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 $\gcd(a, b)$.

Example: The definition based algorithm for matrix multiplication.

Brute Force Approach

Cons:

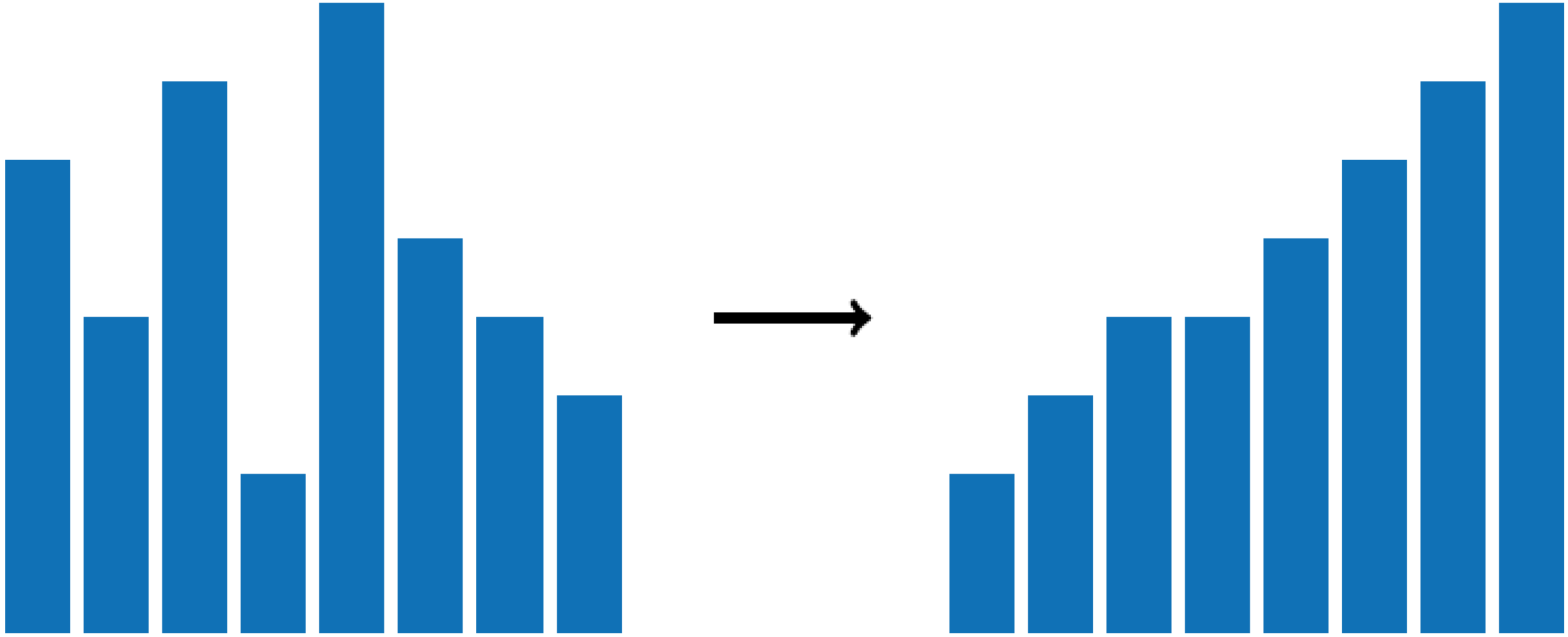
Inefficient

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) $\mathrm{A}[0..\mathrm{n}-1]$

Output: Permutation $\mathrm{A}'[0..\mathrm{n}-1]$ of $\mathrm{A}[0..\mathrm{n}-1]$
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**

$\quad \min \leftarrow i$

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

$\quad\quad$ **if** $A[j] < A[\min]$

$\quad\quad\quad \min \leftarrow j$

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

Analysis of Selection Sort

- **The input size** is the number of elements n in the list.

- **The basic operation** is the key comparison

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

- $\mathrm{C(n)}$ is the **total number of comparisons**:

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

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

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

https://padlet.com/sem2_2018_2019/dsa2

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.

$$A_0, \dots, A_j \overset{?}{\longleftrightarrow} A_{j+1}, \dots, A_{n-i-1} \,|\, A_{n-i} \le \cdots \le A_{n-1}$$

Bubble Sort

89	$\overset{?}{\longleftrightarrow}$	**45**	68	90	29	34	17

Bubble Sort

89	45		68	90	29	34	17
45	**89**	$\overset{?}{\longleftrightarrow}$	**68**	90	29	34	17

Bubble Sort

89	45	68		90	29	34	17
45	89	68		90	29	34	17
45	68	**89**	$\overset{?}{\longleftrightarrow}$	**90**	29	34	17

Bubble Sort

89	45	68	90	29	34	17
45	89	68	90	29	34	17
45	68	89	**90** $\overset{?}{\longleftrightarrow}$	**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**	$\overset{?}{\longleftrightarrow}$	**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** $\overset{?}{\leftrightarrow}$	**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	$\overset{?}{\longleftrightarrow}$	**68**	89	29	34	17	**90**

Bubble Sort

45	**68** $\overset{?}{\longleftrightarrow}$	**89**	29	34	17	**90**

Bubble Sort

45	68	**89**	$\overset{?}{\longleftrightarrow}$	**29**	34	17	**90**

Bubble Sort

45	68	89	29		34	17	**90**
45	68	29	**89**	$\overset{?}{\longleftrightarrow}$	**34**	17	**90**

Bubble Sort

45	68	89	29	34	17	**90**
45	68	29	89	34	17	**90**
45	68	29	34	**89** $\overset{?}{\longleftrightarrow}$	**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**

…

Bubble Sort Algorithm

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

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

//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 $A[j] > A[j+1]$

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

Analysis of Bubble Sort

- **The input size** is the number of elements n in the list.

- **The basic operation** is the key comparison

$$A[j+1] < A[j]$$

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

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

Note: The number of key **swaps**: $S_{worst}(n) = \Theta(n^2)$

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.

$$
\begin{array}{llllllllll}
\textbf{Text } T: & t_0 & \cdots & t_i & \cdots & t_{i+j} & \cdots & t_{i+m-1} & \cdots & t_{n-1} \\
 & & & \updownarrow & & \updownarrow & & \updownarrow & & \\
\textbf{Pattern } P: & & & p_0 & \cdots & p_j & \cdots & p_{m-1} & &
\end{array}
$$

String Matching

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**

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

String Matching: Algorithm

Algorithm BFStringMatch($T[0..n-1], P[0..m-1]$)

// Input: $T[0..n-1]$ & $P[0..m-1]$

//Output: The first index if the search is successful or -1 if it is not successful

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

String Matching: Analysis

- In the **worst case**:
  - m **comparisons** before each shifting
  - $(n - m + 1)$ **shifting**
  - the total number of character comparisons:

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

  - the **time efficiency**: $O(mn)$
- There are algorithms with $O(n)$ for searching in **random texts**.

Closest-Pair Problem

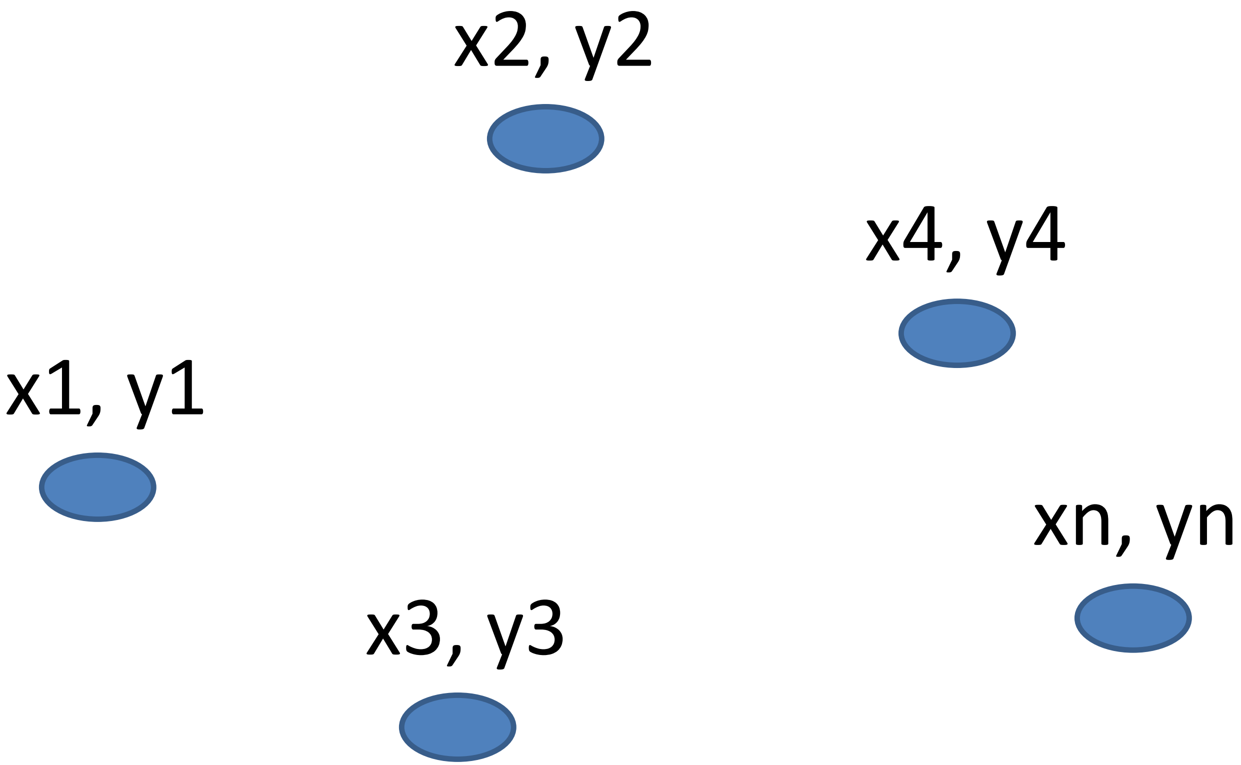
Problem: Find the **two closest points** in a set of n points

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

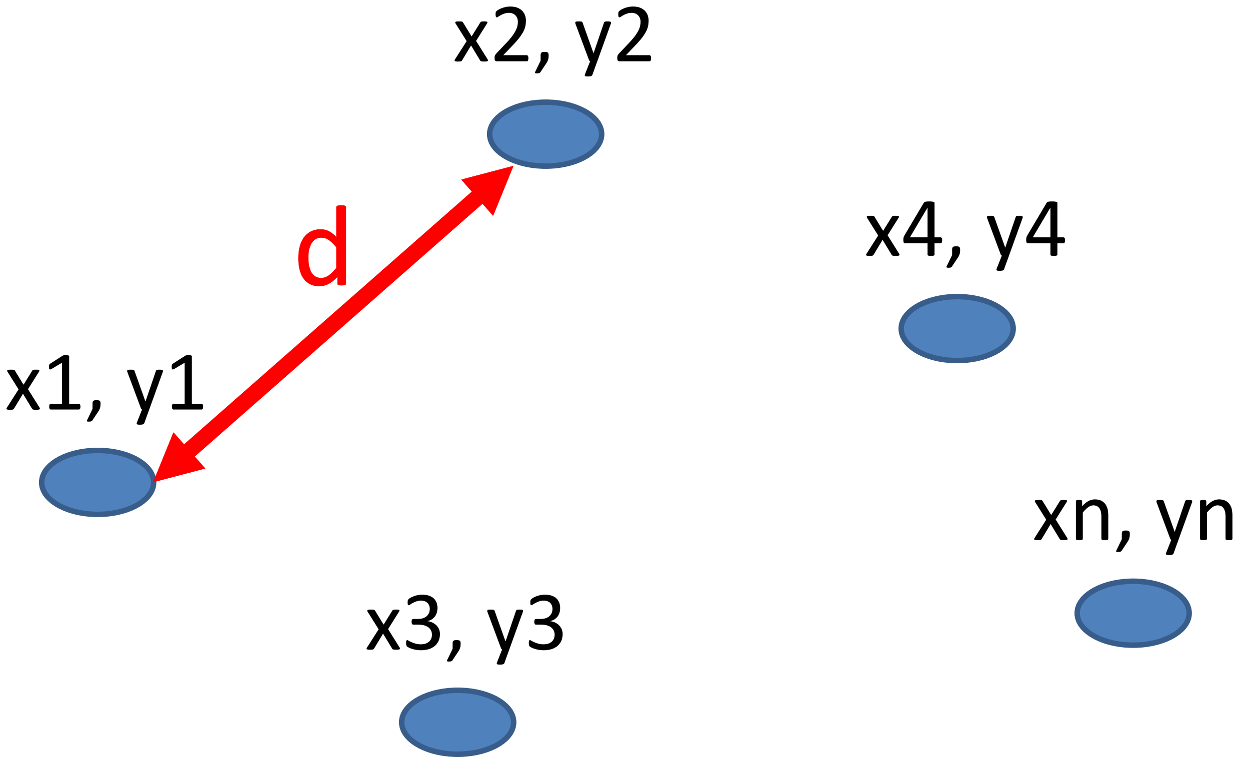
Two-dimensional case: the distance between two points $p_i(x_i, y_i)$ and $p_j(x_j, y_j)$:

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

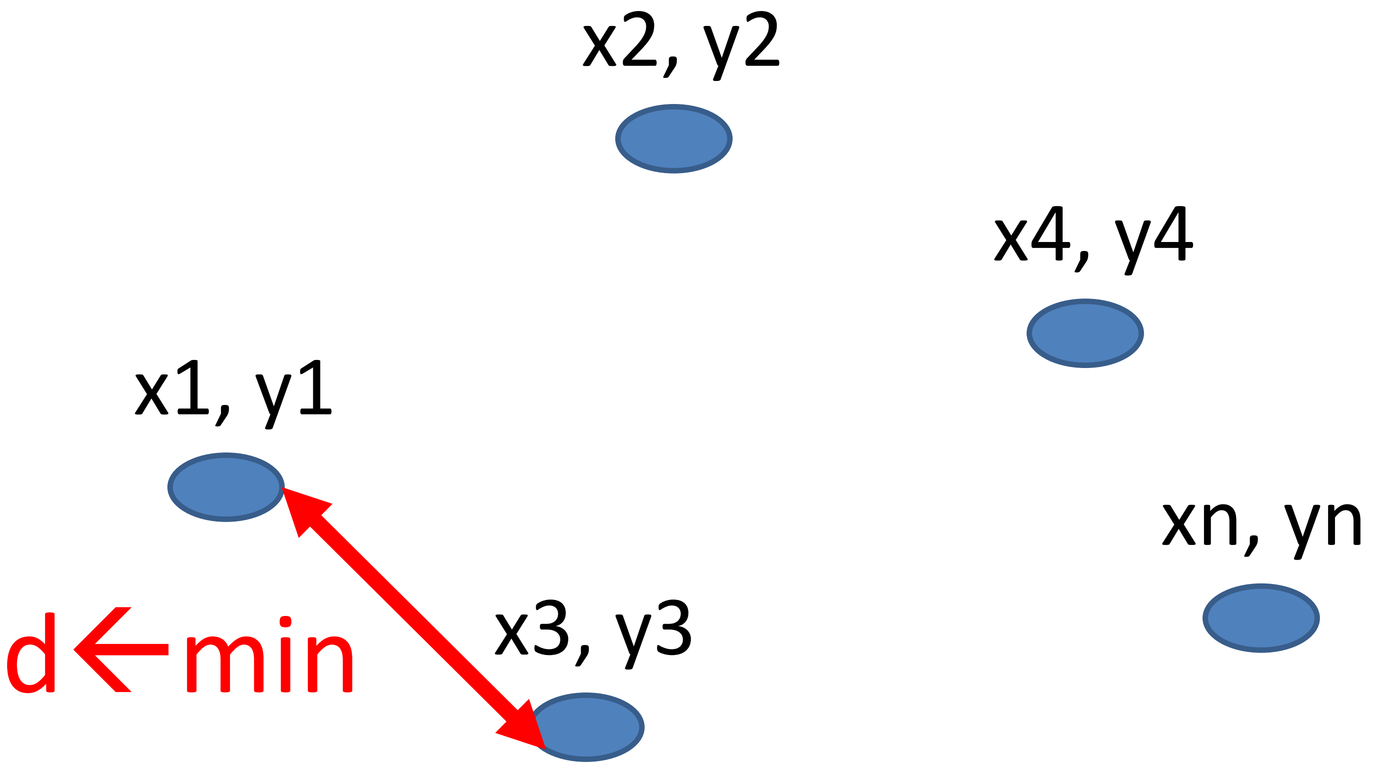
Closest-Pair Problem



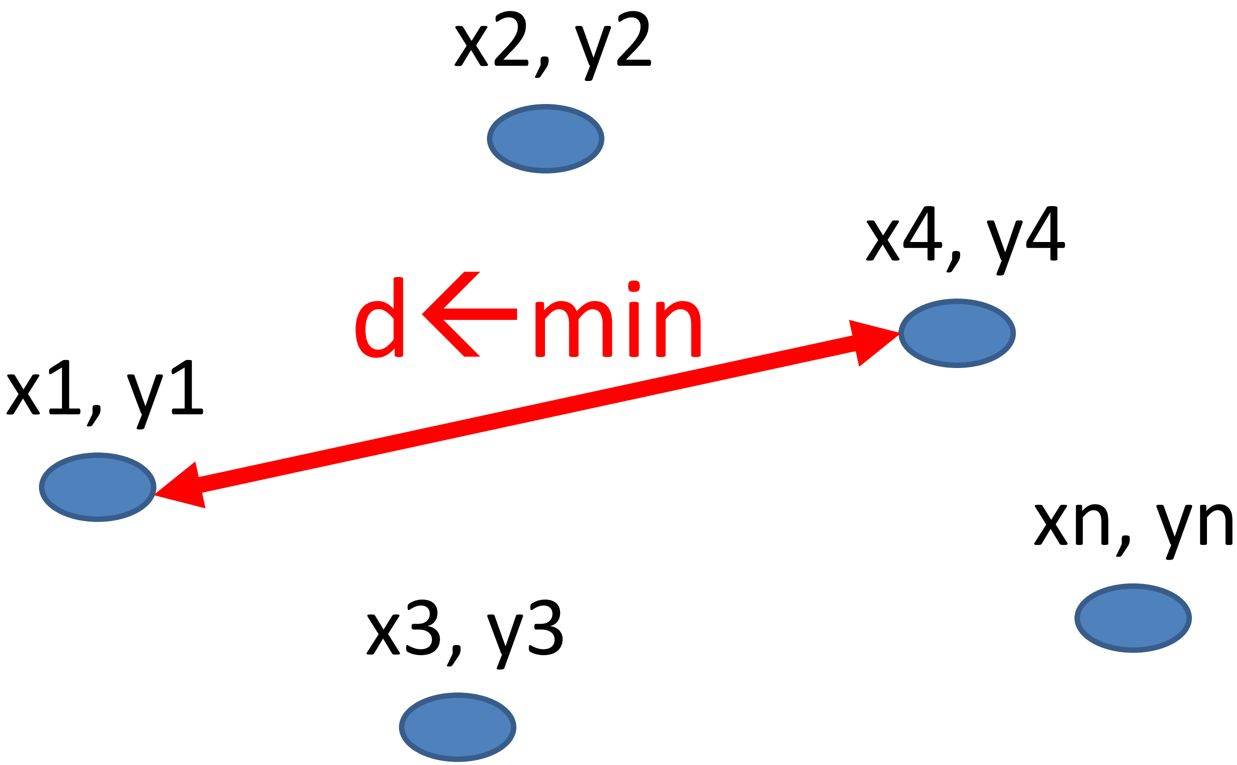
Closest-Pair Problem



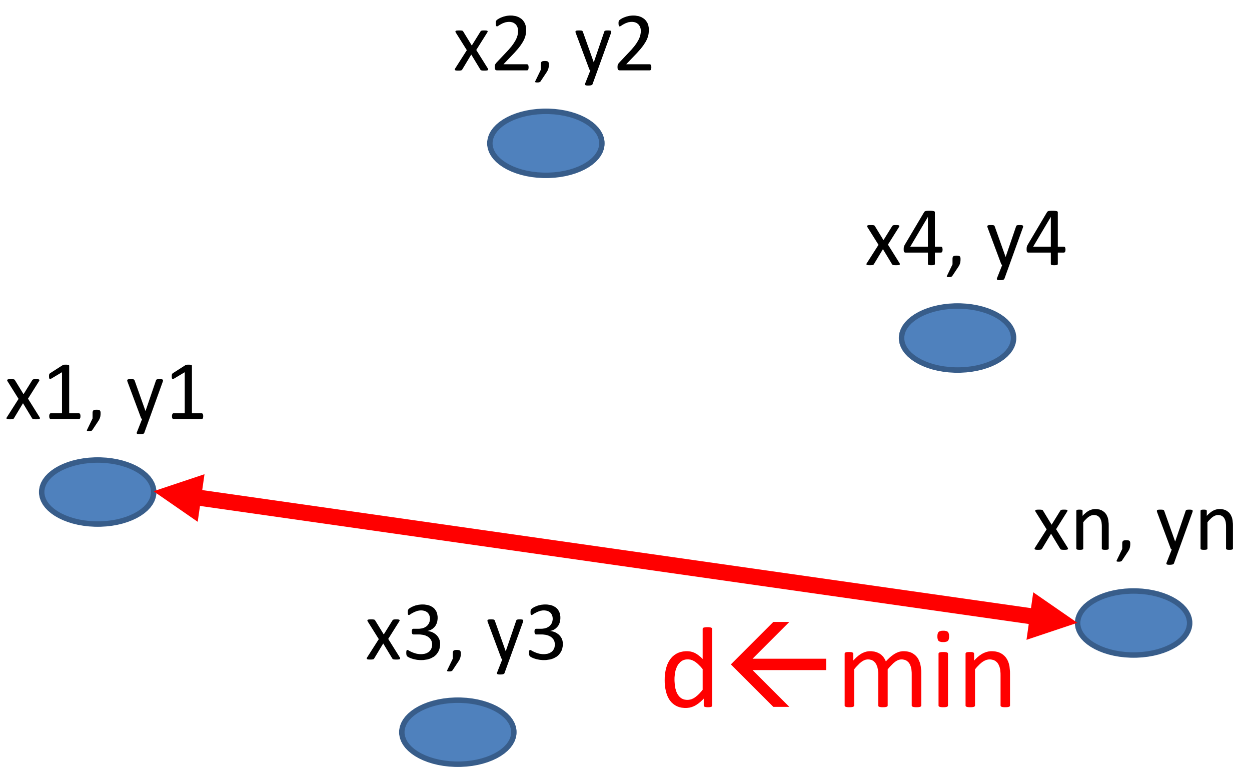
Closest-Pair Problem



Closest-Pair Problem



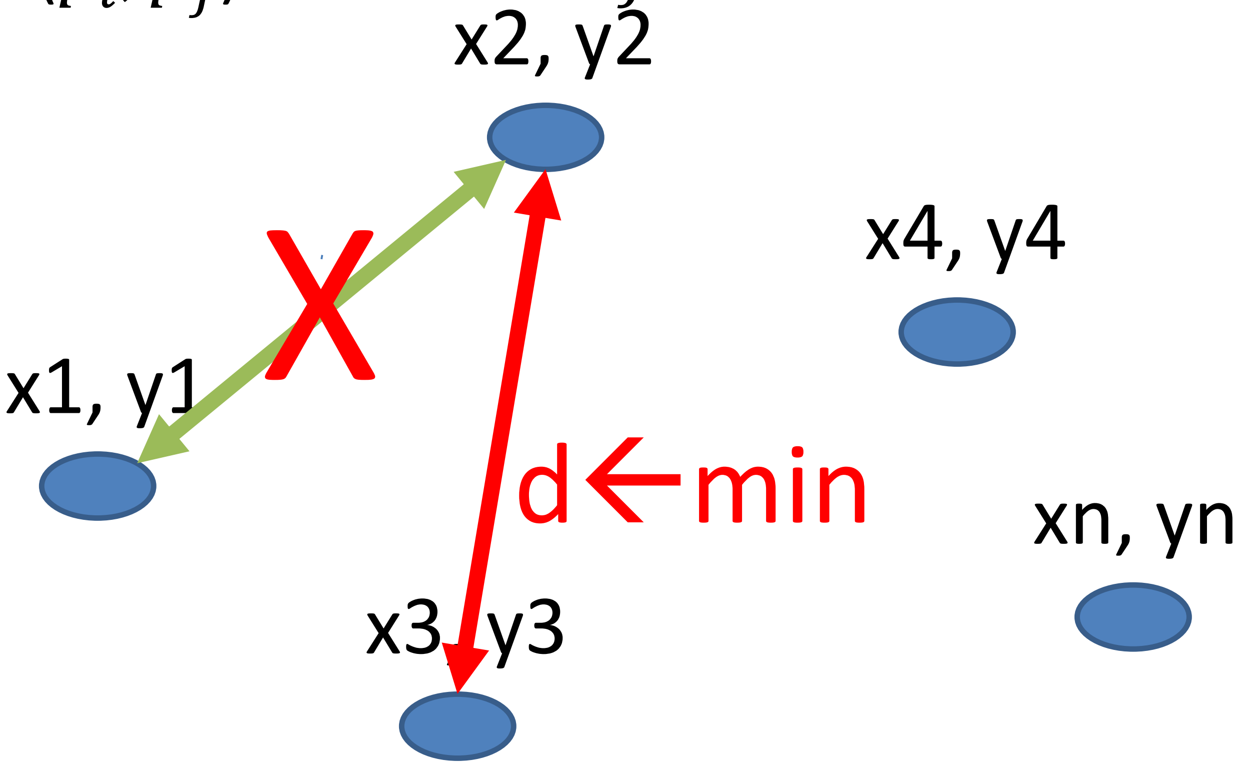
Closest-Pair Problem



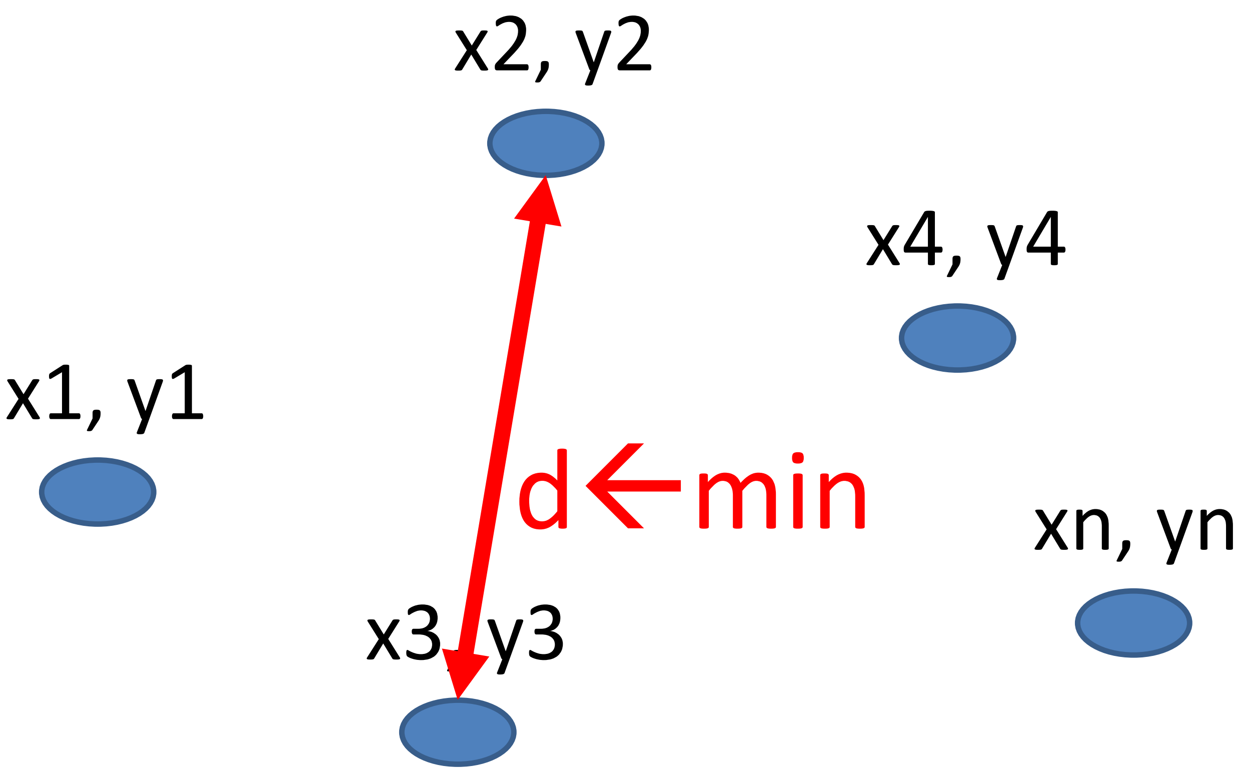
Closest-Pair Problem

Don't want to calculate the same distance:

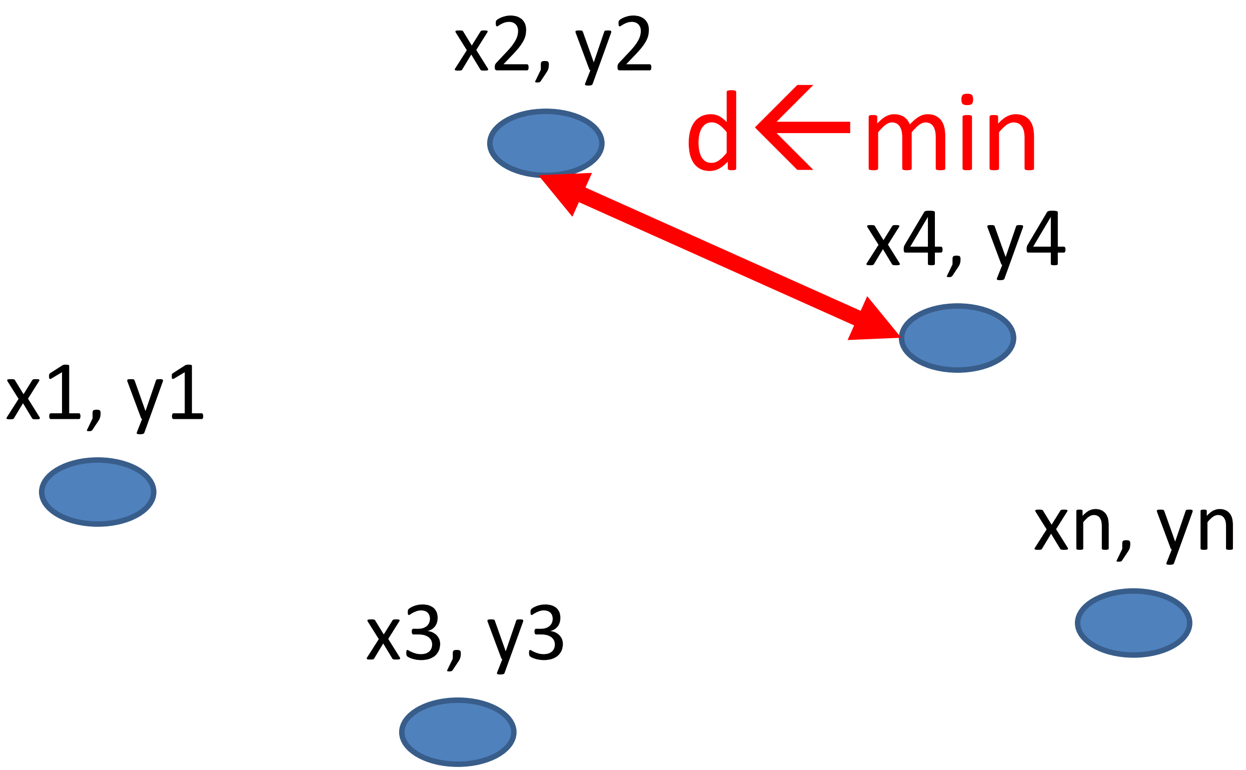
Consider (p_i, p_j) where $i < j$



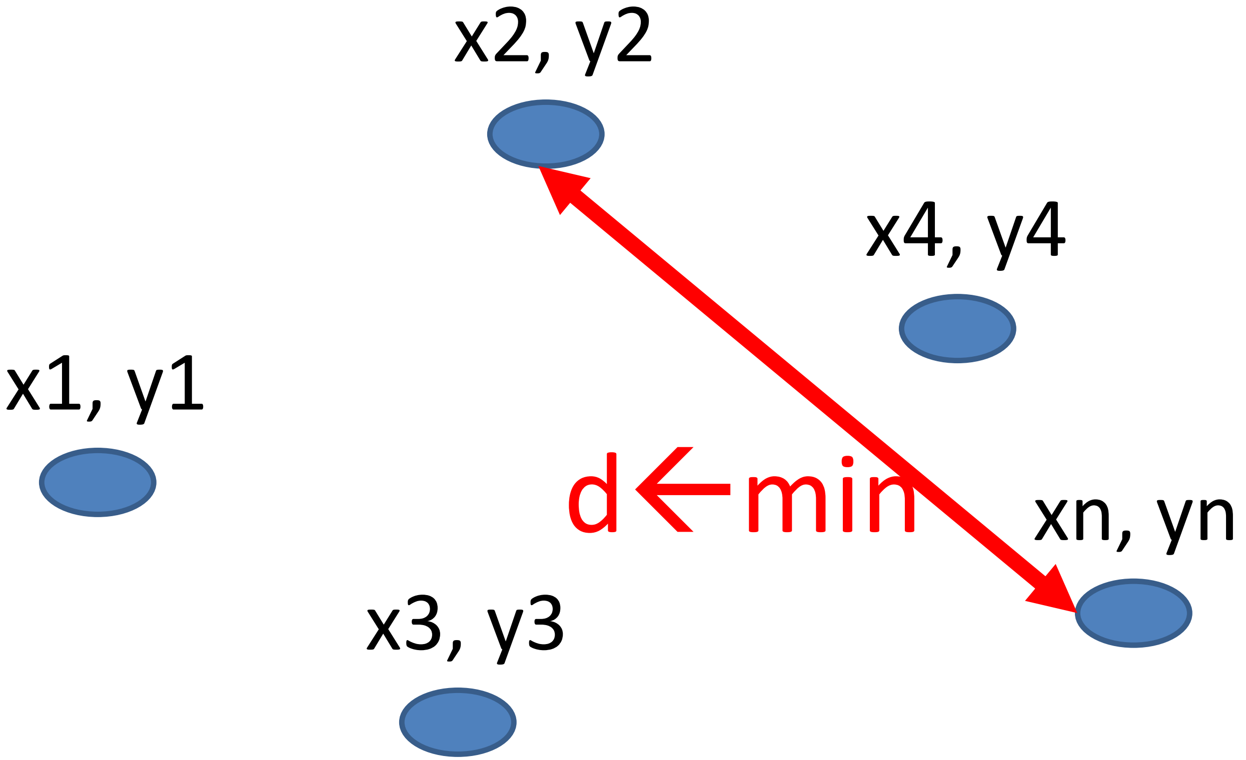
Closest-Pair Problem



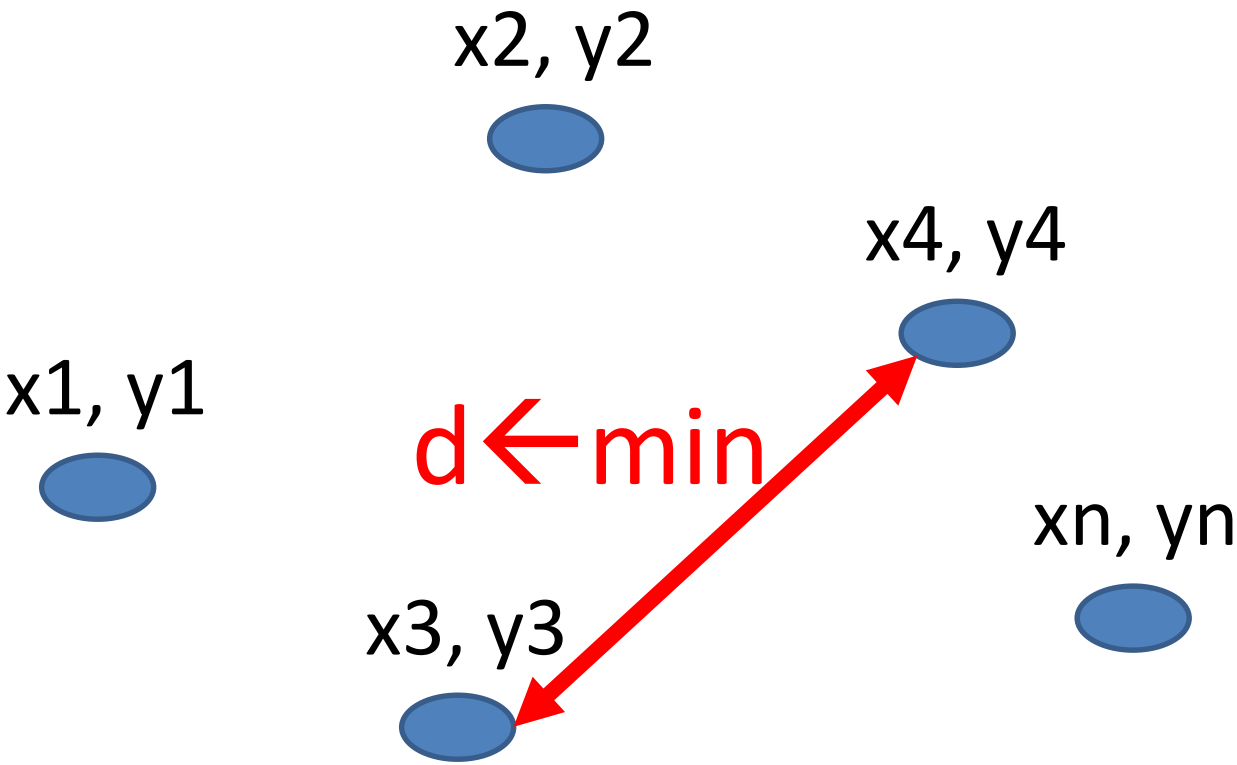
Closest-Pair Problem



Closest-Pair Problem



Closest-Pair Problem



Closest-Pair Problem

Algorithm BFClosestPair(P)

// Input: A list P of $n \geq 2$ points

//Output: The d between the closest pair of points

$d \leftarrow \infty$

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

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

$$d \leftarrow \min\left(d, \text{sqrt}\left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)\right)$$

return d

Closest-Pair Problem

Algorithm BFClosestPair(P)

// Input: A list P of $n \geq 2$ points

//Output: The d between the closest pair of points

$d \leftarrow \infty$

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

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

$$d \leftarrow \min\left(d, \left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2\right)\right)$$

return d

Closest-Pair Problem

- The **basic operation** is computing sqrt, but we can avoid computing sqrt and compare the values

$$\left((x_i - x_j)^2 + (y_i - y_j)^2\right)$$

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

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

$$= (n-1)n \approx n^2 \in \Theta(n^2)$$

Activity (in group)

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

https://padlet.com/sem2_2018_2019/dsa2

EXHAUSTIVE SEARCH ALGORITHMS