Lecture 13: Sublinear Time Algorithms

- Models
  - Classical Approximation
    - diameter of a point set
  (see slides)
  - Property Testing: approximation for decision problems
    - "sortedness" of a list
    - connectedness of a graph
  - More classical approximation
    - minimum spanning tree
Testing "sortedness" of a list

def: list of size \( n \) is \( \varepsilon \)-close to sorted if can delete \( \leq \varepsilon n \) elements to get a sorted list

Requirements of property testing algorithm:

Input \( \varepsilon \), list \( y_1 \ldots y_n \)

Output

- if \( y_1 \leq y_2 \leq \ldots \leq y_n \) output "Pass"
- if \( y_1 \ldots y_n \) not \( \varepsilon \)-close to sorted output "FAIL" with prob \( \geq 3/4 \)

\[ \text{Note: } p > q \quad \text{is equivalent to } 7q \geq 7p \]
so this is the same as
"If likely to output Pass (with prob \( > 1/3 \)) on this list then list must be \( \varepsilon \)-close to sorted."

Comment: we didn't specify what should happen if \( y_i \)'s not sorted, but not far, either output is reasonable but this leeway allows major speedup!
• Two tests that don't work well:
  1) Pick random \( i \) and test whether \( y_i < y_{i+1} \)
  2) Pick random \( i \neq j \) and test whether \( y_i < y_j \)

Why not?

• Wlog, assume list distinct (i.e., \( x_1 < x_2 < x_3 \) ...)
  (if not, append index to each element)
  \( x_1 \ldots x_n \Rightarrow (x_1, 1), (x_2, 2) \ldots (x_n, n) \)
  breaks ties without changing order

• The test:

  Do \( O(\sqrt{\epsilon}) \) times:
  Pick random \( i \)
  Look at \( y_i \)'s value
  Do binary search on list of \( y_i \)'s for this value
  If don't end up at location \( i \)
  or find an inconsistency along search path, output "FAIL" and halt
  (If no problems found) Output "PASS"

• Runtime \( O(\frac{1}{\epsilon} \log n) \)
Behavior

index "good" if binary search for $y_i$ is successful.

\[ \begin{align*}
e.g., & \quad 1 \quad 4 \quad 2 \quad 5 \quad 7 \quad 1 \quad 14 \quad 19 \\
& \text{bin search for } 7-19 \text{ won't find any inconsistency} \Rightarrow 5-8 \text{ are good} \\
& \text{bin search for index } 1: \text{ (value 1)} \\
& \quad \quad 1 \leftarrow 4 \leftarrow 5 \quad \text{ok} \Rightarrow \text{index } 1 \text{ is good} \\
& \quad \text{bin search for index } 3: \text{ (value 2)} \\
& \quad \quad 1 \leftarrow 4 \leftarrow 5 \quad 2 \text{ not found} \Rightarrow \text{index } 3 \text{ is not good} \\
& \quad \text{bin search for index } 2: \text{ (value 4)} \\
& \quad \quad 4 \leftarrow 5 \quad 2 \text{ found} \Rightarrow \text{index } 2 \text{ is good} \\
& \quad \text{bin search for index } 4: \text{ (value 5)} \\
& \quad \quad 5 \text{ found} \Rightarrow \text{index } 4 \text{ good}
\end{align*} \]
Behavior (cont.)

- if list sorted,
  all is good $\Rightarrow$ list always passes $\uparrow$
  uses distinctness

- if list not $\varepsilon$-close,
  to show: test fails with prob $\geq 3/4$

  Equivalently will show:
  If list passes test with prob $> 1/4$ then
  must be $\varepsilon$-close.

why?
If list passes test with prob $> 1/4$
then $\geq (1-\varepsilon)n$ i's are good
(if $> \varepsilon n$ i's are bad,
then in $\varepsilon/n$ loops will
choose one with prob $\geq 1 - (1 - \varepsilon)^{\varepsilon/n}$
$\geq 1 - e^{-\varepsilon}$
$\geq 3/4$

which contradicts that test passes with prob $> 1/4$
Behavior (cont. again)

Note good elements are in the right order!

Claim if $i < j$ are both good then $y_i < y_j$

Why? let $k$ be least common ancestor of $i, j$

search for $i$ went "left" $\Rightarrow y_i < y_k$

$j$ "right" $\Rightarrow y_k < y_j$

so $y_i < y_j$!

So, delete any bad elements

gives sorted list.
Testing "connectedness" of a graph

Given: graph $G$ in vertices, $m$ edges
max degree $d$
adjacency list representation

def $G$ is $\varepsilon$-close to connected
if adding $\leq \varepsilon dn$ edges transforms it to
connected (today it is ok if transformation
violates max degree $d$ requirement)

Property tester requirements:

Input: $\varepsilon$, $G$

Output:
- if $G$ connected, output "Pass"
- if $G$ not $\varepsilon$-close, output "FAIL" with
  prob $\geq 3/4$

Idea for tester

If $G$ far from connected,
$\Rightarrow$ must have many $(\geq \varepsilon n)$ connected components
$\Rightarrow$ many small connected components
$\Rightarrow$ many nodes in small connected components
Algorithm

Do $O(\frac{1}{\epsilon^2 d})$ times:

Pick random node $s$, run BFS from $s$ until

(a) $\frac{2}{\sqrt{d}}$ distinct nodes seen ≤ good case, continue

or

(b) see that $s$ is in component of size $\leq \frac{2}{\sqrt{d}}$ nodes

↑ bad case, output "FAIL" and halt

(If never entered (b)) Output "PASS"

Runtime

$O(\frac{1}{\epsilon^2 d})$ loops

$O(\frac{1}{\epsilon^2})$ steps of BFS × $O(d)$ time per step

$\Rightarrow O(\frac{1}{\epsilon^2 d})$ time

Behavior

Lemmal If $G$ $\epsilon$-far from connected

Then has $\geq \epsilon d n$ connected components

Pf assume $\leq \epsilon d n$ components

↑ add $\leq \epsilon d n$ edges connects up graph can not $\epsilon$-far! $\Rightarrow n$
Lemma 2: if \[ \geq \frac{edn}{2} \] components of size \( \leq \frac{2}{ed} \), then \[ \geq \frac{edn}{2} \] components of size \( \leq \frac{2}{ed} \).

Proof:

\[ L \leftarrow \text{# conn comp} \]
\[ L' \leftarrow \text{# conn comp of size } \leq \frac{2}{ed} \]
\[ L \leftarrow \text{# conn comp of size } \leq \frac{2}{ed} \]
\[ L' \leq 2/ed \]
\[ L = L + L' \]

Note: \( L \cdot \frac{2}{ed} \leq n \) (else too many nodes!)

so \( L \leq \frac{edn}{2} \)

so \( L' = L - L \geq \frac{edn}{2} \)

Observation: if \( \geq \frac{edn}{2} \) components of size \( \leq \frac{2}{ed} \)

then \( \geq \left( \frac{ed}{2} \right) \cdot n \) nodes in components of size \( \leq \frac{2}{ed} \)

(since each small component has \( \leq 1 \) node)

Putting it together (Lemma 1, 2 + observation):

If \( G \) \( \varepsilon \)-far from connected,

\[ \geq \frac{ed}{2} \] fraction of nodes in small components

\[ \text{We reach (6)} + \text{FAIL!} \]

so \[ \text{Prob}[\text{fail}] \geq 1 - \left( 1 - \frac{ed}{2} \right)^{4/ed} \]

\[ \geq 1 - e^{-4/2} = 3/4 \] for \( C \) big enough.