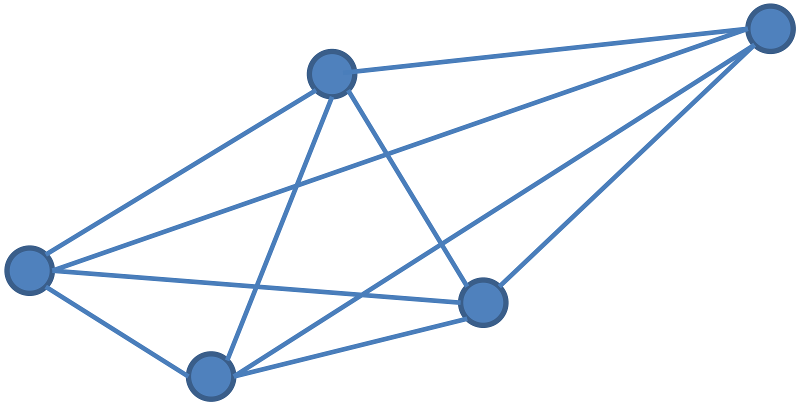
Example: P_1 = (5, 3), P_2 = (2, 8)

d(P_1, P_2) = $\sqrt{3^2 + 5^2}$ = 5.831

Brute-Force Algorithm

Compute the distance between each pair of distinct points and return the pair with the smallest distance!

C(*n*, 2) *pairs!*



Brute-Force Algorithm

Compute the distance between each pair of distinct points and return the pair with the smallest distance! *C(n, 2) pairs!*

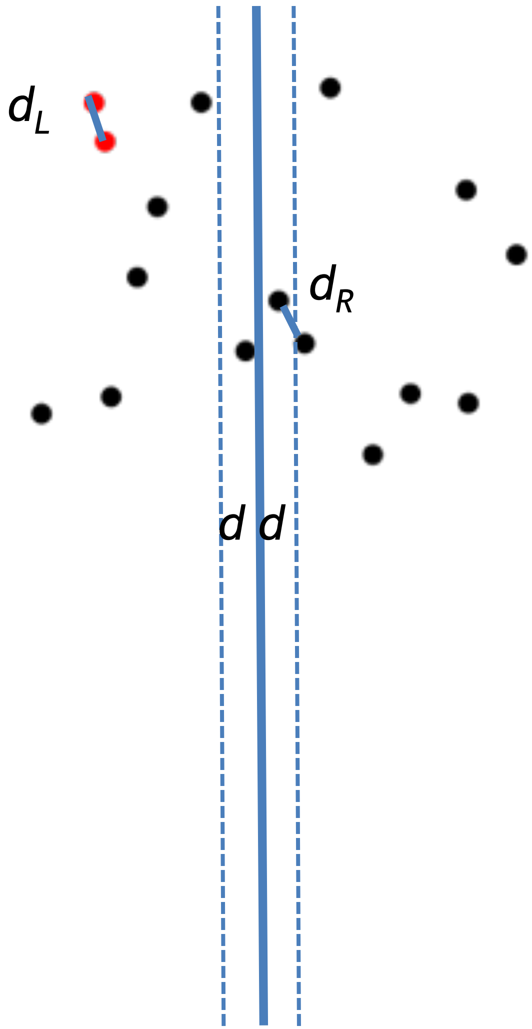
ALGORITHM *BruteForceClosestPoints(P)*

//Finds two closest points in the plane by brute force
//Input: A list P of n ($n \geq 2$) points $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$
//Output: Indices $index1$ and $index2$ of the closest pair of points

$dmin \leftarrow \infty$
for $i \leftarrow 1$ **to** $n - 1$ **do**
  for $j \leftarrow i + 1$ **to** n **do**
    $d \leftarrow sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$ //$sqrt$ is the square root function
    if $d < dmin$
      $dmin \leftarrow d$; $index1 \leftarrow i$; $index2 \leftarrow j$
return $index1, index2$

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = (n-1) + (n-2) + \ldots + 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Divide-and-Conquer Algorithm



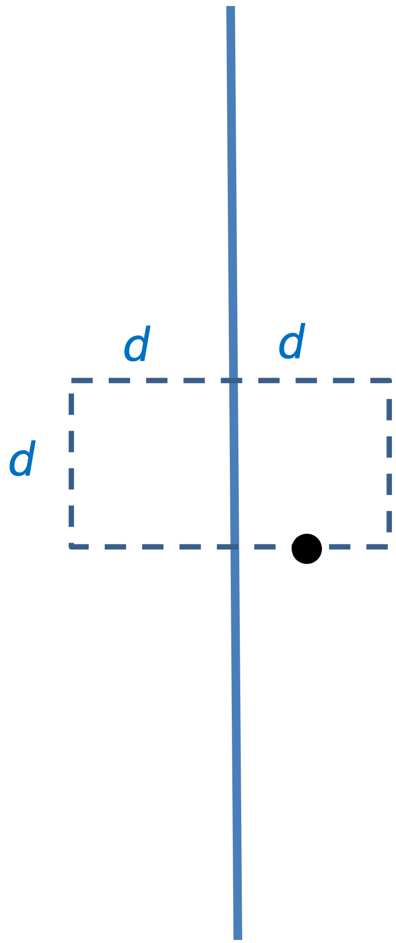
1. Divide: Bisect the point set into two sets P_L and P_R with same sizes
2. Conquer: Make two recursive calls—one to find the closest pair in P_L and the other to find the closest pair in P_R. Let $d = \min(d_L, d_R)$.

$T(n) = 2T(n/2) + \ldots$

3. Combine: Choose either d or a pair of points with one in P_L and the other in P_R

$O(n)$ using pre-sorted lists

Divide-and-Conquer Algorithm



*at most 1 point can reside in each d/2*d/2 square! check the following 7 points!*

1. Divide: Bisect the point set into two sets P_L and P_R with same sizes
2. Conquer: Make two recursive calls—one to find the closest pair in P_L and the other to find the closest pair in P_R. Let $d = \min(d_L, d_R)$.

   $T(n) = 2T(n/2) + \ldots$
3. Combine: Choose either d or a pair of points with one in P_L and the other in P_R

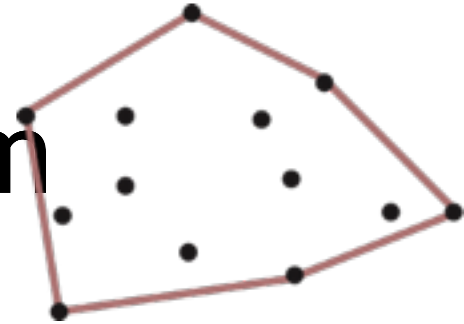
   $O(n)$ using pre-sorted lists

Divide-and-Conquer Algorithm

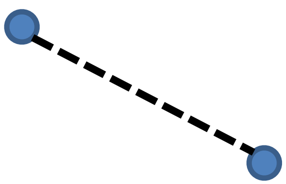
- $T(n) = 2T(n/2) + O(n)$
  - $T(n) =$ **$O(n\log n)$ < $O(n^2)$** *What are eliminated?*
  - Master theorem: case 2!

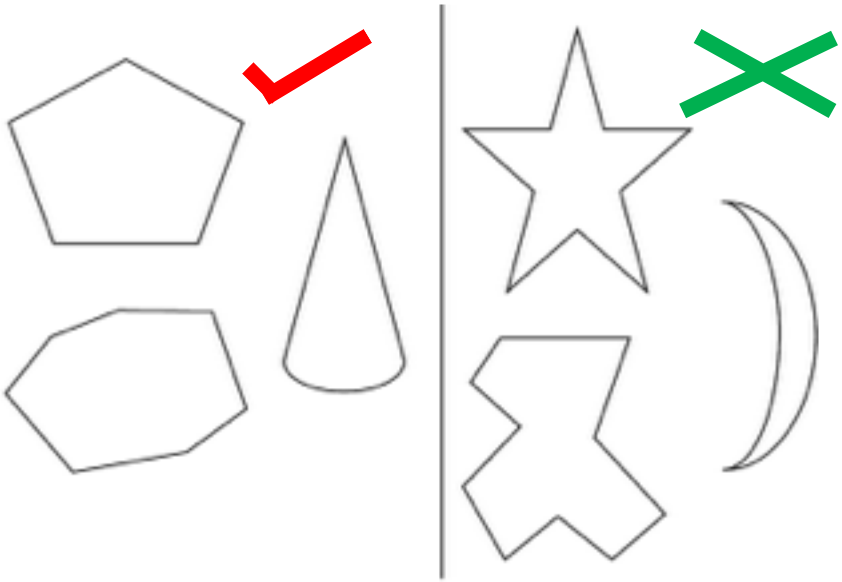
<u>Aside</u>:
Sort a sequence of *n* elements: $O(n\log n)$

Convex-Hull Problem



- Find the ***convex hull*** of a set of *n* points
- Convex set

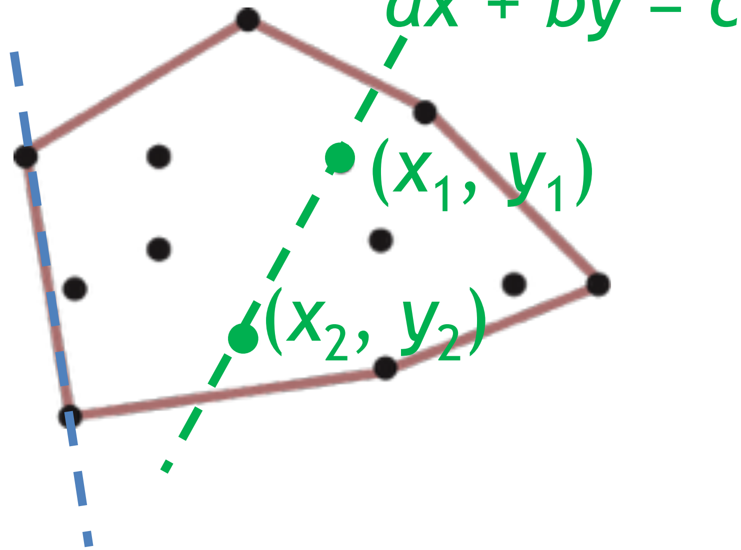




- Convex hull

The ***smallest*** convex set that contains all points

Brute-Force Algorithm



$a = y_2 - y_1$

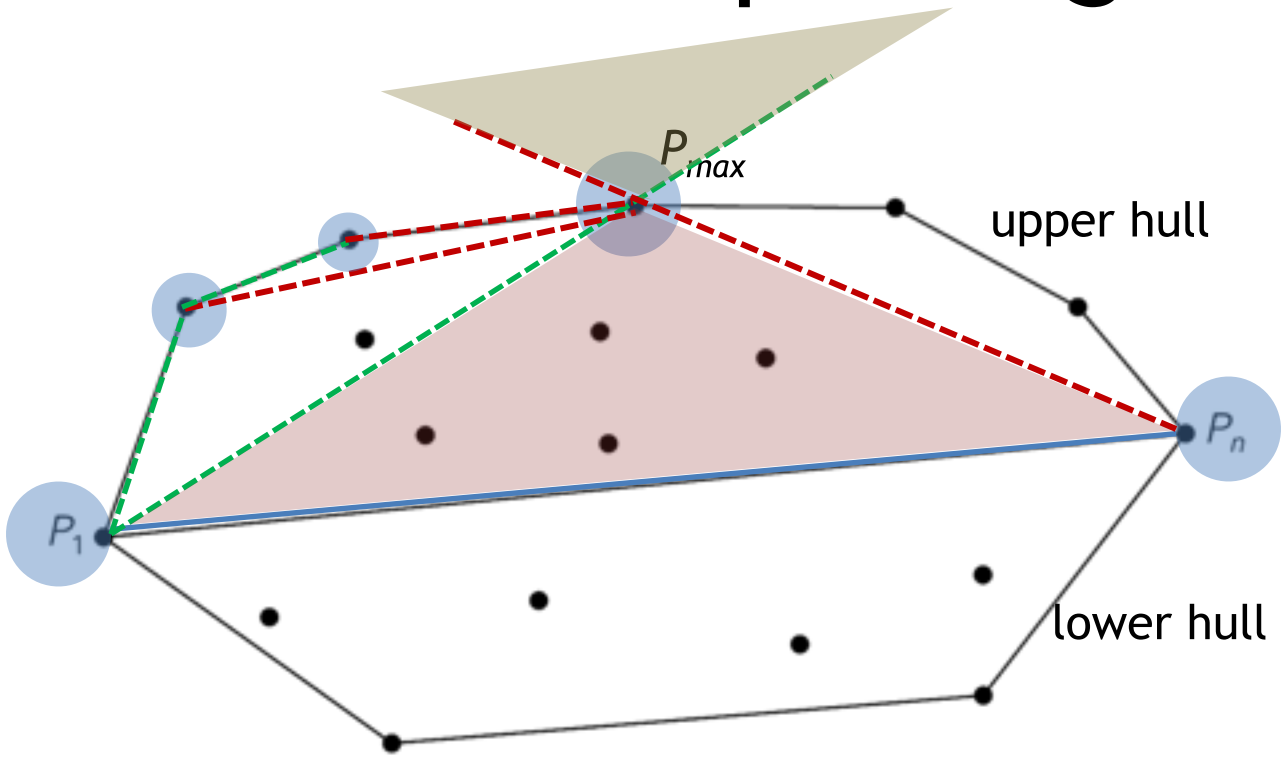
$b = x_1 - x_2$

$c = x_1y_2 - y_1x_2$

- For each of $n(n\text{-}1)/2$ pairs of distinct points
  - Check the sign of $ax+by-c$ for each of the other n-2 points

$O(n^3)$

Divide-and-Conquer Algorithm



$$T(n) = 2T(n/2) + O(n) = \boldsymbol{O(n\log n)}$$

Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method.
- The divide-and-conquer strategy often leads to efficient algorithms.

Coming up

- Sorting (Chapter 7-9)