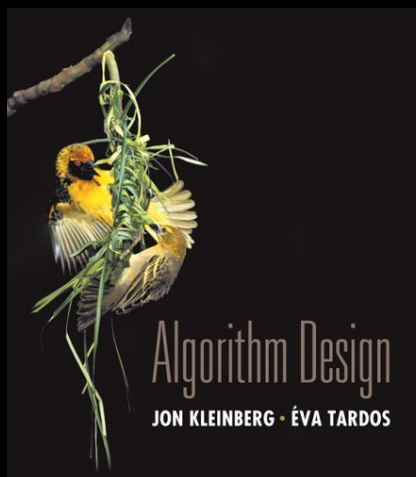


Chapter 5

Divide and Conquer

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 $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: n^2.
- Divide-and-conquer: n log n.

2

5.1 Mergesort

Sorting

Sorting. Given n elements, rearrange in ascending order.

Applications.
- Sort a list of names.
- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

obvious applications

- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

problems become easy once items are in sorted order

- Data compression.
- Computer graphics.
- Computational biology.
- Supply chain management.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

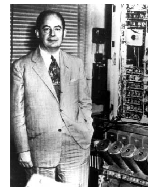
. . .

non-obvious applications

4

Mergesort

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



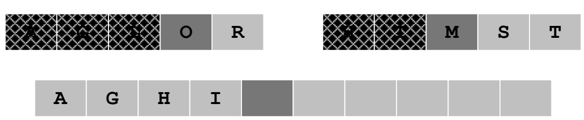
Jon von Neumann (1945)

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	$O(n)$

Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?
- Linear number of comparisons.
- Use temporary array.



Challenge for the bored. In-place merge. [Kronrud, 1969]

using only a constant amount of extra storage

A Useful Recurrence Relation

Def. T(n) = number of comparisons to mergesort an input of size n.

Mergesort recurrence.

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

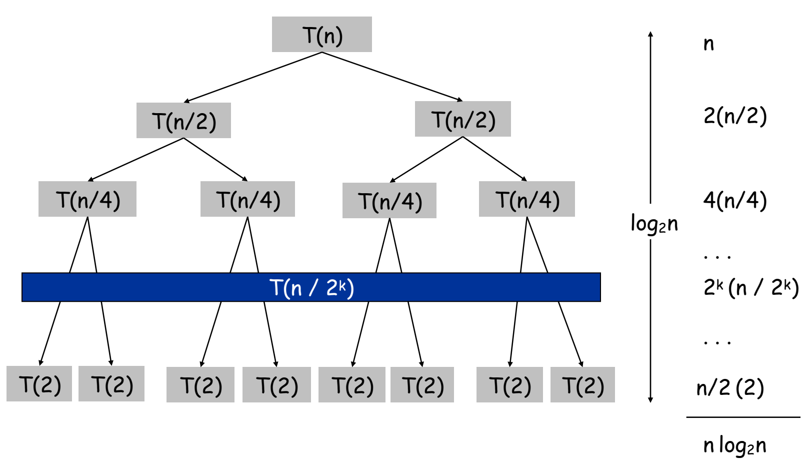
Solution. $T(n) = O(n \log_2 n)$.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace ≤ with =.

7

Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



8

Proof by Telescoping

Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

9

Some General Recurrence Relations

(5.1) For some constant c,

T(n) <= 2T(n/2) + cn, when n>2

T(2) <= c

(5.2) T(n) is bounded by O(nlogn) when n>1.

(5.3) For some constant c,

T(n) <= qT(n/2) + cn, when n>2

T(2) <=c

(5.4) T(n) with q > 2 is bounded by $O(n^{\log_2 q})$

(5.5) T(n) with q = 1 is bounded by O(n)

(5.6) For some constant c,

T(n) <= 2T(n/2) + cn^2, when n > 2

T(2) <= c

T(n) is bounded by $O(n^2)$ when n > 1.

12

A General Format

Suppose a complexity function $T(n)$ is eventually nondecreasing and satisfies

$$\boxed{\begin{array}{ll} T(n) = aT\left(\frac{n}{b}\right) + cn^k & \text{for } n > 1,\ n \text{ a power of } b \\ T(1) = d & \end{array}}$$

where $b \geq 2$ and $k \geq 0$ are constant integers, and a, c, and d are constants such that $a > 0$, $c > 0$, and $d \geq 0$. Then

$$T(n) \in \begin{cases} \Theta\left(n^k\right) & \text{if } a < b^k \\ \Theta\left(n^k \lg n\right) & \text{if } a = b^k \\ \Theta\left(n^{\log_b a}\right) & \text{if } a > b^k. \end{cases} \tag{B.5}$$

13

5.3 Counting Inversions

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, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.

Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Inversions
3-2, 4-2

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

15

Applications

Applications.
- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

16

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: $O(1)$.

1	5	4	8	10	2

6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2		6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2		6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

Counting Inversions: Combine

Combine: count blue-green inversions
- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

Count: O(n)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \leq T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)$$

21

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

22

Merge and Count

```
Merge-and-Count(A,B)
  Maintain a Current pointer into each list, initialized to
    point to the front elements
  Maintain a variable Count for the number of inversions,
    initialized to 0
  While both lists are nonempty:
    Let a_i and b_j be the elements pointed to by the Current pointer
    Append the smaller of these two to the output list
    If b_j is the smaller element then
      Increment Count by the number of elements remaining in A
    Endif
    Advance the Current pointer in the list from which the
      smaller element was selected.
  EndWhile
```


5.4 Closest Pair of Points

Closest Pair of Points

Closest pair. 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.

fast closest pair inspired fast algorithms for these problems

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

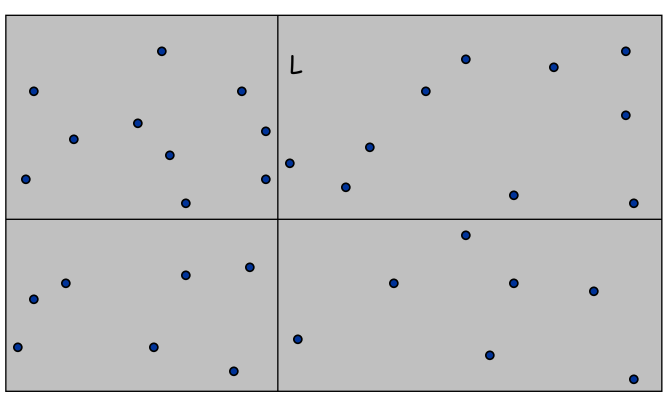
Assumption. No two points have same x coordinate.

to make presentation cleaner

25

Closest Pair of Points: First Attempt

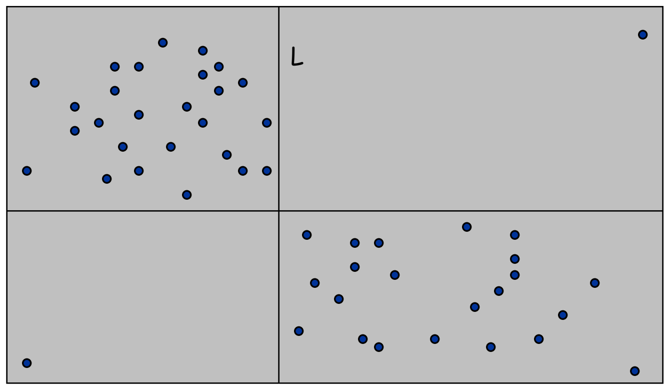
Divide. Sub-divide region into 4 quadrants.



26

Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.
Obstacle. Impossible to ensure n/4 points in each piece.

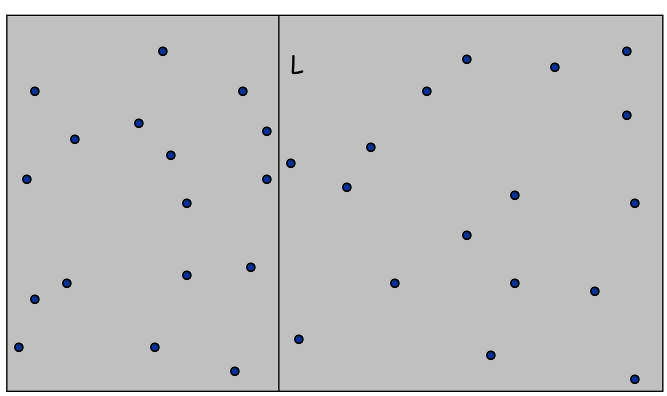


27

Closest Pair of Points

Algorithm.

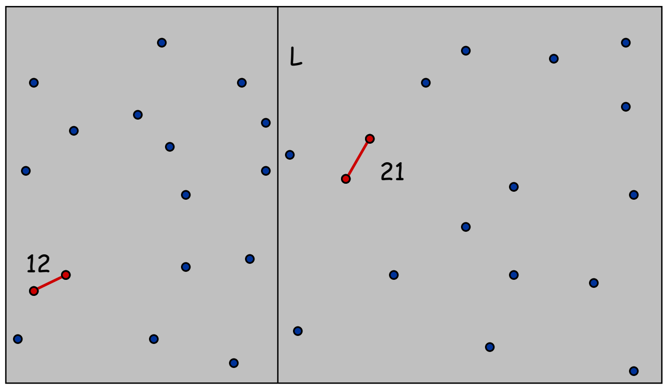
- Divide: draw vertical line L so that roughly $\frac{1}{2}$n points on each side.



28

Closest Pair of Points

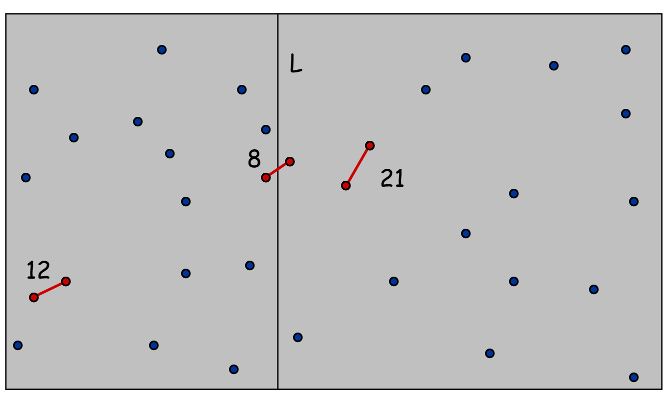
Algorithm.
- Divide: draw vertical line L so that roughly $\frac{1}{2}$n points on each side.
- Conquer: find closest pair in each side recursively.



29

Closest Pair of Points

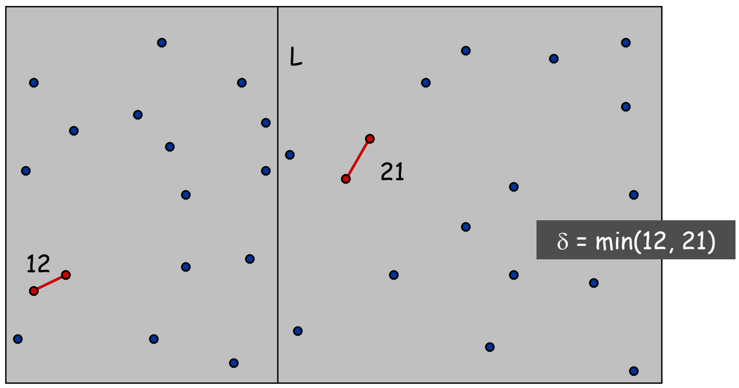
Algorithm.
- Divide: draw vertical line L so that roughly $\frac{1}{2}$n points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



30

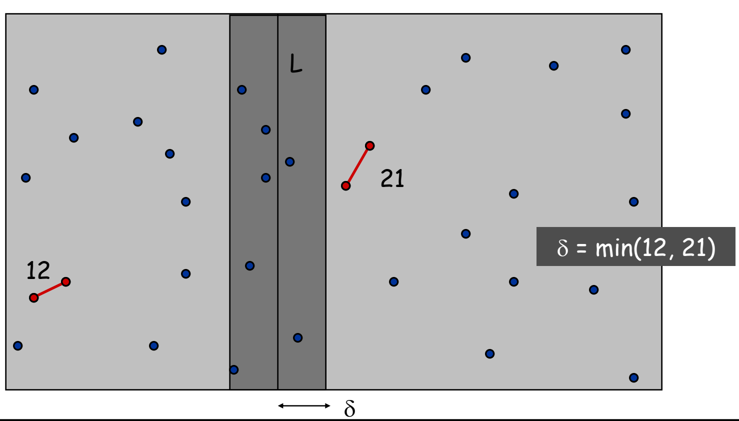
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.



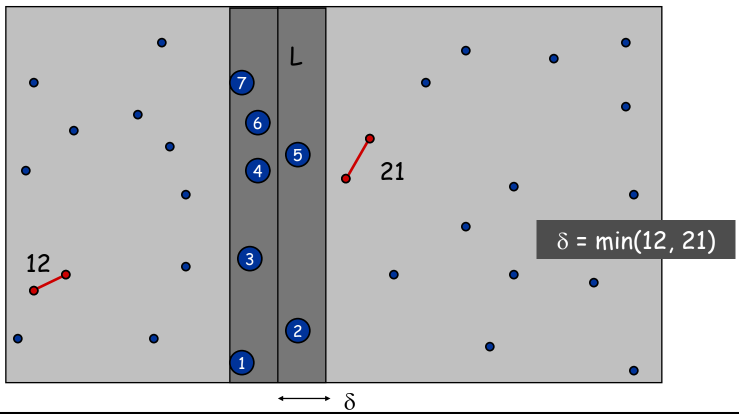
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

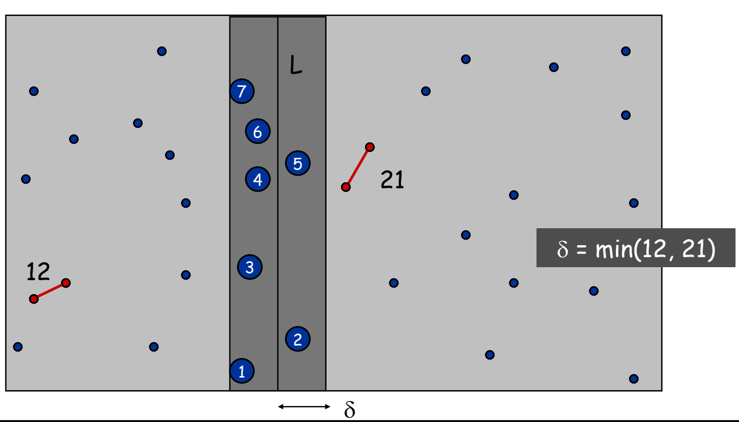
Find closest pair with one point in each side, assuming that distance < δ.
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



33

Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!



34

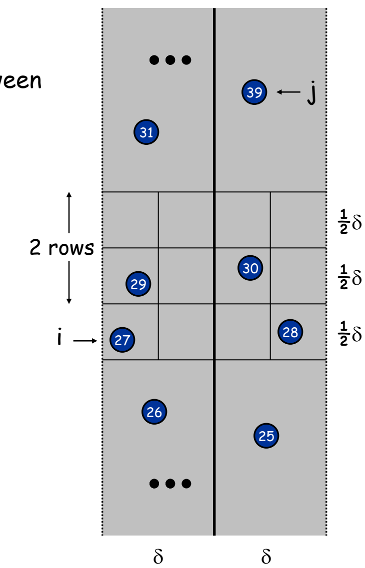
Closest Pair of Points

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

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪



35

Closest Pair Algorithm

```
Closest-Pair(p₁, …, pₙ) {
   Compute separation line L such the half the points        O(n log n)
   are on one side and half on the other side.

   δ₁ = Closest-Pair(left half)                              2T(n / 2)
   δ₂ = Closest-Pair(right half)
   δ  = min(δ₁, δ₂)

   Delete all points further than δ from separation line L   O(n)

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

   Scan points in y-order and compare distance between       O(n)
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

36

Closest Pair of Points: Analysis

Running time.

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

Q. Can we achieve O(n log n)?

A. Yes. Don't sort points in strip from scratch each time.
- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

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