DIVIDE & CONQUER

Divide & Conquer (3)

Oct 4^{th}, 2018

Algorithm Design and Analysis

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學 National Taiwan University

Slides credited from Hsueh-I Lu, Hsu-Chun Hsiao, & Michael Tsai

Outline

- Recurrence (遞迴)
- Divide-and-Conquer
- D&C #1: Tower of Hanoi (河內塔)
- D&C #2: Merge Sort
- D&C #3: Bitonic Champion
- D&C #4: Maximum Subarray

Divide-and-Conquer 首部曲

- Solving Recurrences
  - Substitution Method
  - Recursion-Tree Method
  - Master Method
- D&C #5: Matrix Multiplication
- D&C #6: Selection Problem
- D&C #7: Closest Pair of Points Problem

Divide-and-Conquer 之神乎奇技

3

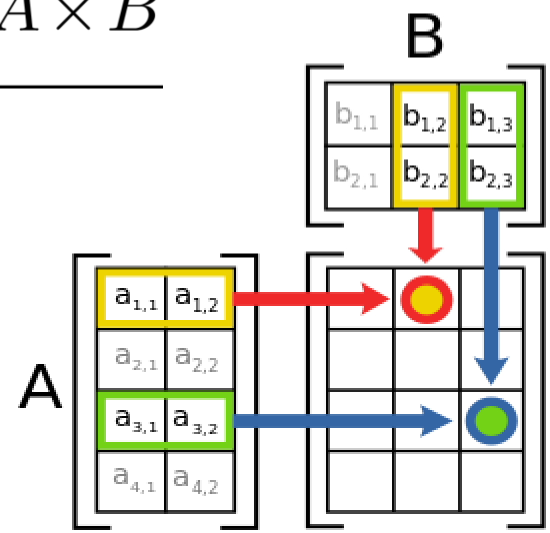
D&C #5: Matrix Multiplication

Textbook Chapter 4.2 – Strassen’s algorithm for matrix multiplication

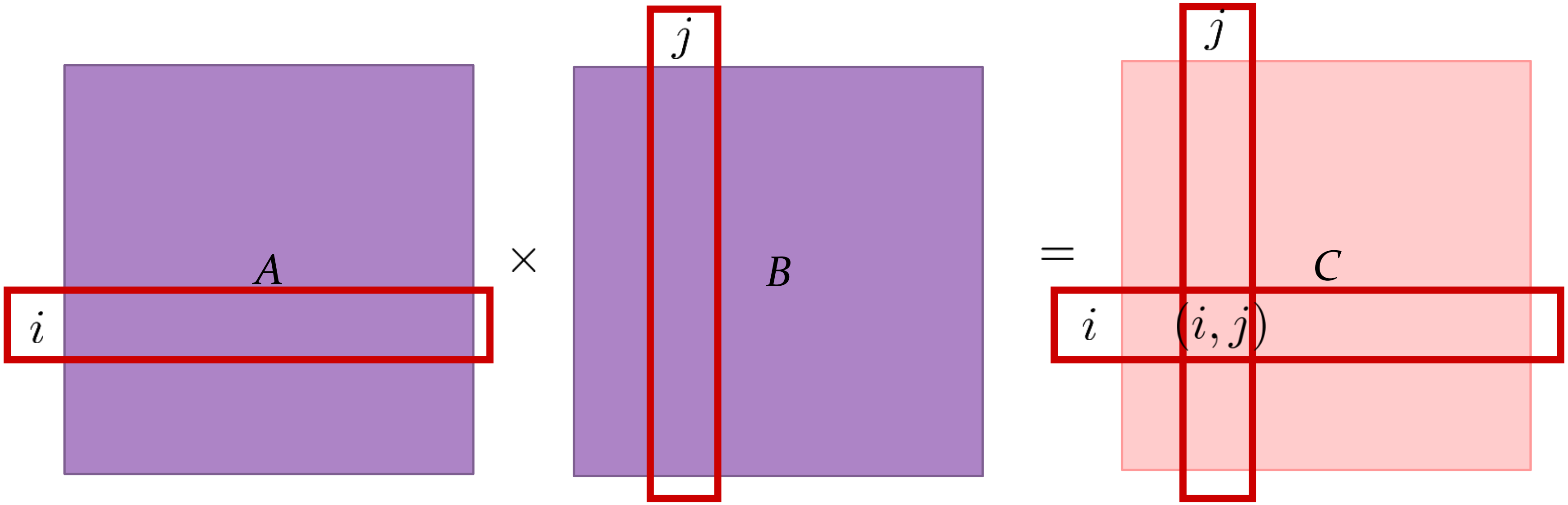
Matrix Multiplication Problem

Input: two $n \times n$ matrices A and B.

Output: the product matrix $C = A \times B$



Naïve Algorithm



$$C(i,j) = \sum_{k=1}^{n} A(i,k) \cdot B(k,j)$$

- Each entry takes n multiplications
- There are total n^2 entries

$\Rightarrow \Theta(n)\Theta(n^2) = \Theta(n^3)$

Matrix Multi. Problem Complexity

$$\text{Upper bound} = O(n^3)$$

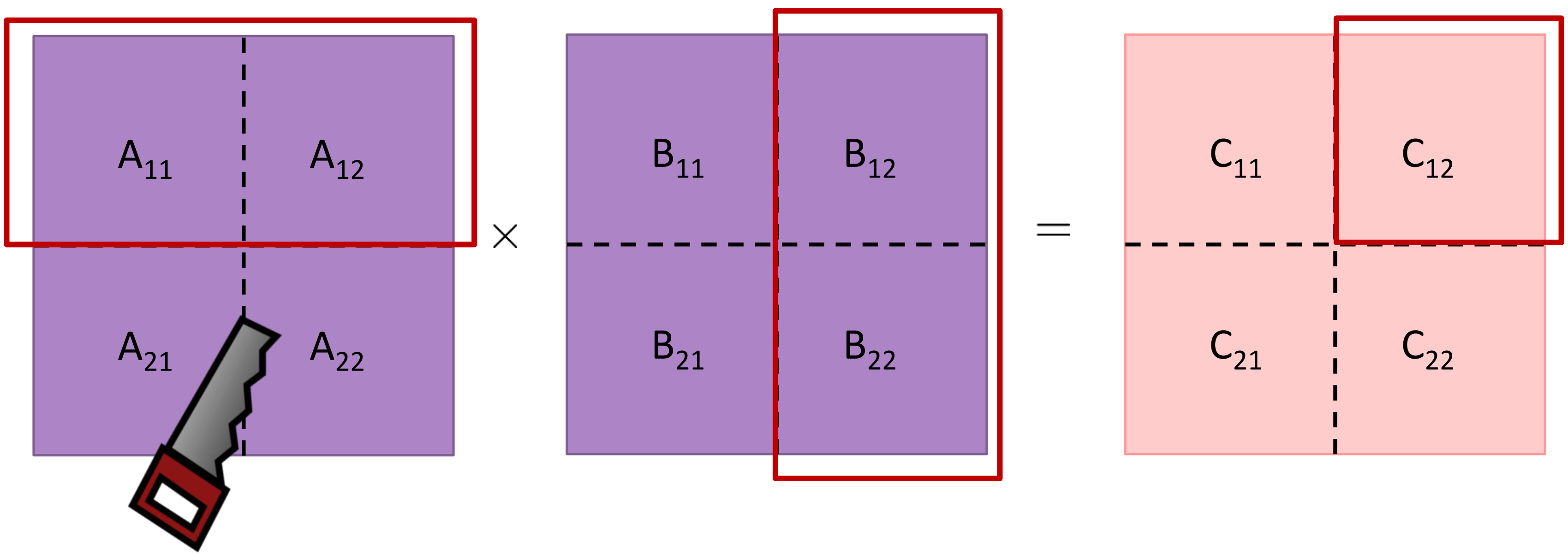
$$\text{Lower bound} = \Omega(n^2)$$

Why?

Divide-and-Conquer

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

- We can assume that $n = 2^k$ for simplicity
  - Otherwise, we can increase n s.t. $n = 2^{\lceil \log_2 n \rceil}$
  - n may not be twice large as the original in this modification



Algorithm Time Complexity

```
MatrixMultiply(n, A, B)
   //base case
   if n == 1
      return AB
   //recursive case
   Divide A and B into n/2 by n/2 submatrices
   C11 = MatrixMultiply(n/2,A11,B11) + MatrixMultiply(n/2,A12,B21)
   C21 = MatrixMultiply(n/2,A11,B12) + MatrixMultiply(n/2,A12,B22)
   C21 = MatrixMultiply(n/2,A21,B11) + MatrixMultiply(n/2,A22,B21)
   C22 = MatrixMultiply(n/2,A21,B12) + MatrixMultiply(n/2,A22,B22)
   return C
```

Base case: $\Theta(1)$

Divide $\Theta(1)$

Conquer $8T(n/2)$

Combine $4\Theta((n/2)^2) = \Theta(n^2)$

- $T(n)$ = time for running `MatrixMultiply(n, A, B)`

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } n \geq 2 \end{cases} \Rightarrow \Theta(n^{\log_2 8}) = \Theta(n^3)$$

Strassen's Technique

- Important theoretical breakthrough by Volker Strassen in 1969
- Reduces the running time from $\Theta(n^3)$ to $\Theta(n^{log^{27}}) \approx \Theta(n^{2.807})$
- The key idea is to <u>reduce the number of recursive calls</u>
  - From 8 recursive calls to 7 recursive calls $T(n/2)$
  - At the cost of extra addition and subtraction operations $\Theta((n/2)^2)$

轉換調整
加加減減
兜出答案

<u>Intuition:</u>

$$ac + ad + bc + bd = (a+b)(c+d)$$

4 multiplications
3 additions
→
1 multiplication
2 additions

Strassen’s Algorithm

- $C = A \times B$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{array}{rcll}
C_{11} & = & M_1 + M_4 - M_5 + M_7 & \mathbf{2+1-} \\
C_{12} & = & M_3 + M_5 & \mathbf{1+} \\
C_{21} & = & M_2 + M_4 & \mathbf{1+} \\
C_{22} & = & M_1 - M_2 + M_3 + M_6 & \mathbf{2+1-} \\
M_1 & = & (A_{11} + A_{22})(B_{11} + B_{22}) & \mathbf{2+1\times} \\
M_2 & = & (A_{21} + A_{22})B_{11} & \mathbf{1+1\times} \\
M_3 & = & A_{11}(B_{12} - B_{22}) & \mathbf{1-1\times} \\
M_4 & = & A_{22}(B_{21} - B_{11}) & \mathbf{1-1\times} \\
M_5 & = & (A_{11} + A_{12})B_{22} & \mathbf{1+1\times} \\
M_6 & = & (A_{21} - A_{11})(B_{11} + B_{12}) & \mathbf{1+1-1\times} \\
M_7 & = & (A_{12} - A_{22})(B_{21} + B_{22}) & \mathbf{1+1-1\times} \\
\hline
 & & & \mathbf{12+6-7\times}
\end{array}$$

$$18\Theta((n/2)^2) + 7T(n/2)$$

Verification of Strassen's Algorithm

$$\begin{aligned} C_{12} &= M_3 + M_5 \\ &= A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22} \\ &= A_{11}B_{12} + A_{12}B_{22} \end{aligned}$$

$$\begin{aligned} C_{21} &= M_2 + M_4 \\ &= (A_{21} + A_{22})B_{11} + A_{22}(B_{21} - B_{11}) \\ &= A_{21}B_{11} + A_{22}B_{21} \end{aligned}$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

- Practice

$$\begin{aligned} C_{11} &= M_1 + M_4 - M_5 + M_7 \\ C_{22} &= M_1 - M_2 + M_3 + M_6 \end{aligned}$$

Strassen's Algorithm Time Complexity

```
Strassen(n, A, B)
  // base case
  if n == 1
    return AB
  // recursive case
  Divide A and B into n/2 by n/2 submatrices
  M1  = Strassen(n/2, A11+A22, B11+B22)
  M2  = Strassen(n/2, A21+A22, B11)
  M3  = Strassen(n/2, A11, B12-B22)
  M4  = Strassen(n/2, A22, B21-B11)
  M5  = Strassen(n/2, A11+A12, B22)
  M6  = Strassen(n/2, A11-A21, B11+B12)
  M7  = Strassen(n/2, A12-A22, B21+B22)
  C11 = M1 + M4 - M5 + M7
  C12 = M3 + M5
  C21 = M2 + M4
  C22 = M1 - M2 + M3 + M6
  return C
```

Base case: $\Theta(1)$

Divide $\Theta(1)$

Conquer

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

Combine

$\Theta(n^2)$

- $T(n)$ = time for running `Strassen(n,A,B)`

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T(n/2) + \Theta(n^2) & \text{if } n \geq 2 \end{cases}$$

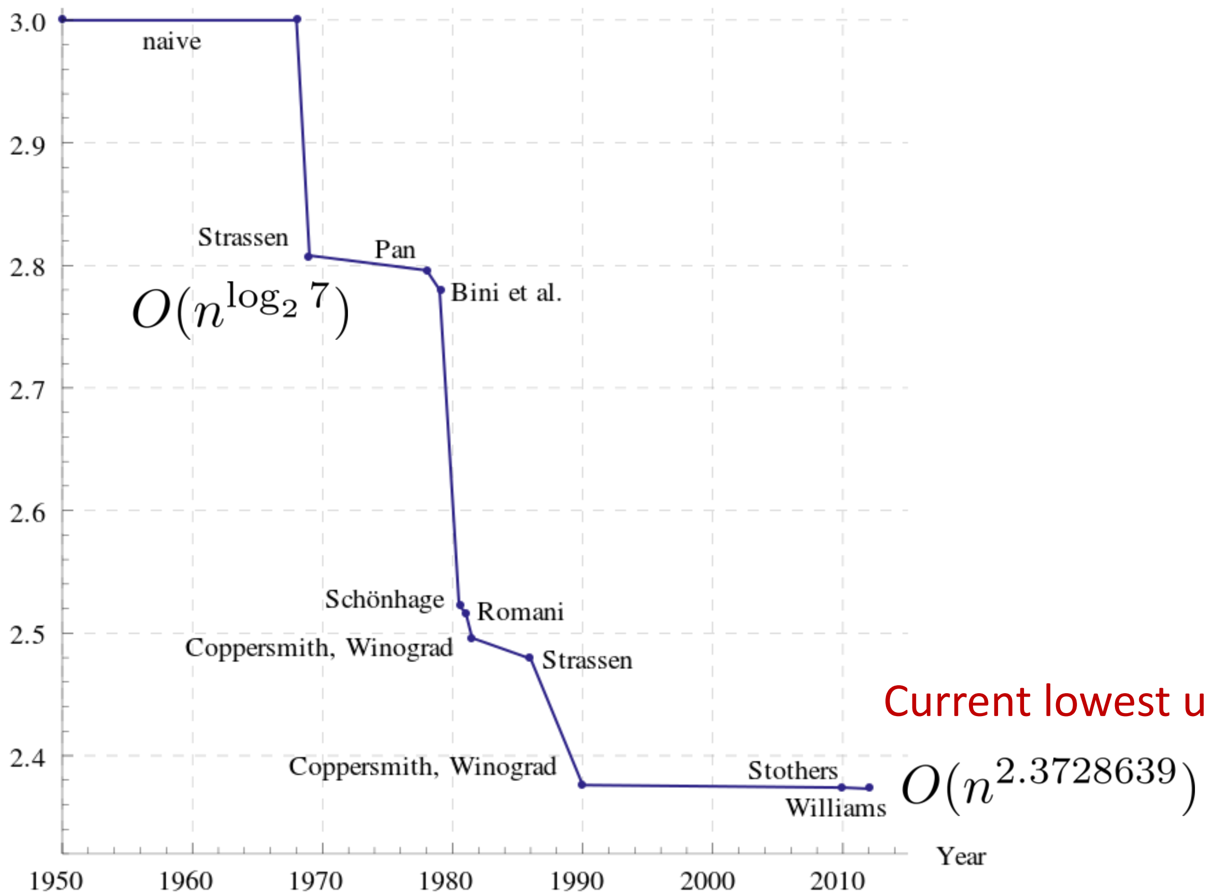
➡ $\Theta(n^{\log_2 7}) \sim \Theta(n^{2.807})$

Practicability of Strassen's Algorithm

- Disadvantages
  1. Larger constant factor than it in the naïve approach
  $$c_1 n^{\log_2 7}, c_2 n^3 \to c_1 > c_2$$
  2. Less numerical stable than the naïve approach
     - Larger errors accumulate in non-integer computation due to limited precision
  3. The submatrices at the levels of recursion consume space
  4. Faster algorithms exist for sparse matrices
- Advantages: find the crossover point and combine two subproblems

Matrix Multiplication Upper Bounds

- Each algorithm gives an upper bound



Matrix Multi. Problem Complexity

Upper bound $= O(n^{2.3728639})$

Lower bound $= \Omega(n^2)$

16

D&C #6: Selection Problem

Textbook Chapter 9.3 – Selection in worst-case linear time

Selection Problem

- Input:
  - An array A of n distinct integers.
  - An index k with $1 \leq k \leq n$.
- Output:

  The k-th largest number in A.

$n = 10, k = 5$

3 7 9 17 5 2 21 18 33 4

Selection Problem ≦ Sorting Problem

- If the sorting problem can be solved in $O\big(f(n)\big)$, so can the selection problem based on the algorithm design
  - Step 1: sort A into increasing order
  - Step 2: output $A[n-k+1]$

Selection Problem Complexity

Upper bound $= O(n \log n)$

Can we make the upper bound better if we do not sort them?

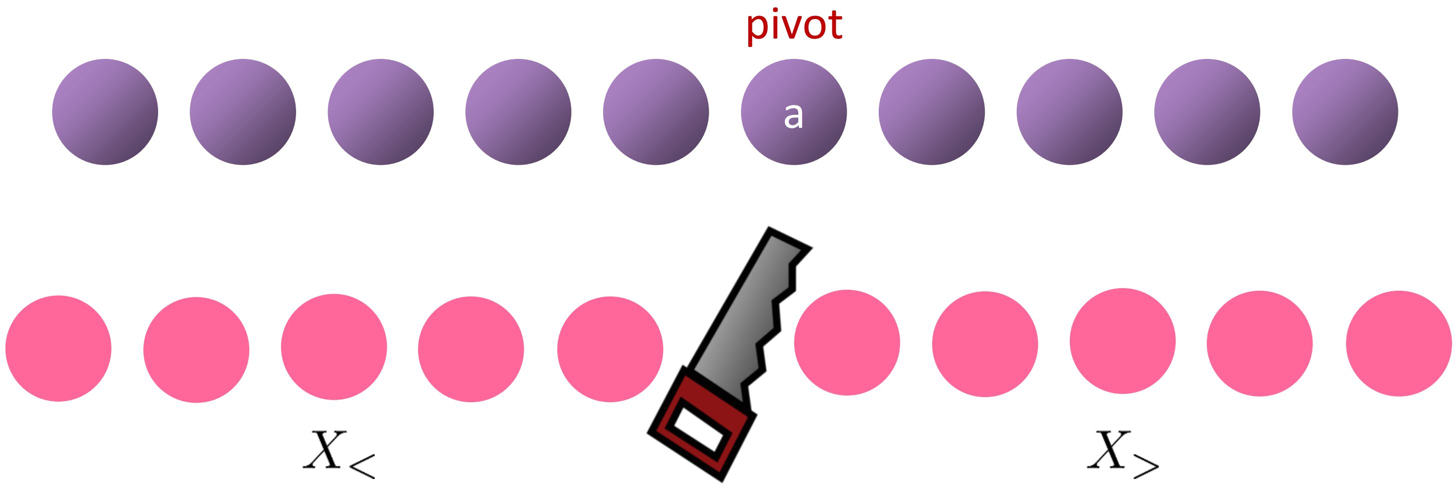
Lower bound $= \Omega(n)$

Hardness of Selection Problem

- Upper bounds in terms of #comparisons
  - $3n + o(n)$ by Schonhage, Paterson, and Pippenger (*JCSS* 1975).
  - $2.95n$ by Dor and Zwick (*SODA* 1995, *SIAM Journal on Computing* 1999).
- Lower bounds in terms of #comparisons
  - $2n+o(n)$ by Bent and John (*STOC* 1985)
  - $(2+2^{-80})n$ by Dor and Zwick (*FOCS* 1996, *SIAM Journal on Discrete Math* 2001).

Divide-and-Conquer

- Idea
  - Select a pivot and divide the inputs into two subproblems
  - If $k \leq |X_>|$, we find the k-th largest
  - If $k > |X_>|$, we find the $(k - |X_>|)$-th largest



We want these subproblems to have similar size
→ The better pivot is the medium in the input array

Homework Practice

D&C #7: Closest Pair of Points Problem

Textbook Chapter 33.4 – Finding the closest pair of points

Closest Pair of Points Problem

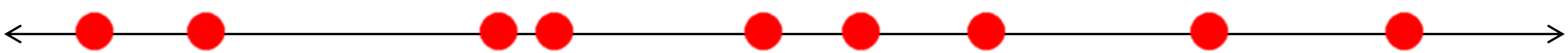
- Input: $n \geq 2$ points, where $p_i = (x_i, y_i)$ for $0 \leq i < n$
- Output: two points p_i and p_j that are closest
  - "Closest": smallest Euclidean distance
  - Euclidean distance between p_i and p_j: $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

- Brute-force algorithm
  - Check all pairs of points: $\Theta(C_2^n) = \Theta(n^2)$

Closest Pair of Points Problem

- 1D:
  - Sort all points $\Theta(n \log n)$
  - Scan the sorted points to find the closest pair in one pass $\Theta(n)$
    - We only need to examine the adjacent points

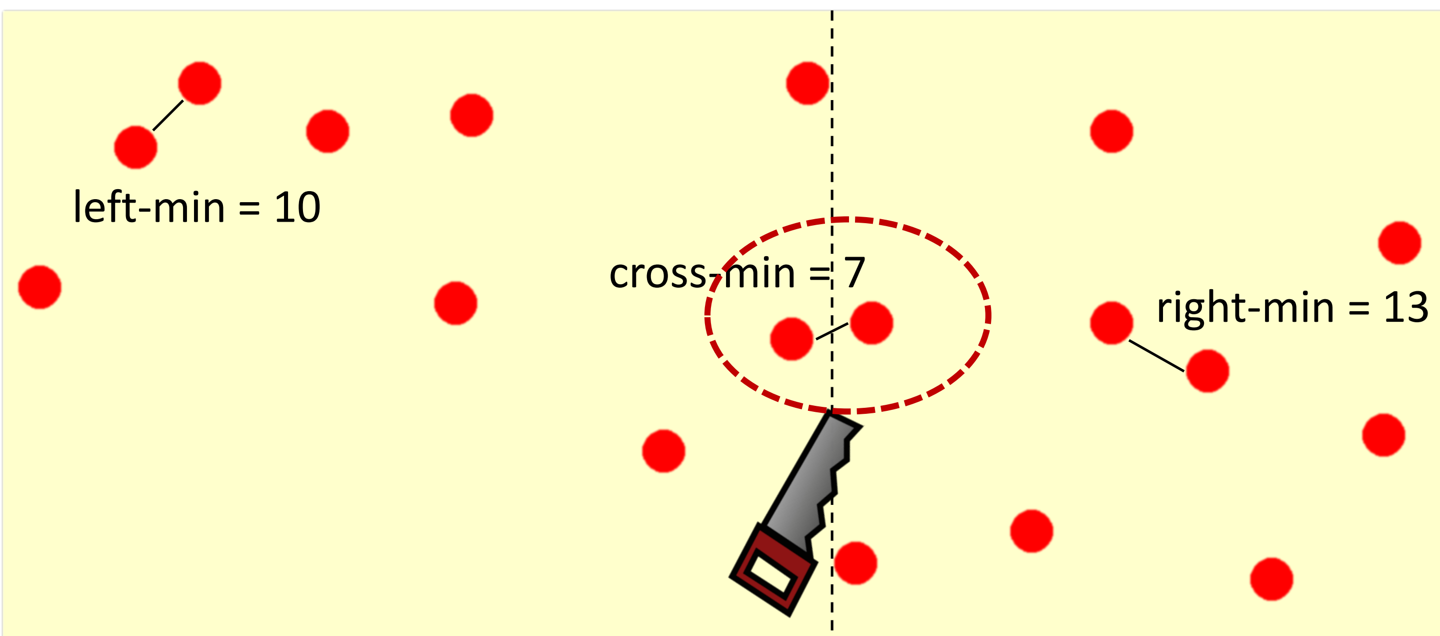
➡ $T(n) = \Theta(n \log n)$



- 2D:

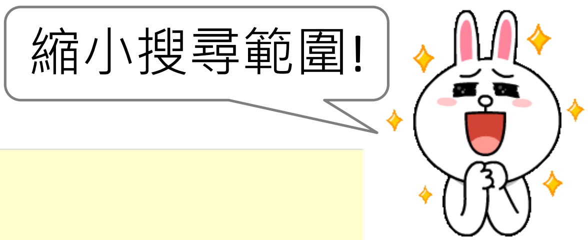
Divide-and-Conquer Algorithm

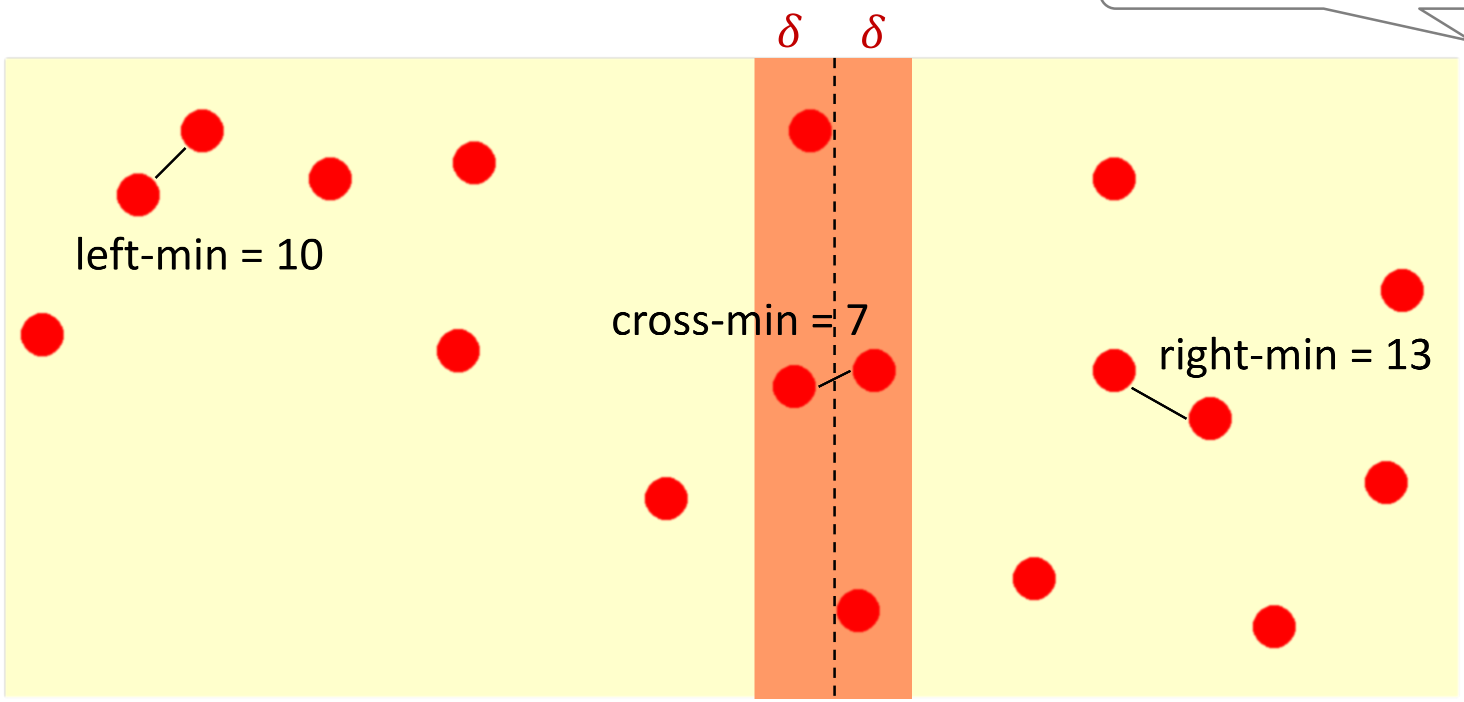
- Divide: divide points evenly along x-coordinate
- Conquer: find closest pair in each region recursively
- Combine: find closet pair with one point in each region, and return the best of three solutions



Cross Two Regions

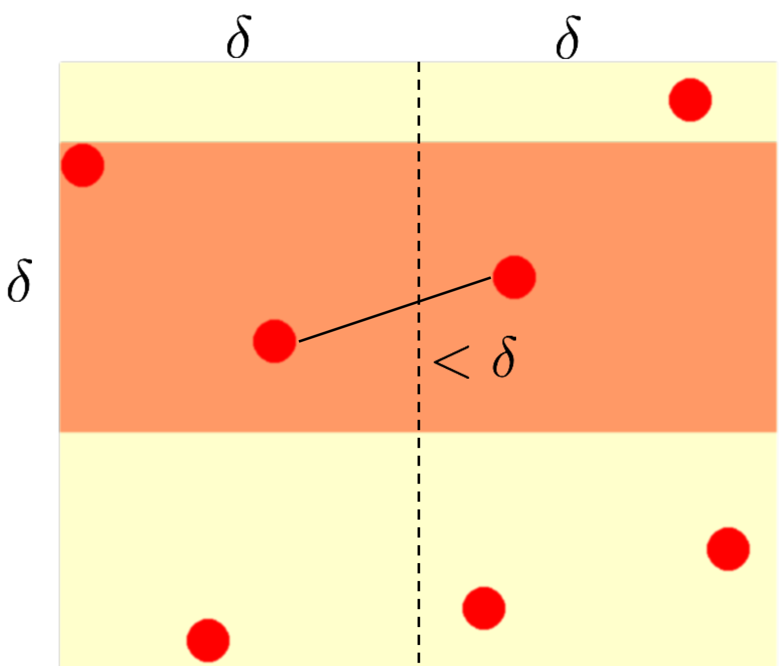
- Algo 1: check all pairs that cross two regions → $n/2 \times n/2$ combinations
- Algo 2: only consider points within δ of the cut, $\delta = \min\{\text{l}-\text{min}, \text{r}-\text{min}\}$
  - Other pairs of points must have distance larger than δ

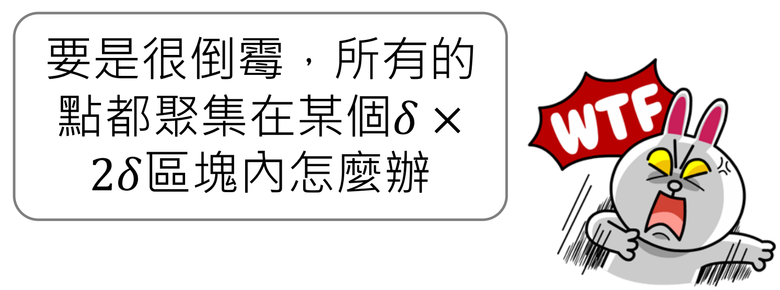




Cross Two Regions

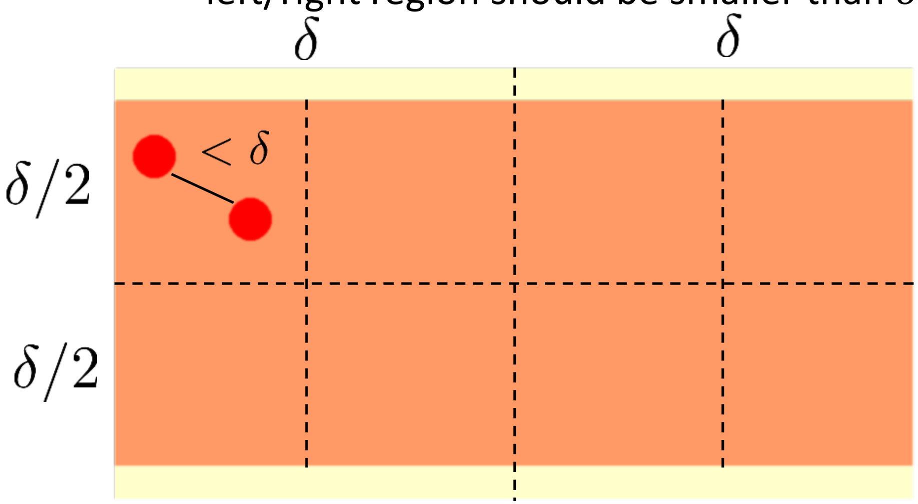
- Algo 1: check all pairs that cross two regions → $n/2 \times n/2$ combinations
- Algo 2: only consider points within δ of the cut, $\delta = \min\{\text{l−min}, \text{r−min}\}$
- Algo 3: only consider pairs within $\delta \times 2\delta$ blocks
  - Obs 1: every pair with smaller than δ distance must appear in a $\delta \times 2\delta$ block





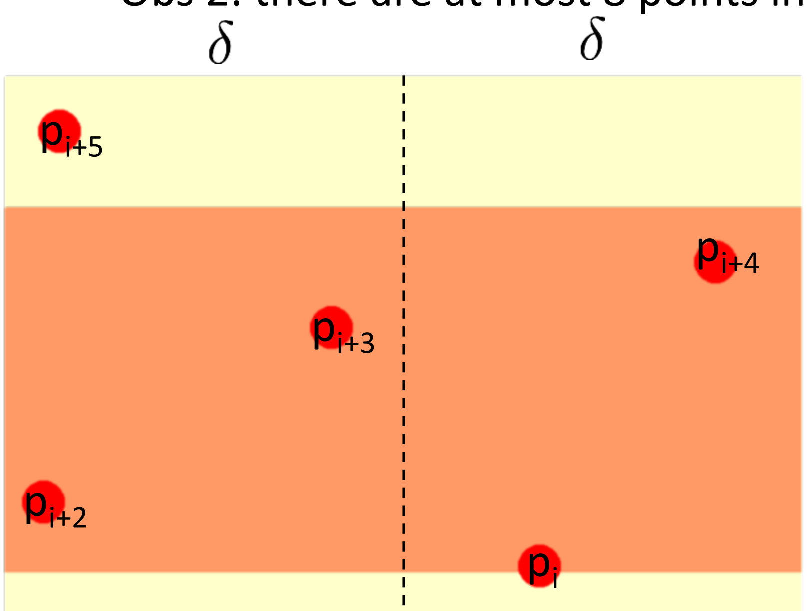
Cross Two Regions

- Algo 1: check all pairs that cross two regions → $n/2 \times n/2$ combinations
- Algo 2: only consider points within δ of the cut, $\delta = \min\{\mathrm{l{-}min}, \mathrm{r{-}min}\}$
- Algo 3: only consider pairs within $\delta \times 2\delta$ blocks
  - Obs 1: every pair with smaller than δ distance must appear in a $\delta \times 2\delta$ block
  - Obs 2: there are at most 8 points in a $\delta \times 2\delta$ block
    - Each $\delta/2 \times \delta/2$ block contains at most 1 point, otherwise the distance returned from left/right region should be smaller than δ



Cross Two Regions

- Algo 1: check all pairs that cross two regions → $n/2 \times n/2$ combinations
- Algo 2: only consider points within δ of the cut, $\delta = \min\{\mathrm{l{-}min}, \mathrm{r{-}min}\}$
- Algo 3: only consider pairs within $\delta \times 2\delta$ blocks
  - Obs 1: every pair with smaller than δ distance must appear in a $\delta \times 2\delta$ block
  - Obs 2: there are at most 8 points in a $\delta \times 2\delta$ block



`Find-closet-pair-across-regions`

1. Sort the points by y-values within δ of the cut (yellow region)
2. For the sorted point p_i, compute the distance with $p_{i+1}, p_{i+2}, \ldots, p_{i+7}$
3. Return the smallest one

At most 7 distance calculations needed

Algorithm Complexity

```
Closest-Pair(P)
  // termination condition (base case)
  if |P| <= 3 brute-force finding closest pair and return it
  // Divide
  find a vertical line L s.t. both planes contain half of the points
  // Conquer (by recursion)
  left-pair, left-min = Closest-Pair(points in the left)
  right-pair, right-min = Closest-Pair(points in the right)
  // Combine
  delta = min{left-min, right-min}
  remove points that are delta or more away from L // Obs 1
  sort remaining points by y-coordinate into p0, …, pk
  for point pi:
    compute distances with pi+1, pi+2, …, pi+7 // Obs 2
    update delta if a closer pair is found
  return the closest pair and its distance
```

$\Theta(1)$

$\Theta(n \log n)$

$2T(n/2)$

$\Theta(n \log n)$

$\Theta(n)$

- $T(n)$ = time for running `Closest-Pair(P)` with |P| = n

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 3 \\ 2T\left(\frac{n}{2}\right) + \Theta(n \log n) & \text{if } n > 3 \end{cases} \Rightarrow T(n) = \Theta(n \log^2 n)$$

Exercise 4.6-2

Preprocessing

- Idea: do not sort inside the recursive case

```
Closest-Pair(P)
  sort P by x- and y-coordinate and store in Px and Py                  Θ(n log n)
  // termination condition (base case)
  if |P| <= 3 brute-force finding closest pair and return it            Θ(1)
  // Divide
  find a vertical line L s.t. both planes contain half of the points    Θ(n)
  // Conquer (by recursion)
  left-pair, left-min = Closest-Pair(points in the left)                2T(n/2)
  right-pair, right-min = Closest-Pair(points in the right)
  // Combine
  delta = min{left-min, right-min}
  remove points that are delta or more away from L // Obs 1
  for point p_i in sorted candidates                                    Θ(n)
    compute distances with p_i+1, p_i+2, …, p_i+7 // Obs 2
    update delta if a closer pair is found
  return the closest pair and its distance
```

$$T'(n) = \begin{cases} \Theta(1) & \text{if } n \leq 3 \\ 2T'\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 3 \end{cases} \Rightarrow \begin{array}{l} T'(n) = \Theta(n \log n) \\ T(n) = \Theta(n \log n) \end{array}$$

Closest Pair of Points Problem

- $O(n)$ algorithm
  - Taking advantage of randomization
    - Chapter 13.7 of Algorithm Design by Kleinberg & Tardos
    - Samir Khuller and Yossi Matias. 1995. A simple randomized sieve algorithm for the closest-pair problem. Inf. Comput. 118, 1 (April 1995), 34-37.

Concluding Remarks

- When to use D&C
  - Whether the problem with small inputs can be solved directly
  - Whether subproblem solutions can be combined into the original solution
  - Whether the overall complexity is better than naïve
- Note
  - Try different ways of dividing
  - D&C may be suboptimal due to repetitive computations
  - Example.
    - D&C algo for Fibonacci: $\Omega((\frac{1+\sqrt{5}}{2})^n)$
    - Bottom-up algo for Fibonacci: $\Theta(n)$

```
Fibonacci(n)
  if n < 2
    return 1
  a[0]=1
  a[1]=1
  for i = 2 … n
    a[i]=a[i-1]+a[i-2]
  return a[n]
```

1. Divide

2. Conquer

3. Combine

Our next topic: **Dynamic Programming**
"a technique for solving problems with overlapping subproblems"

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw