Example: Computational Geometry

Closest Pair (in the plane).

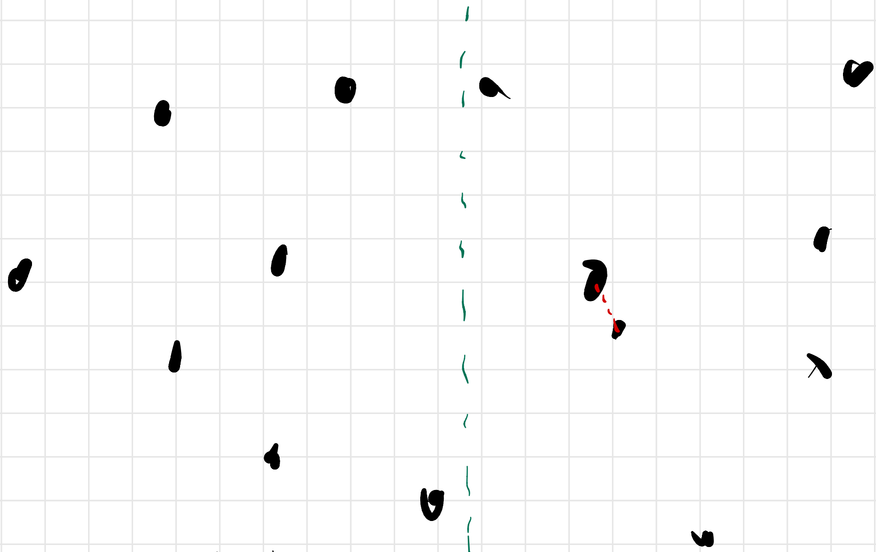
Given n points in the plane $\mathbb{R}^2$.

as $X[1..n]$ & $Y[1..n]$.

ith point located at $(X[i], Y[i])$.

Goal: Find pair with smallest Euclidean distance.

Focus on returning distance



Obvious solution: Check all $\Theta(n^2)$ pairs.

Idea: Split points by median x-coordinate & recurse?

Recurse on each side & check pairs spanning the middle.

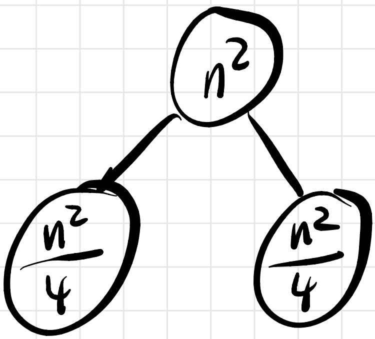
```
ClosestPair(X[1 .. n], Y[1 .. n]):
  if n ≤ 3
      solve by brute force

  XL[1 .. ⌊n/2⌋] and YL[1 .. ⌊n/2⌋] ← leftmost ⌊n/2⌋ points
  ℓ ← ClosestPair(XL[1 .. ⌊n/2⌋], YL[1 .. ⌊n/2⌋])  ⟨⟨Recurse!⟩⟩
  XR[1 .. ⌈n/2⌉] and YR[1 .. ⌈n/2⌉] ← rightmost ⌈n/2⌉ points
  r ← ClosestPair(XR[1 .. ⌈n/2⌉], YR[1 .. ⌈n/2⌉])  ⟨⟨Recurse!⟩⟩

  m ← ∞  ⟨⟨Find closest pair between two halves⟩⟩
  for i ← 1 to ⌊n/2⌋
      for j ← 1 to ⌈n/2⌉
          if Distance(XL[i], YL[i], XR[j], YR[j]) < m
              m ← Distance(XL[i], YL[i], XR[j], YR[j])

  return min {ℓ, r, m}
```

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^2)$$



n^2

$\frac{1}{2} n^2$

$\vdots$

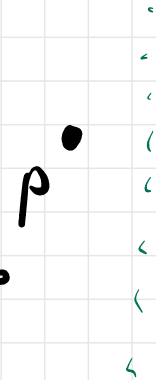
Two facts: $d := \min\{\ell, r\}$

So solution is $\leq d$.

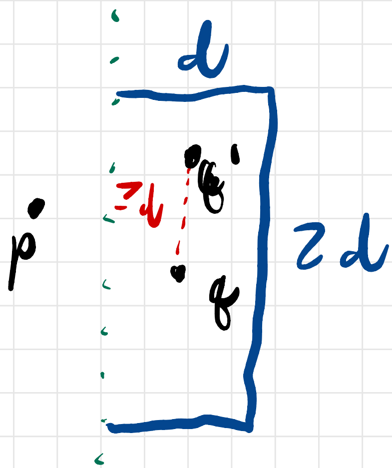
(p, q): closest pair

If (p, q) spans middle line...

1) p + q are distance $\leq d$ from middle line



2)



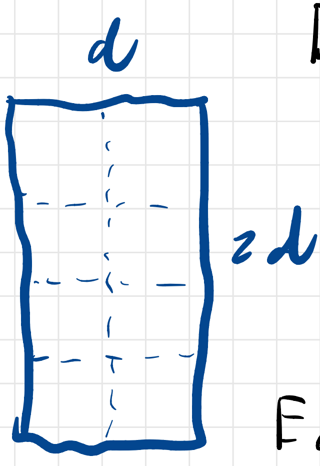
q has a y-coor within d of p's.

$\Rightarrow$ q lives in a $d \times 2d$ box vertically centered on p with a side on middle line

There are few choices for q!

No more than 6.

Proof # is ≤ 8.



Divide $d \times 2d$ box into $(d/2) \times (d/2)$ squares.

Each of diameter

$$\left(\frac{d}{2}\right) \cdot \sqrt{2} = \frac{d}{\sqrt{2}}$$

Choices for q are distance $\geq d$ apart, so ≤ 1 per square. $\square$

$\textsc{ClosestPairFast}(X[1..n], Y[1..n])$:

⟨⟨*Assumes points come pre-sorted by y-coordinate*⟩⟩

if $n \le 3$

  solve by brute force

$XL[1..\lfloor n/2\rfloor]$ and $YL[1..\lfloor n/2\rfloor] \leftarrow$ leftmost $\lfloor n/2\rfloor$ points

$\ell \leftarrow \textsc{ClosestPairFast}(XL[1..\lfloor n/2\rfloor], YL[1..\lfloor n/2\rfloor])$ ⟨⟨*Recurse!*⟩⟩

$XR[1..\lceil n/2\rceil]$ and $YR[1..\lceil n/2\rceil] \leftarrow$ rightmost $\lceil n/2\rceil$ points

$r \leftarrow \textsc{ClosestPairFast}(XR[1..\lceil n/2\rceil], YR[1..\lceil n/2\rceil])$ ⟨⟨*Recurse!*⟩⟩

$d \leftarrow \min\{\ell, r\}$

middle line $-d$

⟨⟨*Find closest pair between two halves*⟩⟩

$XL'[1..k]$ and $YL'[1..k] \leftarrow$ subset of leftmost $\lfloor n/2\rfloor$ points with x-coordinate $\ge \cancel{XR[1]} - d$

$XR'[1..o]$ and $YR'[1..o] \leftarrow$ subset of rightmost $\lceil n/2\rceil$ points with x-coordinate $\le \cancel{XR[1]} + d$

" $+d$

$m \leftarrow \infty$

$jmin \leftarrow 1$

for $i \leftarrow 1$ to k

  while $jmin \le o$ and $YR'[jmin] < YL'[i] - d$

    $jmin \leftarrow jmin + 1$

  $j \leftarrow jmin$

  while $j \le o$ and $YR'[j] \le YL'[i] + d$

    if $\textsc{Distance}(XL'[i], YL'[i], XR'[j], YR'[j]) < m$

      $m \leftarrow \textsc{Distance}(XL'[i], YL'[i], XR'[j], YR'[j])$

    $j \leftarrow j + 1$

return $\min\{\ell, r, m\}$

$$T(n) = 2\,T(n/2) + \Theta(n)$$
$$= \Theta(n \log n)$$

Fibonacci Numbers

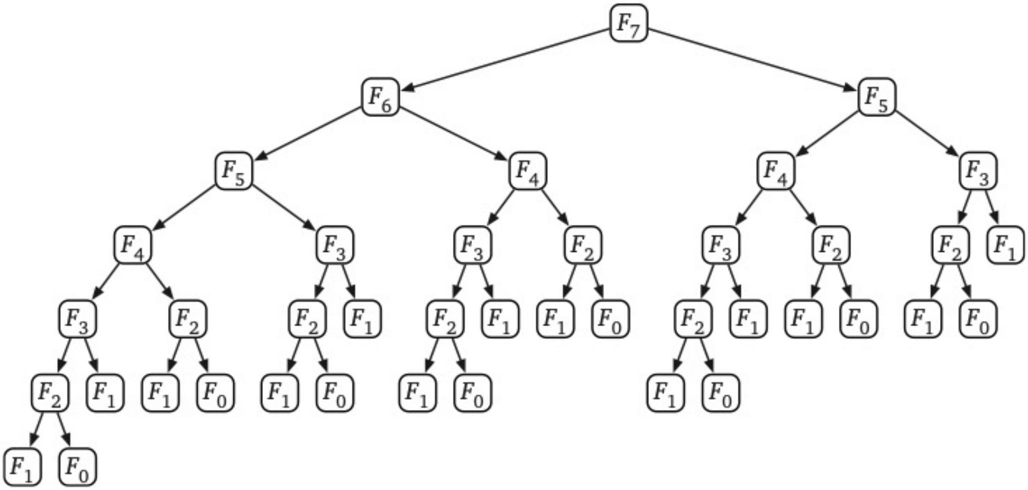
$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise (o.w.)} \end{cases}$$

```
RecFibo(n):
  if n = 0
      return 0
  else if n = 1
      return 1
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

$$T(n) = T(n-1) + T(n-2) + 1$$

$$= 2F_{n+1} - 1$$

$$= \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$



There are F_n leaves of value F_1.

Memoization:

Keep a table/array of known values

$(O(n))$

```
MEMFIBO(n):
  if n = 0
      return 0
  else if n = 1
      return 1
  else
      if F[n] is undefined
          F[n] ← MEMFIBO(n − 1) + MEMFIBO(n − 2)
      return F[n]
```

Solves subproblems in order from 0 to n....

So lets make it explicit!

```
ITERFIBO(n):
    F[0] ← 0
    F[1] ← 1
    for i ← 2 to n
        F[i] ← F[i−1] + F[i−2]
    return F[n]
```

$O(n)$ (operations)

Dynamic Programming:

Solving recursive subproblems in order using a table to store answers

Memory?

Only need to rember last two values, so $O(1)$ possible.

Rod Cutting:

Given an integer n & an array $P[1..n]$ of numbers.

We're handed a rod of length n. Can cut it into smaller pieces (of int length)

We can sell a piece of length i for $P[i]$ USD.

Want to maximize sum of piece prices.

In other words, want a list $\langle i_1, i_2, \ldots, i_k \rangle$ of lengths s.t. $\sum_{j=1}^{k} i_j = n$

s.t. $\sum_{j=1}^{k} P[i_j]$ is maximized.

Ex: $P[1..n] = \langle 1, 5, 8, 9 \rangle$

Best option is $\langle 2, 2 \rangle$

for $5 + 5 = 10$ USD.

If I know the first length i_1, I want to find best solution for remaining $n - i$.

"optimal substruture property"

Backtracking: Guess part of solution, using recursive calls to learn consequences of each choice.

```
RodCutting(P[1 .. n], i):
    if i = 0
        return 0
    maxRev ← P[1] + RodCutting(P, i-1)
    for j ← 2 to i
        optionalRev ← P[j] + RodCutting(P[1 .. n], i − j)
        if optionalRev > maxRev
            maxRev ← optionalRev
    return maxRev
```

As a recurrence...

MaxRevenue(i): max amount from ~~selling~~ chopping a rod of length i.

$$\text{MaxRevenue}(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\limits_{1 \le j \le i} \left(P[j] + \text{MaxRevenue}[i-j]\right) & \text{o.w} \end{cases}$$

next time: memoize it!