CSE 421
Introduction to Algorithms

Lecture 8: Divide and Conquer

Algorithm Design Techniques

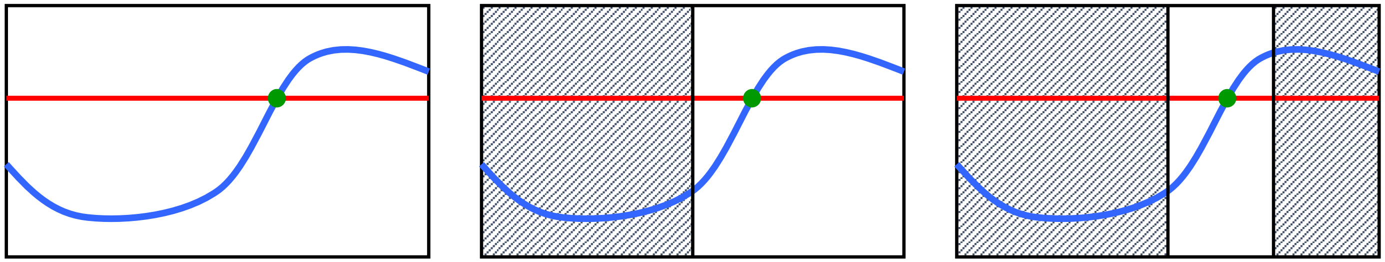
Divide & Conquer

- Divide instance into subparts.
- Solve the parts recursively.
- Conquer by combining the answers

To truly fit Divide & Conquer

- each sub-part should be at most a constant fraction of the size of the original input instance
  - e.g. Mergesort, Binary Search, Quicksort (sort of), etc.

Binary search for roots (bisection method)



Given:

- Continuous function $\boldsymbol{f}$ and two points $\boldsymbol{a < b}$ with $\boldsymbol{f(a) \leq 0}$ and $\boldsymbol{f(b) > 0}$

Find:

- Approximation within $\boldsymbol{\varepsilon}$ of $\boldsymbol{c}$ s.t. $\boldsymbol{f(c) = 0}$ and $\boldsymbol{a < c < b}$

Bisection method

Bisection($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{\varepsilon}$)

if $(\boldsymbol{b}-\boldsymbol{a}) \leq \boldsymbol{\varepsilon}$ then

return($\boldsymbol{a}$)

else {

$\boldsymbol{c} \leftarrow (\boldsymbol{a}+\boldsymbol{b})/\mathbf{2}$

if $\boldsymbol{f}(\boldsymbol{c}) \leq \mathbf{0}$ then

return(Bisection($\boldsymbol{c}$, $\boldsymbol{b}$, $\boldsymbol{\varepsilon}$))

else

return(Bisection($\boldsymbol{a}$, $\boldsymbol{c}$, $\boldsymbol{\varepsilon}$))

}

Time Analysis

At each step we halved the size of the interval

- It started at size $\boldsymbol{b} - \boldsymbol{a}$
- It ended at size $\boldsymbol{\varepsilon}$

So # of calls to $\boldsymbol{f}$ is $\log_2 ((\boldsymbol{b} - \boldsymbol{a})/\boldsymbol{\varepsilon})$

Old Favorites

Binary search:

- One subproblem of half size plus one comparison
- Recurrence* for time in terms of # of comparisons
  - $T(n) = T(n/2) + 1$ for $n \geq 2$
  - $T(1) = 0$
- Solving shows that $T(n) = \lceil \log_2 n \rceil + 1$

Mergesort:

- Two subproblems of half size plus merge cost of $n - 1$ comparisons
- Recurrence* for time in terms of # of comparisons
  - $T(n) \leq 2T(n/2) + n - 1$ for $n \geq 2$
  - $T(1) = 0$
- Roughly n comparisons at each of $\log_2 n$ levels of recursion so $T(n)$ is roughly $n \log_2 n$

*We will implicitly assume that every input to $T(\cdot)$ is rounded up to the nearest integer.

Euclidean Closest Pair

Given:

- A sequence of $\boldsymbol{n}$ points $\boldsymbol{p_1, \dots, p_n}$ with real coordinates in $\boldsymbol{d}$ dimensions ($\mathbb{R}^{\boldsymbol{d}}$)

Find:

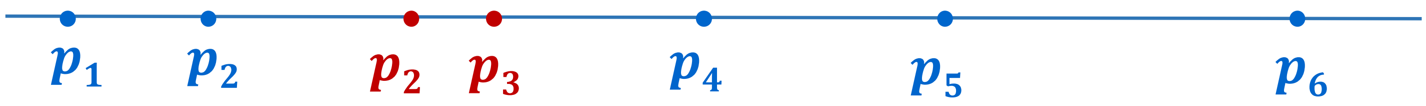
- A pair of points $\boldsymbol{p_i, p_j}$ s.t. the Euclidean distance $\boldsymbol{d(p_i, p_j)}$ is minimized

What is the first algorithm you can think of?

- Try all $\Theta(\boldsymbol{n^2})$ possible pairs

Can we do better if dimension $\boldsymbol{d = 1}$?

Closest Pair in 1 Dimension



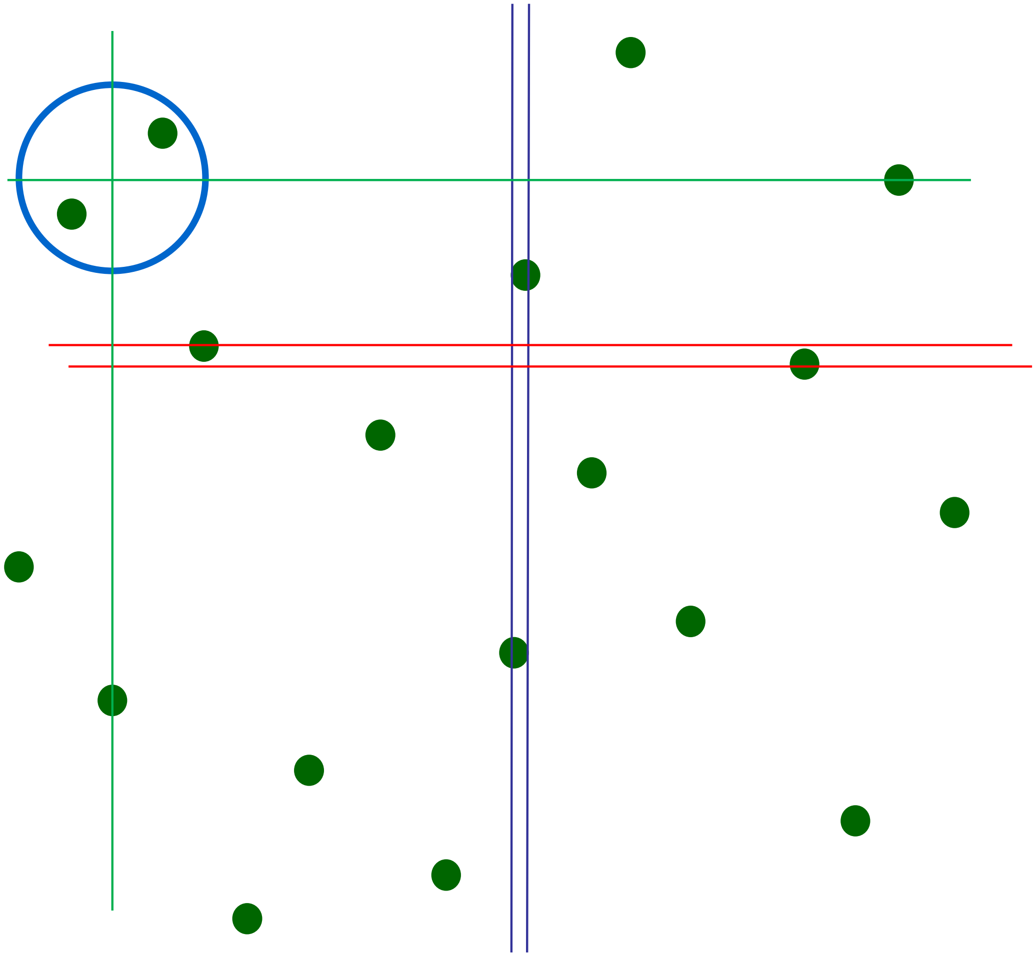
Algorithm:

- Sort points so $p_1 \leq p_2 \leq \cdots \leq p_n$
- Find closest adjacent pair p_i, p_{i+1}.

Running time: $O(n \log n)$

What about $d = 2$?

Closest Pair in 2 Dimensions



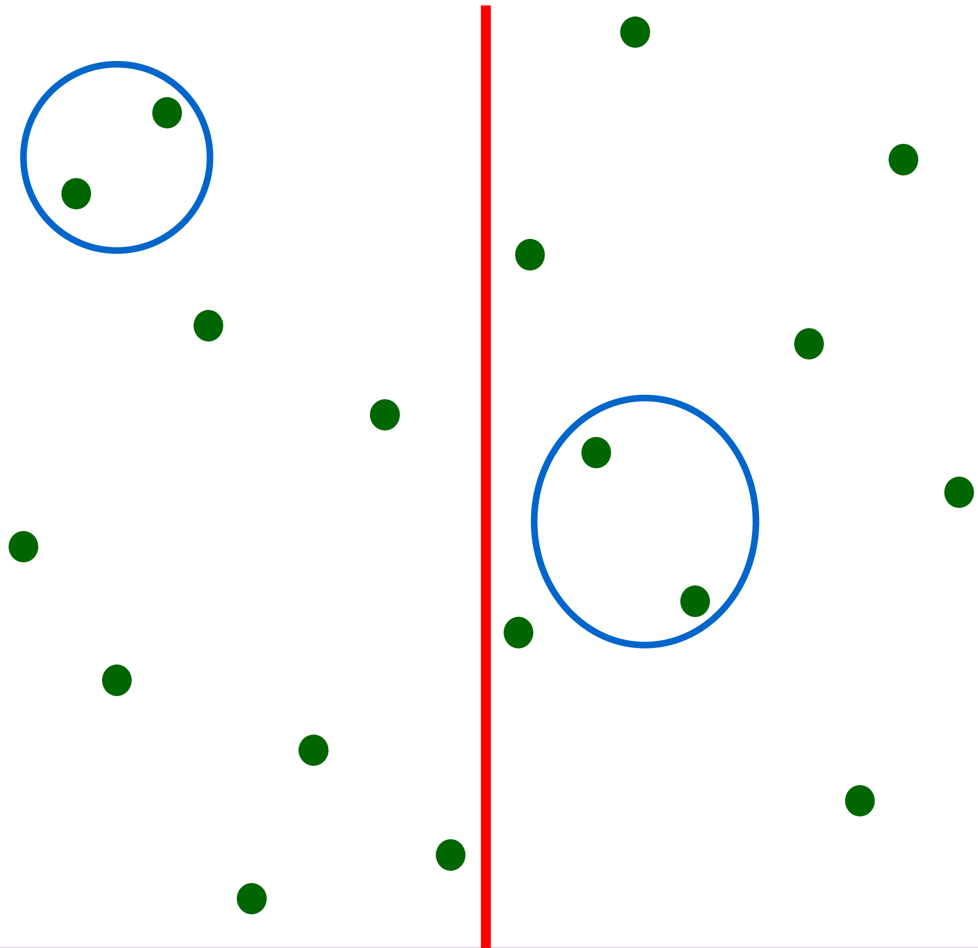
Sorting on 1st coordinate doesn't work

No single direction to sort points to guarantee success!

Let's try divide & conquer…

How might we divide the points so that each subpart is a constant factor smaller?

Closest Pair in 2 Dimensions: Divide and Conquer



How might we divide the points so that each subpart is a constant factor smaller?

Split using median x-coordinate!

- each subpart has size $n/2$.

Conquer:

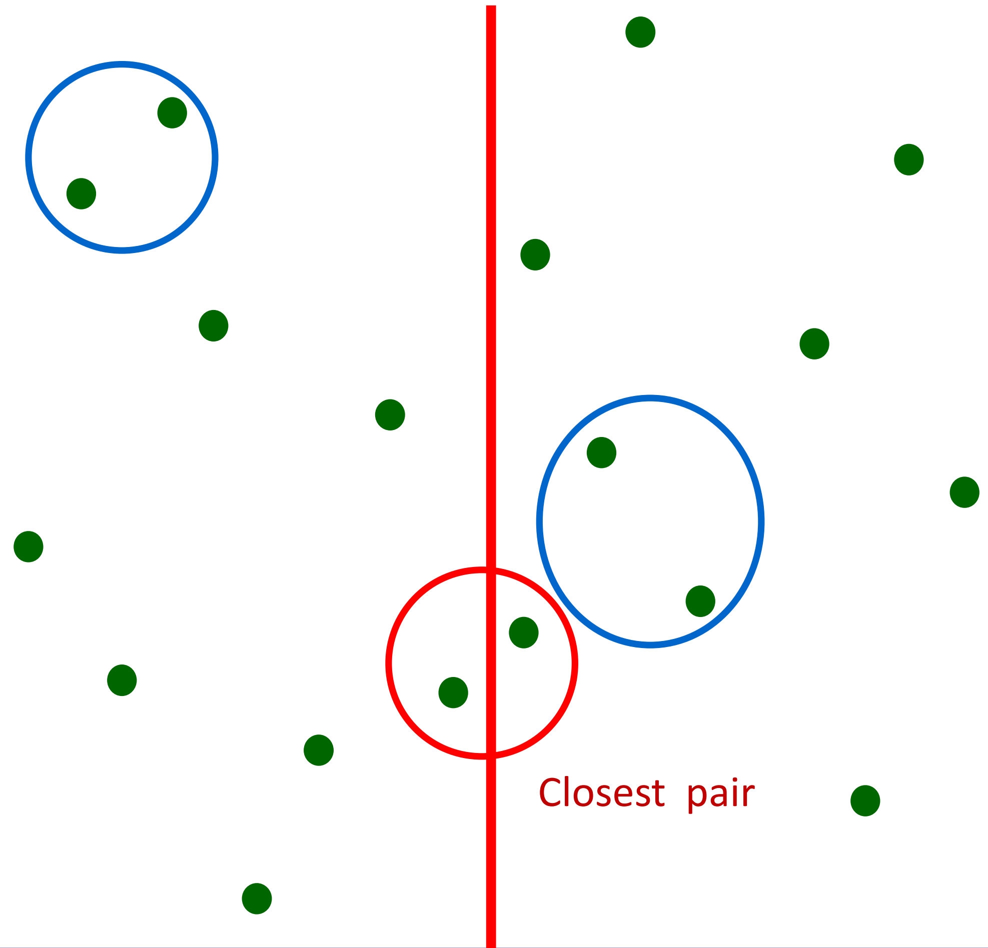
- Solve both size $n/2$ subproblems recursively

Recombine to get overall answer?

Take the closer of the two answers?

- works here but....

Closest Pair in 2 Dimensions: Divide and Conquer



How might we divide the points so that each subpart is a constant factor smaller?

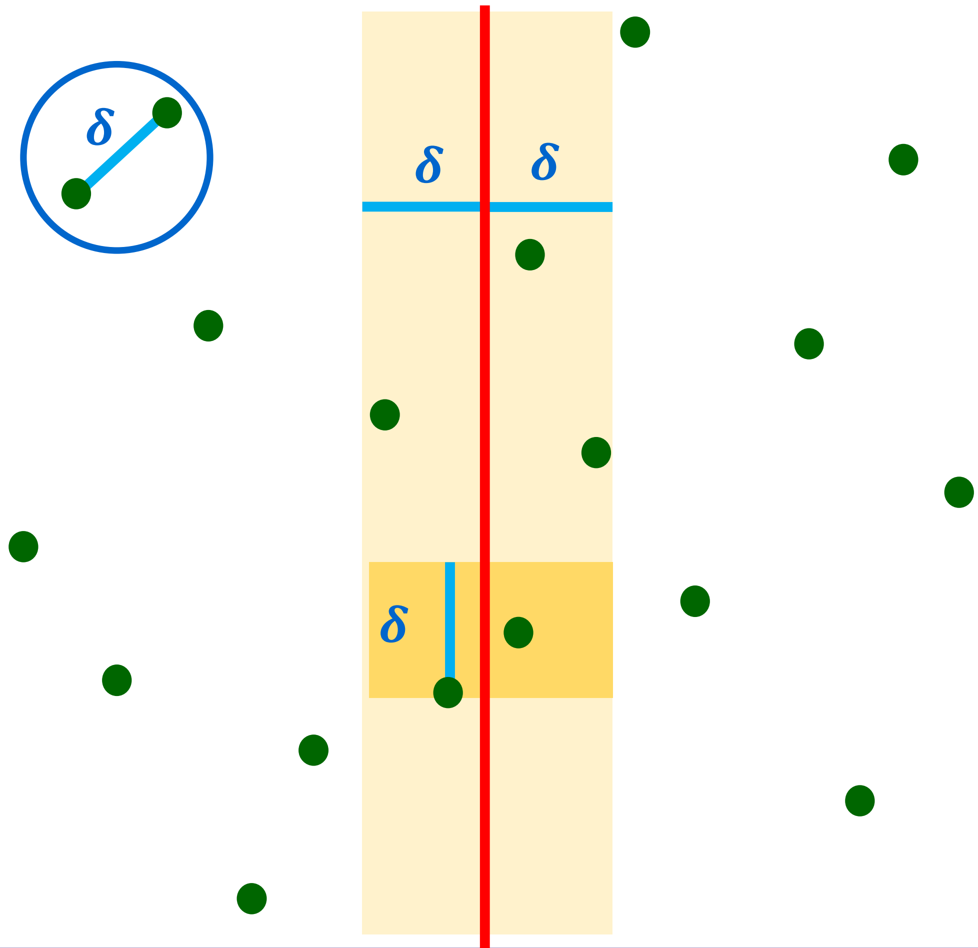
Split using median x-coordinate!
- each subpart has size $n/2$.

Conquer:
- Solve both size $n/2$ subproblems recursively

Recombine to get overall answer?

Take the closer of the two answers?
- …but not always!

Closest Pair in 2 Dimensions: Divide and Conquer



Need to worry about pairs across the split!

New idea to handle them

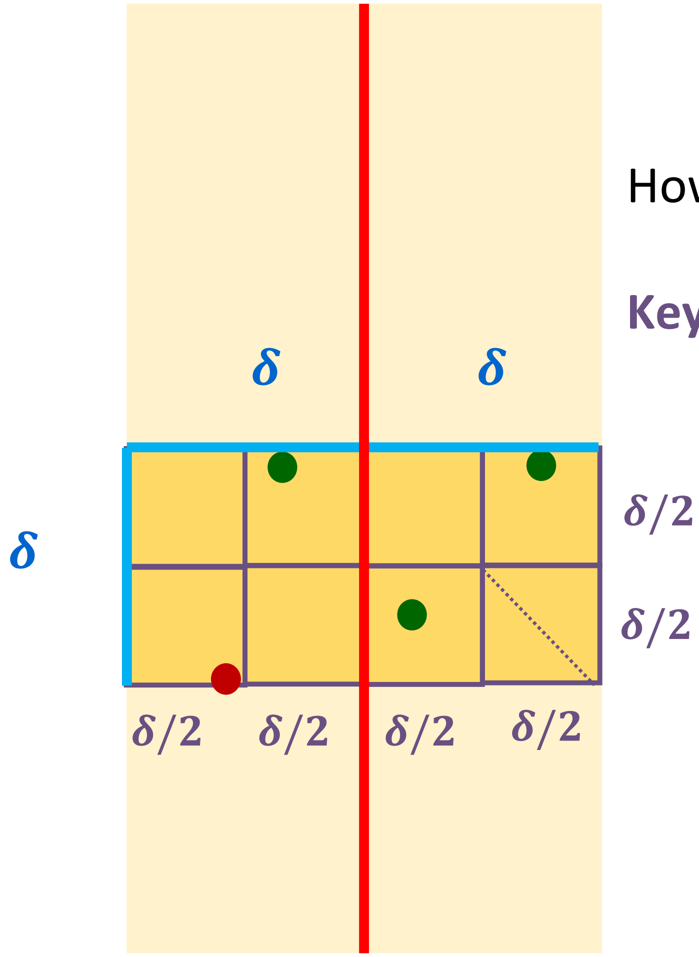
- Let δ be the distance of the closest pair in the 2 subparts
- This pair is a candidate
- Only need to check width δ band either side of the median

Within that band …

- only need to compare each point with the other points in the rectangle of height δ above it.

How many points can that be?

Closest Pair in 2 Dimensions: Divide and Conquer



How many points can there be in that δ by 2δ rectangle?

Key idea: We know that no pair on either side is closer than δ apart so there can't be too many!

- Each of the 8 squares of side $\delta/2$ can contain at most 1 point!
  - Because diagonal has length $< \delta$
- So….only need to compare each point with the next 7 points above it to guarantee you'll find a partner closer than δ in the rectangle if there is one!

Closest Pair in 2 Dimensions: Divide and Conquer

Fleshing out the algorithm:

Divide:

- At top level we need median $\boldsymbol{x}$ coordinate to split points
- At next level down we'll need median $\boldsymbol{x}$ coordinate for each side
- Might as well sort all points by $\boldsymbol{x}$ coordinate up front to get all medians at once!

$O(\boldsymbol{n}\log\boldsymbol{n})$ total over all calls

Conquer: Solve the two sub-problems to get two candidate pairs $\mathbf{2}\,\boldsymbol{T}(\boldsymbol{n}/\mathbf{2})$

Recombine:

- Choose closer candidate pair and let its distance be $\boldsymbol{\delta}$ $O(\mathbf{1})$
- Select $\boldsymbol{B}$ = all points in band with $\boldsymbol{x}$ coordinates within $\boldsymbol{\delta}$ of median $O(\boldsymbol{n})$
- Sort $\boldsymbol{B}$ by $\boldsymbol{y}$ coordinate May involve repeated work for different calls $O(\boldsymbol{n}\log\boldsymbol{n})$
- Compare each point in $\boldsymbol{B}$ with next $\mathbf{7}$ points and update if closer pair found. $O(\boldsymbol{n})$

Closest Pair in 2 Dimensions: Divide and Conquer

Fleshing out the algorithm: A better version:

Preprocess: Compute sorted list $\boldsymbol{X}$ of points by $\boldsymbol{x}$ coordinate $O(\boldsymbol{n} \log \boldsymbol{n})$

- Subparts will be defined by two indices into this list

Compute sorted list $\boldsymbol{Y}$ of points by $\boldsymbol{y}$ coordinate $O(\boldsymbol{n} \log \boldsymbol{n})$

Divide: Use median in $\boldsymbol{X}$ to get $\boldsymbol{X_L}$ and $\boldsymbol{X_R}$ and filter points of $\boldsymbol{Y}$ to produce sorted sublists $\boldsymbol{Y_L}$ and $\boldsymbol{Y_R}$ $O(\boldsymbol{n})$

Conquer: Solve the two sub-problems to get two candidate pairs $\boldsymbol{2\,T(n/2)}$

Recombine:

- Choose closer candidate pair and let its distance be $\boldsymbol{\delta}$ $O(\boldsymbol{1})$
- Filter $\boldsymbol{Y}$ to get $\boldsymbol{B}$ = points in band w/ $\boldsymbol{x}$ coordinates within $\boldsymbol{\delta}$ of median $O(\boldsymbol{n})$
- Compare each point in $\boldsymbol{B}$ with next $\boldsymbol{7}$ points and update if closer pair found. $O(\boldsymbol{n})$

Closest Pair in 2 Dimensions: Divide and Conquer

Total runtime = Preprocessing time + Divide and Conquer time

Let $T(n)$ be Divide and Conquer time:

Recurrence:

- $T(n) \leq 2\,T(n/2) + O(n)$ for $n \geq 3$
- $T(2) = 1$

Solution: $T(n)$ is $O(n \log n)$.

With preprocessing, total runtime is $O(n \log n)$.

Sometimes two sub-problems aren't enough

More general divide and conquer

- You've broken the problem into a different sub-problems
- Each has size at most n/b
- The cost of break-up and recombining sub-problem solutions is $O(n^k)$
  - "cost at the top level"

Recurrence

- $T(n) = a \cdot T(n/b) + O(n^k)$ for $n \geq b$
- T is constant for inputs $< b$.
  - For solutions correct up to constant factors no need for exact base case

Solving Divide and Conquer Recurrence

Master Theorem: Suppose that $T(n) = a \cdot T(n/b) + O(n^k)$ for $n > b$.

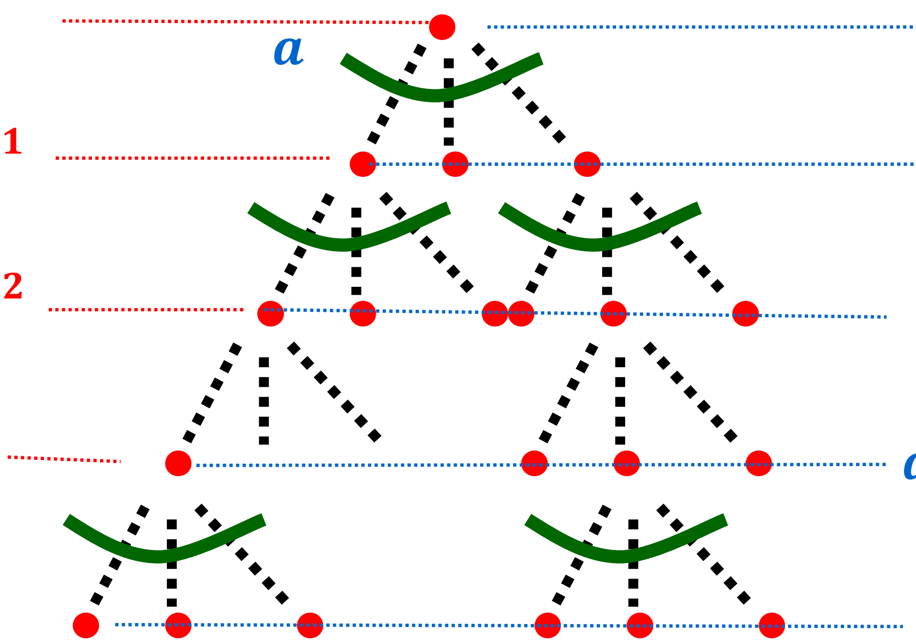
- If $a < b^k$ then $T(n)$ is $O(n^k)$
  - Cost is dominated by work at top level of recursion
- If $a = b^k$ then $T(n)$ is $O(n^k \log n)$
  - Total cost is the same for all $\log_b n$ levels of recursion
- If $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$
  - Note that $\log_b a > k$ in this case
  - Cost is dominated by total work at lowest level of recursion

Binary search: $a = 1$, $b = 2$, $k = 0$ so $a = b^k$: Solution: $O\left(n^0 \log n\right) = O(\log n)$

Mergesort: $a = 2$, $b = 2$, $k = 1$ so $a = b^k$: Solution: $O(n^1 \log n) = O(n \log n)$

Proving Master Theorem for $T(n) = a \cdot T(n/b) + c \cdot n^k$

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



problem size	# of problems	level work/problem
$n \le b^d$	1	n^k
$n/b \le b^{d-1}$	a	n^k/b^k
$n/b^2 \le b^{d-2}$	a^2	n^k/b^{2k}
…	…	…
$n/b^{d-1} \le b$	a^{d-1}	b^k
$n/b^d \le 1$	a^d	1

Proving Master Theorem for $T(n) = a \cdot T(n/b) + c \cdot n^k$

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

# of problems	level work/problem	total work/level
1	n^k	n^k
a	n^k/b^k	$(a/b^k) \cdot n^k$
a^2	n^k/b^{2k}	$(a/b^k)^2 \cdot n^k$
…	…	…
a^{d-1}	b^k	…
a^d	1	$a^{\log_b n}$

total work

If $a < b^k$ sum of geometric series with biggest term $O(n^k)$

If $a = b^k$ sum of $O(\log n)$ terms each $O(n^k)$

If $a > b^k$ sum of geometric series with biggest term $O(a^{\log_b n})$

Claim: $a^{\log_b n} = n^{\log_b a}$

Proof: Take $\log_b$ of both sides ∎