Goal: given integers X and Y, each of n bits, compute $X \times Y$.

- Grade school algorithm is $O(n^2)$.
- Algorithm of al-Khwarizmi depends
- But if brute force is polynomial, ...

Example

$156 = 1 \cdot 10^2 + 56$

If $X = 156 = 10011100$ and $Y = 225 = 11100001$, then:

X_H	X_L	Y_H	Y_L
1001	1100	1110	0001
$(9)_{10}$	$(12)_{10}$	$(14)_{10}$	$(1)_{10}$

$9 \cdot 16 + 12$ $\quad\quad 14 \cdot 16 + 1$

$\cancel{144 + 12}$

156

$(\overset{X_H}{9} \cdot 16 + 12)(\overset{Y_H}{14} \cdot 16 + 1) = (\overset{X_H}{9} \cdot \overset{Y_H}{14}) \cdot 16^2 + (12 \cdot 14 \cdot 16 + (9 \cdot 16 \cdot 1) + 12 \cdot 1$

First Divide and Conquer

$$
\begin{aligned}
X \times Y &= (X_H \times 2^{n/2} + X_L) \times (Y_H \times 2^{n/2} + Y_L) \\
&= X_H \cdot Y_H \times 2^n + (X_H Y_L + X_L Y_H) \times 2^{n/2} + X_L Y_L
\end{aligned}
$$

mult(X, Y):

Divide X and Y into X_H etc.
$A \leftarrow$ mult(X_H, Y_H)
$B \leftarrow$ mult(X_H, Y_L) + mult(X_L, Y_H)
$C \leftarrow$ mult(X_L, Y_L)
return $A \times 2^n + B \times 2^{n/2} + C$

$T(n) = \underline{4}\, T(\frac{n}{2}) + O(n)$
$= O(n^2)$

- Express this running time as a recurrence.

Second Divide and Conquer

- $T(n) = 4T(n/2) + O(n)$. How can we do better?
  - Would it help to reduce the $O(n)$ to $O(1)$? NO

$$X \times Y = (X_H \times 2^{n/2} + X_L) \times (Y_H \times 2^{n/2} + Y_L)$$
$$= \underbrace{X_H \cdot Y_H}_{} \times 2^n + \underline{(X_H Y_L + X_L Y_H)} \times 2^{n/2} + \underbrace{X_L Y_L}_{}$$

one call

$$(X_H + X_L)(Y_H + Y_L) = \cancel{X_H Y_H} + X_H Y_L + X_L Y_H + \cancel{X_L Y_L}$$

As an algorithm...

break into X_H etc $\Big\}\ O(n)$

$A \leftarrow X_H \cdot Y_H$

$C \leftarrow X_L Y_L$ $\Big\}\ 2T(n/2)$

$B_{temp} \leftarrow (X_H + X_L)(Y_H + Y_L)$ $\Big\}\ T(\frac{n}{2}) + O(n)$

$B \leftarrow B_{temp} - A - C$ $\Big\}\ O(n)$

return $A \cdot 2^n + B \cdot 2^{n/2} + C$

$T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\log_2 3})$

There is a similar approach for Matrix Multiplication...

- To multiply two $n \times n$ matrices:
  $\begin{bmatrix} I & J \\ K & L \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}$
- You saw this divide and conquer algorithm in linear algebra.
  - $I = AE + BG$
  - $J = AF + BH$
  - $K = CE + DG$
  - $L = CF + DH$
- That is $T(n) = 8T(n/2) + O(n^2) \longrightarrow O(n^3)$
- Strassen discovered a way to do this with only 7 recursive calls
- So $T(n) = 7T(n/2) + O(n^2)$, for $O(n^{\log_2 7})$ time.
- Best I know of is $O(n^{2.376})$

Brute Force

`Closest-Pair`

Input: n points in $2D$-space

Output: The pair that has the smallest distance between them.

min $= \infty$
for $i = 2 \to n$ **do**
  for $j = 1 \to i - 1$ **do**
    if $(x_j - x_i)^2 + (y_j - y_i)^2 <$ min **then**
      min $= (x_j - x_i)^2 + (y_j - y_i)^2$
      closestPair $= ((x_i, y_i), (x_j, y_j))$
return closestPair

Starting Divide and Conquer

L R

First, sort P by x-coordinate, and then call:
Closest-Pair(P)

Let $L = P[1 \ldots n/2]$
Let $R = P[n/2 + 1 \ldots n]$
$(\delta_l, l_1, l_2) \leftarrow$ Closest-Pair(L)
$(\delta_r, r_1, r_2) \leftarrow$ Closest-Pair(R)
if $\delta_l \leq \delta_r$ **then**
  return (δ_l, l_1, l_2) // closest pair from L
else
  return (δ_r, r_1, r_2) // closest pair from R

Starting Divide and Conquer

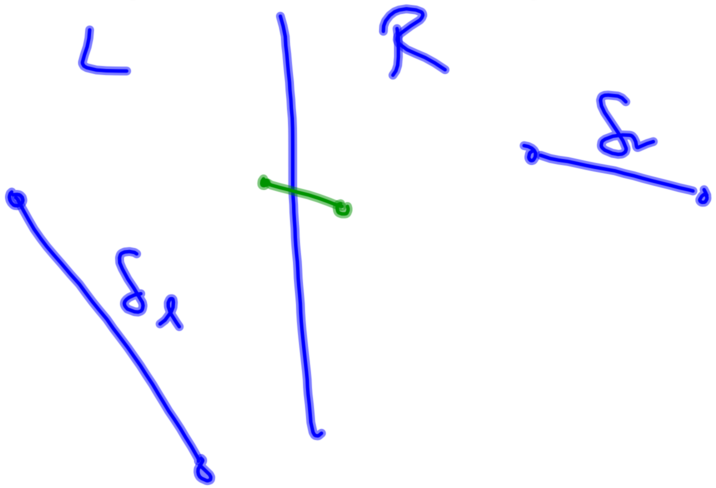
- What is the running time of the previous algorithm?

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

or $+\,O(n) \rightarrow O(n \lg n)$

Starting Divide and Conquer

- What is wrong with the previous algorithm?



Better Divide and Conquer

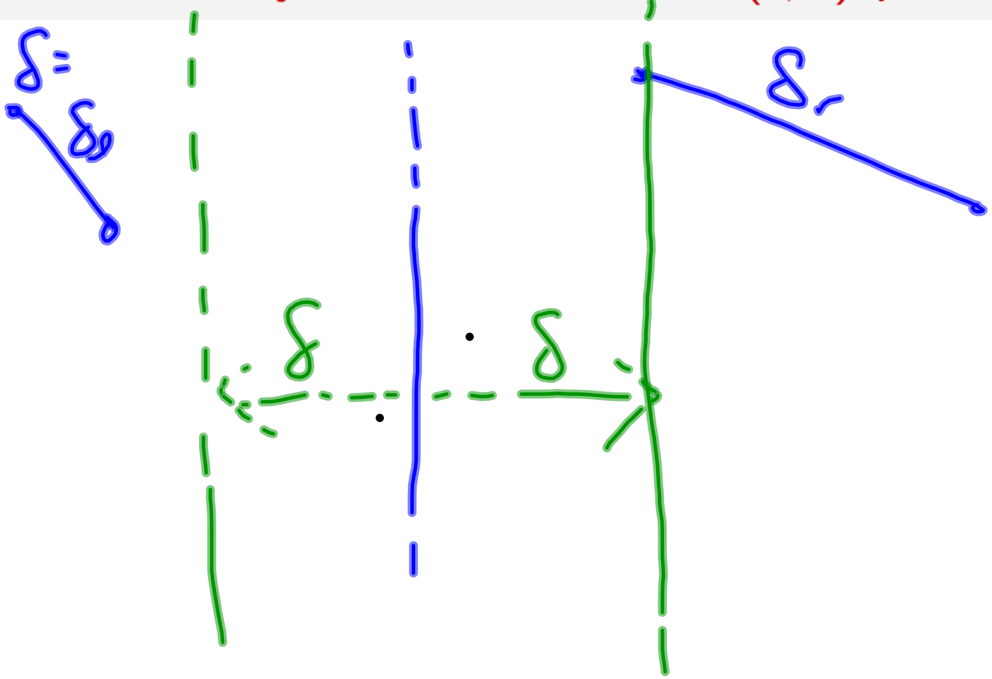
. . .
$(\delta_l, l_1, l_2) \leftarrow$ Closest-Pair(L)
$(\delta_r, r_1, r_2) \leftarrow$ Closest-Pair(R)
if $\delta_l \leq \delta_r$ **then**
    $(\delta, p_1, p_2) \leftarrow (\delta_l, l_1, l_2)$ // $\delta =$ smallest distance found so far
else
    $(\delta, p_1, p_2) \leftarrow (\delta_r, r_1, r_2)$
for all $p_l \in L$ **do**
    for all $p_r \in R$ **do**
        if $d(p_l, p_r) < \delta$ **then**
            $(\delta, p_1, p_2) \leftarrow (d(p_l, p_r), p_l, p_r)$
return (δ, p_1, p_2)

What can we improve?

- Goal: get this algorithm better than $O(n^2)$ (and be correct)
- $T(n) = 2T(n/2) + O(n^2)$

↑ big factor in runtime

Do we really need to check all (l, r) pairs? No.



How many points within δ?

· all n points?

So only check those...

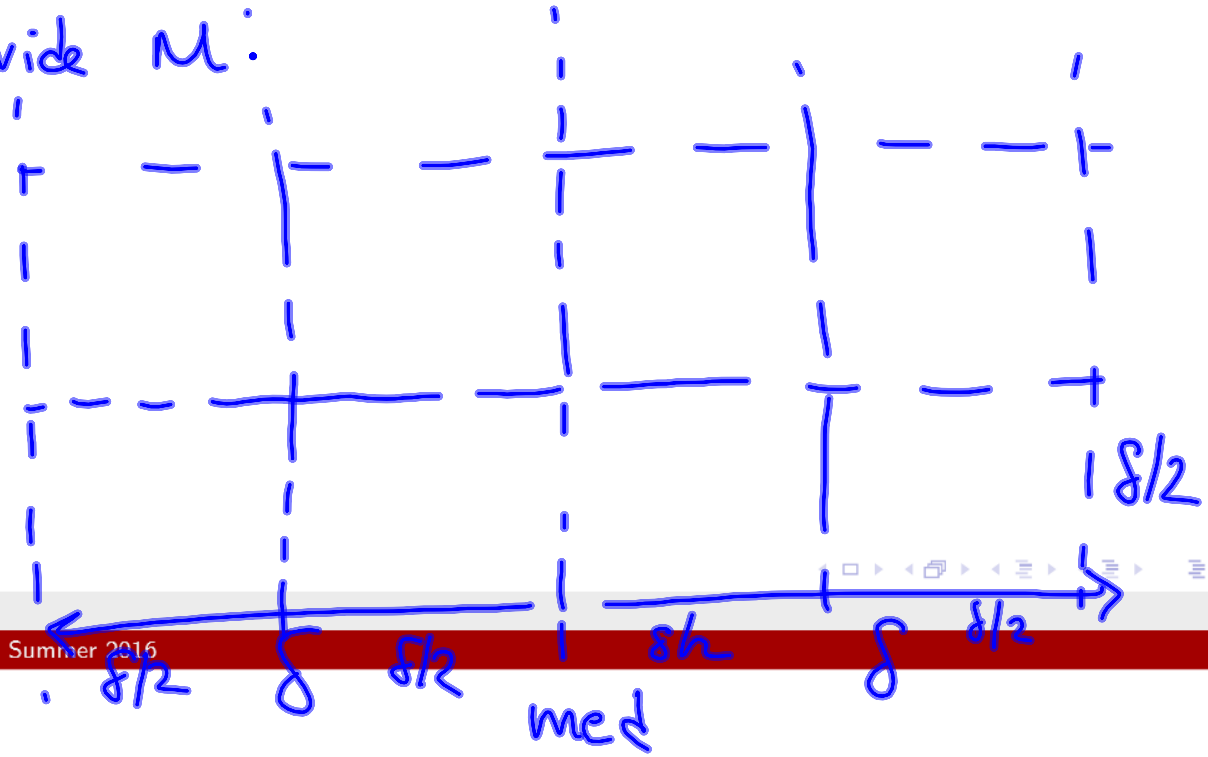
First, sort P by y-**coordinate**, and then call:
Closest-Pair(P)

Compute x-based midpoint of P // $O(n)$: stay tuned
Let L = points from P left of middle + middle
Let R = points from P right of middle
$(\delta_l, l_1, l_2) \leftarrow$ Closest-Pair(L)
$(\delta_r, r_1, r_2) \leftarrow$ Closest-Pair(R)
δ, p_1, p_2 as before...
$M \leftarrow$ points from L and R within δ of middle (x-coordinate), sorted by y-coordinate.
* (few slides →)

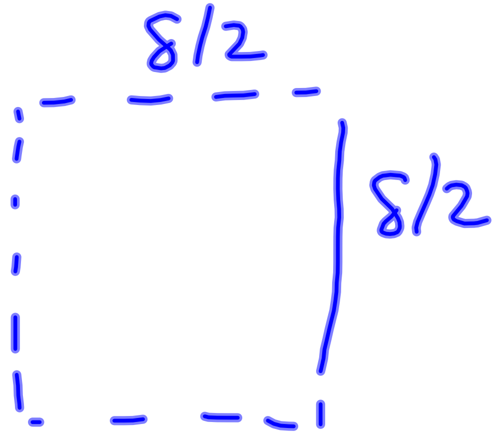
Do we need to find the closest pair in M? No

answer: either closest pair
OR know such is $> \delta$

Divide M:



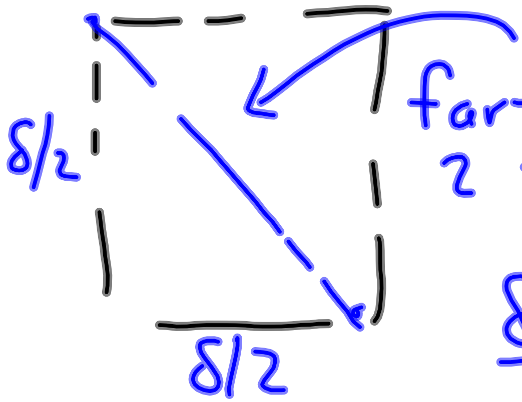
Q1:



within M.

How many points?

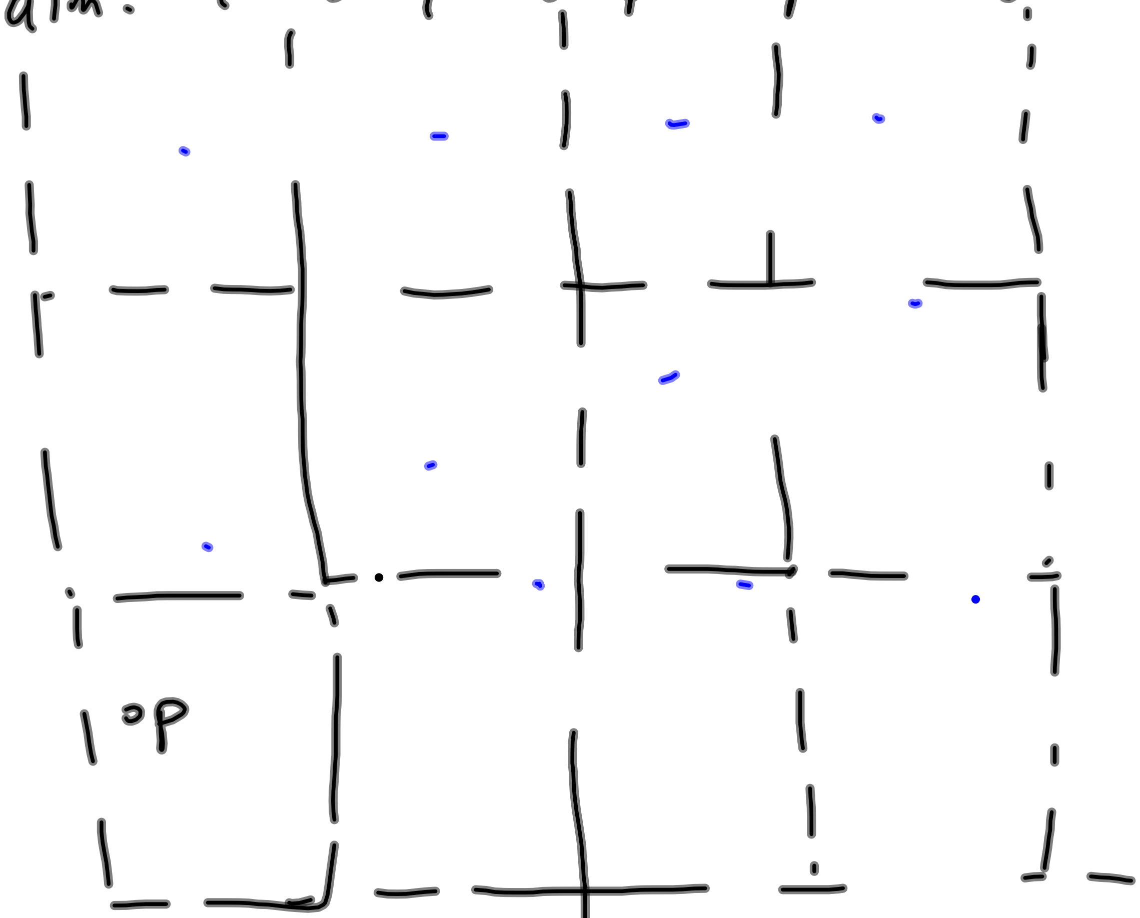
could be vacant. Could have one.



farthest poss 2 pts in a box

$$\frac{\delta\sqrt{2}}{2} < \delta \quad \rightarrow\leftarrow$$

Claim: only compare p to $O(1)$ points



p is M[i]
M[i+1 to i+11]

prev: $2T(\frac{n}{2}) + O(n) +$ this!

$O(n)$ iter

for $i = 1$ to $|M|$

for $j = i+1$ to $\min(|M|, i+11)$

$O(1)$

if $d(M[i], M[j]) < \delta$

$O(1)$

$\delta = d(M[i], M[j])$

$P_1 = M[i]$

$P_2 = M[j]$

Tot: $T(n) = 2T(n/2) + O(n) = O(n \log n)$

Review exercise (time permitting)

Suppose we have an array A such that the first and last elements are ∞. Our goal is to find a *local minimum* in the array. The value $A[i]$ is a local minimum if - and only if - $A[i-1] \geq A[i] \leq A[i+1]$. *For example*, if your input array A is:

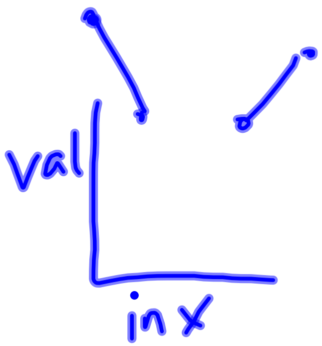
∞	10	23	17	20	1	2	3	4	-3	5	∞

then any of the grayed cells could be returned as a valid answer

~~i=1 j=n~~ Find(start, end)

~~while i<j~~

~~mid = $\frac{i+j}{2}$~~



if A[mid-1] ≥ A[mid] ≤ A[mid+1]

return mid

else if A[mid-1] < A[mid]

j = mid // recurse A[start ... mid-1]

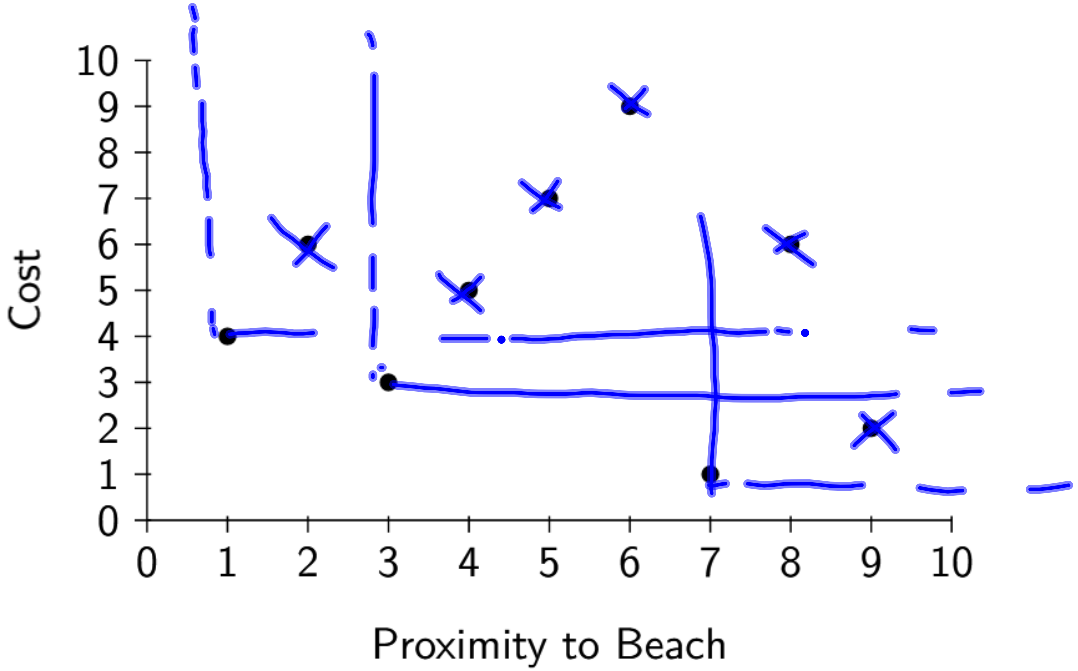
else reeurse A[Mid ... end]

Problem Statement

- We have a database of hotels.
- Each hotel has:
  - a proximity to the beach (x-coordinate)
  - ~~a restaurant quality (y-coordinate)~~

    cost

Example



Brute Force

Sort hotels along any dimension
for $i = 1 \rightarrow n-1$ **do**
  for $j = i+1 \rightarrow n$ **do**
    if A_i is cheaper and ~~has a better restaurant~~ is closer to the beach than A_j **then**
      Remove A_j
return All hotels that we did not remove

- This is $O(n^2)$.

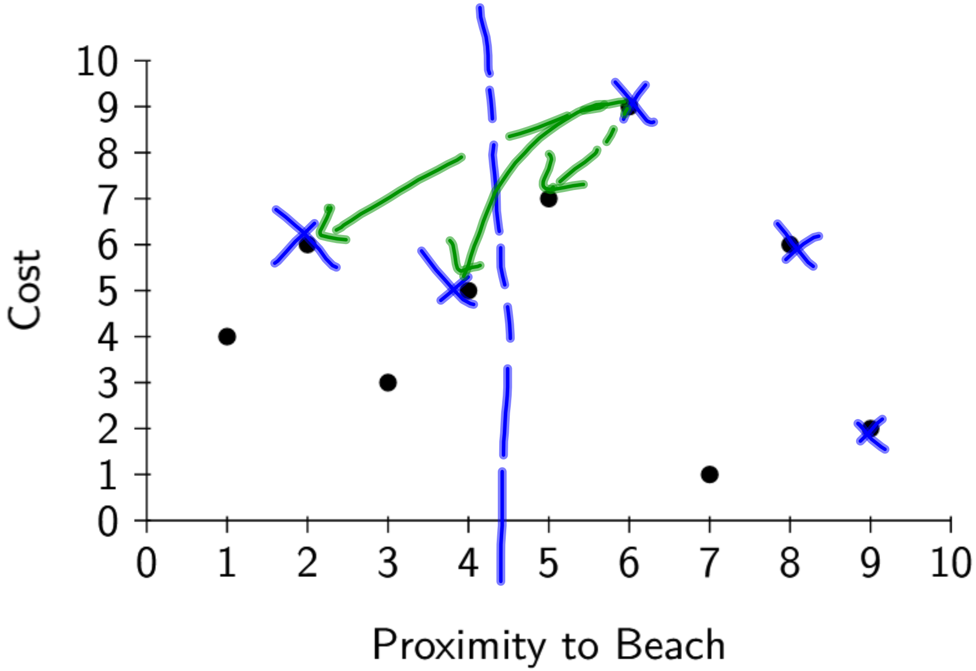
Beginning Divide and Conquer

MaximaSet(S)

if $n \leq 1$ **then**
  return S
$p \leftarrow$ median point in S by x-coordinate // $O(n)$: stay tuned
$L \leftarrow$ points less than p
$G \leftarrow$ points greater than **or equal to** p
$M_1 \leftarrow$ MaximaSet(L)
$M_2 \leftarrow$ MaximaSet(G)

- return $M_1 \cup M_2$? NO... see next slide

Example revisited



Beginning Divide and Conquer

MaximaSet(S)

if $n \leq 1$ **then**

  return S

$p \leftarrow$ median point in S by x-coordinate // $O(n)$: stay tuned

$L \leftarrow$ points less than p

$G \leftarrow$ points greater than **or equal to** p

$M_1 \leftarrow$ MaximaSet(L)

$M_2 \leftarrow$ MaximaSet(G)

- Which point(s) belong for sure? every M_1
- How can we finish the "conquer" step?

Naïve Conquer step

for each $p \in M_1$
    for each $q \in M_2$
        if p dominates q
            remove q

return $M_1 \cup M_2$

no real savings: $2T(n/2) + O(n^2)$

Better conquer step:

$p \leftarrow$ lowest cost hotel in M_1

for each $q \in M_2$

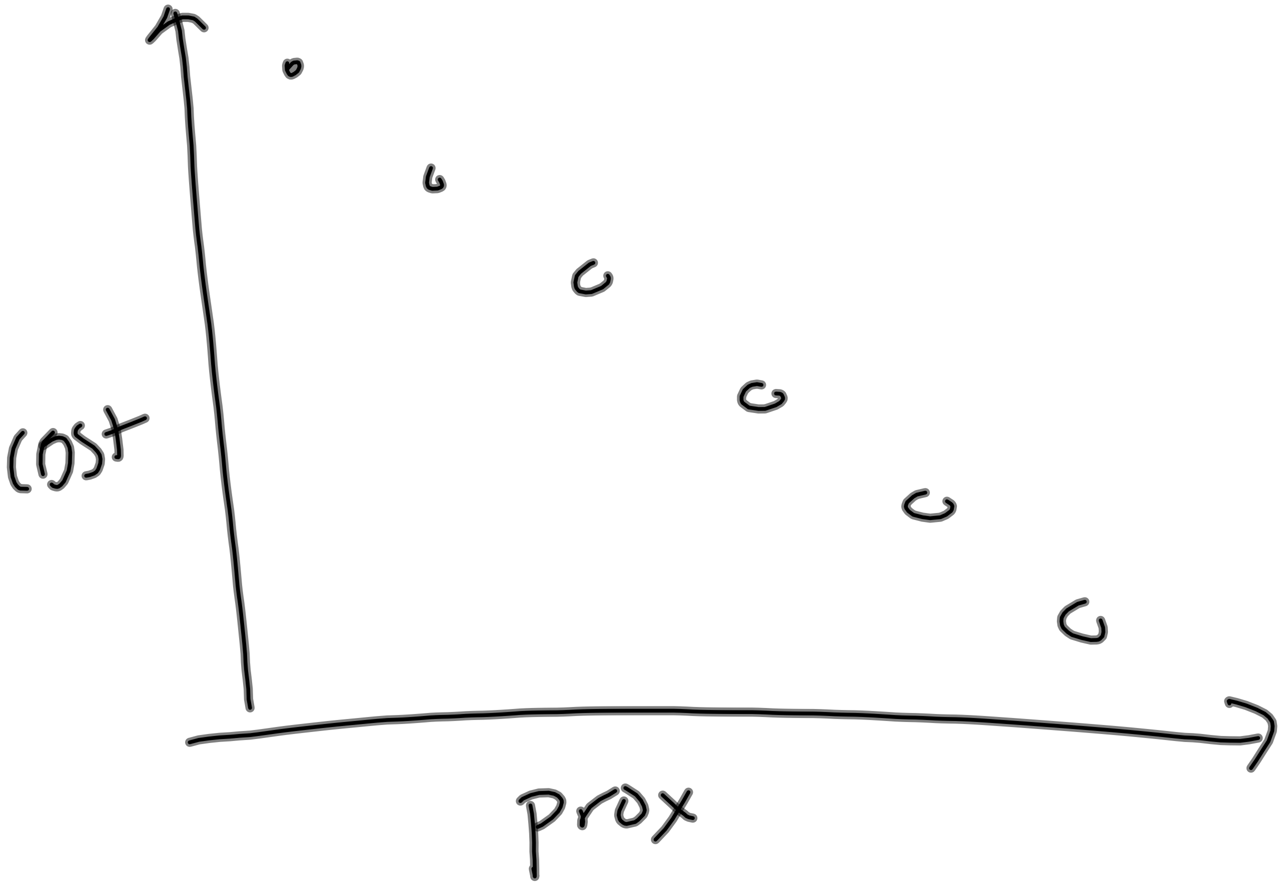
  if p dom. q

    remove q from M_2

return $M_1 \cup M_2$

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

$\Rightarrow O(n \log n)$


cost
prox

What is QuickSort?

- QuickSort is another recursive sorting algorithm.
- For non-base cases, it selects an arbitrary *pivot*, x
- QuickSort *partition*s the array by comparison to x:

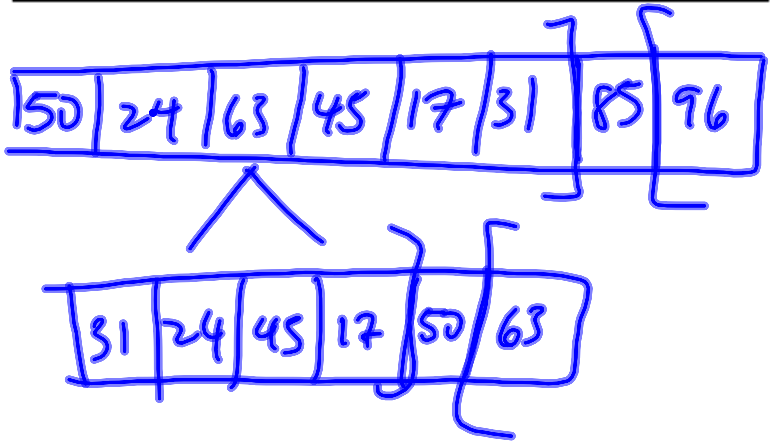
$< x$	x	$> x$

This takes $\Theta(n)$ to do.
- QuickSort then recursively sorts the two "sides" of x.

QuickSort (Review)

pivot

85	24	63	45	17	31	96	50



- Worst-case running time of QuickSort?

$T(n) = n + T(n-1) \to O(n^2)$

- Suppose x chosen uniformly at random. Expected running time?

Is it in blue?

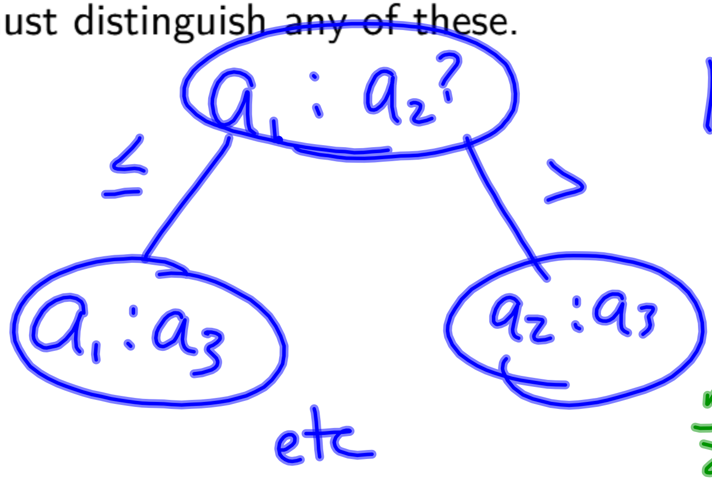
W/ prob ≈ 50%

$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$

or better

Can we sort better than $\Omega(n \log n)$?

- Any of the $n!$ permutations could be input
- Output must distinguish any of these.



height: $\log(n!)$

$$\log(n!) \leq \log n^n \leq n \log n$$

$$\log\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) \leq \log(n!)$$

$$\frac{n}{2} \log\left(\frac{n}{2}\right) \leq$$

$$\frac{n}{2}(\log n - \log 2) \leq$$

Order Selection (A, n, k)

- Given an array A of n elements, we want to find the kth smallest.
- Could sort array and return $A[k]$. Let's do better.
- Recall the partition portion of QuickSort. $\theta(n)$

$< x$	x	$> x$

i

- Is x the kth smallest?

  Maybe. If yes, we're done

- If it isn't, where is the kth smallest?

  if $k < i$, k^{th} smallest of $< x$

  else $(k-i)^{th}$ smallest of $> x$

Deterministic QuickSelect

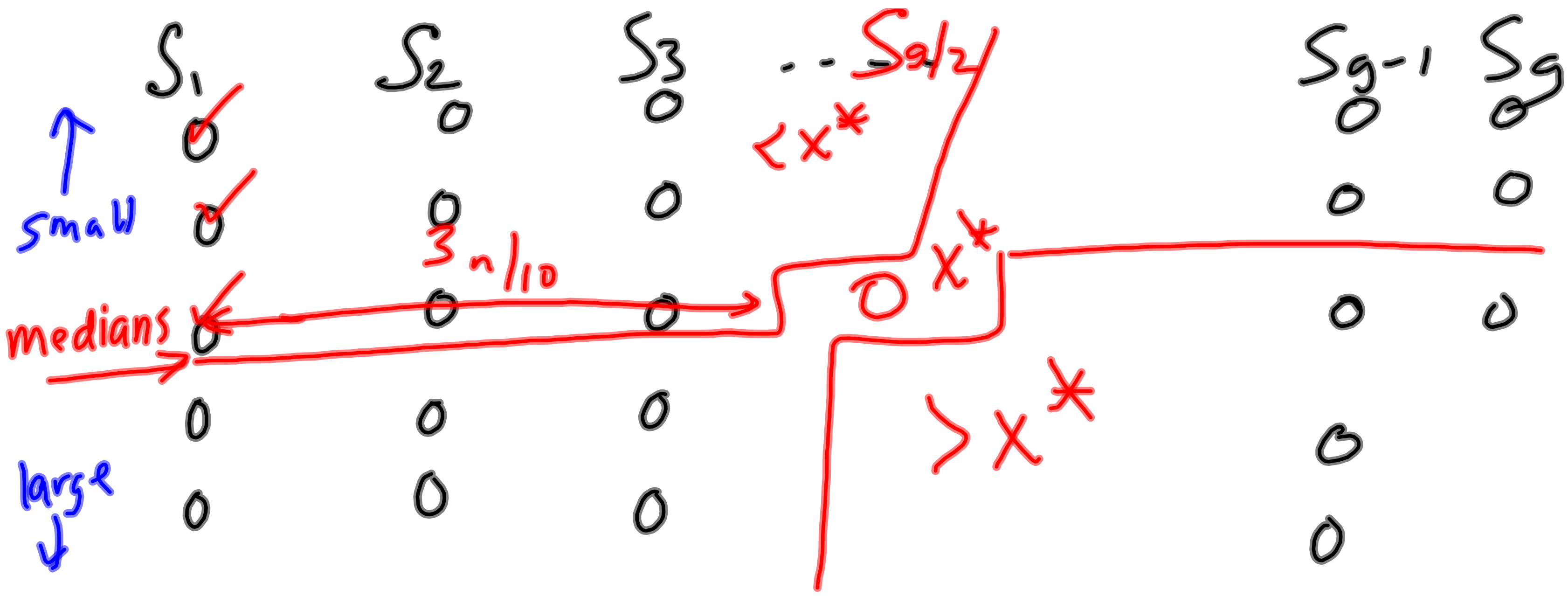
For non-trivial cases:

Divide A into $g = \lceil n/5 \rceil$ groups, $S_1, \ldots, S_g$
for $i \leftarrow 1$ to g **do**
  Compute the median x_i of S_i.

$\Big\}\ \Theta(n) \because g$ is $O(n)$, $|S_i|$ is $O(1)$

$T\left(\frac{n}{5}\right)$ Compute the median x^* of the $\{x_i\}$ set.
Partition A using x^* as our pivot. : $\Theta(n)$

$< x^*$	x^*	$> x^*$



Imagine: - sort each col

- sort columns by x_i

$\approx \frac{3n}{10}$ smaller than x_i^*

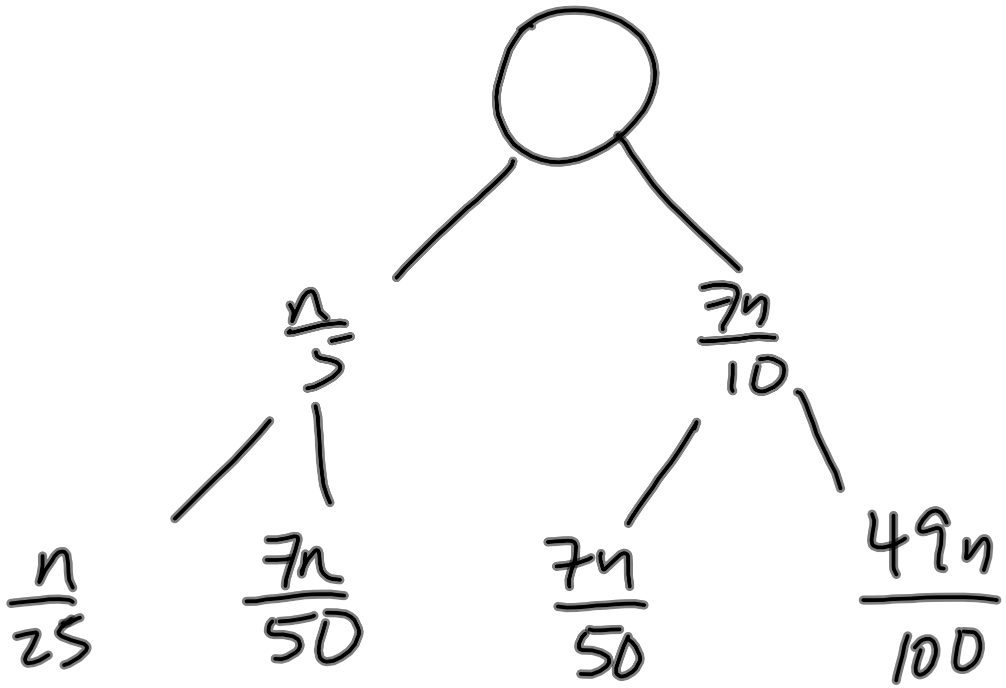
$\approx \frac{3n}{10}$ larger

$<x^*$	x^*	$>x^*$

size of larger subset $\leq \frac{7n}{10}$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n)$$

$$\rightarrow \Theta(n)$$


n/5
7n/10
n/25
7n/50
7n/50
49n/100