Algorithm Design and Analysis
Divide and Conquer (3)

http://ada.miulab.tw slido: #ADA2022

Yun-Nung (Vivian) Chen

(Slides modified from hil, hchsiao)

國立臺灣大學
National Taiwan University

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

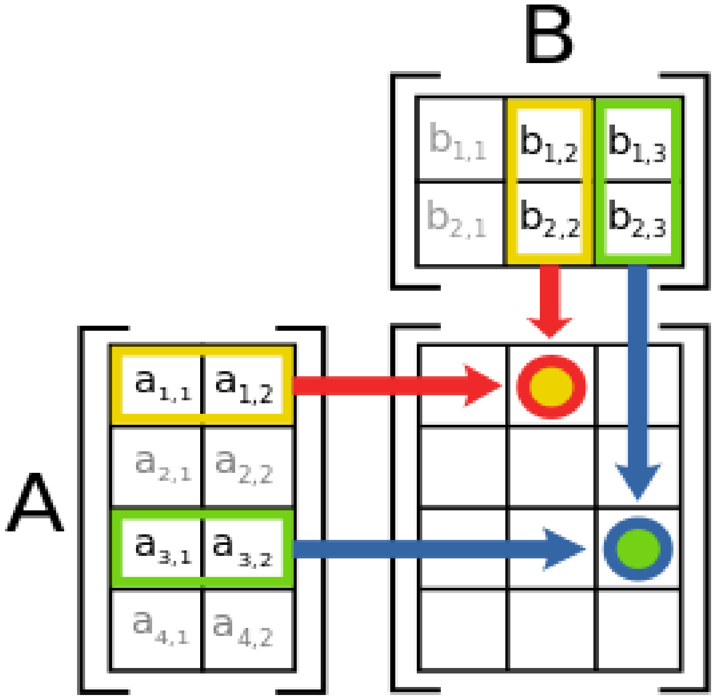
Divide-and-Conquer
之神乎奇技

D&C #5: Matrix Multiplication

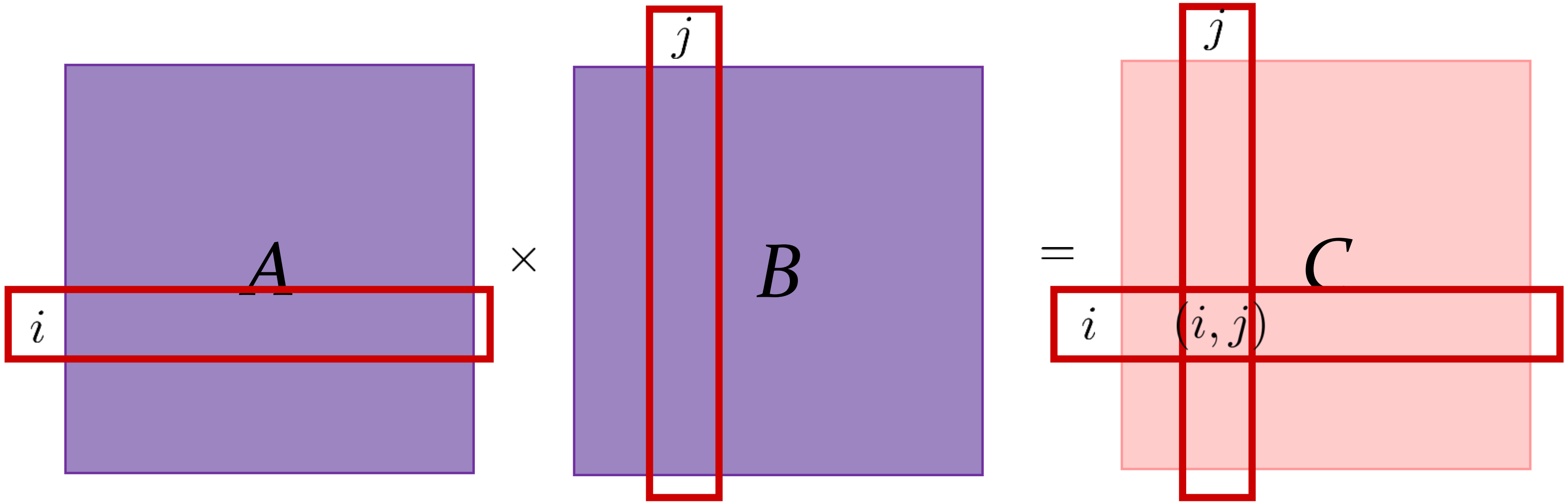
Textbook Chapter 4.2 – Strassen’s algorithm for matrix multiplication

Matrix Multiplication Problem

- Input: two $n \times n$ matrices, A and B.
- Output: a 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

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

Matrix Multi. Problem Complexity

Upper bound $= O(n^3)$

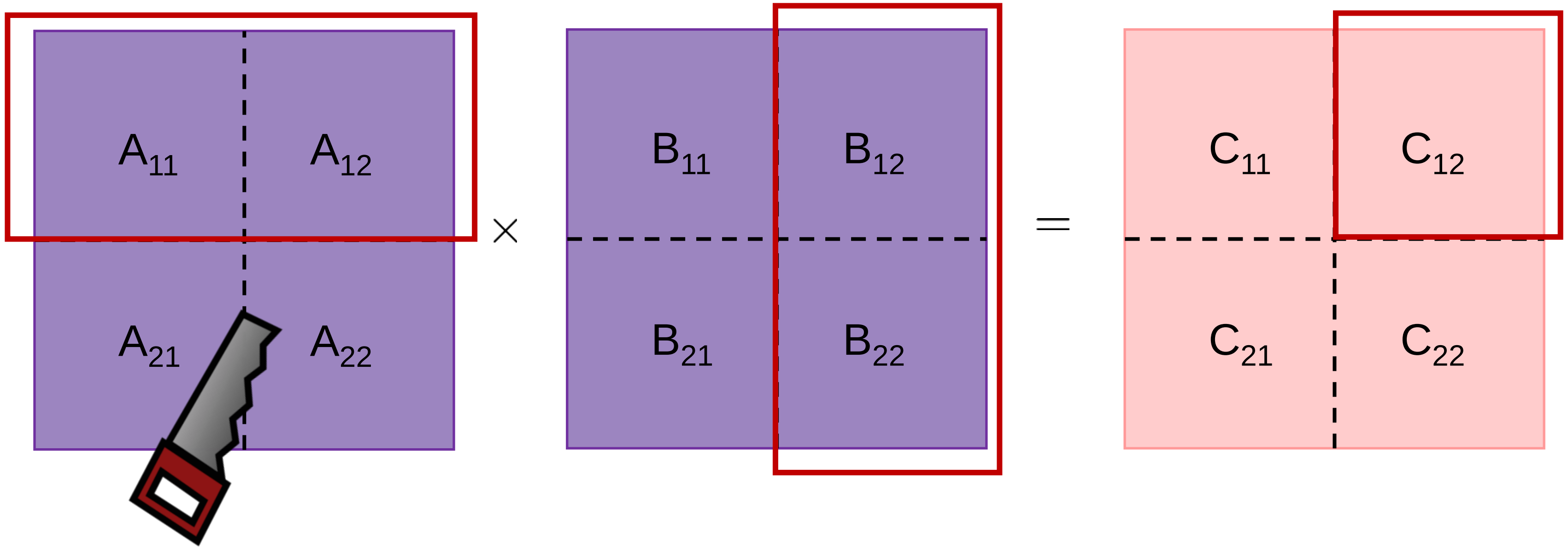
Why?

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

Divide-and-Conquer

- 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

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



Algorithm Time Complexity

```
MatrixMultiply(n, A, B)
   //base case
   if n == 1
      return AB   $\Theta(1)$
   //recursive case
   Divide A and B into n/2 by n/2 submatrices   Divide $\Theta(1)$
   $C_{11}$ = MatrixMultiply(n/2,$A_{11}$,$B_{11}$) + MatrixMultiply(n/2,$A_{12}$,$B_{21}$)
   $C_{12}$ = MatrixMultiply(n/2,$A_{11}$,$B_{12}$) + MatrixMultiply(n/2,$A_{12}$,$B_{22}$)
   $C_{21}$ = MatrixMultiply(n/2,$A_{21}$,$B_{11}$) + MatrixMultiply(n/2,$A_{22}$,$B_{21}$)
   $C_{22}$ = MatrixMultiply(n/2,$A_{21}$,$B_{12}$) + MatrixMultiply(n/2,$A_{22}$,$B_{22}$)
   return C
```

Conquer $8T(n/2)$

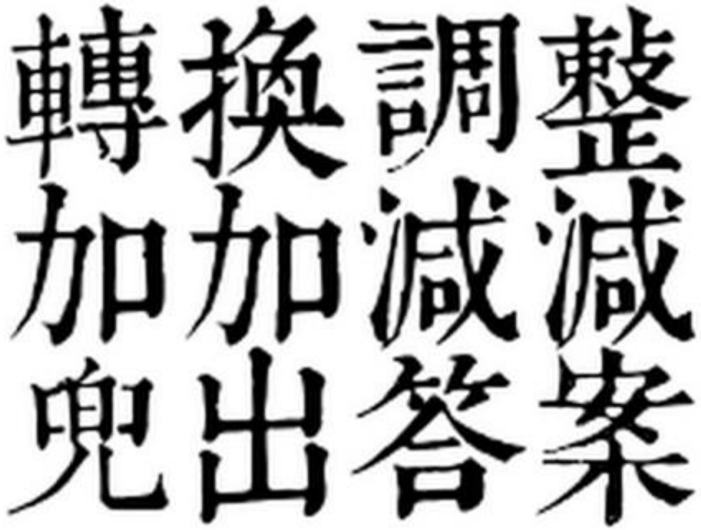
Combine $4\Theta((n/2)^2) = \Theta(n^2)$

- $T(n)$ = 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 $\Theta(n^{log_2 7}) \approx \Theta(n^{2.807})$
- 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

- $C = A \times B$

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

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

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

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

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

$\mathbf{12 + 6 - 7\times}$

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    $\Theta(1)$
  // recursive case
  Divide A and B into n/2 by n/2 submatrices
  $M_1$ = Strassen(n/2, $A_{11}+A_{22}$, $B_{11}+B_{22}$)
  $M_2$ = Strassen(n/2, $A_{21}+A_{22}$, $B_{11}$)
  $M_3$ = Strassen(n/2, $A_{11}$, $B_{12}-B_{22}$)
  $M_4$ = Strassen(n/2, $A_{22}$, $B_{21}-B_{11}$)
  $M_5$ = Strassen(n/2, $A_{11}+A_{12}$, $B_{22}$)
  $M_6$ = Strassen(n/2, $A_{11}-A_{21}$, $B_{11}+B_{12}$)
  $M_7$ = Strassen(n/2, $A_{12}-A_{22}$, $B_{21}+B_{22}$)
  $C_{11}$ = $M_1 + M_4 - M_5 + M_7$
  $C_{12}$ = $M_3 + M_5$
  $C_{21}$ = $M_2 + M_4$
  $C_{22}$ = $M_1 - M_2 + M_3 + M_6$
  return C
```

Divide $\Theta(1)$

Conquer

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

Combine

$\Theta(n^2)$

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

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

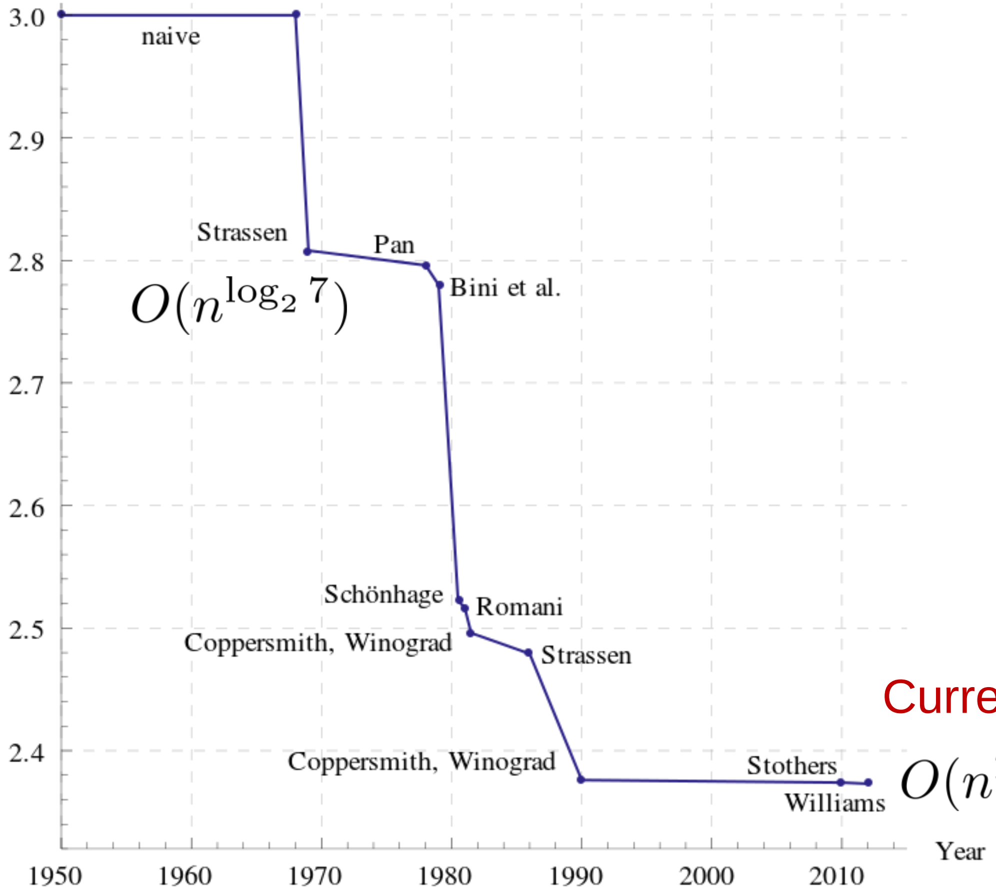
➡ $\Theta(n^{\log_2 7}) \sim \Theta(n^{2.807})$

Practicability of Strassen's Algorithm

- Disadvantages
  1. Larger constant factor than it in the naïve approach
     $$c_1 n^{\log_2 7}, c_2 n^3 \rightarrow c_1 > c_2$$
  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



$O(n^{\log_2 7})$

Current lowest upper bound

$O(n^{2.3728639})$

Matrix Multi. Problem Complexity

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

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

D&C #6: Selection Problem

Textbook Chapter 9.3 – Selection in worst-case linear time

Selection Problem

- **Input**
  - A set A of n (distinct) numbers.
  - An integer k, with $0 \leq k < n$.
- **Output**
  - The k-th largest element in A.

$n = 10, k = 4$

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

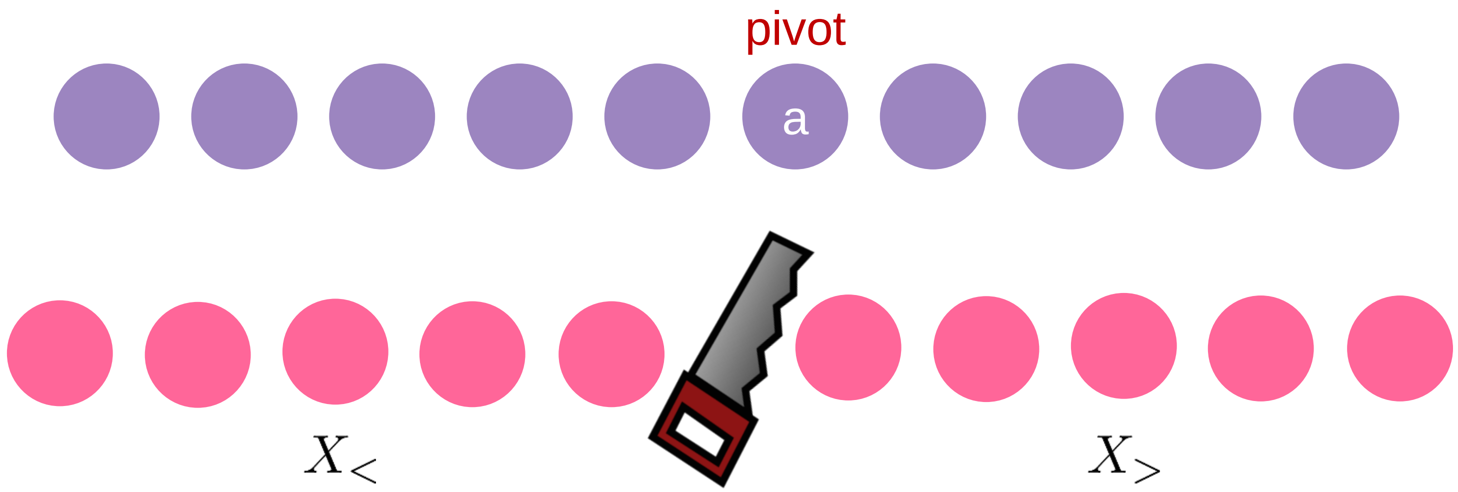
$$\text{Upper bound} = O(n \log n)$$

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

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

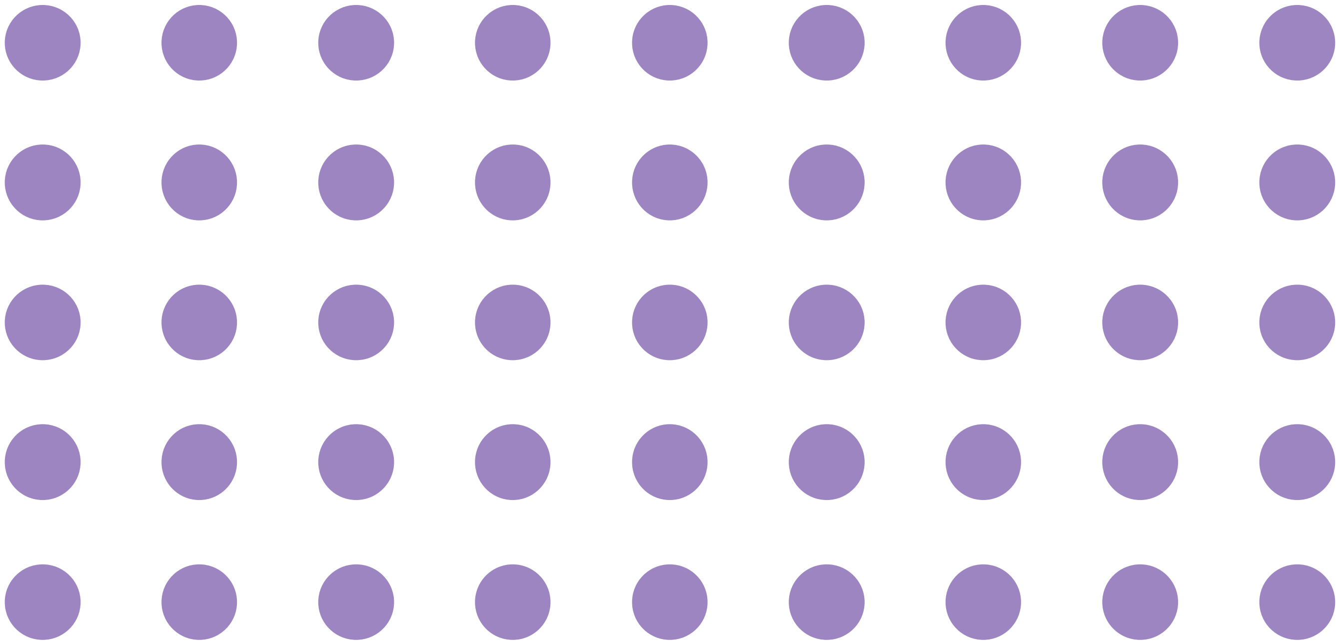
Divide-and-Conquer

- 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

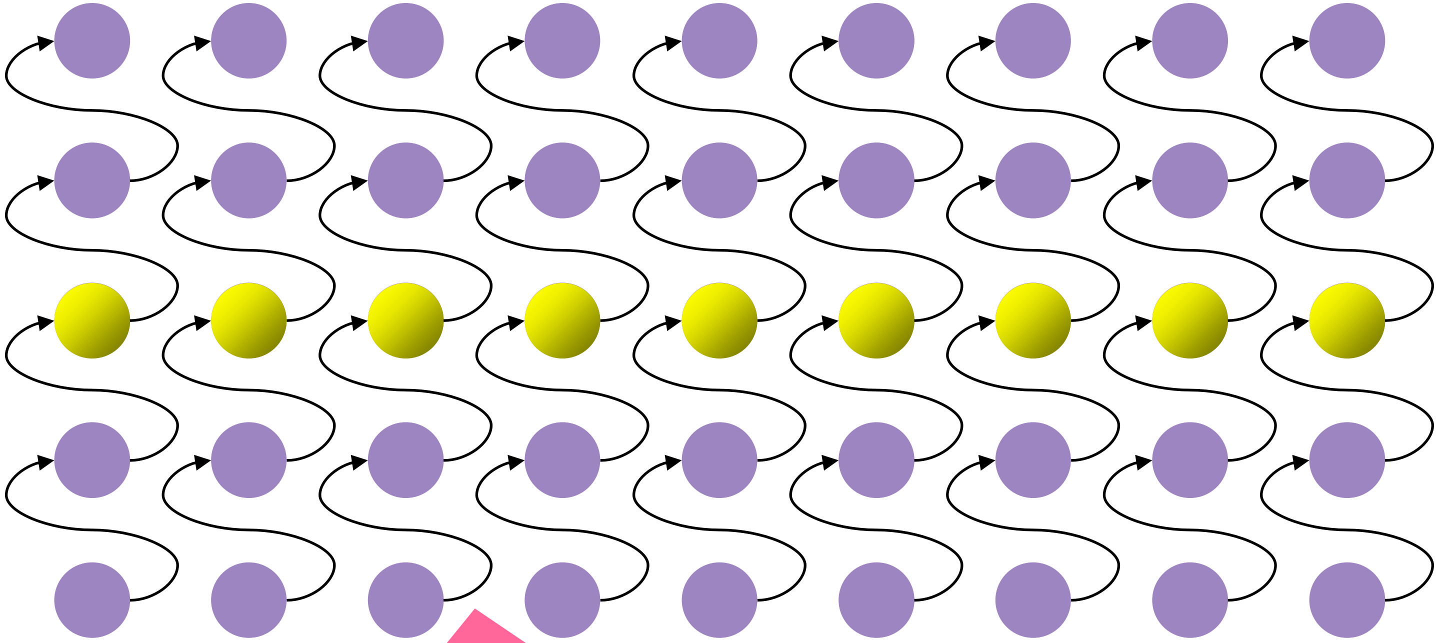


We want these subproblems to have similar size
→ The better pivot is the medium in the input array

(1) Five Guys per Group

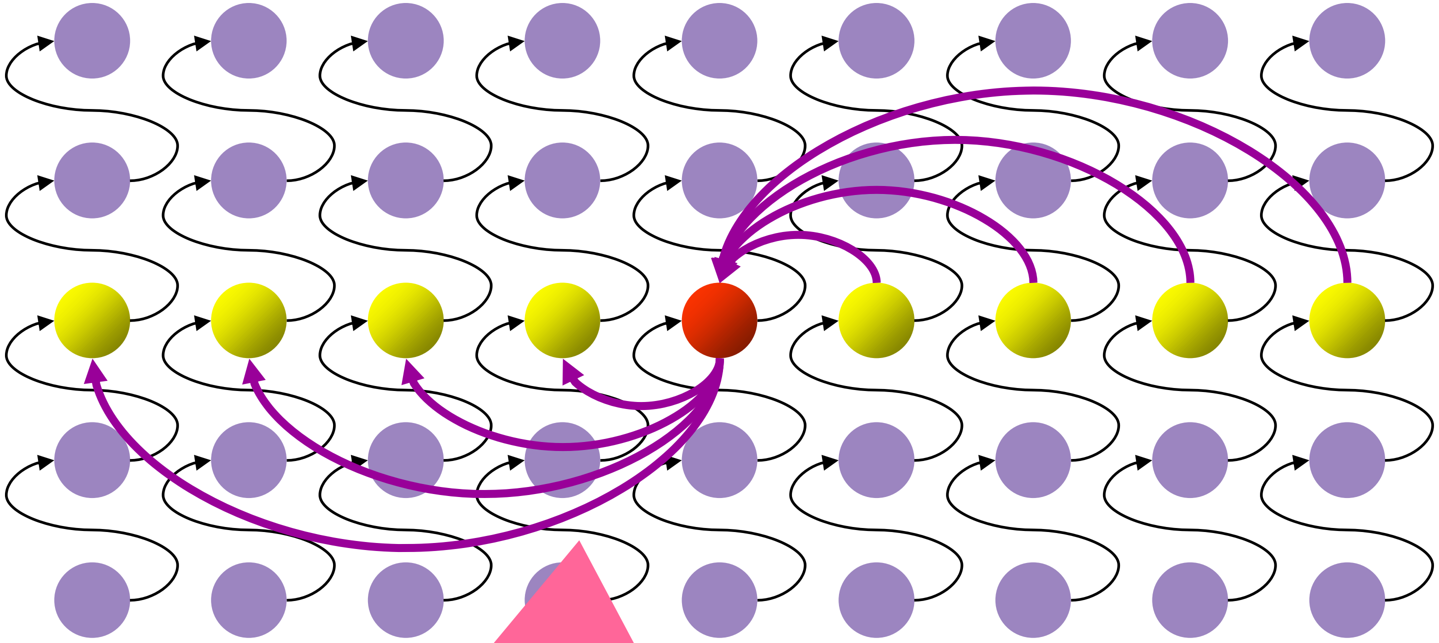


(2) A Median per Group



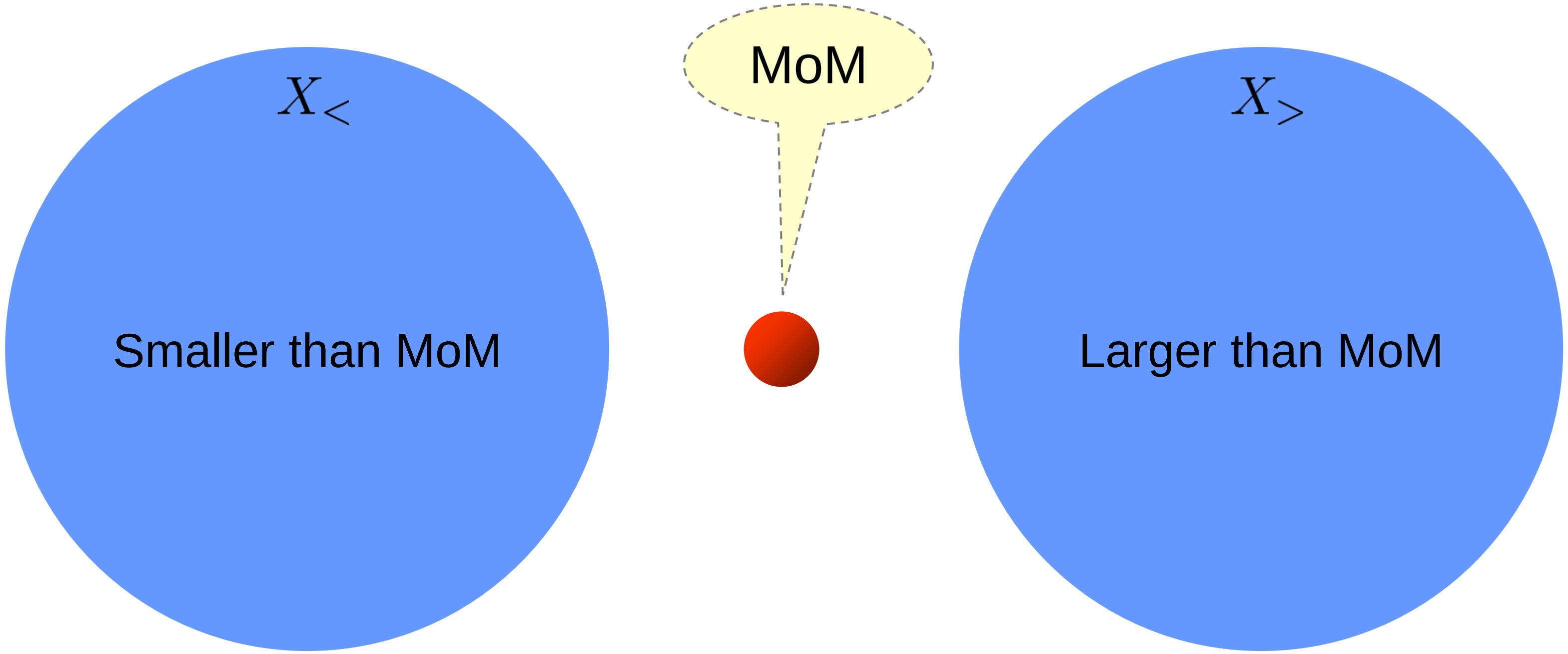
small number → large number

(3) Median of Medians (MoM)



small number → large number

(4) Partition via MoM



$X_<$

MoM

$X_>$

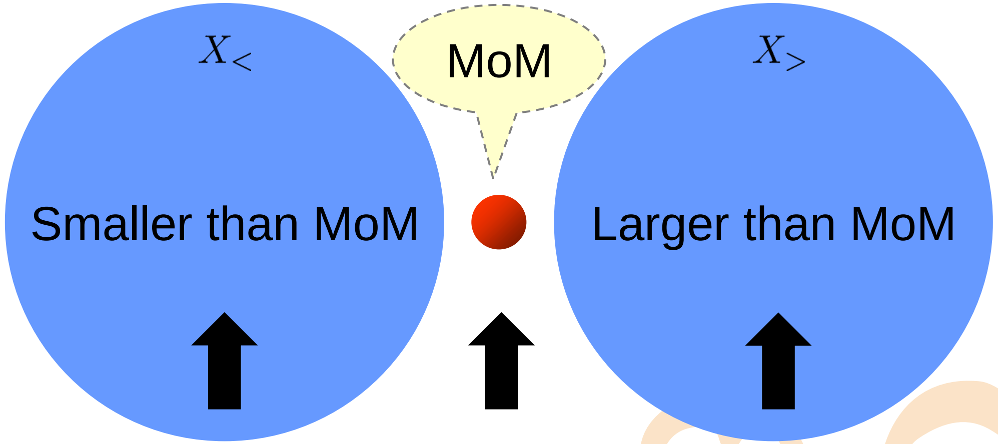
Smaller than MoM

Larger than MoM

(5) Recursion

- Three cases
  1. If $k \le |X_>|$, then output the k-th largest number in $X_>$
  2. If $k = |X_>| + 1$, then output MoM
  3. If $k > |X_>| + 1$, then output the $(k - |X_>| - 1)$-th largest number in $X_<$

- Practice to prove by induction



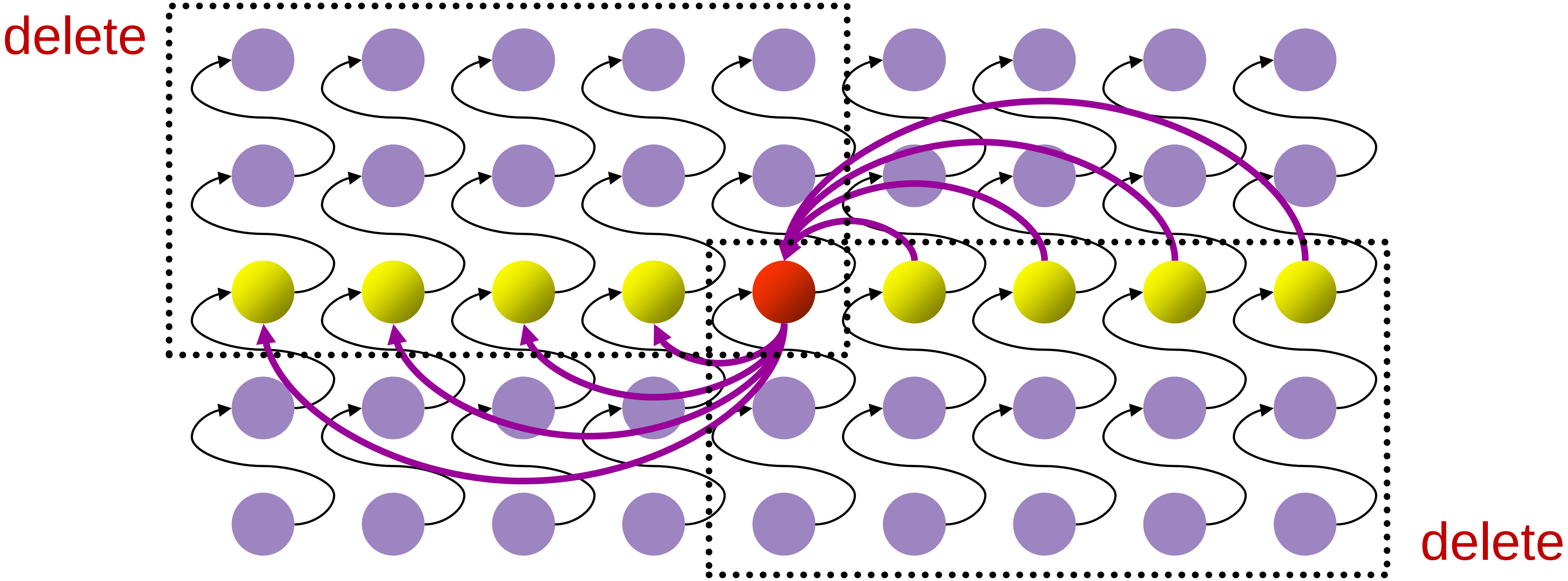
Two Recursive Steps

- Step (2): Determining MoM
- Step (5): Selection in $X_<$ or $X_>$

Divide-and-Conquer for Selection

```
Selection(X, k)
  // base case
  if |X| <= 4
    sort X and return X[k]  $\Theta(1)$
  // recursive case
  Divide X into |X|/5 groups with size 5  $\Theta(1)$
  M[i] = median from group i  $\Theta(1) \cdot \Theta(n/5) = \Theta(n)$
  MoM = Selection(M, |M|/2)  $T(n/5)$
  for i = 1 … |X|
    if X[i] > MoM
      insert X[i] into X2        $\Theta(n)$
    else
      insert X[i] into X1
  if |X2| == k - 1
    return x                     $\Theta(1)$
  if |X2| > k - 1
    return Selection(X2, k)                  $T(|X2|)$
  return Selection(X1, k - |X2| - 1)         $T(|X1|)$
```

Candidates for Consideration



- If $k \leq |X_>|$, then output the k-th largest number in $X_>$
- If $k > |X_>| + 1$, then output the $(k - |X_>| - 1)$-th largest number in $X_<$

Deleting at least $\frac{n}{5} \div 2 \times 3 = \frac{3}{10} n$ guys

D&C Algorithm Complexity

- $T(n)$ = time for running `Selection(X, k)` with |X| = n

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T\left(\frac{n}{5}\right) + \max(T(|X_>|), T(|X_<|)) + \Theta(n) & \text{if } n > 1. \end{cases}$$

➡ $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T\left(\frac{n}{5}\right) + T(\frac{7n}{10}) + \Theta(n) & \text{if } n > 1 \end{cases}$ ➡ $\Theta(n)$

- Intuition

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T\left(\frac{9n}{10}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

Theorem

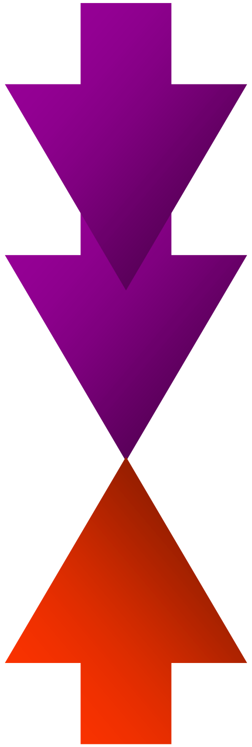
- Theorem $T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) & \text{if } n > 1 \end{cases}$ ➡ $T(n) = O(n)$
- Proof
  - There exists positive constant a, b s.t. $T(n) \leq \begin{cases} a & \text{if } n = 1 \\ T(n/5) + T(7n/10) + b \cdot n & \text{if } n \geq 2 \end{cases}$
  - Use induction to prove $T(n) \leq c \cdot n$
    - n = 1, $a > c$
    - n > 1, $T(n) \leq T(n/5) + T(7n/10) + b \cdot n$

Inductive hypothesis $\leq \frac{1}{5}cn + \frac{7}{10}cn + bn = \frac{9}{10}cn + bn = cn - (\frac{1}{10}cn - bn)$

select $c > 10b$

$\leq cn$

Selection Problem Complexity



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

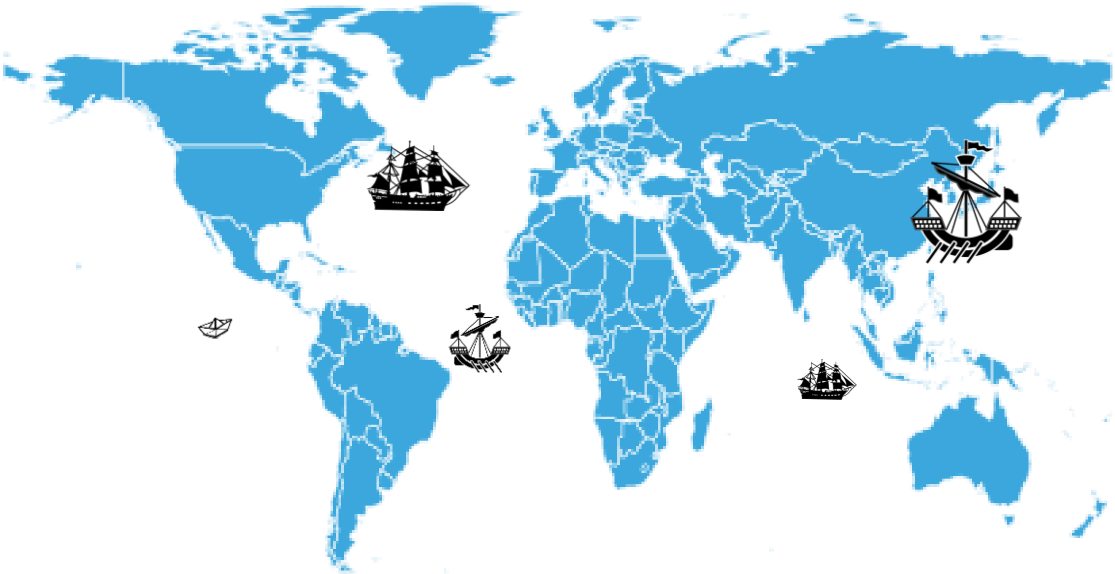
$$\text{Lower bound} = \Omega(n)$$

D&C #7: Closest Pair of Points

Textbook Chapter 33.4 – Finding the closest pair of points
Section 5.4 of Algorithm Design by Kleinberg & Tardos

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

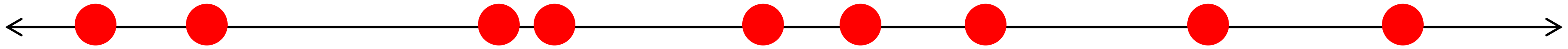


- Brute-force algorithm
  - Check all pairs of points: $\Theta(C_2^n) = \Theta(n^2)$

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

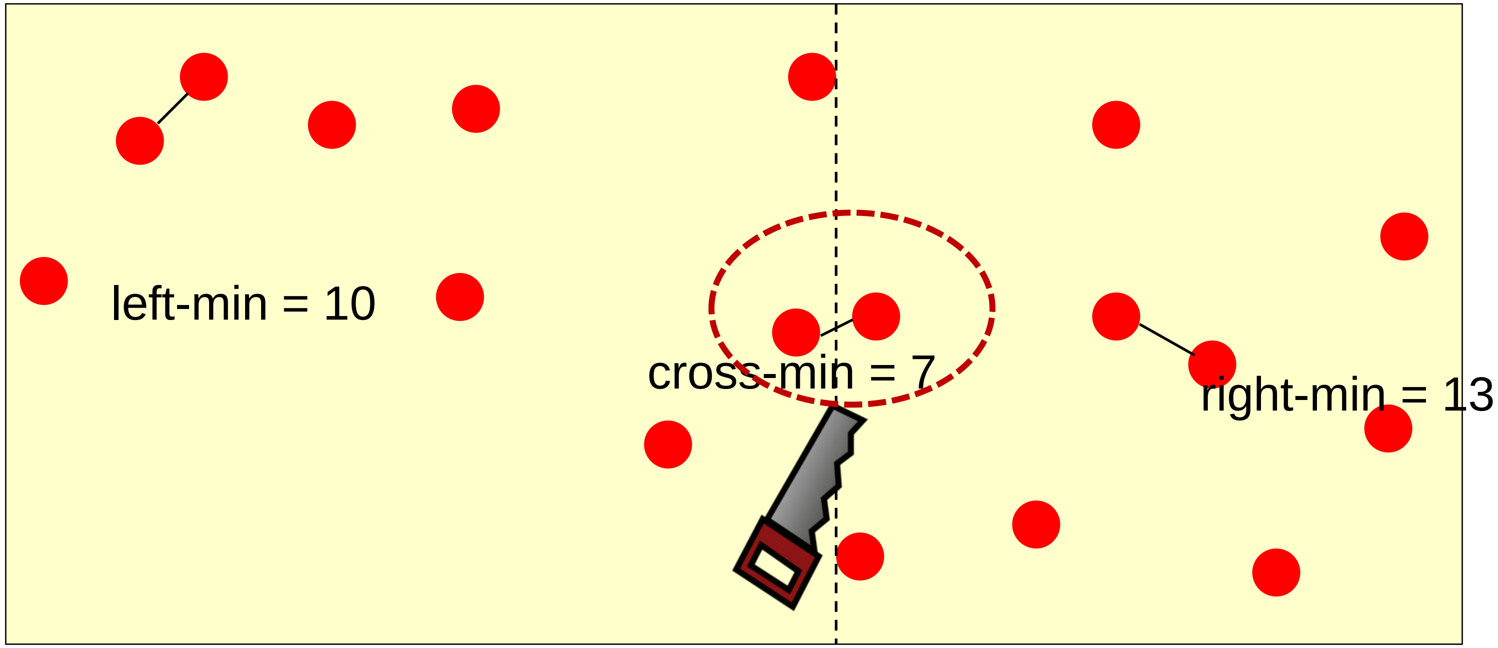
➡ $T(n) = \Theta(n \log n)$



- 2D:

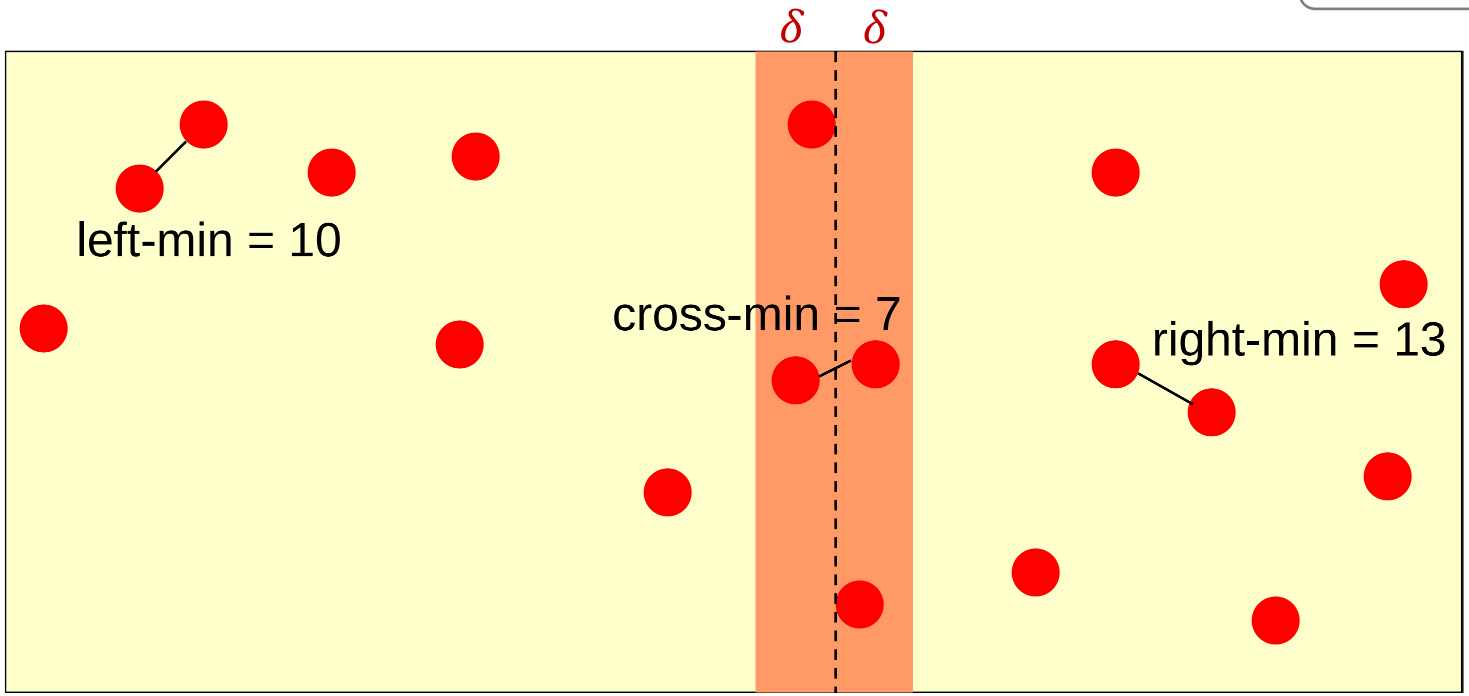
Divide-and-Conquer Algorithm

- 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



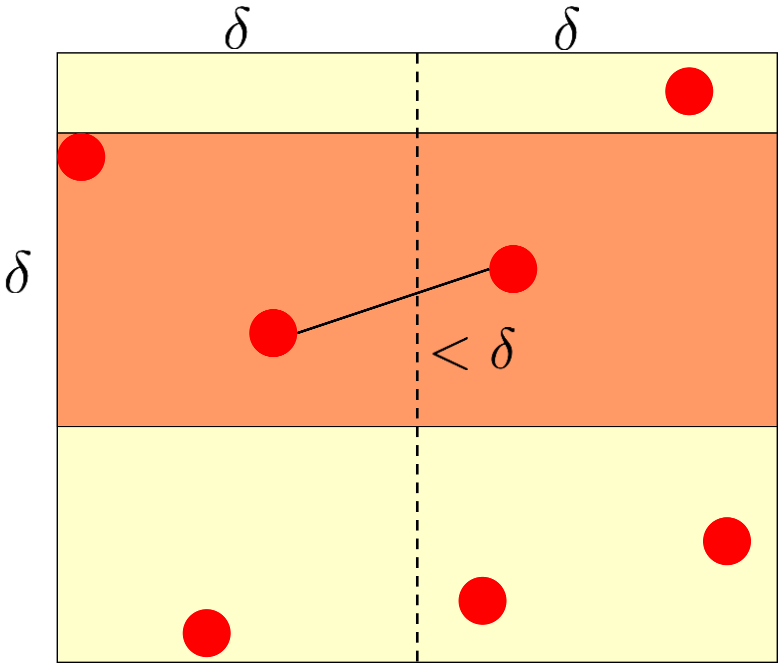
Cross Two Regions

- 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, $\delta = \min\{\text{l-min}, \text{r-min}\}$
  - 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

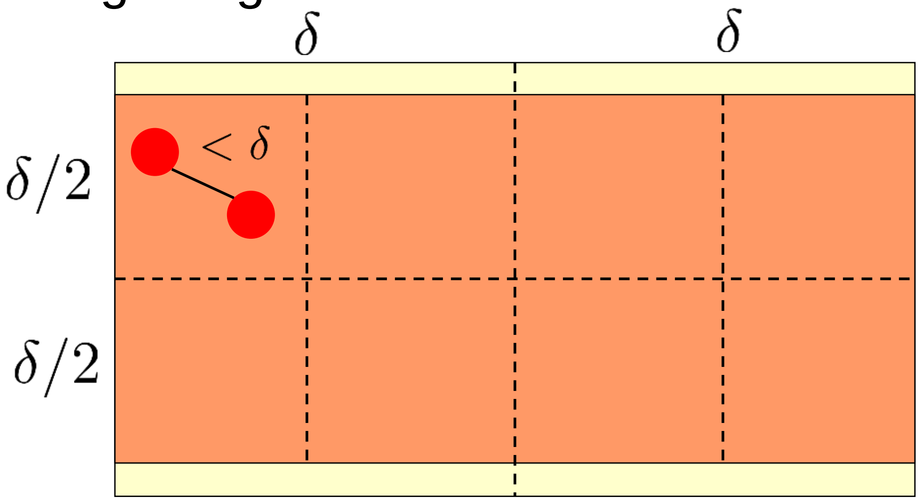


縮小搜尋範圍!

要是很倒霉，所有的點都聚集在某個$\delta \times 2\delta$區塊內怎麼辦

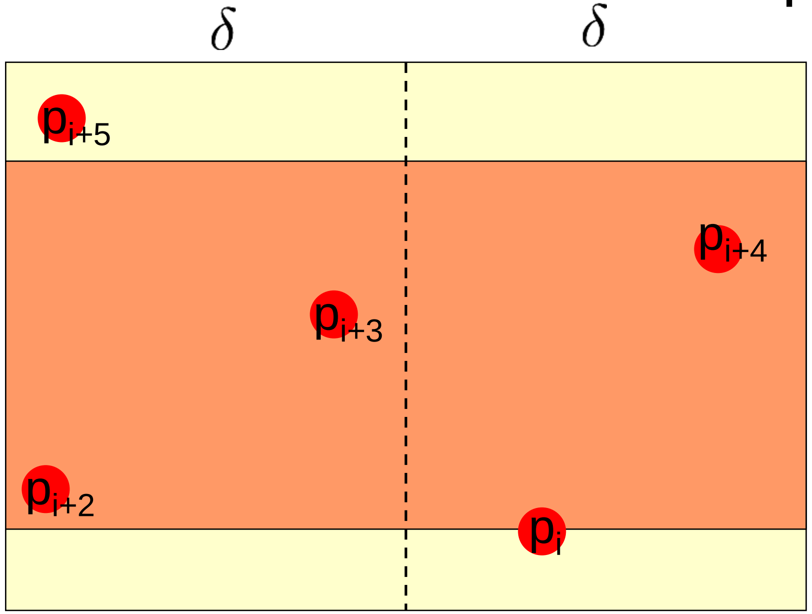
Cross Two Regions

- 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
  - 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, $\delta = \min\{\text{l−min}, \text{r−min}\}$
- 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
  - 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}, …, 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            $\Theta(1)$
  // Divide
  find a vertical line L s.t. both planes contain half of the points   $\Theta(n \log n)$
  // Conquer (by recursion)
  left-pair, left-min = Closest-Pair(points in the left)
  right-pair, right-min = Closest-Pair(points in the right)            $2T(n/2)$
  // 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 $p_0$, …, $p_k$            $\Theta(n \log n)$
  for point $p_i$:                                                      $\Theta(n)$
    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
```

- $T(n)$ = 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

```
sort P by x- and y-coordinate and store in Px and Py                     $\Theta(n \log n)$
Closest-Pair(P)
  // termination condition (base case)
  if |P| <= 3 brute-force finding closest pair and return it             $\Theta(1)$
  // Divide
  find a vertical line L s.t. both planes contain half of the points     $\Theta(n)$
  // 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                                   $\Theta(n)$
    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
```

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

Slides modified from Prof. Hsu-Chun Hsiao

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