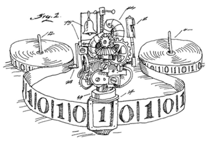
eecs376.org

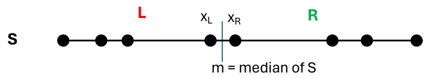
# EECS 376: Foundations of Computer Science

**Lecture 03 - Divide and Conquer 2**





## Building intuition: Divide and Conquer 1D




Not any better than before, but again we're here to build intuition.

**Algorithm:** Closest-Pair(set of n points S)
- Find the median m (There is an O(n) algorithm…)
- Split the points according to m to L and R
- $\delta_L$ = dist(Closest-Pair(L)) ①
- $\delta_R$ = dist(Closest-Pair(R)) ②
- Find the **maximal** element $x_L \in L$ $|x_L - x_R|$ ③
- Find the **minimal** element $x_R \in R$ ↓ compare ①②③
- **Return**: the pair that lies within distance min($\delta_L$, $\delta_R$, $|x_L - x_R|$)

**Runtime Analysis:**
- T(n) = 2T(n/2) + O(n)

$\frac{k}{b^d} = \frac{2}{2^1} = 1 \Rightarrow O(n \log n)$

**Formally:** Consider the recurrence relation $T(n) = kT(n/b) + O(n^d)$, when $k, b > 1$. Then:

$$T(n) = \begin{cases} O(n^d) & \text{if } (k/b^d) < 1 \\ O(n^d \log n) & \text{if } (k/b^d) = 1 \\ O(n^{\log_b k}) & \text{if } (k/b^d) > 1 \end{cases}$$



Another example of divide and conquer:

## Closest Pair of Points

**Closest Pair Data Structures: Applications**

The following algorithms and applications can be implemented efficiently using our new closest pair data structures, or involve closest pair computation as important subroutines.

- Dynamic minimum spanning trees
- Two-optimization heuristics in combinatorial optimization
- Straight skeletons and roof design
- Ray-intersection diagram
- Other collision detection applications
- Hierarchical clustering
- Traveling salesman heuristics
- Greedy matching
- Constructive induction
- Gröbner bases

David Eppstein, Information & Computer Science, UC Irvine, .



## $O(n)$ Median Selection Algorithms

- An algorithm called "quickselect", was developed by Tony Hoare who also invented the similarly-named quicksort in 1959-1960. A recursive algorithm can find any element (not just the median). This algorithm has an *average* performance of $O(n)$.
- Another algorithm called PICK was originally developed in 1973 by the mouthful of Blum, Floyd, Pratt, Rivest, and Tarjan. It has $O(n)$ performance in the worst case.

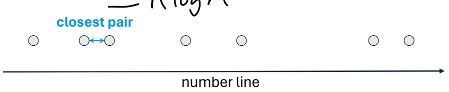
Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, Robert E. Tarjan. (1973). "Time bounds for selection" (PDF). Journal of Computer and System Sciences. **7** (4): 448–461



## Warm-up: Closest Pair of Points in 1-D

- **Problem:** Given $n$ real numbers $x_1, x_2, \dots, x_n$, find $i \neq j$ with the smallest $|x_i - x_j|$.
- **Solution:**
  - Sort the points. $(n \log n)$
  - Go over the list and compute the distances between the adjacent points. $(n)$
  - Return the pair of adjacent points with the min distance.
- **Runtime:**

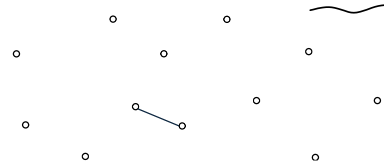
$= n \log n$






## Closest Pair in 2D

- Given a set of $n \geq 2$ points in the *plane*.
- **Goal:** Find ***minimum distance*** between any pair of points.
- A point $p = (x_p, y_p)$ is represented by a pair of numbers.
- (Pythagorean Theorem) $dist(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$.
- **How fast is the trivial algorithm for this problem?** $O(n^2)$





## Warm-up: Closest Pair in 1D

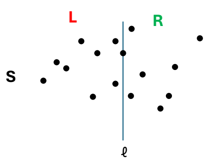
- You're given a set of $n \geq 2$ *distinct* points on a line.
- **Goal:** Find minimum distance between any pair of points
- **Q:** Can you think of a fast algorithm?
  - (1) Sort the points in increasing order as $(p_1, p_2, \dots, p_n)$ $\boxed{O(n \log n) \text{ time}}$
  - (2) Scan the list of sorted points; return $\min_{1 \leq i < n} \{p_{i+1} - p_i\}$. $\boxed{O(n) \text{ time}}$





## Finding the 2-D Closest Pair of Points With Divide and Conquer

Input: A list of n points in $\mathbb{R}^2$: $[(x_1,y_1), (x_2,y_2), \dots, (x_n,y_n)]$
Output: The closest pair of points
Goal: O(n log n) algorithm $\rightarrow$ $\boxed{T(n) = 2T(n/2)+O(n)}$
Working backwards: We want a "mergesort" recurrence relation!

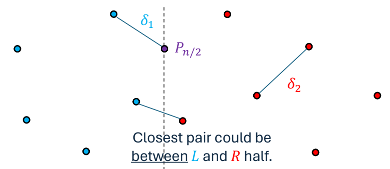



**Formally:** Consider the recurrence relation $T(n) = kT(n/b) + O(n^d)$, when $k, b > 1$. Then:

$$T(n) = \begin{cases} O(n^d) & \text{if } (k/b^d) < 1 \\ O(n^d \log n) & \text{if } (k/b^d) = 1 \\ O(n^{\log_b k}) & \text{if } (k/b^d) > 1 \end{cases}$$

## Divide and Conquer?

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n \leq 3$ then return min dist among $P_1, P_2, P_3$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // $P_{n/2}$ : median in x-coord
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ ✓ // min dist on left n/2 pts
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ ✓ // min dist on right n/2 pts
*...What comes next?* // *...how?*
*...Just return* $\min\{\delta_1, \delta_2\}$? ✗



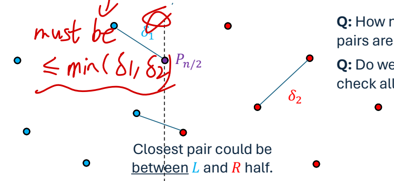

Closest pair could be between $L$ and $R$ half.



## Divide and Conquer?

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n \leq 3$ then return min dist among $P_1, P_2, P_3$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // $P_{n/2}$ : median in x-coord
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ // min dist on left n/2 pts
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ // min dist on right n/2 pts
$\boldsymbol{\delta_3} \leftarrow$ **min distance between** $L$ and $R$ half // *...how??*
return $\min\{\delta_1, \delta_2, \delta_3\}$

must be $\leq \min(\delta_1, \delta_2)$




**Q:** How many blue/red pairs are there?

**Q:** Do we need to check all of them?

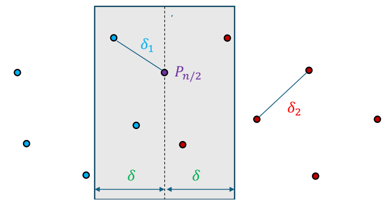
Closest pair could be between $L$ and $R$ half.



## **Key Insight:** The $\delta$-strip

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n \leq 3$ then return min dist among $P_1, P_2, P_3$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ // min dist on right
need to know min dist between L and R // ...look at $\delta$-strip

- Let $\delta = \min\{\delta_1, \delta_2\}$.
- **Observation:**
  - If a closest pair is between blue and red,
  - then their $x$-coord are within $\delta$ of $P_{n/2}$'s $x$-coord (the "$\delta$-strip").



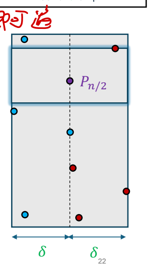


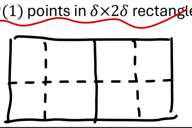

## Closest blue/red pair is $\delta \times 2\delta$ rectangle!

从下往上！只需计算每个点以它 y-coord 为底的 $\delta \times 2\delta$ square之间的点的距离即可

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n \leq 3$ then return min dist among $P_1, P_2, P_3$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ // min dist on right
need to know min dist between L and R // ...look at $\delta$-strip

- Let $\delta = \min\{\delta_1, \delta_2\}$.
- Consider any pair of points $(p, q)$ in $\delta$-strip.
- **Observe**: If diff in y-coord of $p$ and $q > \delta$, then $(p, q)$ is not closest
- **So:** if $(p, q)$ is closest, then they are in $\delta \times 2\delta$ rectangle!
- **Next**: there are $O(1)$ points in $\delta \times 2\delta$ rectangle. Why?





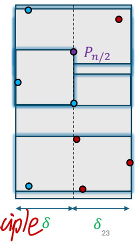
## Properties of the $\delta$-strip

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n \leq 3$ then return min dist among $P_1, P_2, P_3$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ // min dist on right
need to know min dist between L and R // ...look at $\delta$-strip

- Let $\delta = \min\{\delta_1, \delta_2\}$.
- **Q:** How many blue pts can there be in a $\delta \times \delta$ square?
- **A:** 4 = O(1)
- **Q:** How many pts can there be in a $\delta \times 2\delta$ rectangle?
- **A:** 8 = O(1)

$\leq 8$, by pigeon hole principle

**How to find a closest red/blue pair:**
Slide a $\delta \times 2\delta$ rectangle!






## Pause and Think

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n = 2$ then return dist$(P_1, P_2)$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ // min dist on right
Let $(P_1', P_2', \ldots, P_m')$ be points in the $\delta$-strip, // m≤n sorted by *y-coordinate* $O(n \log n)$
$\boldsymbol{\delta_3 \leftarrow ???}$ **// O(n) distances computed**
return $\min\{\delta_1, \delta_2, \delta_3\}$

$(\leq 8n) \Rightarrow O(n)$



## Analysis of ClosestPair

**ClosestPair**$(P_1, \ldots, P_n)$: // $n \geq 2$ pts in the plane, $x$-sorted asc.
if $n = 2$ then return dist$(P_1, P_2)$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median
$\delta_1 \leftarrow$ **ClosestPair**$(L)$ // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**$(R)$ // min dist on right
Let $(P_1', P_2', \ldots, P_m')$ be points in the $\delta$-strip, // m≤n sorted by *y-coordinate* $O(n \log n)$
$\boldsymbol{\delta_3} \leftarrow \min\limits_{1 \leq i < m, 1 \leq c \leq 7} \{dist(P_i', P_{i+c}')\}$ // ≤7m distances computed $+ O(n)$
return $\min\{\delta_1, \delta_2, \delta_3\}$

$= O(n \log n)$

- **Runtime:** let $T(n)$ be the runtime of **ClosestPair** on $n$ points.
- $T(n) = 2T(n/2) + O(n \log n) = \boldsymbol{O(n \log^2 n)}$
- How can we improve this to $T(n) = 2T(n/2) + O(n)$ ?



## Sort all points from the beginning

| Sorted by $x$ | Sorted by $y$ |
| --- | --- |

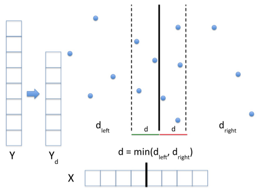
**ClosestPair**$(P_1, \ldots, P_n, P_1', \ldots, P_n')$: // $n \geq 3$ pts in the plane
if $n \leq 3$ then return min dist among $P_1, P_2, P_3$ // base case
$(L, R) \leftarrow$ partition points by $P_{n/2}$ // split by median x-coordinate
$\delta_1 \leftarrow$ **ClosestPair**($L$ (sorted by x), $L$ (sorted by y)) // min dist on left
$\delta_2 \leftarrow$ **ClosestPair**($R$ (sorted by x), $R$ (sorted by y)) // min dist on right
$\delta_3 \leftarrow$ min distance in $\delta$-strip // details in notes
return $\min\{\delta_1, \delta_2, \delta_3\}$

- **Runtime:**
  - Sort points in x and y: $O(n \log n)$.
  - Recursive algo: $T(n) = 2T(n/2) + O(n) = O(n \log n)$.
- **Total time:** $O(n \log n)$

# Total running time

- Sort once by x-coordinate: O(n log n)
- Sort once by y-coordinate: O(n log n)
- Recursive algorithm: T(n) = 2T(n/2) + O(n)

T(2) = O(1)

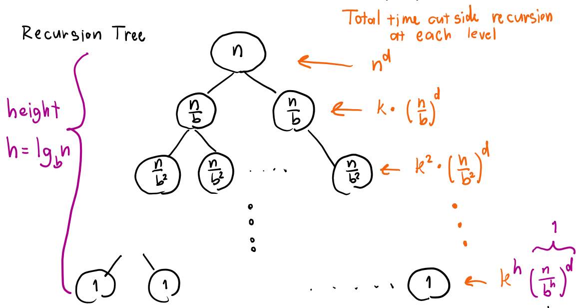



Discovered in 1976 by
Jon Louis Bentley and Michael Ian Shamos



## Proof of Master Theorem

$$T(n) = kT(n/b) + O(n^d)$$




So total time over all levels $= n^d\left(1 + \frac{k}{b^d} + \left(\frac{k}{b^d}\right)^2 + \dots \left(\frac{k}{b^d}\right)^h\right)$

## Proof of Master Theorem

$$T(n) = kT(n/b) + O(n^d)$$

- $T(n) = n^d\left(1 + \frac{k}{b^d} + \left(\frac{k}{b^d}\right)^2 + \cdots + \left(\frac{k}{b^d}\right)^h\right)$
  - where height $h = \log_b n$
- By properties of geometric series, $T(n) = \begin{cases} O(n^d) & \text{if } k < b^d \\ O(n^d \log n) & \text{if } k = b^d \\ O(n^{\log_b k}) & \text{if } k > b^d \end{cases}$



# Conclude: Divide-and-Conquer Algorithms

**Main Idea:**

1. **Divide** the problem into smaller sub-problems (creative step)
2. **Conquer** (solve) each sub-problem *recursively* (easy step)
3. **Combine** the solutions (creative step)

**Examples:** Merge-sort, Closest pair in 2d, Karatsuba's algorithm

**Note:**
- Not every recurrence is captured by the Master Theorem;
- good enough for us in this class (can derive more general versions)

**Next:** Dynamic Programming
(very efficient recursive algorithms when the number subproblems is small)

## **Bonus**: Ultimate way to solve recursions

- Guess and check.
- How formally?
  1. Guess ☺ (maybe by drawing the recursion tree)
  2. Check via induction

**Examples:**
- $T(n) = n - 1 + \frac{2}{n-1}\sum_{r=n/2}^{n} T(r-1)$
- $T(n) = T(n^{1/2}) + 1$
- $T(n) = T(0.2n) + T(0.7n) + n$



# Example How to Use Induction

- **Analysis:** For some c, $T(n) \le cn$ for all $n \ge 1$.
- **Proof:** By induction on n.

**Inductive Step:**
Assume $T(n') \le cn'$ for all $n' < n$.

$$T(n) = n - 1 + \frac{2}{n-1}\sum_{r=n/2}^{n} T(r-1)$$
$$\le n - 1 + \frac{2}{n-1}\sum_{r=n/2}^{n} c(r-1)$$
$$\le n - 1 + \frac{3c}{4}n + 2$$
$$\le cn \quad \text{(by choice of } c \ge 8)$$