Algorithm Design and Analysis

Divide & Conquer (2)

Oct 5th, 2017

YUN-NUNG (VIVIAN) CHEN

HTTP://ADA17.CSIE.ORG

國立臺灣大學
National Taiwan University

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

Announcement

- Mini-HW 3 released
  - Due on 10/12 (Thu) 17:20
  - Print out the A4 hard copy and submit before the lecture finishes
- Homework 1 released
  - Due on 10/19 (Thur) 17:20 (2 weeks left)
  - Writing: print out the A4 hard copy and submit before the lecture finishes
  - Programming: submit to Online Judge – http://ada-judge.csie.org
- Mid-term date changed
  - Original: 11/09 (Thu)
  - New: 11/16 (Thu)

Mini-HW 3

Mini HW #3

Due Time: 2017/10/12 (Thu.) 17:20

Contact TAs: `ada-ta@csie.ntu.edu.tw`

Read the code below and prove the time complexity of f(1, N, 1, M) is $O(M \log_2 N)$.

```
1  void f( int l , int r , int bl , int br ){
2      if( l > r ) return;
3      int mid = (l + r) / 2;
4      int bmid = bl;
5      for( int i = bl + 1 ; i <= br ; i ++ )
6          if( rand() % ( i - bl + 1 ) == 0 )
7              bmid = i;
8      f( l , mid - 1 , bl , bmid );
9      f( mid + 1 , r , bmid , br );
10 }
11 f( 1 , N , 1 , M );
```

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
- 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 首部曲

Divide-and-Conquer 之神乎奇技

What is Divide-and-Conquer?

- Solve a problem recursively
- Apply three steps at each level of the recursion
  1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem (比較小的同樣問題)
  2. **Conquer** the subproblems by solving them recursively
     If the subproblem sizes are *small enough*
     - then solve the subproblems — base case
     - else recursively solve itself — recursive case
  3. **Combine** the solutions to the subproblems into the solution for the original problem

Solving Recurrences

Textbook Chapter 4.3 – The substitution method for solving recurrences

Textbook Chapter 4.4 – The recursion-tree method for solving recurrences

Textbook Chapter 4.5 – The master method for solving recurrences

D&C Algorithm Time Complexity

- $T(n)$: running time for input size n
- $D(n)$: time of Divide for input size n
- $C(n)$: time of Combine for input size n
- a: number of subproblems
- n/b: size of each subproblem

$$T(n) = \begin{cases} O(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Solving Recurrences

1. **Substitution Method** (取代法)
   - Guess a bound and then prove by induction
2. **Recursion-Tree Method** (遞迴樹法)
   - Expand the recurrence into a tree and sum up the cost
3. **Master Method** (套公式大法/大師法)
   - Apply Master Theorem to a specific form of recurrences

- Useful simplification tricks
  - Ignore floors, ceilings, boundary conditions (proof in Ch. 4.6)
  - Assume base cases are constant (for small *n*)

Substitution Method

Textbook Chapter 4.3 – The substitution method for solving recurrences

Review

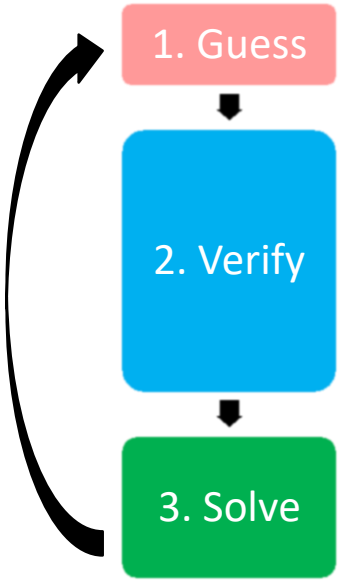
- Time Complexity for Merge Sort
- Theorem

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(n \log n)$$

- Proof
  - There exists positive constant a, b s.t. $T(n) \leq \begin{cases} a & \text{if } n = 1 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$
  - Use induction to prove $T(n) \leq b \cdot n \log n + a \cdot n$
    - n = 1, trivial
    - n > 1, $$\begin{aligned} T(n) &\leq 2T(n/2) + bn \\ &\leq 2[b \cdot \frac{n}{2} \log \frac{n}{2} + a \cdot \frac{n}{2}] + b \cdot n \\ &= b \cdot n \log n - b \cdot n + a \cdot n + b \cdot n \\ &= b \cdot n \log n + a \cdot n \end{aligned}$$

Substitution Method (取代法)
guess a bound and then prove by induction

Substitution Method (取代法)



- Guess the form of the solution
- Verify by mathematical induction (數學歸納法)
  - Prove it works for $n = 1$
  - Prove that if it works for $n = m$, then it works for $n = m + 1$

  → It can work for all positive integer n
- Solve constants to show that the solution works
- Prove O and Ω separately

Substitution Method Example

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

- Proof
  - $T(n) = O(n^3)$
    There exists positive constants n_0, c s.t. for all $n \geq n_0$, $T(n) \leq cn^3$ — Guess
  - Use induction to find the constants n_0, c — Verify
    - n = 1, trivial
    - n > 1,

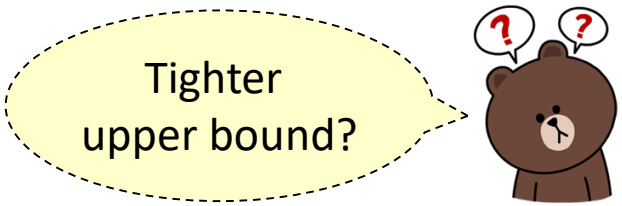
$$\begin{aligned} T(n) &\leq 4T(n/2) + bn \\ &\leq 4c(n/2)^3 + bn \quad \text{(Inductive hypothesis)} \\ &= cn^3/2 + bn \\ &= cn^3 - (cn^3/2 - bn) \\ &\leq cn^3 \end{aligned}$$

$cn^3/2 - bn \geq 0$ e.g. $c \geq 2b, n \geq 1$

  - $T(n) \leq cn^3$ holds when $c = 2b, n_0 = 1$ — Solve

Substitution Method Example

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$



- Proof
  - $T(n) = O(n^2)$
    There exists positive constants n_0, c s.t. for all $n \geq n_0$, $T(n) \leq cn^2$
  - Use induction to find the constants n_0, c
    - n = 1, trivial
    - n > 1,

$$\begin{aligned} T(n) &\leq 4T(n/2) + bn \\ &\leq 4c(n/2)^2 + bn \\ &= cn^2 + bn \end{aligned}$$

Inductive hypothesis

証不出來...
猜錯了？還是推導錯了？

沒猜錯 推導也沒錯
這是取代法的小盲點

Substitution Method Example

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(n/2) + O(n) & \text{if } n \geq 2 \end{cases}$$

Strengthen the inductive hypothesis by subtracting a low-order term

- Proof
  - $T(n) = O(n^2)$
    There exists positive constants n_0, c_1, c_2 s.t. for all $n \geq n_0, T(n) \leq c_1 n^2 - c_2 n$
  - Use induction to find the constants n_0, c_1, c_2
    - n = 1, $T(1) \leq c_1 - c_2$ holds for $c_1 \geq c_2 + 1$
    - n > 1,

$$\begin{aligned} T(n) &\leq 4T(n/2) + bn \\ &\leq 4[c_1(n/2)^2 - c_2(n/2)] + bn \\ &= c_1 n^2 - 2c_2 n + bn \\ &= c_1 n^2 - c_2 n - (c_2 n - bn) \\ &\leq c_1 n^2 - c_2 n \end{aligned}$$

Inductive hypothesis

$c_2 n - bn \geq 0$
e.g. $c_2 \geq b, n \geq 0$

  - $T(n) \leq c_1 n^2 - c_2 n$ holds when $c_1 = b + 1, c_2 = b, n_0 = 0$

Guess

Verify

Solve

Useful Tricks

- Guess based on seen recurrences
- Use the recursion-tree method
- From loose bound to tight bound
- Strengthen the inductive hypothesis by subtracting a low-order term
- Change variables
  - E.g., $T(n) = 2T(\sqrt{n}) + \log n$
  1. Change variable: $k = \log n, n = 2^k \rightarrow T(2^k) = 2T(2^{k/2}) + k$
  2. Change variable again: $S(k) = T(2^k) \rightarrow S(k) = 2S(k/2) + k$
  3. Solve recurrence

$S(k) = \Theta(k \log k) \rightarrow T(2^k) = \Theta(k \log k) \rightarrow T(n) = \Theta(\log n \log \log n)$

Recursion-Tree Method

Textbook Chapter 4.4 – The recursion-tree method for solving recurrences

Review

- Time Complexity for Merge Sort
- Theorem

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(n \log n)$$

- Proof

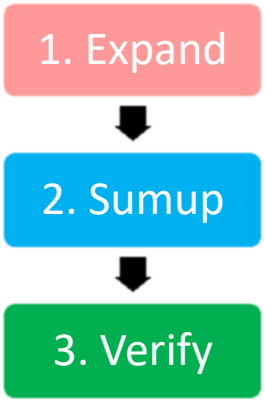
Recursion-Tree Method (遞迴樹法)
Expand the recurrence into a tree and sum up the cost

$$\begin{aligned} T(n) &\leq 2T(\frac{n}{2}) + cn \\ &\leq 2[2T(\frac{n}{4}) + c\frac{n}{2}] + cn = 4T(\frac{n}{4}) + 2cn \quad \text{1st expansion} \\ &\leq 4[2T(\frac{n}{8}) + c\frac{n}{4}] + 2cn = 8T(\frac{n}{8}) + 3cn \quad \text{2nd expansion} \\ &\vdots \\ &\leq 2^k T(\frac{n}{2^k}) + kcn \quad \text{kth expansion} \end{aligned}$$

The expansion stops when $2^k = n$

$$\begin{aligned} T(n) &\leq nT(1) + cn \log_2 n \\ &= O(n) + O(n \log n) \\ &= O(n \log n) \end{aligned}$$

Recursion-Tree Method (遞迴樹法)



- Expand a recurrence into a tree
- Sum up the cost of all nodes as a good guess
- Verify the guess as in the substitution method

- Advantages
  - Promote intuition
  - Generate good guesses for the substitution method

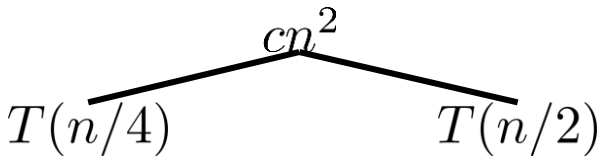
Recursion-Tree Example

$$T(n) = T(n/4) + T(n/2) + cn^2$$

$$T(n)$$

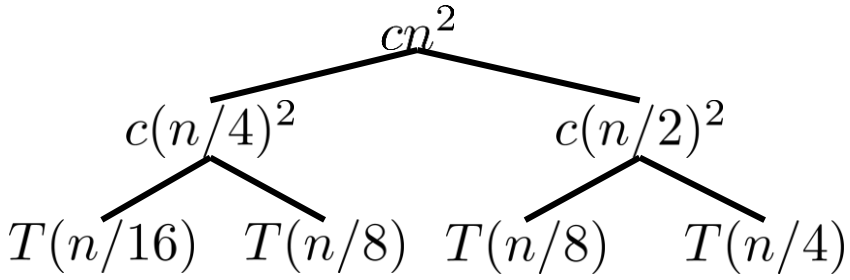
Recursion-Tree Example

$$T(n) = T(n/4) + T(n/2) + cn^2$$



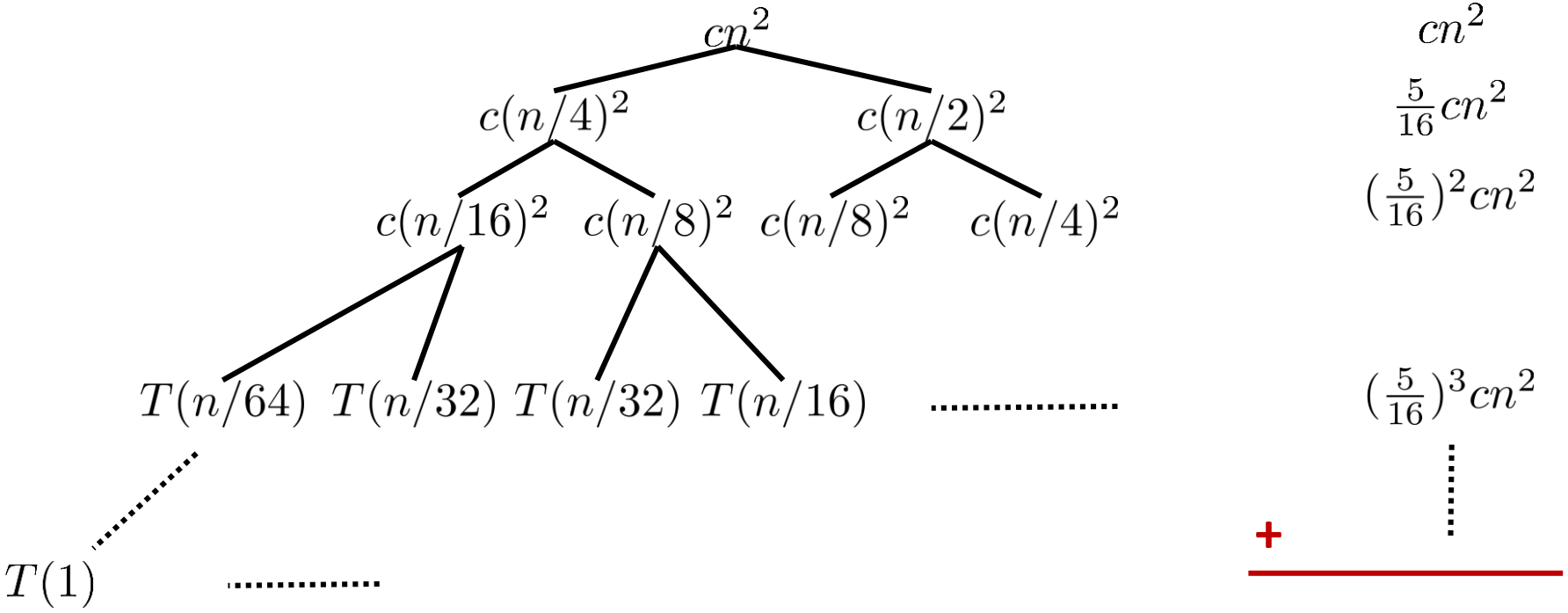
Recursion-Tree Example

$$T(n) = T(n/4) + T(n/2) + cn^2$$



Recursion-Tree Example

$$T(n) = T(n/4) + T(n/2) + cn^2$$



$$T(n) \leq (1 + \frac{5}{16} + (\frac{5}{16})^2 + (\frac{5}{16})^3 + \cdots)cn^2 = \frac{1}{1 - \frac{5}{16}}cn^2 = \frac{16}{11}cn^2 = O(n^2)$$

Master Theorem

Textbook Chapter 4.5 – The master method for solving recurrences

Master Theorem

divide a problem of size n into a subproblems
each of size $\frac{n}{b}$ is solved in time $T\left(\frac{n}{b}\right)$ recursively

The proof is in Ch. 4.6

Let $T(n)$ be a positive function satisfying the following recurrence relation

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{if } n > 1, \end{cases}$$

Should follow this format

where $a \geq 1$ and $b > 1$ are constants.

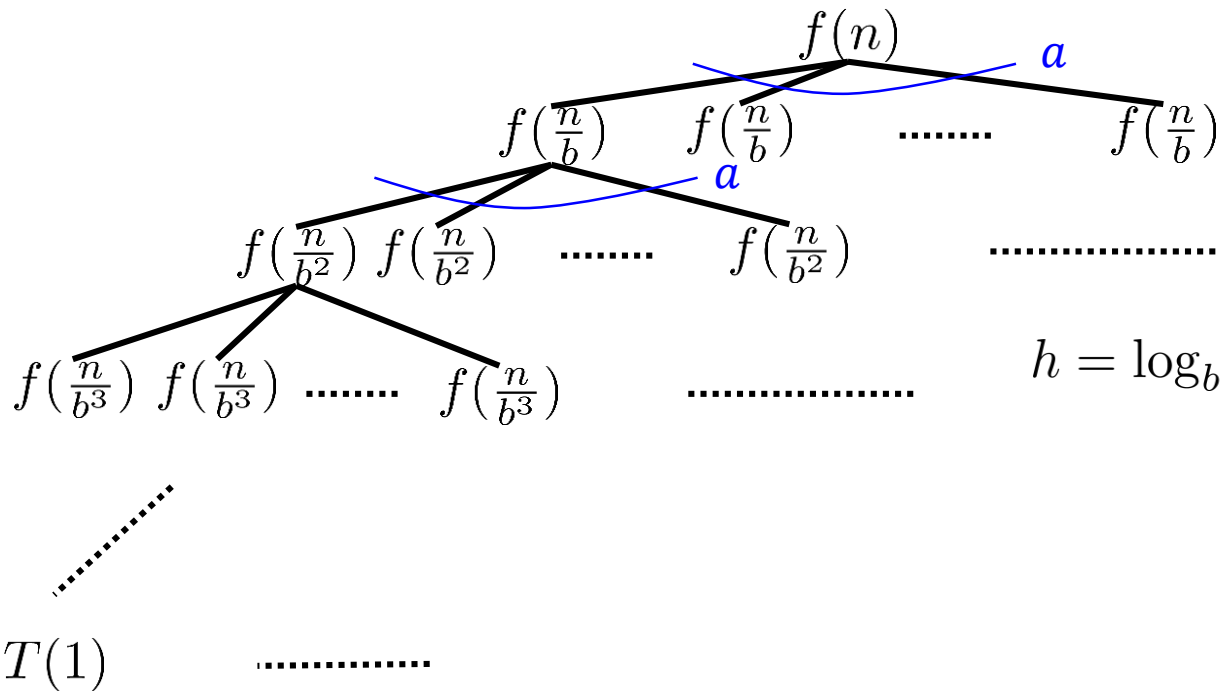
- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

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

compare $f(n)$ with $n^{\log_b a}$

Recursion-Tree for Master Theorem

$$T(n) = aT(\tfrac{n}{b}) + f(n)$$



$h = \log_b n + 1$

$f(n)$

$af(\frac{n}{b})$

$a^2 f(\frac{n}{b^2})$

$a^3 f(\frac{n}{b^3})$

$+ \; a^{\log_b n} T(1)$

$$T(n) = f(n) + af(\tfrac{n}{b}) + a^2 f(\tfrac{n}{b^2}) + a^3 f(\tfrac{n}{b^3}) + \cdots + a^{\log_b n} T(1)$$

$$a^{\log_b n} T(1) = n^{\log_b a} T(1)$$

Three Cases

- $T(n) = aT(\frac{n}{b}) + f(n)$
  - $a \geq 1$, the number of subproblems
  - $b > 1$, the factor by which the subproblem size decreases
  - $f(n)$ = work to divide/combine subproblems

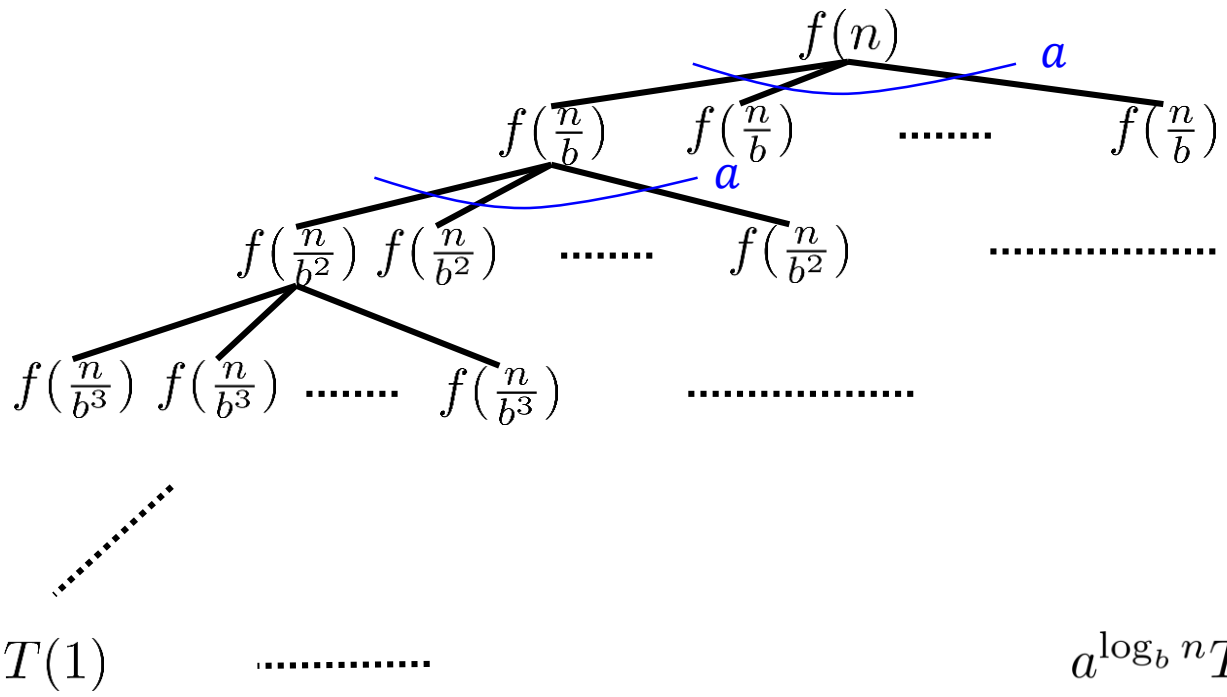
$$T(n) = f(n) + af(\tfrac{n}{b}) + a^2 f(\tfrac{n}{b^2}) + a^3 f(\tfrac{n}{b^3}) + \cdots + n^{\log_b a} T(1)$$

- Compare $f(n)$ with $n^{\log_b a}$
  1. Case 1: $f(n)$ grows polynomially slower than $n^{\log_b a}$
  2. Case 2: $f(n)$ and $n^{\log_b a}$ grow at similar rates
  3. Case 3: $f(n)$ grows polynomially faster than $n^{\log_b a}$

Case 1:

Total cost dominated by the leaves

$$T(n) = 9T(\tfrac{n}{3}) + n, T(1) = 1$$



$$f(n) = n$$

$$af(\tfrac{n}{b}) = \tfrac{9}{3}n$$

$$a^2 f(\tfrac{n}{b^2}) = (\tfrac{9}{3})^2 n$$

$$a^3 f(\tfrac{n}{b^3}) = (\tfrac{9}{3})^3 n$$

$$a^{\log_b n} T(1) = 9^{\log_3 n} = (\tfrac{9}{3})^{\log_3 n} n$$

$f(n)$ grows polynomially slower than $n^{\log_b a}$

Case 1:
Total cost dominated by the leaves

$$T(n) = 9T(\tfrac{n}{3}) + n, T(1) = 1$$

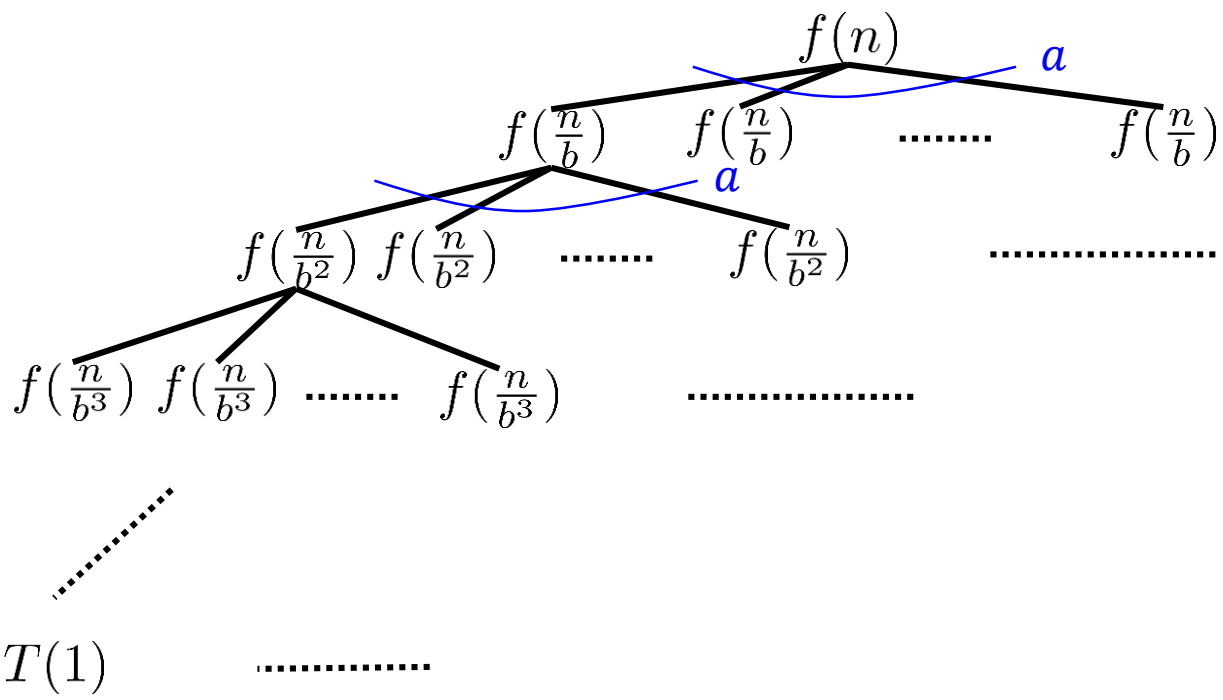
$$\begin{aligned}
T(n) &= (1 + \frac{9}{3} + (\frac{9}{3})^2 + \cdots + (\frac{9}{3})^{\log_3 n})n \\
&= \frac{(\frac{9}{3})^{1+\log_3 n} - 1}{3-1} n \\
&= \frac{3n}{2} \cdot \frac{9^{\log_3 n}}{3^{\log_3 n}} - \frac{1}{2}n \\
&= \frac{3n}{2} \cdot \frac{n^{\log_3 9}}{n} - \frac{1}{2}n \\
&= \Theta(n^{\log_3 9}) = \Theta(n^2)
\end{aligned}$$

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

Case 2:

Total cost evenly distributed among levels

$$T(n) = T(\tfrac{2n}{3}) + 1, T(1) = 1$$



$f(n) = 1$

$af(\frac{n}{b}) = 1$

$a^2 f(\frac{n}{b^2}) = 1$

$a^3 f(\frac{n}{b^3}) = 1$

$a^{\log_b n} T(1) = 1$

$f(n)$ and $n^{\log_b a}$ grow at similar rates

Case 2:

Total cost evenly distributed among levels

$$T(n) = T(\tfrac{2n}{3}) + 1, T(1) = 1$$

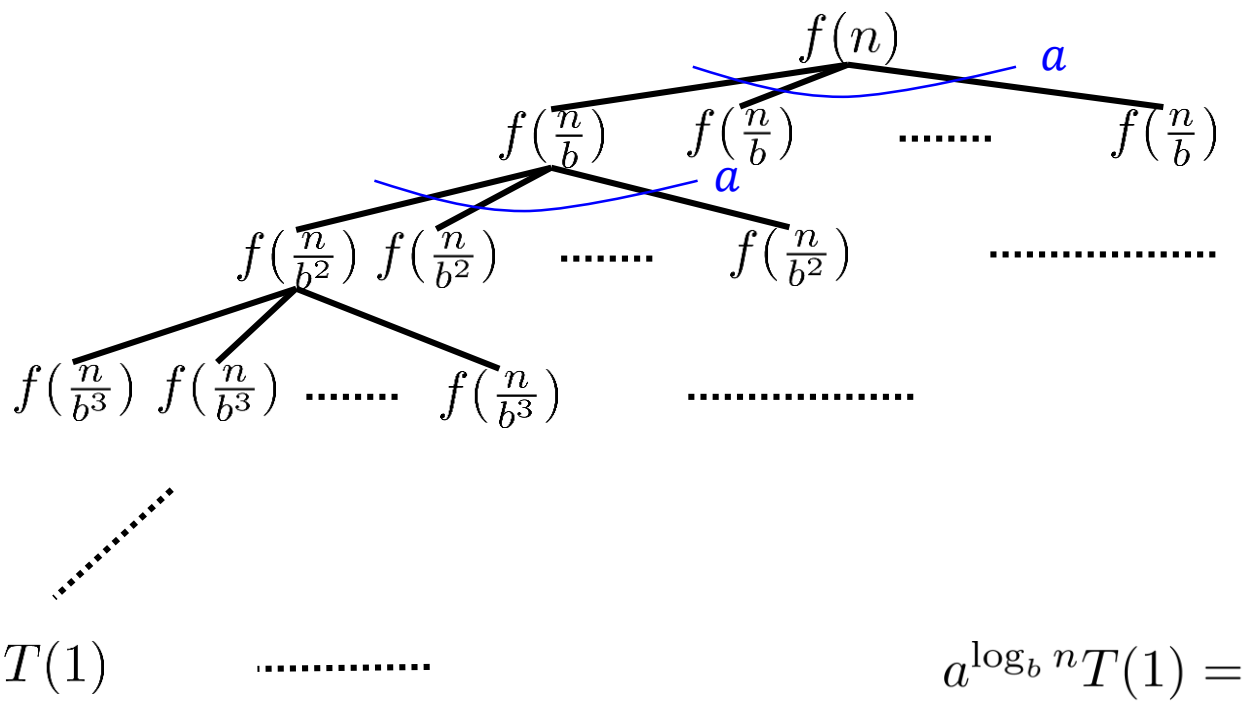
$$\begin{aligned} T(n) &= 1 + 1 + 1 + \cdots + 1 \\ &= \log_{\frac{3}{2}} n + 1 \\ &= \Theta(\log n) \end{aligned}$$

- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Case 3:

Total cost dominated by root cost

$$T(n) = 3T(\tfrac{n}{4}) + n^5, T(1) = 1$$



$$f(n) = n^5$$

$$af(\tfrac{n}{b}) = 3(\tfrac{n}{4})^5$$

$$a^2 f(\tfrac{n}{b^2}) = 3^2(\tfrac{n}{4^2})^5$$

$$a^3 f(\tfrac{n}{b^3}) = 3^3(\tfrac{n}{4^3})^5$$

$$a^{\log_b n} T(1) = 3^{\log_4 n} = 3^{\log_4 n}(\tfrac{n}{4^{\log_4 n}})^5$$

$f(n)$ grows polynomially faster than $n^{\log_b a}$

Case 3:
Total cost dominated by root cost

$$T(n) = 3T(\tfrac{n}{4}) + n^5, T(1) = 1$$

$$T(n) = (1 + \tfrac{3}{4^5} + (\tfrac{3}{4^5})^2 + \cdots + (\tfrac{3}{4^5})^{\log_4 n})n^5$$

$$T(n) > n^5$$

$$T(n) \leq \tfrac{1}{1-\frac{3}{4^5}} n^5$$

$$T(n) = \Theta(n^5)$$

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

Master Theorem

divide a problem of size n into a subproblems
each of size $\frac{n}{b}$ is solved in time $T\left(\frac{n}{b}\right)$ recursively

The proof is in Ch. 4.6

Let $T(n)$ be a positive function satisfying the following recurrence relation

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{if } n > 1, \end{cases}$$

where $a \geq 1$ and $b > 1$ are constants.

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- 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 $T(n) = \Theta(f(n))$.

compare $f(n)$ with $n^{\log_b a}$

Examples

compare $f(n)$ with $n^{\log_b a}$

- Case 1: If $T(n) = 9 \cdot T(n/3) + n$, then $T(n) = \Theta(n^2)$.

  Observe that $n = O(n^2) = O(n^{\log_3 9})$.

- Case 2: If $T(n) = T(2n/3) + 1$, then $T(n) = \Theta(\log n)$.

  Observe that $1 = \Theta(n^0) = \Theta(n^{\log_{3/2} 1})$.

- Case 3: If $T(n) = 3 \cdot T(n/4) + n^5$, then $T(n) = \Theta(n^5)$.
  - $n^5 = \Omega(n^{\log_4 3+\epsilon})$ with $\epsilon = 0.00001$.
  - $3(\frac{n}{4})^5 \leq cn^5$ with $c = 0.99999$.

Floors and Ceilings

- Master theorem can be extended to recurrences with floors and ceilings
- The proof is in the Ch. 4.6

$$T(n) = aT(\lceil \frac{n}{b} \rceil) + f(n)$$

$$T(n) = aT(\lfloor \frac{n}{b} \rfloor) + f(n)$$

Theorem 1

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- 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 $T(n) = \Theta(f(n))$.

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(n \log n)$$

- Case 2

$$f(n) = \Theta(n) = \Theta(n^1) = \Theta(n^{\log_2 2}) = \Theta(n^{\log_a b})$$

$$T(n) = \Theta(f(n) \log n) = O(n \log n)$$

Theorem 2

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- 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 $T(n) = \Theta(f(n))$.

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

- Case 1

$$f(n) = O(1) = O(n) = O(n^{\log_2 2}) = O(n^{\log_a b})$$

$$T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$$

Theorem 3

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- 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 $T(n) = \Theta(f(n))$.

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ T(n/2) + O(1) & \text{if } n \geq 2 \end{cases} \Rightarrow T(n) = O(\log n)$$

- Case 2

$$f(n) = \Theta(1) = \Theta(n^0) = \Theta(n^{\log_2 1}) = \Theta(n^{\log_a b})$$

$$T(n) = \Theta(f(n) \log n) = O(\log n)$$

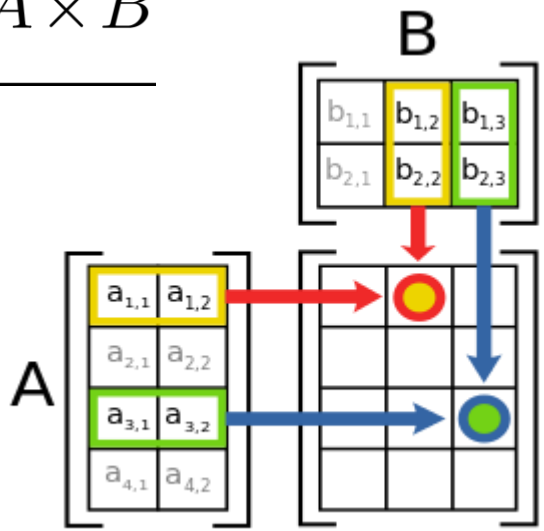
D&C #5: Matrix Multiplication

Textbook Chapter 4.2 – Strassen's algorithm for matrix multiplication

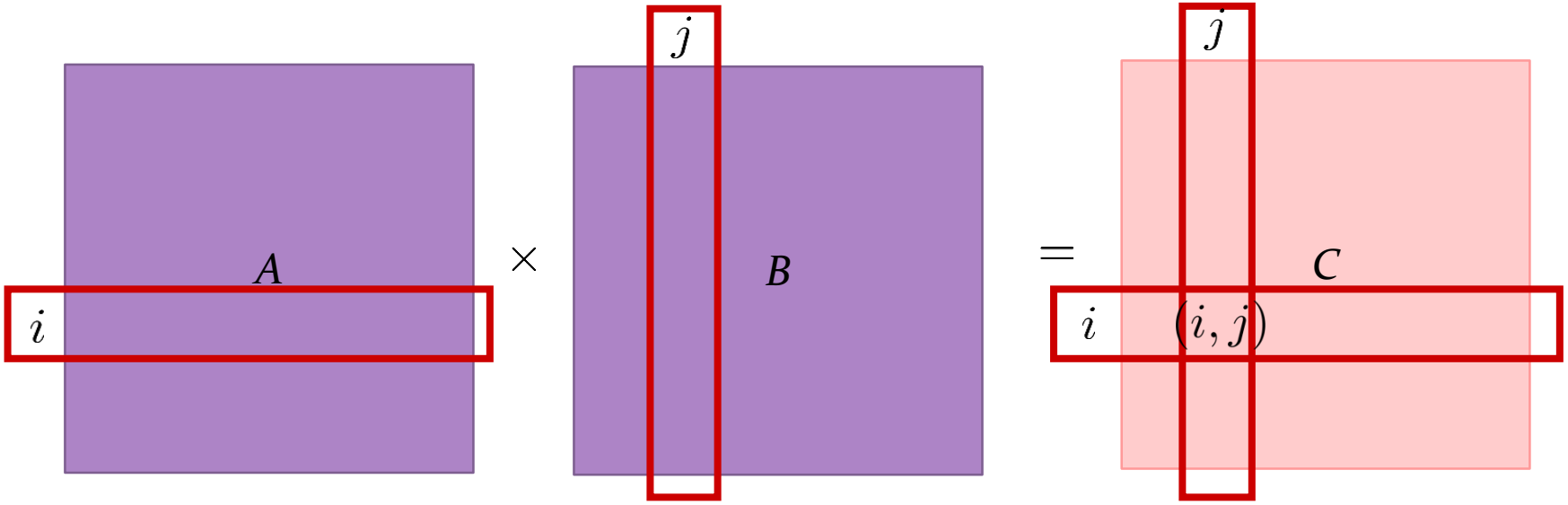
Matrix Multiplication Problem

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

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



Naïve Algorithm

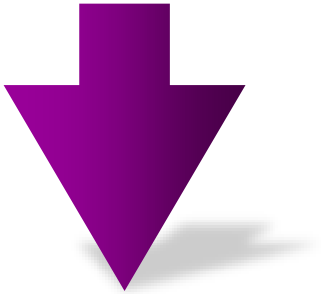


$$C(i,j) = \sum_{k=1}^{n} A(i,k) \cdot B(k,j)$$

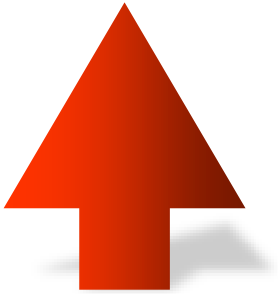
- Each entry takes n multiplications
- There are total n^2 entries

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

Matrix Multi. Problem Complexity



Upper bound $= O(n^3)$



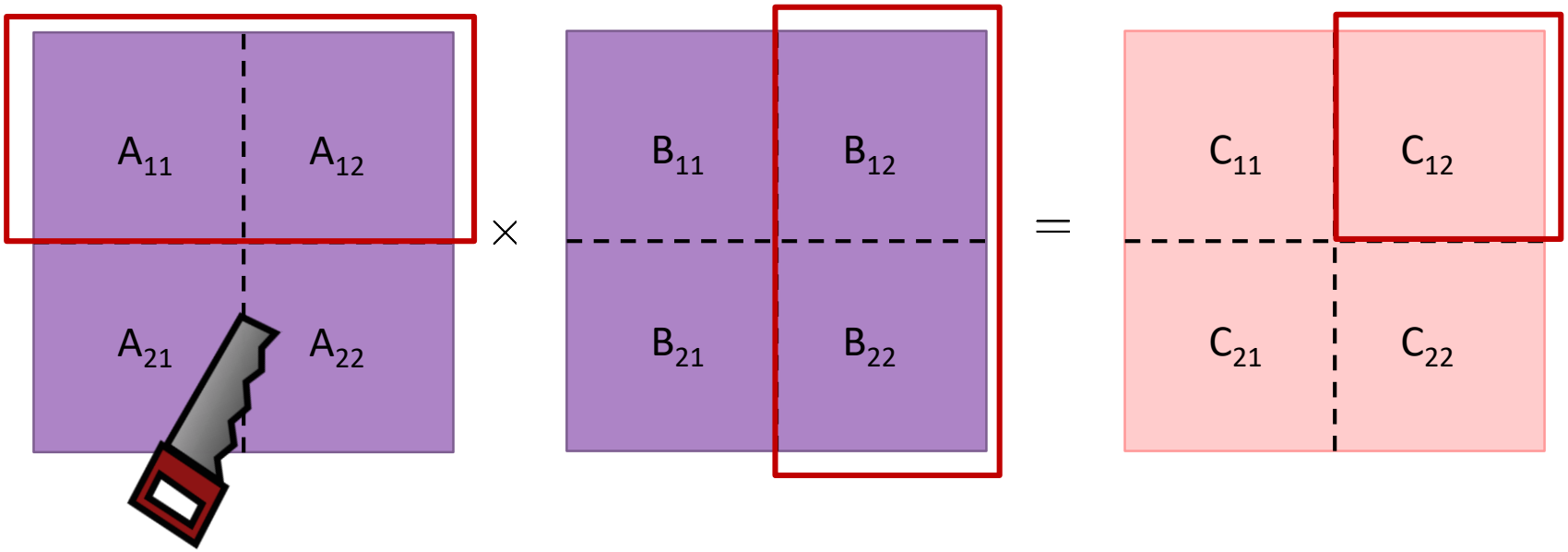
Lower bound $= \Omega(n^2)$

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

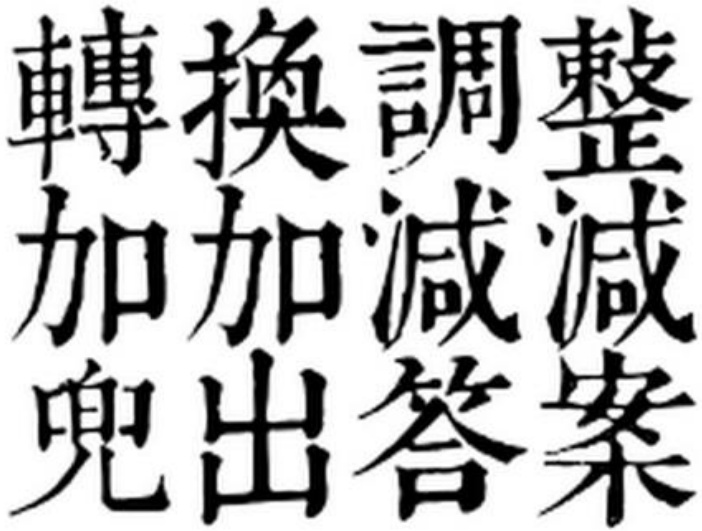
```
MatrixMultiply(n, A, B)
  //base case
  if n == 1
     return AB                                  Θ(1)
  //recursive case
  Divide A and B into n/2 by n/2 submatrices    Divide Θ(1)
  C11 = MatrixMultiply(n/2,A11,B11) + MatrixMultiply(n/2,A12,B21)
  C21 = MatrixMultiply(n/2,A11,B12) + MatrixMultiply(n/2,A12,B22)
  C21 = MatrixMultiply(n/2,A21,B11) + MatrixMultiply(n/2,A22,B21)
  C22 = MatrixMultiply(n/2,A21,B12) + MatrixMultiply(n/2,A22,B22)
                                                 Conquer 8T(n/2)
  return C
                                    Combine 4Θ((n/2)²) = Θ(n²)
```

- $T(n)$ = 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 reduce the number of recursive calls
  - From 8 recursive calls to 7 recursive calls $T(n/2)$
  - At the cost of extra addition and subtraction operations $\Theta((n/2)^2)$



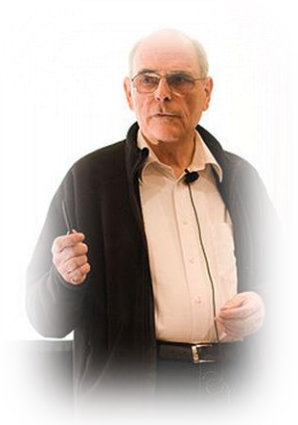
Intuition:

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

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

$$
\begin{aligned}
C_{12} &= M_3 + M_5 \\
&= A_{11}(B_{12} - B_{22}) + (A_{11} + A_{12})B_{22} \\
&= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= M_2 + M_4 \\
&= (A_{21} + A_{22})B_{11} + A_{22}(B_{21} - B_{11}) \\
&= A_{21}B_{11} + A_{22}B_{21}
\end{aligned}
$$

$$
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}
\quad
B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}
\quad
C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}
$$

- 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                                  Θ(1)
  // recursive case
  Divide A and B into n/2 by n/2 submatrices   Divide Θ(1)
  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
```

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

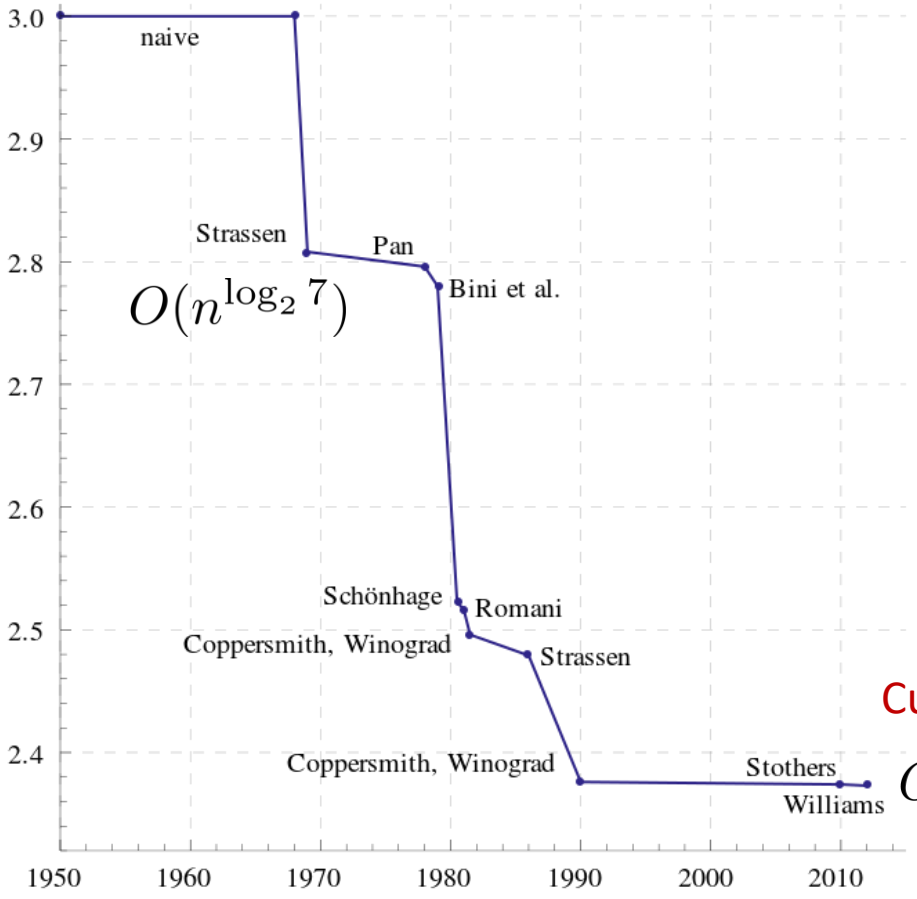
- Disadvantages
  1. Larger constant factor than it in the naïve approach

$$c_1 n^{\log_2 7}, c_2 n^3 \rightarrow c_1 > c_2$$

  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

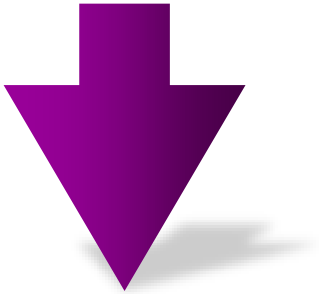
Matrix Multiplication Upper Bounds

- Each algorithm gives an upper bound

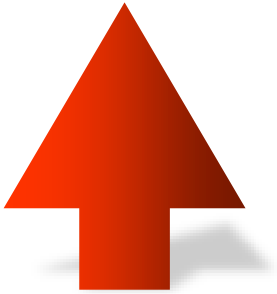


Current lowest upper bound

Matrix Multi. Problem Complexity



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



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

D&C #6: Selection Problem

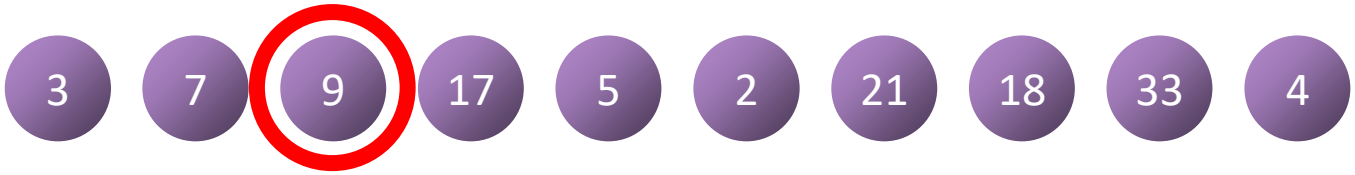
Textbook Chapter 9.3 – Selection in worst-case linear time

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.

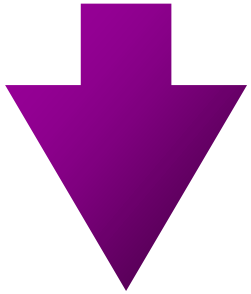
$n = 10, k = 5$



Selection Problem $\leqq$ 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?

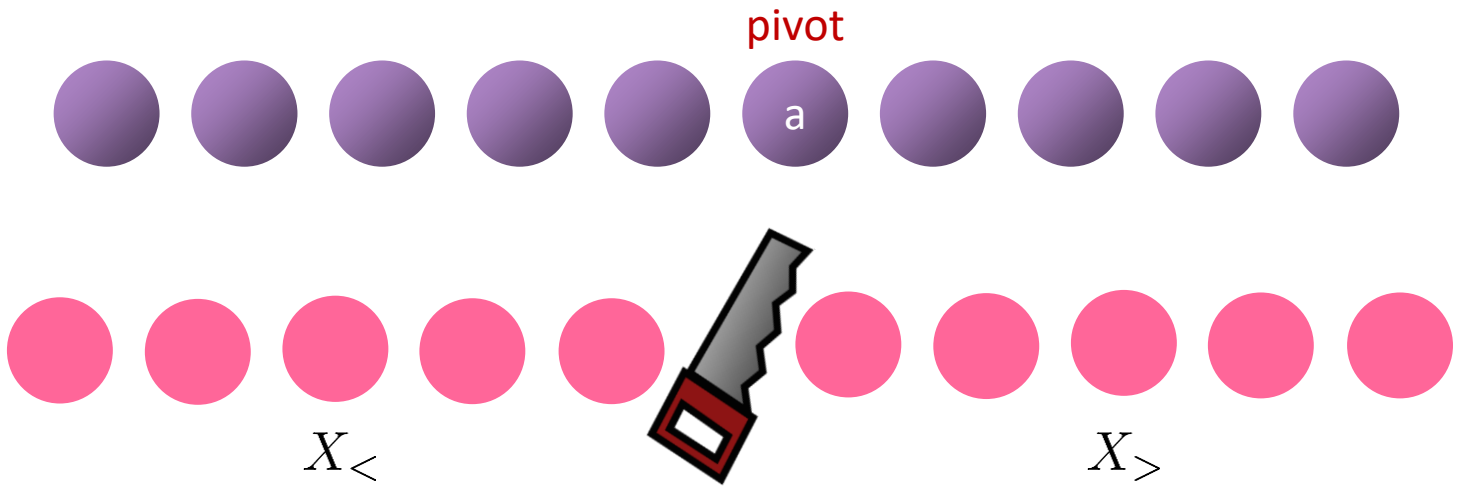
$$\text{Lower bound} = \Omega(n)$$

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).

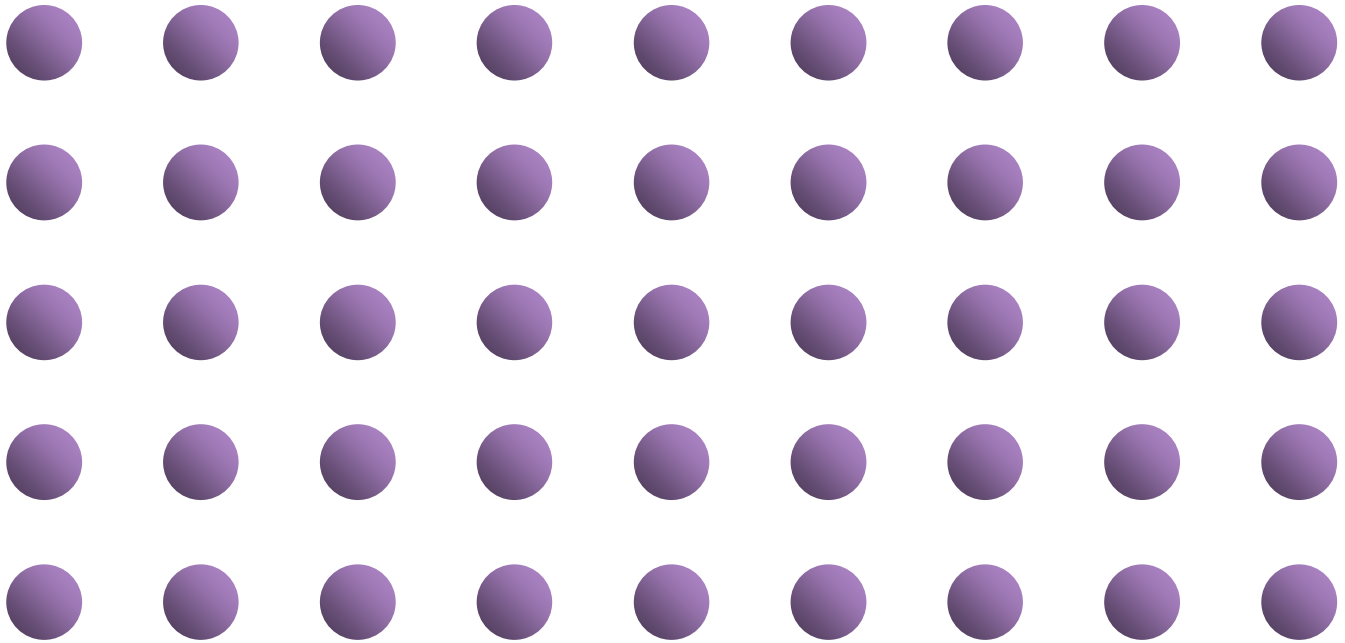
Divide-and-Conquer

- 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

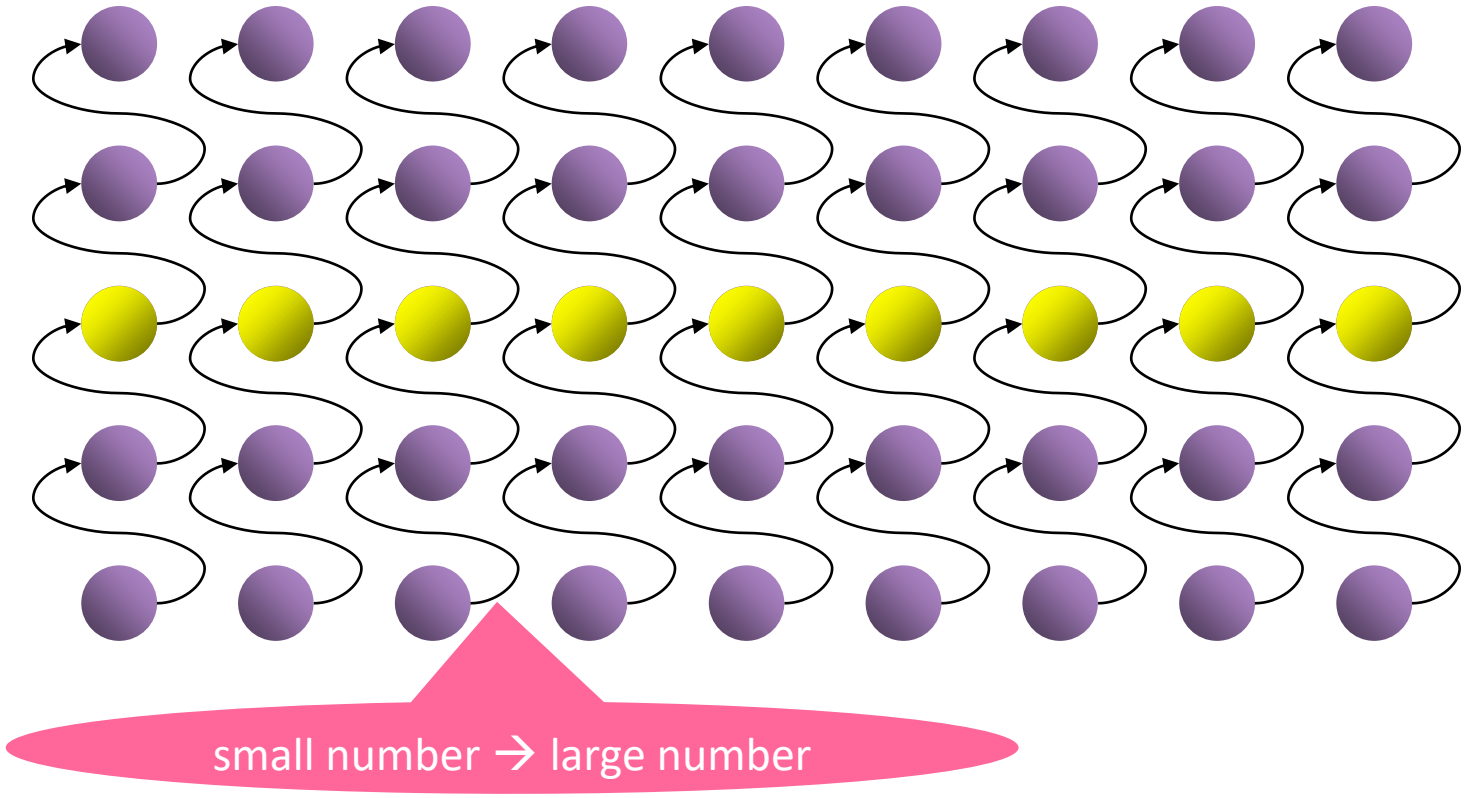


We want these subproblems to have similar size
→ The better pivot is the medium in the input array

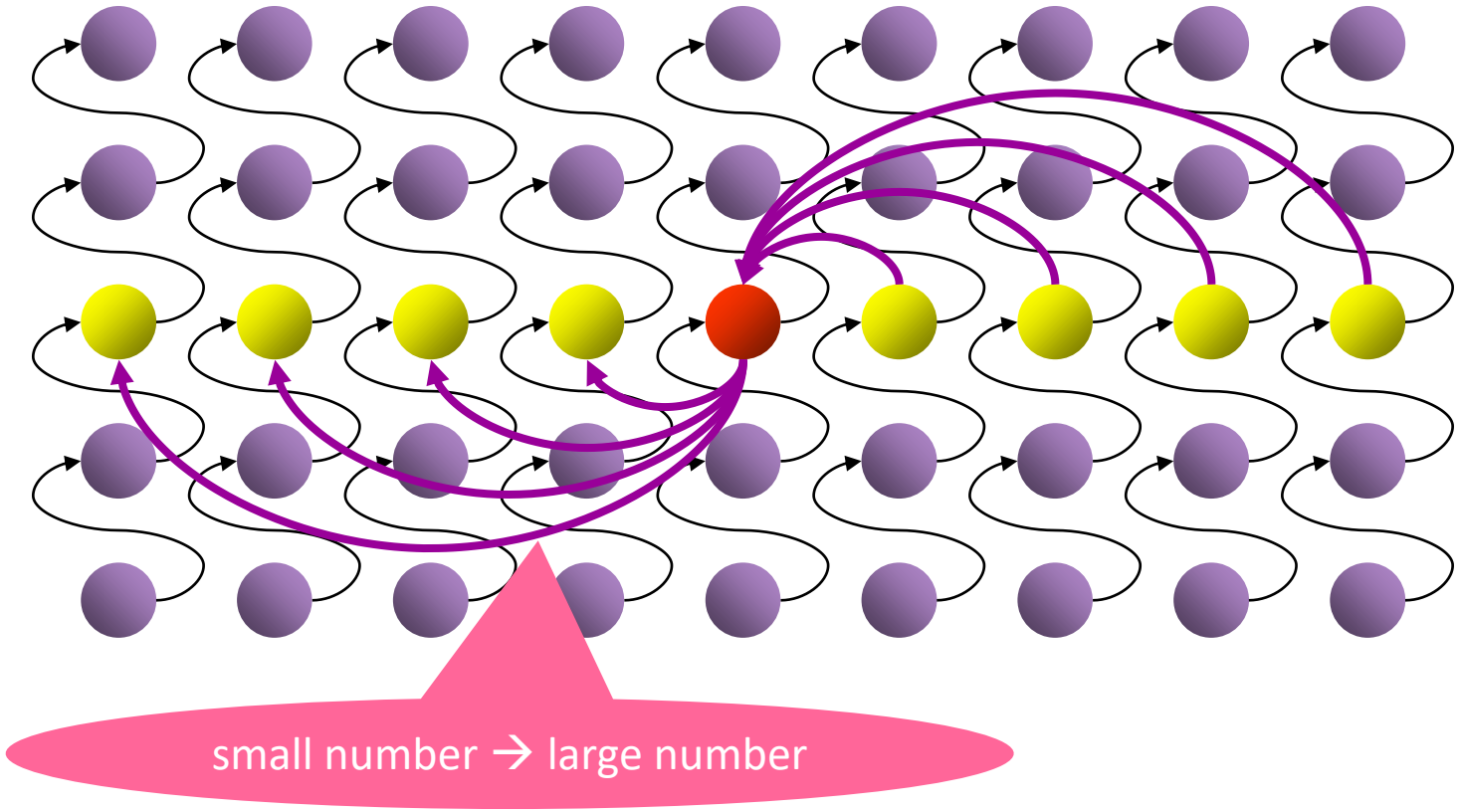
(1) Five Guys per Group



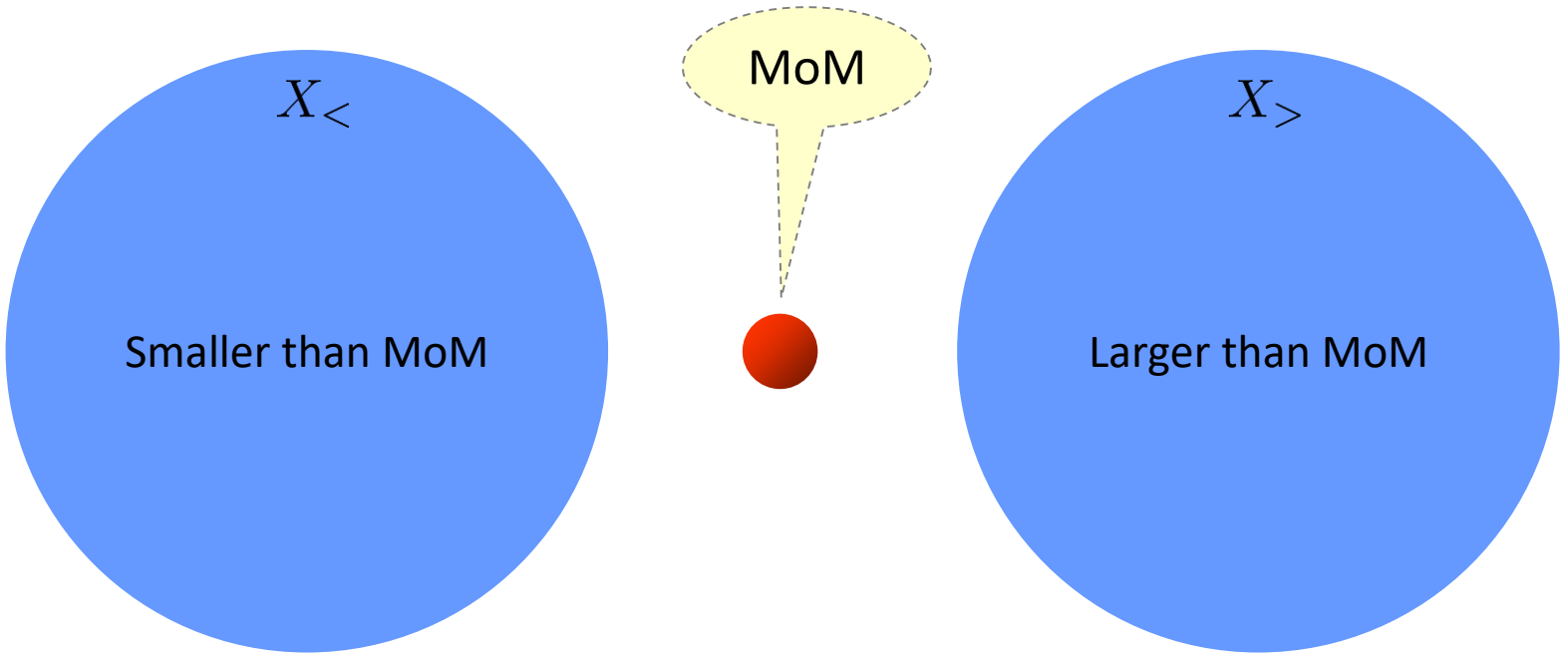
(2) A Median per Group



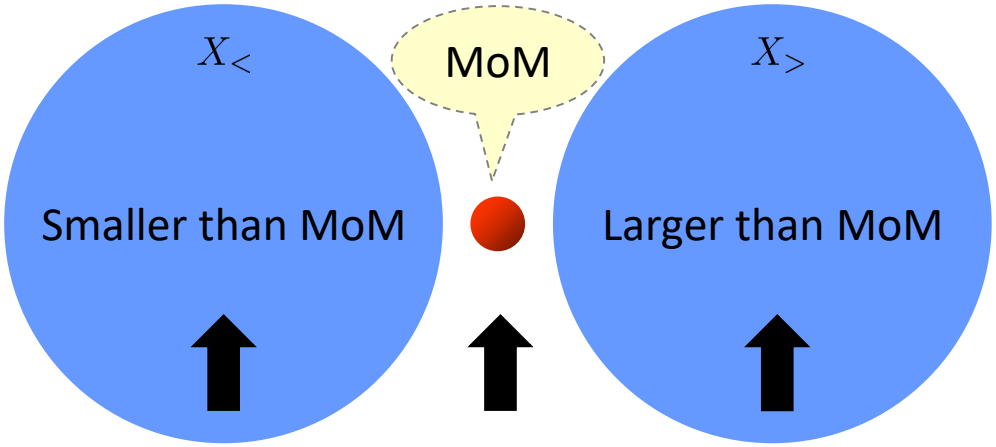
(3) Median of Medians (MoM)



(4) Partition via MoM



(5) Recursion



- Three cases
  1. If $k \leq |X_>|$, 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

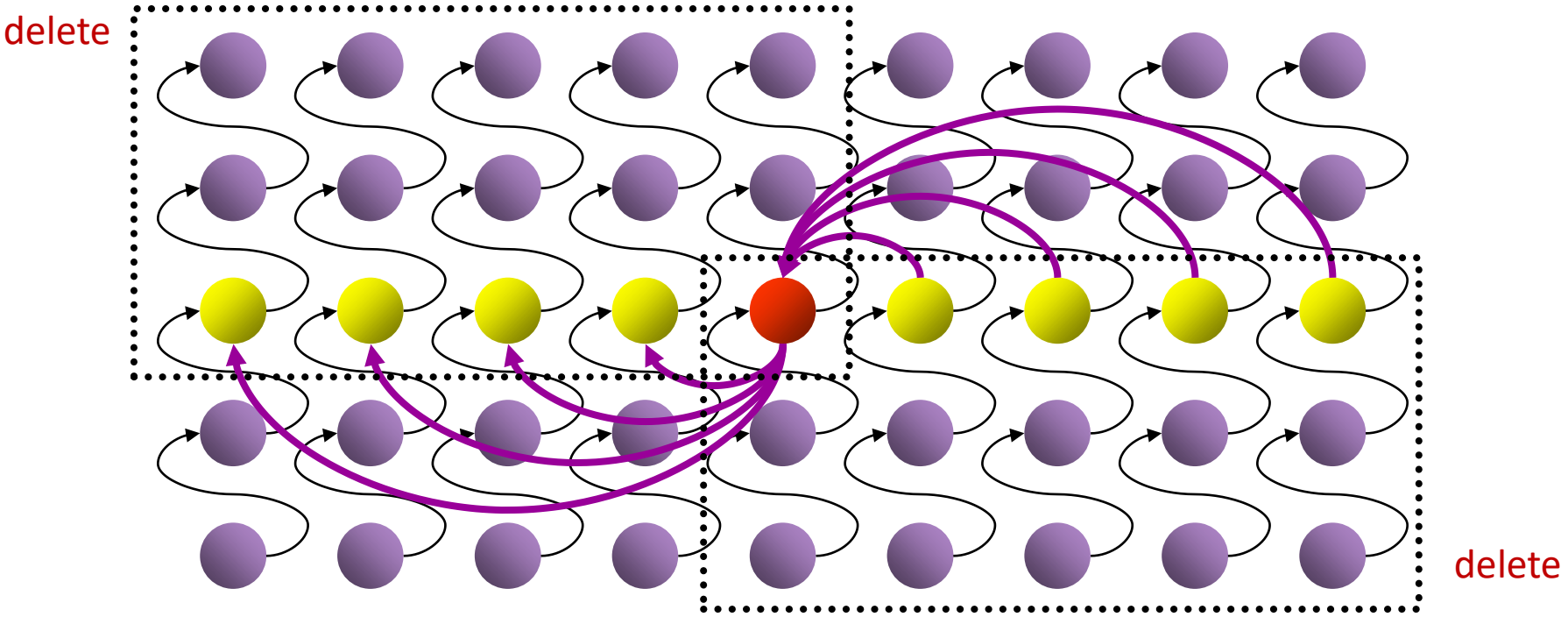
Two Recursive Steps

- 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]                          Θ(1)
  // recursive case
  Divide X into |X|/5 groups with size 5            Θ(1)
  M[i] = median from group i                        Θ(1) · Θ(n/5) = Θ(n)
  MoM = Selection(M, |M|/2)                         T(n/5)
  for i = 1 … |X|                                   ┐
    if X[i] > MoM                                   │
      insert X[i] into X2                           ├ Θ(n)
    else                                            │
      insert X[i] into X1                           ┘
  if |X2| == k - 1
    return x                                        Θ(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

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

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

- 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

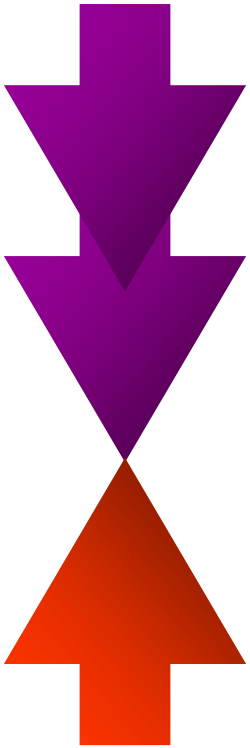
- Theorem

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

- 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,

$$\begin{aligned} T(n) &\leq T(n/5) + T(7n/10) + b \cdot n \\ &\overset{\text{Inductive hypothesis}}{\leq} \frac{1}{5}cn + \frac{7}{10}cn + bn = \frac{9}{10}cn + bn = cn - (\frac{1}{10}cn - bn) \quad \text{select } c > 10b \\ &\leq cn \end{aligned}$$

Selection Problem Complexity



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

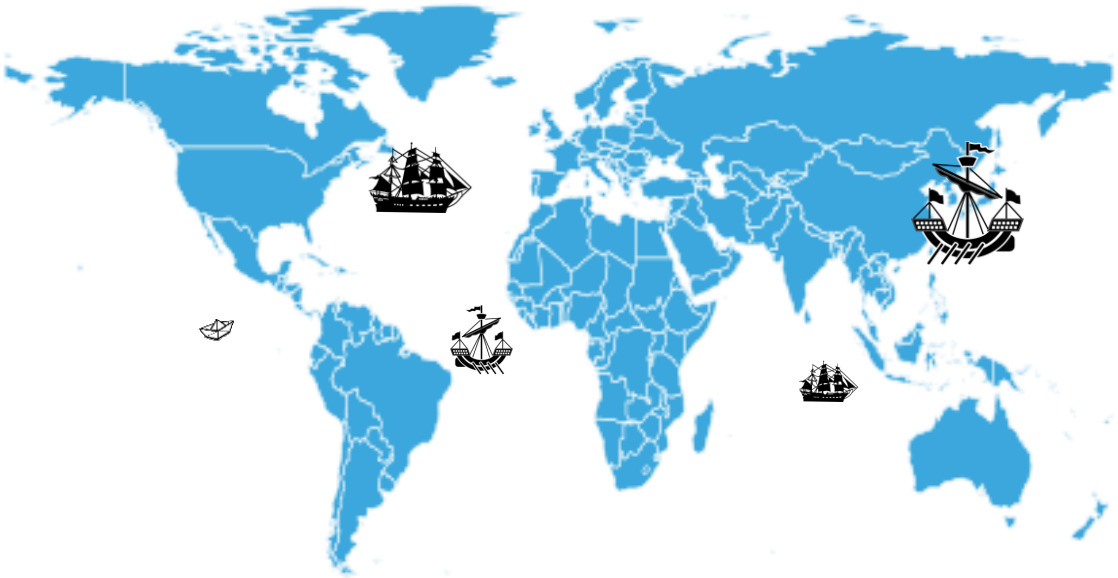
$$\text{Lower bound} = \Omega(n)$$

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

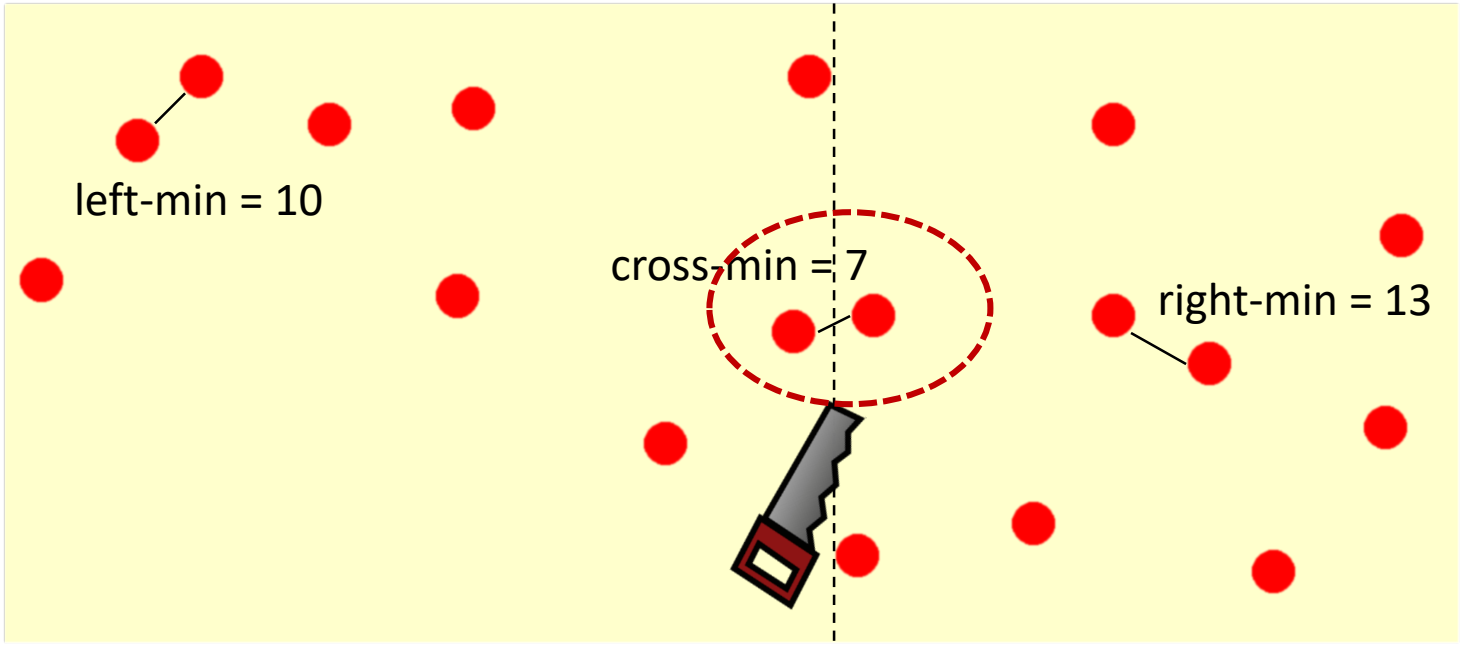
- 1D:
  - Sort all points $\Theta(n \log n)$
  - Scan the sorted points to find the closest pair in one pass $\Theta(n)$
    - We only need to examine the adjacent points

➡ $T(n) = \Theta(n \log n)$

- 2D:

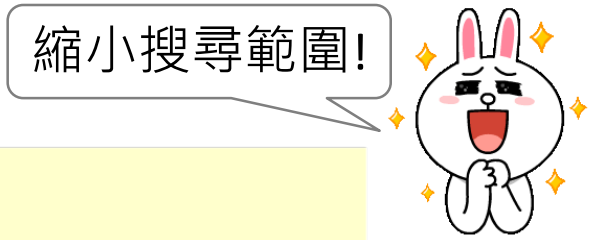
Divide-and-Conquer Algorithm

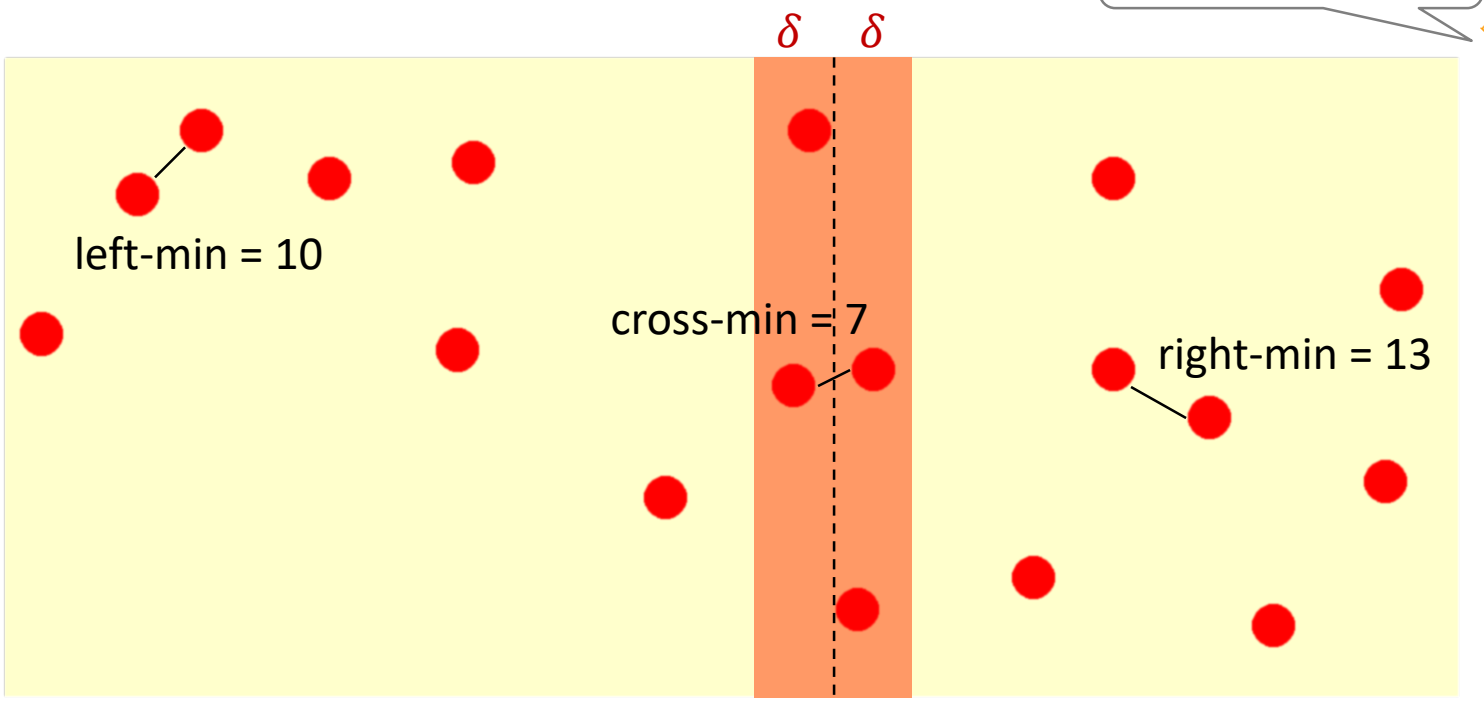
- Divide: divide points evenly along x-coordinate
- Conquer: find closest pair in each region recursively
- Combine: find closet pair with one point in each region, and return the best of three solutions



Cross Two Regions

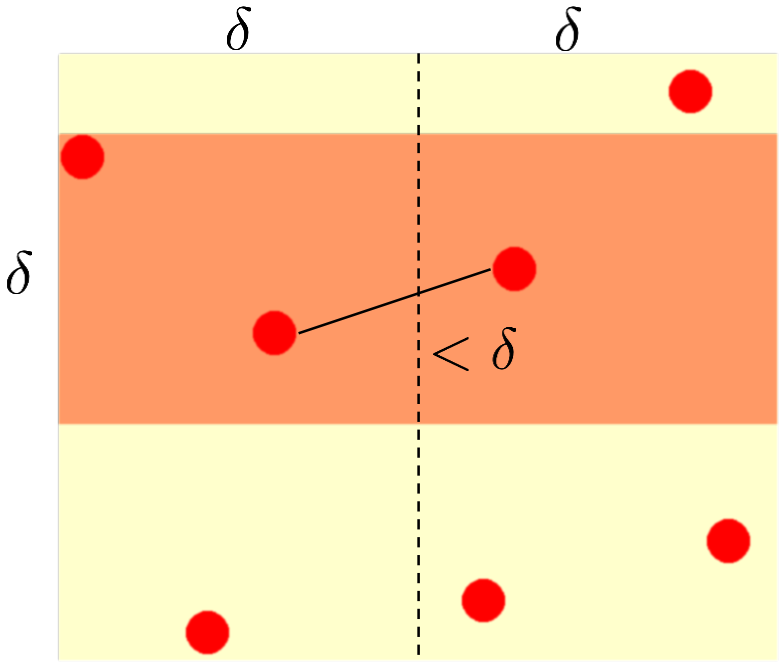
- Algo 1: check all pairs that cross two regions → $n/2 \times n/2$ combinations
- Algo 2: only consider points within δ of the cut, $\delta = \min\{\mathrm{l-min}, \mathrm{r-min}\}$
  - Other pairs of points must have distance larger than δ

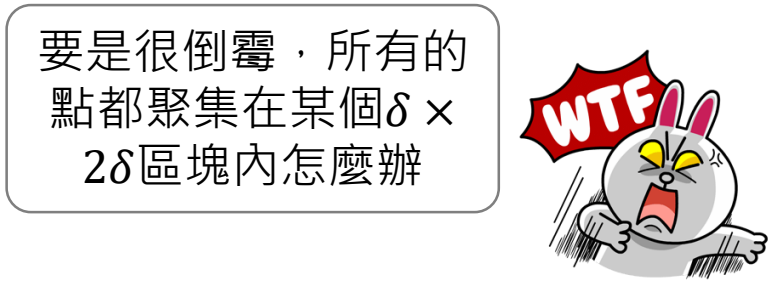




Cross Two Regions

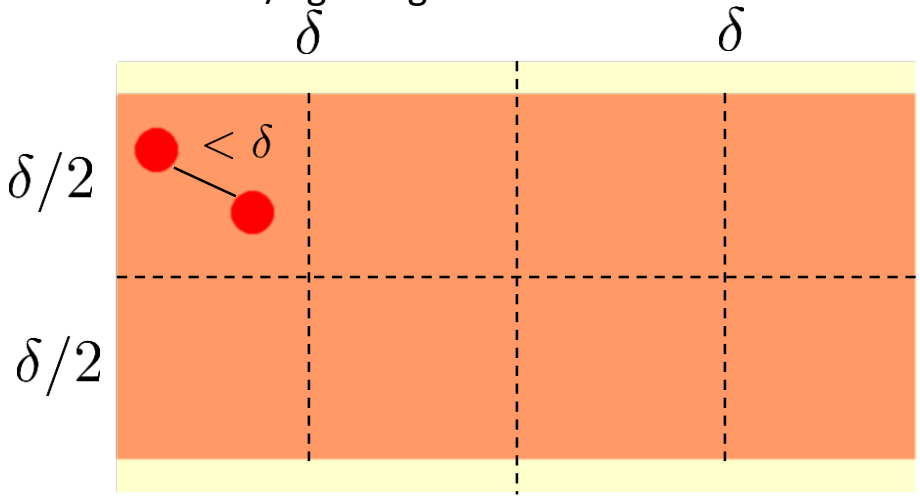
- Algo 1: check all pairs that cross two regions → $n/2 \times n/2$ combinations
- Algo 2: only consider points within δ of the cut, $\delta = \min\{\text{l}-\text{min}, \text{r}-\text{min}\}$
- Algo 3: only consider pairs within $\delta \times 2\delta$ blocks
  - Obs 1: every pair with smaller than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





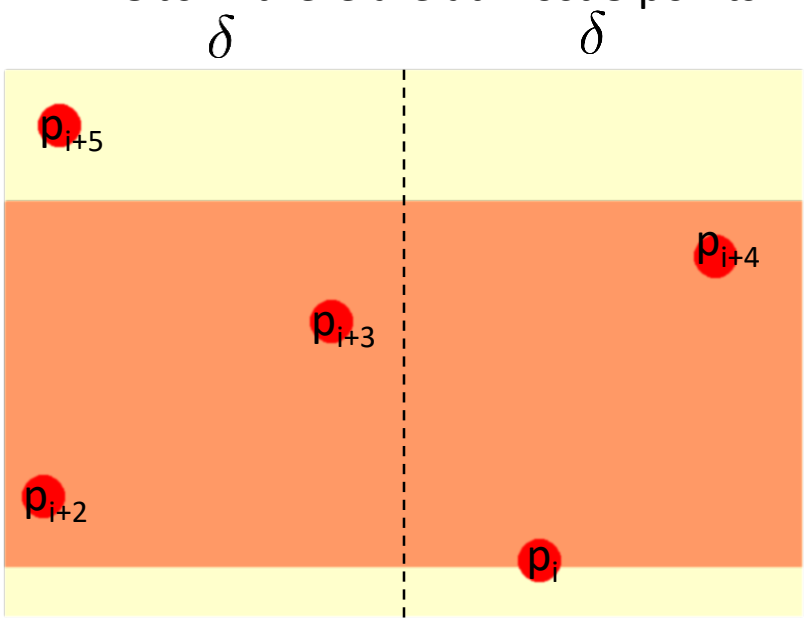
Cross Two Regions

- 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, $\delta = \min\{\text{l}-\text{min}, \text{r}-\text{min}\}$
- 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)                          Θ(1)
  if |P| <= 3 brute-force finding closest pair and return it
  // Divide                                                     Θ(n log n)
  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)    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 p0, …, pk         Θ(n log n)
  for point pi:                                                Θ(n)
    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
```

- $T(n)$ = 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

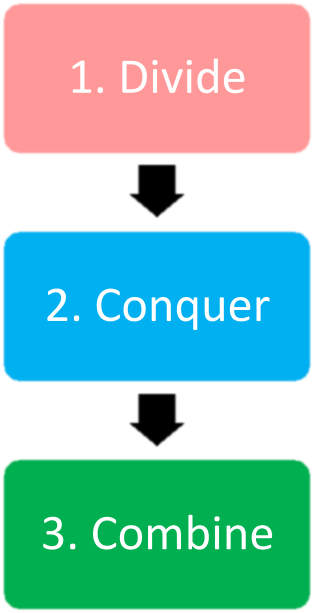
- $O(n)$ algorithm
  - Taking advantage of randomization
    - Chapter 13.7 of Algorithm Design by Kleinberg & Tardos
    - Samir Khuller and Yossi Matias. 1995. A simple randomized sieve algorithm for the closest-pair problem. Inf. Comput. 118, 1 (April 1995), 34-37.

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: $\Theta(n)$

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

```
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]
```



Question?

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

Course Website: http://ada17.csie.org

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