BILL AND NATHAN
START RECORDING
Hw 07, Problem 2a

\[ \text{CLIQ} = \{ (G, k) : \ G \ \text{has a clique of size } k \} . \]

\( \omega(G) \) is the size of the largest clique in \( G \).
CLIQ = \{(G, k) : G has a clique of size k \}.

\(\omega(G)\) is the size of the largest clique in \(G\).

**Show that if CLIQ ∈ P then there is a poly time algorithm to compute \(\omega(G)\). How many queries?**
Hw 07, Problem 2a

\[ \text{CLIQ} = \{ (G, k) : \text{G has a clique of size } k \} . \]

\( \omega(G) \) is the size of the largest clique in \( G \).

**Show that if CLIQ \( \in \text{P} \) then there is a poly time algorithm to compute \( \omega(G) \). How many queries?**

Given \( G \) do the following.

(1) On Input \( G \) on \( n \) vertices, ask CLIQ \((G, n/2) \in \text{CLIQ}\).
CLIQ = \{(G, k) : G has a clique of size k \}.

\(\omega(G)\) is the size of the largest clique in \(G\).

**Show that if CLIQ \(\in\) P then there is a poly time algorithm to compute \(\omega(G)\). How many queries?**

Given \(G\) do the following.

1. On Input \(G\) on \(n\) vertices, ask CLIQ \((G, \frac{n}{2})\) \(\in\) CLIQ.
2. Proceed by binary search to find \(k\) such that \((G, k) \notin\) CLIQ and \((G, k + 1)\) \(\in\) CLIQ.
CLIQ = \{(G, k) : G has a clique of size k \}. 

ω(G) is the size of the largest clique in G. 

**Show that if CLIQ ∈ P then there is a poly time algorithm to compute ω(G). How many queries?**

Given G do the following.
(1) On Input G on n vertices, ask CLIQ \((G, \frac{n}{2}) \) ∈ CLIQ.
(2) Proceed by binary search to find k such that \((G, k) \notin CLIQ\) and \((G, k + 1) \in CLIQ\).
(3) Output k.
Hw 07, Problem 2a

\[ \text{CLIQ} = \{(G, k) : G \text{ has a clique of size } k\}. \]

\( \omega(G) \) is the size of the largest clique in \( G \).

**Show that if CLIQ \( \in \mathbb{P} \) then there is a poly time algorithm to compute \( \omega(G) \). How many queries?**

Given \( G \) do the following.

1. On Input \( G \) on \( n \) vertices, ask CLIQ \((G, \frac{n}{2}) \in \text{CLIQ}\).
2. Proceed by binary search to find \( k \) such that \((G, k) \notin \text{CLIQ}\) and \((G, k + 1) \in \text{CLIQ}\).
3. Output \( k \).

The algorithm takes \( \lg n + O(1) \) calls to CLIQ.
FCLIQ(G) returns a clique of size \(\omega(G)\).
Show that if CLIQ \(\in P\) then there is a poly time algorithm to compute FCLIQ(G). How many queries?
FCLIQ(G) returns a clique of size $\omega(G)$.
Show that if CLIQ $\in$ P then there is a poly time algorithm to compute FCLIQ(G). How many queries?

Given $G$ do the following:
(1) Use the alg above to find $k = \omega(G)$. $(\lg(n) + O(1))$.
FCLIQ(G) returns a clique of size \( \omega(G) \).

Show that if CLIQ \( \in \mathbb{P} \) then there is a poly time algorithm to compute FCLIQ(G). How many queries?

Given G do the following:
(1) Use the alg above to find \( k = \omega(G) \). \((\lg(n) + O(1))\).
(2) Next goto next page.
(1) We have $G$ and know $k = \omega(G)$. Want to find cliq of size $k$. 
Hw 07, Problem 2b (cont)

(1) We have $G$ and know $k = \omega(G)$. Want to find cliq of size $k$.
(2) For each node $v$ ask the question $\text{CLIQ}(G - \{v\}, k)$. 

Queries $\lg n + O(1)$ to find $\omega(G)$.
$n^2 + \lg(n) + O(1)$ queries to find cliq.
(1) We have $G$ and know $k = \omega(G)$. Want to find cliq of size $k$. 
(2) For each node $v$ ask the question $\text{CLIQ}(G - \{v\}, k)$.
(3a) If YES then we know that $v$ IS NOT NEEDED for the max clique, so remove $v$ and proceed to the next vertex.
(1) We have $G$ and know $k = \omega(G)$. Want to find cliq of size $k$.
(2) For each node $v$ ask the question $\text{CLIQ}(G - \{v\}, k)$.
(3a) If YES then we know that $v$ IS NOT NEEDED for the max clique, so remove $v$ and proceed to the next vertex.
(3b) If NO then we know $v$ IS NEEDED so keep it.
(1) We have $G$ and know $k = \omega(G)$. Want to find cliq of size $k$.
(2) For each node $v$ ask the question $\text{CLIQ}(G - \{v\}, k)$.
(3a) If YES then we know that $v$ IS NOT NEEDED for the max clique, so remove $v$ and proceed to the next vertex.
(3b) If NO then we know $v$ IS NEEDED so keep it.
The graph you are left with is a clique of size $k$. 

$\text{Queries} \ lg(n) + O(1) \ \text{to find} \ \omega(G)$.
$n + \ lg(n) + O(1) \ \text{queries}$
(1) We have $G$ and know $k = \omega(G)$. Want to find cliq of size $k$.
(2) For each node $v$ ask the question $\text{CLIQ}(G - \{v\}, k)$.
(3a) If YES then we know that $v$ IS NOT NEEDED for the max clique, so remove $v$ and proceed to the next vertex.
(3b) If NO then we know $v$ IS NEEDED so keep it.

The graph you are left with is a clique of size $k$.

**Queries** $\lg n + O(1)$ to find $\omega(G)$. $n$ to the find cliq.

$n + \lg(n) + O(1)$ queries
Find a function $f$ such that the following holds:
If you could compute $\text{FCLIQ}(G)$ with $f(n)$ queries to $\text{CLIQ}$ then $P=NP$.
Find a function $f$ such that the following holds:
If you could compute $\text{FCLIQ}(G)$ with $f(n)$ queries to CLIQ then $P=NP$.

$f(n) = \lg n$.

Assume that there is a poly time algorithm for $\text{FCLIQ}(G)$ that makes $\lg n$ queries to CLIQ.
Find a function $f$ such that the following holds:
If you could compute $FCLIQ(G)$ with $f(n)$ queries to CLIQ
then $P=NP$.

$f(n) = \lg n$.

Assume that there is a poly time algorithm for $FCLIQ(G)$ that
makes $\lg n$ queries to CLIQ.

(1) On input $G$ run the algorithm, simulate both the YES and NO
answer for all queries.
Find a function $f$ such that the following holds:
If you could compute $\text{FCLIQ}(G)$ with $f(n)$ queries to CLIQ then $P=NP$.

$f(n) = \lg n$.

Assume that there is a poly time algorithm for $\text{FCLIQ}(G)$ that makes $\lg n$ queries to CLIQ.

(1) On input $G$ run the algorithm, simulate both the YES and NO answer for all queries.

(2) Since there are $\lg n$ queries, there are $\leq 2^{\lg n} = n$ diff outputs.
Find a function $f$ such that the following holds:

If you could compute $\text{FCLIQ}(G)$ with $f(n)$ queries to $\text{CLIQ}$ then $P=NP$.

$f(n) = \lg n$.

Assume that there is a poly time algorithm for $\text{FCLIQ}(G)$ that makes $\lg n$ queries to $\text{CLIQ}$.

(1) On input $G$ run the algorithm, simulate both the YES and NO answer for all queries.

(2) Since there are $\lg n$ queries, there are $\leq 2^{\lg n} = n$ diff outputs.

(3) Hence there will be $n$ candidates for the largest clique. (Some may be garbage- not even cliques!)
Find a function $f$ such that the following holds:

If you could compute $\text{FCLIQ}(G)$ with $f(n)$ queries to $\text{CLIQ}$ then $P=NP$.

$f(n) = \lg n$.

Assume that there is a poly time algorithm for $\text{FCLIQ}(G)$ that makes $\lg n$ queries to $\text{CLIQ}$.

1. On input $G$ run the algorithm, simulate both the YES and NO answer for all queries.
2. Since there are $\lg n$ queries, there are $\leq 2^{\lg n} = n$ diff outputs.
3. Hence there will be $n$ candidates for the largest clique. (Some may be garbage- not even cliques!)
4. Output the largest candidate that is a clique.
Let $A, B, C$ be sets of strings.
Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$. 

Note that $x \in A$ iff $y \in B$ iff $z \in C$.
Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.
$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.
Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.

$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.

$B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.
Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.

$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.
$B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.

Here is a reduction $A \leq C$:

1. Given $x$, compute $y = M_1(x)$. Time $\leq p_1(|x|)$.
   Note $|y| \leq p_1(|x|)$.
2. Then compute $z = M_2(y)$. Time $\leq p_2(|y|) \leq p_2(p_1(|x|))$.

Poly closed under composition, so reduction is poly time.
Note that $x \in A$ iff $y \in B$ iff $z \in C$.
So we have a reduction $A \leq C$. 
Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.

$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.

$B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.

Here is a reduction $A \leq C$:

(1) Given $x$, compute $y = M_1(x)$. Time $p_1(|x|)$. Note $|y| \leq p_1(|x|)$. 
Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.

$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.

$B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.

Here is a reduction $A \leq C$:

(1) Given $x$, compute $y = M_1(x)$. Time $p_1(|x|)$. Note $|y| \leq p_1(|x|)$.

(2) Then compute $z = M_2(y)$. Time $\leq p_2(|y|) \leq p_2(p_1(|x|))$. 

Poly closed under composition, so reduction is poly time.

Note that $x \in A$ iff $y \in B$ iff $z \in C$.

So we have a reduction $A \leq C$. 

Hw 07, Problem 3a

Let $A, B, C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.
$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.
$B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.
Here is a reduction $A \leq C$:

(1) Given $x$, compute $y = M_1(x)$. Time $p_1(|x|)$. Note $|y| \leq p_1(|x|)$.
(2) Then compute $z = M_2(y)$. Time $\leq p_2(|y|) \leq p_2(p_1(|x|))$.
Poly closed under composition, so reduction is poly time.
Let $A$, $B$, $C$ be sets of strings.
Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.

$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly.

$B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.

Here is a reduction $A \leq C$:

(1) Given $x$, compute $y = M_1(x)$. Time $p_1(|x|)$. Note $|y| \leq p_1(|x|)$.

(2) Then compute $z = M_2(y)$. Time $\leq p_2(|y|) \leq p_2(p_1(|x|))$.

Poly closed under composition, so reduction is poly time.

Note that $x \in A$ iff $y \in B$ iff $z \in C$
Let $A, B, C$ be sets of strings. Show that if $A \leq B$ and $B \leq C$ then $A \leq C$.

$A \leq B$ by a poly red via program $M_1$; time $p_1$, a poly. $B \leq C$ by a poly red via program $M_2$; time $p_2$, a poly.

Here is a reduction $A \leq C$:

1. Given $x$, compute $y = M_1(x)$. Time $p_1(|x|)$. Note $|y| \leq p_1(|x|)$.
2. Then compute $z = M_2(y)$. Time $\leq p_2(|y|) \leq p_2(p_1(|x|))$.

Poly closed under composition, so reduction is poly time.

Note that $x \in A$ iff $y \in B$ iff $z \in C$.

So we have a reduction $A \leq C$. 
Show that if $A \leq B$ and $B \in P$ then $A \in P$
Show that if $A \leq B$ and $B \in P$ then $A \in P$

Omitted, but similar to the prior problem.
HW 07 Problem 4a

\[ \text{COL}_k = \{ G : G \text{ is } k\text{-colorable} \}. \]

Show that \( \text{COL}_3 \leq \text{COL}_4 \).
HW 07 Problem 4a

\[ \text{COL}_k = \{ G : G \text{ is } k\text{-colorable} \}. \]

Show that \( \text{COL}_3 \leq \text{COL}_4 \).

Given \( G \), form \( G' \) by adding a vertex and connecting it to all of the original vertices.
HW 07 Problem 4a

\[ \text{COL}_k = \{ G : G \text{ is } k\text{-colorable} \} . \]

Show that \( \text{COL}_3 \leq \text{COL}_4 \).

Given \( G \), form \( G' \) by adding a vertex and connecting it to all of the original vertices.

\( G \) is 3-col iff \( G' \) is 4-col.
COL_k = \{ G : G \text{ is } k\text{-colorable}\}.

Show that COL_3 \leq COL_4.

Given G, form G' by adding a vertex and connecting it to all of the original vertices.

G is 3-col iff G' is 4-col.

Is COL_4 \leq COL_3? Vote.
Show that $\text{COL}_3 \leq \text{COL}_4$.

Given $G$, form $G'$ by adding a vertex and connecting it to all of the original vertices.

$G$ is 3-col iff $G'$ is 4-col.

Is $\text{COL}_4 \leq \text{COL}_3$? Vote.

YES.

$\text{COL}_4 \leq \text{SAT}$ by Cook-Levin.
COL_k = \{ G : G \text{ is } k\text{-colorable} \}.

Show that \(\text{COL}_3 \leq \text{COL}_4\).

Given \(G\), form \(G'\) by adding a vertex and connecting it to all of the original vertices.

\(G\) is 3-col iff \(G'\) is 4-col.

Is \(\text{COL}_4 \leq \text{COL}_3\)? Vote.

YES.

\(\text{COL}_4 \leq \text{SAT}\) by Cook-Levin. \(\text{SAT} \leq \text{COL}_3\) we showed in class.
COL_k = \{ G : G \text{ is } k\text{-colorable} \}.

Show that COL_3 \leq COL_4.

Given G, form G' by adding a vertex and connecting it to all of the original vertices.

G is 3-col iff G' is 4-col.

Is COL_4 \leq COL_3? Vote.

YES.

COL_4 \leq SAT by Cook-Levin. SAT \leq COL_3 we showed in class.

SO by Prior Problem, COL_4 \leq COL_3.
COL_k = \{ G : G \text{ is } k\text{-colorable} \}.

Show that COL_3 \leq COL_4.

Given G, form G' by adding a vertex and connecting it to all of the original vertices.

G is 3-col iff G' is 4-col.

Is COL_4 \leq COL_3? Vote.

YES.

COL_4 \leq SAT by Cook-Levin. SAT \leq COL_3 we showed in class.

SO by Prior Problem, COL_4 \leq COL_3.

This reduction is INSANE.
\text{HW 07 Problem 4a}

\[ \text{COL}_k = \{ G : G \text{ is } k\text{-colorable} \}. \]

Show that \( \text{COL}_3 \leq \text{COL}_4 \).

Given \( G \), form \( G' \) by adding a vertex and connecting it to all of the original vertices.

\( G \) is 3-col iff \( G' \) is 4-col.

Is \( \text{COL}_4 \leq \text{COL}_3 \)? Vote.

YES.

\( \text{COL}_4 \leq \text{SAT} \) by Cook-Levin. \( \text{SAT} \leq \text{COL}_3 \) we showed in class.

SO by Prior Problem, \( \text{COL}_4 \leq \text{COL}_3 \).

This reduction is INSANE.

Is there a sane reduction? YES!
COL_k = \{G : G is k-colorable\}.

Show that COL_3 \leq COL_4.

Given G, form G' by adding a vertex and connecting it to all of the original vertices.

G is 3-col iff G' is 4-col.

Is COL_4 \leq COL_3? Vote.

YES.

COL_4 \leq SAT by Cook-Levin. SAT \leq COL_3 we showed in class.

SO by Prior Problem, COL_4 \leq COL_3.

This reduction is INSANE.

Is there a sane reduction? YES!
$\text{VC}_{17} = \{ G : \text{There is a vertex cover of size } \leq 17 \}.$
HW07, Problem 5a

$VC_{17} = \{ G : \text{There is a vertex cover of size } \leq 17 \}.$

Show that $VC_{17} \in P$ and give the run time.
Show that $\text{VC}_{17} \in \mathbb{P}$ and give the run time. Given $G$: 

\[
\text{VC}_{17} = \{ G : \text{There is a vertex cover of size } \leq 17 \}.
\]
VC_{17} = \{ G : \text{There is a vertex cover of size } \leq 17 \}.

Show that \( VC_{17} \in P \) and give the run time.

Given \( G \):

(1) check every combo of 17 vertices to see if it is a vertex cover.
Show that $\text{VC}_{17} \in \text{P}$ and give the run time. Given $G$:

(1) check every combo of 17 vertices to see if it is a vertex cover.
(2) There are $\binom{n}{17} \leq n^{17}$ such combo, and for each combo, there are $\leq n^2$ edges to check, so our alg runs in $n^{19}$ time.
VCₖ = \{ G : \text{There is a vertex cover of size } \leq k \}.

Find a function f such that the following is true.
VCₖ can be solved in time f(n, k).
\( \text{VC}_k = \{ G : \text{There is a vertex cover of size } \leq k \} \).

Find a function \( f \) such that the following is true.

\( \text{VC}_k \) can be solved in time \( f(n, k) \).

Given \( G \)

(1) check every combo of \( k \) vertices to see if it is a vertex cover.
Find a function $f$ such that the following is true.

$\text{VC}_k$ can be solved in time $f(n, k)$.

Given $G$

(1) check every combo of $k$ vertices to see if it is a vertex cover.

(2) There are $\binom{n}{k} \leq n^k$ such combo, and for each combo, there are $\leq n^k$ edges to check, so our alg runs in $n^{k+2}$ time.
Vote for one of the following.
1. There is an algorithm for $\text{VC}_k$ where the deg doesn't depend on $k$.
2. If $P \neq NP$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)
Answer is (1).
See next slide.
Vote for one of the following.

1. There is an algorithm for \( VC_k \) where the deg doesn't depend on \( k \).
2. If \( P \neq NP \) then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1).
See next slide.
Vote for one of the following.

1. There is an algorithm for $VC_k$ where the deg doesn’t dep on $k$. 
Vote for one of the following.

1. There is an algorithm for $VC_k$ where the deg doesn’t dep on $k$.
2. If $P \neq NP$ then there is no such algorithm.
Vote for one of the following.

1. There is an algorithm for $VC_k$ where the deg doesn’t dep on $k$.
2. If $P \neq NP$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)
Vote for one of the following.

1. There is an algorithm for $VC_k$ where the deg doesn’t dep on $k$.
2. If $P \neq NP$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1).
Vote for one of the following.

1. There is an algorithm for $\text{VC}_k$ where the deg doesn’t dep on $k$.
2. If $P \neq \text{NP}$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1).

See next slide.
Given $(G, k)$ form the following tree.

1. Root has $G$ and an edge $(u, v)$. **Key** either $u$ or $v$ has to be in the VC.

2. Left child is $G - \{u\}$, Right child is $G - \{v\}$. For both pick an edge and do Left-Right as in step (1).

3. Grow the tree to $k$ levels.

4. If any of tree-nodes is the empty graph then have a VC of size $k$. If none do then there is no VC of size $k$.

$\leq 2^k$ tree-nodes. Each one takes $O(n)$ to process. Time $O(2^k n)$.

Can we do even better? Next Slide.
VC}_k

Given \((G, k)\) form the following tree.

(1) Root has \(G\) and an edge \((u, v)\).

**Key** either \(u\) or \(v\) has to be in the VC.

(2) Left child is \(G - \{u\}\), Right child is \(G - \{v\}\).

For both pick an edge and do Left-Right as in step (1).
Given \((G, k)\) form the following tree.

1. Root has \(G\) and an edge \((u, v)\). **Key** either \(u\) or \(v\) has to be in the VC.

2. Left child is \(G - \{u\}\), Right child is \(G - \{v\}\). For both pick an edge and do Left-Right as in step (1).

3. Grow the tree to \(k\) levels.
Given \((G,k)\) form the following tree.

1. Root has \(G\) and an edge \((u,v)\).
   - **Key**: either \(u\) or \(v\) has to be in the VC.

2. Left child is \(G - \{u\}\), Right child is \(G - \{v\}\).
   - For both pick an edge and do Left-Right as in step (1).

3. Grow the tree to \(k\) levels.

4. If any of tree-nodes is the empty graph then have a VC of size \(k\).
   - If none do then there is no VC of size \(k\).
Given \((G, k)\) form the following tree.

1. Root has \(G\) and an edge \((u, v)\).
   **Key** either \(u\) or \(v\) has to be in the VC.

2. Left child is \(G - \{u\}\), Right child is \(G - \{v\}\).
   For both pick an edge and do Left-Right as in step (1).

3. Grow the tree to \(k\) levels.

4. If any of tree-nodes is the empty graph then have a VC of size \(k\).
   If none do then there is no VC of size \(k\).

\(\leq 2^k\) tree-nodes. Each one takes \(O(n)\) to process. Time \(O(2^kn)\).
Given $(G, k)$ form the following tree.
(1) Root has $G$ and an edge $(u, v)$.
**Key** either $u$ or $v$ has to be in the VC.
(2) Left child is $G - \{u\}$, Right child is $G - \{v\}$.
For both pick an edge and do Left-Right as in step (1).
(3) Grow the tree to $k$ levels.
(4) If any of tree-nodes is the empty graph then have a VC of size $k$.
If none do then there is no VC of size $k$.

$\leq 2^k$ tree-nodes. Each one takes $O(n)$ to process. Time $O(2^k n)$.

Can we do even better? Next Slide.
We have run time $O(2^k n)$. We think of $k \ll n$.
We have run time $O(2^k n)$. We think of $k \ll n$.
Vote for one of the following.

1. $(\exists)$ alg for $\text{VC}$ $k$ and, time $n O(1) + f(k)$ for some $f(k)$.
2. If $P \neq \text{NP}$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1). See next slide.
Better?

We have run time $O(2^k n)$. We think of $k \ll n$.

Vote for one of the following.

1. $(\exists) \text{ alg for } \mathcal{VC}_k \text{ and, time } n^{O(1)} + f(k) \text{ for some } f$. 

If $P \neq NP$ then there is no such algorithm.

Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1). See next slide.
Better?

We have run time $O(2^k n)$. We think of $k \ll n$.

Vote for one of the following.

1. $(\exists)$ alg for $VC_k$ and, time $n^{O(1)} + f(k)$ for some $f$.
2. If $P \neq NP$ then there is no such algorithm.
We have run time $O(2^k n)$. We think of $k \ll n$.

Vote for one of the following.

1. \((\exists)\) alg for $\text{VC}_k$ and, time $n^{O(1)} + f(k)$ for some $f$.
2. If $P \neq \text{NP}$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)
Better?

We have run time $O(2^k n)$. We think of $k \ll n$.

Vote for one of the following.

1. $(\exists)$ alg for $VC_k$ and, time $n^{O(1)} + f(k)$ for some $f$.
2. If $P \neq NP$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1).
We have run time $O(2^k n)$. We think of $k \ll n$.

Vote for one of the following.

1. $(\exists)$ alg for $VC_k$ and, time $n^{O(1)} + f(k)$ for some $f$.
2. If $P \neq NP$ then there is no such algorithm.
3. Neither (a) or (b) is known (Guido will determine which one this summer.)

Answer is (1).

See next slide.
Given $G$ notice that

*if vertex $v$ has degree $\geq k + 1$ then it HAS to be in the VC*

1. Input $G = (V, E)$. Let $A = \emptyset$. Let $t = k$.
2. For $v \in V$
   2.1 If $v$ has degree $\geq t + 1$ then
      \[ A := A \cup \{v\}, \]
      \[ t := t - 1, \]
      \[ G := G - \{v\}. \]
3. If the graph is empty, you are done, and $A$ is your VC.
4. If the graph is not empty then you have a graph with every vertex degree $\leq k$. Leave the rest to you.

Run time is

\[ n + \alpha^k \]

More refined approaches have gotten $\alpha = 1.34$. 