Lecture 8

Divide and conquer

Chinmay Nirkhe | CSE 421 Winter 2026

Midterm (logistics)

- Mon Feb 2nd 5:30 - 7:20 pm in BAG 131
- Lecture 11 (next Friday) will be 50% a review session
- Contents covered: Everything through Divide and Conquer algorithms
  - Since we won't any HW problems on D&C before the midterm, exam questions will only be conceptual on D&C
- Exam consists of multiple choice questions and long form
  - Long form are similar to HW long forms but tailored for less writing
  - So read instructions carefully and only answer what is asked of you
- Practice midterm will be released sometime this weekend
- Poll: Cheat sheet vs. repeat problem

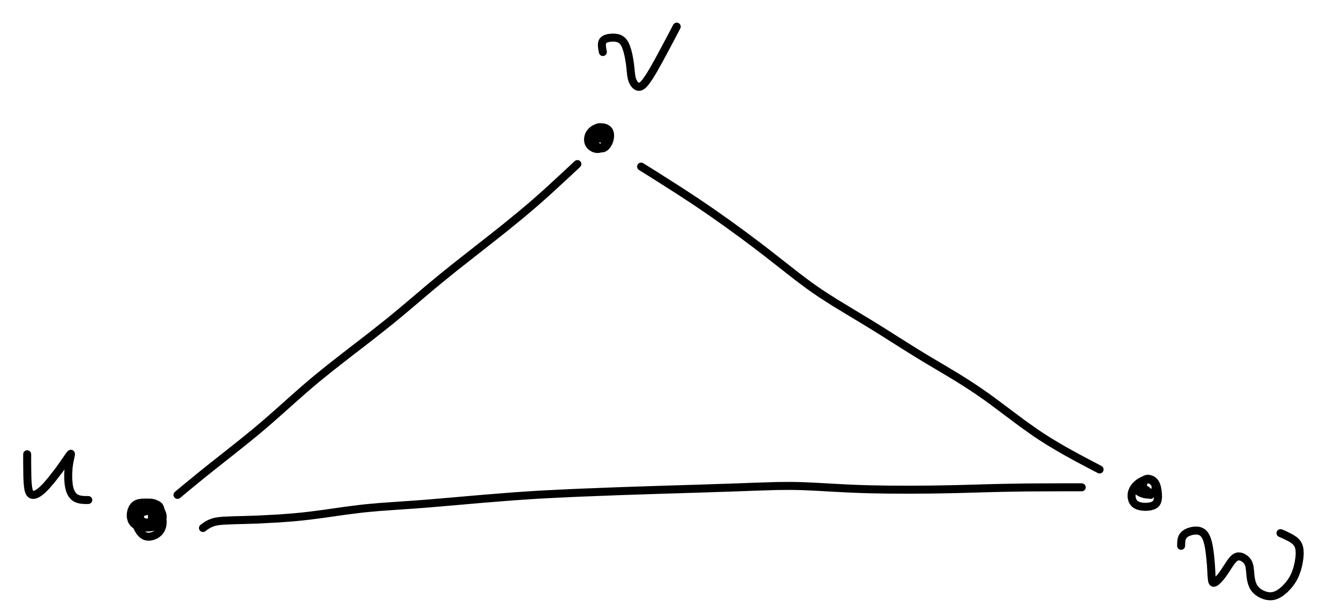
MST applications

Applications of MST

- Network design — minimal connectivity for telephone, electrical, cable, internet networks
- Approximation algorithms for computational problems - traveling problem, Steiner trees
- Indirect applications
  - Max bottleneck paths
  - LDPC error correcting codes
  - Image restoration under Renyí entropy
  - Reducing data storage in sequencing amino acids
  - Modeling local particle interaction in turbulence flows
  - Autoconfig protocol for Ethernet bridging to avoid network cycles

k-clustering of data points

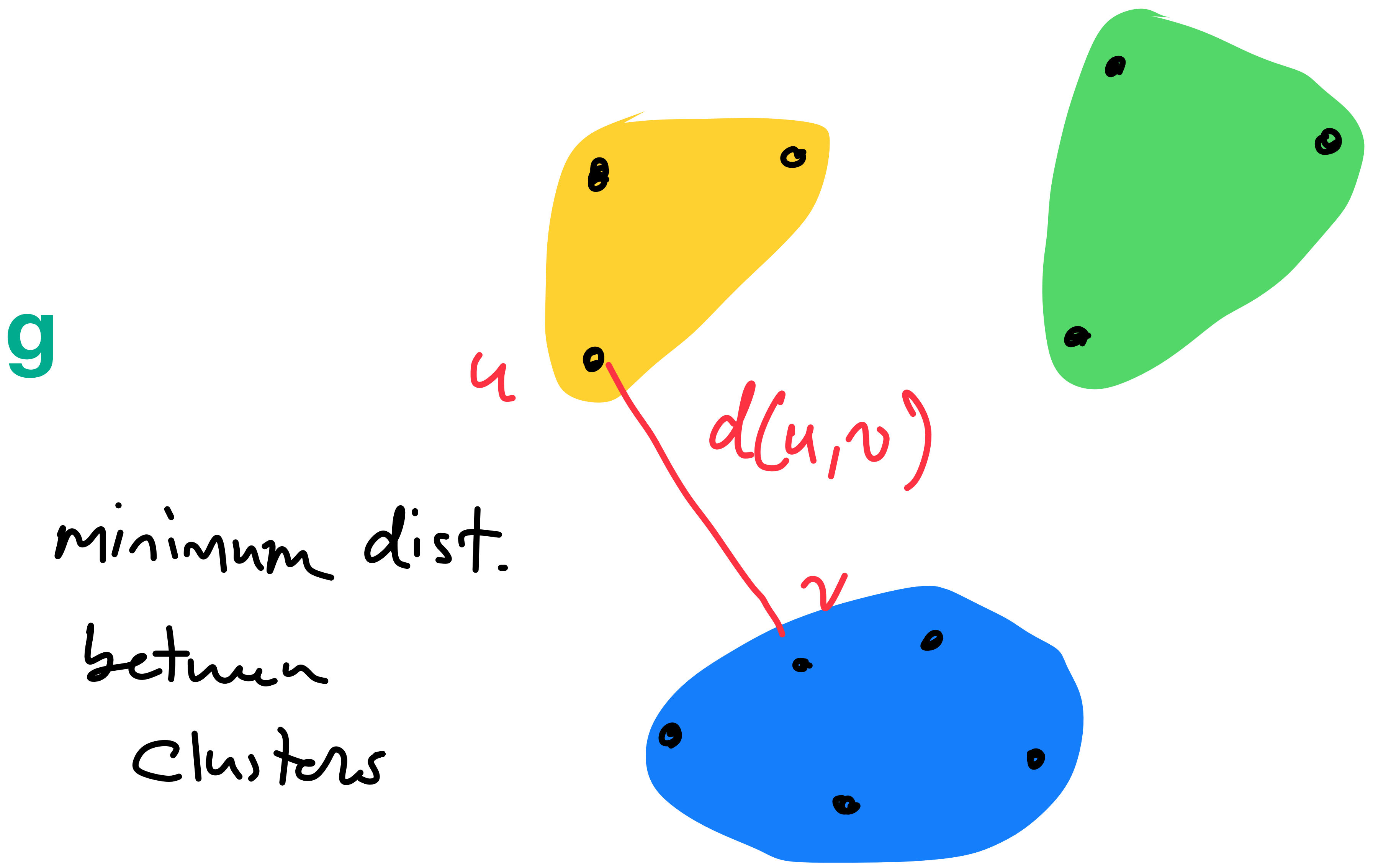
Maximum distance clustering



- **Input:** A set U of n elements, a **metric** $d : U^2 \to \mathbb{R}_{\geq 0}$, and $k \in \mathbb{N}$
  - Metric satisfies $d(u, u) = 0$, $d(u, v) = d(v, u)$
  - and triangle inequality $d(u, v) + d(v, w) \geq d(u, w)$
- **Output:** A clustering function $a : U \to [k]$ **maximizing**

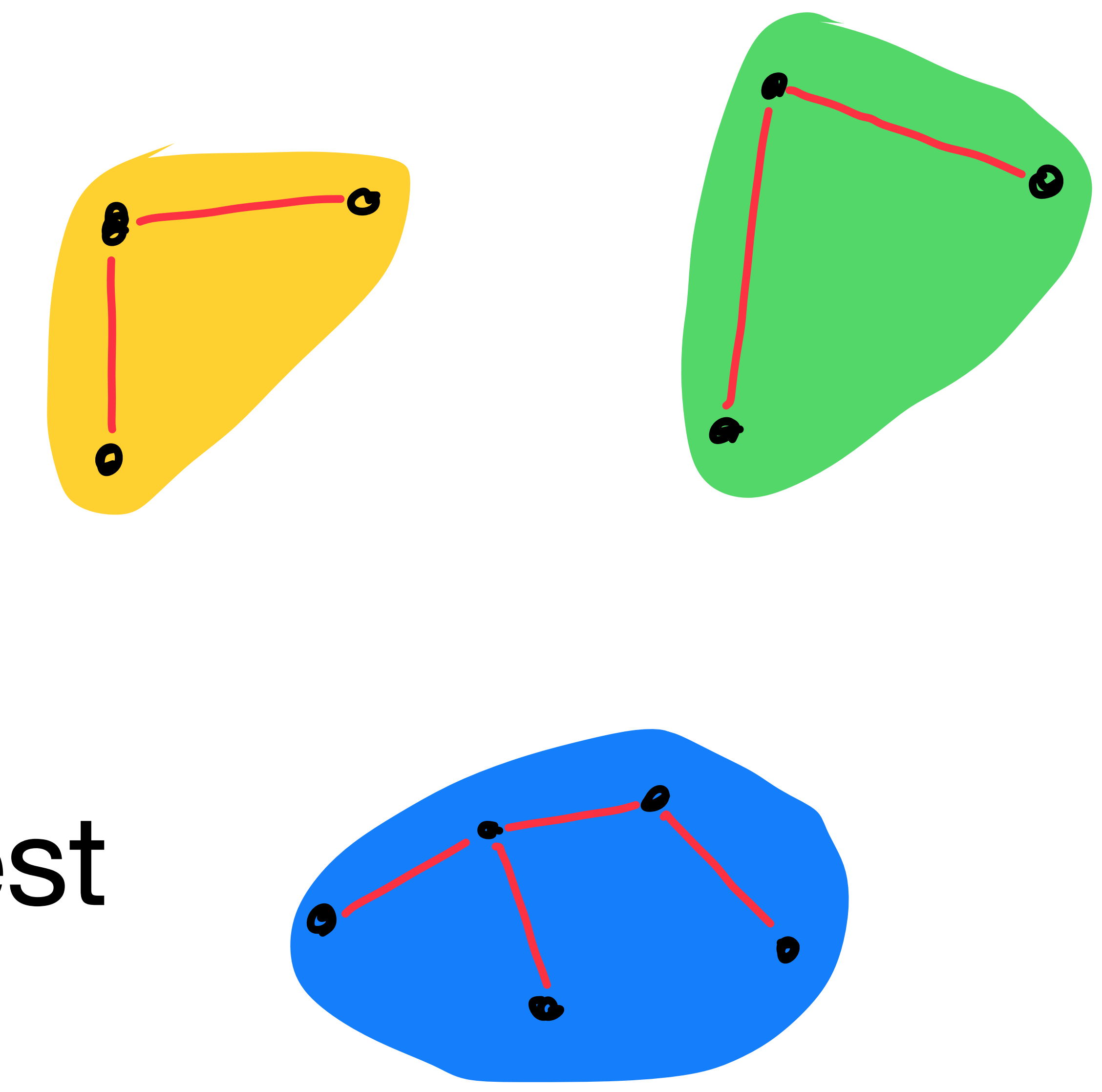
$$\min_{u,v \in U:\ a(u) \neq a(v)} d(u, v),$$

the minimum distance between the clusters



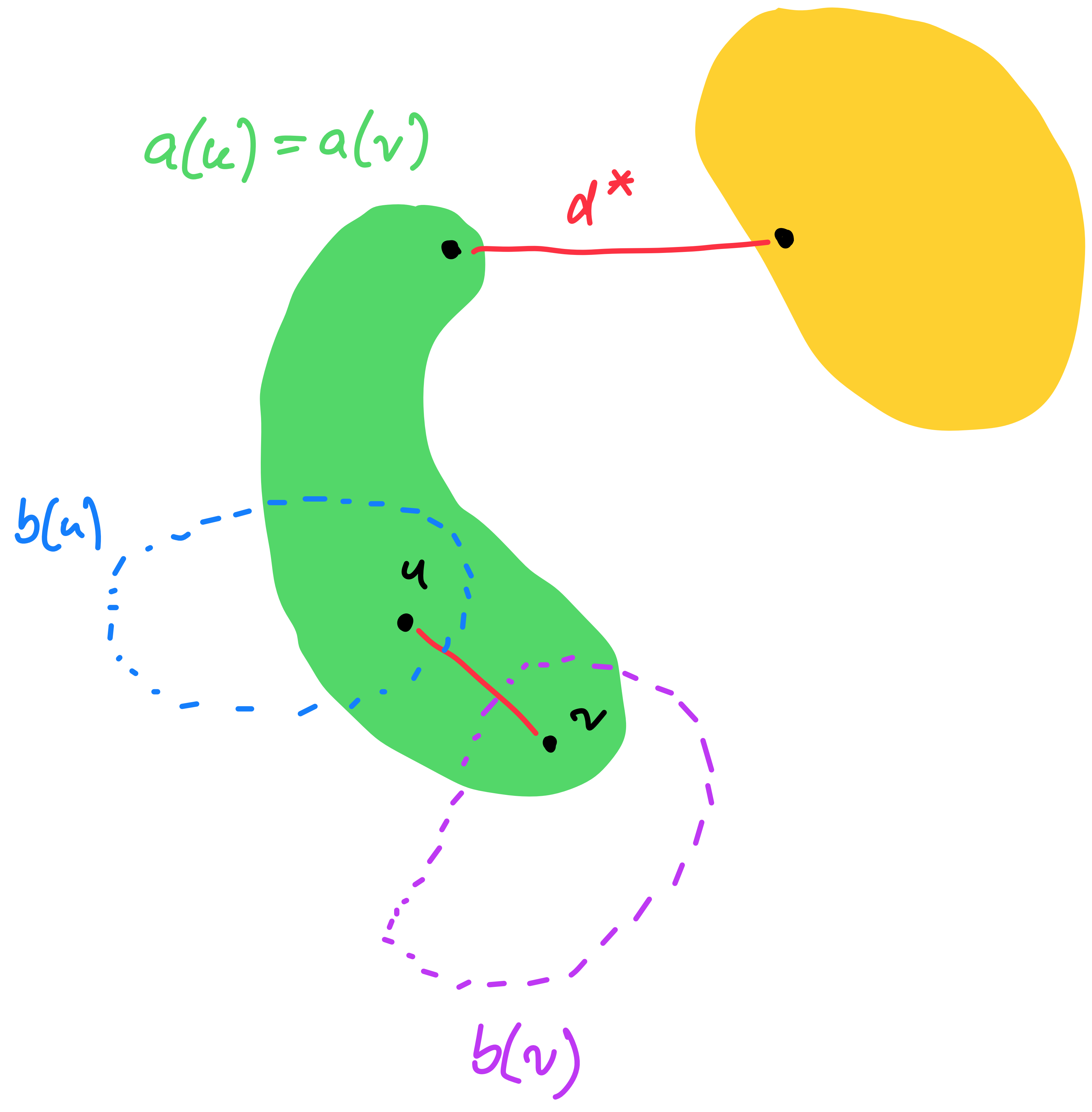
Kruskal's based algorithm

- Let $V = U$ and $E = V^2$ (all-to-all) with weight $w(e) = d(e)$.
- Run Kruskal's until $n - k$ edges are added.
    - Ensures that there are k trees in the forest.
    - Assign a cluster for every tree.
- Alternatively, run any MST algorithm and delete the heaviest $k - 1$ edges from the output tree.



Maximum distance clustering optimality

- Let d^* be the dist. between clustering a generated by Kruskal's
- By our alg. design, $d^* \geq d(u, v)$ for u, v in the same cluster: $a(u) = a(v)$.
- Consider a *different* clustering $b : U \to [k]$
  - There exist two points such that $a(u) = a(v)$ but $b(u) \neq b(v)$.
  - Then the **max** spacing between clusters of b is at most $d(u, v) \leq d^*$.
  - So the **max** spacing of b is $\leq$ the **max** spacing of a. So a is optimal.



Divide and conquer

Principles of divide and conquer

- Identity a division of the problem into a self-similar parts of size n/b
- Recursively solve each subpart of the problem
- Stitch the solutions from each subpart together

- Runtime is defined by the following recursively defined formula:

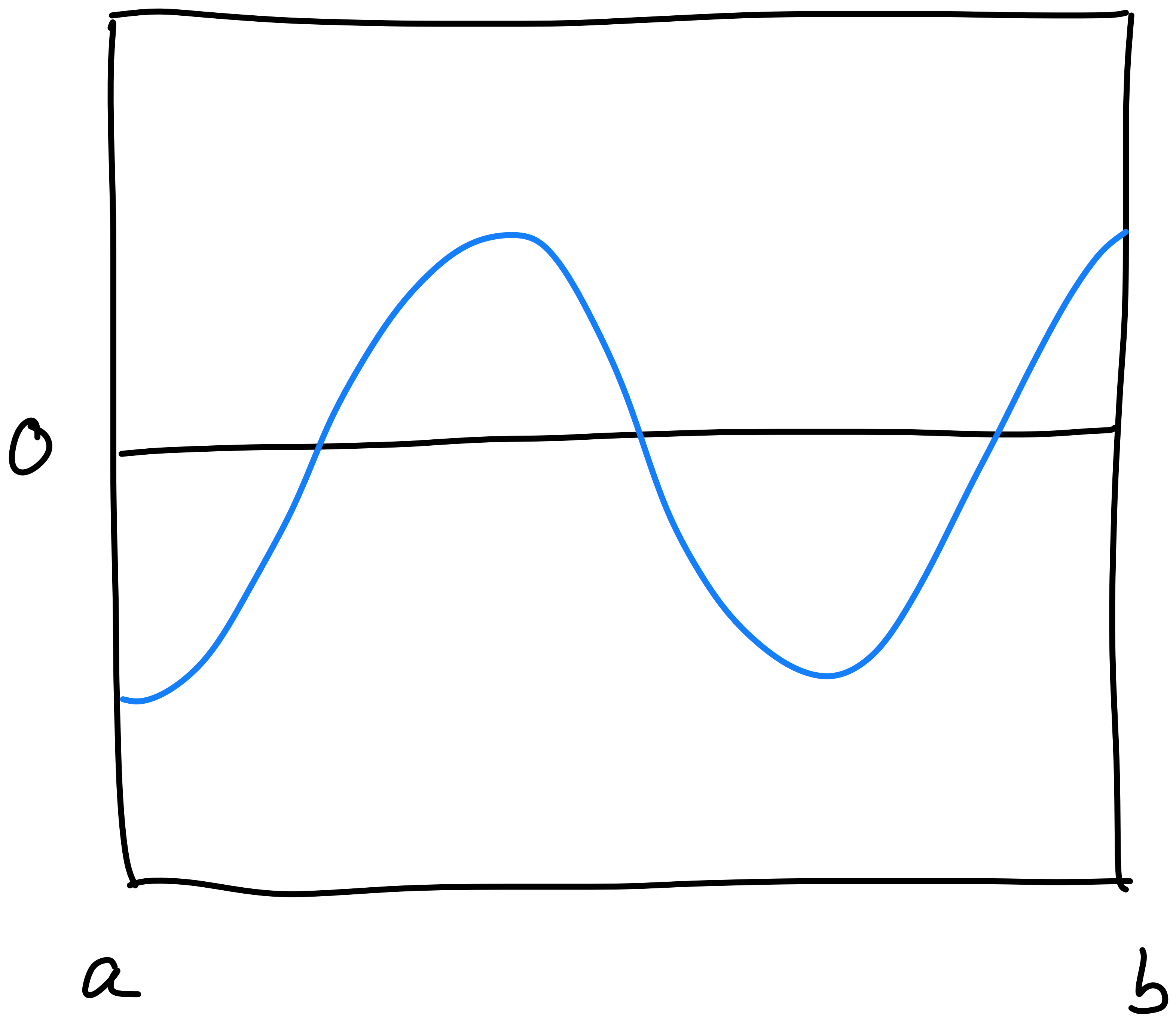
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \text{ and } T(< b) = O(1)$$

Examples of divide and conquer

- Mergesort, Quicksort
- Binary search
- Euclidean closest pair
- Rank selection, Median finding

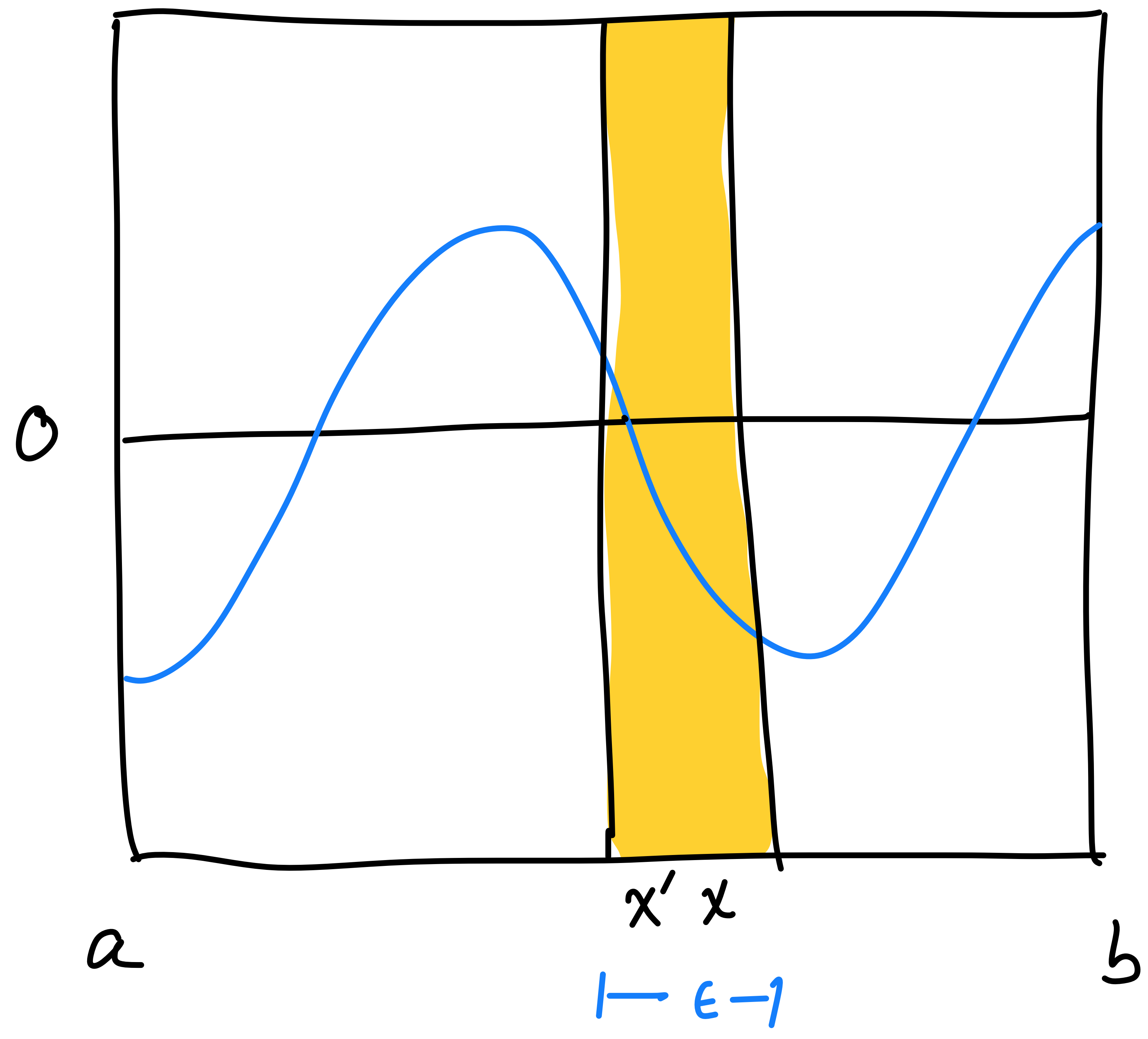
Binary search for roots of a function

- **Input**: Description of
  - a continuous $f : \mathbb{R} \to \mathbb{R}$,
  - $a < b \in \mathbb{R}$ such that $f(a) \leq 0 < f(b)$
  - and $\epsilon > 0$



Binary search for roots of a function

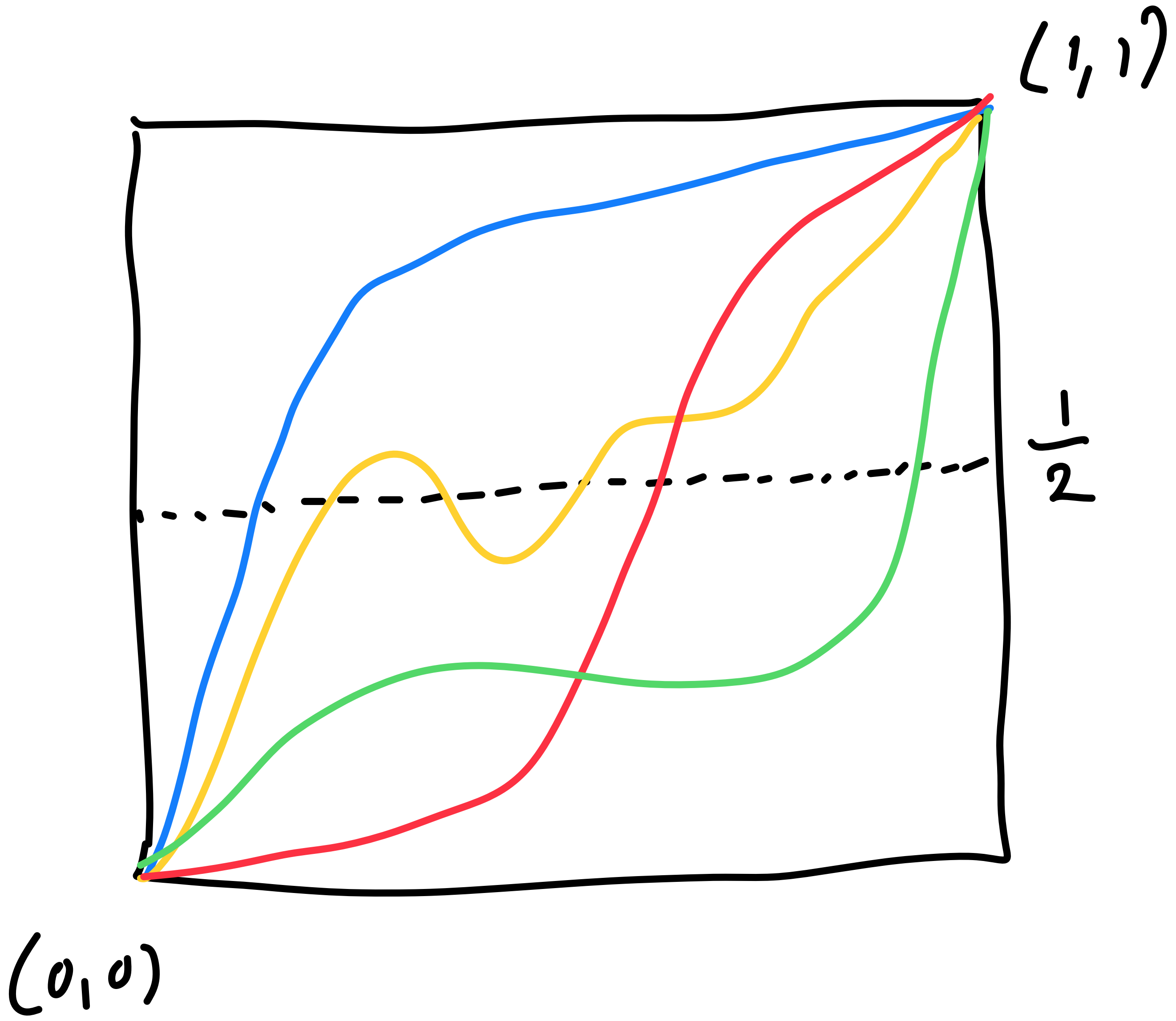
- **Input**: Description of
  - a continuous $f : \mathbb{R} \to \mathbb{R}$,
  - $a < b \in \mathbb{R}$ such that $f(a) \leq 0 < f(b)$
  - and $\epsilon > 0$
- **Output**: A value $x \in [a, b]$ such that $f(x') = 0$ for some $|x' - x| \leq \epsilon$.



Bisection method

- **Intermediate value theorem (IVT)**: If $f(0) = 0$, $f(1) = 1$ and f is continuous, there exists an $x \in (0,1)$ such that $f(x) = 1/2$.

- **Proof by picture**:

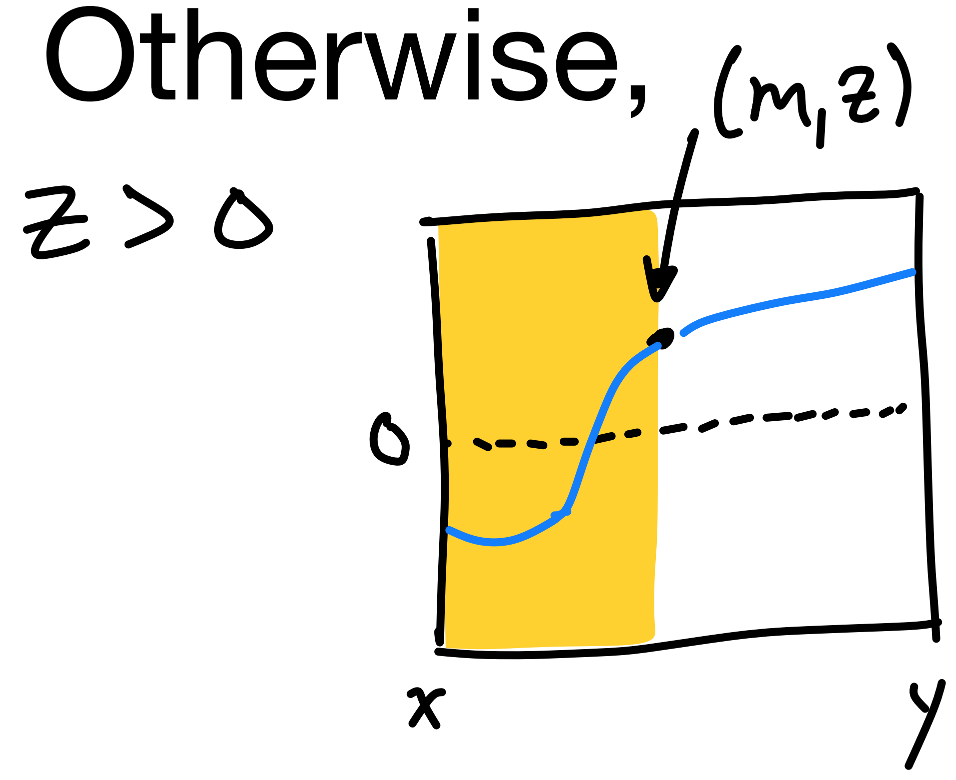


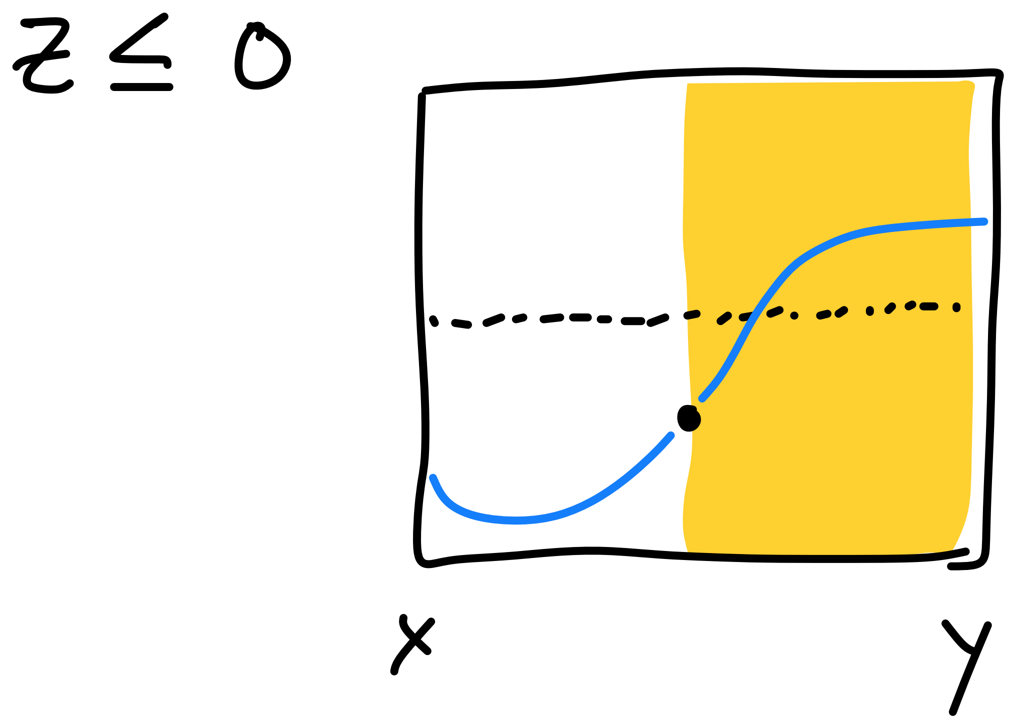
Any function must cross the midline at some point by continuity.

Bisection method

- **Algorithm** $g(x, y)$**:**
  - Let $m \leftarrow (x + y)/2$.
  - If $y - x \leq 2\epsilon$, return m.
  - Let $z \leftarrow f(m)$.
  - If $z > 0$, return $g(x, m)$.
  - Else, return $g(m, y)$.

- **Claim:** If $f(x) \leq 0 < f(y)$ for $x < y$, then $g(x, y)$ outputs an m such that $f(m') = 0$ for $|m' - m| \leq \epsilon$.
- **Proof:**
  - Base case, follows from IVT.
  - Otherwise,





Runtime analysis

Binary search problem

- Therefore, running $g(a, b)$ will solve the bisection problem.
- Each iteration of g is on an interval of half the length
  - starting from $b - a$ until the length is $\leq 2\epsilon$
  - Therefore, $\log_2 \left(\dfrac{b-a}{2\epsilon} \right)$ recursions.
- Each recursion costs $O(1)$ arithmetic operations plus 1 query to f.
- Runtime: $O(\log(b-a) + \log(1/\epsilon))$ queries to f.

Runtime analysis

- Simple version of generalized runtime analysis.
- Let $k = (b - a)/(2\epsilon)$.
- Then, $T(k) = T(k/2) + 1$ and $T(1) = 0$ for number of queries.
- Solves to $T(k) = \lceil \log_2 k \rceil + 1$.

Another classic divide and conquer problem

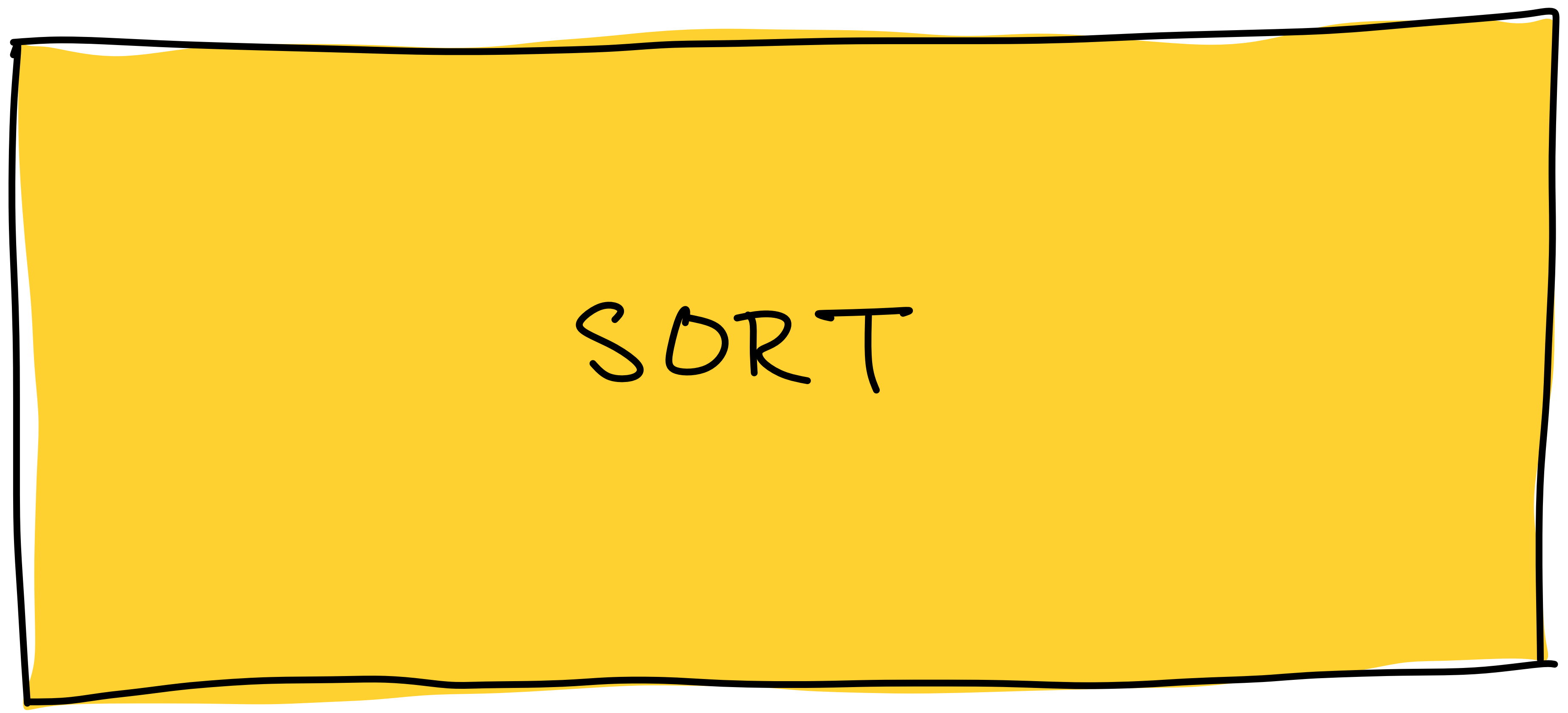
Mergesort

- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.

- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays

- Recursive time equation:
$T(n) \leq 2T(n/2) + O(n)$ with $T(1) = 0$.

- Solution: $T(n) \leq O(n \log n)$



Another classic divide and conquer problem

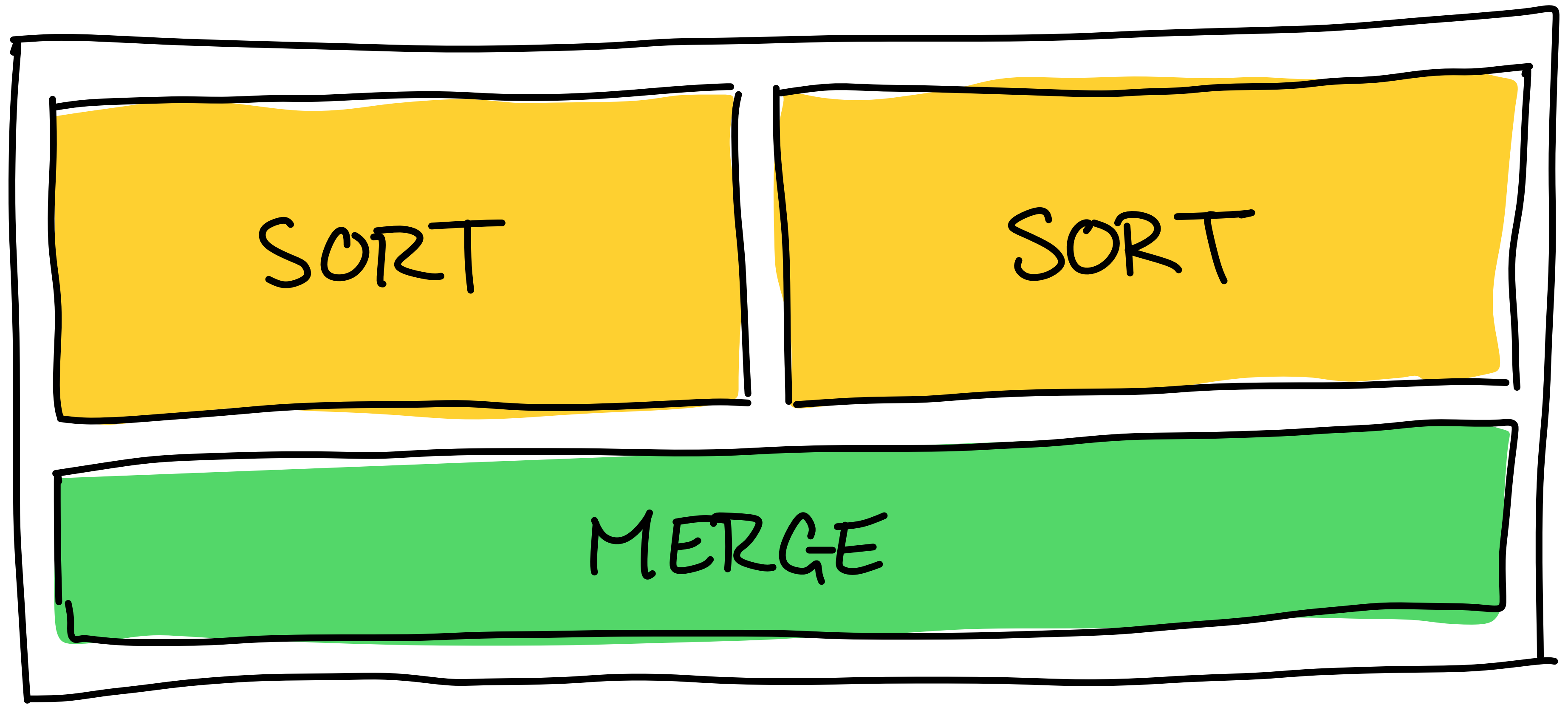
Mergesort

- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.

- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays

- Recursive time equation:
$T(n) \leq 2T(n/2) + O(n)$ with $T(1) = 0$.

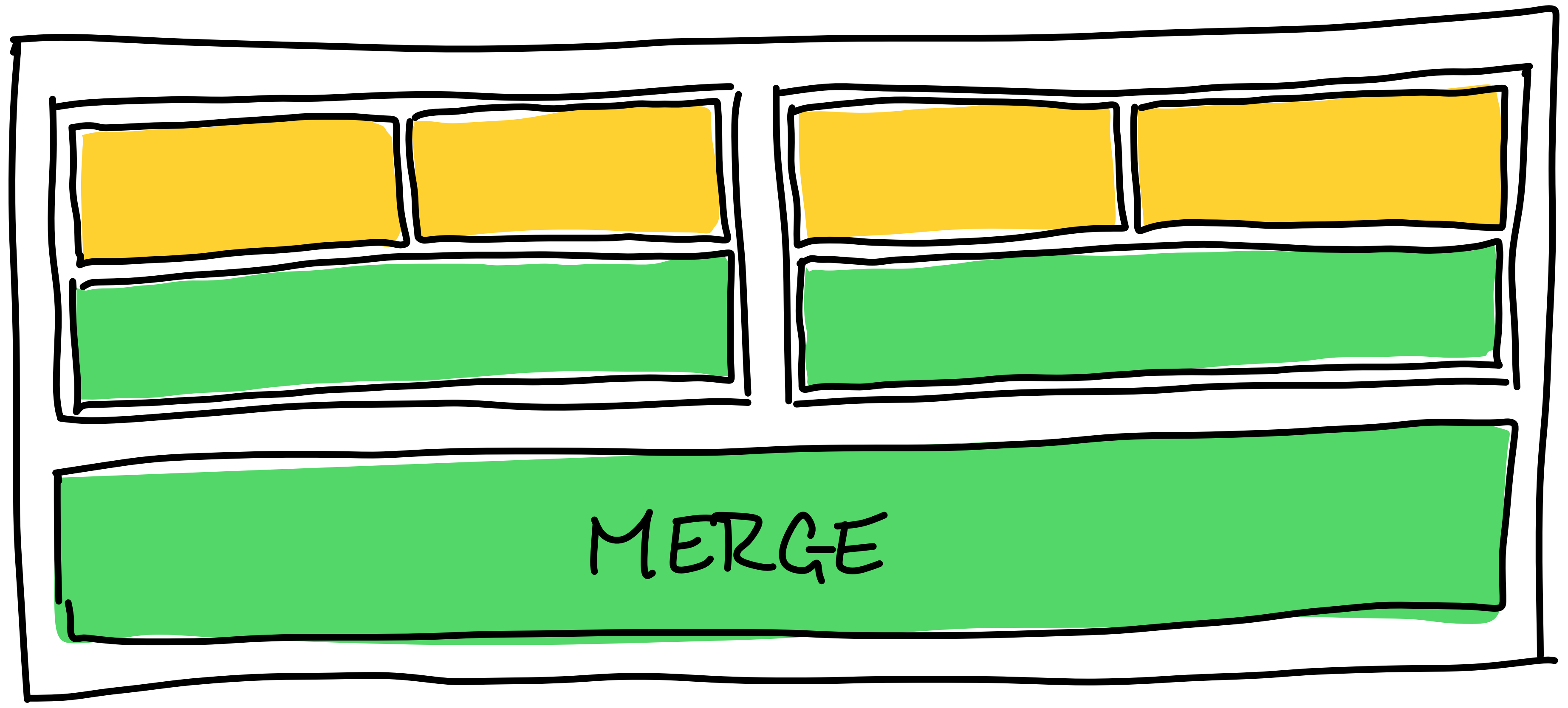
- Solution: $T(n) \leq O(n \log n)$



Another classic divide and conquer problem

Mergesort

- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.
- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays
- Recursive time equation: $T(n) \leq 2T(n/2) + O(n)$ with $T(1) = 0$.
- Solution: $T(n) \leq O(n \log n)$



Another classic divide and conquer problem

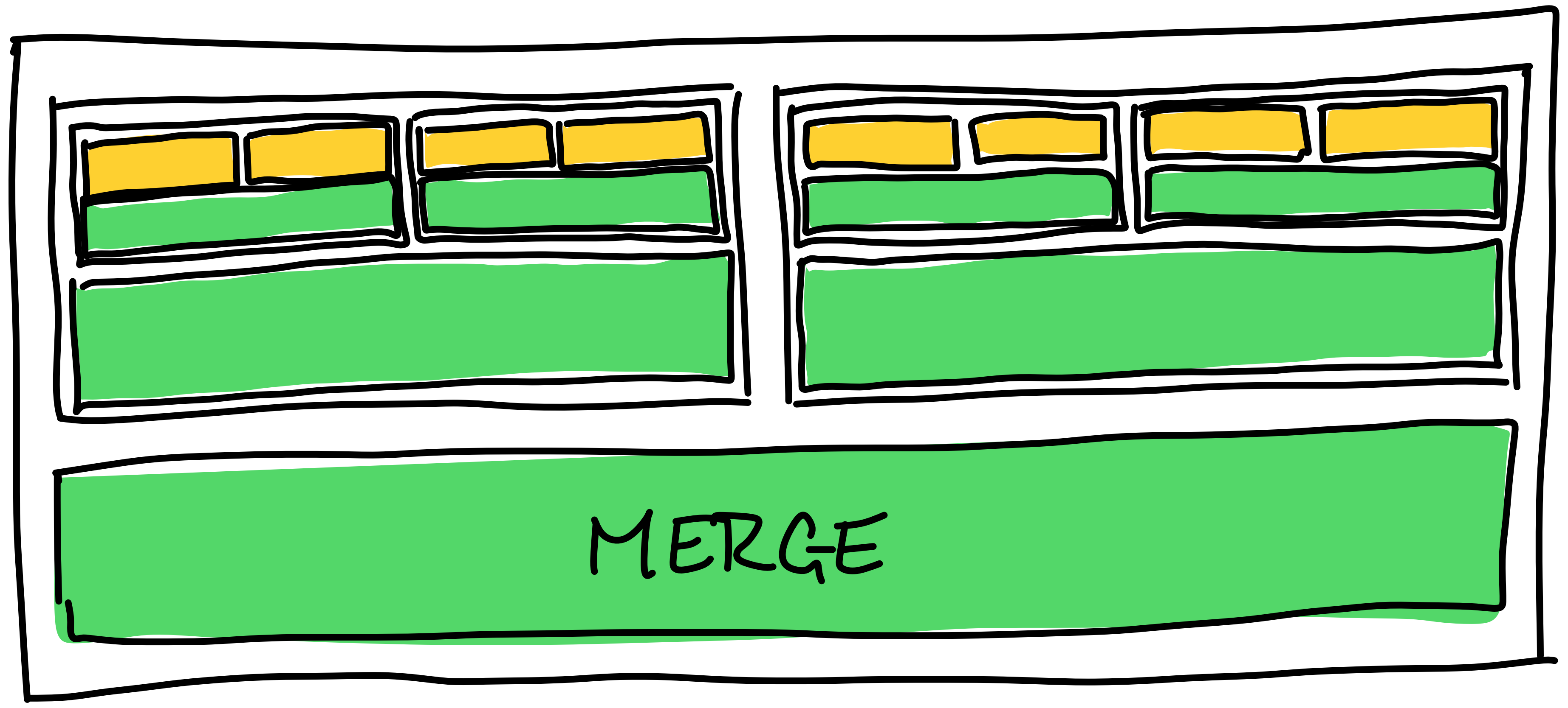
Mergesort

- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.

- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays

- Recursive time equation:
  $T(n) \leq 2T(n/2) + O(n)$ with $T(1) = 0$.

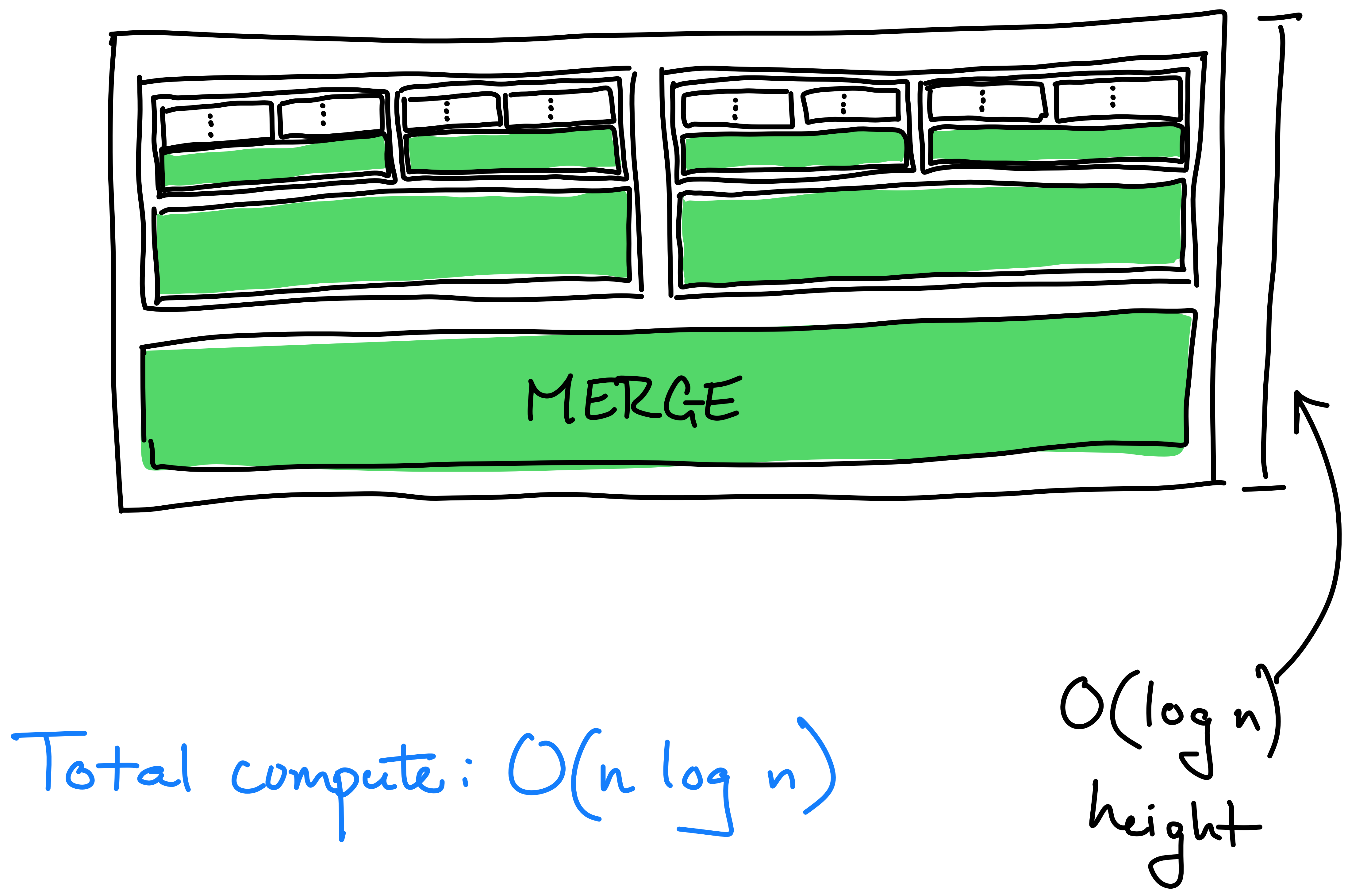
- Solution: $T(n) \leq O(n \log n)$



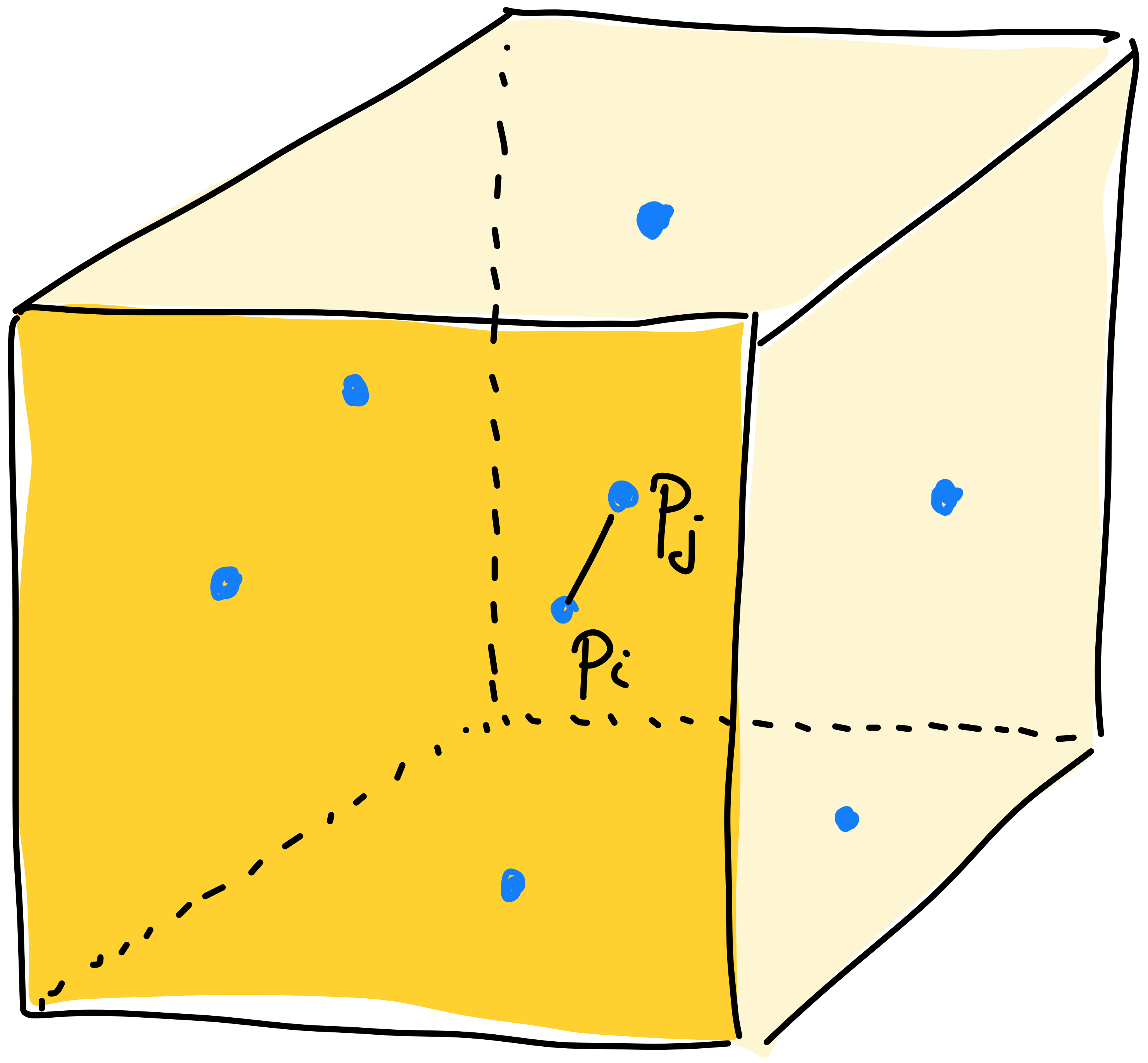
Another classic divide and conquer problem

Mergesort

- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.
- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays
- Recursive time equation: $T(n) \leq 2T(n/2) + O(n)$ with $T(1) = 0$.
- Solution: $T(n) \leq O(n \log n)$



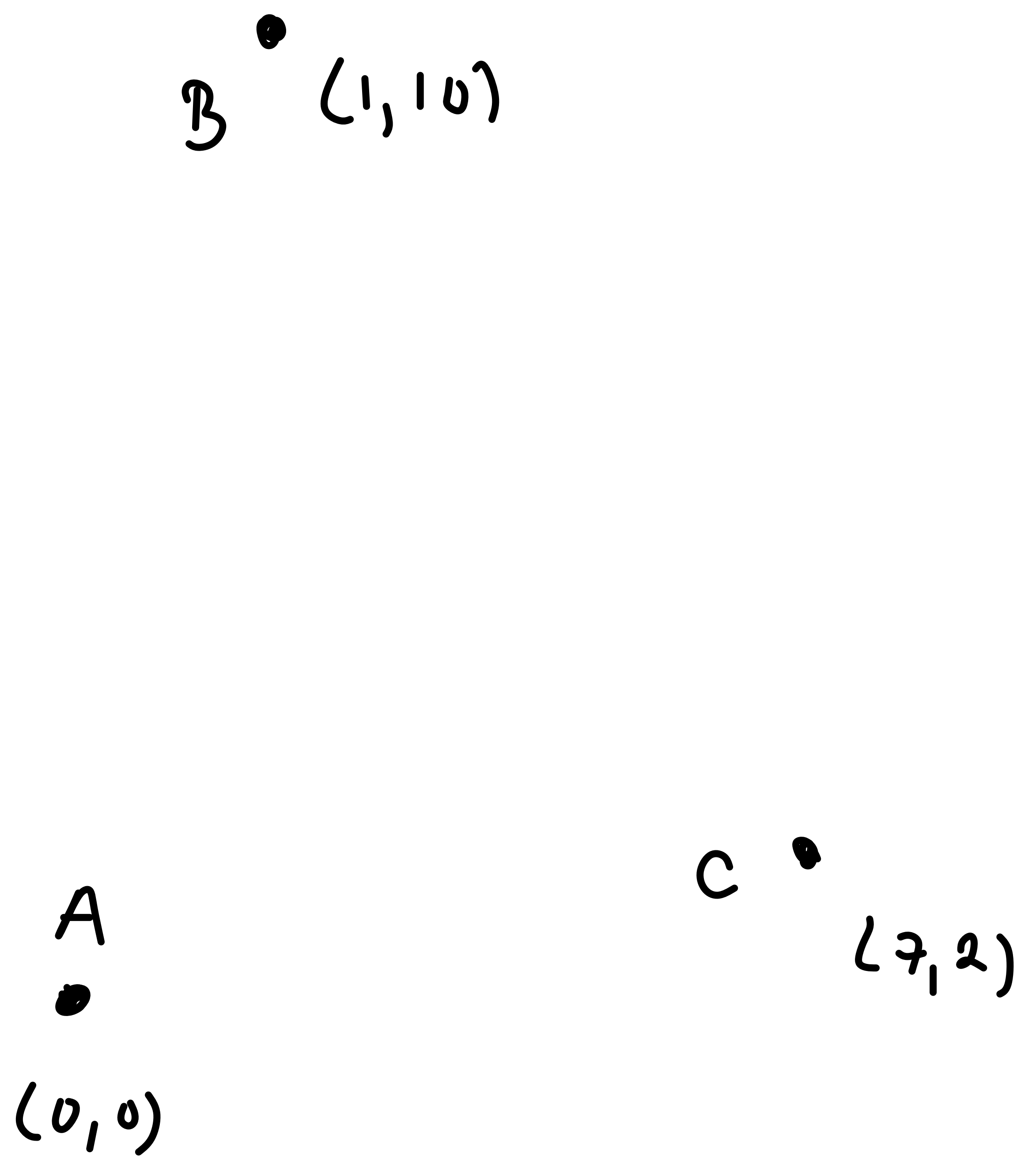
Euclidean closest pair



- **Input**: A sequence of n points $p_1, \ldots, p_n \in \mathbb{R}^d$
- **Find**: The pair p_i, p_j minimizing $||p_i - p_j||$.
- **Brute force algorithm**: Try all pairs. $O(n^2 d)$ time.
- Is there a better algorithm for small d?
  - In 1D for example, we can sort and then compare nearest neighbors for $O(n \log n)$.
  - Can we do better?

2D Euclidean closest pair

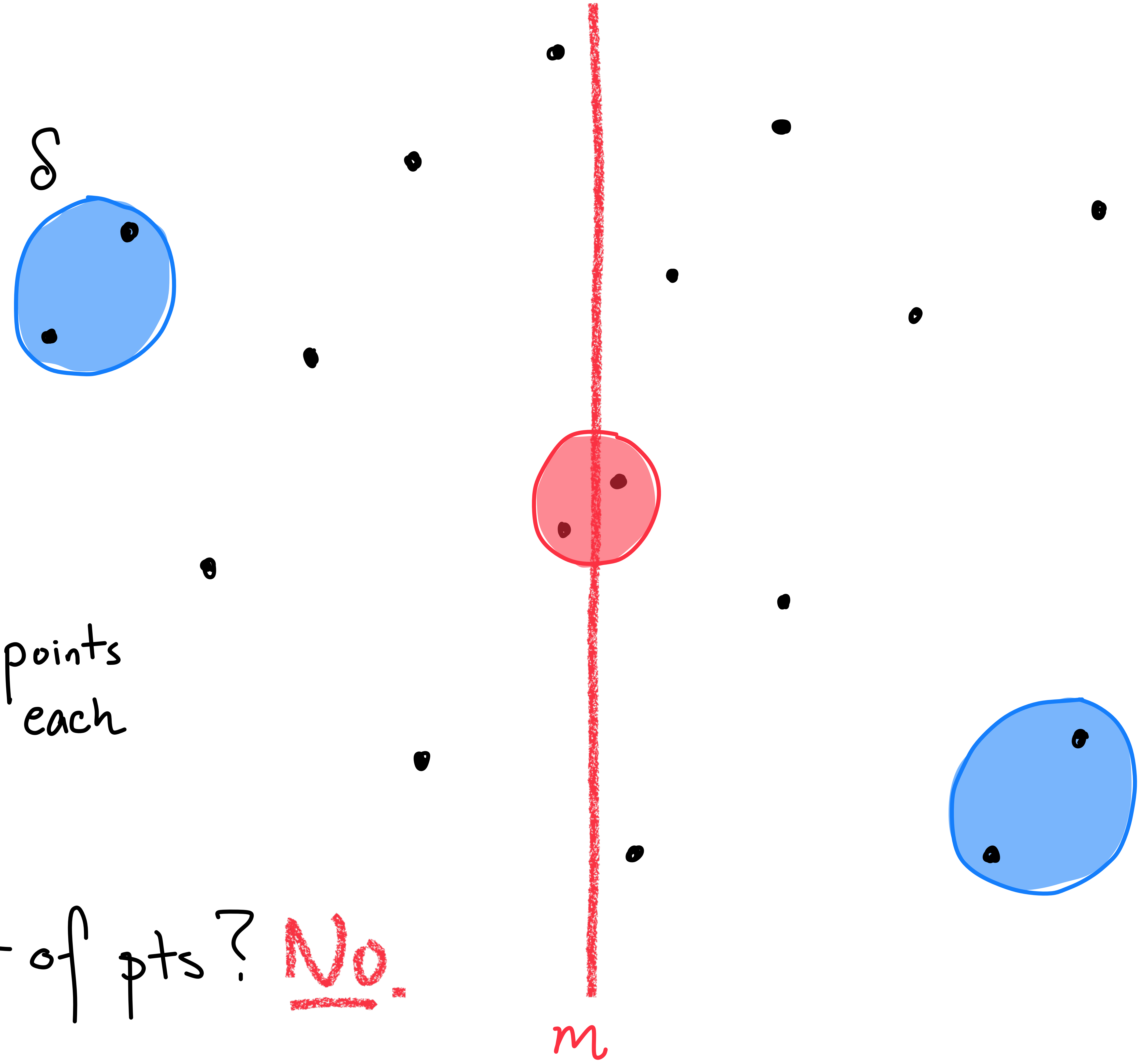
- Sorting on first coordinate will not work
- No single direction for sorting guarantees success
- **Divide and conquer algorithm:**
  - Need to figure out a way to subdivide the problem
  - Then build solution from best solutions to both halves. This will require extra processing



Sorting gives A, B, C while shortest pair is A–C.

Split across x-coordinate anyways

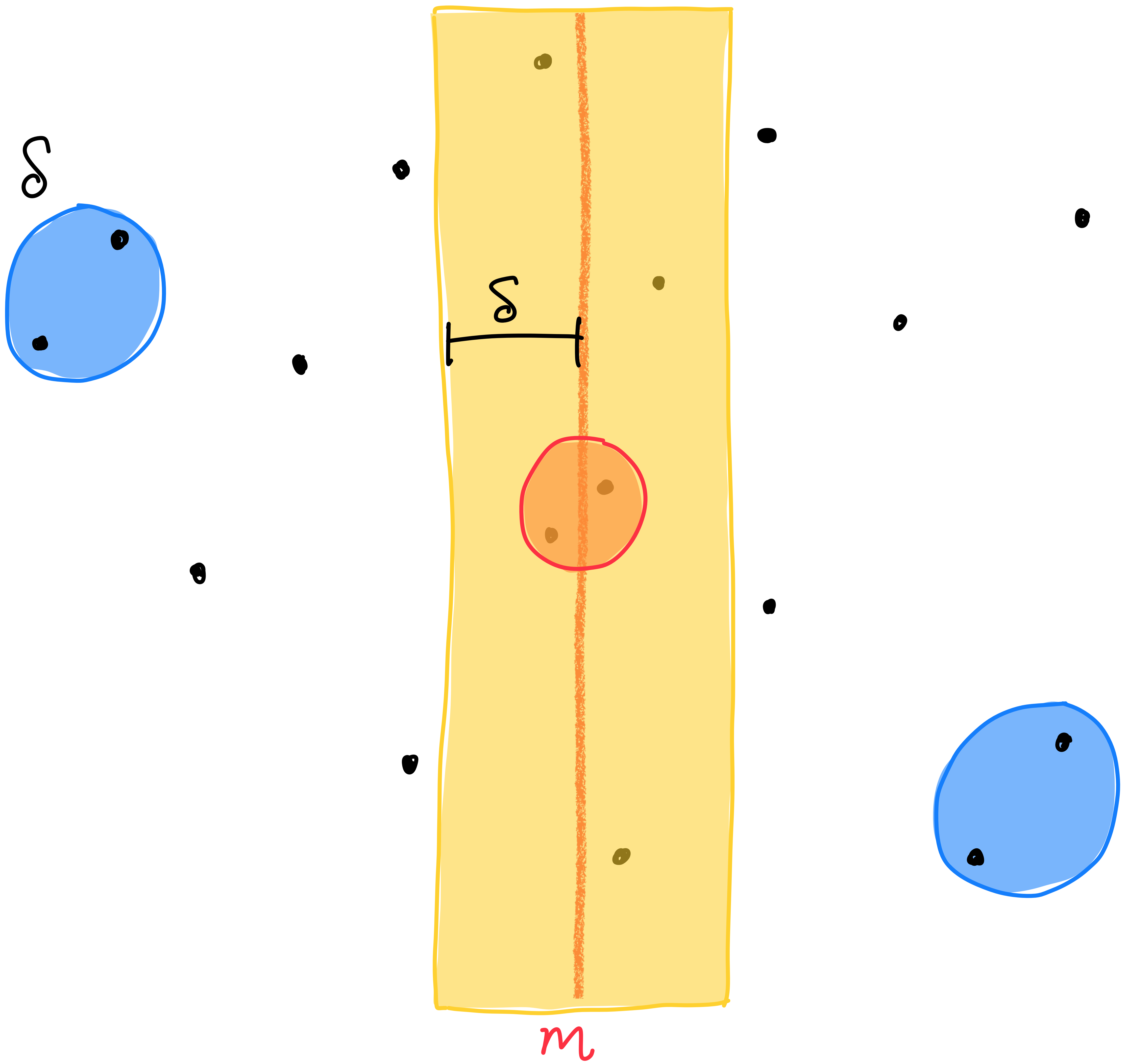
- Let's split according to x-coordinate anyways
- Let m be the median x-coordinate
- Divide the set into points $\{p : p_1 \leq m\}$ and $\{p : p > m\}$ ← $\frac{n}{2}$ points each
- Let δ be the minimum of the two solutions



Is this guaranteed to be the closest set of pts? No.

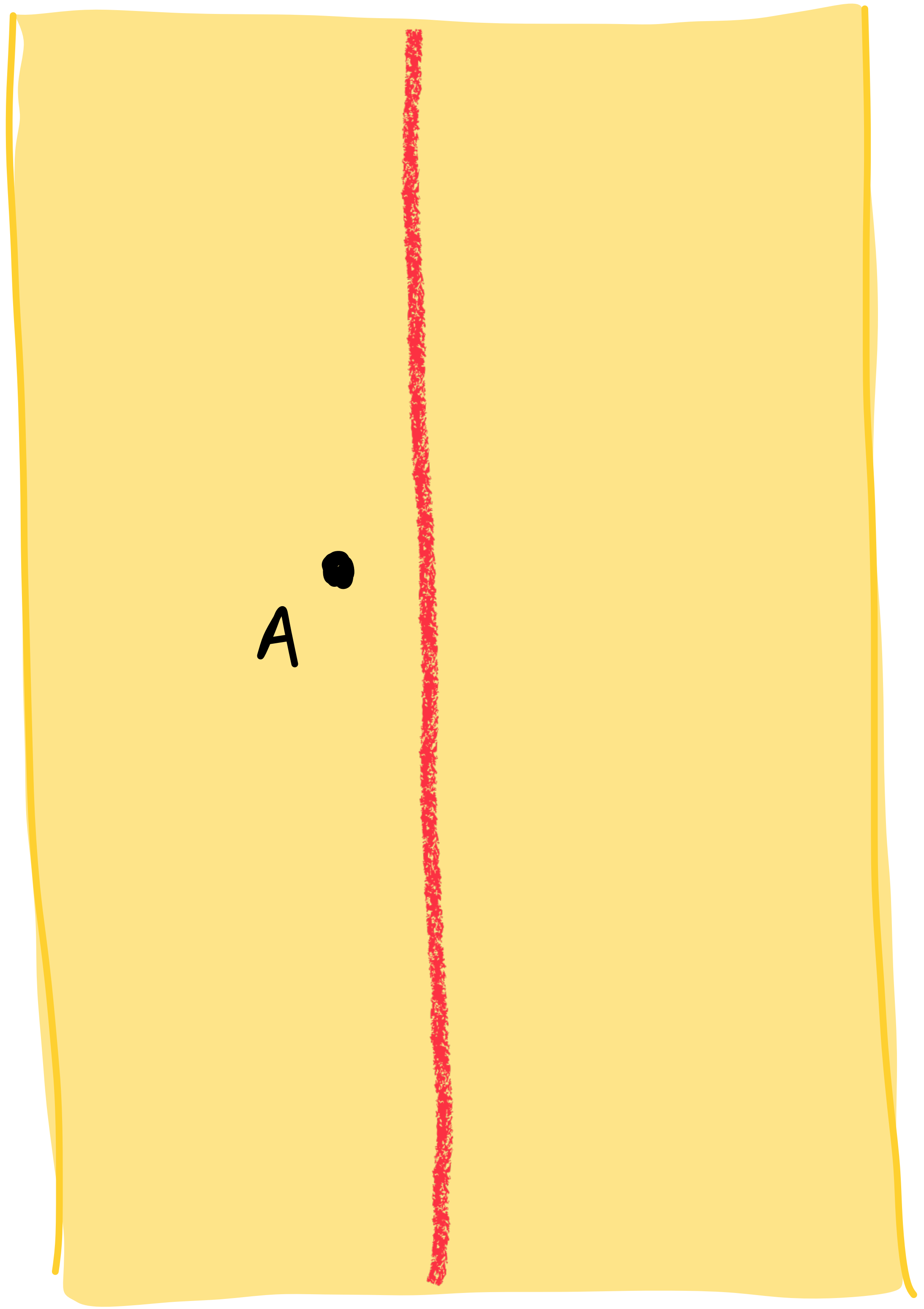
The “conquer” aspect of the algorithm

- We only need to worry about pairs that are **both** split by the median and $< \delta$ distance apart
  - During “conquer” step, only need to look at vertices in the δ-width band
  - Within the band, only need to compare points with y-coordinates that differ by $< \delta$

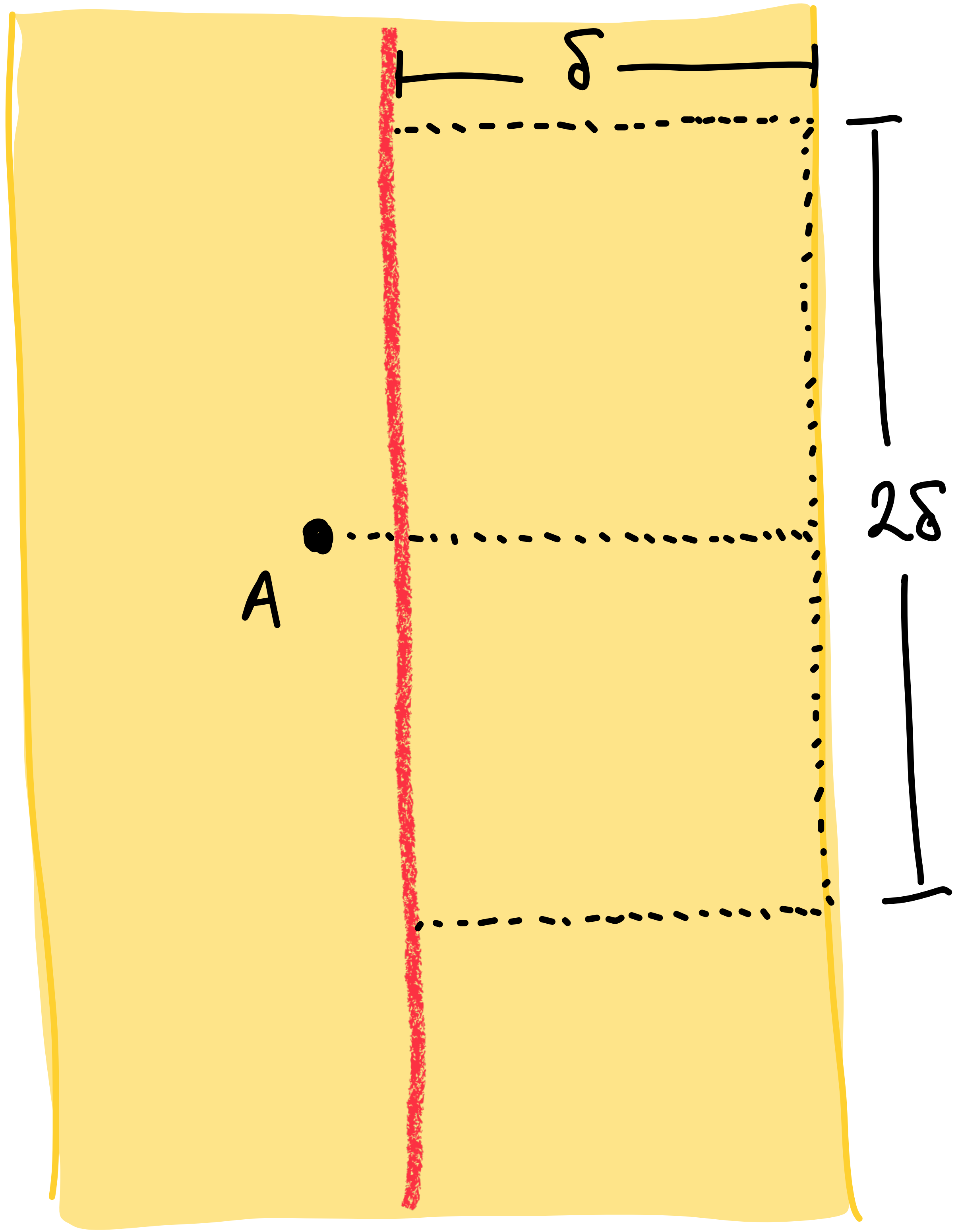


Close-up analysis

How many pts across the median do we have to compare with A ?



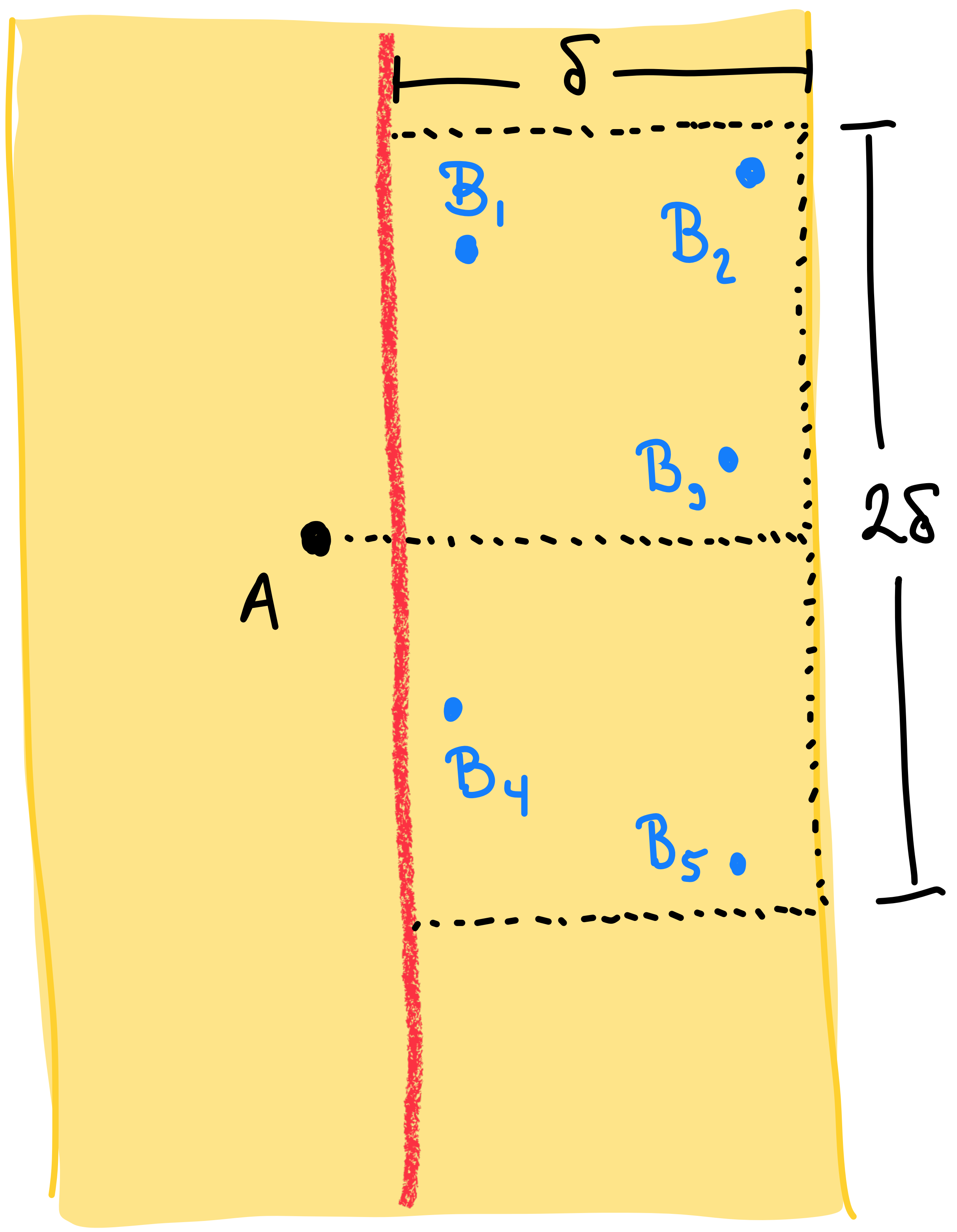
Close-up analysis



How many pts across the median do we have to compare with A?

In order to be distance ≤ δ from A, a pt B must lie in this box.

Close-up analysis



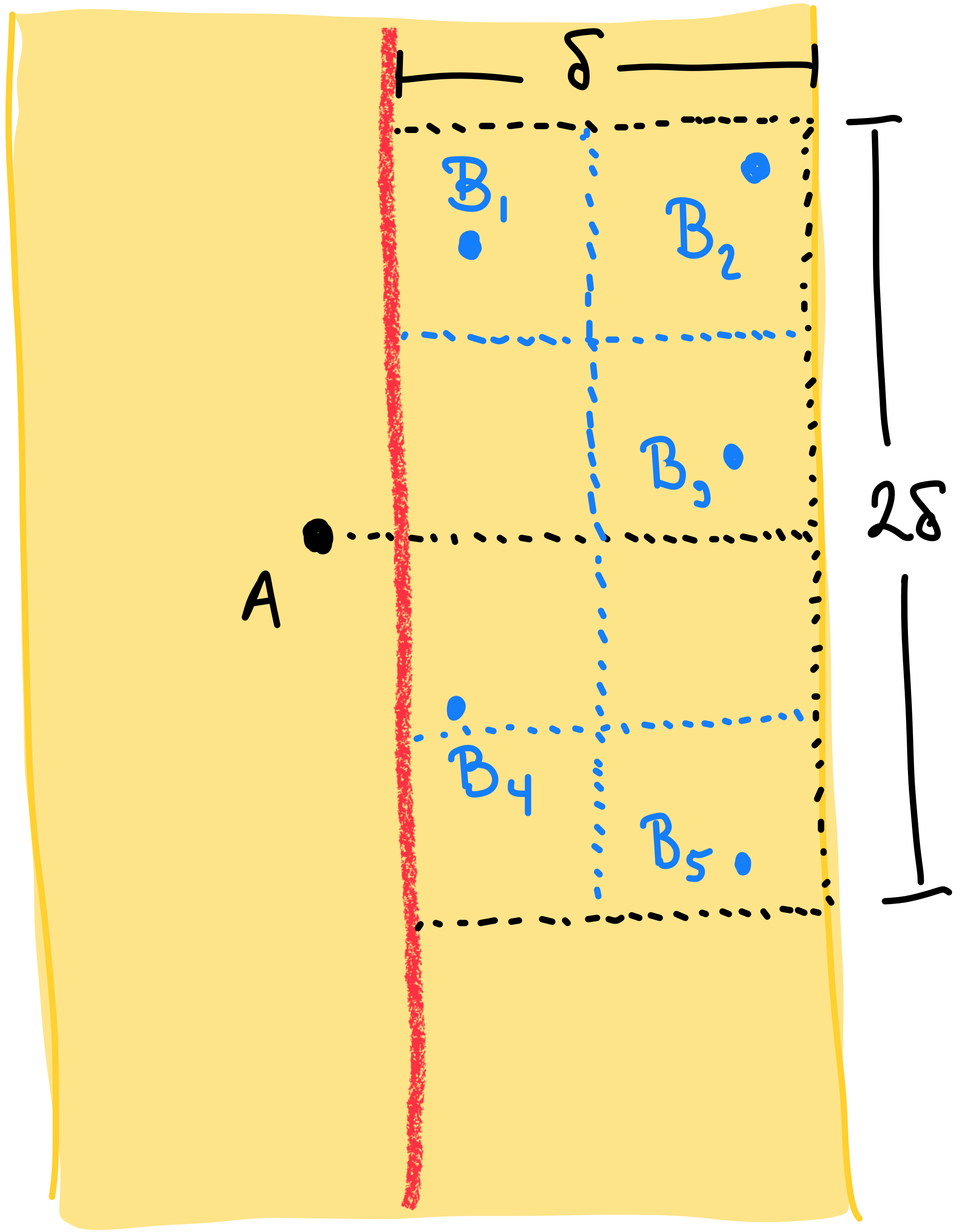
How many pts across the median do we have to compare with A?

In order to be distance $\leq \delta$ from A, a pt B must lie in this box.

How many such points B_i can exist?

Note: $\| B_i - B_j \| \geq \delta$ since on the same side.

Close-up analysis



How many pts across the median do we have to compare with A?

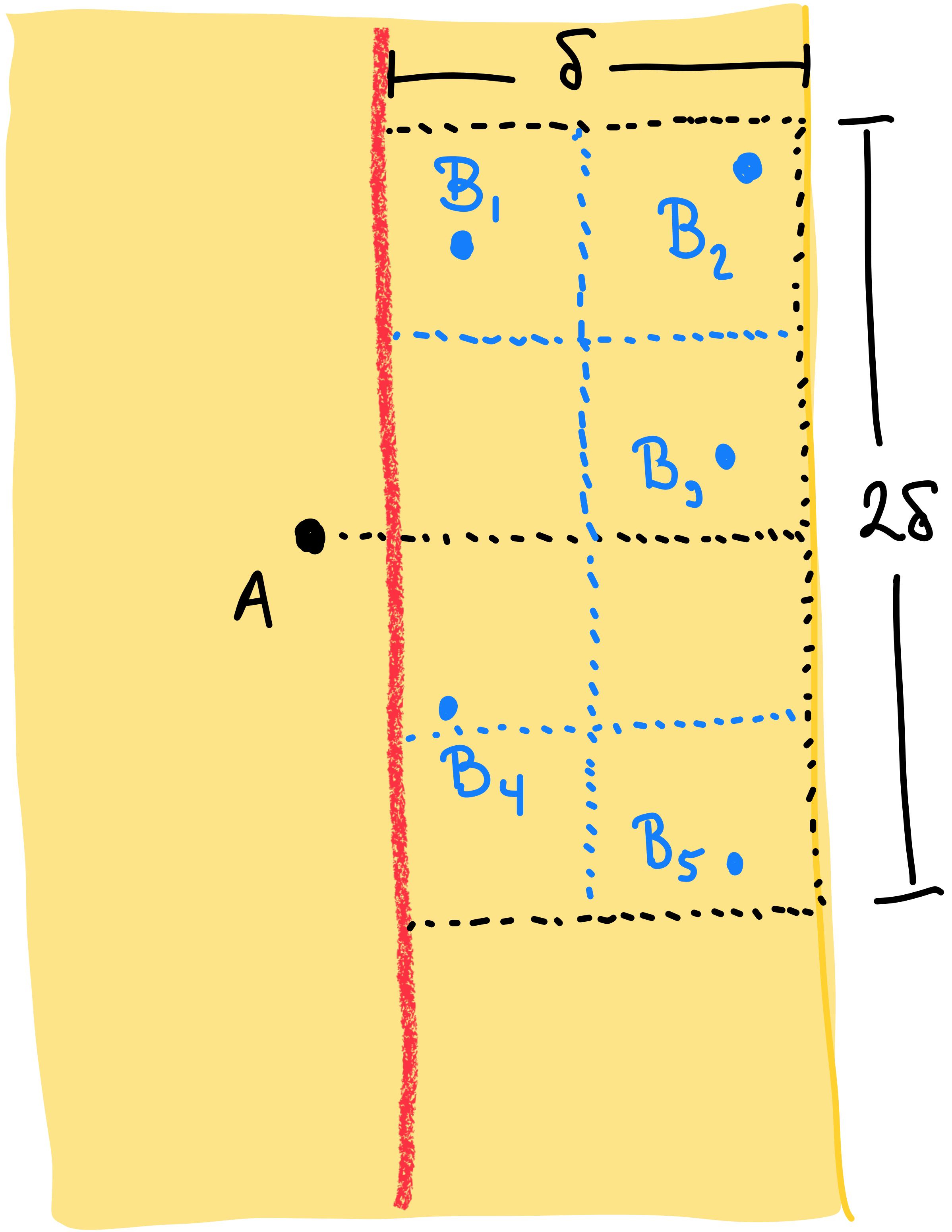
In order to be distance $\leq \delta$ from A, a pt B must lie in this box.

How many such points B_i can exist?

Note: $\|B_i - B_j\| \geq \delta$ since on the same side.

Each $\frac{\delta}{2} \times \frac{\delta}{2}$ box can have at most 1 point B_i. So at most 8 points.

The full conquer subroutine



- Let M be the set of points in band.
- Sort the points in M by y-coordinate
- For each point $a \in M$, compare a to the 8 points before and 8 points after a in the sorting.
- By analysis, this checks all possible pairs of distance $< \delta$.

Divide and conquer algorithm

Total: $T(n) = 2T(\frac{n}{2}) + O(n \log n)$

- **Divide step**:
  - Compute median m and divide into two subproblems ← $O(n \log n)$ time with sorting
  - Recursively calculate shortest distance for subproblems ← $2T(\frac{n}{2})$ by recursion
- **Conquer step**:
  - Compute the set of points in the band M ← $O(1)$ time since we sorted for the median
  - Sort M by y-coordinate ← $O(n \log n)$ time as potentially n points in band.
  - Compare points in sorted M with the next 8 points and update if closer pair found.

↑ $O(n)$ time since sorted.

Better divide and conquer algorithm

- **Preprocessing**:
  - Sort points according to x-coordinate for list X
  - Sort points according to y-coordinate for list Y

  } $O(n \log n)$ time. Only once.

- **Divide step** (sorted lists X, Y):
  - Compute median m by x-coordinate ← $O(1)$ time since sorted
  - Divide X into X_L, X_R. Filter Y into Y_L and Y_R. ← $O(n)$ time, once-through the list
  - Recursively solve (X_L, Y_L) and (X_R, Y_R) problems for δ ← $2\,T(\frac{n}{2})$ by recursion
- **Conquer step**:
  - Filter Y into the band M of x-coordinates $m \pm \delta$ ← $O(n)$ time
  - Compare M to the next 8 points and update if closer point is found. ← $O(n)$ time

Total time:

$$T(n) = 2T(\tfrac{n}{2}) + O(n)$$

$$\Downarrow$$

$$T(n) = O(n \log n)$$

Analysis divide and conquer runtimes

The master theorem

- For solving recursive equations of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \text{ and } T(<b) = O(1)$$

- Different cases based on how $f(n)$, a, and b compare:

Analysis divide and conquer runtimes

The master theorem

- For solving recursive equations of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^k) \text{ and } T(<b) = O(1)$$

- Different cases based on how $f(n)$, a, and b compare:
  - If $a < b^k$, then $T(n) = O(n^k)$
  - If $a = b^k$, then $T(n) = O(n^k \log n)$
  - If $a > b^k$, then $T(n) = O(n^{\log_b a})$