

CSCI 355: ALGORITHM DESIGN AND ANALYSIS

6. DIVIDE AND CONQUER I

- › mergesort
- › randomized quicksort
- › median and selection
- › closest pair of points

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide a problem of size n into two subproblems of size $n/2$. ← $O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. ← $O(n)$ time

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $O(n \log n)$.

3

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Examples.

- Mergesort and quicksort (sorting).
- Mergehull and quickhull (convex hull).
- Shamos–Hoey algorithm (closest pair).
- Median-of-medians algorithm (selection).
- Strassen's algorithm (matrix multiplication).
- Karatsuba's algorithm (integer multiplication).
- Cooley–Tukey algorithm (fast Fourier transform).



4

CSCI 355: ALGORITHM DESIGN AND ANALYSIS
6. DIVIDE AND CONQUER I

- ▶ mergesort
- ▶ randomized quicksort
- ▶ median and selection
- ▶ closest pair of points

Sorting problem

Problem. Given a list L of n elements from a totally ordered universe, rearrange them in ascending order.



Sorting applications

Obvious applications.

- Organize a music library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.

- Identify statistical outliers.
- Perform binary search in a database.
- Remove duplicates in a mailing list.

Non-obvious applications.

- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Scheduling to minimize maximum lateness.
- Minimum spanning trees (Kruskal's algorithm).
- ...

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.



input

A L G O R I T H M S

sort left half

A G L O R I T H M S

sort right half

A G L O R H I M S T

merge results

A G H I L M O R S T

8

Mergesort: merging



Goal. 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).

sorted list A

3 7 10 a_i 18



sorted list B

2 11 b_j 20 23



merge to form sorted list C

2 3 7 10 11



9

Mergesort: implementation

Input. List L of n elements from a totally ordered universe.

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). $\leftarrow T(n/2)$

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

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

RETURN L .

10

Mergesort: proof of correctness

Proposition. Mergesort sorts any list of n elements.

Pf. [by strong induction on n]

- Base case: $n = 1$. Clearly, a list of 1 element is already sorted.
- Inductive hypothesis: assume true for $1, 2, \dots, n-1$.
- By the inductive hypothesis, mergesort sorts both left and right halves.
- Merging operation combines two sorted lists into a sorted whole. ▀

11

A useful recurrence relation

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

Recurrence.

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

↙ between $\lfloor n/2 \rfloor$ and $n-1$ comparisons

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

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.

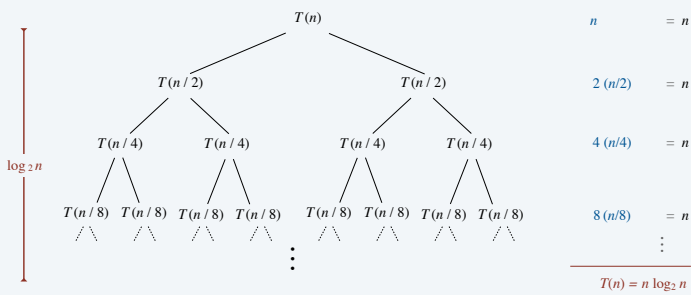
12

Divide-and-conquer recurrence: recursion tree

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

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

↙ assuming n is a power of 2



13

Divide-and-conquer recurrence: proof by induction

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

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

← assuming n is a power of 2

Pf. [by induction on n]

- Base case: when $n = 1$, $T(1) = 0 = n \log_2 n$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) & \stackrel{\text{recurrence}}{=} 2T(n) + 2n \\ \text{inductive hypothesis} \rightarrow & = 2n \log_2 n + 2n \\ & = 2n (\log_2 (2n) - 1) + 2n \\ & = 2n \log_2 (2n). \quad \blacksquare \end{aligned}$$

14

Mergesort: analysis

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

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

← no longer assuming n is a power of 2

Pf. [by strong induction on n]

- Base case: $n = 1$.
- Define $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$ and note that $n = n_1 + n_2$.
- Induction step: assume true for $1, 2, \dots, n-1$.

$$\begin{aligned} T(n) & \leq T(n_1) + T(n_2) + n \\ \text{inductive hypothesis} \rightarrow & \leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ & \leq n_1 \lceil \log_2 n \rceil + n_2 \lceil \log_2 n \rceil + n \\ & = n \lceil \log_2 n \rceil + n \\ & \leq n (\lceil \log_2 n \rceil - 1) + n \leftarrow \log_2 n_2 \leq \lceil \log_2 n \rceil - 1 \\ & = n \lceil \log_2 n \rceil. \quad \blacksquare \end{aligned}$$

$$\begin{aligned} n_2 & = \lceil n/2 \rceil \\ & \leq \lceil 2^{\lceil \log_2 n \rceil} / 2 \rceil \\ & = 2^{\lceil \log_2 n \rceil} / 2 \end{aligned}$$

← an integer

Sorting lower bound

Challenge. How to prove a lower bound for **all** conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of comparisons.

Q. Realistic model?

A1. Yes: Java, Python, C++, ...

A2. Yes: Mergesort, insertion sort, quicksort, heapsort, ...

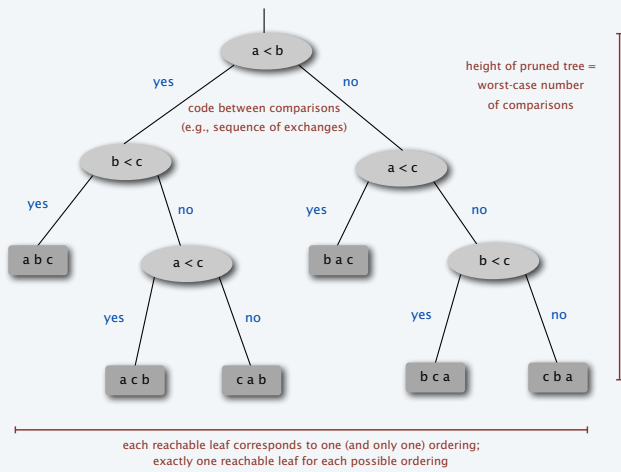
A3. No: Bucket sort, radix sorts, ...

sort(*, key=None, reverse=False)

This method sorts the list in place, using only $<$ comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

16

Comparison tree (for 3 distinct keys a, b, and c)



17

Sorting lower bound

Theorem. Any deterministic comparison-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n .
- Worst-case number of comparisons = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow n!$ reachable leaves.

$$\begin{aligned} 2^h &\geq \# \text{ reachable leaves} = n! \\ \Rightarrow h &\geq \log_2(n!) \\ &\geq n \log_2 n - n / \ln 2 \quad \blacksquare \end{aligned}$$

↑
Stirling's formula



Note. Lower bound can be extended to include randomized algorithms.

19

CSCI 355: ALGORITHM DESIGN AND ANALYSIS

6. DIVIDE AND CONQUER I

- ▶ mergesort
- ▶ randomized quicksort
- ▶ median and selection
- ▶ closest pair of points

Randomized quicksort: analysis

Proposition. The expected number of comparisons to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- $\Pr [a_i \text{ and } a_j \text{ are compared}] = 2 / (j - i + 1)$, where $i < j$.

$$\begin{aligned} \text{Expected number of comparisons} &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} = 2 \sum_{i=1}^n \sum_{j=2}^{n-i+1} \frac{1}{j} \\ &\leq 2n \sum_{j=1}^n \frac{1}{j} \\ &\leq 2n (\ln n + 1) \quad \blacksquare \end{aligned}$$

↑ all pairs i and j
↑ harmonic sum

Remark. # of comparisons decreases only if there are equal elements.

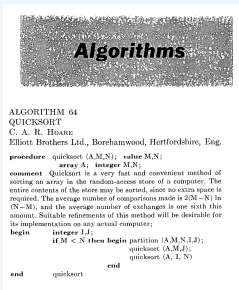
28

Randomized quicksort

- Tony Hoare invented quicksort to translate Russian into English.
[after realizing insertion sort wasn't good enough!]
- Learned Algol 60 (and recursion) to implement it.
- Published the algorithm in CACM.



Tony Hoare



Communications of the ACM (July 1961)

29

CSCI 355: ALGORITHM DESIGN AND ANALYSIS

6. DIVIDE AND CONQUER I

- › mergesort
- › randomized quicksort
- › median and selection
- › closest pair of points

Median and selection problems

Selection. Given n elements from a totally ordered universe, find k^{th} smallest.

- Minimum: $k = 1$; maximum: $k = n$.
- Median: $k = \lfloor (n + 1) / 2 \rfloor$.
- $O(n)$ compares for min or max.
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap. ← max heap with k smallest

Applications. Order statistics; finding the “top k ”; bottleneck paths, ...

Q. Can we do it with $O(n)$ comparisons?

A. Yes! Selection is easier than sorting.

31

Randomized quickselect

see 6B

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

QUICK-SELECT(A, k)

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, p)$. ← $\Theta(n)$

IF $(k \leq |L|)$ RETURN QUICK-SELECT(L, k). ← $T(i)$

ELSE IF $(k > |L| + |M|)$ RETURN QUICK-SELECT($R, k - |L| - |M|$) ← $T(n - i - 1)$

ELSE IF $(k = |L|)$ RETURN p .

32

Randomized quickselect: analysis

Intuition. Split candy bar uniformly \Rightarrow expected size of larger piece is $3/4$.

$$T(n) \leq T(3n/4) + n \Rightarrow T(n) \leq 4n$$

not rigorous: can't assume $E[T(i)] \leq T(E[i])$



Def. $T(n, k)$ = expected # comparisons to select k^{th} smallest in array of length $\leq n$.

Def. $T(n) = \max_k T(n, k)$.

Proposition. $T(n) \leq 4n$.

Pf. [by strong induction on n]

- Assume true for $1, 2, \dots, n - 1$.
- $T(n)$ satisfies the following recurrence:

$$T(n) \leq n + 1/n [2T(n/2) + \dots + 2T(n-3) + 2T(n-2) + 2T(n-1)]$$

$$\leq n + 1/n [8(n/2) + \dots + 8(n-3) + 8(n-2) + 8(n-1)]$$

$$\leq n + 1/n (3n^2)$$

$$= 4n. \quad \blacksquare$$

tiny cheat: sum should start at $T(\lfloor n/2 \rfloor)$

can assume we always recur of larger of two subarrays since $T(n)$ is monotone non-decreasing

inductive hypothesis

33

Selection in worst-case linear time

Goal. Find a pivot element p that divides the list of n elements into two pieces so that each piece is **guaranteed** to have $\leq 7/10 n$ elements.

Q. How to find approximate median in linear time?

A. Recursively compute median of sample of $\leq 2/10 n$ elements.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(7/10 n) + T(2/10 n) + \Theta(n) & \text{otherwise} \end{cases}$$

two subproblems of different sizes!

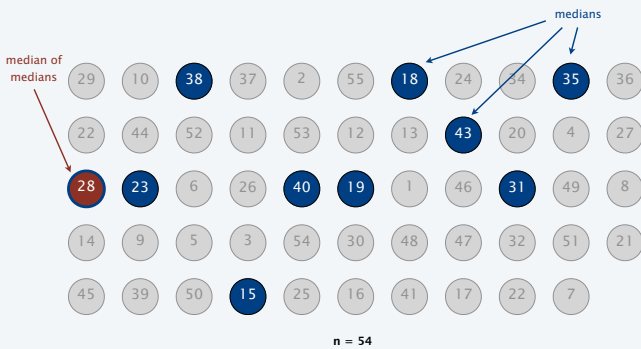
$$\rightarrow T(n) = \Theta(n)$$

we'll need to show this

34

Choosing the pivot element

- Divide n elements into $\lfloor n/5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of $\lfloor n/5 \rfloor$ medians recursively.
- Use median-of-medians as pivot element.



37

Median-of-medians selection algorithm

MOM-SELECT(A, k)

$n \leftarrow |A|$.

IF ($n < 50$)

 RETURN k^{th} smallest of element of A via mergesort.

Group A into $\lfloor n/5 \rfloor$ groups of 5 elements each (ignore leftovers).

$B \leftarrow$ median of each group of 5.

$p \leftarrow$ MOM-SELECT($B, \lfloor n/10 \rfloor$) ← median of medians

$(L, M, R) \leftarrow$ PARTITION-3-WAY(A, p).

IF ($k \leq |L|$) RETURN MOM-SELECT(L, k).

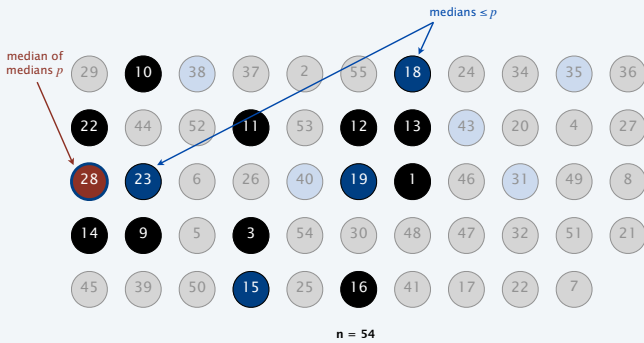
ELSE IF ($k > |L| + |M|$) RETURN MOM-SELECT($R, k - |L| - |M|$)

ELSE RETURN p .

38

Median-of-medians selection algorithm: analysis

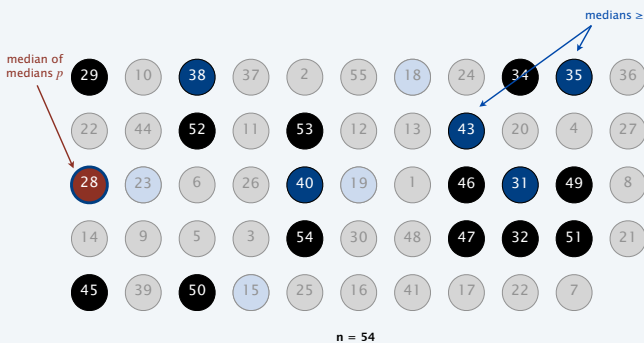
- At least half of 5-element medians $\leq p$.
- At least $\lfloor \lfloor n/5 \rfloor / 2 \rfloor = \lfloor n/10 \rfloor$ medians $\leq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$.



41

Median-of-medians selection algorithm: analysis

- At least half of 5-element medians $\geq p$.
- At least $\lfloor \lfloor n/5 \rfloor / 2 \rfloor = \lfloor n/10 \rfloor$ medians $\geq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\geq p$.



44

Median-of-medians selection algorithm: recurrence

Median-of-medians selection algorithm recurrence.

- Select called recursively with $\lfloor n/5 \rfloor$ elements to compute MOM p .
- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\geq p$.
- Select called recursively with at most $n - 3 \lfloor n/10 \rfloor$ elements.

Def. $C(n)$ = max # of comparisons on any array of n elements.

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n$$

median of medians
recursive select
computing median of 5 (≤ 6 compares per group) partitioning ($\leq n$ compares)

Intuition.

- $C(n)$ is going to be at least linear in $n \Rightarrow C(n)$ is super-additive.
- Ignoring floors, this implies that $C(n) \leq C(n/5 + n - 3n/10) + 11/5 n$
 $= C(9n/10) + 11/5 n$
 $\Rightarrow C(n) \leq 22n$.

45

Median-of-medians selection algorithm: recurrence

Analysis of selection algorithm recurrence.

- $T(n) = \max$ # comparisons on any array of $\leq n$ elements.
- $T(n)$ is monotone non-decreasing, but $C(n)$ is not!

$$T(n) \leq \begin{cases} 6n & \text{if } n < 50 \\ \max\{T(n-1), T(\lfloor n/5 \rfloor) + T(n-3\lfloor n/10 \rfloor) + \frac{11}{5}n\} & \text{if } n \geq 50 \end{cases}$$

Claim. $T(n) \leq 44n$.

Pf. [by strong induction]

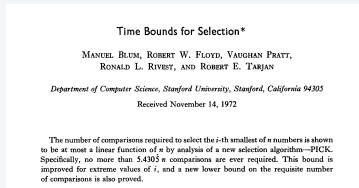
- Base case: $T(n) \leq 6n$ for $n < 50$ (mergesort).
- Inductive hypothesis: assume true for $1, 2, \dots, n-1$.
- Induction step: for $n \geq 50$, we have either $T(n) \leq T(n-1) \leq 44n$ or

$$\begin{aligned} T(n) &\leq T(\lfloor n/5 \rfloor) + T(n-3\lfloor n/10 \rfloor) + 11/5 n \\ \text{inductive hypothesis} \longrightarrow &\leq 44(\lfloor n/5 \rfloor) + 44(n-3\lfloor n/10 \rfloor) + 11/5 n \\ &\leq 44(n/5) + 44n - 44(n/4) + 11/5 n \longleftarrow \text{for } n \geq 50, 3\lfloor n/10 \rfloor \geq n/4 \\ &= 44n. \quad \blacksquare \end{aligned}$$

46

Linear-time selection retrospective

Proposition. [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a comparison-based selection algorithm whose worst-case running time is $O(n)$.



Theory.

- Optimized version of BFPRT: $\leq 5.4305 n$ comparisons.
- Upper bound: [Dor–Zwick 1995] $\leq 2.95 n$ comparisons.
- Lower bound: [Dor–Zwick 1999] $\geq (2 + 2^{-80}) n$ comparisons.



Practice. BFPRT constants too large to be useful.

47

CSCI 355: ALGORITHM DESIGN AND ANALYSIS 6. DIVIDE AND CONQUER I

- › mergesort
- › randomized quicksort
- › median and selection
- › closest pair of points

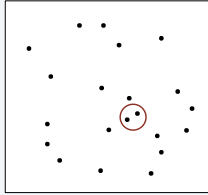
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the 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



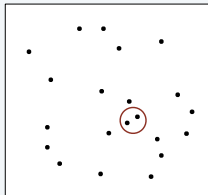
49

Closest pair 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.

Non-degeneracy assumption. No two points have the same x -coordinate.

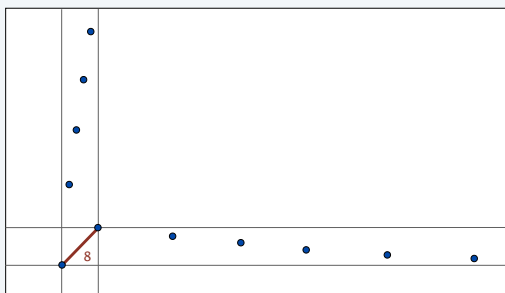


50

Closest pair of points: first attempt

Sorting solution.

- Sort by x -coordinate and consider nearby points.
- Sort by y -coordinate and consider nearby points.

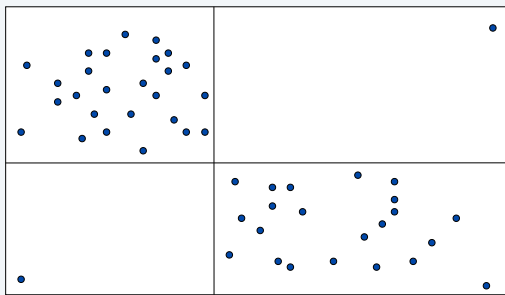


52

Closest pair of points: second attempt

Plan. Subdivide region into 4 quadrants.

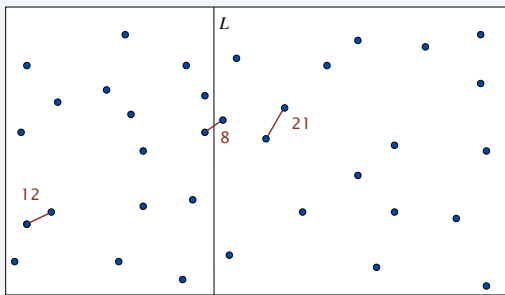
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest pair of points: divide-and-conquer algorithm

- **Divide:** draw 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.

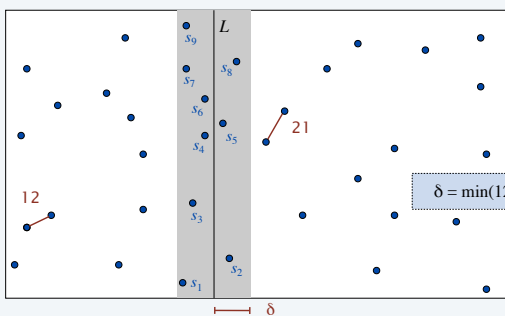
seems like $\Theta(n^2)$



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

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

why?



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR(p_1, p_2, \dots, p_n)

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

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

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

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

$A \leftarrow$ list of all points closer than δ to line L . $\leftarrow O(n)$

Sort points in A by y-coordinate. $\leftarrow O(n \log n)$

Scan points in A in y-order and compare distance between each point and next 7 neighbors. $\leftarrow O(n)$

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

RETURN δ .

58

Closest pair of points: analysis

Theorem. The divide-and-conquer algorithm for finding a closest pair of points in the plane can be implemented in $O(n \log^2 n)$ time.

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

59

Digression: computational geometry

Ingenious divide-and-conquer algorithms for core geometric problems.

problem	brute	clever
closest pair	$O(n^2)$	$O(n \log n)$
convex hull	$O(n^2)$	$O(n \log n)$
farthest pair	$O(n^2)$	$O(n \log n)$
Delaunay/Voronoi	$O(n^4)$	$O(n \log n)$
Euclidean MST	$O(n^2)$	$O(n \log n)$



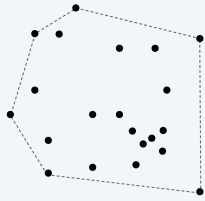
running time to solve a 2D problem with n points

Note. 3D and higher dimensions test limits of our ingenuity.

60

Convex hull

The **convex hull** of a set of n points is the smallest perimeter fence enclosing the points.



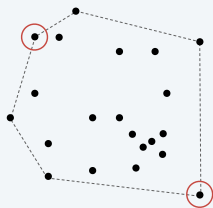
Equivalent definitions.

- Smallest area convex polygon enclosing the points.
- Intersection of all convex set containing all the points.

61

Farthest pair

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

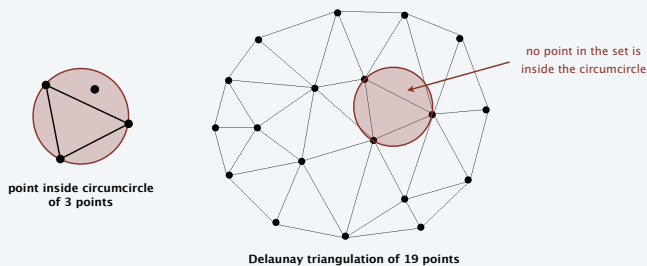


Fact. Points in farthest pair are extreme points on convex hull.

62

Delaunay triangulation

The **Delaunay triangulation** is a triangulation of n points in the plane such that no point is inside the circumcircle of any triangle.



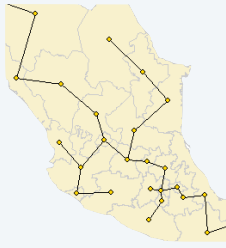
Some useful properties.

- No edges cross.
- Among all triangulations, it maximizes the minimum angle.
- Contains an edge between each point and its nearest neighbor.

63

Euclidean MST

Given n points in the plane, find MST connecting them.
[distances between point pairs are Euclidean distances]



Fact. Euclidean MST is subgraph of Delaunay triangulation.

Implication. We can compute the Euclidean MST in $O(n \log n)$ time.

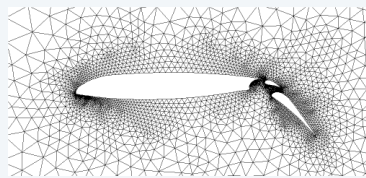
- Compute Delaunay triangulation.
- Compute MST of Delaunay triangulation. ← it's planar ($\leq 3n$ edges)

64

Computational geometry applications

Applications.

- Robotics.
- VLSI design.
- Data mining.
- Medical imaging.
- Computer vision.
- Scientific computing.
- Finite-element meshing.
- Astronomical simulation.
- Models of physical world.
- Geographic information systems.
- Computer graphics (movies, games, virtual reality).



airflow around an aircraft wing

65
