Algorithms Design
Chap05-Divide and Conquer

College of Computer Science

Nankai University

Tianjin, P.R.China

Chap05-Divide and Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into two equal parts of size $n/2$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

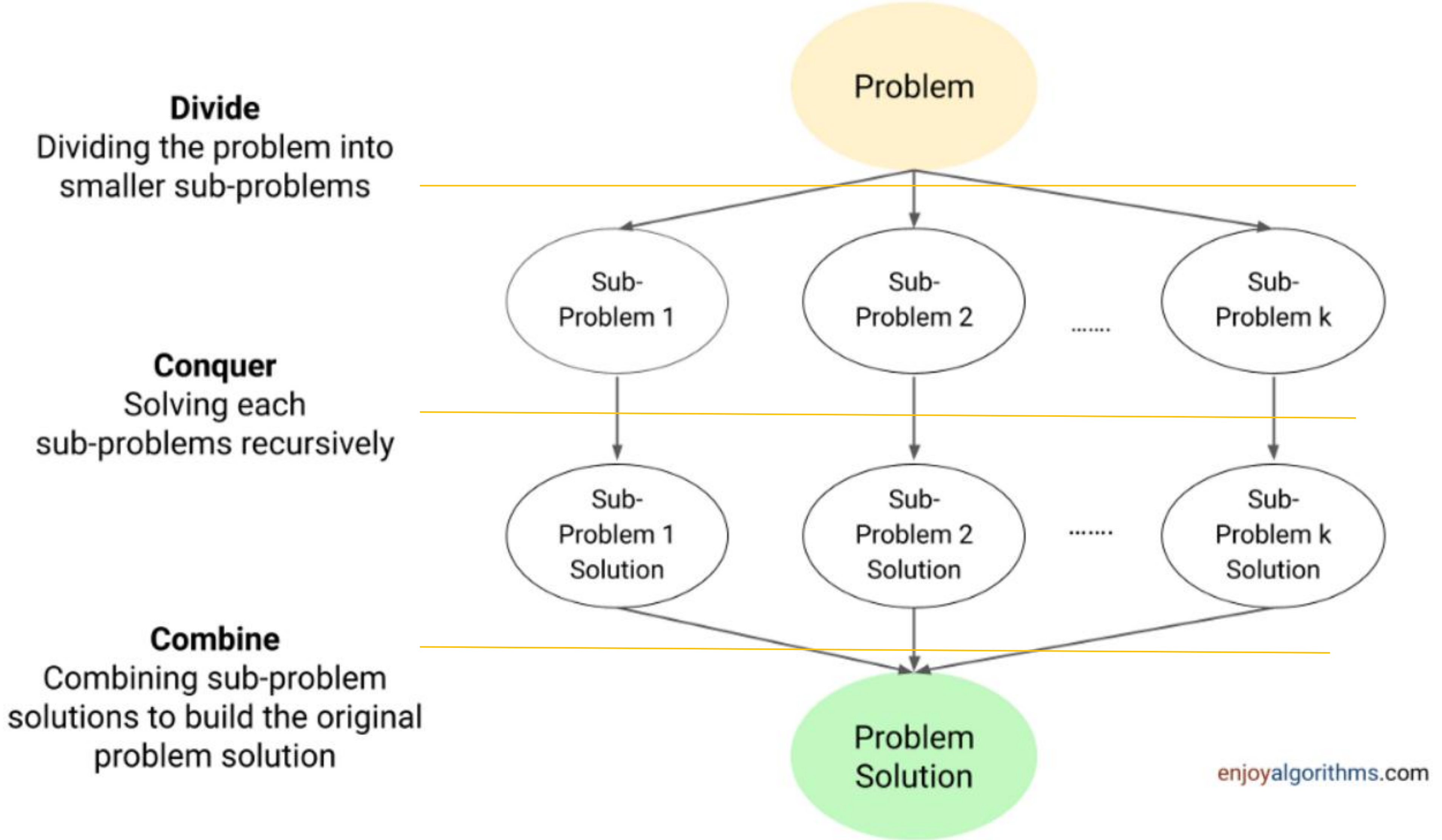
Chap05-Divide and Conquer

Divide-and-Conquer is a technique used for solving problems by breaking them down into smaller subproblems that can be solved independently and then combining these solutions to obtain the solution to the problem.

Properties

- Optimal substructure if any optimal solution to the problem contains the optimal solutions to subproblems.
- The problem can be divided into independent and no overlapping subproblems.
- The solution to the subproblems can be combined to form a solution to the problem.

Chap05-Divide and Conquer



Chap05-Divide and Conquer Outline

5.1 Mergesort algorithm

5.2 Recurence relations

5.3 Counting inversions

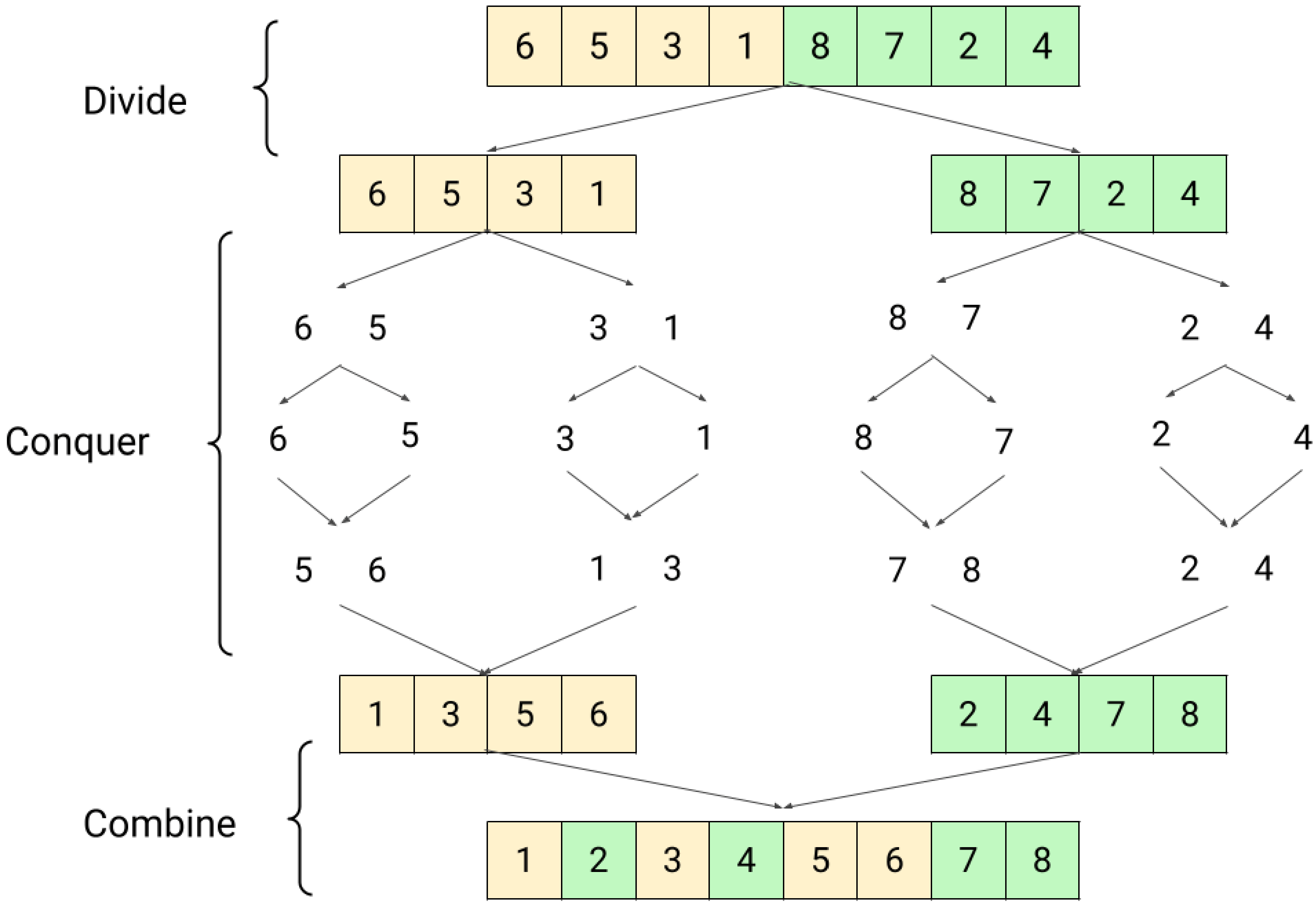
5.4 Finding the closest pairs of points

5.1 Mergesort algorithm

Mergesort.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

A	L	G	O	R	I	T	H	M	S		
A	L	G	O	R	I	T	H	M	S	divide	$O(1)$
A	G	L	O	R	H	I	M	S	T	sort	$2T(n/2)$
A	G	H	I	L	M	O	R	S	T	merge	$O(n)$


Divide
6 5 3 1 8 7 2 4
6 5 3 1
8 7 2 4
Conquer
6 5
3 1
8 7
2 4
6
5
3
1
8
7
2
4
5 6
1 3
7 8
2 4
1 3 5 6
2 4 7 8
Combine
1 2 3 4 5 6 7 8

5.1 Mergesort algorithm

Merging: Combine two sorted lists A and B into a sorted whole C.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i \leq b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).

5.1 Mergesort algorithm

5.1 Mergesort algorithm

Input. List L of n elements

Output. The n elements in ascending order.

MERGE-SORT(L)

IF (list L has one element)

  RETURN L.

Divide the list into two halves A and B.

$A \leftarrow$ MERGE-SORT(A). $\longleftarrow$ $T(n / 2)$

$B \leftarrow$ MERGE-SORT(B). $\longleftarrow$ $T(n / 2)$

$L \leftarrow$ MERGE(A, B). $\longleftarrow$ $\Theta(n)$

RETURN L.

5.1 Mergesort algorithm

Def. $T(n)$= max number of compares to mergesort a list of length n.

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & \text{if } n > 1 \end{cases}$$

Solution. $T(n)$ is $O(n \log_2 n)$.

5.1 Mergesort algorithm

Assorted proofs.

- We describe several ways to solve this recurrence.
- Initially we assume n is a power of 2 and replace $\leq$ with $=$ in the recurrence.

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

5.1 Mergesort algorithm

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n\log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

Pf. For $n > 1$, $T(n) = 2T\left(\frac{n}{2}\right) + n \Rightarrow \frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1 = \frac{T(n/4)}{n/4} + 1 + 1 = \cdots = \frac{T\left(\frac{n}{n}\right)}{\frac{n}{n}} + \overbrace{1 + 1 + \cdots + 1}^{\log_2 n}$$

5.1 Mergesort algorithm

Pf. [by induction on n]

Base case: when n = 1, $T(1) = 0 = n \log_2 n$.

Inductive hypothesis: assume $T(1) = n \log_2 n$.

Goal: show that $T(2n) = \ 2n \log_2(2n)$.

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n\left(\log_2(2n) - 1\right) + 2n \\ &= 2n \log_2(2n) \end{aligned}$$

Chap05-Divide and Conquer Outline

5.1 Mergesort algorithm

5.2 Recurence relations

5.3 Counting inversions

5.4 Finding the closest pairs of points

5.2 Recurence relations

Goal. Recipe for solving common divide-and-conquer recurrences: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

with T(0) = 0 and T(1) = Θ(1).

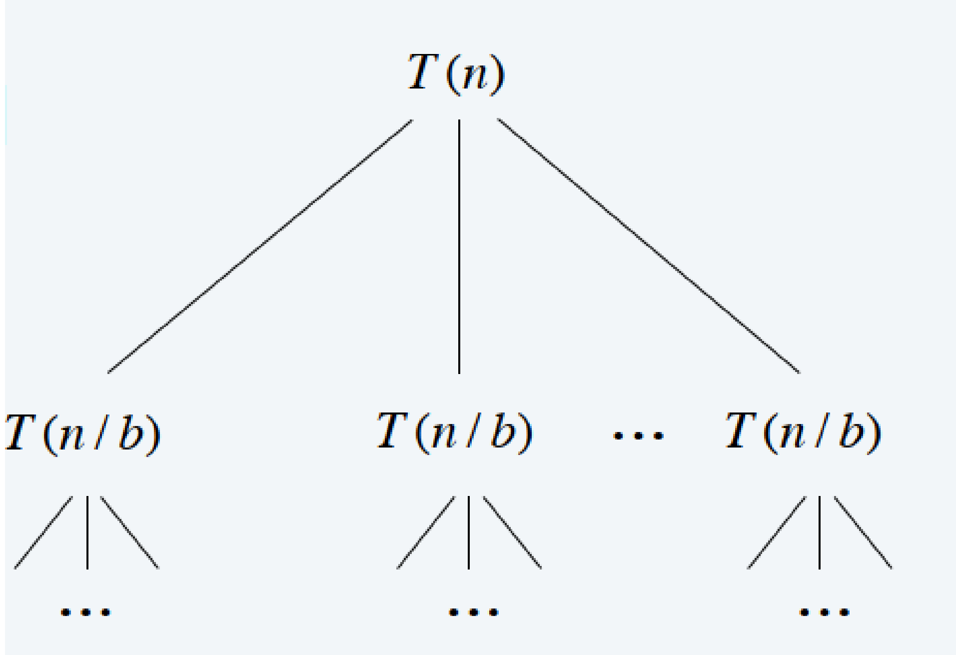
Terms.

- $a \geq 1$ is the number of subproblems.
- $b \geq 2$ is the factor by which the subproblem size decreases.
- $f(n) \geq 0$ is the work to divide and combine subproblems.

5.2 Recurence relations

Recursion tree. [assuming n is a power of b]

- a = branching factor.
- a^i = number of subproblems at level i.
- $1 + \log_b n$ levels.
- n / b^i = size of subproblem at level i.



5.2 Recurence relations

Suppose $T(n)$ satisfies $T(n) = aT(n/\,b) + n^c$ with $T(1) = 1$, for n a power of b.

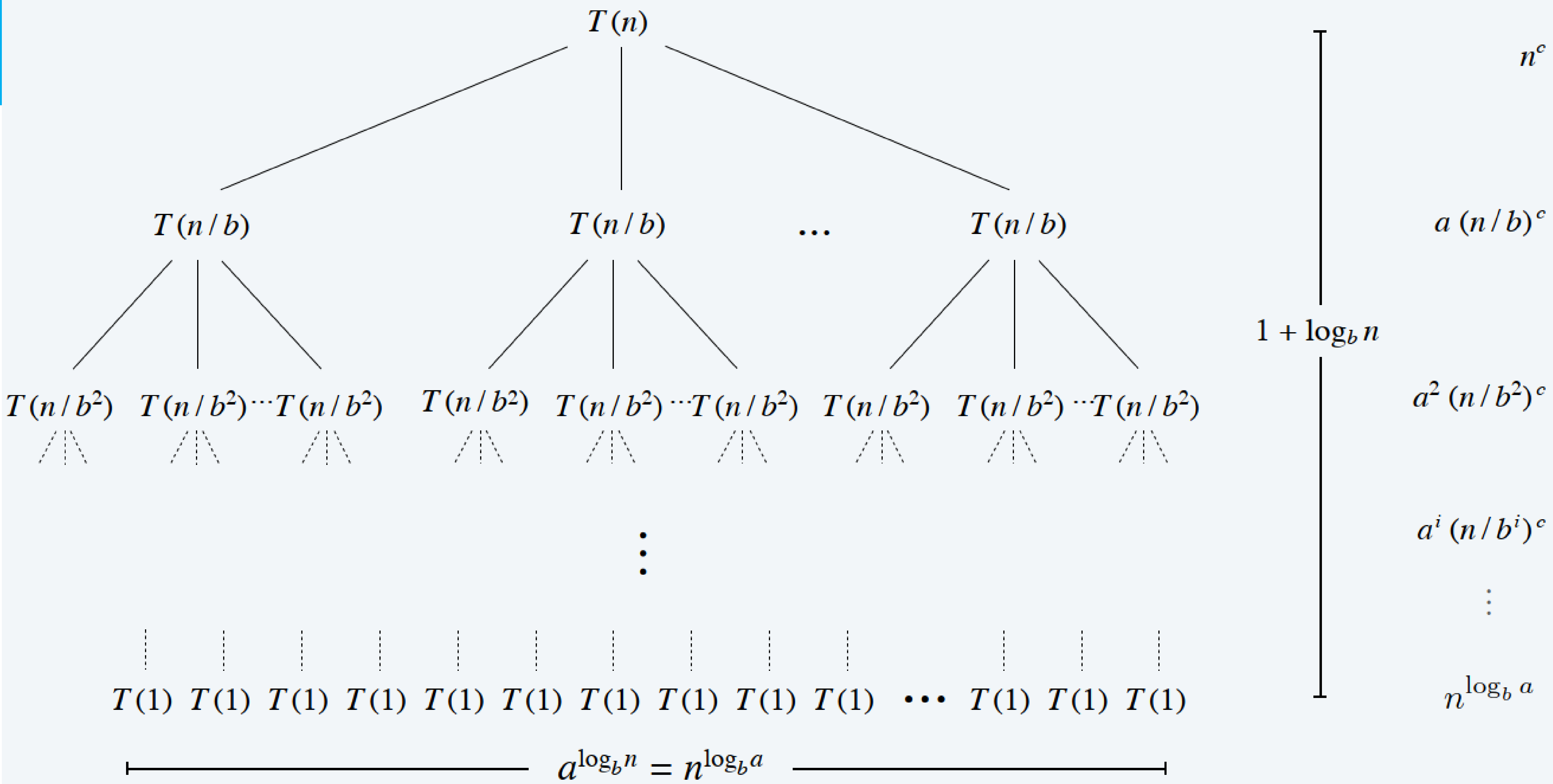
Let $r = a/b^c$. Note that $r < 1$ iff $c > \log_b a$.

$$T(n) = n^c \sum_{i=0}^{\log_b n} r^i = \begin{cases} \Theta(n^c) & \text{if } r < 1 & c > \log_b a & \longleftarrow \\ \Theta(n^c \log n) & \text{if } r = 1 & c = \log_b a & \longleftarrow \\ \Theta(n^{\log_b a}) & \text{if } r > 1 & c < \log_b a & \longleftarrow \end{cases}$$

cost dominated by cost of root

cost evenly distributed in tree

cost dominated by cost of leaves


$T(n)$
n^c
$T(n/b)$ $T(n/b)$ ... $T(n/b)$
$a\,(n/b)^c$
$T(n/b^2)$ $T(n/b^2)$ ⋯ $T(n/b^2)$ $T(n/b^2)$ $T(n/b^2)$ ⋯ $T(n/b^2)$ $T(n/b^2)$ $T(n/b^2)$ ⋯ $T(n/b^2)$
$a^2\,(n/b^2)^c$
$1 + \log_b n$
$a^i\,(n/b^i)^c$
$T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ ••• $T(1)$ $T(1)$ $T(1)$
$n^{\log_b a}$
$a^{\log_b n} = n^{\log_b a}$

5.2 Recurence relations

Master theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) \;=\; a\,T\left(\frac{n}{b}\right) \;+\; \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

Case 1. If $c > \log_b a$, then $T(n) = \Theta(n^c)$. n^c grow faster than Leaves

Case 2. If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$. Leaves grow at the same rate as n^c

Case 3. If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$. Leaves grow faster than n^c

5.2 Recurence relations

E.g.1,

$T(n) = 3\,T(n\,/\,2) + 5n$

- $a = 3,\ \ b = 2,\, c = 1 < \log_b a \approx 1.5849$
- $T(n) = \Theta\left(n^{log_2 3}\right) = O(n^{1.58})$.

E.g.2,

$T(n) = 2T(n/2) + 17n$

- $a = 2,\ \ b\ = \ 2,\, c = 1 = log_b a.$
- $T(n) = \Theta(n \log n)$.

5.2 Recurence relations

E.g.3

$$T(n) = 48\,T(n/4) + n^3$$

- $a = 48,\, b = 4,\, c = 3 > log_b a \;\approx 2.7924$
- $T(n) = \Theta(n^3)$

E.g.4

$$T(n) = 9\,T(n/3) + n$$

- $a = 9, b = 3, c = 1 \; < 2 = log_b a$
- $T(n) = \Theta\left(n^{log_b a}\right) = O(n^2).$

5.2 Recurence relations

E.g.5

$$T(n) = T(2n/3) + 1$$

- $a = 1, b = \frac{3}{2}, c = 0 = log_b a$
- $T(n) = \Theta\left(n^{log_b a} \log n\right) = O(\log n)$.

Chap05-Divide and Conquer Outline

5.3 Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \cdots, n$.
- Your rank: $a_1, a_2, \cdots, a_n$.
- Songs i and j are inverted if $i < j$, but $a_i > a_j$.

Brute force: check all $\Theta(n^2)$ pairs.

5.3 Counting inversions

divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with $a \in A$ and $b \in B$.
- Return sum of three counts.

5.3 Counting inversions

1	5	4	8	10	2	6	9	3	7

1	5	4	8	10

5-4

2	6	9	3	7

6-3 9-3 9-7

1	5	4	8	10	2	6	9	3	7

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

Output 1+3+13=17

5.3 Counting inversions

How to count inversions (a, b) with $a \in A$ and $b \in B$?

- Sort A and B.
- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

5.3 Counting inversions

list A

7	10	18	3	14

list B

20	23	2	11	16

sort A

3	7	10	14	18

sort B

2	11	16	20	23

count inversions (a, b) with a ∈ A and b ∈ B

3	7	10	a_i	18

2	11	b_j	20	23
5	2			

merge to form sorted list C

2	3	7	10	11					

5.3 Counting inversions

divide-and-conquer

- Divide: separate list into two halves A and B. O(1)
- Conquer: recursively count inversions in each list. 2T(n/2)
- Combine: count inversions (a, b) with $a \in A$ and $b \in B$. O(n)
- Return sum of three counts.

$$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = o(n \log n)$$

5.3 Counting inversions

Input. List L.

Output. Number of inversions in L and L in sorted order.

```
SORT-AND-COUNT(L)
________________________________________
IF (list L has one element)
    RETURN (0, L).

Divide the list into two halves A and B.
(r_A, A)  ← SORT-AND-COUNT(A).          ←—— T(n / 2)
(r_B, B)  ← SORT-AND-COUNT(B).          ←—— T(n / 2)
(r_AB, L) ← MERGE-AND-COUNT(A, B).      ←—— Θ(n)

RETURN (r_A + r_B + r_AB, L).
```

Chap05-Divide and Conquer Outline

5.1 Mergesort algorithm

5.2 Recurence relations

5.3 Counting inversions

5.4 Finding the closest pairs of points

5.4 Finding the closest pairs of points

Closest pair problem.

- Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

5.4 Finding the closest pairs of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

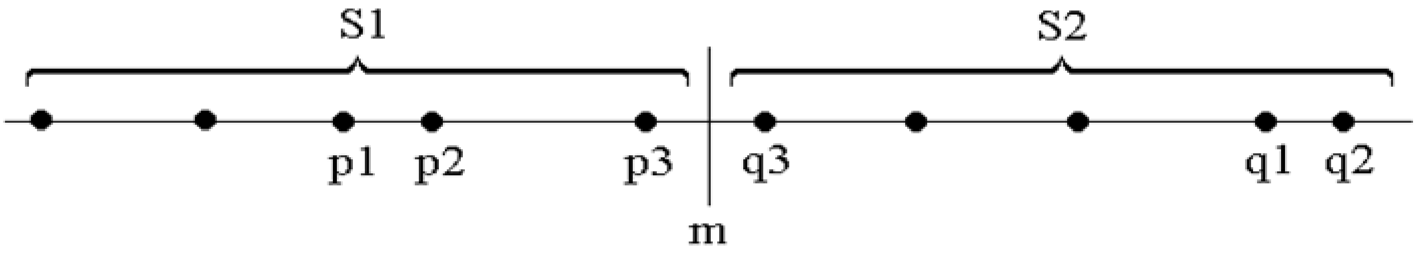
1D version. Easy $O(n \log n)$ algorithm if points are on a line.

Assumption. No two points have the same x-coordinate.

5.4 Finding the closest pairs of points

Divide-and-conquer algorithm for 1D closest pair of points

- Divide: find a point so that $n/2$ 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.

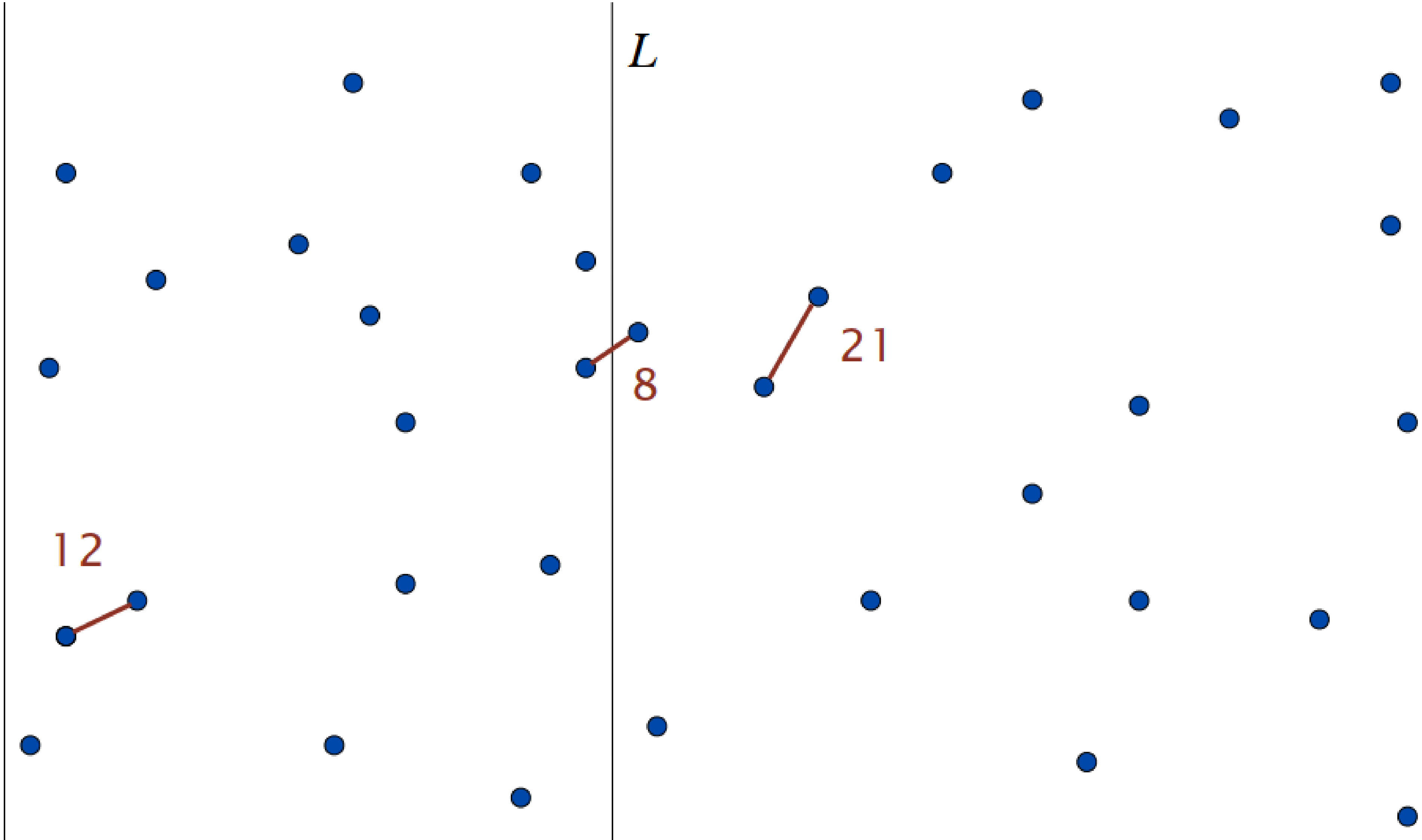


5.4 Finding the closest pairs of points

Divide-and-conquer algorithm for 2D closest pair of points

- Divide: find vertical line L so that $n/2$ 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.

5.4 Finding the closest pairs of points

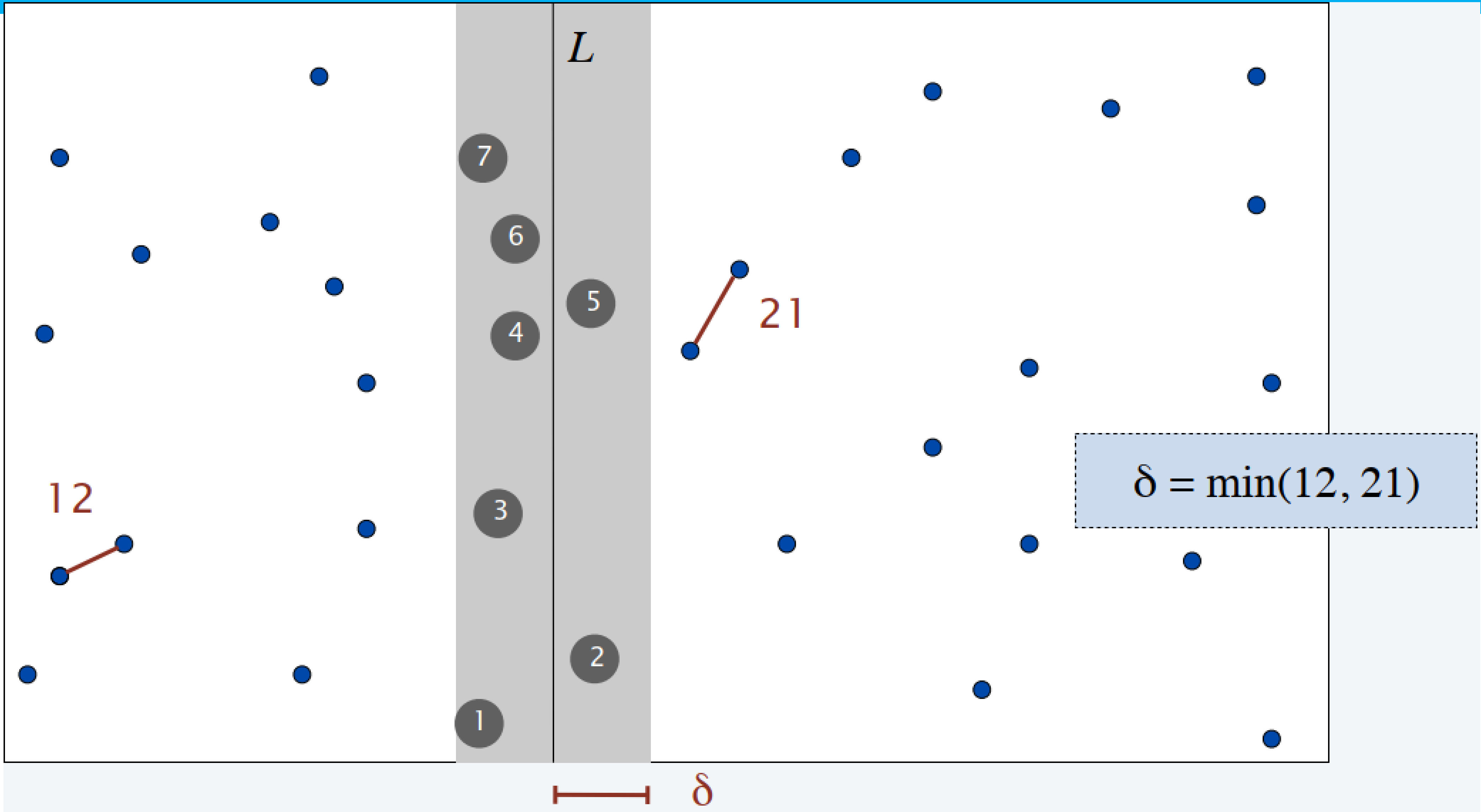


5.4 Finding the closest pairs of points

How to find closest pair with one point in each side?

- Assuming that distance $< \delta$
- Observation: suffices to consider only those points within δ of line L.
- Sort points in $2\ \delta$-strip by their y-coordinate.
- Check distances of only those points within 7 positions in sorted list!

5.4 Finding the closest pairs of points



5.4 Finding the closest pairs of points

Def. Let s_i be the point in the 2 δ-strip, with the i^{th} smallest y-coordinate.

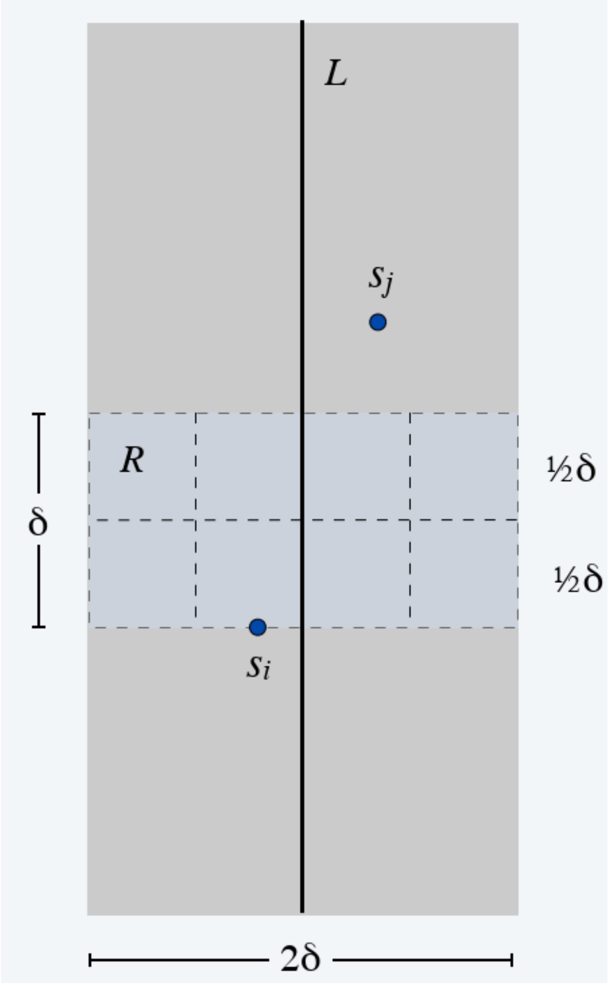
Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least δ.

5.4 Finding the closest pairs of points

Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least δ.

Pf.

- Consider the 2δ-by-δ rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
- Distance between s_i and any point s_j above R is $\geq \delta$.
- Subdivide R into 8 squares.
- At most 1 point per square.
- At most 7 other points can be in R.



CLOSEST-PAIR($p_1, p_2, \ldots, p_n$)

Compute vertical line L such that half the points are on each side of the line. ← $O(n)$

$\delta_1 \leftarrow$ CLOSEST-PAIR(points in left half). ← $T(n / 2)$

$\delta_2 \leftarrow$ CLOSEST-PAIR(points in right half). ← $T(n / 2)$

$\delta \leftarrow \min \{ \delta_1 , \delta_2 \}$.

Delete all points further than δ from line L. ← $O(n)$

Sort remaining points by y-coordinate. ← $O(n \log n)$

Scan points in y-order and compare distance between each point and next 7 neighbors. If any of these distances is less than δ, update δ. ← $O(n)$

RETURN δ.

5.4 Finding the closest pairs of points

Running Time

$T(n) = 2T(n/2) + O(n \log n) \Rightarrow T(n) = o(n \log^2 n)$

How to improve to O(n log n) ?

Don't sort points in strip from scratch each time.

- Each recursive call returns two lists: all points sorted by x-coordinate, and all points sorted by y-coordinate.
- Sort by merging two pre-sorted lists.

Thanks for Listening

College of Computer Science
Nankai University
Tianjin, P.R.China