There are n people in a village. Everyone knows a subset of the other people in the village, and knowing is a symmetric relation (that is, if a knows b, then b also knows a). One day, one person claims that President Obama is visiting the village. That person shares the rumor with everybody he or she knows. People who hear it further spread it to all their friends, and so on.

1. How would we represent the data as a graph? What data gets modeled as vertices? As edges? Are the edges directed or undirected?

vertices: people
edges: knowing
(undirected)

There are n people in a village. Everyone knows a subset of the other people in the village, and knowing is a symmetric relation (that is, if a knows b, then b also knows a). One day, one person claims that President Obama is visiting the village. That person shares the rumor with everybody he or she knows. People who hear it further spread it to all their friends, and so on.

Same connected component

2. (3 points) Suppose that we know who started the rumor; how would we determine who has heard the rumor? Your answer should be at most one sentence.

There are n people in a village. Everyone knows a subset of the other people in the village, and knowing is a symmetric relation (that is, if a knows b, then b also knows a). One day, one person claims that President Obama is visiting the village. That person shares the rumor with everybody he or she knows. People who hear it further spread it to all their friends, and so on.

connected (all one component)

3. (4 points) What would have to be true about the graph for *everyone* in the village to know about the visit?

- Week i, we will sell p_i pizzas, for n weeks.
- We need to get dough.
- Company A charges $\$a_i$ per pizza if we buy on week i.
- The second company charges us $\$B$ per week, and their price is the same every week. Must buy for 3 weeks at a time.

def OPT(i) : optimal cost wks 1.. i
if $i < 1$ return 0
else
cost a = $a_i \cdot p_i$ + OPT(i-1)
cost b = $3 \cdot B$ + OPT(i-3)
return min(cost a, cost b)

would be O(n) if iterative

Warm Up: Recall the definition of an *inversion* in an array: a pair of indices i, j are an *inverted pair* if $i < j$ and $A[i] > A[j]$. That is, an inverted pair is when the larger ~~pair~~ element appears earlier in the array. Give an $O(n^2)$ time algorithm to count how many inverted pairs exist in an array A.

count = 0
for i = 1 to n
    for j = i+1 to n
        if A[i] > A[j]
            count++

return count

Question: Now suppose you want to count the number of inverted pairs in an array A, but we also know that $A[1 \ldots \frac{n}{2}]$ is sorted, as is $A[\frac{n}{2}+1 \ldots n]$. Can we use this information to count inverted pairs faster?

B[1.. n]. i = 1. j = n/2 + 1. K = 1 Count = 0

while i ≤ n/2 and j ≤ n

if A[i] ≤ A[j]

B[K] = A[i]. K++ i++

else // if A[j] < A[i]

count += # left LHS

B[K] = A[j]. K++ j++

Copy B to A

return count

Question: Can we use the algorithm from question 2 to count the number of inverted pairs in an *unsorted* array faster than $\Theta(n^2)$?

count & sort

- If A sufficiently small, brute force
- else
  - count-and-sort $A[1..n/2]$
  - count-and-sort $A[\frac{n}{2}+1..n]$
  - merge-and-count

$2T(\frac{n}{2})$

$O(n)$

$$T(n) = 2\underbrace{T\left(\tfrac{n}{2}\right)} + n \quad / \quad T(n \le 3) \text{ const.}$$

$$= 2\left(\overline{2T\left(\tfrac{n}{4}\right) + \tfrac{n}{2}}\right) + n = 4T\left(\tfrac{n}{4}\right) + 2n$$

$$= 4\left(2T\left(\tfrac{n}{8}\right) + \tfrac{n}{4}\right) + 2n = 8T\left(\tfrac{n}{8}\right) + 3n$$

$$\vdots$$

$$= 2^i \cdot T\left(\frac{n}{2^i}\right) + in$$

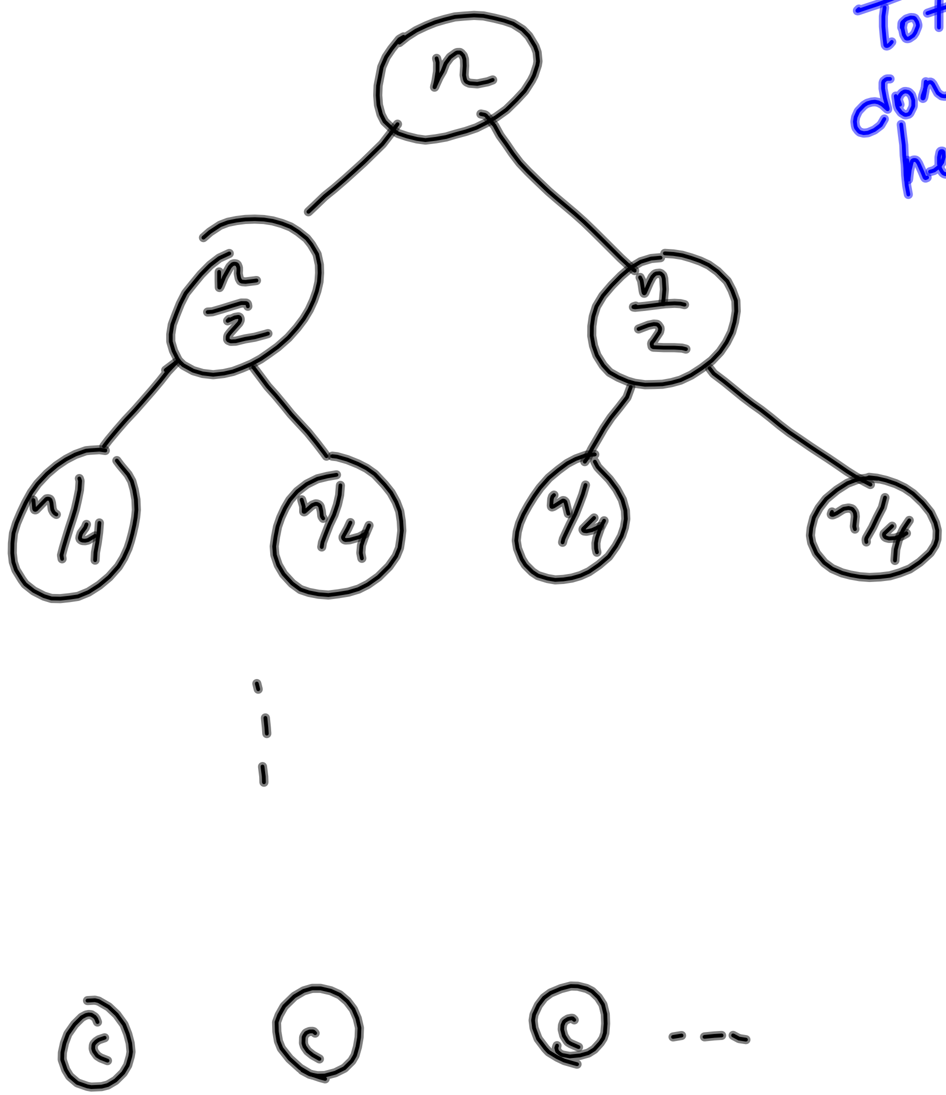
$i = \log_2 n$

$$T(n) = 2^{\log_2 n} \cdot T\left(\frac{n}{2^{\log_2 n}}\right) + (\log_2 n)n$$

$$= n \cdot \text{const} + n\log n$$

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

n		Total done here	n
$\frac{n}{2}$			n
$\frac{n}{4}$			n
$\vdots$			$\vdots$
$\frac{n}{2^{\log n}}$			n
			$O(n \log n)$



Master Theorem

It is common for a divide-and-conquer algorithm's running time to have a recurrence relation of the following form:
$T(n) = aT(n/b) + f(n)$, for some $a \geq 1$, $b > 1$, and $f(n)$ is asymptotically positive.

1. If there is a small constant $\varepsilon > 0$ such that $f(n)$ is $O(n^{\log_b a - \varepsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
2. If there is a constant $k \geq 0$, such that $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
3. If there is a constants $\varepsilon > 0$ such that $f(n)$ is $\Omega(n^{\log_b a + \varepsilon})$, then $T(n)$ is $\Theta(f(n))$.

Use the Master Theorem to solve $T(n) = 4T(n/2) + n$

$a=4 \quad b=2 \quad f(n)=n \quad \log_b a = 2$

for $\varepsilon > 0$

If n is $O(n^{2-\varepsilon})$ ✓

then $T(n)$ is $\Theta(n^2)$

(case 1 holds)

Use the Master Theorem to solve $T(n) = 2T(n/2) + n \log n$

Case 1? $\exists\, \varepsilon > 0$ such that $n \log n$ is $O(n^{1-\varepsilon})$? X

Case 2: $\exists\, k \geq 0$ s.t. $n \log n$ is $\Theta(n \log^{k} n)$, $k =$

$T(n)$ is $\Theta(n \log^2 n)$

Use the Master Theorem to solve $T(n) = T(n/3) + n$

Case 1? $\exists \varepsilon > 0$ s.t. n is $O(n^{0-\varepsilon})$?

Case 2? $\exists k \geq 0$ s.t. n is $\Theta(\cancel{n^0} \cdot \log^k n)$

Case 3? $\exists \varepsilon > 0$ s.t. n is $\Omega(n^{0+\varepsilon})$?
Yes. So $T(n)$ is $\Theta(n)$

Use the Master Theorem to solve $T(n) = 9T(n/3) + n^{2.5}$

$a=9 \quad b=3 \quad f(n)=n^{2.5} \quad \log_3 9 = 2$

Case 1: $\exists\, \varepsilon > 0$ s.t. $n^{2.5}$ is $O(n^{2-\varepsilon})$?

Case 2: $\exists\, k \geq 0$ s.t. $n^{2.5}$ is $\Theta(n^2 \log^k n)$

Case 3: $\exists\, \varepsilon > 0$ s.t. $n^{2.5}$ is $\Omega(n^{2+\varepsilon})$?

Yes

$\textsc{Stooge-Sort}(A, i, j)$

$\left.\begin{array}{l}\textbf{if } A_i > A_j \textbf{ then}\\ \quad \text{swap } A_i \leftrightarrow A_j\end{array}\right\} c$

$\left.\begin{array}{l}\textbf{if } i+1 \geq j \textbf{ then}\\ \quad \textbf{return}\end{array}\right\} c$

$k \leftarrow \lfloor \frac{j-i+1}{3} \rfloor \quad \} c$

$\textsc{Stooge-Sort}(A, i, j-k)$ // First two-thirds
$\textsc{Stooge-Sort}(A, i+k, j)$ // Last two-thirds
$\textsc{Stooge-Sort}(A, i, j-k)$ // First two-thirds, again

- Express the worst-case runtime of $\textsc{Stooge-Sort}$ as a recurrence relation.

$$T(n) = 3T\left(\frac{2}{3}n\right) + \theta(1)$$

Express the worst-case runtime of Stooge-Sort in O or Θ-notation.

$$T(n) = 3T\left(\tfrac{2}{3}n\right) + \Theta(1)$$

$a = 3$ $b = 3/2$ $f(n) = 1$

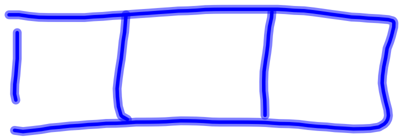
case 1: $\exists\, \varepsilon > 0$ s.t. 1 is $O\left(n^{\log_{3/2} 3 - \varepsilon}\right)$

so $T(n)$ is $\Theta\left(n^{\log_{3/2} 3}\right)$

Does this correctly sort the array, if i is initially 1 and j is initially n? Either explain why you think that it does or provide an array on which it will not correctly sort.

Yes. Base: small trivially correct.

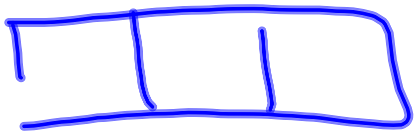
Rec 1:



1st 2/3 sorted.

Where are items belong in final 1/3? now in 2nd or 3rd 1/3

Rec 2:



those are now in final 1/3 and are sorted

1011 1101

11 x 13

5 x 26

~~2 x 52~~ 1 1 1

1 x 104

143

$$
\begin{array}{r l}
1101 & \\
\times\ 1011 & \\
\hline
1101 & 13 \\
1101\text{x} & 26 \\
\text{xx} & \\
1101\text{xxx} & 104 \\
\hline
10001111 &
\end{array}
$$

128 8 4 2 1

13	19
~~6~~	~~38~~
3	76
1	152

$$\begin{array}{r} 152 \\ 95 \\ \hline 247 \end{array}$$