Lecture 8

Divide and conquer

Chinmay Nirkhe | CSE 421 Spring 2025

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:

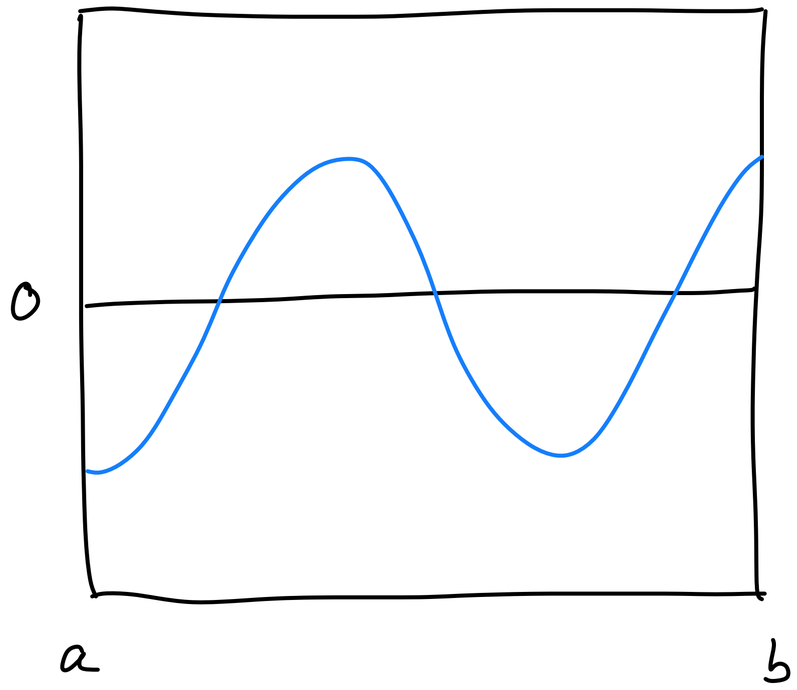
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \text{ and } T(<b) = O(1)$$

Examples of divide and conquer

- Mergesort, Quicksort (sort of)
- Binary search (today)
- Euclidean closest pair (today)
- Rank selection, Median finding

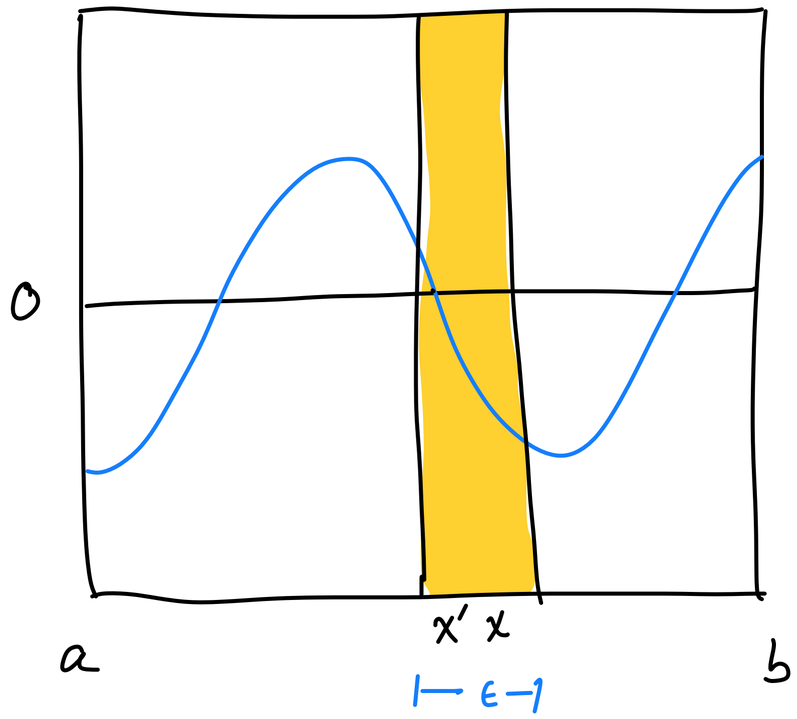
Binary search for roots of a function

- **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 $\epsilon > 0$



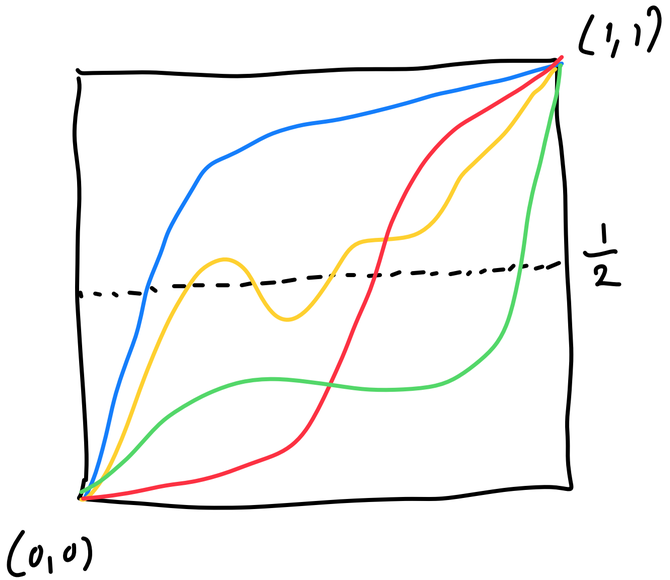
Binary search for roots of a function

- **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**:

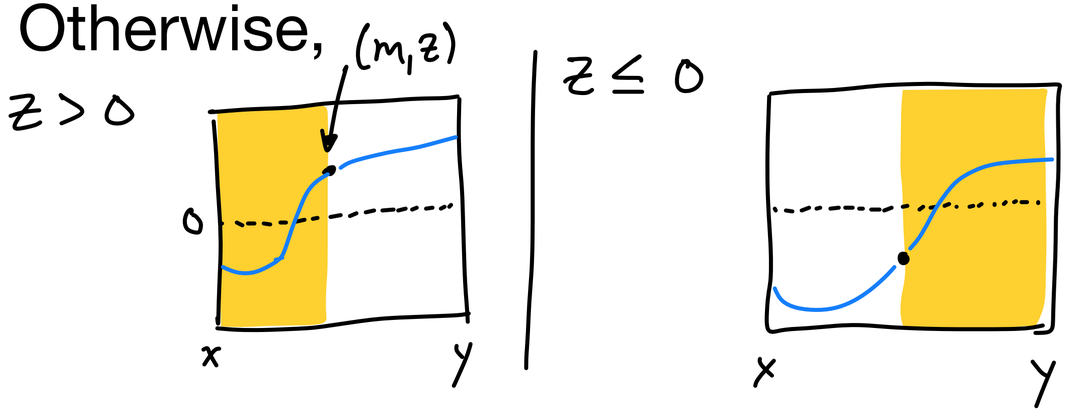


Any function must cross the midline at some point by continuity.

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, $\log_2\left(\frac{b-a}{2\epsilon}\right)$ recursions.
- Each recursion costs $O(1)$ arithmetic operations plus 1 query to f.
- Runtime: $O(\log(b-a) + \log(1/\epsilon))$ queries to f.

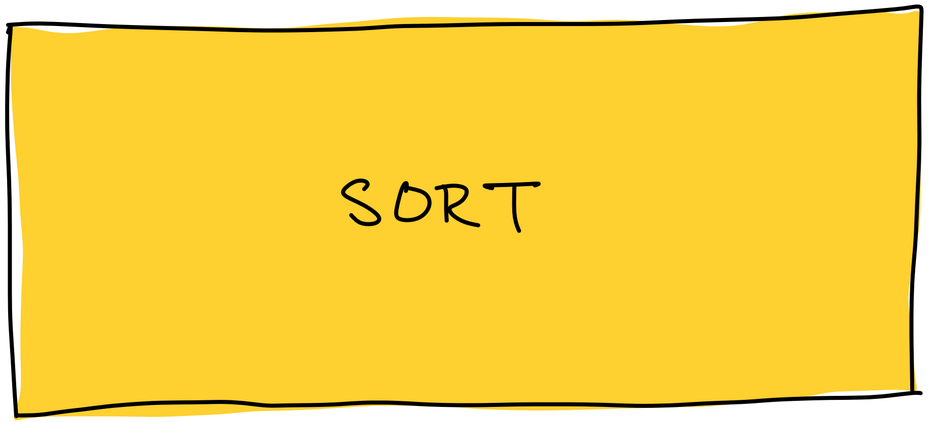
Runtime analysis

- 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

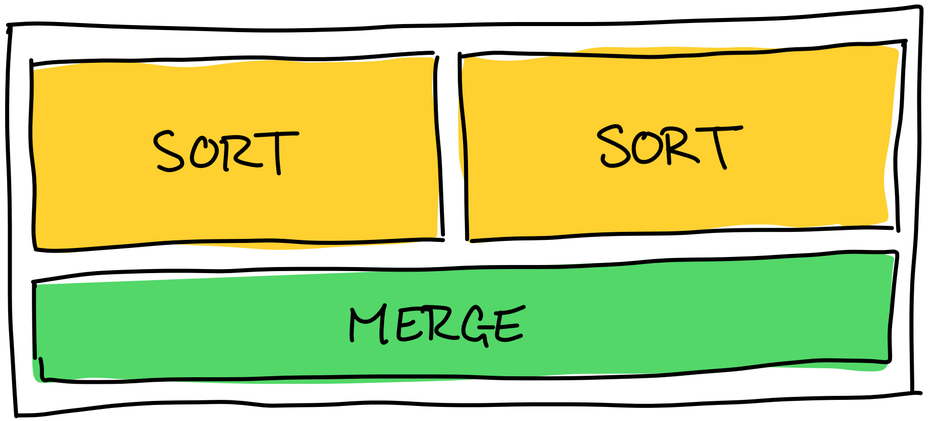
- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.
- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays
- Recursive time equation:
  $T(n) \leq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



Another classic divide and conquer problem

Mergesort

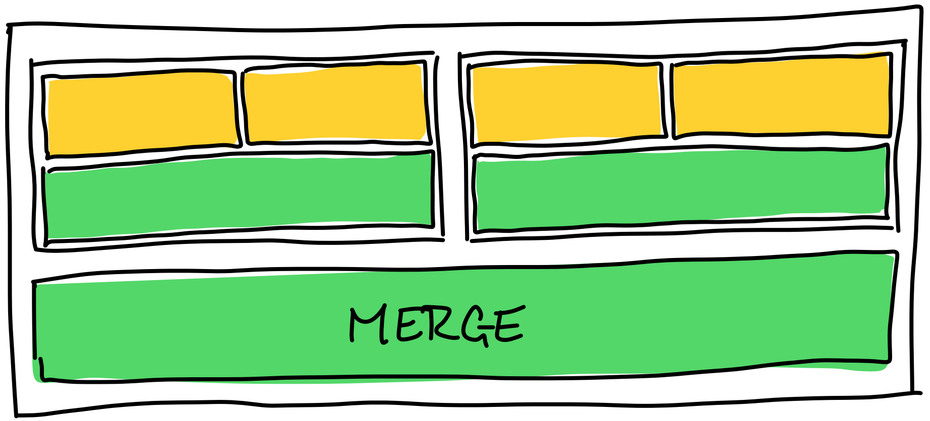
- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.
- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays
- Recursive time equation:
  $T(n) \leq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



Another classic divide and conquer problem

Mergesort

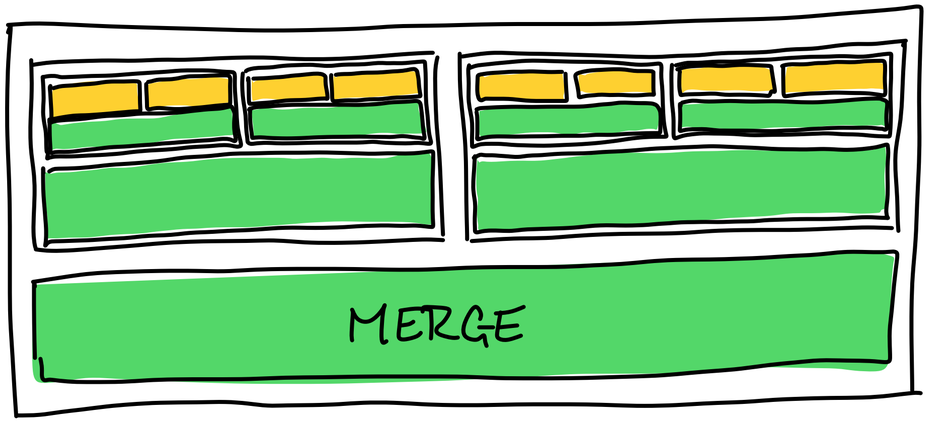
- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.
- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays
- Recursive time equation:
  $T(n) \leq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



Another classic divide and conquer problem

Mergesort

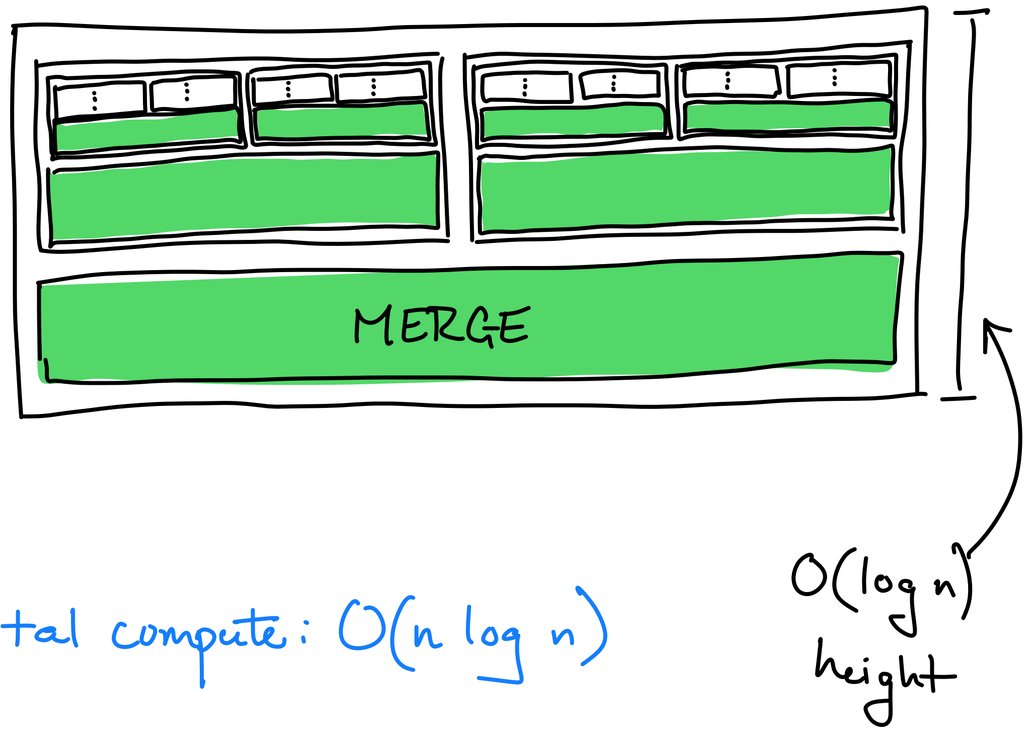
- To sort an array of n entries, recursively sort the first half and recursively sort the second half. Then *merge* the two sorted lists.
- Merging two sorted arrays takes $O(n)$ time as we only have to compare current elements as we iterate through both arrays
- Recursive time equation:
  $T(n) \leq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



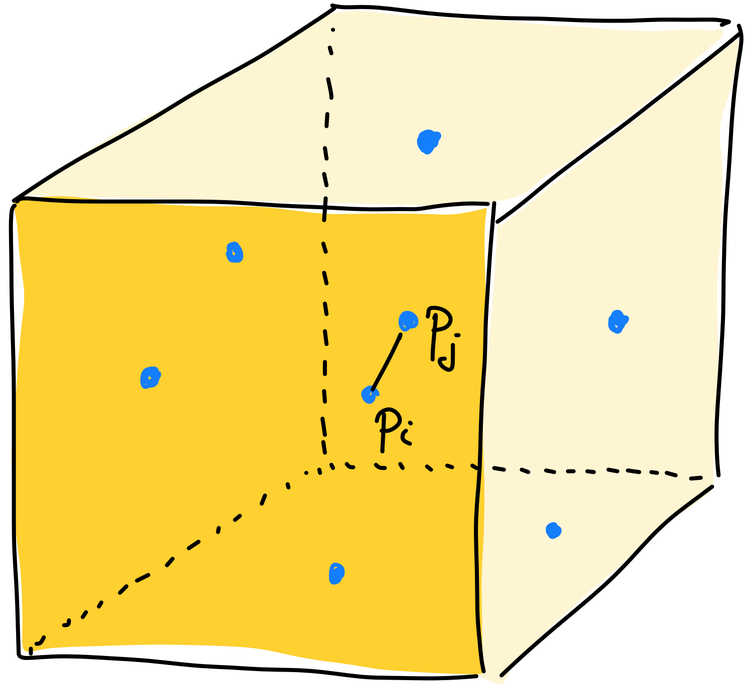
Another classic divide and conquer problem

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

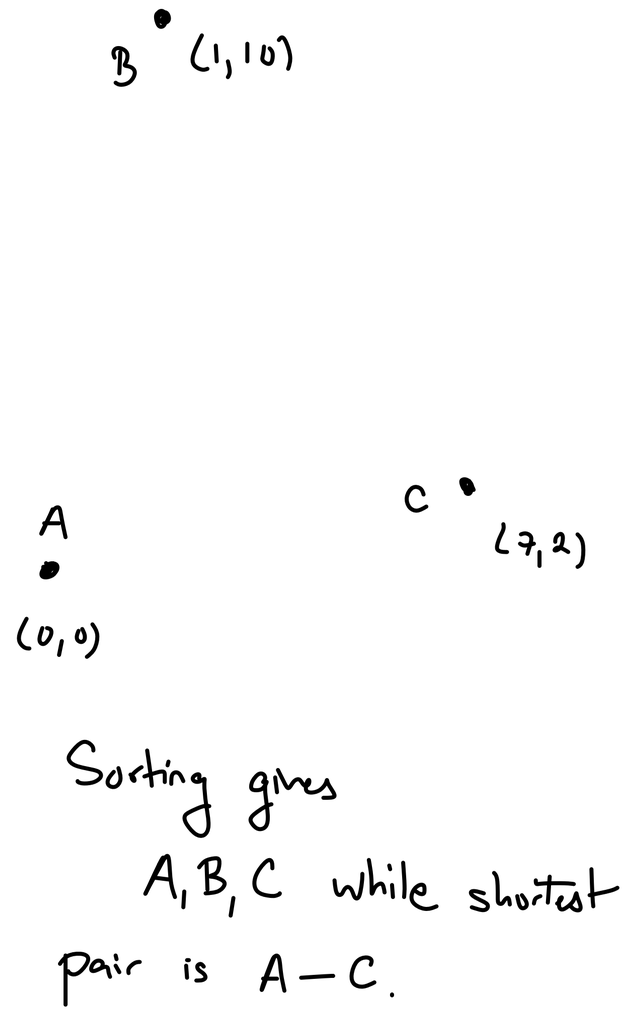


Euclidean closest pair



- **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?

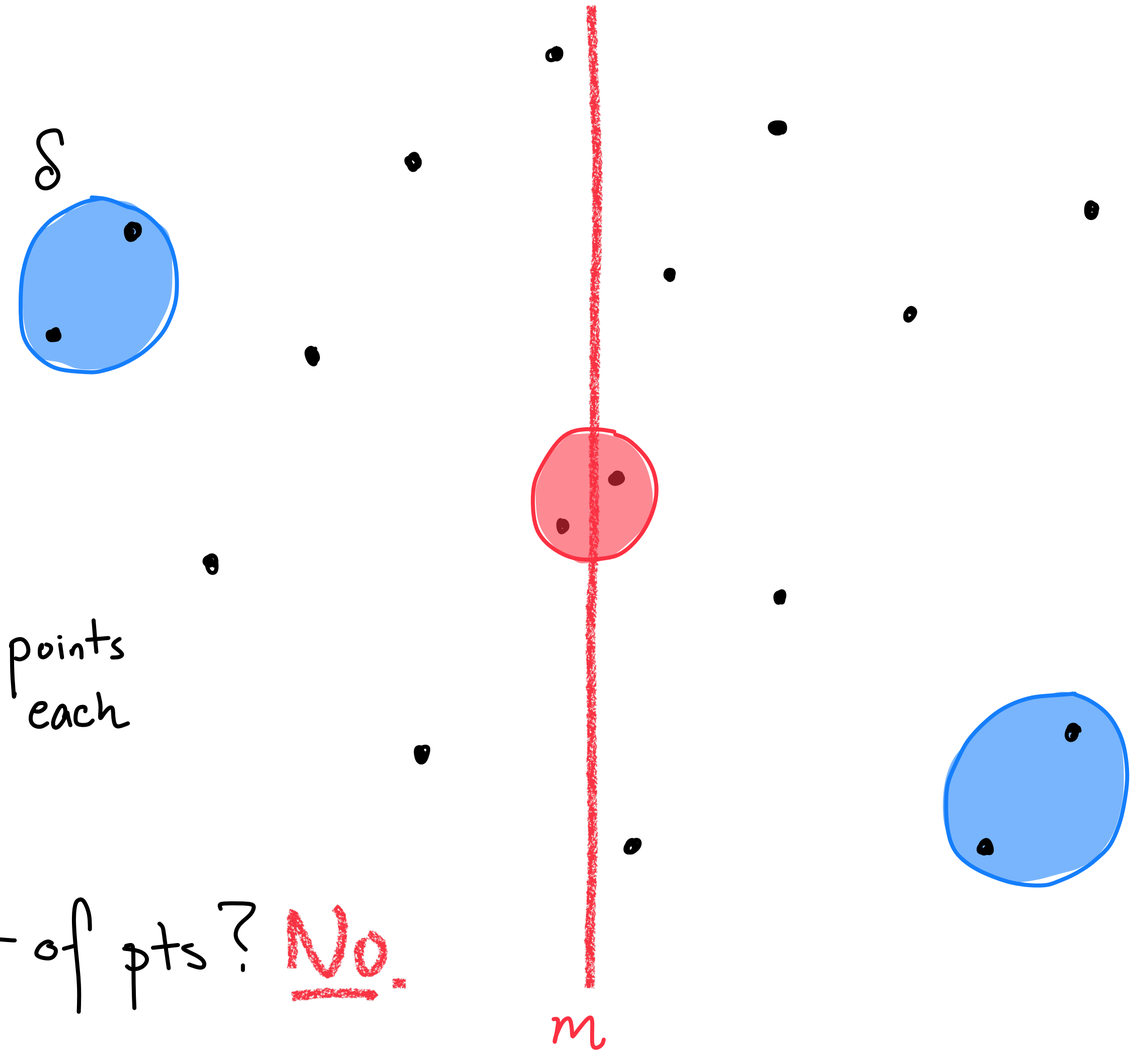
2D Euclidean closest pair



- 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

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 $\{p : p > m\}$
- Let δ be the minimum of the two solutions

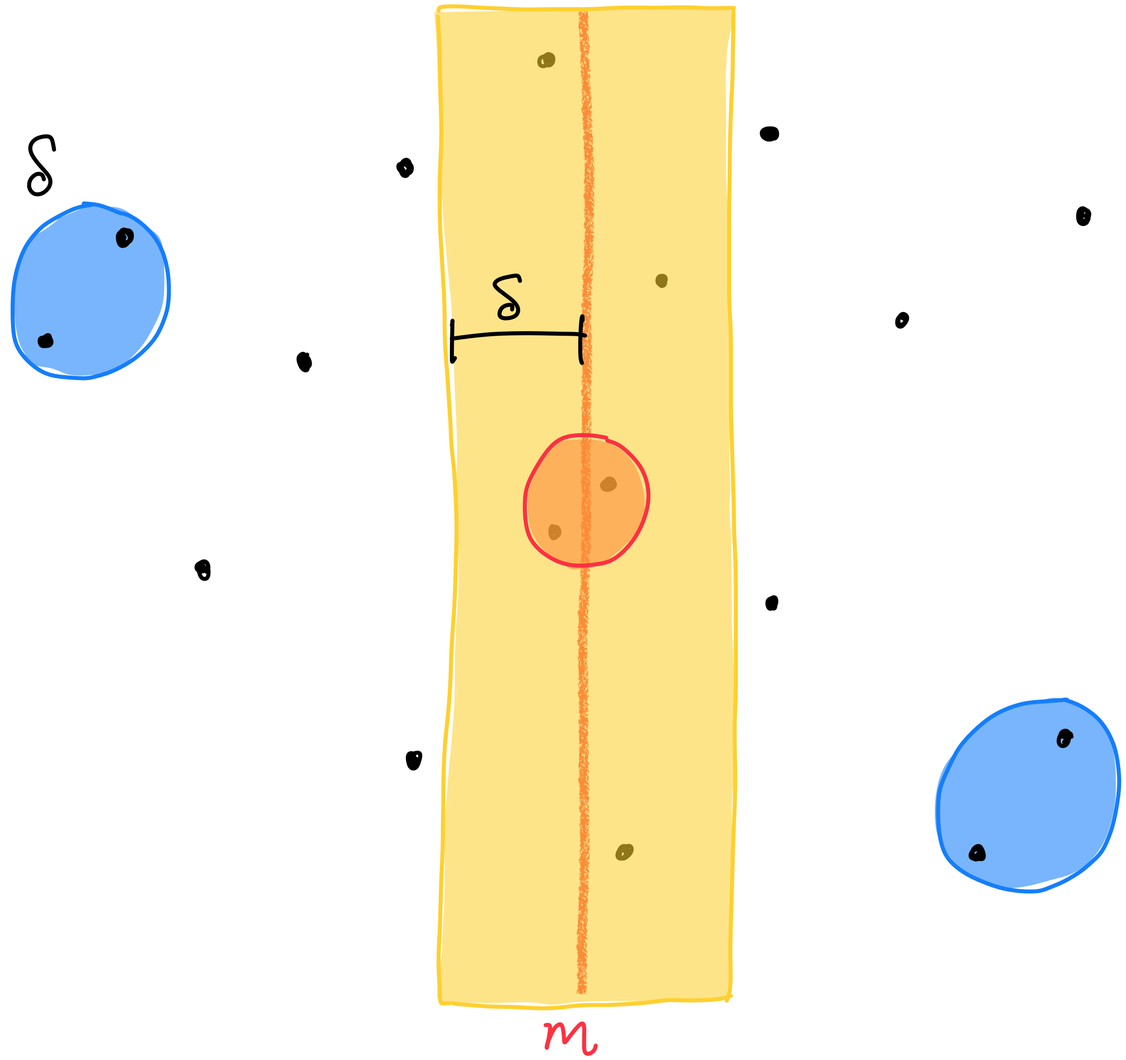


$\frac{n}{2}$ points each

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

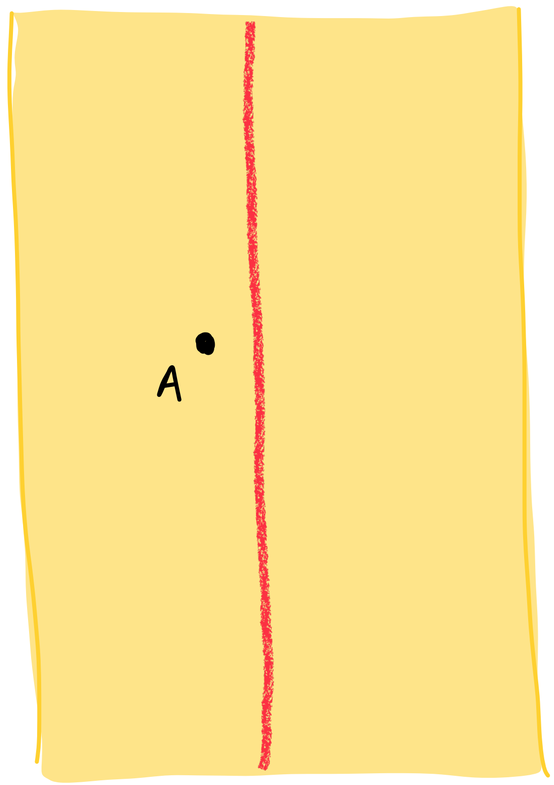
The “conquer” aspect of the algorithm

- 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 $< \delta$

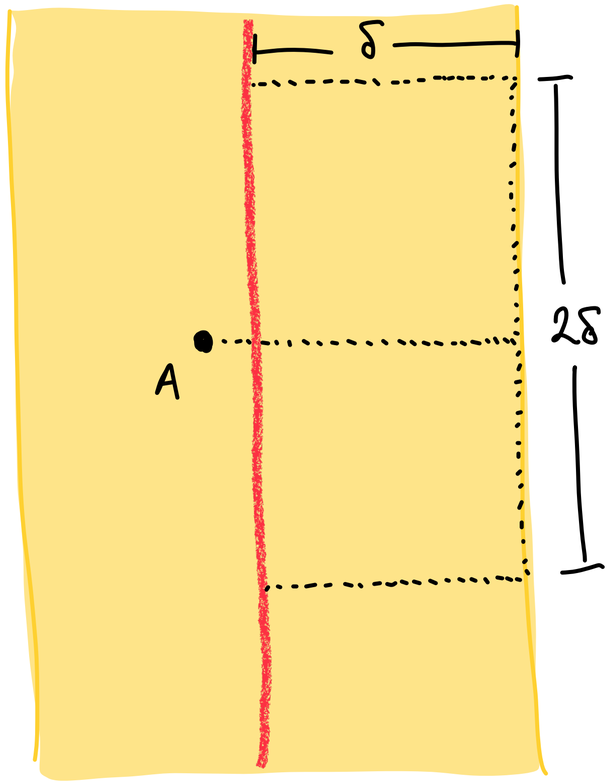


Close-up analysis

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



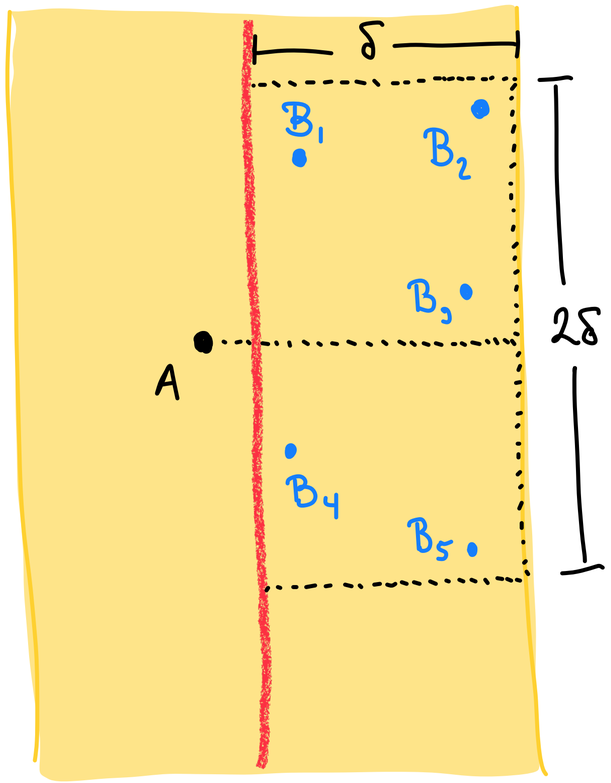
Close-up analysis



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



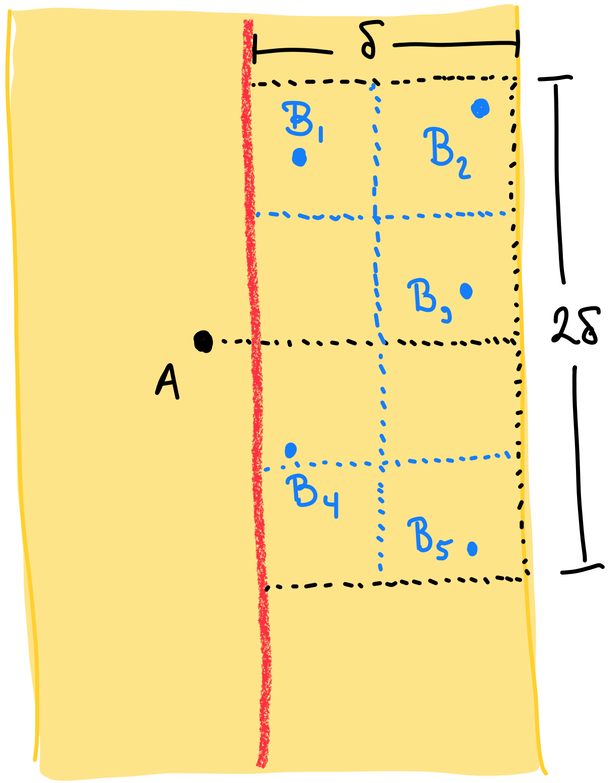
How many pts across the median do we have to compare with A?

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?

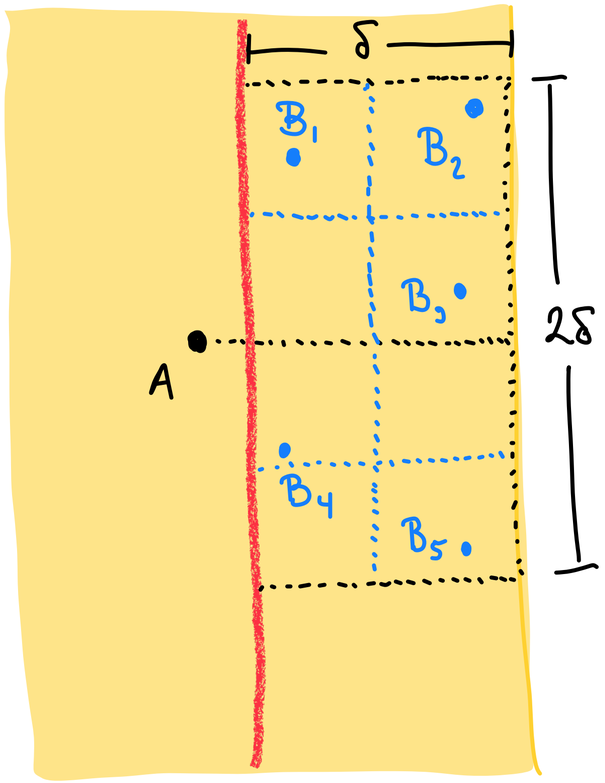
In order to be distance $\leq \delta$ from A, a pt B must lie in this box.

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 ← $2\,T(\frac{n}{2})$ 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:

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

⇓

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

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

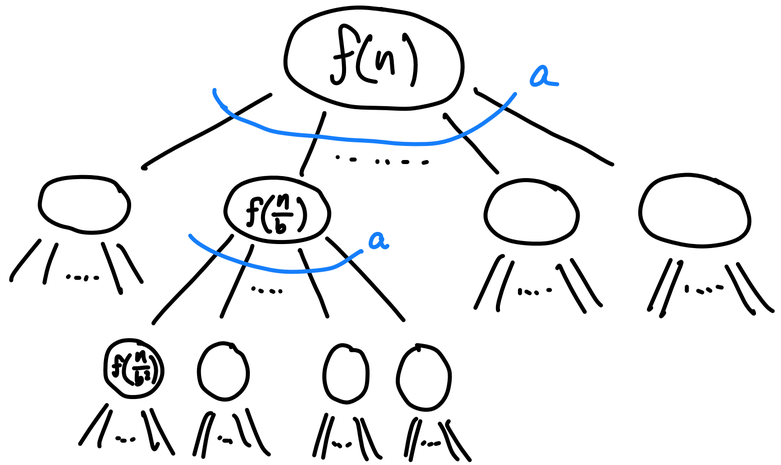
- For solving recursive equations of the form

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

- 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 $T(n) = O(n^{\log_b a})$

Proof of the master theorem

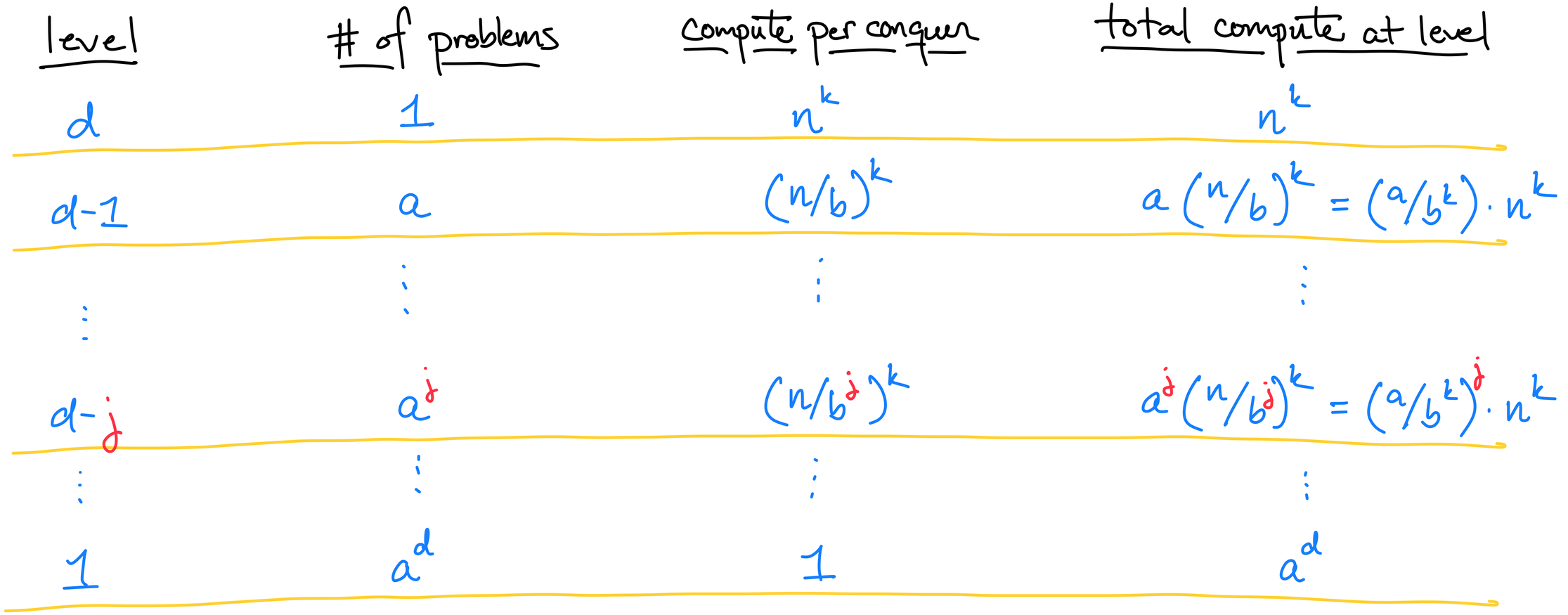
- **Proof strategy**:
  - Due to recursion, the problem has a tree like structure



  - Calculate the amount of work done by the “conquer” step at each level
  - Count how many levels of computation there are

Proof the master theorem

- Let $d = \lceil \log_b n \rceil$ so $n \leq b^d$



Proof the master theorem

- Let $d = \lceil \log_b n \rceil$ so $n \le b^d$

Total compute $= \sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j \cdot n^k$

- If $a < b^k$, then $\sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j \le \sum_{j=0}^{\infty} \left(\frac{a}{b^k}\right)^j = \left(1 - \frac{a}{b^k}\right)^{-1} \Rightarrow O(n^k)$.

- If $a = b^k$, then $\sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j = \sum_{j=0}^{d} 1 = d + 1 \Rightarrow O(n^k \log n)$.

- If $a > b^k$, then $\sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j = \frac{\left(\frac{a}{b^k}\right)^d - 1}{\left(\frac{a}{b^k}\right) - 1} \Rightarrow O\left(\left(\frac{a}{b^k}\right)^d \cdot n^k\right) = O(a^d) = O(n^{\log_b a})$

last row of prev table.

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 $T(n) = O(n^k)$ ← most of the compute is in the largest conquer step
  - If $a = b^k$, then $T(n) = O(n^k \log n)$ ← each level has a commensurate ammount of compute
  - If $a > b^k$, then $T(n) = O(n^{\log_b a})$ ← the number of leaves dominates the computation