

CSC 411
Design and Analysis of Algorithms

**Chapter 3 Brute Force and
Exhaustive Search
- Part 1**

Instructor: Minhee Jun

junm@cua.edu

Brute Force

- A straightforward approach, usually based directly on the problem's statement and definitions of the concepts involved.
- “Force’ implied by the strategy’s definition is that of a computer and not that of one’s intellect. “Just do it!
- Examples:
 1. Computing a^n ($a > 0$, n a nonnegative integer)
$$a^n = a \cdot a \cdot \cdots \cdot a$$
 2. Computing $n!$
 3. Multiplying two matrices
 4. Searching for a key of a given value in a list

Brute Force

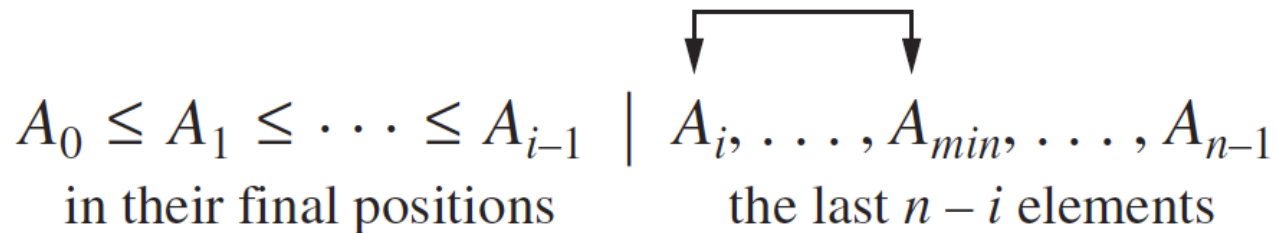
- Applicable to a very wide variety of problems.
 - Maybe the only general approach
 - Reasonable algorithms of at least some practical value with no limitation on instance size for some important problems,
 - e.g., sorting, searching, matrix multiplication, string matching
 - If only a few instances of a problem need to be solved, a brute-force algorithm can solve those instances with acceptable speed.
 - the expense of designing a more efficient algorithm may be unjustifiable
 - Useful for solving small-size instances of a problem.
 - even if too inefficient in general,
- Judge more efficient alternatives for solving a problem.

3.1 Selection Sort and Bubble Sort

- What would be the most straightforward method for solving the sorting problem?


Selection Sort: idea

- Scan the array to find its smallest element and swap it with the first element. Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second elements. Generally, *on pass i* ($0 \leq i \leq n-2$), *find the i th smallest element in $A[i..n-1]$ and swap it with $A[i]$:*



Selection Sort: Example

$A_0 \leq A_1 \leq \dots \leq A_{i-1}$ | $A_i, \dots, A_{min}, \dots, A_{n-1}$
 in their final positions the last $n - i$ elements



	89	45	68	90	29	34	17
17		45	68	90	29	34	89
17	29		68	90	45	34	89
17	29	34		90	45	68	89
17	29	34	45		90	68	89
17	29	34	45	68		90	89
17	29	34	45	68	89		90

Selection Sort: Pseudocode

ALGORITHM *SelectionSort*($A[0..n - 1]$)

//Sorts a given array by selection sort

//Input: An array $A[0..n - 1]$ of orderable elements

//Output: Array $A[0..n - 1]$ sorted in ascending order

for $i \leftarrow 0$ **to** $n - 2$ **do**

$min \leftarrow i$

for $j \leftarrow i + 1$ **to** $n - 1$ **do**

if $A[j] < A[min]$ $min \leftarrow j$

 swap $A[i]$ and $A[min]$

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i).$$

Selection Sort: Analysis

ALGORITHM *SelectionSort*($A[0..n - 1]$)

//Sorts a given array by selection sort

//Input: An array $A[0..n - 1]$ of orderable elements

//Output: Array $A[0..n - 1]$ sorted in ascending order

for $i \leftarrow 0$ **to** $n - 2$ **do**

$min \leftarrow i$

for $j \leftarrow i + 1$ **to** $n - 1$ **do**

if $A[j] < A[min]$ $min \leftarrow j$

 swap $A[i]$ and $A[min]$

- Time efficiency: $\Theta(n^2)$
- Space efficiency: $\Theta(n)$

Exercise 3.1

8. Sort the list E, X, A, M, P, L, E in alphabetical order by selection sort.
10. Is it possible to implement selection sort for linked lists with the same $\Theta(n^2)$ efficiency as the array version?

Bubble Sort: idea

- Compare adjacent elements of the list and exchange them if they are out of order. By doing it repeatedly, we end up “bubbling up” the largest element to the last position on the list. The next pass bubbles up the second largest element, and so on, until after $n - 1$ passes the list is sorted.

$$A_0, \dots, A_j \overset{?}{\leftrightarrow} A_{j+1}, \dots, A_{n-i-1} \mid A_{n-i} \leq \dots \leq A_{n-1}$$

in their final positions

Bubble Sort: Example

$$A_0, \dots, A_j \stackrel{?}{\leftrightarrow} A_{j+1}, \dots, A_{n-i-1} \mid A_{n-i} \leq \dots \leq A_{n-1}$$

in their final positions

89	$\stackrel{?}{\leftrightarrow}$	45		68		90		29		34		17
45		89	$\stackrel{?}{\leftrightarrow}$	68		90		29		34		17
45		68		89	$\stackrel{?}{\leftrightarrow}$	90	$\stackrel{?}{\leftrightarrow}$	29		34		17
45		68		89		29		90	$\stackrel{?}{\leftrightarrow}$	34		17
45		68		89		29		34		90	$\stackrel{?}{\leftrightarrow}$	17
45		68		89		29		34		17		90
45	$\stackrel{?}{\leftrightarrow}$	68	$\stackrel{?}{\leftrightarrow}$	89	$\stackrel{?}{\leftrightarrow}$	29		34		17		90
45		68		29		89	$\stackrel{?}{\leftrightarrow}$	34		17		90
45		68		29		34		89	$\stackrel{?}{\leftrightarrow}$	17		90
45		68		29		34		17		89		90

etc.

Bubble Sort: Pseudocode

ALGORITHM *BubbleSort*($A[0..n - 1]$)

//Sorts a given array by bubble sort

//Input: An array $A[0..n - 1]$ of orderable elements

//Output: Array $A[0..n - 1]$ sorted in nondecreasing order

for $i \leftarrow 0$ **to** $n - 2$ **do**

for $j \leftarrow 0$ **to** $n - 2 - i$ **do**

if $A[j + 1] < A[j]$ swap $A[j]$ and $A[j + 1]$

$$\begin{aligned} C(n) &= \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n - 2 - i) - 0 + 1] \\ &= \sum_{i=0}^{n-2} (n - 1 - i) = \frac{(n - 1)n}{2} \in \Theta(n^2). \end{aligned}$$

Bubble Sort: Analysis

ALGORITHM *BubbleSort*($A[0..n - 1]$)

//Sorts a given array by bubble sort

//Input: An array $A[0..n - 1]$ of orderable elements

//Output: Array $A[0..n - 1]$ sorted in nondecreasing order

for $i \leftarrow 0$ **to** $n - 2$ **do**

for $j \leftarrow 0$ **to** $n - 2 - i$ **do**

if $A[j + 1] < A[j]$ swap $A[j]$ and $A[j + 1]$

- Time efficiency: $\Theta(n^2)$
- Space efficiency: $\Theta(n)$

Exercise 3.1

- **11.** Sort the list *E, X, A, M, P, L, E* in alphabetical order by bubble sort.

Exercise 3.1

- 14. Alternating disks** You have a row of $2n$ disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks.



Design an algorithm for solving this puzzle and determine the number of moves it takes. [Gar99]

3.2 Sequential Search and Brute-Force String Matching

- What would be the most straightforward method for solving the searching problem?

Sequential Search: idea & Pseudocode

ALGORITHM *SequentialSearch2*($A[0..n]$, K)

//Implements sequential search with a search key as a sentinel

//Input: An array A of n elements and a search key K

//Output: The index of the first element in $A[0..n - 1]$ whose value is

// equal to K or -1 if no such element is found

$A[n] \leftarrow K$

$i \leftarrow 0$

while $A[i] \neq K$ **do**

$i \leftarrow i + 1$

if $i < n$ **return** i

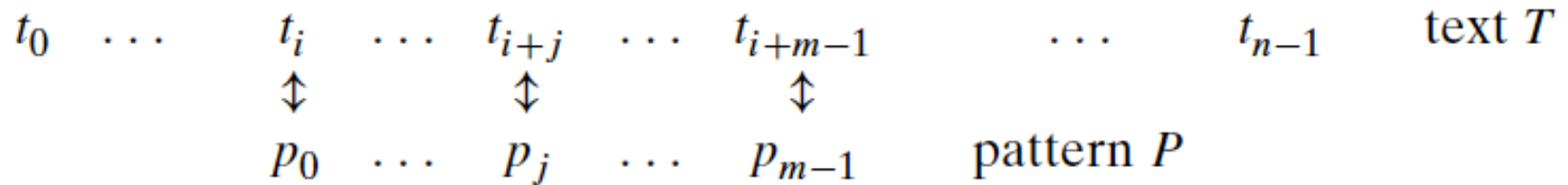
else return -1

Brute-Force String Matching:

- given a string of n characters (called the text) and a string of m characters (called the pattern), find a substring of the text that matches the pattern ($m \leq n$).

t_0	\dots	t_i	\dots	t_{i+j}	\dots	t_{i+m-1}	\dots	t_{n-1}	text T
		\Downarrow		\Downarrow		\Downarrow			
		p_0	\dots	p_j	\dots	p_{m-1}			pattern P

Brute-Force String Matching: idea



Brute-force algorithm

Step 1 Align pattern at beginning of text

Step 2 Moving from left to right, **compare** each character of pattern to the corresponding character in text until

- all characters are found to match (successful search); or
- a mismatch is detected

Step 3 While pattern is not found and the text is not yet exhausted, realign pattern **one position to the right** and repeat Step 2

Brute-Force String Matching: Example

$$\begin{array}{ccccccc}
t_0 & \dots & t_i & \dots & t_{i+j} & \dots & t_{i+m-1} & \dots & t_{n-1} & \text{text } T \\
& & \updownarrow & & \updownarrow & & \updownarrow & & & \\
& & p_0 & \dots & p_j & \dots & p_{m-1} & & & \text{pattern } P
\end{array}$$

N O B O D Y _ N O T I C E D _ H I M
N O N T O N T O N T O N T O N T O T

The following are some examples of how you can use the word "ton".

Brute-Force String Matching: Pseudocode

ALGORITHM *BruteForceStringMatch*($T[0..n - 1]$, $P[0..m - 1]$)

//Implements brute-force string matching
//Input: An array $T[0..n - 1]$ of n characters representing a text and
// an array $P[0..m - 1]$ of m characters representing a pattern
//Output: The index of the first character in the text that starts a
// matching substring or -1 if the search is unsuccessful

for $i \leftarrow 0$ **to** $n - m$ **do**
 $j \leftarrow 0$
 while $j < m$ **and** $P[j] = T[i + j]$ **do**
 $j \leftarrow j + 1$
 if $j = m$ **return** i
return -1

$$C(n) = m(n - m + 1)$$

Brute-Force String Matching: Analysis

ALGORITHM *BruteForceStringMatch*($T[0..n - 1]$, $P[0..m - 1]$)

//Implements brute-force string matching

//Input: An array $T[0..n - 1]$ of n characters representing a text and

// an array $P[0..m - 1]$ of m characters representing a pattern

//Output: The index of the first character in the text that starts a

// matching substring or -1 if the search is unsuccessful

for $i \leftarrow 0$ **to** $n - m$ **do**

$j \leftarrow 0$

while $j < m$ **and** $P[j] = T[i + j]$ **do**

$j \leftarrow j + 1$

if $j = m$ **return** i

return -1

$$C(n) = m(n - m + 1)$$

Time efficiency:

$$O(mn)$$

$$\Theta(n) \quad \text{if} \quad n \gg m$$

Exercise 3.2

4. Determine the number of character comparisons made by the brute-force algorithm in searching for the pattern GANDHI in the text

THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

Assume that the length of the text—it is 47 characters long—is known before the search starts.

Exercise 3.2

5. How many comparisons (both successful and unsuccessful) will be made by the brute-force algorithm in searching for each of the following patterns in the binary text of one thousand zeros?
- a.** 00001 **b.** 10000 **c.** 01010

3.3 Closest-Pair and Convex-Hull Problem by Brute Force

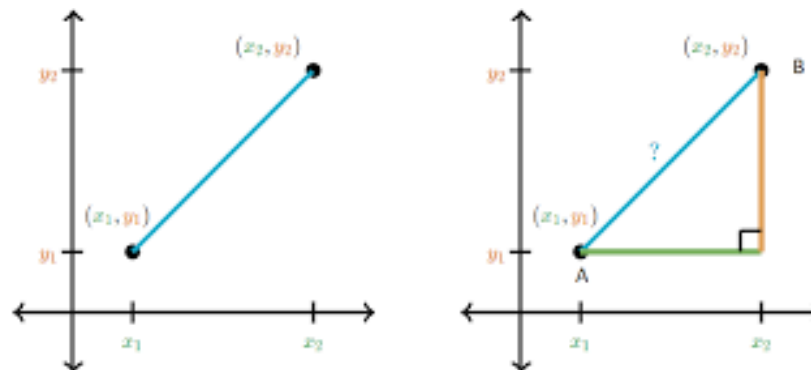
- What would be the straightforward method for the computational geometry and operations?

Closest-Pair Problem: idea

- Find the two closest points in a set of n points (in the two-dimensional Cartesian plane).
- Brute-force algorithm
 - Compute the **distance** between every pair of distinct points and return the indexes of the points for which the distance is the smallest.

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

euclidean vs
manhattan distance



Closest-Pair Brute-Force Algorithm: Pseudocode and Analysis

ALGORITHM *BruteForceClosestPoints(P)*

//Input: A list P of n ($n \geq 2$) points $P_1 = (x_1, y_1), \dots, P_n = (x_n, y_n)$

//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**

$d \leftarrow \text{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2)$ //sqrt is the square root function

if $d < dmin$

$dmin \leftarrow d; index1 \leftarrow i; index2 \leftarrow j$

return $index1, index2$

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

$$= 2[(n - 1) + (n - 2) + \dots + 1] = (n - 1)n$$

$$\Theta(n^2)$$

Exercise 3.3

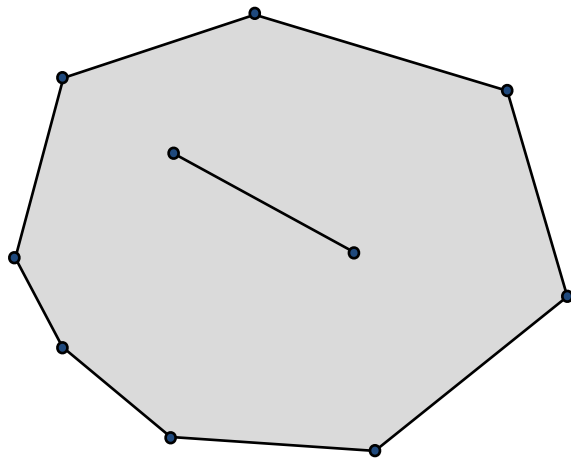
7. The closest-pair problem can be posed in the k -dimensional space, in which the Euclidean distance between two points $p'(x'_1, \dots, x'_k)$ and $p''(x''_1, \dots, x''_k)$ is defined as

$$d(p', p'') = \sqrt{\sum_{s=1}^k (x'_s - x''_s)^2}.$$

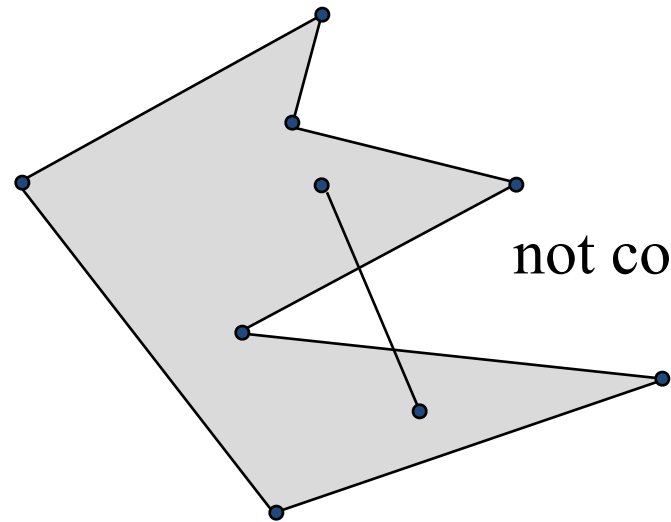
What is the time-efficiency class of the brute-force algorithm for the k -dimensional closest-pair problem?

Convex-Hull Problem: Convex

- Definition of convex set:
A set of points (finite or infinite) in the plane is called **convex** if for any two points P and Q in the set, the entire line segment with the endpoints at P and Q belongs to the set



convex



not convex

Convex Hull Problem:

Examples of Convex Sets

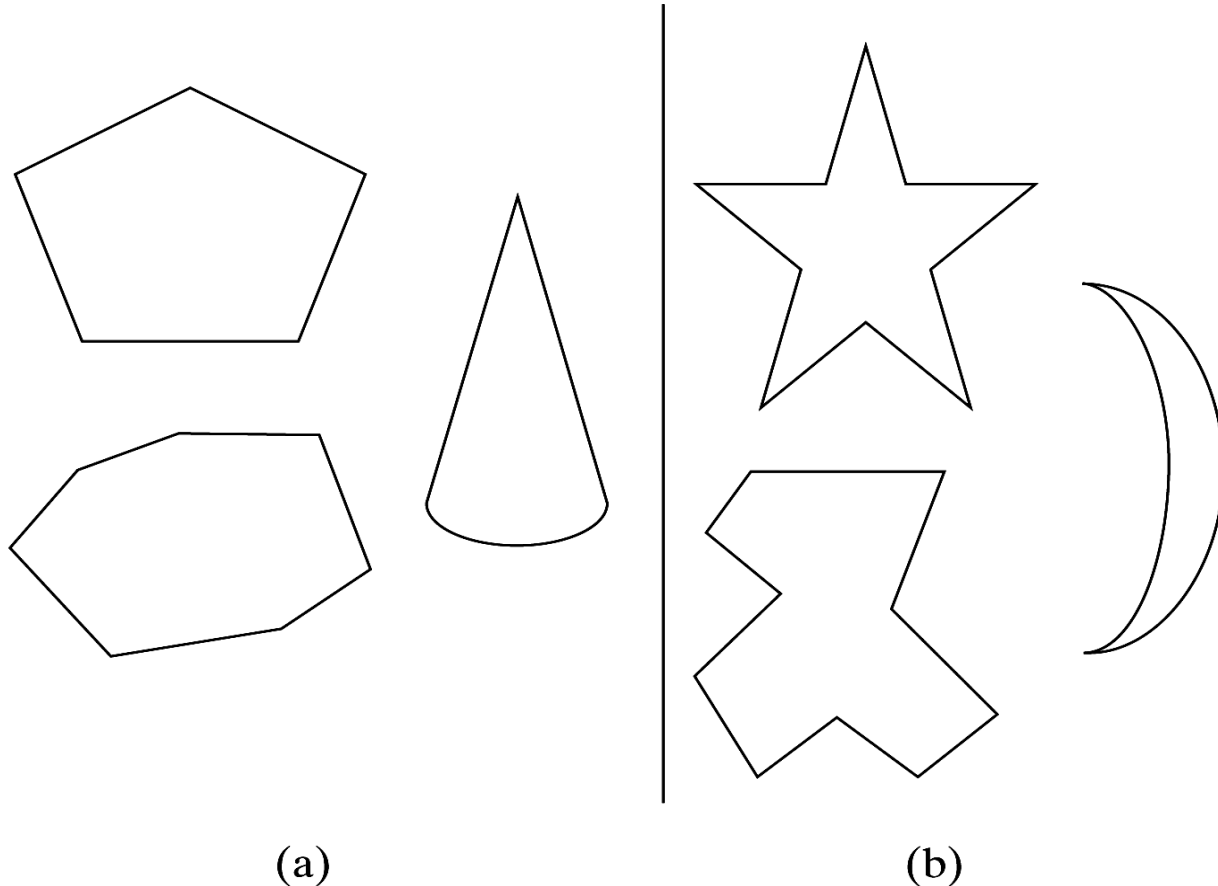
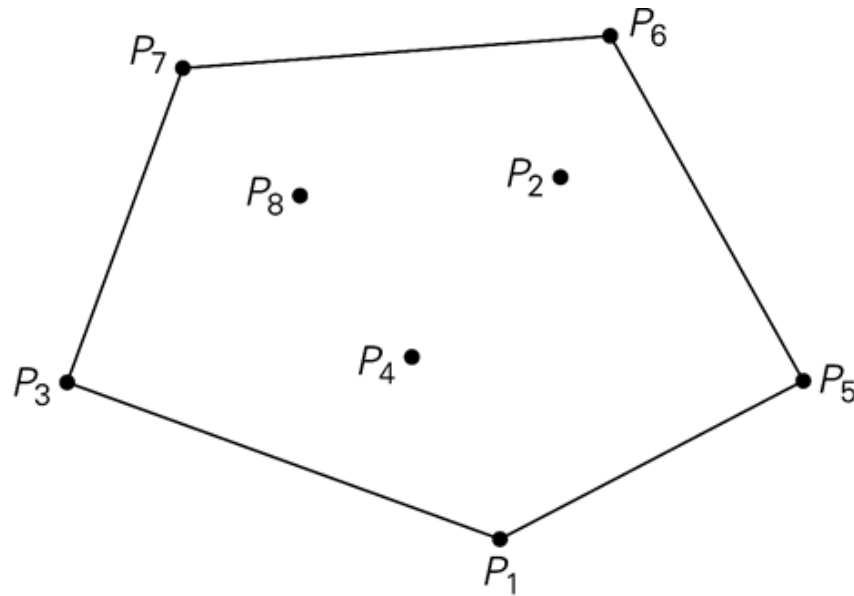


FIGURE 3.4 (a) Convex sets. (b) Sets that are not convex.

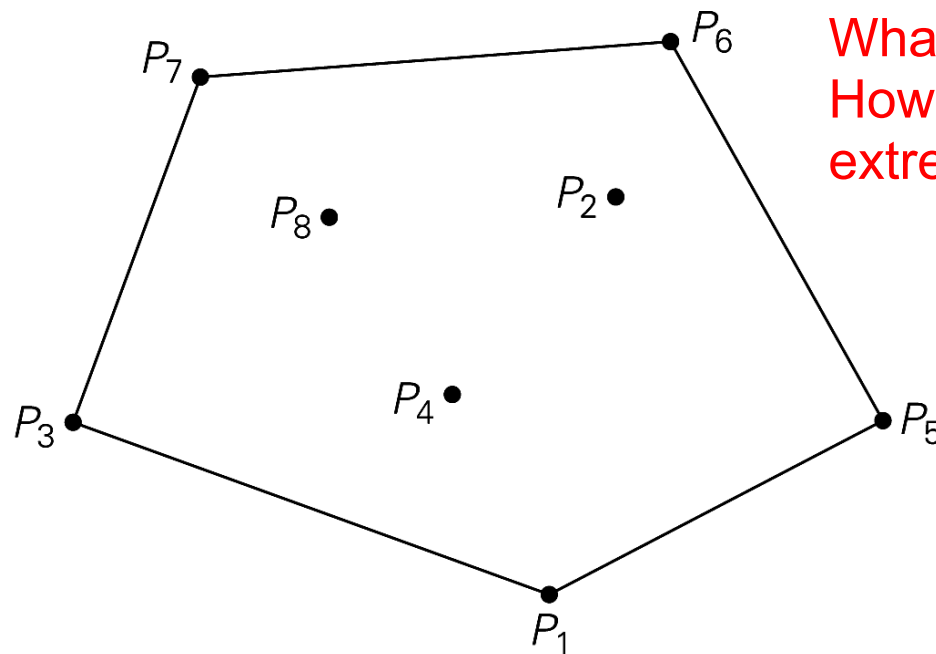
Convex-Hull Problem

- Convex Hull: the smallest convex set containing a set S of points.
- Convex-Hull Problem: construct the convex hull for a given set S of n points.



Convex-Hull Problem

- Extreme points: a point of this set that is not a middle point of any line segment with endpoints in the set

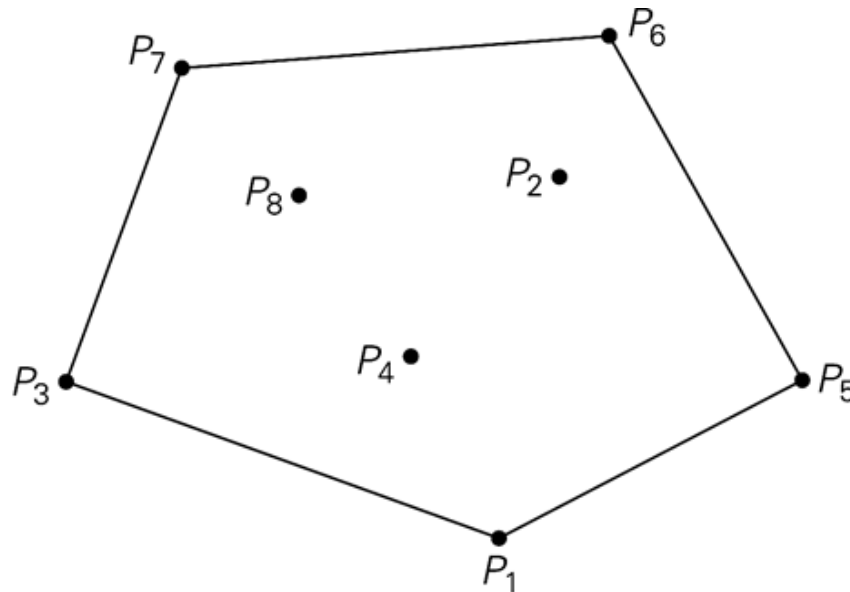


What are extreme points?
How to identify those
extreme points?

FIGURE 3.6 The convex hull for this set of eight points is the convex polygon with vertices at P_1 , P_5 , P_6 , P_7 , and P_3 .

Convex-Hull Brute-Force Algorithm:

- Find the pairs of points (P_i, P_j) from a set of n points
 - The line segment connecting P_i and P_j is a part of its convex hull's boundary if and only if the other points of the set lie on the same side of the straight line through P_i and P_j

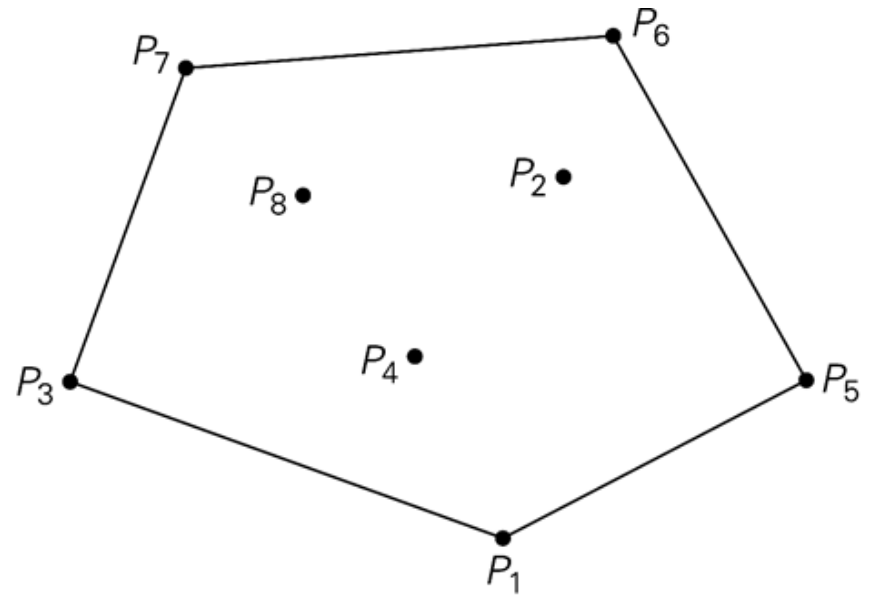


Convex-Hull Brute-Force Algorithm:

- The straight line through two points (x_1, y_1) , (x_2, y_2) in the coordinate plane can be defined by the following equation
 - $ax + by = c$
 - where $a = y_2 - y_1$, $b = x_1 - x_2$, $c = x_1*y_2 - y_1*x_2$
- Such a line **divides** the plane into two half-planes: for all the points in one of them: $ax + by > c$, while for all the points in the other, $ax + by < c$.

Convex-Hull Brute-Force Algorithm:

- Algorithm: For each pair of points p_i and p_j determine whether all other points lie to the same side of the straight line through p_i and p_j , i.e. whether $ax+by-c$ all have **the same sign** (no 3+ points are co-linear)
- Efficiency: $\Theta(n^3)$
- Can we do better?
 - Divide-and-conquer
 - (see quickhull algorithm
 - in chapter 5)



Exercise 3.3

12. Consider the following small instance of the linear programming problem:

$$\begin{array}{ll}\text{maximize} & 3x + 5y \\ \text{subject to} & x + y \leq 4 \\ & x + 3y \leq 6 \\ & x \geq 0, y \geq 0.\end{array}$$

- a. Sketch, in the Cartesian plane, the problem's *feasible region*, defined as the set of points satisfying all the problem's constraints.
- b. Identify the region's extreme points.
- c. Solve this optimization problem by using the following theorem: A linear programming problem with a nonempty bounded feasible region always has a solution, which can be found at one of the extreme points of its feasible region.

Brute-Force Strengths and Weaknesses

- Strengths
 - wide applicability
 - simplicity
 - yields reasonable algorithms for some important problems
(e.g., matrix multiplication, sorting, searching, string matching)
- Weaknesses
 - rarely yields efficient algorithms
 - some brute-force algorithms are unacceptably slow
 - not as constructive as some other design techniques

Brute Force algorithms we have seen

- Selection Sort
- Bubble Sort
- String Matching
- Closest-Pair Problem
- Convex Hull Problem

What we will see next

- Traveling Salesman Problem
- Knapsack Problem
- The Assignment Problem