Objectives

Algorithm Approach: Divide and Conquer

- Counting inversions
- Closest pair of points

Divide-and-Conquer

Divide et impera.
Veni, vidi, vici.
- Julius Caesar

Divide-and-conquer process

- 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 ½n
- Solve two parts recursively
- Combine two solutions into overall solution

Review: Recurrence Relations

Use recurrences to analyze/determine the run time of divide and conquer algorithms

- Number of sub problems
- Size of sub problems
- Number of times divided (number of levels)
- Cost of merging problems

How to solve

- Unrolling
- Substitution

COUNTING INVERSIONS

Comparing Rankings

To determine similarity of rankings, need a metric

Similarity metric: number of inversions between two rankings

- My rank: 1, 2, …, n
- Your rank: a_1, a_2, …, a_n
- Movies i and j inverted if $i < j$, but $a_i > a_j$

Naïve/Brute force solution?

Movies

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

Inversions:
3-2, 4-2

Forming a Better Solution

Better than brute force $\Theta(n^2)$

- Can't look at each inversion individually

Counting Inversions: Divide-and-Conquer

Assume number represents where item should be in the list

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

Counting Inversions: 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

What are the inversions?

Counting Inversions: 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: 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

seems like $\Theta(n^2)$

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

What would make figuring out blue-green inversions easier?

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

What does sorting do for us?
What is our algorithm for counting the inversions and merging?

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

Count: O(n)

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

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

Merge: O(n)

Merge and Count

```
Merge-and-Count(A,B)
   i=0 (front of list A)
   j=0 (front of list B)
   inversions = 0
   output = []
   while A not empty and B not empty:
      output.append( min(A[i], B[j]) )
      if B[j] < A[i]:
         inversions += A.size - i (remaining elements
in A)
      update i or j (whichever had smaller element)
```


Merge and Count Step

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

A	3	7	10	14	18	19	B	2	11	16	17	23	25	two sorted halves

Output array

Total:

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6

Output array: 2

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6

Output array: 2

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6

Output array: 2 3

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6

Output array: 2 3

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6						

2	3	7										Output array

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6						

2	3	7										Output array

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6						

2	3	7	10									Output array

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6						

2	3	7	10									Output array

Total: 6

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3					

2	3	7	10	11								Output array

Total: 6 + 3

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3					

2	3	7	10	11								Output array

Total: 6 + 3

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 **14** 18 19 | 2 11 16 17 23 25 — two sorted halves

6 3

2 3 7 10 11 14 — Output array

Total: 6 + 3

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6 3

2 3 7 10 11 14 — Output array

Total: 6 + 3

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 **16** 17 23 25 — two sorted halves

6 3 2

2 3 7 10 11 14 16 — Output array

Total: 6 + 3 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6 3 2

2 3 7 10 11 14 16 — Output array

Total: 6 + 3 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 **17** 23 25 — two sorted halves

6 3 2 2

2 3 7 10 11 14 16 17 — Output array

Total: 6 + 3 + 2 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3 7 10 14 18 19 | 2 11 16 17 23 25 — two sorted halves

6 3 2 2

2 3 7 10 11 14 16 17 — Output array

Total: 6 + 3 + 2 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

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

6 3 2 2

2 3 7 10 11 14 16 17 18   Output array

Total: 6 + 3 + 2 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

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

6 3 2 2

2 3 7 10 11 14 16 17 18   Output array

Total: 6 + 3 + 2 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

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

6 3 2 2

2 3 7 10 11 14 16 17 18 19   Output array

Total: 6 + 3 + 2 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

first half exhausted

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

6 3 2 2

2 3 7 10 11 14 16 17 18 19   Output array

Total: 6 + 3 + 2 + 2

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

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

6 3 2 2 0

2 3 7 10 11 14 16 17 18 19 23   Output array

Total: 6 + 3 + 2 + 2 + 0

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

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

6 3 2 2 0

2 3 7 10 11 14 16 17 18 19 23   Output array

Total: 6 + 3 + 2 + 2 + 0

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

i = 0

3	7	10	14	18	19		2	11	16	17	23	25	two sorted halves
							6	3	2	2	0	0	

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

Total: 6 + 3 + 2 + 2 + 0 + 0

Merge and Count

Given two sorted halves, count number of inversions where a_i and a_j are in different halves

Combine two sorted halves into sorted whole

3	7	10	14	18	19		2	11	16	17	23	25	two sorted halves
							6	3	2	2	0	0	

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

Total: 6 + 3 + 2 + 2 + 0 + 0 = 13

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
```

Recurrence relation?

Analysis: What is the Recurrence Relation?

Recurrence Relation:

$T(n) \leq T(n/2) + T(n/2) + O(n)$

$\rightarrow T(n) \in O(n \log n)$

```
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
```


CLOSEST PAIR OF POINTS

Computational Geometry

Algorithms and data structures for geometrical objects

- Points, line segments, polygons, etc.
- Common motivator: large data sets → efficiency

Some Applications

- Graphics
- Robotics (motion planning and visibility problems)
- Geographic information systems (GIS) (geometrical location and search, route planning)
  - Terraflow

Closest Pair of Points

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

- Special case of nearest neighbor, Euclidean MST, Voronoi

fast closest pair inspired fast algorithms for these problems

Brute force?

Closest Pair of Points

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

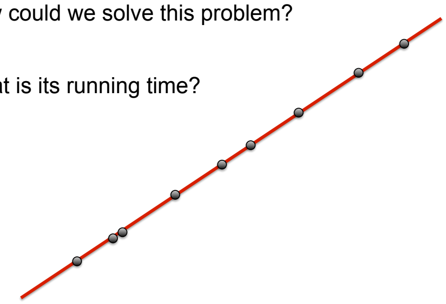
- Special case of nearest neighbor, Euclidean MST, Voronoi.

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

Simplify: All Points on a Line

How could we solve this problem?

What is its running time?



Simplify: All Points on a Line

How could we solve this problem?

- Sort the points
  - Monotonically increasing x/y coordinates
  - No closer points than neighbors in sorted list
- Step through, looking at the distances between each pair

What is it's running time?

- O(n logn)

Why won't this work for 2D?

Closest Pair of Points

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

- Special case of nearest neighbor, Euclidean MST, Voronoi.

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

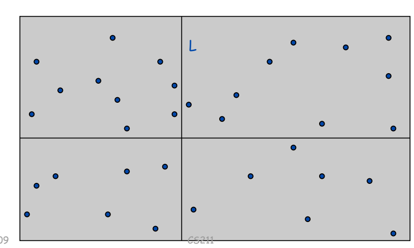
1-D version. O(n log n) easy if points are on a line

Assumption. No two points have same x coordinate

to make presentation cleaner

Closest Pair of Points: First Attempt

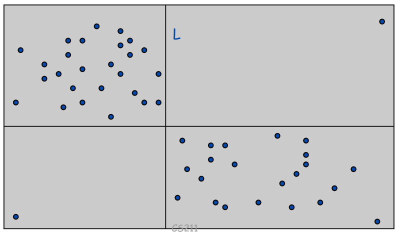
Divide. Sub-divide region into 4 quadrants



Closest Pair of Points: First Attempt

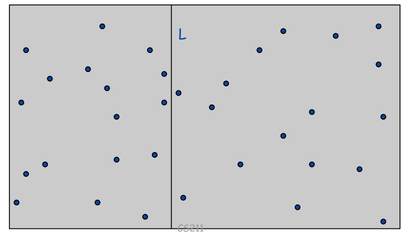
Divide. Sub-divide region into 4 quadrants

Obstacle. Impossible to ensure n/4 points in each piece



Closest Pair of Points

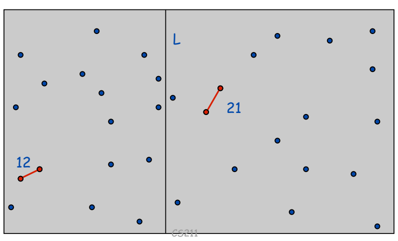
Divide: draw vertical line L so that roughly ½n points on each side



Closest Pair of Points

Divide: draw vertical line L so that roughly ½n points on each side

Conquer: find closest pair in each side recursively



Closest Pair of Points

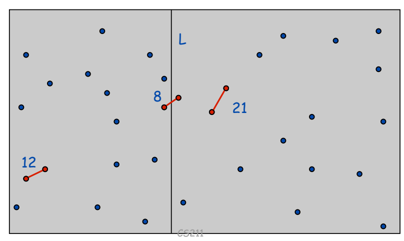
Divide: draw vertical line L so that roughly ½n points on each side

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

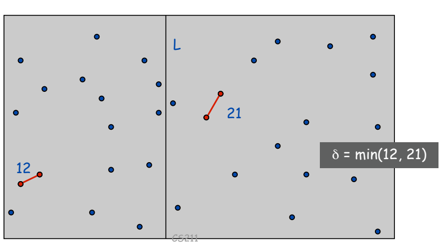
Do we need to check all pairs?



Closest Pair of Points

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

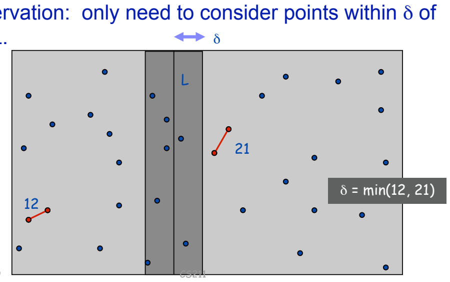
where δ = min(left_min_dist, right_min_dist)



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.

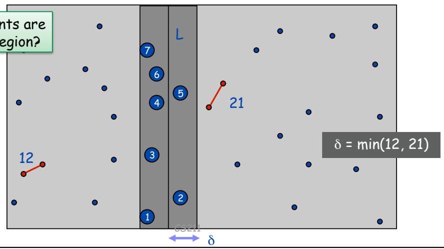


Closest Pair of Points

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

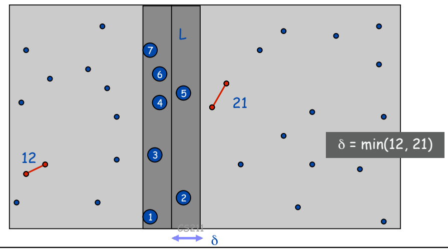
How many points are within that region?



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 checks distances of those within 11 positions in sorted list!



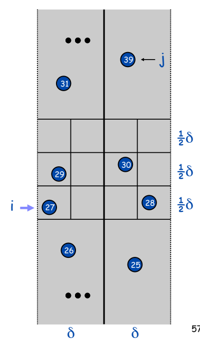
Analyzing Cost of Combining

Prepare minds to be blown…

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

- What is the distance of the box?
- How many points can be in a box?
- When do we know that points are > δ apart?



Analyzing Cost of Combining

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

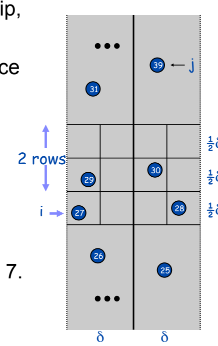
Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ

Pf.

- No two points lie in same ½δ-by-½δ box
- Two points at least 2 rows apart have distance ≥ 2(½δ). ▪

Fact. Still true if we replace 12 with 7.

Cost of combining is therefore…?



Closest Pair Algorithm

```
Closest-Pair(p1, …, pn)                                          
   Compute separation line L such that 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              O(n)
line L

   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 7 neighbors. If any of these
   distances is less than δ, update δ.

   return δ
```

Total running time?

Closest Pair of Points: Analysis

Running time. Solved in 5.2

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

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

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

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