

Brute-Force Algorithms

Brute Force Approach

Brute force is a **straightforward approach** to solving a problem, directly based on the **problem statement** and **definitions** of the concepts involved.

Example: An algorithm for **computing** a^n for a nonzero number a and non-negative integer n .

$$a^n = \underbrace{a \cdot a \cdot \cdots \cdot a}_n$$

Brute Force Approach

Example: The consecutive integer checking algorithm for computing $\text{gcd}(a, b)$.

Example: The definition based algorithm for matrix multiplication.

Brute Force Approach

Cons:

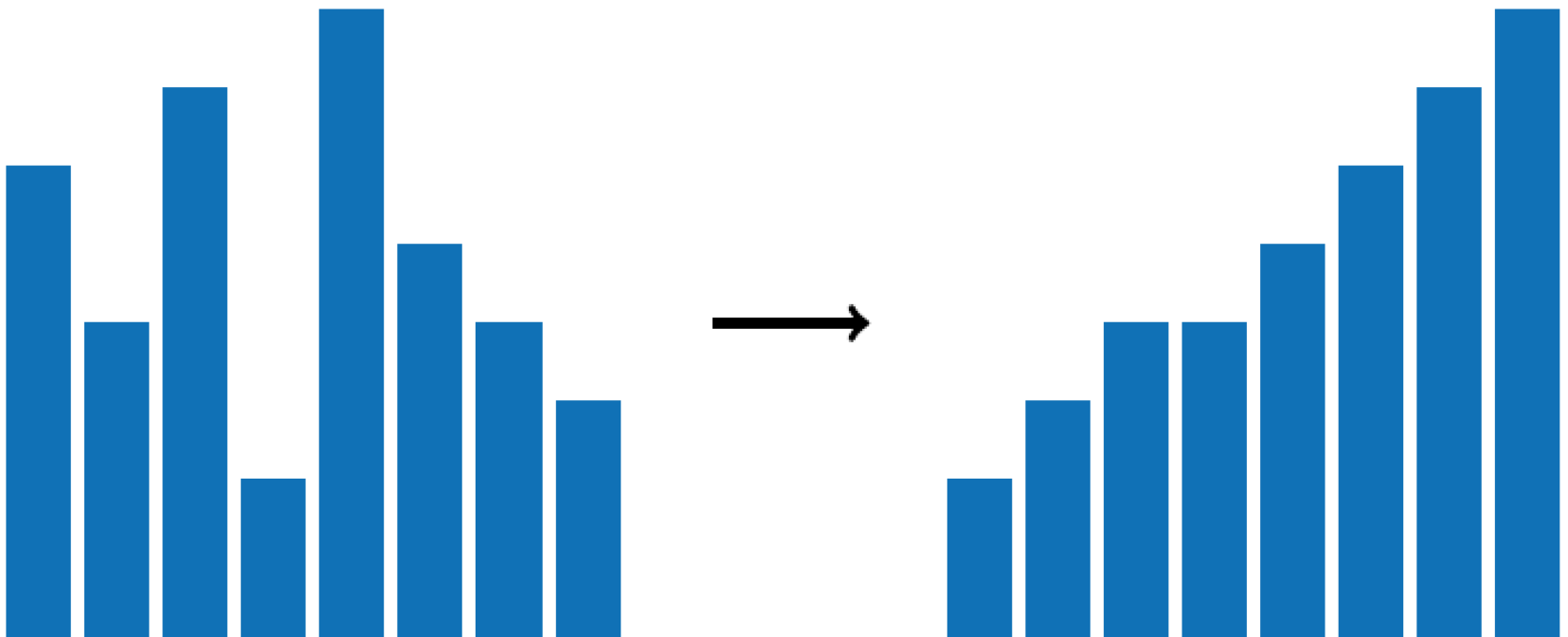
Inefficient

Brute Force Approach

Pros:

- Brute-force is **applicable** to many problems.
- For some important problems brute-force yields **reasonable** algorithms.
- The expenses of designing a more efficient algorithm is **unjustifiable** if only a few instances of problem need to be solved, and brute-force can solve in **acceptable speed**.
- A brute-force is useful for solving **small size instance** of a problem.

Sorting



Sorting Problem

Input: Sequence (array) $A[0..n - 1]$

Output: Permutation $A'[0..n - 1]$ of $A[0..n - 1]$
in **increasing (nondecreasing) order**

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort

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

- Find a **minimum** by scanning the array
- **Swap** it with the first element
- **Repeat** with the remaining part of the array

Selection Sort: Algorithm

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

// Input: Array $A[0..n - 1]$

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

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

$\text{min} \leftarrow i$

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

if $A[j] < A[\text{min}]$

$\text{min} \leftarrow j$

 swap $A[i]$ and $A[\text{min}]$

Analysis of Selection Sort

- **The input size** is the number of elements **n** in the list.
- **The basic operation** is the key comparison

$$A[j] < A[\text{min}]$$

- **C(n)** is the **total number of comparisons**:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \dots = \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2)$$

Note: The number of key **swaps** is $\Theta(n)$

Activity (in group)

- ‘Design’ a brute-force algorithm
- Write a pseudocode (input, output, algorithm)
- Analyze
 - Input size?
 - What is the basic operation?
 - Number of times the basic operation is executed?
($T(n)$)
- Check & Share
- Extra: find more efficient algorithm

Activity (in group)

- ‘Design’ a brute-force algorithm
 - Sorting based on bubble sort
 - String matching
 - Closest-pair problem

Bubble Sort

Idea:

- Compare adjacent elements and swap them if they are “out of order”.
- Repeat “bubbling up” the largest element to the last position.
- In the next pass, bubble up the second largest element, and so on.

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

Bubble Sort

89 45 68 90 29 34 17



Bubble Sort

89 45 68 90 29 34 17

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

Bubble Sort

89 45 68 90 29 34 17

45 89 68 90 29 34 17

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

Bubble Sort

89 45 68 90 29 34 17

45 89 68 90 29 34 17

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

Bubble Sort

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



Bubble Sort

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



Bubble Sort

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

Bubble Sort

45 ↔ ? 68

89

29

34

17

90


Bubble Sort

45 68  89 29 34 17 


Bubble Sort

45 68 89  29 34 17 

Bubble Sort

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

Bubble Sort

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

Bubble Sort

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

...

String Matching

Problem: 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.

Text T: $t_0 \cdots t_i \cdots t_{i+j} \cdots t_{i+m-1} \cdots t_{n-1}$

Pattern P: $p_0 \cdots p_j \cdots p_{m-1}$

$\downarrow \qquad \qquad \downarrow \qquad \qquad \downarrow$

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

String Matching: Example

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

Closest-Pair Problem

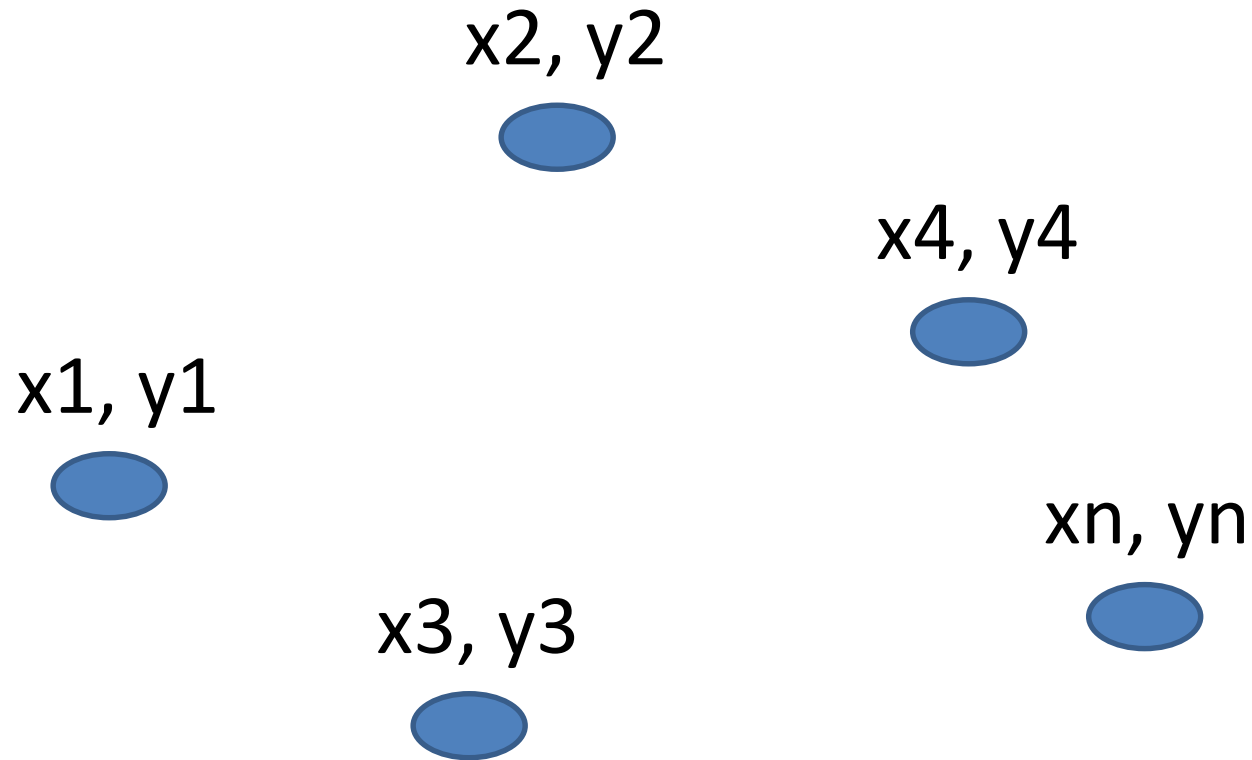
Problem: Find the **two closest points** in a set of **n** points

- Post-offices, airplanes, databases, statistical samples, DNA sequences, etc.
- For numerical data, use **Euclidean distance**, for nonnumeric data (texts), use **Hamming distance**.

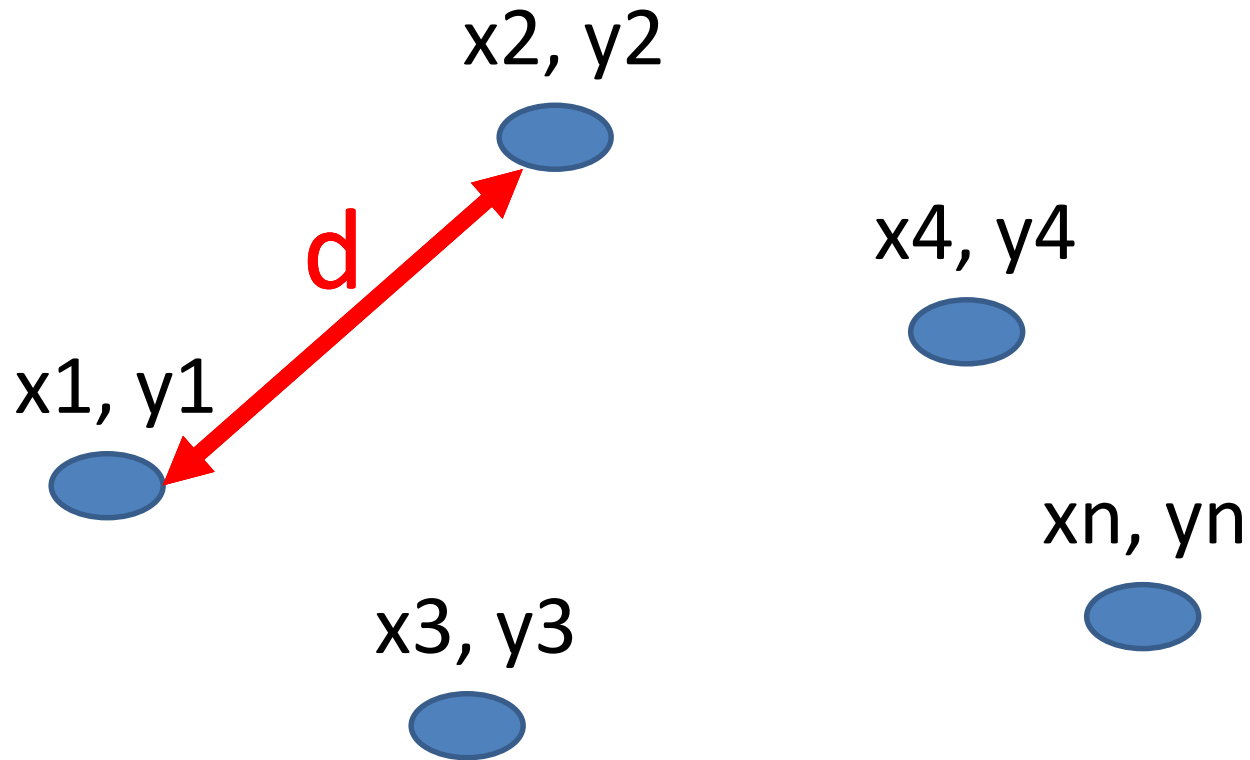
Two-dimensional case: the distance between two points $p_i(x_i, y_i)$ and $p_j(x_j, y_j)$:

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

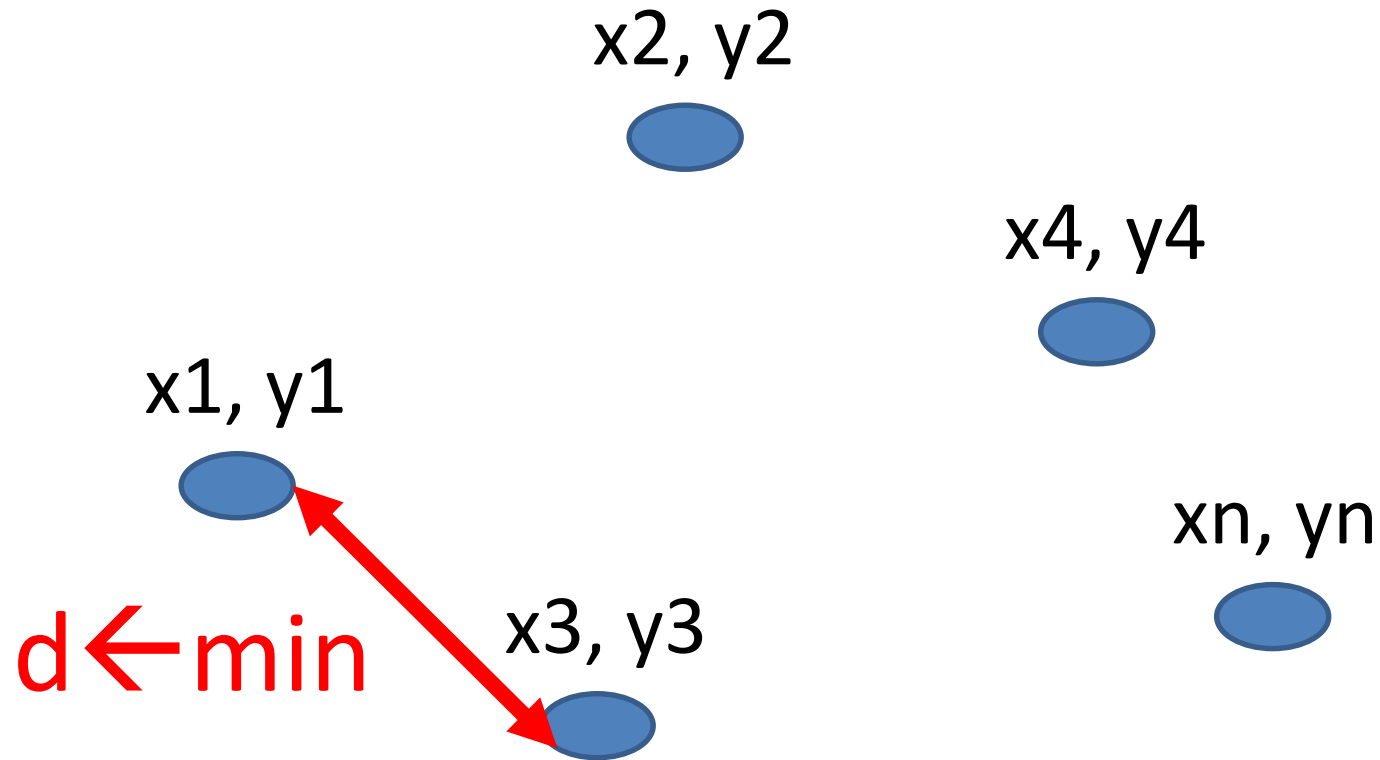
Closest-Pair Problem



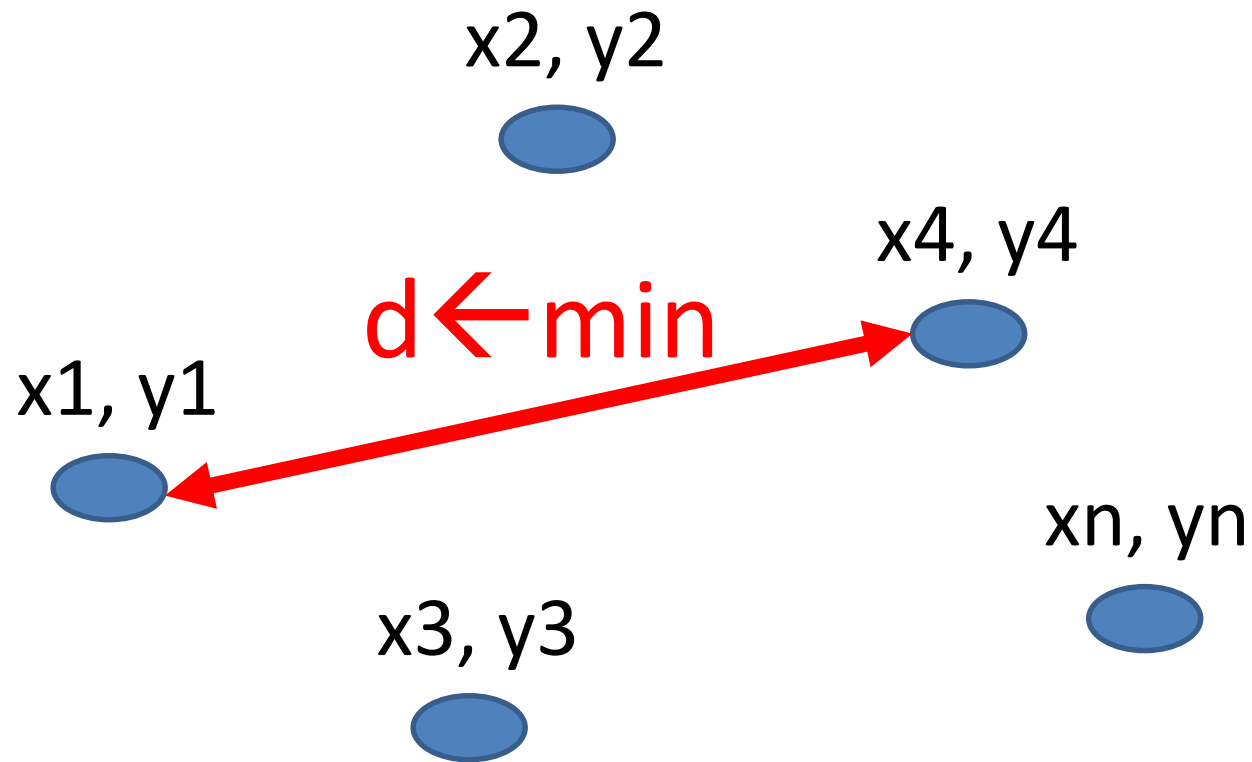
Closest-Pair Problem



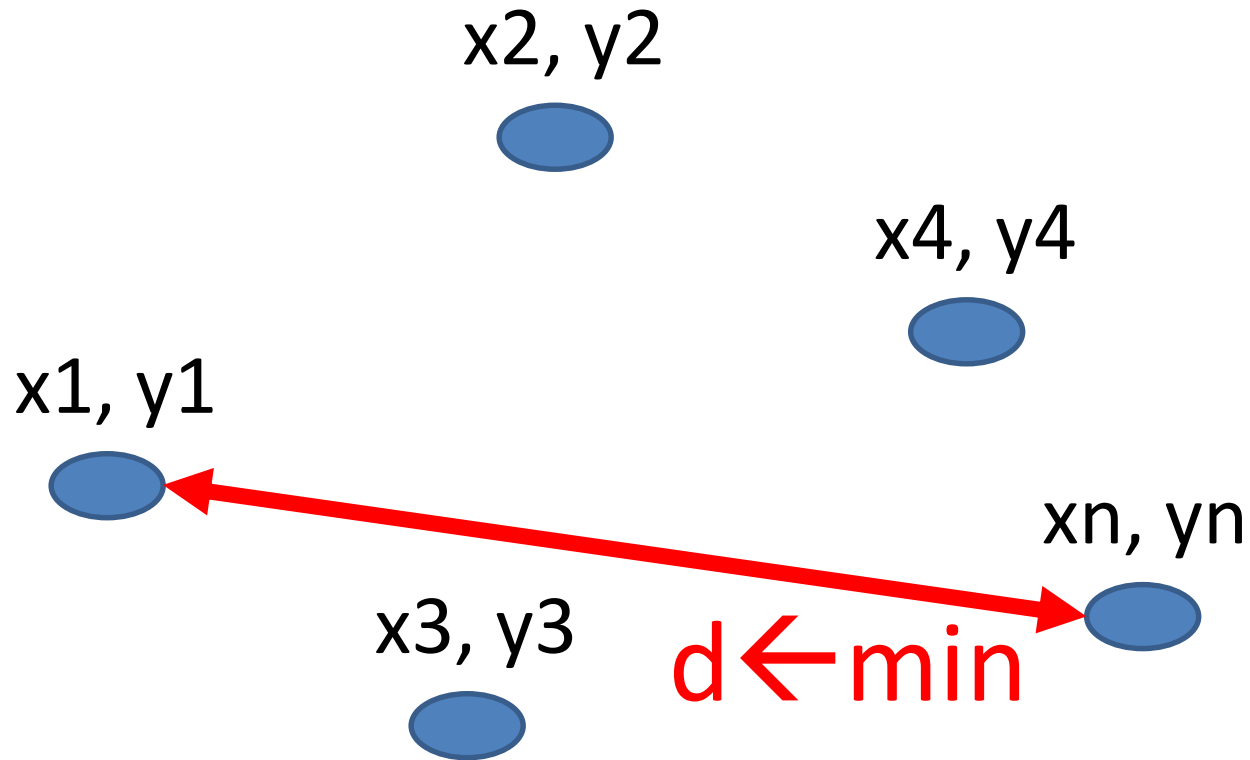
Closest-Pair Problem



Closest-Pair Problem



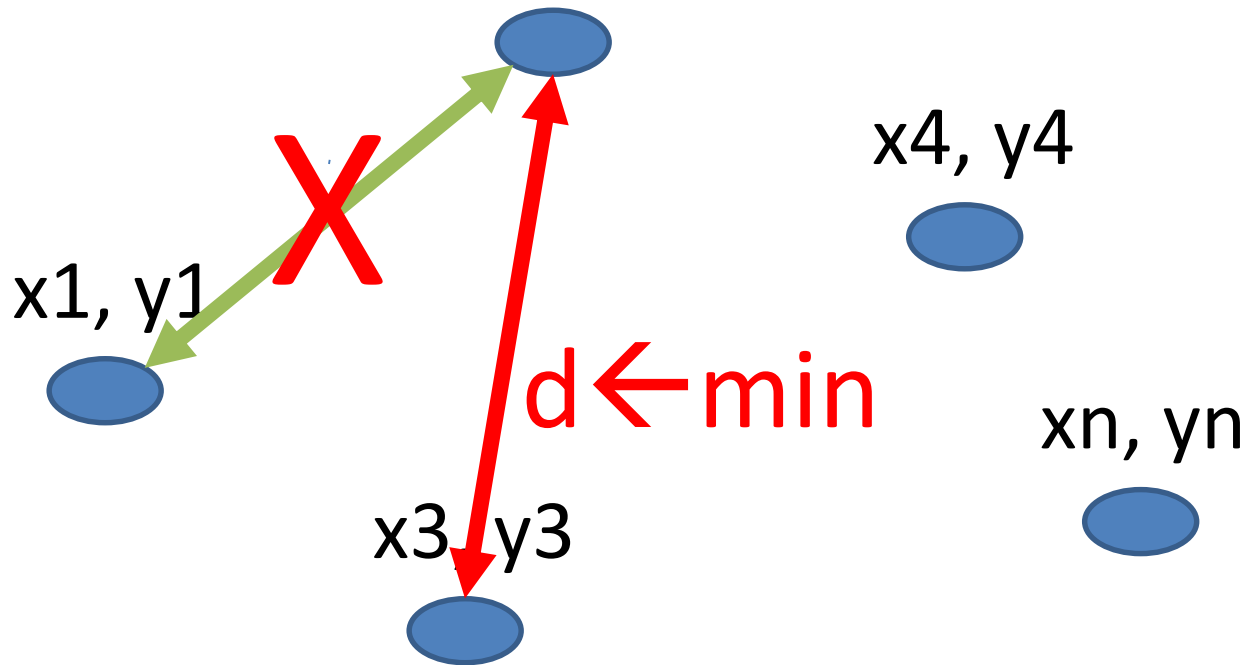
Closest-Pair Problem



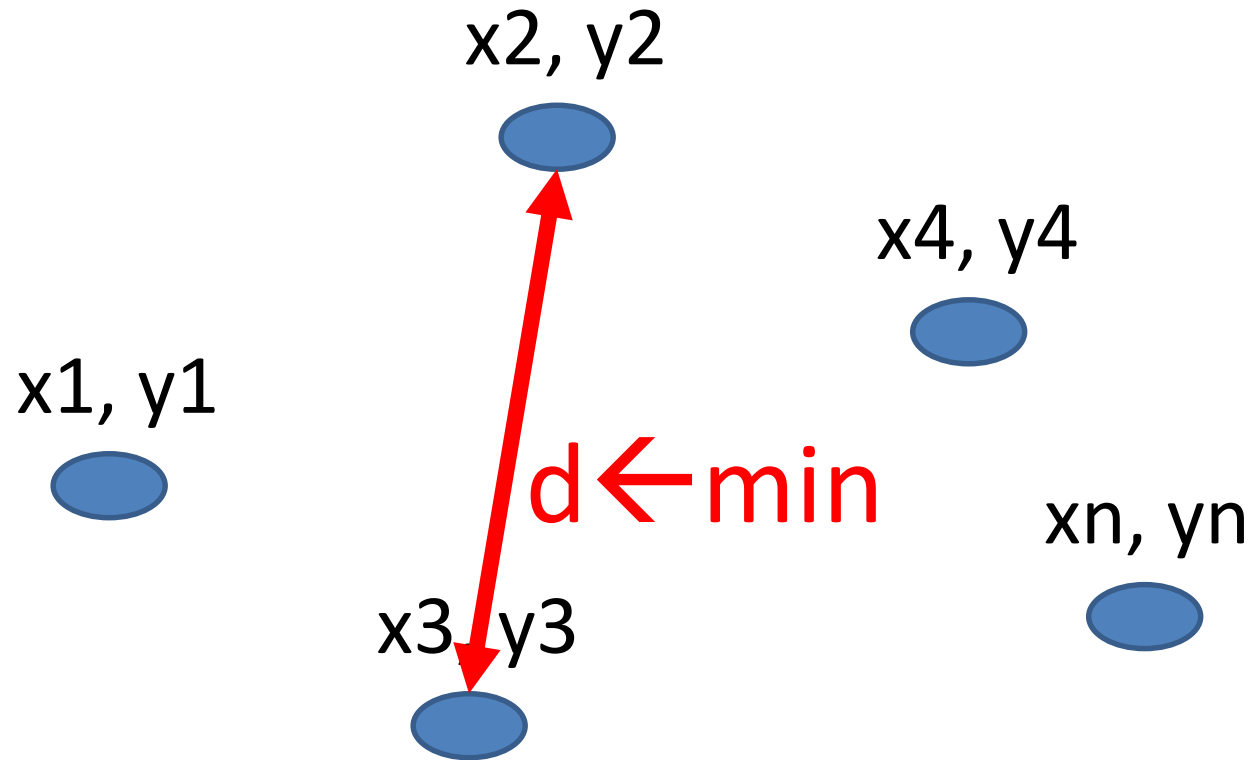
Closest-Pair Problem

Don't want to calculate the same distance:

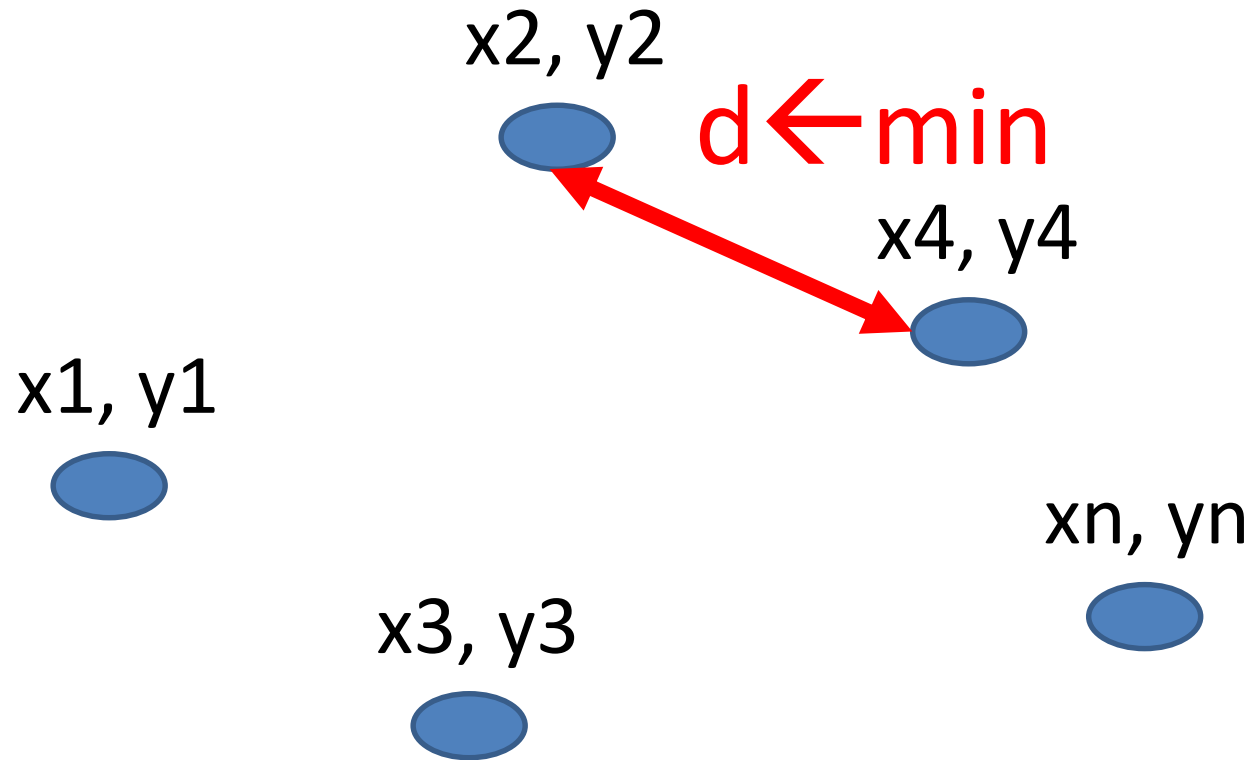
Consider (p_i, p_j) where $i < j$



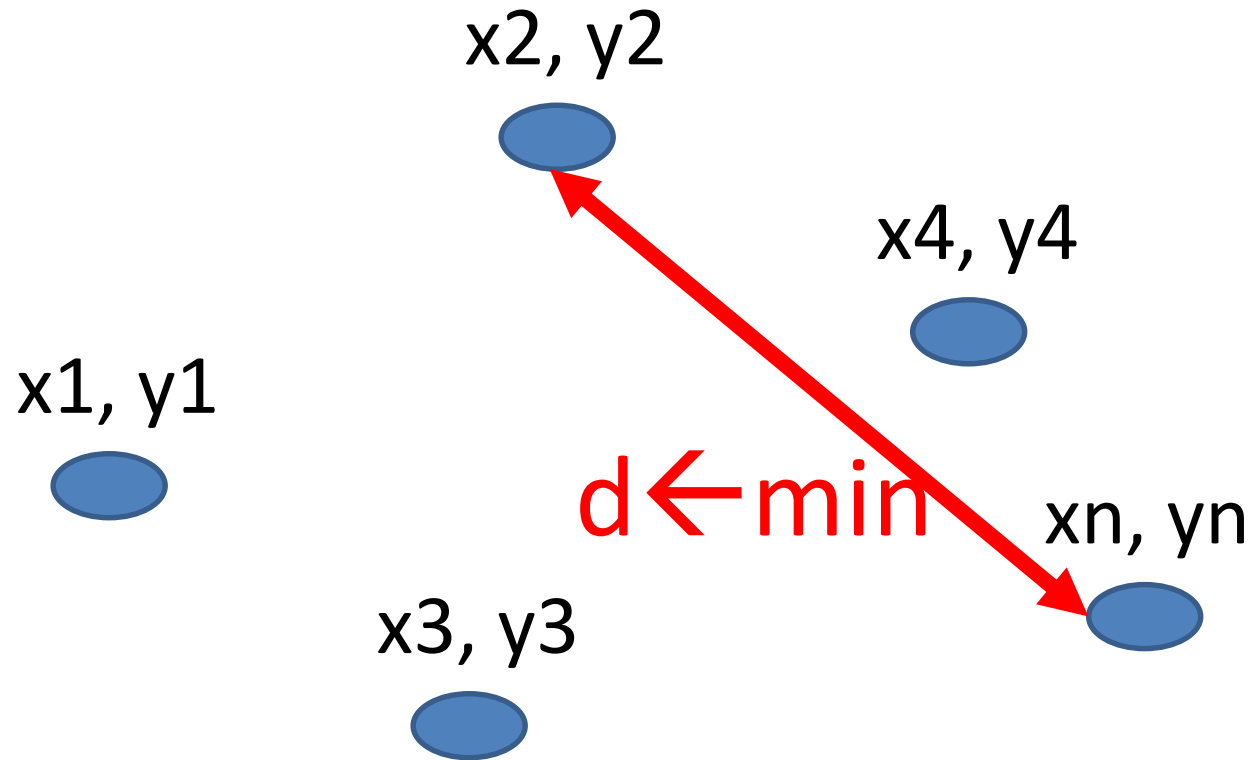
Closest-Pair Problem



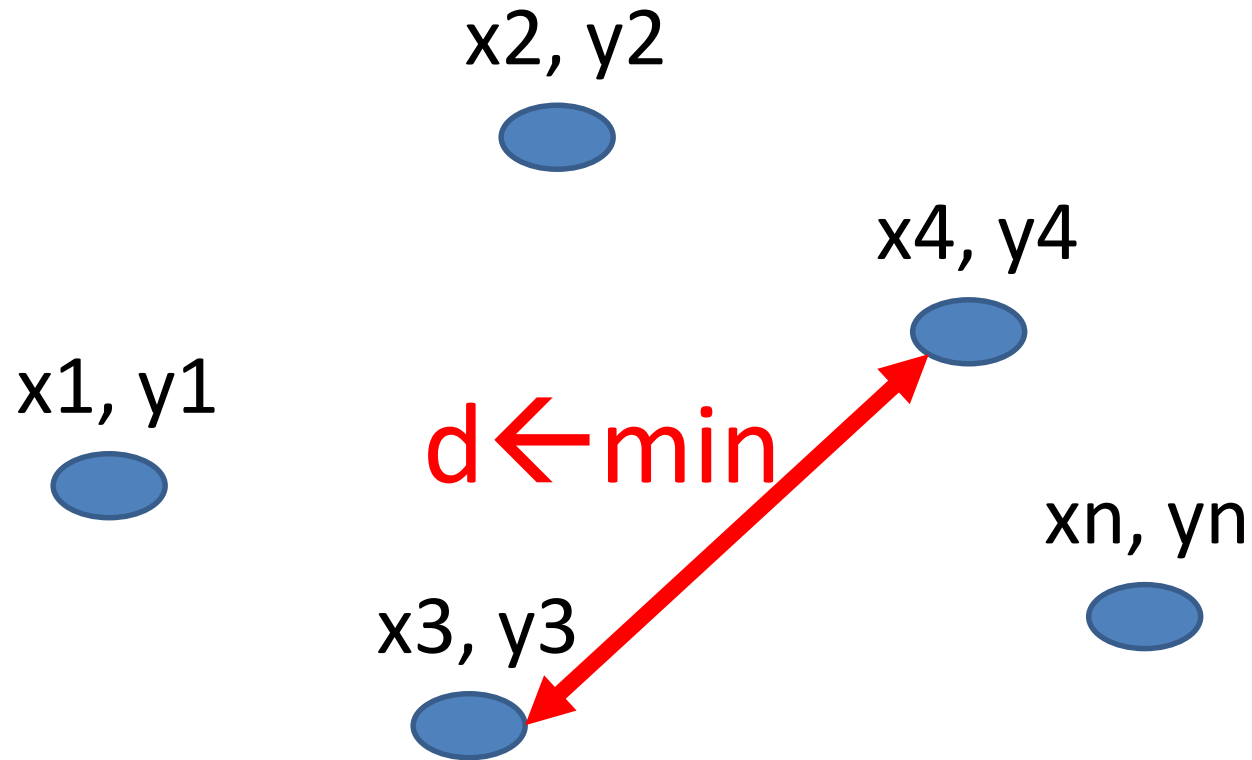
Closest-Pair Problem



Closest-Pair Problem



Closest-Pair Problem



EXHAUSTIVE SEARCH ALGORITHMS