Divide and Conquer

Algorithm

2014 Fall Semester

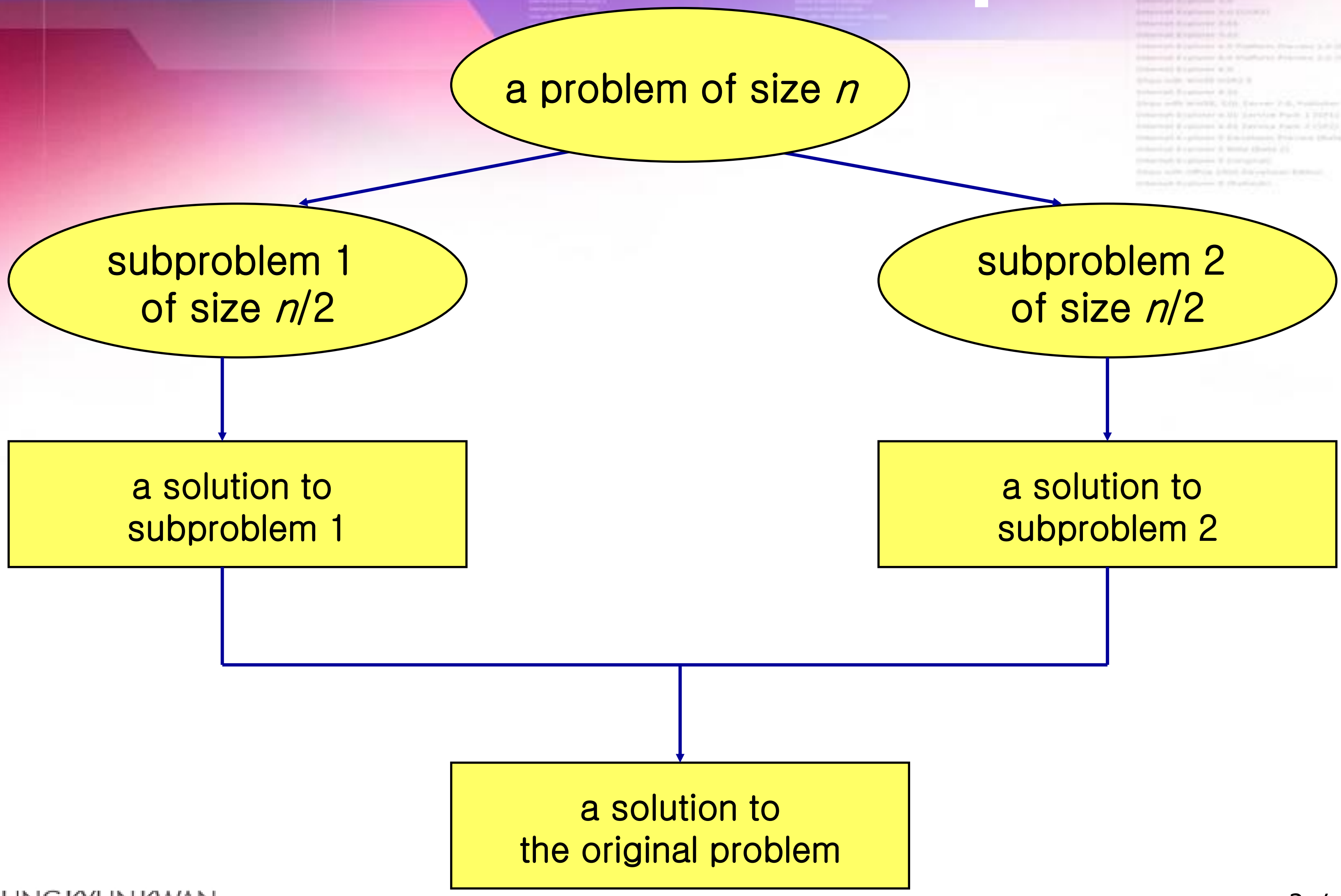
SUNG KYUN KWAN UNIVERSITY

Divide-and-Conquer

The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances
2. Solve smaller instances recursively
3. Obtain solution to original (larger) instance by combining these solutions

Divide-and-Conquer



SUNG KYUN KWAN
UNIVERSITY

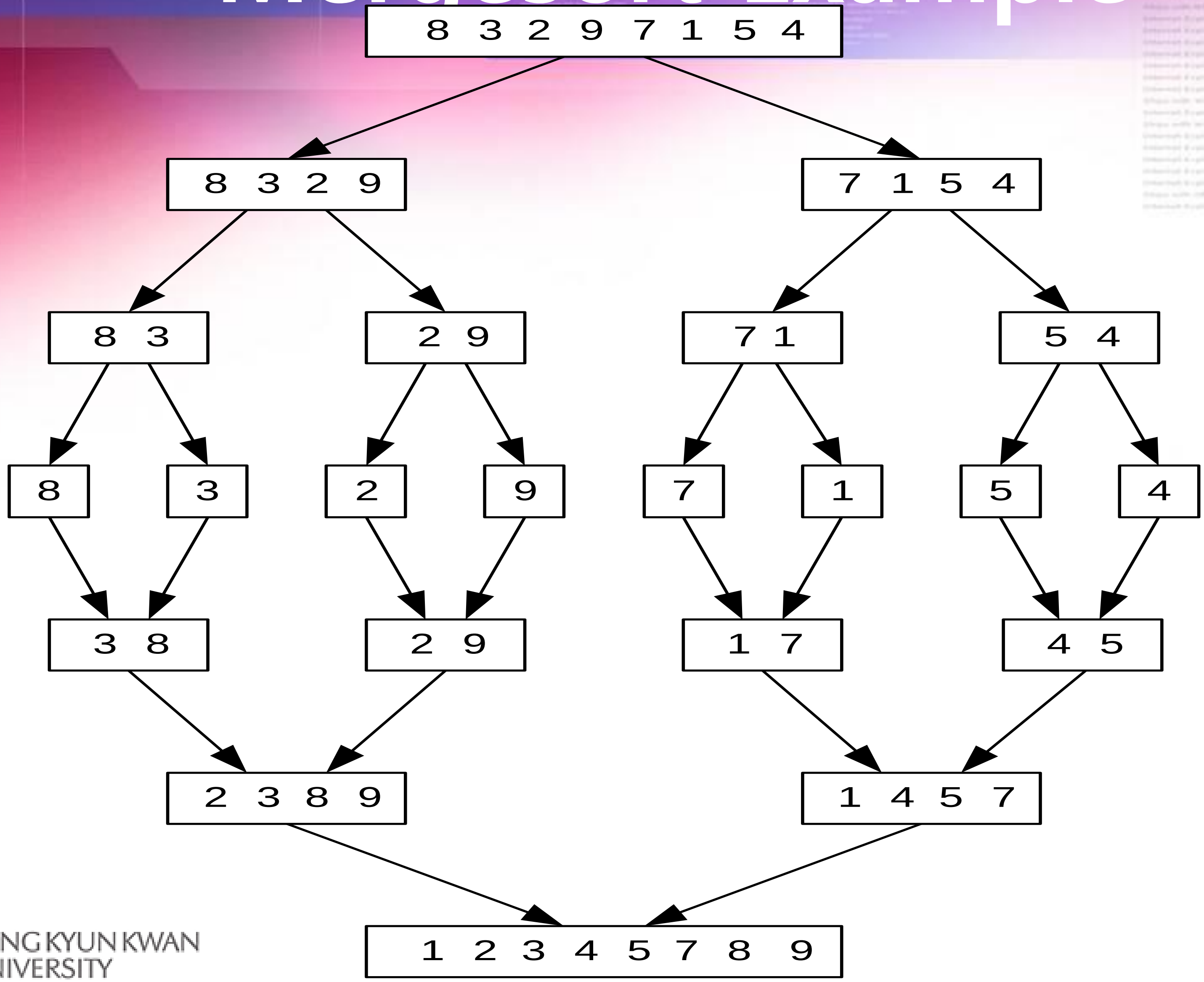
Divide-and-Conquer Examples

- Sorting: merge sort and quick sort
- Binary search
- Multiplication of large integers
- Matrix multiplication: Strassen's algorithm
- Closest-pair algorithms
- Convex-hull algorithms

Mergesort

- Split array A[0..n-1] in two about equal halves and make copies of each half in arrays B and C
- Sort arrays B and C recursively
- Merge sorted arrays B and C into array A as follows:
  - Repeat the following until no elements remain in one of the arrays:
    - compare the first elements in the remaining unprocessed portions of the arrays
    - copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
  - Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

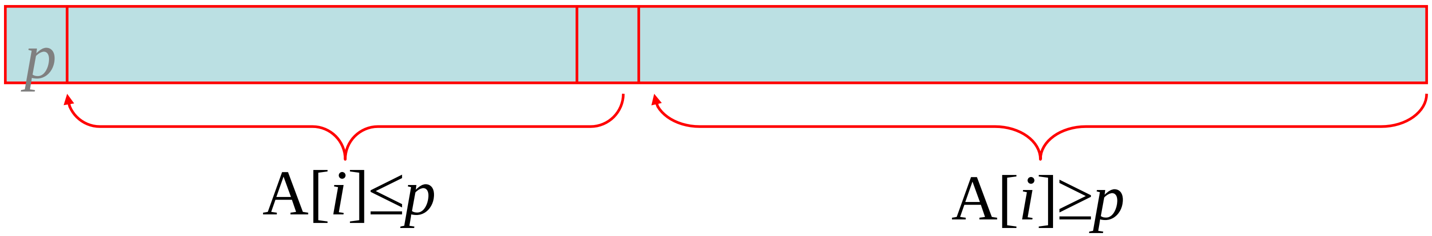
Mergesort Example



SUNG KYUN KWAN UNIVERSITY

Quicksort

- Select a *pivot* (partitioning element) – here, the first element
- Rearrange the list so that all the elements in the first *s* positions are smaller than or equal to the pivot and all the elements in the remaining *n-s* positions are larger than or equal to the pivot (see next slide for an example)



- Exchange the pivot with the last element in the first (i.e., ≤) subarray — the pivot is now in its final position
- Sort the two subarrays recursively

Quicksort Example

1. 5 - 3 - 7 - 6 - 2 - 1 - 4
   p
2. 5 - 3 - 7 - 6 - 2 - 1 - 4
   i j p
3. 1 - 3 - 7 - 6 - 2 - 5 - 4
   i j p
4. 1 - 3 - 7 - 6 - 2 - 5 - 4
   i j p
5. 1 - 3 - 7 - 6 - 2 - 5 - 4
   i j p
6. 1 - 3 - 2 - 6 - 7 - 5 - 4
   i j p
7. 1 - 3 - 2 - 6 - 7 - 5 - 4
   p
8. 1 - 3 - 2 - 4 - 7 - 5 - 6
   p

Sort sub-list : 1 - 3 - 2

1 - 2 - 3

Final Result :

1 - 2 - 3 - 4 - 5 - 6 - 7

In-Class Exercise

- Quicksort

24 32 11 15 62 3 9 13 22 5 10

Binary Search

Very efficient algorithm for searching in sorted array:

K

vs

A[0] . . . A[m] . . . A[n-1]

If K = A[m], stop (successful search); otherwise, continue searching by the same method in A[0..m-1] if K < A[m] and in A[m+1..n-1] if K > A[m]

$l \leftarrow 0$; $r \leftarrow n-1$
while $l \leq r$ do
$m \leftarrow \lfloor (l+r)/2 \rfloor$
if K = A[m] return m
else if K < A[m] $r \leftarrow m-1$
else $l \leftarrow m+1$
return -1

SUNGKYUNKWAN UNIVERSITY

Binary Search Example

Sorted list :

1 7 14 17 26 59 63 77 79 87 88 90 92 96 98 99

Key value : 63

Step 1) M=(0+15) div 2

A[7] = 77

Step 2) Is the number greater than 77? (No)

M=(0+6) div 2

A[3] = 17

Step 3) Is the number greater than 17? (Yes)

.....

Multiplication of Large Integers

Consider the problem of multiplying two (large) *n*-digit integers represented by arrays of their digits such as:

A = 12525678901357986429

B = 87654321284820912836

Divide & Conquer Algorithm

A small example: $A * B$ where $A = 2135$ and $B = 4014$

$A = (21 \cdot 10^2 + 35)$, $B = (40 \cdot 10^2 + 14)$

So, $A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$

$= 21 * 40 \cdot 10^4 +$

$(21 * 14 + 35 * 40) \cdot 10^2 +$

$35 * 14$

In general, if $A = A_1A_2$ and $B = B_1B_2$

(where A and B are *n*-digit, A_1, A_2, B_1, B_2 are *n/2*-digit numbers),

$$\mathbf{A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2}$$

SUNG KYUN KWAN UNIVERSITY

Divide & Conquer Algorithm

Let $P_1 = (I_h + I_l) \times (J_h + J_l) = I_h x J_h + I_h x J_l + I_l x J_h + I_l x J_l$

$P_2 = I_h \times J_h$, and

$P_3 = I_l \times J_l$

Now, note that

$$P_1 - P_2 - P_3 = I_h x J_h + I_h x J_l + I_l x J_h + I_l x J_l - I_h x J_h - I_l x J_l$$
$$= I_h x J_l + I_l x J_h$$

Then we have the following:

$$I \times J = P_2 \times 2^n + [P_1 - P_2 - P_3] x\, 2^{n/2} + P_3.$$

Divide & Conquer exercise

- Multiplication of Large Integers

11010011 * 01011001

SUNG KYUN KWAN UNIVERSITY

Let I = 11010011, which is 211 in decimal

Let J = 01011001, which is 89 in decimal.

Then we have I_h = 1101, which is 13 in decimal, and

I_l = 0011, which is 3 in decimal

Also we have J_h = 0101, which is 5 in decimal, and

J_l = 1001, which is 9 in decimal

1) Compute $I_h + I_l$ = 10000, which is 16 in decimal
2) Compute $J_h + J_l$ = 1110, which is 14 in decimal
3) Recursively multiply $(I_h + I_l) \times (J_h + J_l)$, giving us 11100000, which is 224 in decimal. (This is P_1.)
4) Recursively mutliply $I_h \times J_h$, giving us 01000001, which is 65 in decimal. (This is P_2.)
5) Recursively multiply $I_l \times J_l$, giving us 00011011, which is 27 in decimal. (This is P_3.)
6) Compute $P_1 - P_2 – P_3$ using 2 subtractions to yield 10000100, which is 132 in decimal
7) Now compute the product as 01000001x100000000 +

10000100x 00010000 +

00011011 =

```
   0100000100000000  (P2x2^8)
       100001000000  ((P1- P2- P3) x2^4)
+          00011011  (P3)
--------------------------------
   0100100101011011, which is 18779 in decimal, the correct answer.
```

(This is also $65 \times 2^8 + 132 \times 2^4 + 27$.)

SUNG KYUN KWAN UNIVERSITY

Strassen's Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed as follows:

$$\begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} * \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

$$\mathbf{C_{00} = A_{00} B_{00} + A_{01} B_{10}}$$
$$\mathbf{C_{01} = A_{00} B_{01} + A_{01} B_{11}}$$
$$\mathbf{C_{10} = A_{10} B_{00} + A_{11} B_{10}}$$
$$\mathbf{C_{11} = A_{10} B_{01} + A_{11} B_{11}}$$

$$= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

2x2 matrix multiplication can be accomplished in 8 multiplication.$\mathbf{(2^{\log_2 8} = 2^3)}$

SUNG KYUN KWAN UNIVERSITY

Basic Matrix Multiplication

- Algorithm

```
void matrix_mult ()
{
  for (i = 1; i <= N; i++)
        for (j = 1; j <= N; j++)
              compute Ci,j;
 }
```

- Time analysis

$$C_{i,j} = \sum_{k=1}^{N} a_{i,k} b_{k,j}$$

$$\text{Thus } T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} c = cN^3 = O(N^3)$$

Strassens's Matrix Multiplication

- Strassen showed that 2x2 matrix multiplication can be accomplished in 7 multiplication and 18 additions or subtractions.

- $(2^{\log_2 7} = 2^{2.807})$

- This reduce can be done by Divide and Conquer Approach.

Formulas for Strassen's Algorithm

$M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$

$M_2 = (A_{10} + A_{11}) * B_{00}$

$M_3 = A_{00} * (B_{01} - B_{11})$

$M_4 = A_{11} * (B_{10} - B_{00})$

$M_5 = (A_{00} + A_{01}) * B_{11}$

$M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$

$M_7 = (A_{01} - A_{11}) * (B_{10} + B_{11})$

$C_{00} = M_1 + M_4 - M_5 + M_7$

$C_{01} = M_3 + M_5$

$C_{10} = M_2 + M_4$

$C_{11} = M_1 + M_3 - M_2 + M_6$

Closest-Pair Problem: Divide and Conquer

- Brute force approach requires comparing every point with every other point
- Given n points, we must perform 1 + 2 + 3 + ⋯ + n−2 + n−1 comparisons.

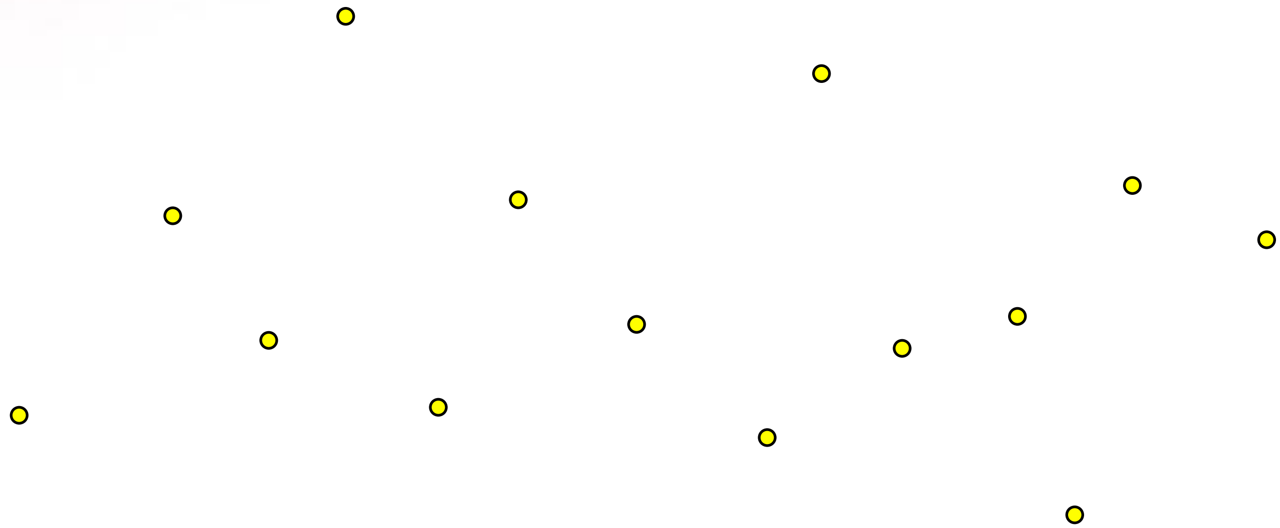
$$\sum_{k=1}^{n-1} k = \frac{(n-1)\cdot n}{2}$$

- Brute force → $O(n^2)$
- The Divide and Conquer algorithm yields → $O(n \log n)$
- Reminder: if n = 1,000,000 then
  - n^2 = 1,000,000,000,000 whereas
  - n log n = 20,000,000

SUNG KYUN KWAN UNIVERSITY

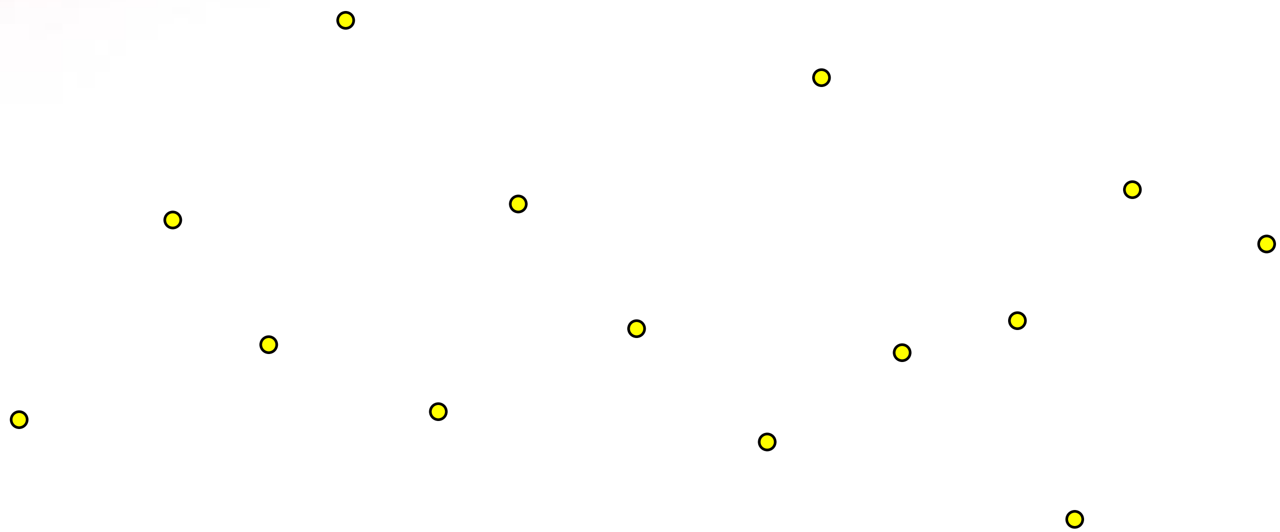
Closest-Pair Algorithm

Given: A set of points in 2-D



Closest-Pair Algorithm

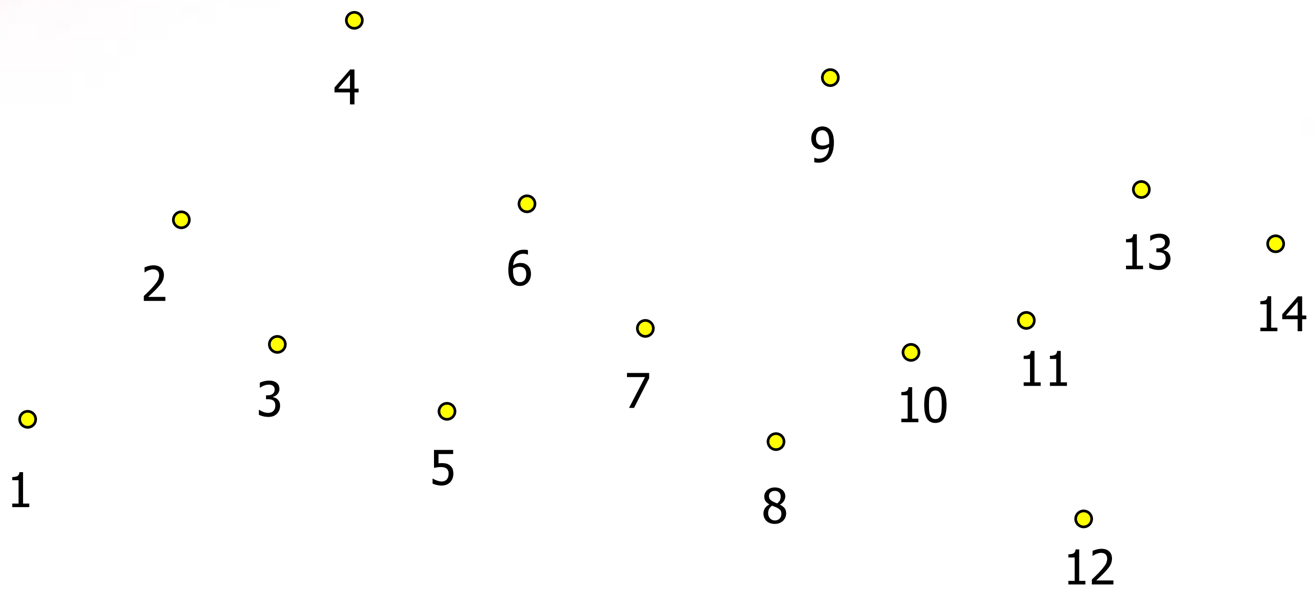
Step 1: Sort the points in one D



Closest-Pair Algorithm

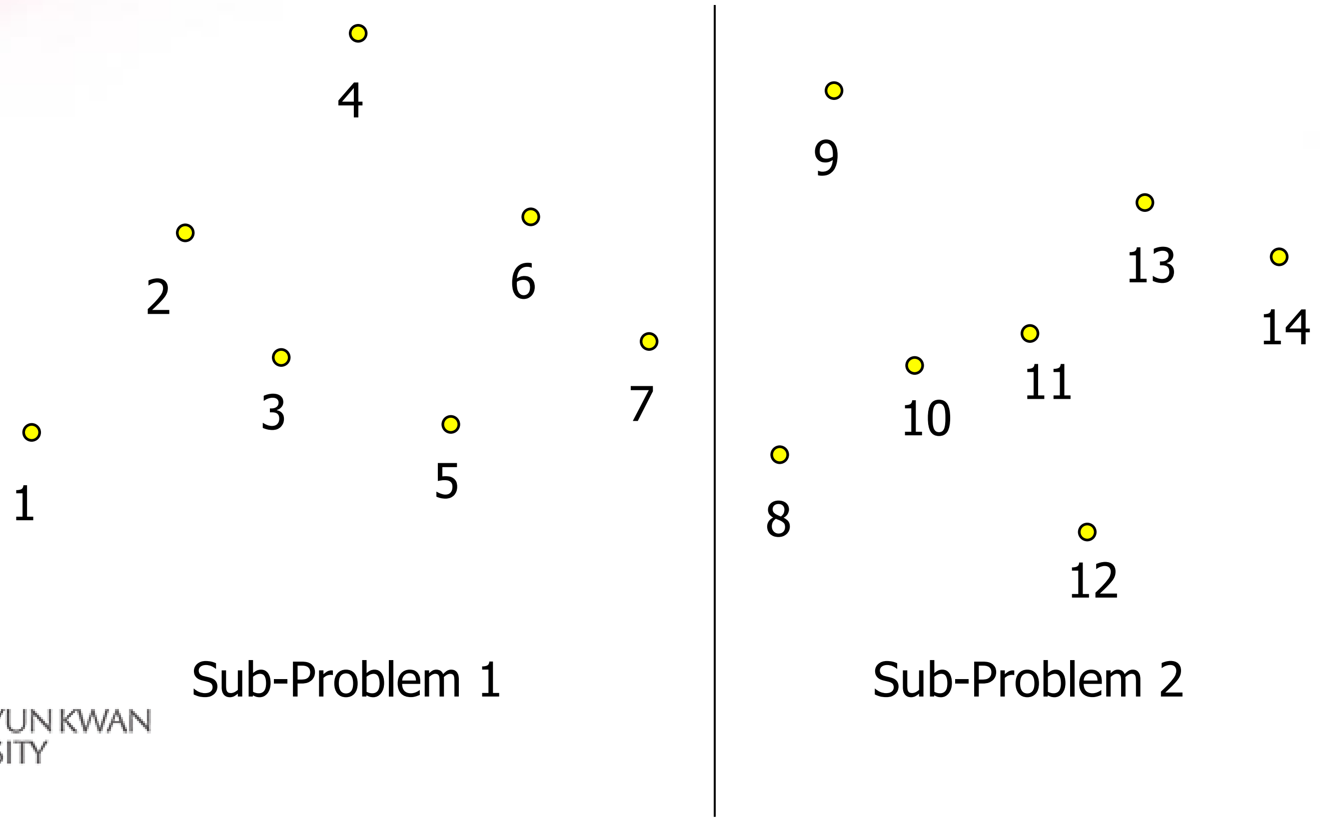
Lets sort based on the X-axis

O(n log n) using quicksort or mergesort



Closest-Pair Algorithm

Step 2: Split the points, i.e.,
Draw a line at the mid-point between 7 and 8

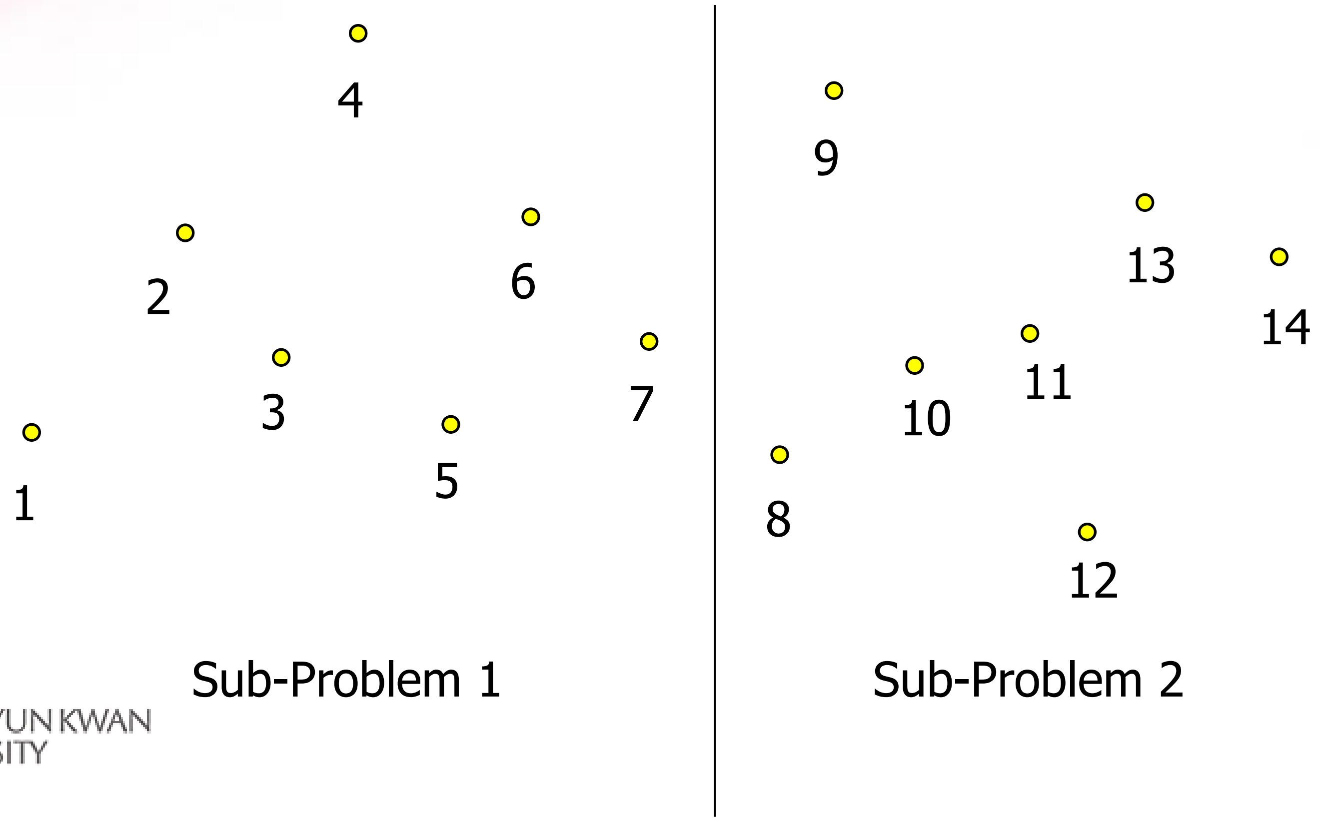


SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

Advantage: Normally, we'd have to compare each of the 14 points with every other point.

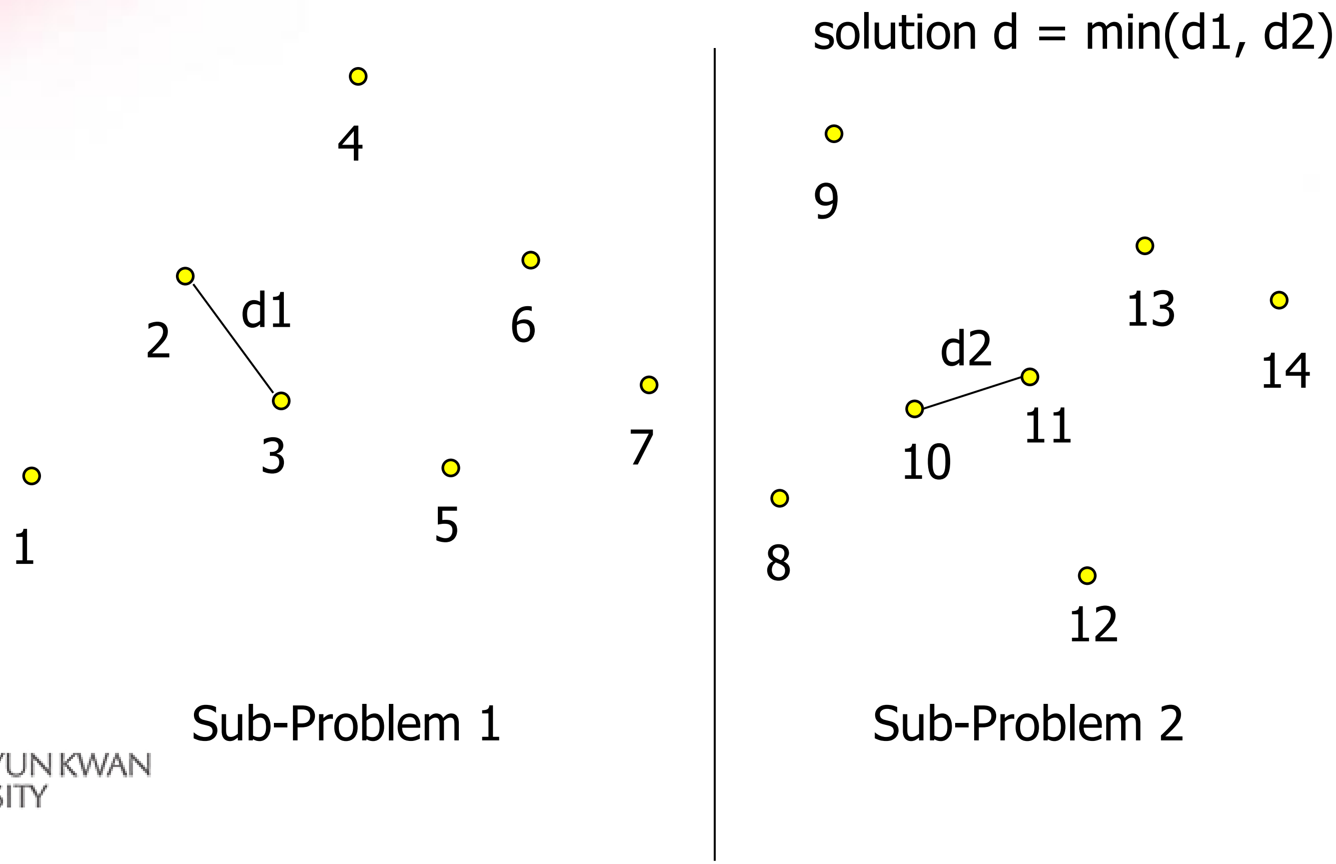
(n-1)n/2 = 13*14/2 = **91 comparisons**



SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

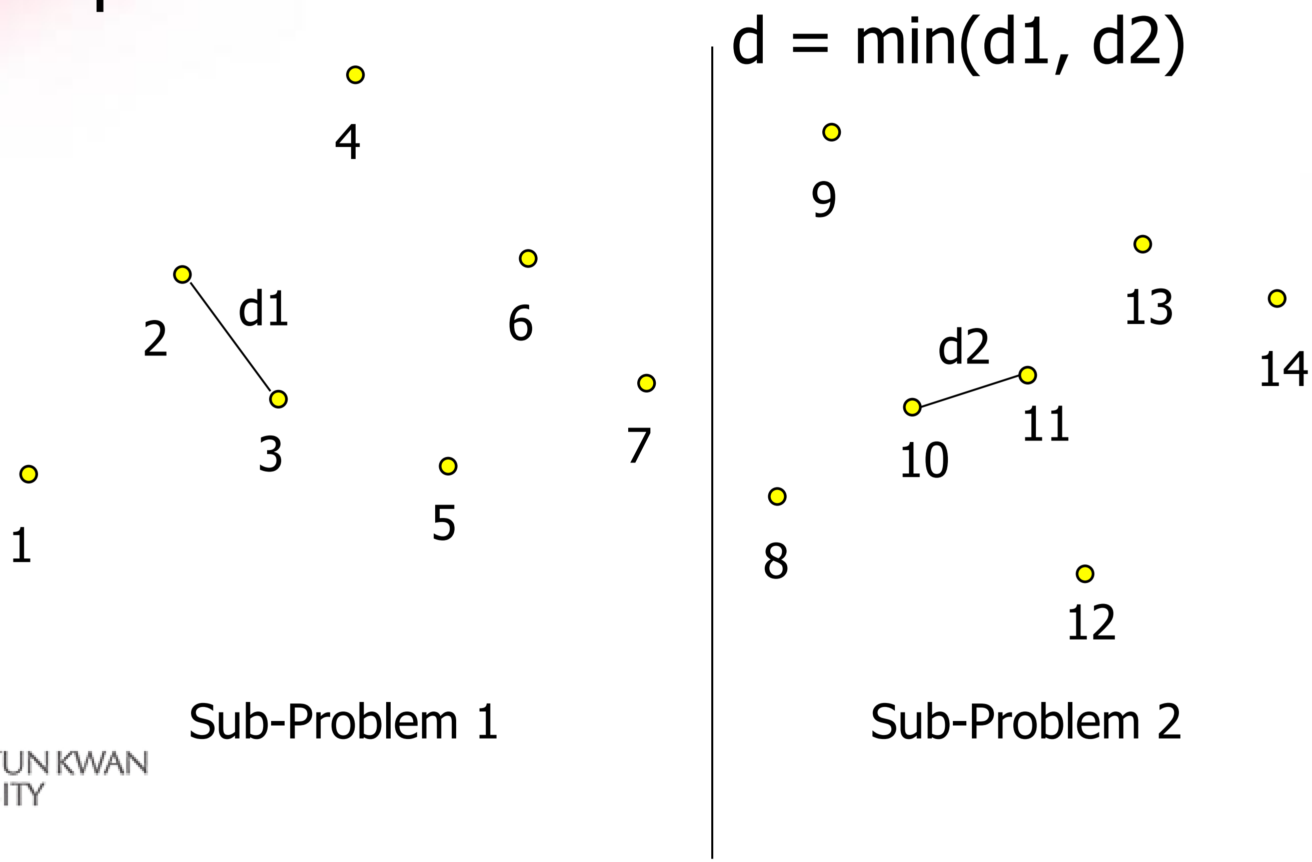
Advantage: Now, we have two sub-problems of half the size. Thus, we have to do 6*7/2 comparisons twice, which is 42 comparisons



SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

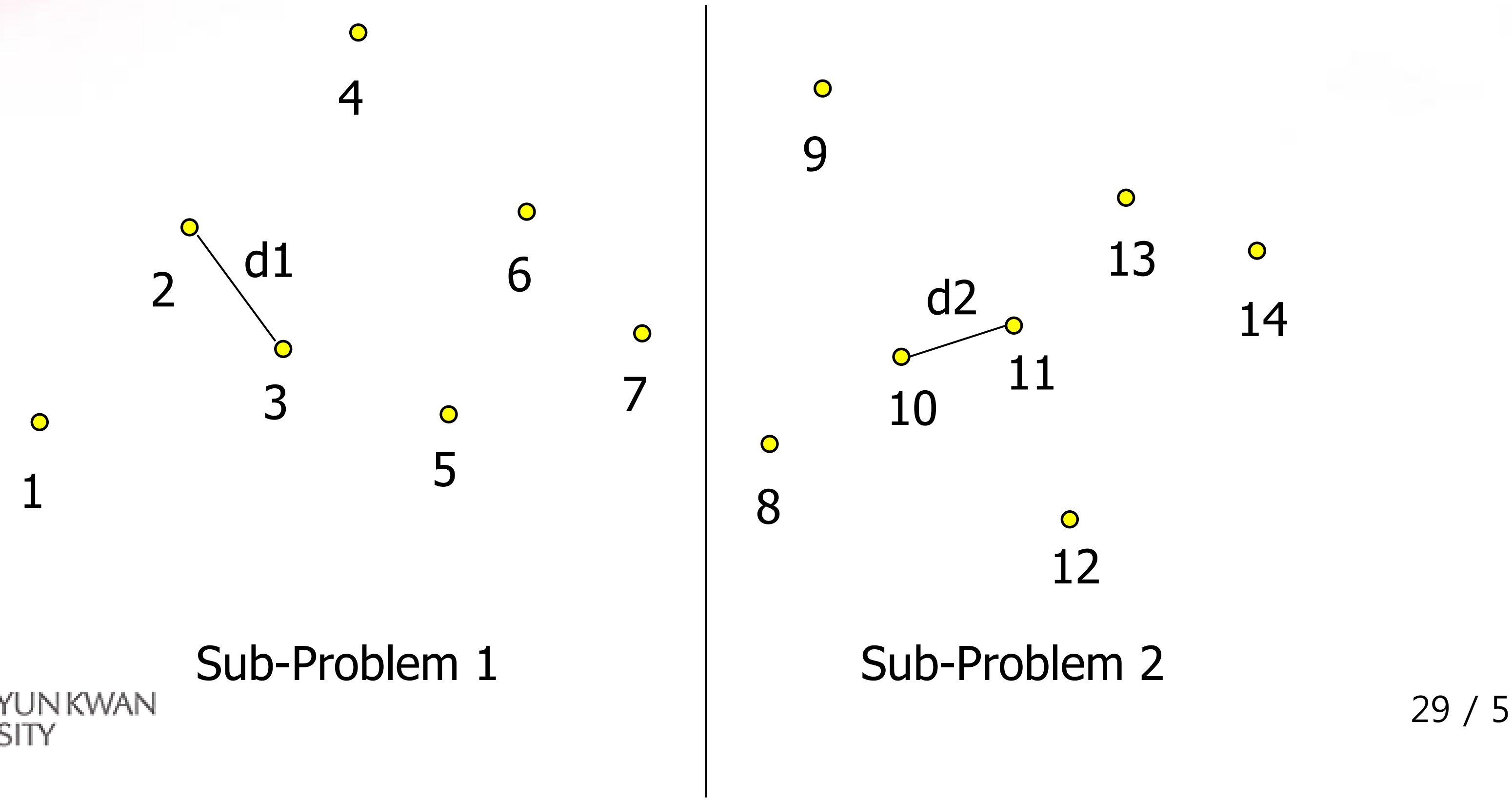
Advantage: With just one split we cut the number of comparisons in half. Obviously, we gain an even greater advantage if we split the sub-problems.



SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

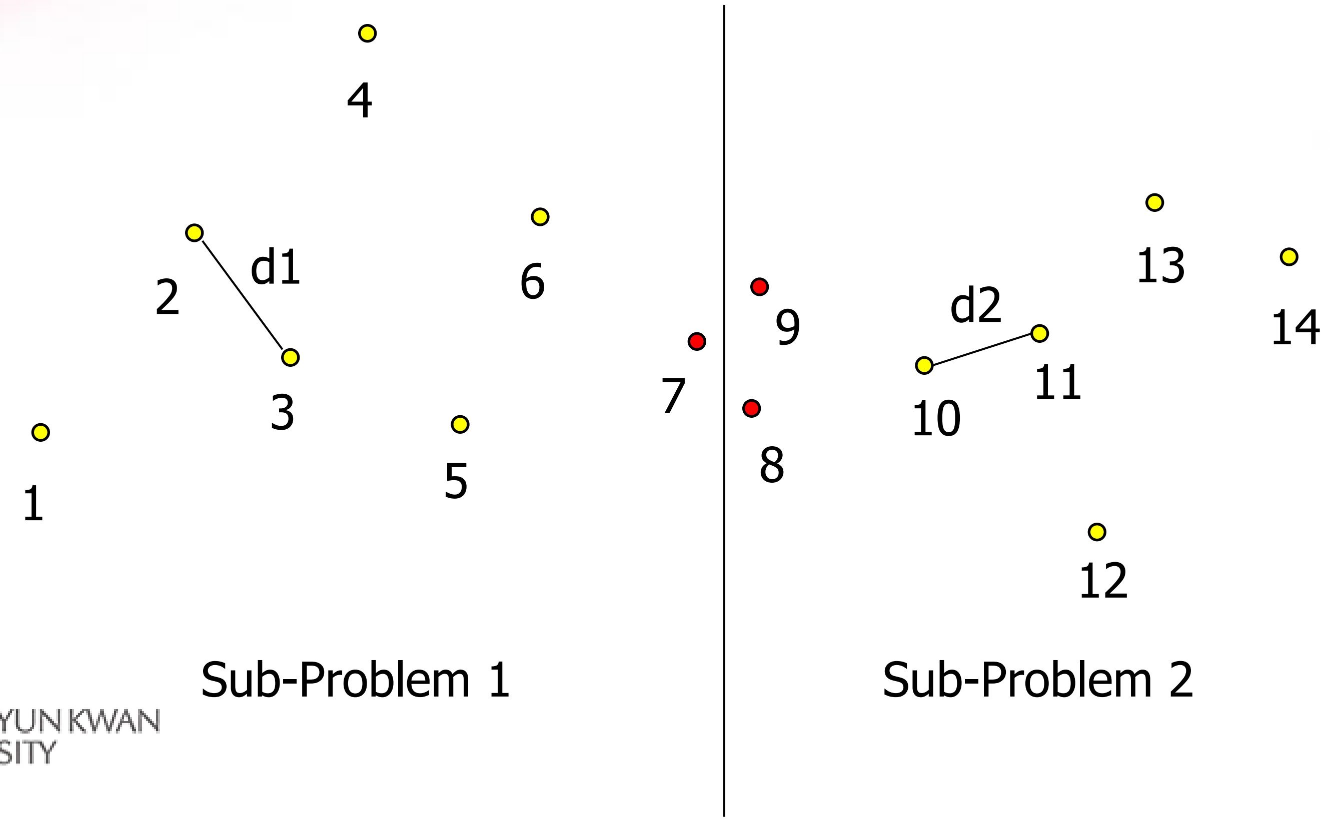
Problem: However, what if the closest two points are each from different sub-problems?



SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

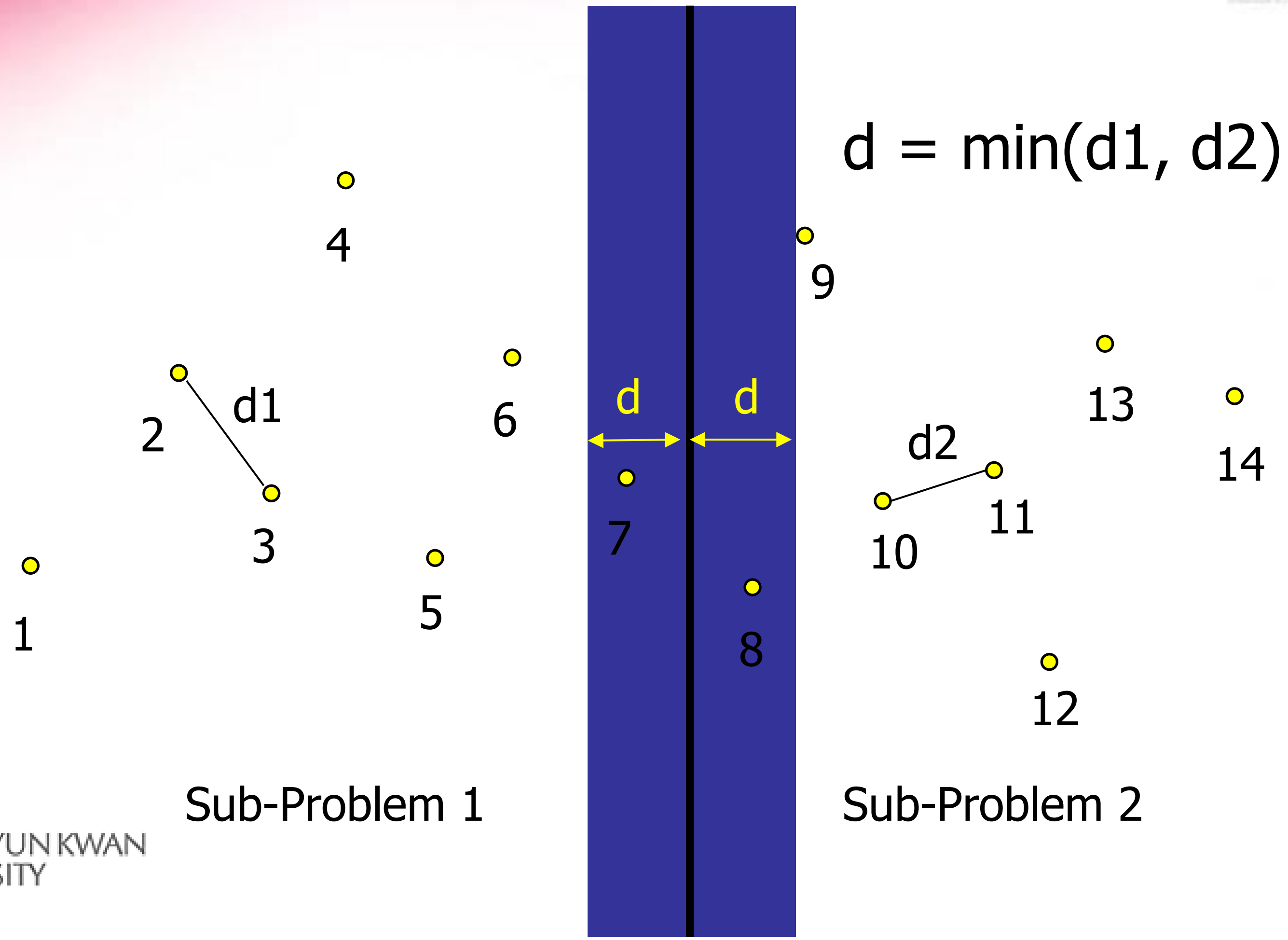
Here is an example where we have to compare points from sub-problem 1 to the points in sub-problem 2.



SUNG KYUN KWAN UNIVERSITY

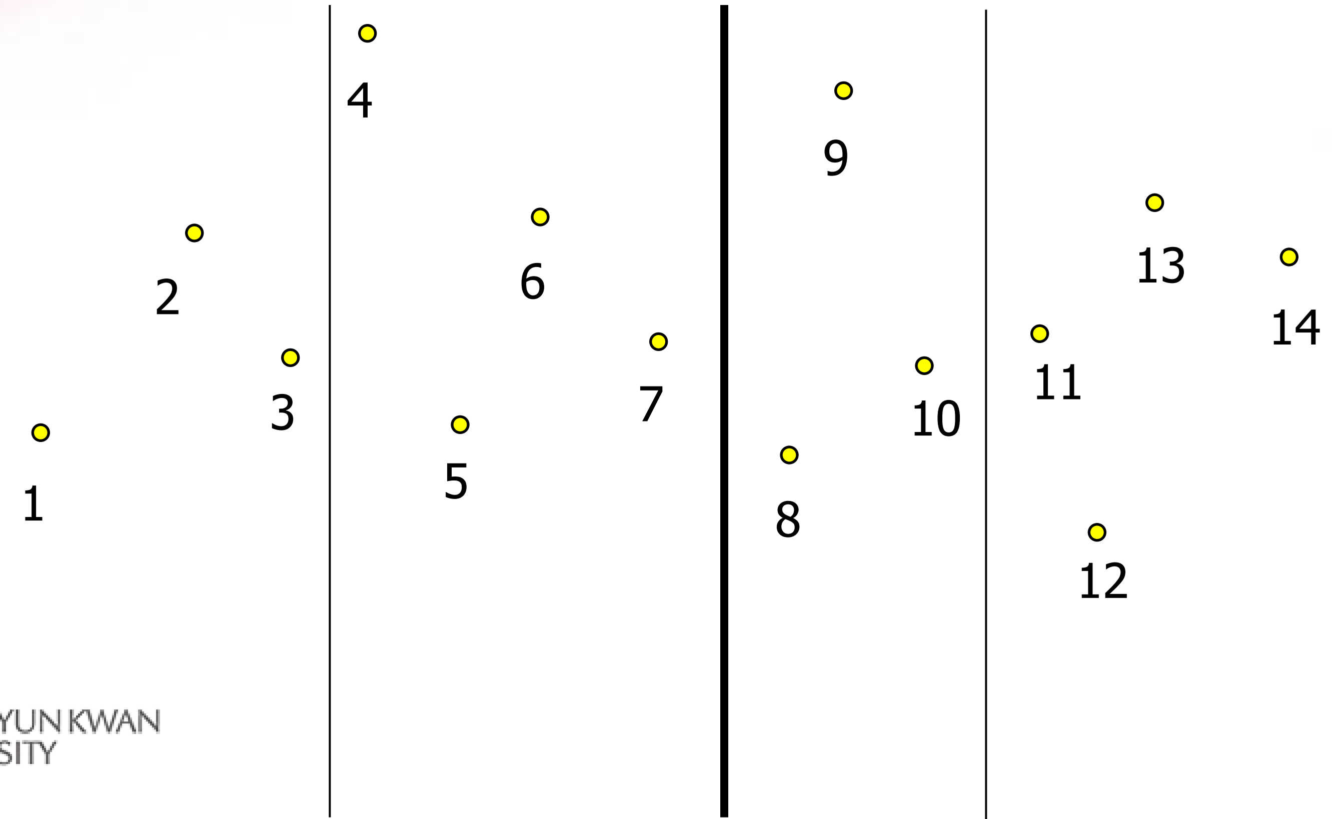
Closest-Pair Algorithm

However, we only have to compare points inside the following "strip."



Closest-Pair Algorithm

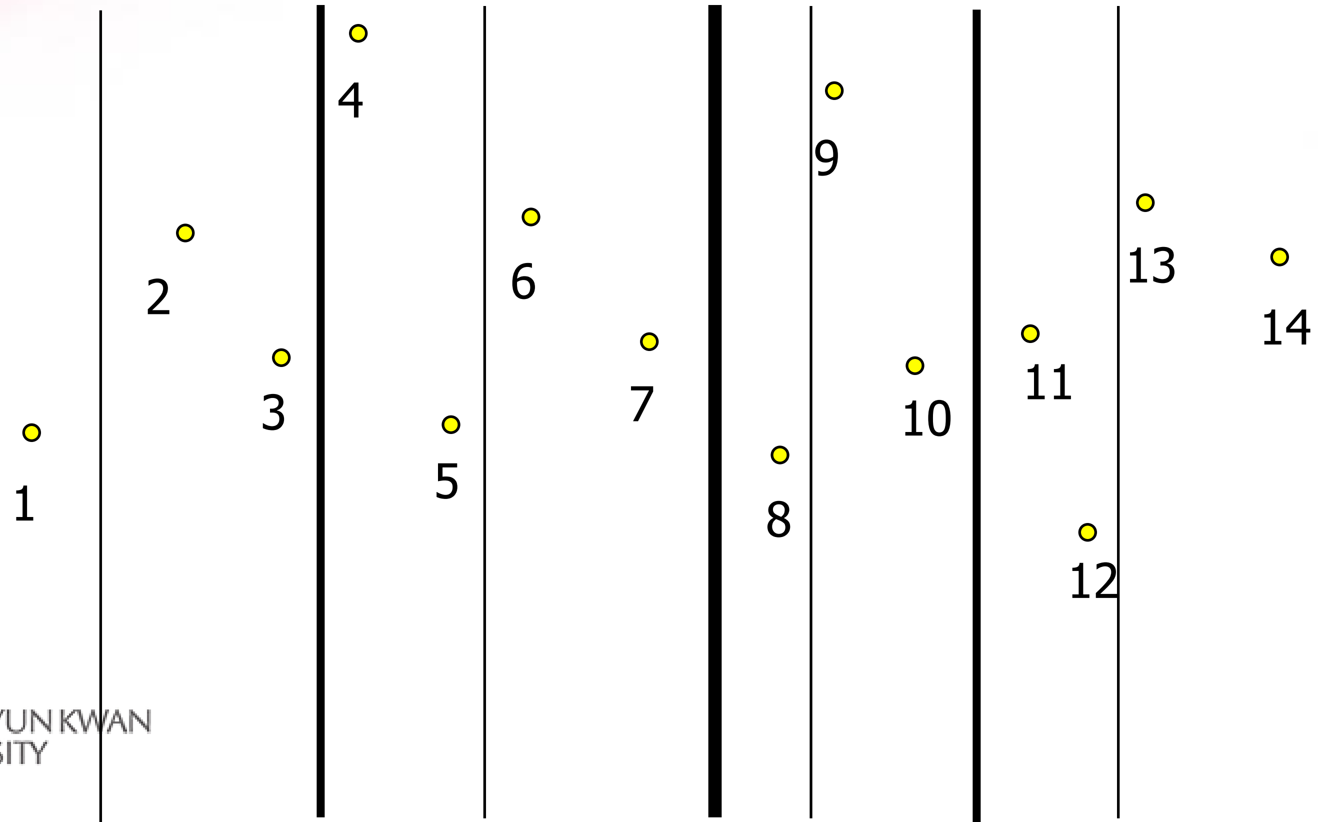
Step 3: But, we can continue the advantage by splitting the sub-problems.



SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

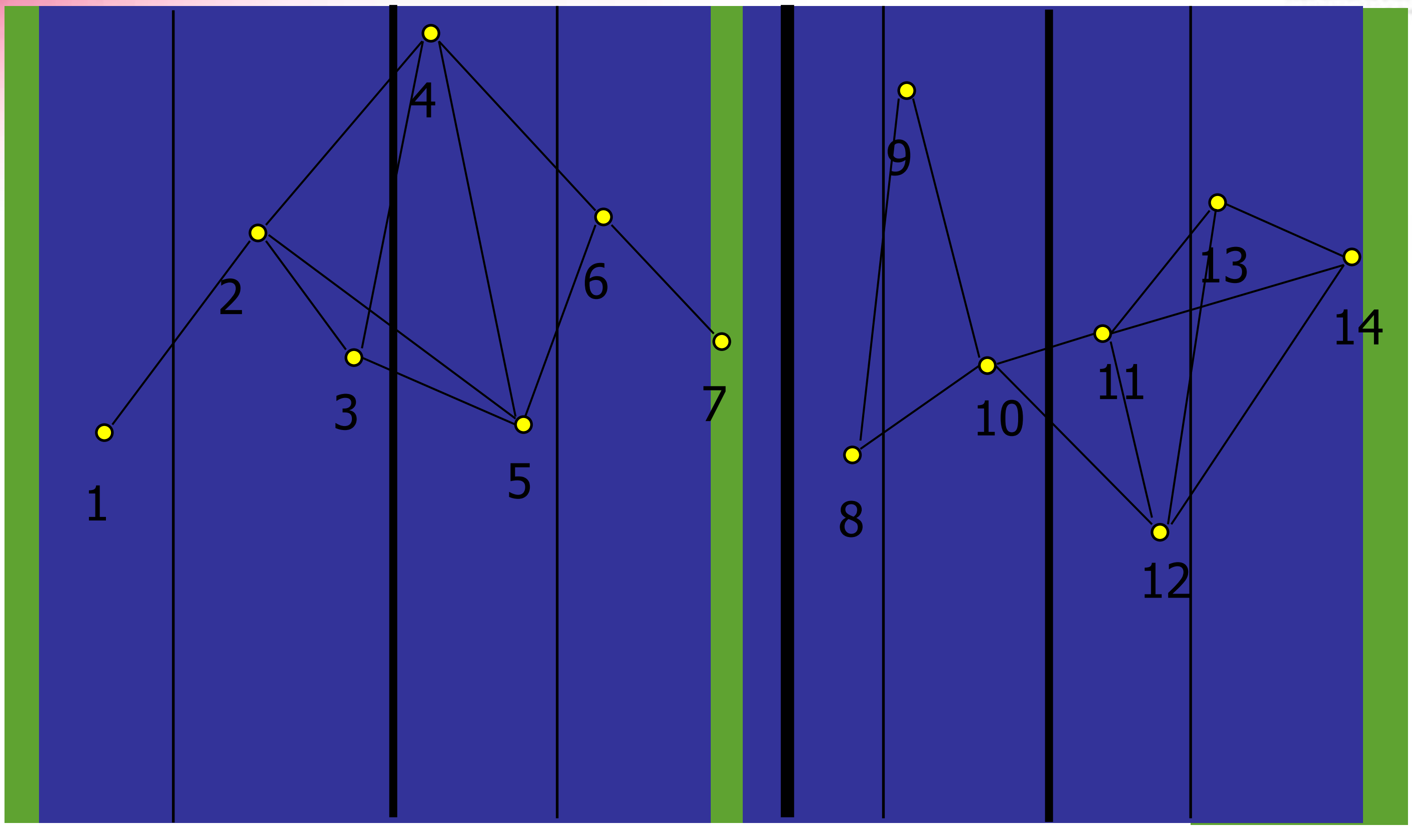
Step 3: In fact we can continue to split until each sub-problem is trivial, i.e., takes one comparison.



SUNG KYUN KWAN UNIVERSITY

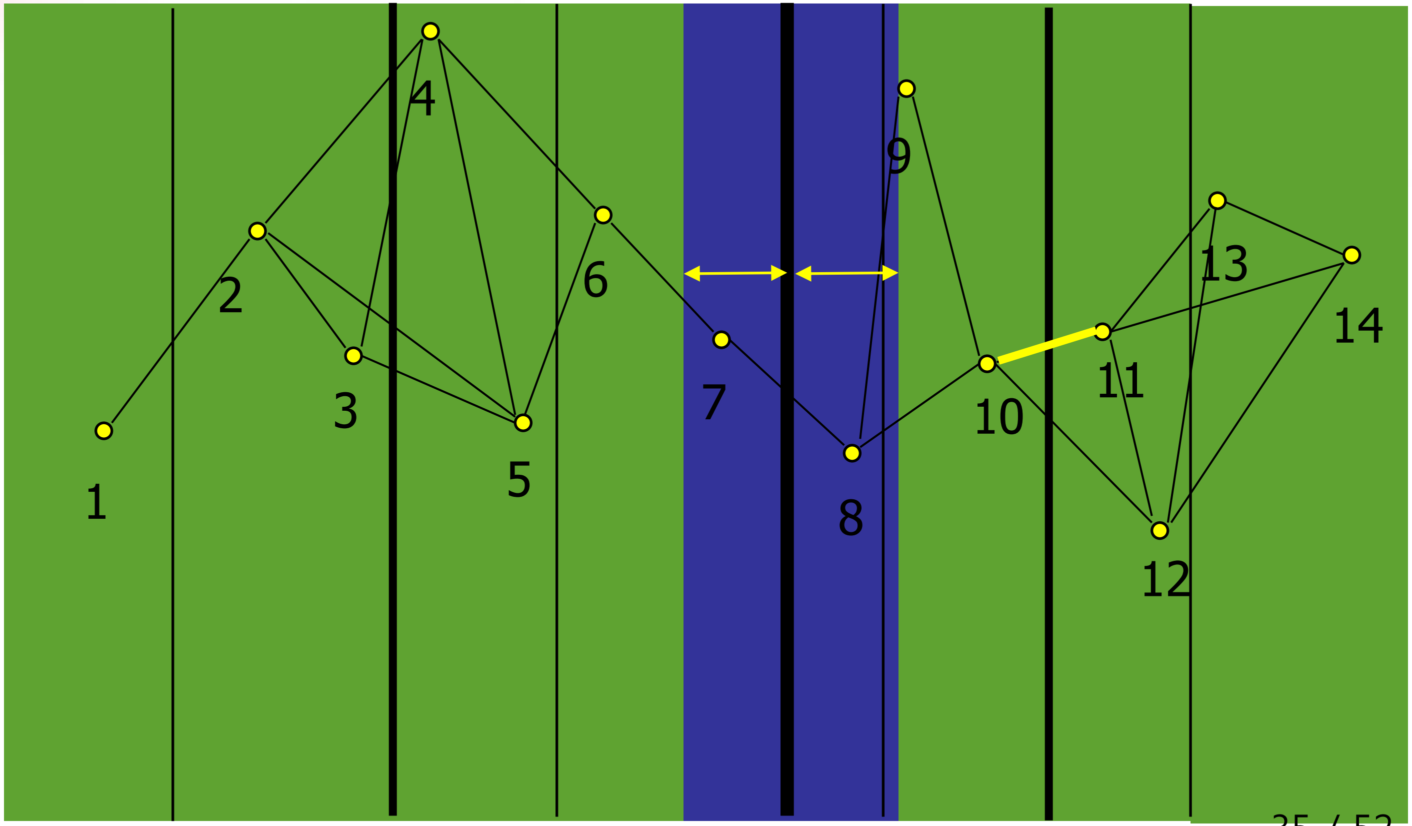
Closest-Pair Algorithm

Finally: The solution to each sub-problem is combined until the final solution is obtained



Closest-Pair Algorithm

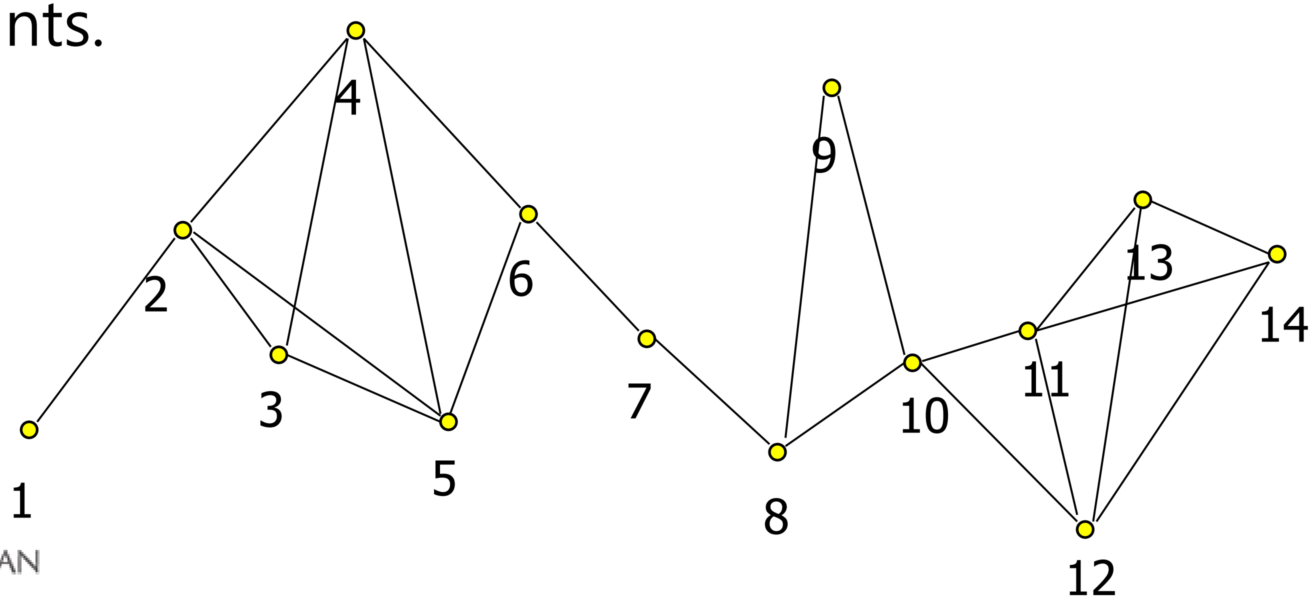
Finally: On the last step the 'strip' will likely be very small. Thus, combining the two largest sub-problems won't require much work.



SUNG KYUN KWAN UNIVERSITY

Closest-Pair Algorithm

- In this example, it takes 22 comparisons to find the closets-pair.
- The brute force algorithm would have taken 91 comparisons.
- But, the real advantage occurs when there are millions of points.



Closest Pair by Divide & Conquer

- Divide:
  - Sort halves by *x-coordinate.*
  - Find vertical line splitting points in half.
- Conquer:
  - Recursively find closest pairs in each half.
- Combine:
  - Check vertices near the border to see if any pair straddling the border is closer together than the minimum seen so far.

Closest Pair by Divide & Conquer

Step 1 Divide the points given into two subsets S_1 and S_2 by a vertical line $x = c$ so that half the points lie to the left or on the line and half the points lie to the right or on the line.

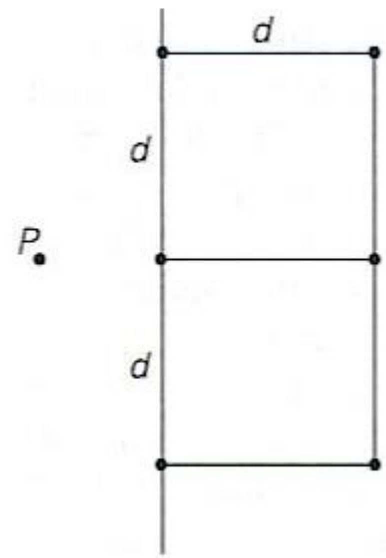
Step 2 Find recursively the closest pairs for the left and right subsets.

Step 3 Set $d = \min\{d_1, d_2\}$
We can limit our attention to the points in the symmetric vertical strip of width $2d$ as possible closest pair. Let C_1 and C_2 be the subsets of points in the left subset S_1 and of the right subset S_2, respectively, that lie in this vertical strip. The points in C_1 and C_2 are stored in increasing order of their *y* coordinates, which is maintained by merging during the execution of the next step.

Step 4 For every point $P(x,y)$ in C_1, we inspect points in C_2 that may be closer to P than d. There can be no more than 6 such points (because $d \le d_2$)!

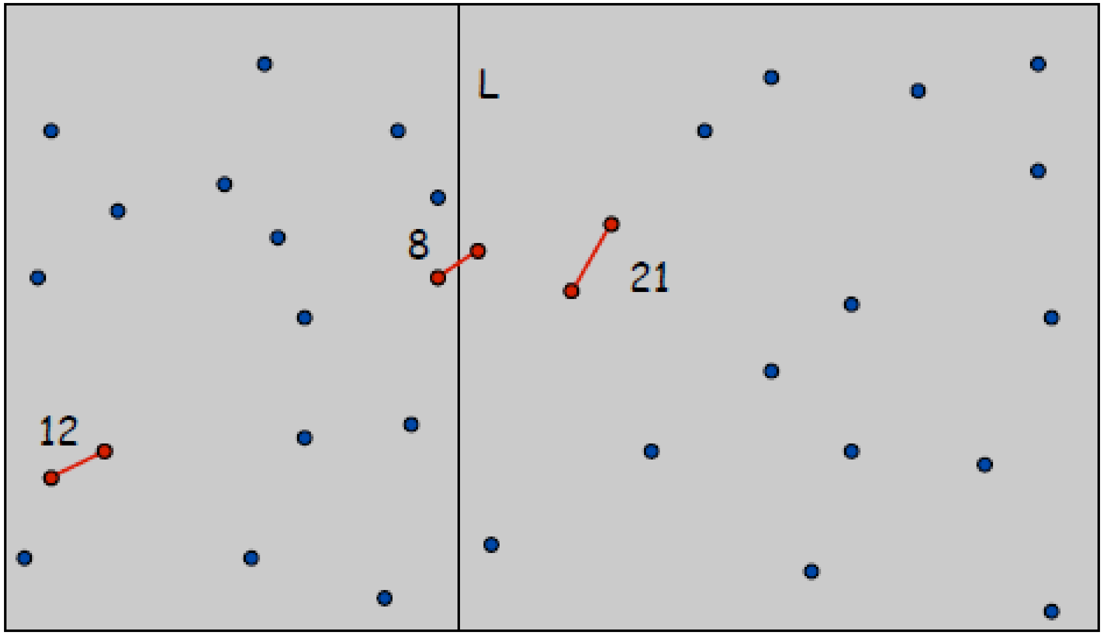
Closest Pair by Divide & Conquer: Worst Case

The worst case scenario is depicted below:



Closest Pair by Divide & Conquer: Algorithm

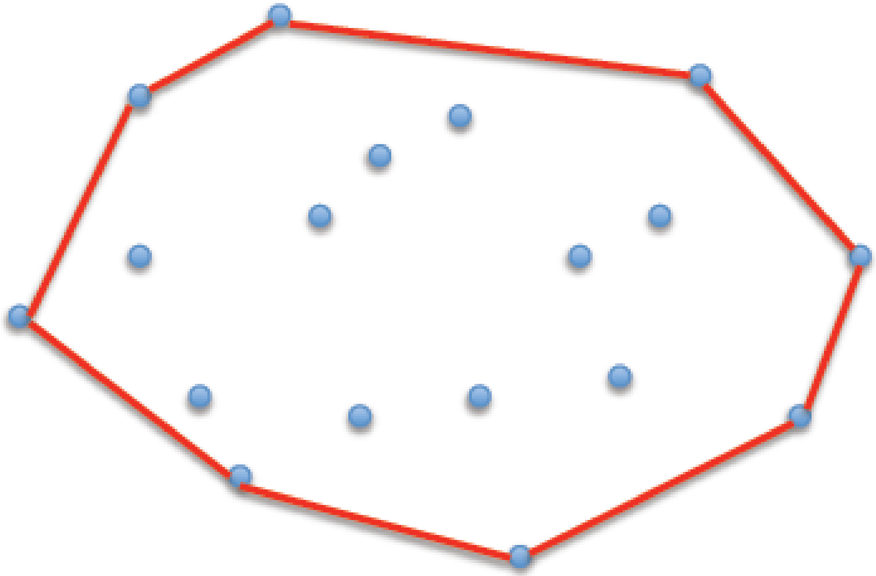
- Divide: draw vertical line L so that roughly ½n points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.



SUNG KYUN KWAN UNIVERSITY

Convex hull (definition)

- H is the smallest convex polygon that contains all the points of Q



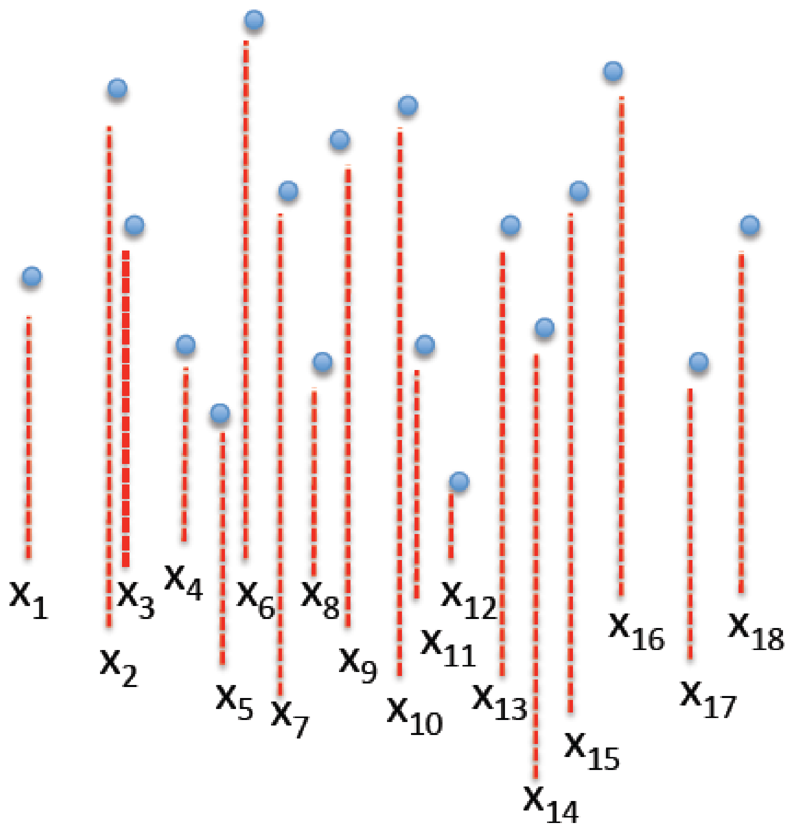
SUNG KYUN KWAN UNIVERSITY

Convex hull (principle)

- Decompose the set of points in equal parts (Qleft and Qright)
- Solve the sub-problems respectively on Qleft and Qright
- Merge both convex hulls Hleft and Hright

Convex hull Algorithm

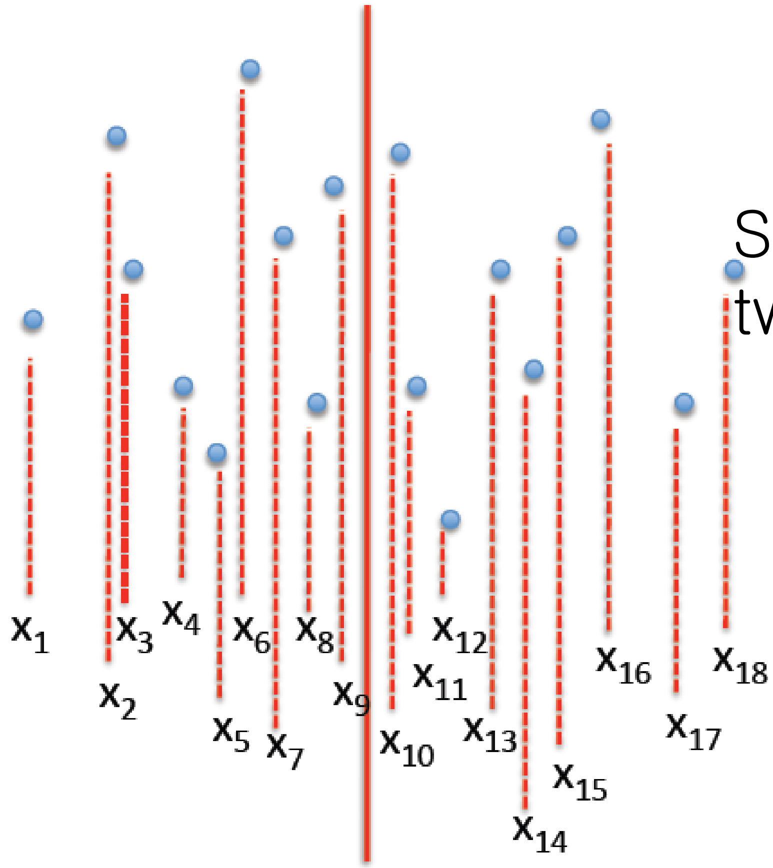
Step 1 : Decomposition



Sort the points

Convex hull Algorithm

Step 2 : Decomposition – Split part

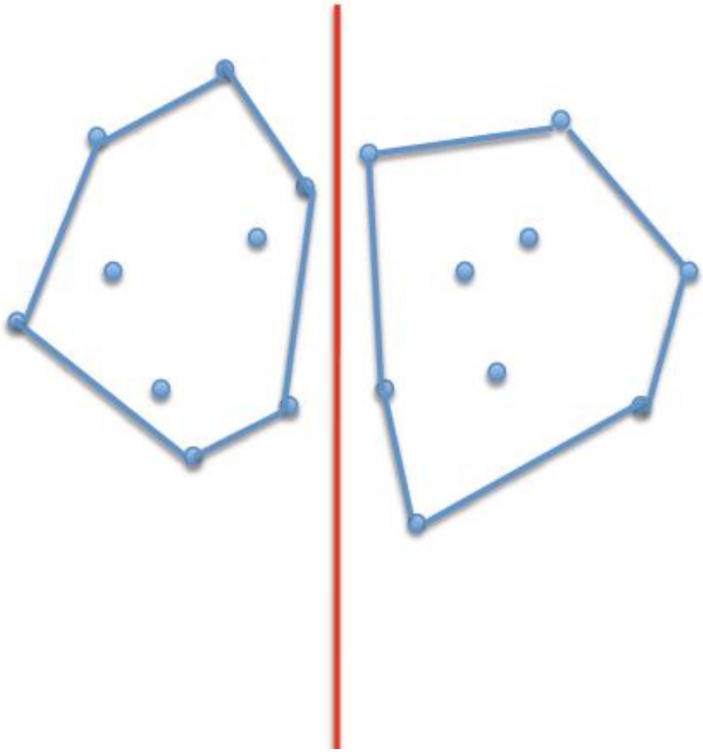


Split the points into two sets of equal size

SUNG KYUN KWAN UNIVERSITY

Convex hull Algorithm

Step 3 : Solve Sub-Problems

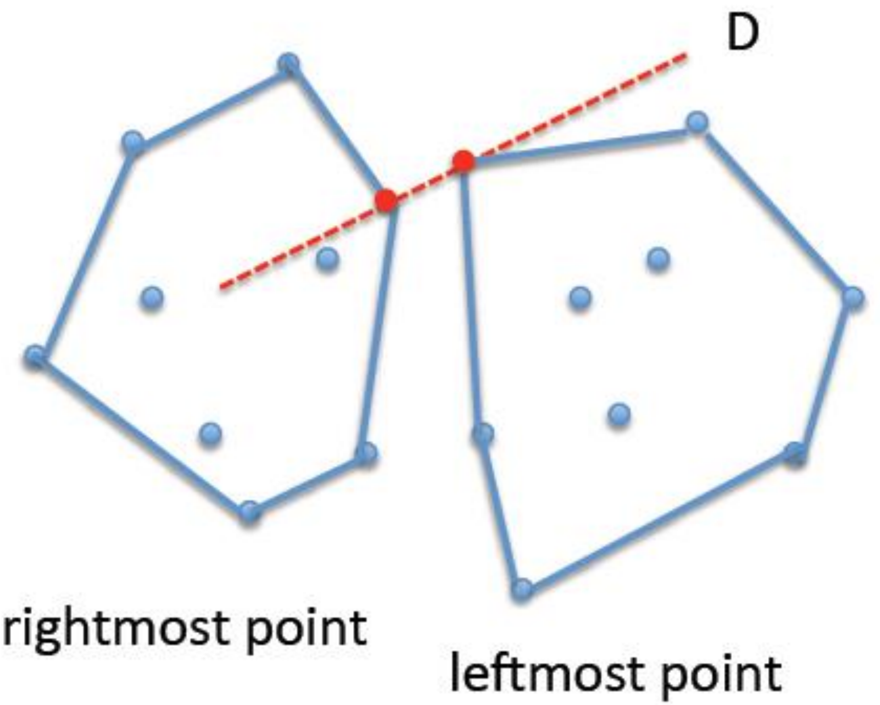


Compute
the convex hulls on
Qleft and Qright

SUNG KYUN KWAN UNIVERSITY

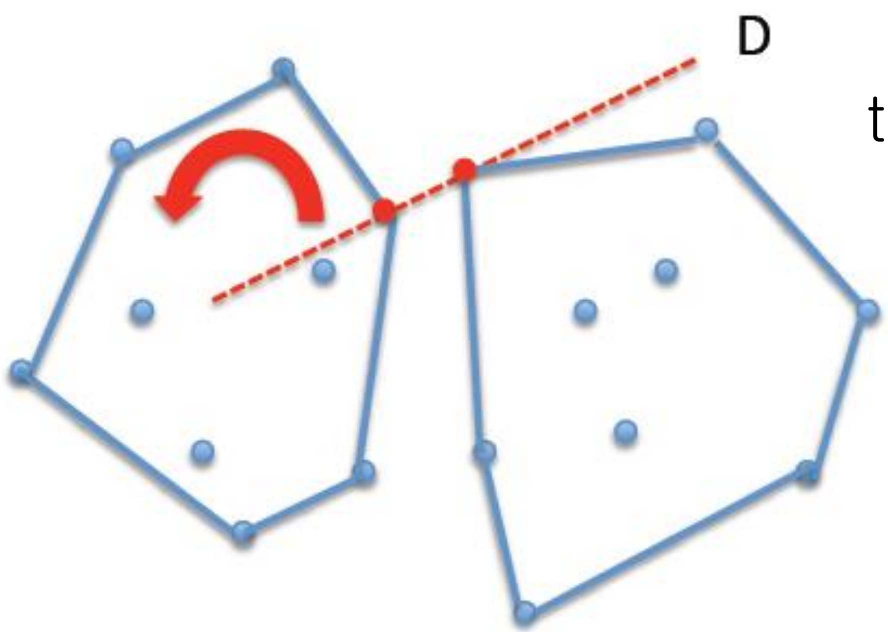
Convex hull Algorithm

Step 4 : Merge Both Solutions



Convex hull Algorithm

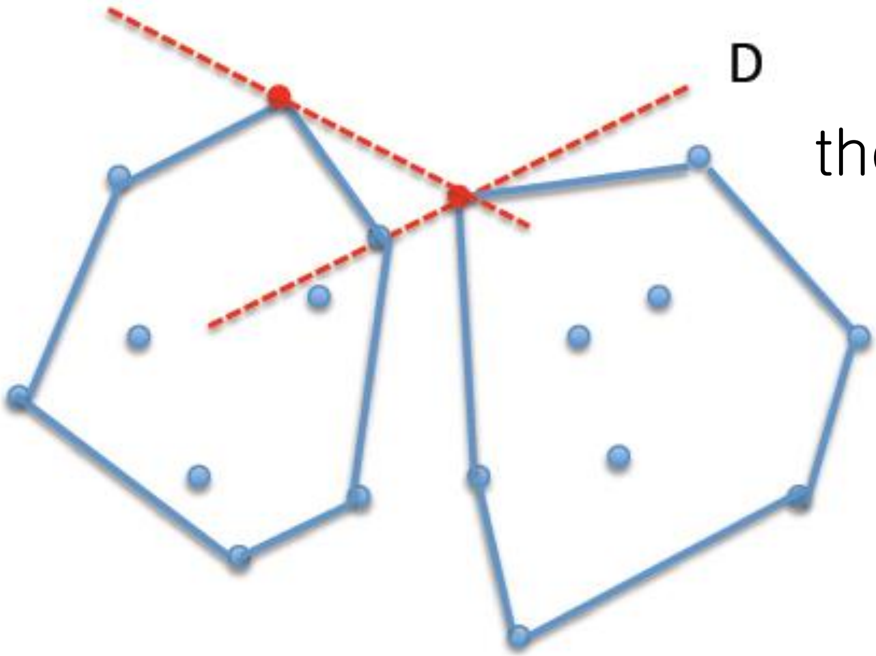
Step 4 : Merge Both Solutions



Determine
the upper tangent

Convex hull Algorithm

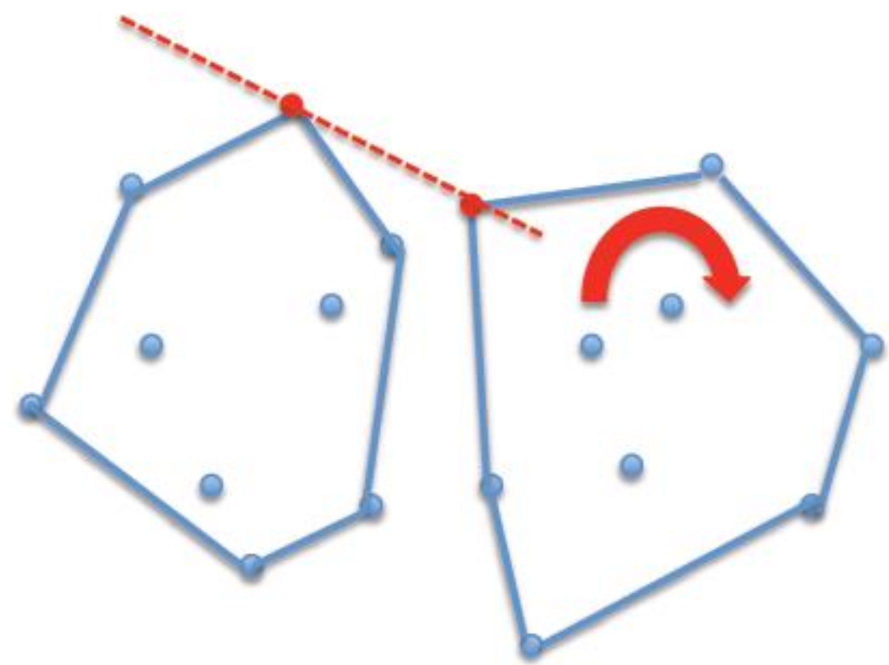
Step 4 : Merge Both Solutions



Determine
the upper tangent

Convex hull Algorithm

Step 4 : Merge Both Solutions

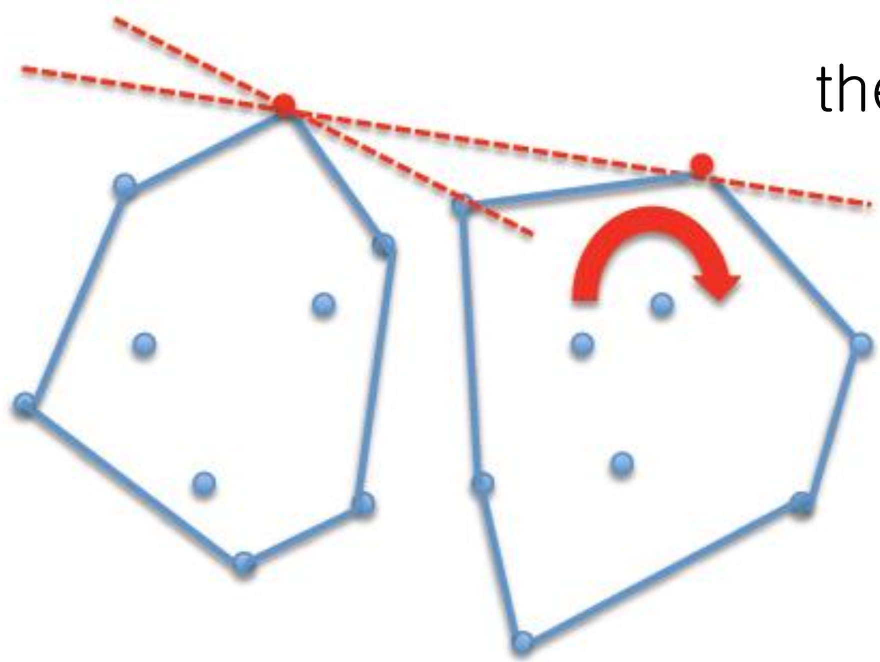


Determine
the upper tangent

Convex hull Algorithm

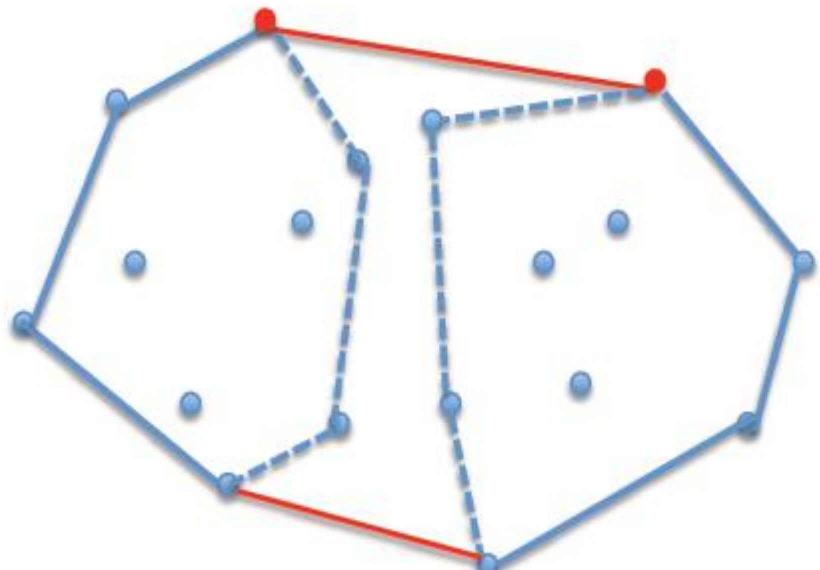
Step 4 : Merge Both Solutions

Determine
the upper tangent



Convex hull Algorithm

Step 4 : Merge Both Solutions



Determine
the lower tangent

Q & A