CSE 421

Divide and Conquer: Finding Root
Closest Pair of Points

Shayan Oveis Gharan

Finding the Root of a Function

Finding the Root of a Function

Given a continuous function f and two points a < b such that

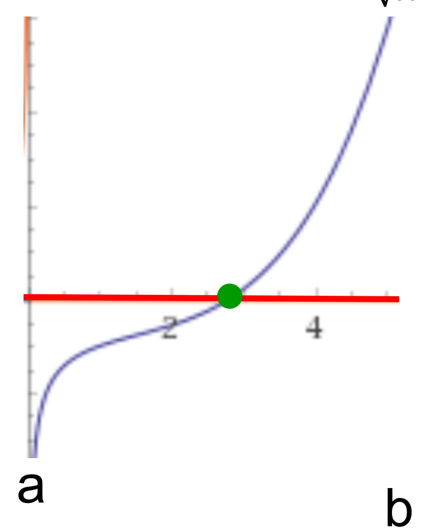
$$f(a) \leq 0$$
$$f(b) \geq 0$$

Find an approximate root of f (a point c where $f(c) = 0$).

f has a root in $[a, b]$ by intermediate value theorem

Note that roots of f may be irrational,
So, we want to approximate
the root with an arbitrary precision!

$$\text{f}(x) = \sin(x) - \frac{100}{\sqrt{x}} + x^4$$



A Naiive Approch

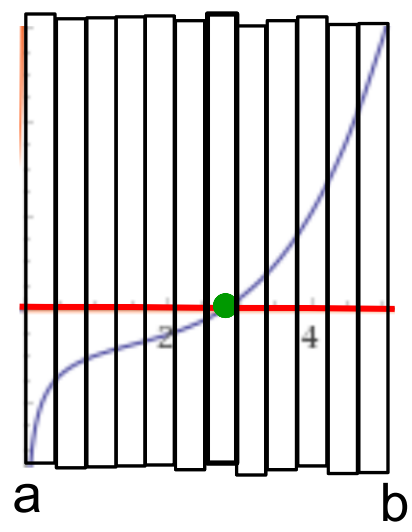
Suppose we want ϵ approximation to a root.

Divide [a,b] into $n = \frac{b-a}{\epsilon}$ intervals. For each interval check

$$f(x) \leq 0, f(x+\epsilon) \geq 0$$

This runs in time $O(n) = O(\frac{b-a}{\epsilon})$

Can we do faster?



D&C Approach (Based on Binary Search)

Bisection(a,b, ε)

if $(b - a) < \epsilon$ then

return (a)

else

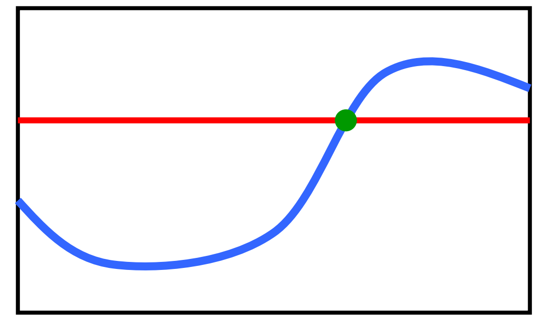
$m \leftarrow (a + b)/2$

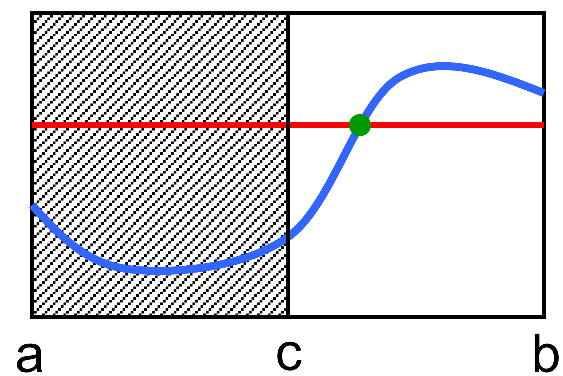
if $f(m) \leq 0$ then

return(Bisection(c, b, ε))

else

return(Bisection(a, c, ε))





Time Analysis

Let $n = \frac{a-b}{\epsilon}$

And $c = (a+b)/2$

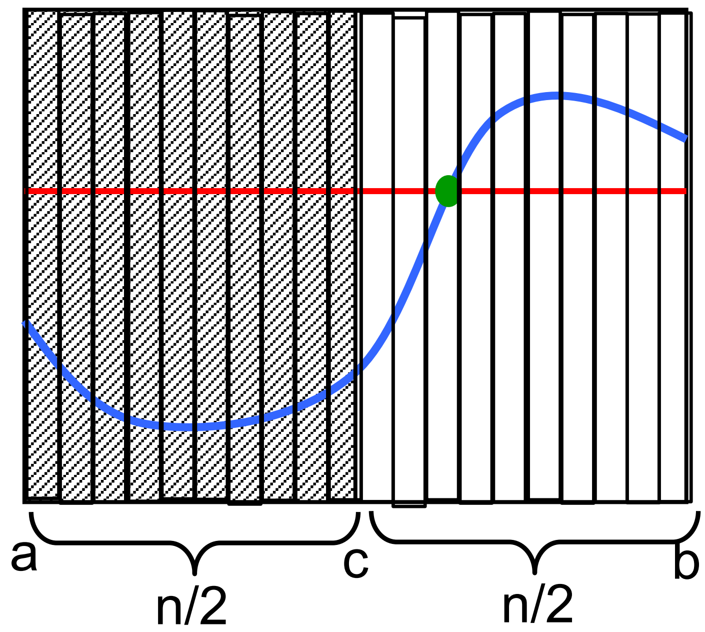
Always half of the intervals lie to the left and half lie to the right of c

$= T(n/8) + c + c + c$

So, $= T(n/4) + c + c$

$$T(n) = T\left(\frac{n}{2}\right) + \underbrace{O(1)}_{c}$$

i.e., $T(n) = O(\log n) = O(\log \frac{a-b}{\epsilon})$



$a = 0$ $b = 1$ $\varepsilon = 2^{-1000}$

Recurrences

Above: Where they come from, how to find them

Next: how to solve them

Master Theorem

Suppose $T(n) = a\,T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $a < b^k$ then $T(n) = \Theta\left(n^k\right)$
- If $a = b^k$ then $T(n) = \Theta\left(n^k \log n\right)$

Works even if it is $\left\lceil \frac{n}{b} \right\rceil$ instead of $\frac{n}{b}$.

We also need $a \geq 1, b > 1\,, k \geq 0$ and $T(n) = O(1)$ for $n \leq b$.

Master Theorem

Suppose $T(n) = a\,T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $a < b^k$ then $T(n) = \Theta\left(n^k\right)$
- If $a = b^k$ then $T(n) = \Theta\left(n^k \log n\right)$

Example: For mergesort algorithm we have

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

So, $k = 1, a = b^k$ and $T(n) = \Theta(n \log n)$

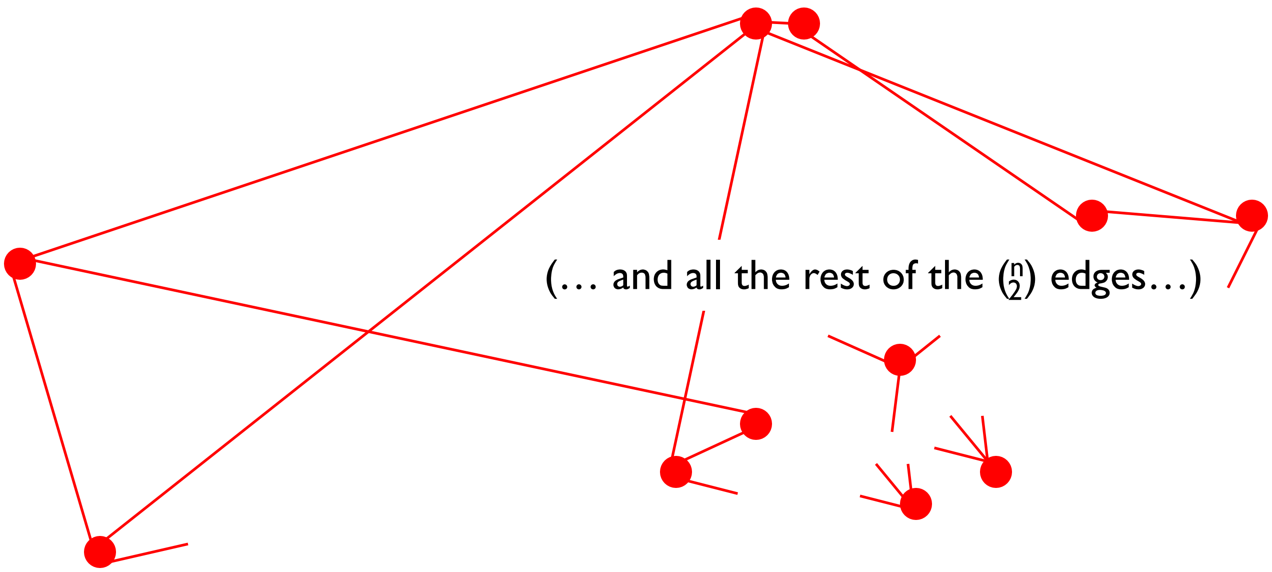
$T(n) = 4\,T(n/2) + O(n)$ $\quad a = 4,\ b = 2,\ k = 1$

$\log_2 4 = 2 \quad T(n) = \Theta(n^2).$

Finding the Closest Pair of Points

Closest Pair of Points (non geometric)

Given n points and arbitrary distances between them, find the closest pair. (E.g., think of distance as airfare – definitely not Euclidean distance!)



Must look at all n choose 2 pairwise distances, else any one you didn't check might be the shortest.

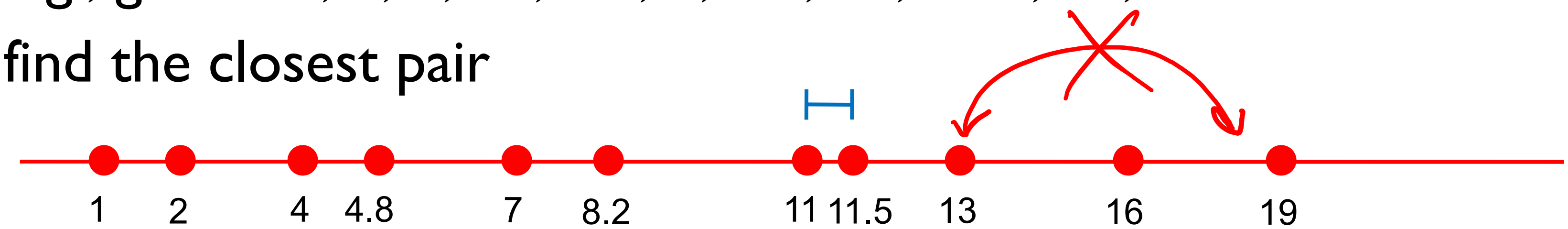
i.e., you have to read the whole input

Closest Pair of Points (1-dimension)

Given n points on the real line, find the closest pair,

e.g., given 11, 2, 4, 19, 4.8, 7, 8.2, 16, 11.5, 13, 1

find the closest pair



Fact: Closest pair is adjacent in ordered list

So, first sort, then scan adjacent pairs.

Time O(n log n) to sort, if needed, Plus O(n) to scan adjacent pairs

Key point: do *not* need to calc distances between all pairs: exploit geometry + ordering

Closest Pair of Points (2-dimensions)

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

(1, 2)   (4, 5)   (1.5, -3) . . .

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force: Check all pairs of points p and q with $\Theta(n^2)$ time.

Assumption: No two points have same x coordinate.

Closest Pair of Points (2-dimensions)

Given n points in the plane, find a pair with 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.

Brute force: Check all pairs of points p and q with $\Theta(n^2)$ time.

Assumption: No two points have same x coordinate.

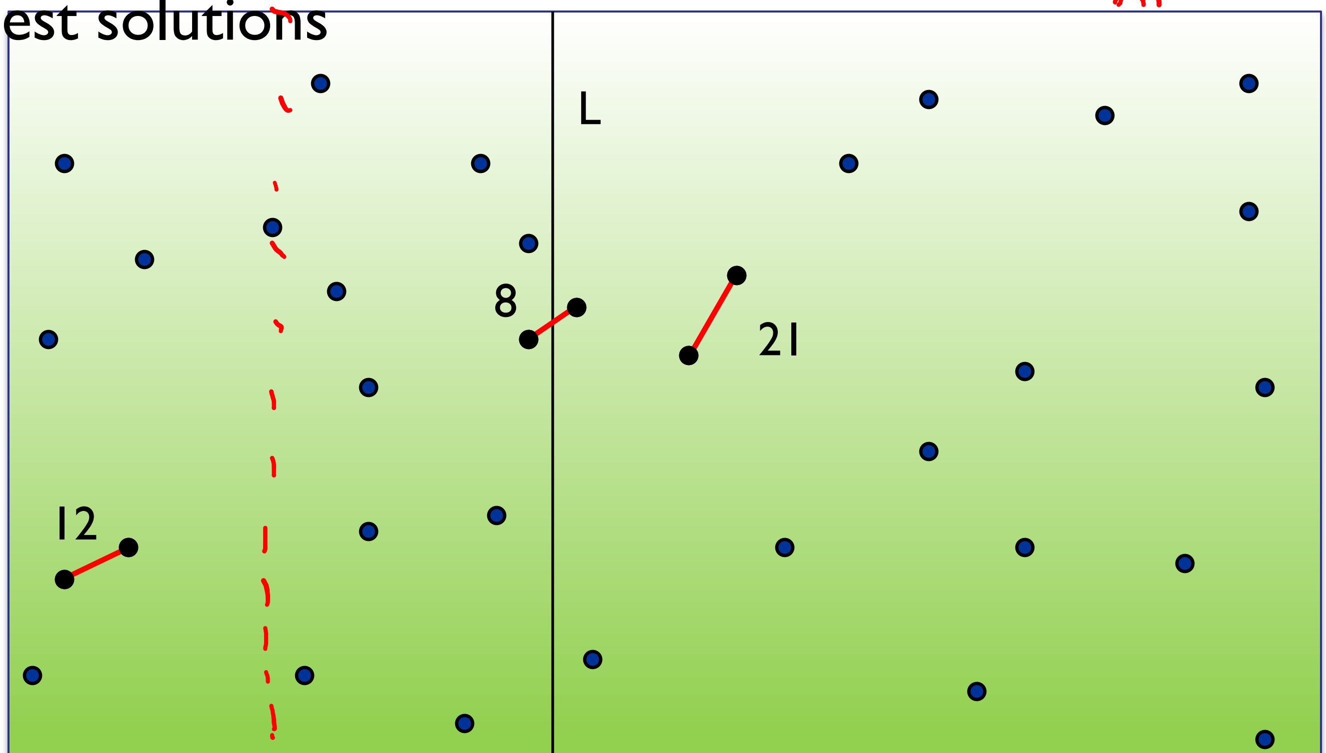
A Divide and Conquer Alg

Divide: draw vertical line L with ≈ **n/2 points on each side.**

Conquer: find closest pair on each side, recursively.

Combine to find closest pair overall ← seems like $\Theta(n^2)$?

Return best solutions



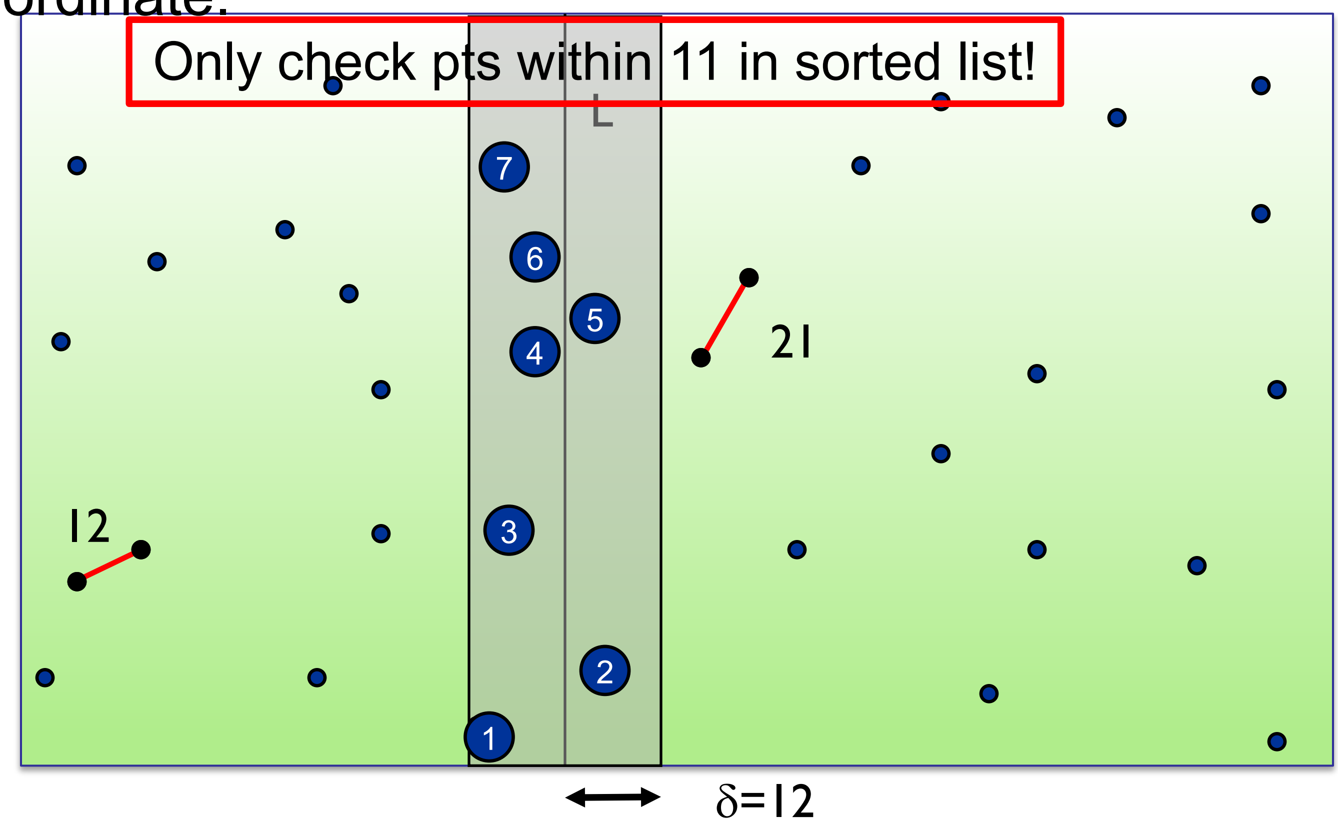
Key Observation

Suppose δ is the minimum distance of all pairs in left/right of L.

$$\delta = \min(12{,}21) = 12.$$

Key Observation: suffices to consider points within δ of line L.

Almost the one-D problem again: Sort points in 2δ-strip by their y coordinate.



Almost 1D Problem

Partition each side of L into $\frac{\delta}{2} \times \frac{\delta}{2}$ squares

Claim: No two points lie in the same $\frac{\delta}{2} \times \frac{\delta}{2}$ box.

Pf: Such points would be within

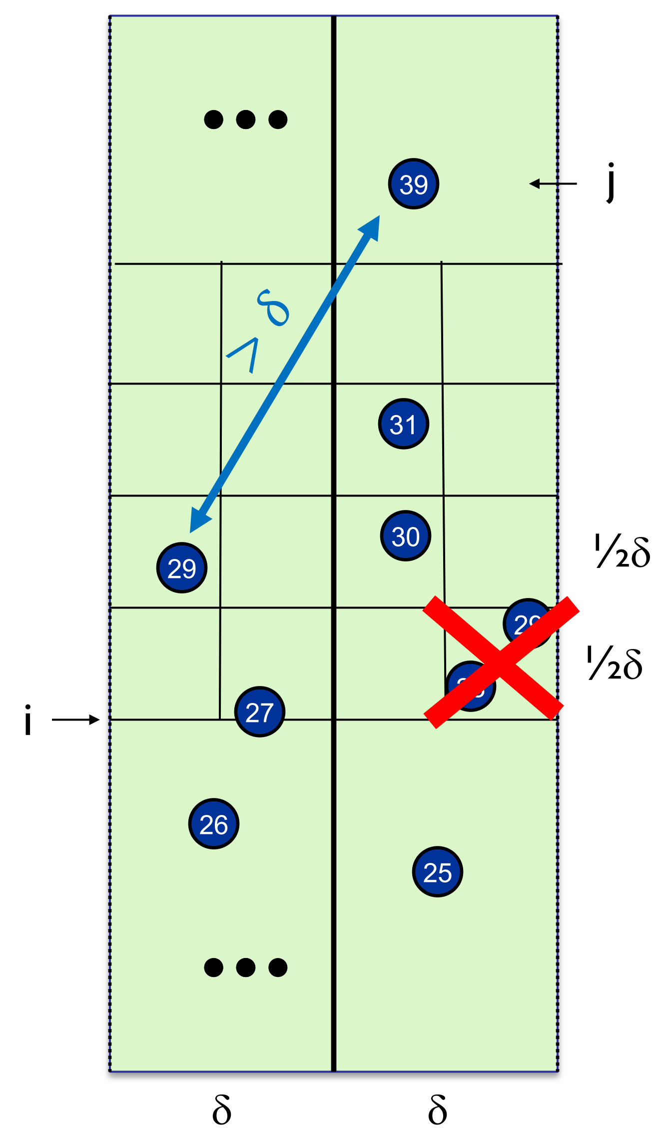
$$\sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \delta\sqrt{\frac{1}{2}} \approx 0.7\delta < \delta$$

But this contradicts IH.

Let s_i have the i^{th} smallest y-coordinate among points in the 2δ-width-strip.

Claim: If $|i - j| > 11$, then the distance between s_i and s_j is $> \delta$.

Pf: only 11 boxes within δ of $y(s_i)$.



Closest Pair (2Dim Algorithm)

```
Closest-Pair(p1, …, pn) {
   if(n <= ??) return ??

   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L

   Sort remaining points p[1]…p[m] by y-coordinate.

   for i = 1..m
      for k = 1…11
        if i+k <= m
           δ = min(δ, distance(p[i], p[i+k]));

   return δ.
}
```

Closest Pair Analysis I

Let D(n) be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on n ≥ 1 points

$$D(n) \le \begin{cases} 1 & \text{if } n = 1 \\ 2D\left(\frac{n}{2}\right) + 11\,n & \text{o.w.} \end{cases} \Rightarrow D(n) = \mathrm{O}(n \log n)$$

BUT, that's only the number of *distance calculations*

What if we counted running time?

$$T(n) \le \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n \log n) & \text{o.w.} \end{cases} \Rightarrow D(n) = \mathrm{O}(n \log^2 n)$$

Can we do better? (Analysis II)

Yes!!

Don't sort by y-coordinates each time.

Sort by x at top level only.

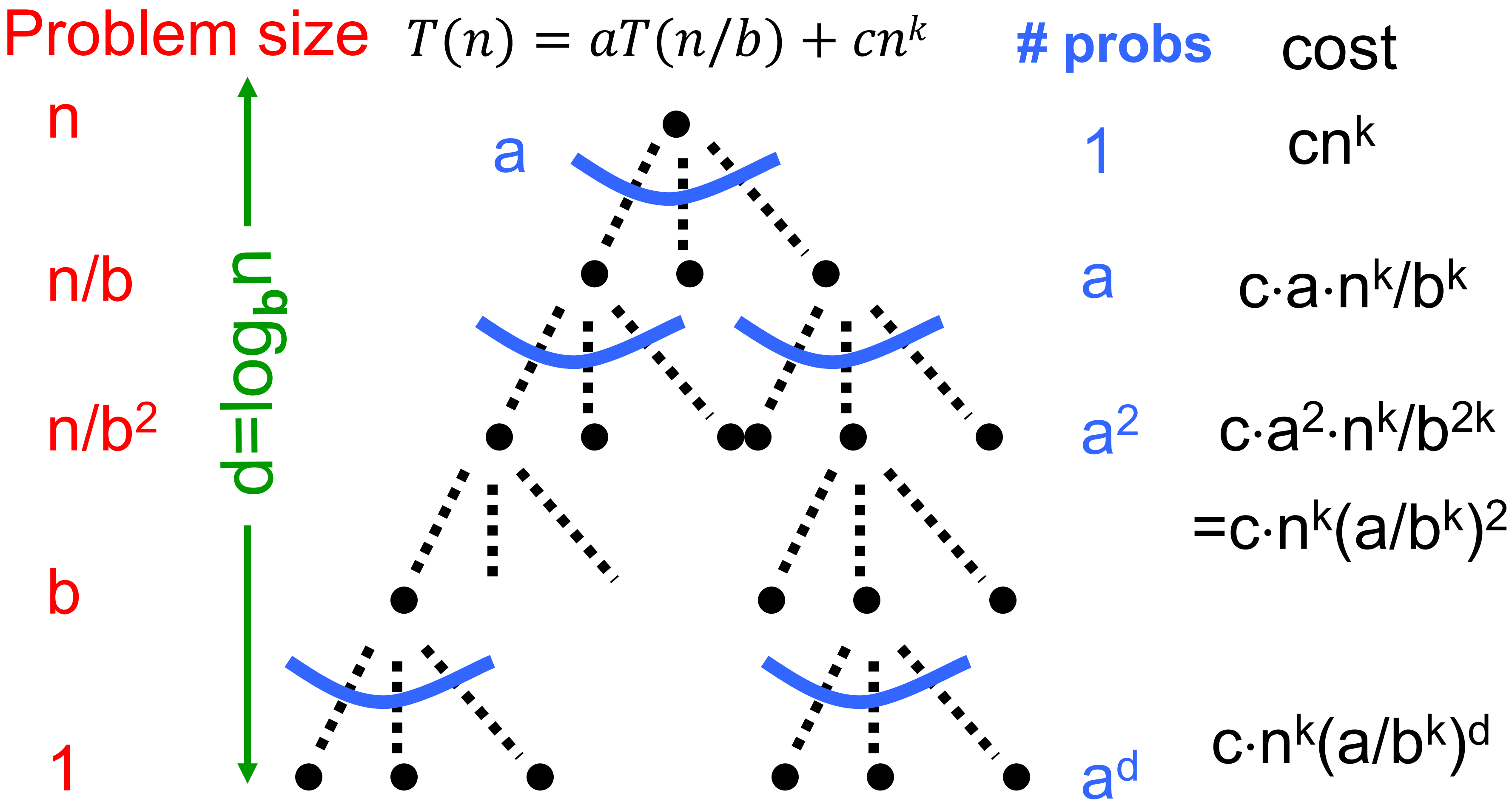
This is enough to divide into two equal subproblems in O(n)

Each recursive call returns δ and list of all points sorted by y

Sort points by y-coordinate by merging two pre-sorted lists.

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & \text{o. w.} \end{cases} \Rightarrow D(n) = O(n \log n)$$

Proving Master Theorem



$$T(n) = cn^k \sum_{i=0}^{d=\log_b n} \left(\frac{a}{b^k}\right)^i$$

A Useful Identity

Theorem: $1 + x + x^2 + \cdots + x^d = \frac{x^{d+1}-1}{x-1}$

Pf: Let $S = 1 + x + x^2 + \cdots + x^d$

Then, $xS = x + x^2 + \cdots + x^{d+1}$

So, $xS - S = x^{d+1} - 1$
i.e., $S(x-1) = x^{d+1} - 1$
Therefore,

$$S = \frac{x^{d+1} - 1}{x - 1}$$

Solve: $T(n) = aT\left(\frac{n}{b}\right) + cn^k,\ a > b^k$

$$T(n) = cn^k \sum_{i=0}^{\log_b n} \left(\frac{a}{b^k}\right)^i$$

$\frac{x^{d+1}-1}{x-1}$ for $x = \frac{a}{b^k}$

$d = \log_b n$

using $x \neq 1$

$$= cn^k \frac{\left(\frac{a}{b^k}\right)^{\log_b n+1} - 1}{\left(\frac{a}{b^k}\right) - 1}$$

$b^{k \log_b n}$

$= \left(b^{\log_b n}\right)^k$

$= n^k$

$$\leq c\left(\frac{n^k}{b^{k \log_b n}}\right) \frac{\left(\frac{a}{b^k}\right)}{\left(\frac{a}{b^k}\right) - 1} a^{\log_b n}$$

$a^{\log_b n}$

$= \left(b^{\log_b a}\right)^{\log_b n}$

$= \left(b^{\log_b n}\right)^{\log_b a}$

$= n^{\log_b a}$

$$\leq 2c\ a^{\log_b n} = O\left(n^{\log_b a}\right)$$

Solve: $T(n) = aT\left(\frac{n}{b}\right) + cn^k,\ a = b^k$

$$T(n) = cn^k \sum_{i=0}^{\log_b n} \left(\frac{a}{b^k}\right)^i$$

$$= cn^k \log_b n$$

Master Theorem

Suppose $T(n) = a\,T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $a < b^k$ then $T(n) = \Theta\left(n^k\right)$
- If $a = b^k$ then $T(n) = \Theta\left(n^k \log n\right)$

Works even if it is $\left\lceil\frac{n}{b}\right\rceil$ instead of $\frac{n}{b}$.

We also need $a \geq 1, b > 1, k \geq 0$ and $T(n) = O(1)$ for $n \leq b$.