CS 341: Algorithms

Lecture 4: Divide and conquer, continued

Slides due to Éric Schost and based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2026

Beyond the master theorem

Beyond the master theorem

Some recursive algorithms

- do recursion in uneven sizes (e.g. both $n/3$ and $n/4$), ...
- or in size $\sqrt{n}$, or even $\log(n)$, ...
- or have extra work more complicated than n^y

No master theorem covers everything. But **recursion trees** still work:

- find out how many nodes, of what size, there are at each level
- gives the total work per level
- **warning**, in case of uneven size recursion, some levels may not be complete
  $\rightarrow$ not always clear that you get a Θ (example: linear time median, later)

Remark:

- to be 100% clean, we should justify that we can work with sloppy recurrences.
- don't worry about it; it can be done (but gets more difficult than for merge sort)

Example: "generalized" master theorem

Suppose that $\boldsymbol{a \geq 1}$ and $\boldsymbol{b > 1}$. Consider the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + \boldsymbol{f(n)}, \qquad T(n) = d \quad (n \leq 1), \qquad \inf_n f(n) > 0$$

Let $\boldsymbol{x = \log_b a}$ (so $a = b^x$).

$$\text{Then } T(n) \in \begin{cases} \Theta(\boldsymbol{f(n)}) & \text{if } f(n) \in \Omega(n^{x+\varepsilon}), \text{ for some } \varepsilon > 0 \\ \Theta(\boldsymbol{n^x \log n}) & \text{if } f(n) \in \Theta(n^x) \\ \Theta(\boldsymbol{n^x}) & \text{if } f(n) \in O(n^{x-\varepsilon}), \text{ for some } \varepsilon > 0 \end{cases}$$

Proof: by recursion tree, no details.

Remark: we can either say "for n a power of b" or extend the definition of T to all $n \in \mathbb{R}$

Examples and non-examples

$T(n) = 3T(n/4) + n\log(n)$.

- $x = \log_4(3) < 1$
- $f(n) \in \Omega(n^1) = \Omega(n^{x+\varepsilon})$ where $\varepsilon = 1 - x > 0$
- so $T(n) \in \Theta(n\log(n))$

$T(n) = 2T(n/2) + n\log(n)$.

- $x = \log_2(2) = 1$
- $f(n) \in \Omega(n)$ implies not case three
- $f(n) \notin \Theta(n)$ implies not case two
- $f(n)/n^{x+\varepsilon} = \log(n)/n^{\varepsilon} \to 0$ implies $f(n) \in o(n^{x+\varepsilon})$ for all $\varepsilon > 0$
- therefore $f(n) \notin \Omega(n^{x+\varepsilon})$ and not case one
- No case of the master theorem applies

Linear time median

Median of medians

Selection: given $A[0..n-1]$ and k in $\{0, \ldots, n-1\}$, find the entry that would be at index k **if A was sorted**

Minimum: select from $A[0..n-1]$ at index $k = 0$
Maximum: select from $A[0..n-1]$ at index $k = n-1$

Median: select from $A[0..n-1]$ at index $\lfloor n/2 \rfloor$

Model: unit cost

Known results: sorting A in $\Theta(n \log(n))$, or a simple randomized algorithm in expected time $\Theta(n)$

The selection algorithm

quick-select(A, k)
A: array of size n, k: integer s.t. $0 \leq k < n$
1. $p \leftarrow$ **choose-pivot**(A)
2. $i, j \leftarrow$ **partition**(A, p)
3. **if** $i \leq k \leq j$ **then**
4. **return** p
5. **else if** $k < i$ **then**
6. **return quick-select**$(A[0, 1, \ldots, i-1], k)$
7. **else if** $j < k$ **then**
8. **return quick-select**$(A[j+1, j+2, \ldots, n-1], k-j-1)$

partition(A, p)**:**

- reorders A so that $[< p,\ \ A[i] = p, \ldots, A[j] = p,\ \ > p]$ **in linear time**

Goal: find a pivot such that both i and $n - j - 1$ are not too large

Median of medians

Sketch of the algorithm:

- divide A into $n/5$ groups $G_1, \ldots, G_{n/5}$ of size 5
- find the medians $m_1, \ldots, m_{n/5}$ of each group $\Theta(n)$
- pivot p is the median of $[m_1, \ldots, m_{n/5}]$ $T(n/5)$

Claim

With this choice of p, the indices i and $n - j - 1$ are **at most** $7n/10$

Median of medians

Sketch of the algorithm:

- divide A into $n/5$ groups $G_1, \ldots, G_{n/5}$ of size 5
- find the medians $m_1, \ldots, m_{n/5}$ of each group $\Theta(n)$
- pivot p is the median of $[m_1, \ldots, m_{n/5}]$ $T(n/5)$

Claim

With this choice of p, the indices i and $n - j - 1$ are **at most** $7n/10$

Proof

- **half** of the m_i's are **greater than or equal to** p $n/10$
- for each m_i, there are **3** elements in G_i **greater than or equal to** m_i
- so **at least** $3n/10$ elements **greater than or equal to** p
- so **at most** $7n/10$ elements **less than** p
- so i is **at most** $7n/10$. Same thing for $n - j - 1$

Median of medians

Sketch of the algorithm:

- divide A into $n/5$ groups $G_1, \ldots, G_{n/5}$ of size 5
- find the medians $m_1, \ldots, m_{n/5}$ of each group — $\Theta(n)$
- pivot p is the median of $[m_1, \ldots, m_{n/5}]$ — $T(n/5)$

Claim

With this choice of p, the indices i and $n - j - 1$ are **at most** $7n/10$

Consequence: (sloppy) recurrence

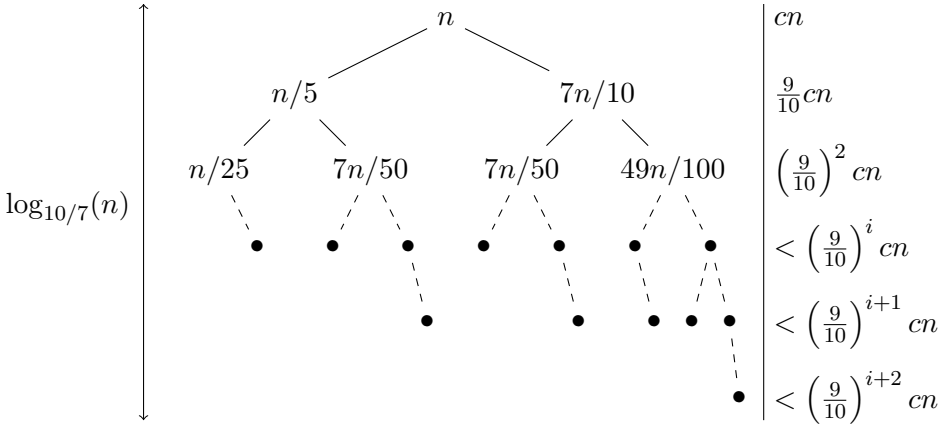
$$T(n) = T(n/5) + T(7n/10) + cn$$

Claim

This gives worst-case $T(n) \in \Theta(n)$ ($\Omega(n)$ clear)

The recursion tree for $T(n)$

Can prove: we can set $T(n) = 0$ for $n \leq 1$.



Key point: $\boldsymbol{T(n) < cn}$ $\times$ (geometric sum of ratio $\mathbf{9/10 < 1}$), so $\boldsymbol{T(n) \in O(n)}$.

Why not median of three?

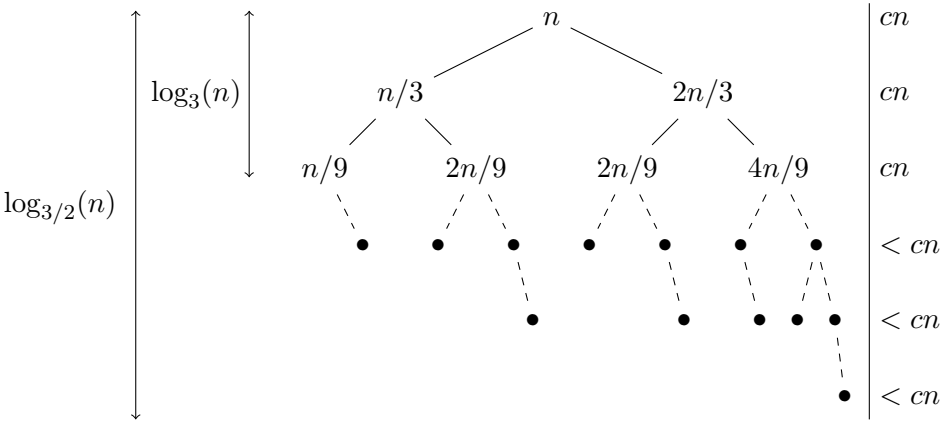
- we do $n/3$ **groups of 3** and find their medians $m_1, \ldots, m_{n/3}$ — $\Theta(n)$
- p is the median of $[m_1, \ldots, m_{n/3}]$ — $T(n/3)$
- half of the m_i's are greater than or equal to p — $n/6$
- in each group, **2** elements greater than or equal to m_i
- so overall at least $n/3$ elements greater than or equal to p
- so at most $2n/3$ elements less than p
- so $i \leq 2n/3$, and $n - j - 1 \leq 2n/3$

Recurrence: $T(n) = T(n/3) + T(2n/3) + cn$

Exercise

This gives $T(n) \in \Theta(n \log(n))$

The recursion tree for $T(n)$



Key point: $\boldsymbol{cn \log_3 \leq T(n) \leq cn \log_{3/2}(n)}$ so $\boldsymbol{T(n) \in \Theta(n \log n)}$.

Closest pairs
(time permitting)

Closest pairs

Goal: given n points (x_i, y_i) in the plane, find a pair (i, j) that minimizes the distance

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Equivalent to minimize

$$d_{i,j}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

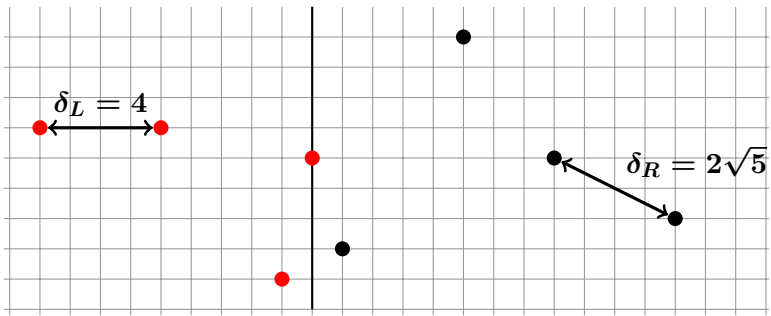
Assumption

all x_i's distinct

Divide-and-conquer

Idea: separate the points into two halves L, R at the median x-value

- L = all $n/2$ points with $x \leq x_{\text{median}}$ (no other point on the median line)
- R = all $n/2$ points with $x > x_{\text{median}}$
- find the closest pairs in both L and R recursively
- the closest pair is either **between points in L** (done), or **between points in R** (done), or **transverse** (one in L, one in R)



Divide-and-conquer

Familiar pattern ...

ClosestPair(P)
P: array of points of size n
1. $L, R \leftarrow$ **PartitionByMedianX**(P)
2. $\delta_L \leftarrow$ **ClosestPair**(L)
3. $\delta_R \leftarrow$ **ClosestPair**(R)
4. $\delta_T \leftarrow$ **TransverseClosestPair**(L, R) ???
5. **return** $\min(\delta_L, \delta_R, \delta_T)$

Divide-and-conquer

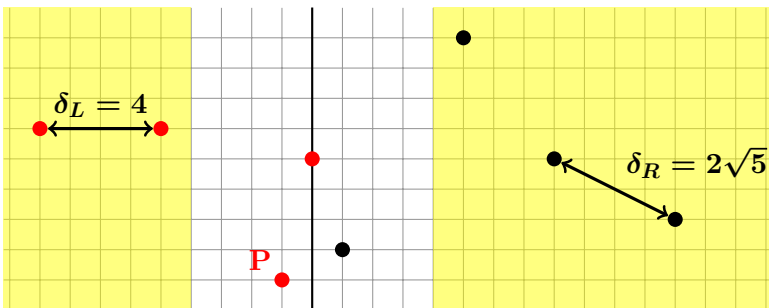
Idea: We only have to find transverse pairs closer than δ

ClosestPair(P)
P: array of points of size n
1. $L, R \leftarrow$ **PartitionByMedianX**(P)
2. $\delta_L \leftarrow$ **ClosestPair**(L)
3. $\delta_R \leftarrow$ **ClosestPair**(R)
4. $\delta \leftarrow \min(\delta_L, \delta_R)$
5. **return** **TransverseCase**(L, R, δ) ???

Finding the shortest transverse distance

Set $\delta = \min(\delta_L, \delta_R)$

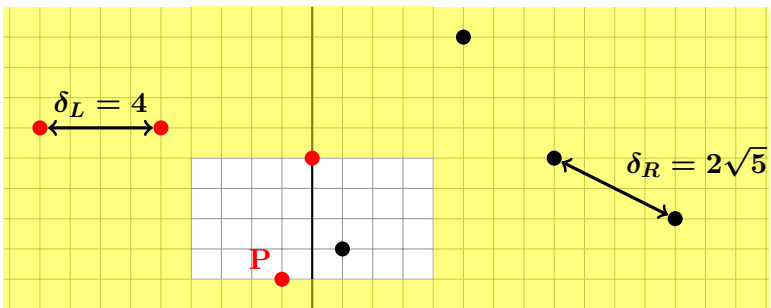
- only need to consider transverse pairs (P, Q) with $\mathbf{dist}(\boldsymbol{P}, \boldsymbol{R}) \leq \boldsymbol{\delta}$ and $\mathbf{dist}(\boldsymbol{Q}, \boldsymbol{L}) \leq \boldsymbol{\delta}$.



Finding the shortest transverse distance

Set $\delta = \min(\delta_L, \delta_R)$

- for any $P = (x_P, y_P)$, enough to look at points with $y_P \leq y \leq y_P + \delta$

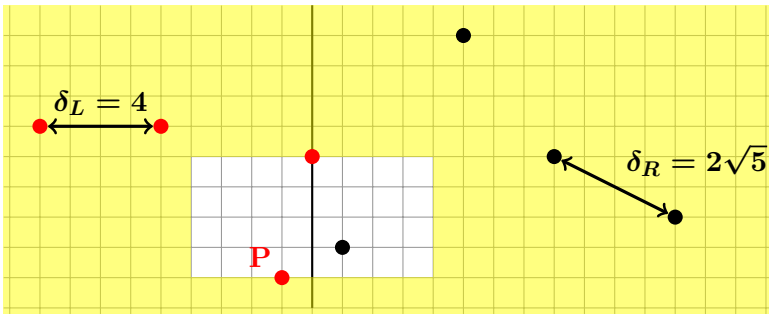


Finding the shortest transverse distance

Set $\delta = \min(\delta_L, \delta_R)$

- for any $P = (x_P, y_P)$, enough to look at points with $y_P \leq y \leq y_P + \delta$

So it is enough to check distances $d(P, Q)$ for Q in the rectangle.



How many points in the rectangle?

Claim

There are at most **8** points from our initial set (including P) in the rectangle.

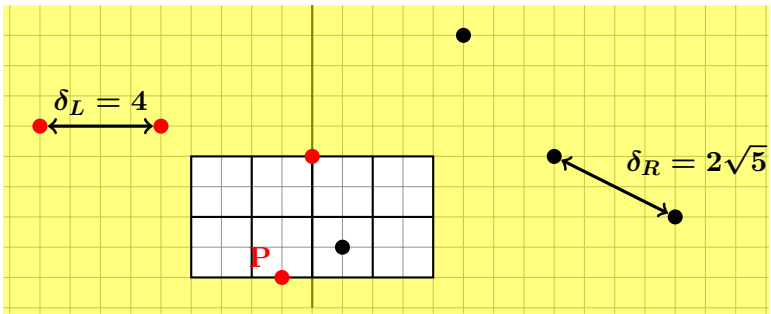
How many points in the rectangle?

Claim

There are at most **8** points from our initial set (including P) in the rectangle.

Proof. Cover the rectangle with **8** squares of side length $\delta/2$

- they overlap along lines, but it's OK



How many points in the rectangle?

Claim

There are at most **8** points from our initial set (including P) in the rectangle.

Proof. Cover the rectangle with **8** squares of side length $\delta/2$

- they overlap along lines, but it's OK
- a square on the left contains **at most one point from L**
- a square on the right contains **at most one point from R**

Consequence: at most 8 points in the range $y_P \leq y \leq y_P + \delta$

Data structures and runtime

Initialization: sort the points **twice**, with respect to x and to y.
One-time cost $\boldsymbol{O(n \log(n))}$, before recursive calls cf kd-trees

Main algo: given two arrays representing the **same points** (A_x sorted in x, A_y sorted in y)

- find the x-median using A_x $\boldsymbol{\Theta(1)}$
- split both A_x and A_y for recursions $\boldsymbol{\Theta(n)}$
- recurse on L and R $\boldsymbol{2T(n/2)}$
- remove the points at distance $\geq \delta$ from the x-median line from A_y $\boldsymbol{\Theta(n)}$
- inspect all remaining points P in A_y (=in increasing y-order)
  for each P, compute the distance to points Q in A_y with $y_P \leq y_Q \leq y_P + \delta$ min.
  at most 8 points Q need to be checked per P $\boldsymbol{\Theta(n)}$

Runtime: $\boldsymbol{T(n) = 2T(n/2) + cn}$ so $\boldsymbol{T(n) \in \Theta(n \log(n))}$