Algorithm Design and Implementation

Principle of Algorithms VI

Divide and Conquer I

Guoqiang Li

School of Software, Shanghai Jiao Tong University

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 problem of size n into two subproblems of size $n/2$. $\longleftarrow \mathcal{O}(n)$ time
- Solve (conquer) two subproblem recursively.
- Combine two solutions into overall solution. $\longleftarrow \mathcal{O}(n)$ time

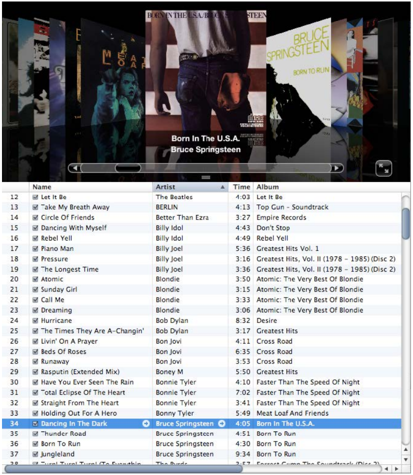
Consequence.

- Brute force: $\Theta\left(n^2\right)$
- Divide-and-conquer: $\mathcal{O}(n \log n)$

Mergesort

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 an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.

- Identify statistical outliers.
- 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).
- $\cdots$

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

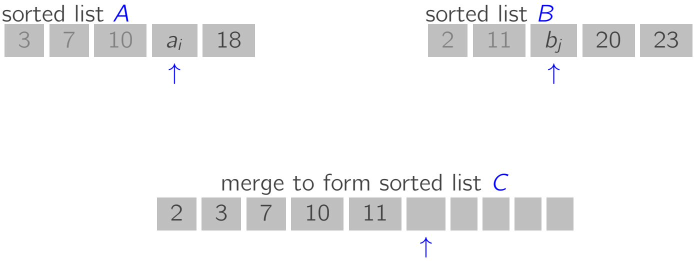
merge results

A	G	H	I	L	M	O	R	S	T

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).



Mergesort implementation

Input. List L of n elements from a totally ordered universe.
Output. The n elements in ascending order.

```
MergeSort(L)

if List L has one element then
    Return L;
end
Divide the list into two halves A and B;
A ← MergeSort (A);
B ← MergeSort (B);
L ← Merge (A,B);
Return L;
```

A useful recurrence relation

Definition

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

Recurrence.

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

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

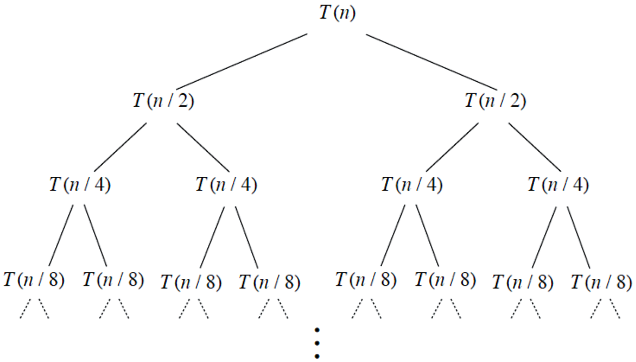
Assorted proofs. We describe several ways to solve this recurrence.
Initially we assume n is a power of 2 and replace $\leq$ with $=$ in the recurrence.

Divide-and-conquer recurrence: recursion tree

Proposition

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

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



Proof by induction

Proposition

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

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

Proof. [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 $T(2n) = 2n \log_2(2n)$.

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

Quiz 1

Which is the exact solution of the following recurrence?

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

A. $T(n) = n \lfloor \log_2 n \rfloor$

B. $T(n) = n \lceil \log_2 n \rceil$

C. $T(n) = n \lfloor \log_2 n \rfloor + 2^{\lfloor \log_2 n \rfloor} - 1$

D. $T(n) = n \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$

E. Not even Knuth knows.

Analysis of mergesort recurrence

Proposition

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

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

Proof. [by 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 $1, 2, \cdots, n-1$.

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n \lceil \log_2 n_2 \rceil + n \\ &\leq n (\lceil \log_2 n \rceil - 1) + n \\ &= n \lceil \log_2 n \rceil \end{aligned}$$

Digression: sorting lower bound

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

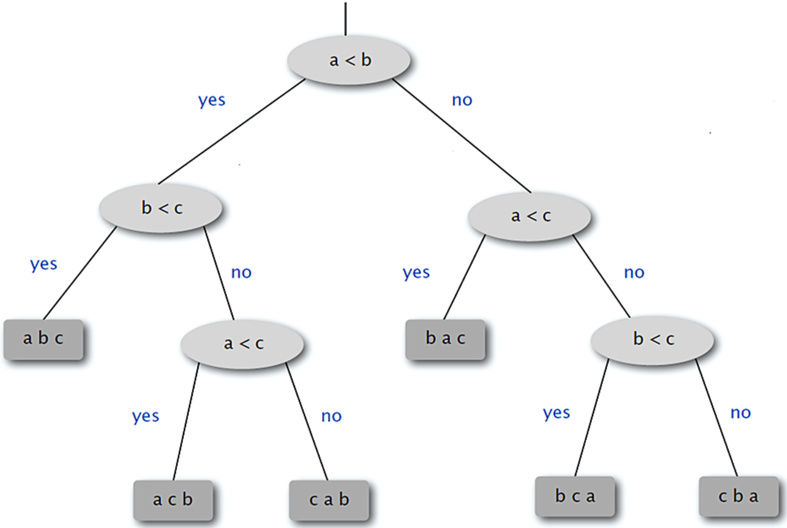
Model of computation. Comparison trees.

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

Cost model. Number of compares.

Q. Realistic model?
A1. Yes. Java, Python, C++, ...
A2. Yes. Mergesort, insertion sort, quicksort, heapsort, ...
A3. No. Bucket sort, radix sorts, ...

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



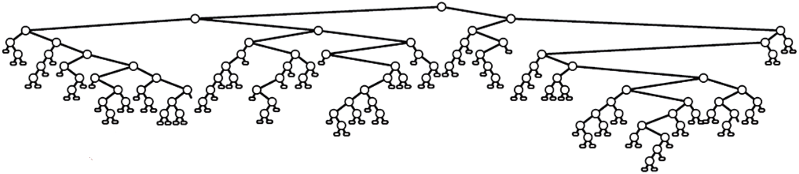
Sorting lower bound

Theorem

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

Proof.

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



Sorting lower bound

Theorem

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

Proof.

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

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

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

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 **inversions**: 3-2, 4-2

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

Counting inversions: 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).

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.

input

1	5	4	8	10	2	6	9	3	7

count inversions in left half A

1	5	4	8	10

5-4

count inversions in right half B

2	6	9	3	7

6-3 9-3 9-7

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

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

Counting inversions: how to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?
A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element $b \in B$,
  - binary search in A to find how elements in A are greater than b.

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

binary search to count inversions (a, b) with $a \in A$ and $b \in B$

3	7	10	14	18

2	11	16	20	23
5	2	1	0	0

Counting inversions: how to combine two subproblems?

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- 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.

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

3	7	10	a_i	18
			$\uparrow$	

2	11	b_j	20	23
5	2	$\uparrow$		

merge to form sorted list C

2	3	7	10	11					
				$\uparrow$					

Counting inversions: divide-and-conquer algorithm implementation

Sort-and-Count(L);
input : List L
output: Number of inversions in L and L in sorted order

if *List L has one element* **then**
  Return $(0, L)$;
end
Divide the list into two halves A and B;
$(r_A, A) \leftarrow$ Sort-and-Count (A);
$(r_B, B) \leftarrow$ Sort-and-Count (B);
$(r_{AB}, L) \leftarrow$ Merge-and-Count (A, B);
Return $(r_A + r_B + r_{AB}, L)$;

Counting inversions: divide-and-conquer algorithm analysis

Proposition

The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Proof.

The worst-case running time $T(n)$ satisfies the recurrence:

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

Median and Selection

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; find the "top k; bottleneck paths, $\cdots$

Q. Can we do it with $O(n)$ compares?
A. Yes! Selection is easier than sorting.

Randomized quicksort

- 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.

Select(A, K)

Pick pivot $p \in A$ uniformly at random;
$(L, M, R) \leftarrow$ Partition(A,p);
if $k \leq |L|$ **then** Return Select(L, k);
else if $k > |L| + |M|$ **then** Return Select(R, $k - |L| - |M|$);
else Return p;

Randomized quickselect analysis

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

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

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

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

Randomized quickselect analysis

Proposition

$T(n) \leq 4n$

Proof. [by strong induction on n]

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

$$\begin{aligned} T(n) &\leq n + 1/n[2T(n/2) + \ldots + 2T(n-3) + 2T(n-2) + 2T(n-1)] \\ &\leq n + 1/n[8(n/2) + \ldots + 8(n-3) + 8(n-2) + 8(n-1)] \\ &\leq n + 1/n\left(3n^2\right) \\ &= 4n. \end{aligned}$$

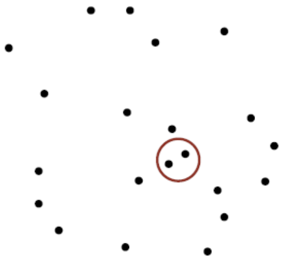
Closest Pair of Points

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.



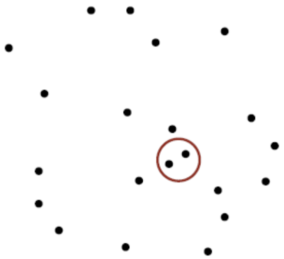
Closest pair of points

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.

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

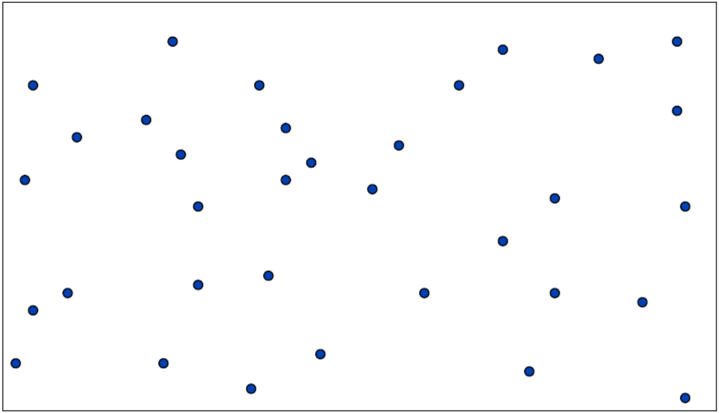
Non-degeneracy assumption. No two points have the same x-coordinate.



Closest pair of points: first attempt

Sorting solution.

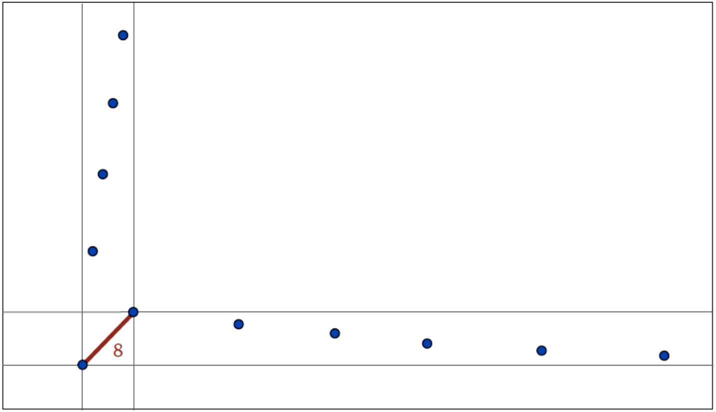
- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest pair of points: first attempt

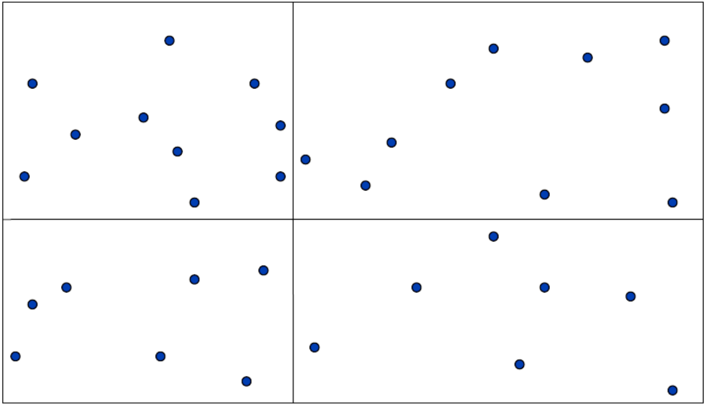
Sorting solution.

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



Closest pair of points: second attempt

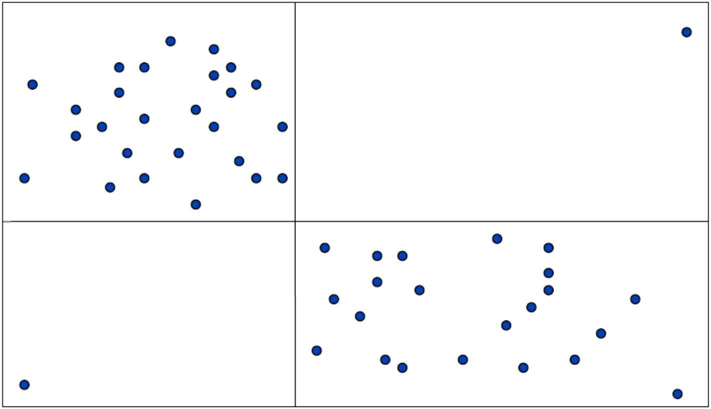
Divide. Subdivide region into 4 quadrants.



Closest pair of points: second attempt

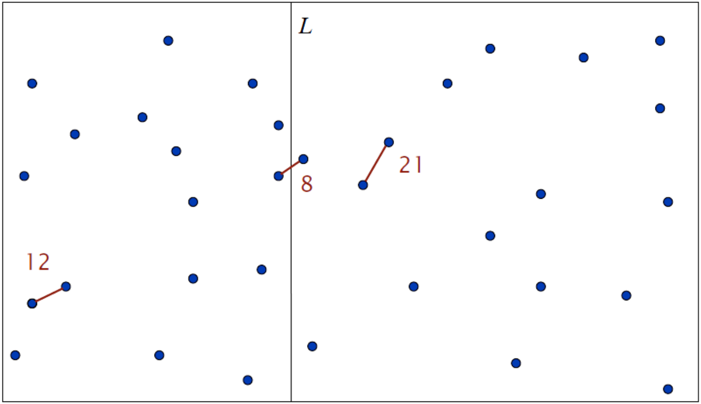
Divide. Subdivide region into 4 quadrants.

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



Closest pair of points: divide-and-conquer algorithm

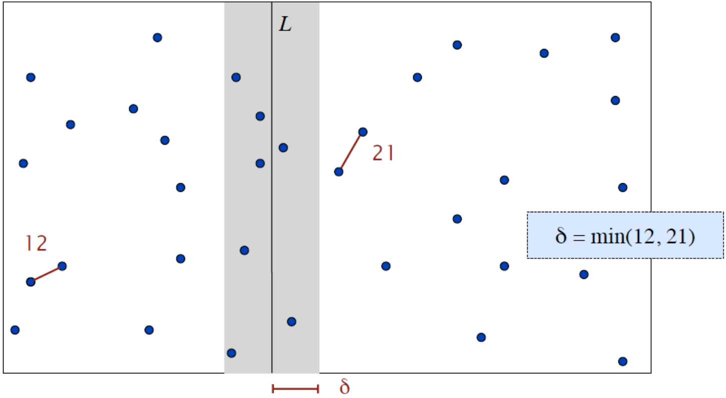
- Divide: draw vertical line L so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.



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

Find closest pair with one point in each side, assuming that distance $< \delta$.

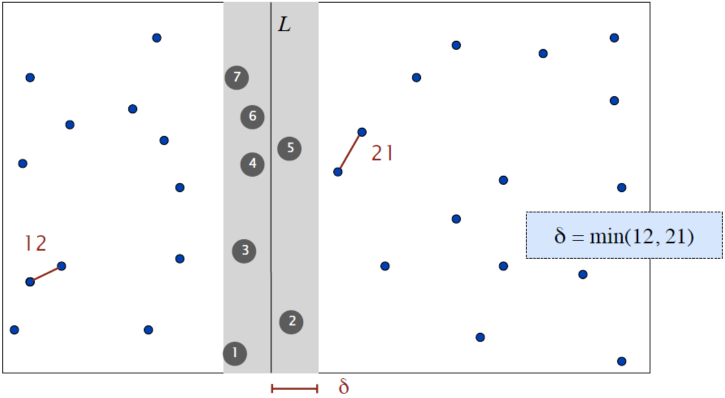
- Observation: suffices to consider only those points within δ of line L.



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

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 δ-strip by their y-coordinate.
- Check distances of only those points within **7** positions in sorted list!



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

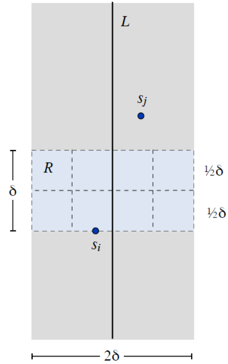
Definition Let s_i be the point in the 2 δ-strip, with the i^{th} smallest y-coordinate.

Proposition

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

Proof.

- 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 of points: divide-and-conquer algorithm

```
Closest-Pair((p_1, p_2, ..., p_n))

Compute vertical line L such that half the points are on each side of
  the line;
δ_1 ← Closest-Pair(points in left half);
δ_2 ← Closest-Pair(points in right half);
δ ← min {δ_1, δ_2};
Delete all points further than δ from line L;
Sort remaining points by y-coordinate;
Scan points in y-order and compare distance between each point
  and next 7 neighbors;
if any of these distances is less than δ then
 |  Update(δ)
end
Return δ;
```

Quiz 6

What is the solution to the following recurrence?

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

A. $T(n) = \Theta(n)$.

B. $T(n) = \Theta(n \log n)$.

C. $T(n) = \Theta\left(n \log^2 n\right)$.

D. $T(n) = \Theta\left(n^2\right)$.

Refined version of closest-pair algorithm

Q. How to improve to $O(n \log n)$

A. 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.

Theorem (Shamos 1975)

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

Quiz 7

What is the complexity of the 2D closest pair problem?

A. $\Theta(n)$.

B. $\Theta\left(n \log^* n\right)$

C. $\Theta(n \log \log n)$.

D. $\Theta(n \log n)$.

E. Not even Tarjan knows.

Computational complexity of closest-pair problem

Theorem (Ben-Or 1983, Yao 1989)

In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.

Theorem (Rabin 1976)

There exists an algorithm to find the closest pair of points in the plane whose expected running time is $O(n)$.

Lower Bounds for Algebraic Computation Trees with Integer Inputs*

Andrew Chi-Chih Yao
Department of Computer Science
Princeton University
Princeton, New Jersey 08544

A NOTE ON RABIN'S NEAREST-NEIGHBOR ALGORITHM*

Steve FORTUNE and John HOPCROFT
Department of Computer Science, Cornell University, Ithaca, NY, U.S.A.

Received 20 July 1978, revised version received 21 August 1978

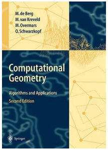
Probabilistic algorithms, nearest neighbor, hashing

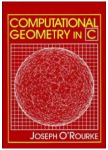
Digression: computational geometry

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

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

running time to solve a 2D problem with n points

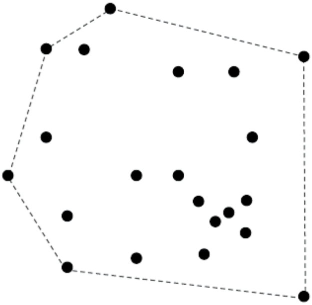




Note. 3D and higher dimensions test limits of our ingenuity.

Convex hull

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

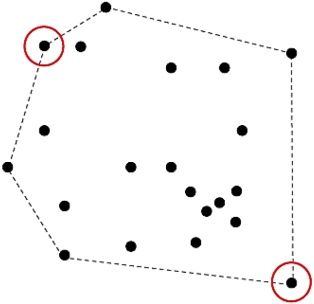


Equivalent definitions.

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

Farthest pair

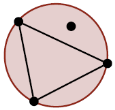
Given n points in the plane, find a pair of points with the largest Euclidean distance between them.

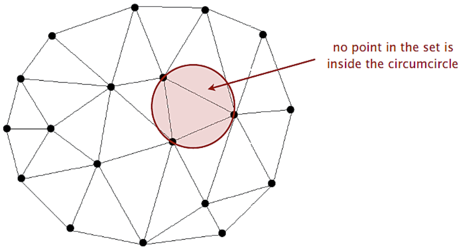


Fact. Points in farthest pair are extreme points on convex hull.

Delaunay triangulation

The Delaunay triangulation is a triangulation of n points in the plane such that no point is inside the circum circle 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.

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.

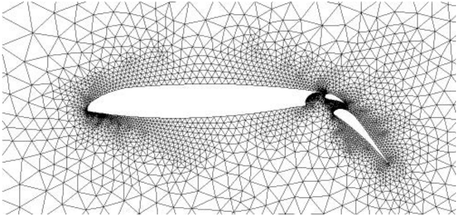
Implication. Can compute Euclidean MST in $O(n \log n)$ time.

- Compute Delaunay triangulation.
- Compute MST of Delaunay triangulation.

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