<table>
<thead>
<tr>
<th>Lecture A</th>
<th>Tiefenbruck</th>
<th>Tu, Th 11am-12:20pm</th>
<th>Center 119</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture B</td>
<td>Tiefenbruck</td>
<td>Tu, Th 9:30am-10:50am</td>
<td>Center 119</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Minnes</td>
<td>Tu, Th 3:30pm-4:50pm</td>
<td>WLH 2005</td>
</tr>
</tbody>
</table>

http://cseweb.ucsd.edu/classes/fa15/cse21-abc/

Nov 12, 2015
Reminders

Midterm 2: Tuesday, November 17.

* In class. Exam will be 1 hour, 15 minutes. (5 minutes for hand in)
* **Practice midterm** available on website/ Piazza.
* Review sessions Saturday & Sunday: see website.
* Extra office hours over the weekend.
* Monday discussion session review.
* **Seating chart on website/ Piazza.**
* 1 handwritten note sheet allowed.
* If you have AFA letter, see me as soon as possible.
Uniquely decodable encoding algorithm for fixed-density binary strings.

For strings of length n, with k 1s.

Idea: use marker bit to indicate when to interpret output as a position.
- Fix window size.
- If there is a 1 in the current "window" in the string, record a 1 to interpret next bits as position, then record its position and move the window over.
- Otherwise, record a 0 and move the window over.
- After we see the last 1, don't need to add 0s to indicated empty windows.
Output size?

Assume $n/k$ is a power of two. Consider $s$ a binary string of length $n$ with $k$ 1s.

$$k \log_2(n/k) + k \leq |E(s)| \leq k \log_2(n/k) + 2k$$
Output size?

Assume \( n/k \) is a power of two. Consider \( s \) a binary string of length \( n \) with \( k \) 1s. Given \( |E(s)| \leq k \log_2(n/k) + 2k \), we need at most \( k \log_2(n/k) + 2k \) bits to represent all length \( n \) binary strings with \( k \) 1s. Hence, there are at most \( 2^{k \log_2(n/k) + 2k} \) many such strings.
Output size?

Assume $n/k$ is a power of two. Consider $s$ a binary string of length $n$ with $k$ 1s. Given $|E(s)| \leq k \log_2(n/k) + 2k$, we need at most $k \log_2(n/k) + 2k$ bits to represent all length $n$ binary strings with $k$ 1s. Hence, there are at most $2 \cdots$ many such strings.

\[
2^{(k \log(n/k)+2k)} = 2^{(k \log(n/k))} \cdot 2^{(2k)} \\
= \left(2^{(\log(n/k))}\right)^k \cdot 2^{(2k)} \\
= (n/k)^k \cdot 4^k = (4n/k)^k
\]
Output size?

Assume $n/k$ is a power of two. Consider $s$ a binary string of length $n$ with $k$ 1s. Given $|E(s)| \leq k \log_2(n/k) + 2k$, we need at most $k \log_2(n/k) + 2k$ bits to represent all length $n$ binary strings with $k$ 1s. Hence, there are at most $2^{k \log_2(n/k) + 2k}$ such strings.

$$2^{(k \log(n/k) + 2k)} = 2^{(k \log(n/k))} \cdot 2^{(2k)} = \left(2^{\log(n/k)}\right)^k \cdot 2^{(2k)} = (n/k)^k \cdot 4^k = (4n/k)^k$$

$$C(n,k) = \# \text{ Length } n \text{ binary strings with } k \text{ 1s} \leq (4n/k)^k$$
Using \texttt{windowEncode()}: \( \binom{n}{k} \leq (4n/k)^k \)

Lower bound?

**Idea:** find a way to count a subset of the fixed density binary strings.

Some fixed density binary strings have one 1 in each of \( k \) chunks of size \( n/k \).

How many such strings are there?

A. \( n^n \)  
B. \( k! \)  
C. \( (n/k)^k \)  
D. \( \text{C}(n,k)^k \)  
E. None of the above.
Bounds for Binomial Coefficients

Using `windowEncode()`:

\[
\binom{n}{k} \leq \left(\frac{4n}{k}\right)^k
\]

Using evenly spread strings:

\[
\left(\frac{n}{k}\right)^k \leq \binom{n}{k}
\]

**Counting** helps us analyze our **compression algorithm**.

**Compression algorithms** help us **count**.
A **theoretically optimal encoding** for length n binary strings with k 1s would use the ceiling of $\log_2 \left( \binom{n}{k} \right)$ bits.

**How?**
- List all length n binary strings with k 1s in some order.
- **To encode:** Store the position of a string in the list, rather than the string itself.
- **To decode:** Given a position in list, need to determine string in that position.
A **theoretically optimal encoding** for length $n$ binary strings with $k$ 1s would use the ceiling of $\log_2 \left( \binom{n}{k} \right)$ bits.

**How?**
- List all length $n$ binary strings with $k$ 1s in some order.
- To encode: Store the **position** of a string in the list, rather than the string itself.
- To decode: Given a position in list, need to determine string in that position.

Use lexicographic (dictionary) ordering ...
String \( a \) comes \textbf{before} \ \textit{string} \( b \) if the \textbf{first time they differ}, \( a \) is smaller.

I.e.

\[
a_1a_2…a_n <_{\text{lex}} b_1b_2…b_n
\]

means there exists \( j \) such that

\[
a_i=b_i \text{ for all } i<j \text{ AND } a_j<b_j
\]

Which of these comes \textbf{last} in lex order?

A. 1001  
B. 0011  
C. 1101  
D. 1010  
E. 0000
E.g. Length n=5 binary strings with k=3 ones, listed in lex order:

<table>
<thead>
<tr>
<th>Original string, s</th>
<th>Encoded string (i.e. position in this list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00111</td>
<td>0 = 0000</td>
</tr>
<tr>
<td>01011</td>
<td>1 = 0001</td>
</tr>
<tr>
<td>01101</td>
<td>2 = 0010</td>
</tr>
<tr>
<td>01110</td>
<td>3 = 0011</td>
</tr>
<tr>
<td>10011</td>
<td>4 = 0100</td>
</tr>
<tr>
<td>10101</td>
<td>5 = 0101</td>
</tr>
<tr>
<td>10110</td>
<td>6 = 0110</td>
</tr>
<tr>
<td>11001</td>
<td>7 = 0111</td>
</tr>
<tr>
<td>11010</td>
<td>8 = 1000</td>
</tr>
<tr>
<td>11100</td>
<td>9 = 1001</td>
</tr>
</tbody>
</table>
Lex Order: Algorithm?

Need two algorithms, given specific n and k:

\[ s \rightarrow E(s,n,k) \]

and

\[ p \rightarrow D(p,n,k) \]

*Idea:* Use recursion (reduce & conquer).
Lex Order: Algorithm?

For $E(s,n,k)$:

- Any string that starts with 0 must have position before $\binom{n - 1}{k}$
- Any string that starts with 1 must have position after $\binom{n - 1}{k}$
Lex Order: Algorithm?

For $E(s,n,k)$:

- Any string that starts with 0 must have position **before** \( \binom{n-1}{k} \).
- Any string that starts with 1 must have position **after** \( \binom{n-1}{k} \).

**procedure** lexEncode \( (b_1b_2\ldots b_n, n, k) \)

1. If $n = 1$,
2. return 0.
3. If $s_1 = 0$,
4. return lexEncode \( (b_2\ldots b_n, n-1, k) \)
5. Else
6. return \( C(n-1,k) + \text{lexEncode}(b_2\ldots b_n, n-1, k-1) \)
Lex Order: Algorithm?

For $D(s,n,k)$:

- Any position **before** $\binom{n-1}{k}$ must correspond to string that starts with 0.
- Any position **after** $\binom{n-1}{k}$ must correspond to string that starts with 1.

```plaintext
procedure lexDecode (p, n, k)
  1. If n = k,
  2. return 1111..1 //length n string of all 1s.
  3. If p < C(n-1,k),
  4. return "0" + lexDecode(p, n-1, k)
  5. Else
  6. return "1" + lexDecode(p-C(n-1,k), n-1, k-1)
```
Using lexEncode, lexDecode, we can represent any fixed density length n binary string with k 1s as a number in the range 1 … C(n,k).

So, it takes $\log_2( C(n,k) )$ bits to store fixed-density binary strings using lex order.

**Theoretical lower bound**: $\log_2( C(n,k) )$.

**Same!** So this encoding algorithm is optimal.
Representing undirected graphs

**Strategy:**

1. **Count** the number of simple undirected graphs.
2. Compute **lower bound** on the number of bits required to represent these graphs.
3. Devise **algorithm** to represent graphs using this number of bits.

What's true about *simple undirected* graphs?

A. Self-loops are allowed.
B. Parallel edges are allowed.
C. There must be at least one vertex.
D. There must be at least one edge.
E. None of the above.

*Rosen p. 641-644*
In a simple undirected graph on n (labeled) vertices, how many edges are possible?

A. $n^2$
B. $n(n-1)$
C. $C(n,2)$
D. $2^{C(n,2)}$
E. None of the above.

** Recall notation: $C(n,k) = \binom{n}{k}$ **
In a simple undirected graph on \( n \) (labeled) vertices, how many edges are possible?

A. \( n^2 \)
B. \( n(n-1) \)
C. \( \binom{n}{2} \)  
   Possibly one edge for each set of two distinct vertices.
D. \( 2^{\binom{n}{2}} \)
E. None of the above.

** Recall notation: \( \binom{n}{k} = \binom{n}{k} \)**
Representing undirected graphs: Counting

How many different undirected graphs on n (labeled) vertices are there?

A. $n^2$
B. $n(n-1)$
C. $C(n,2)$
D. $2^{C(n,2)}$
E. None of the above.
Representing undirected graphs: Lower bound

How many different undirected graphs on n (labeled) vertices are there?

A. $n^2$
B. $n(n-1)$
C. $C(n,2)$
D. $2^{C(n,2)}$
E. None of the above.

*For each possible edge, decide if in graph or not.*

Conclude:
minimum number of bits to represent simple undirected graphs with n vertices is

$$\log_2(2^{C(n,2)}) = C(n,2) = \frac{n(n-1)}{2}$$
Representing undirected graphs: Algorithm

**Goal**: represent a simple undirected graph with $n$ vertices using $n(n-1)/2$ bits.

**Idea**: store adjacency matrix, but since

- diagonal entries all zero  
  *no self loops*
- matrix is symmetric  
  *undirected graph*

only store the entries **above** the diagonal.

**How many entries of the adjacency matrix are above the diagonal?**

A. $n^2$  
B. $n(n-1)$  
C. $C(n,2)$  
D. $2n$  
E. None of the above.
Representing undirected graphs: Algorithm

**Goal**: represent a simple undirected graph with n vertices using \( n(n-1)/2 \) bits

**Idea**: store adjacency matrix, but since

- diagonal entries all zero *no self loops*
- matrix is symmetric *undirected graph*

only store the entries **above** the diagonal.

Can be stored as **0111101100**
which uses \( \binom{5,2} = 10 \) bits.
Representing undirected graphs: Algorithm

**Decoding:** ?

What simple undirected graph is encoded by the binary string 011010 110101 111111 000000 110101 110010 ?

<table>
<thead>
<tr>
<th>A.</th>
<th>B.</th>
</tr>
</thead>
</table>
| \[
\begin{pmatrix}
0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 \\
\end{pmatrix}
| \[
\begin{pmatrix}
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
|

C. Either one of the above.  
D. Neither one of the above.
In a simple **directed** graph on \( n \) (labeled) vertices, how many edges are possible?

A. \( n^2 \)
B. \( n(n-1) \)
C. \( \binom{n}{2} \)
D. \( 2^{\binom{n}{2}} \)
E. None of the above.

Simple graph: no self loops, no parallel edges.
Representing directed graphs: Counting

In a simple directed graph on \( n \) (labeled) vertices, how many edges are possible?

A. \( n^2 \)
B. \( n(n-1) \)
C. \( \binom{n}{2} \)
D. \( 2^{\binom{n}{2}} \)
E. None of the above.

Choose starting vertex, choose ending vertex.

Simple graph: no self loops, no parallel edges.
Representing directed graphs: Counting

How many different directed graphs on \( n \) (labeled) vertices are there?

A. \( n^2 \)
B. \( n(n-1) \)
C. \( C(n,2) \)
D. \( 2^{C(n,2)} \)
E. None of the above.
Another way of counting that there are $2^{n(n-1)}$ directed graphs with $n$ vertices:

Represent a graph by
For each of the $C(n,2)$ pairs of distinct vertices $\{v,w\}$, specify whether there is
* no edge between them
* an edge from $v$ to $w$ but no edge from $w$ to $v$
* an edge from $w$ to $v$ but no edge from $v$ to $w$
* edges both from $v$ to $w$ and from $v$ to $w$. 
Another way of counting that there are $2^{n(n-1)}$ directed graphs with $n$ vertices:

Represent a graph by

For each of the $C(n,2)$ pairs of distinct vertices \{v,w\}, specify whether there is

* no edge between them
* an edge from $v$ to $w$ but no edge from $w$ to $v$
* an edge from $w$ to $v$ but no edge from $v$ to $w$
* edges both from $v$ to $w$ and from $v$ to $w$.

Product rule!

$$4 \cdot 4 \cdot \ldots \cdot 4 = 4^{C(n,2)} = 4^{n(n-1)/2} = 2^{n(n-1)}$$
Conclude: minimum number of bits to represent simple directed graphs with $n$ vertices is

$$\log_2(2^{n(n-1)}) = n(n-1)$$
Representing directed graphs: Algorithm

**Encoding:**
For each of the n vertices, indicate which of the other vertices it has an edge to.

How would you encode this graph using bits (0s and 1s)?

A. 123232443
B. 0110 0000 0101 0010
C. 110 000 011 001
D. None of the above.
Representing directed graphs: Algorithm

Decoding:

Given a string of 0s and 1s of length $n(n-1)$,

• Define vertex set $\{1, \ldots, n\}$.
• First $n-1$ bits indicate edges from vertex 1 to other vertices.
• Next $n-1$ bits indicate edges from vertex 2 to other vertices.
• etc.

What graph does this binary string encode? 0110 1001 0001 1011 0100
Representing graphs

So far …

Undirected simple graphs on n vertices: at least $n(n-1)/2$ bits.

Directed simple graphs on n vertices: at least $n(n-1)$ bits.

What if we know a little more about the graphs?
Representing **sparse** graphs: Counting

**Sparse graph:** simple directed graph with \( n \) vertices and \( O(n) \) edges.

\[ \text{i.e. Low density of edges compared to number of vertices} \]

How many simple directed graphs with \( n \) vertices and \( m \) edges are there?

A. \( 2^n \)
B. \( 2^{C(n,m)} \)
C. \( C(n(n-1),m) \)
D. \( mn \)
E. None of the above.
Representing **sparse** graphs: Lower bound

**Sparse graph:** simple directed graphs with $n$ vertices and $O(n)$ edges.

Recall the bounds on binomial coefficients: $\binom{n}{k}^k \leq \binom{n}{k} \leq \left(\frac{An}{k}\right)^k$

So number of bits to store a sparse graph is

$$\log_2 \left( \binom{n(n-1)}{m} \right) \geq \log_2 \left( \frac{n(n-1)}{m} \right)^m$$

$$= m \log_2 \left( \frac{n(n-1)}{m} \right)$$
Representing **sparse** graphs: Lower bound

**Sparse graph:** simple directed graphs with $n$ vertices and $O(n)$ edges.

So number of bits to store a sparse graph is

$$\log_2 \left( \binom{n(n-1)}{m} \right) \geq \log_2 \left( \frac{n(n-1)^m}{m} \right)$$

Under the assumption that $m$ is in $\Theta(n)$, the number of bits needed is at least

A. $m \log n$
B. $mn$
C. $n \log n$
D. $n^2$
E. None of the above.
Representing **sparse** graphs: Algorithm

**Sparse graph:** simple directed graphs with $n$ vertices and $O(n)$ edges.

How to represent a sparse graph with **n vertices, m edges** using $O(n \log n)$ bits?
Sparse graph: simple directed graphs with n vertices and O(n) edges.

How to represent a sparse graph with n vertices, m edges using $O(n \log n)$ bits?

For each of the n vertices, initialize a list of its neighbors.

For each of the m edges, store the end point of the edge in the list corresponding to the edge's start point.

It takes $\log n$ bits to represent each vertex from the set of n vertices.

This adjacency list thus uses $O(n \log n)$ bits.
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements

*What's the fastest possible worst case* for any sorting algorithm?
Another application of counting ... lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements

*What's the *fastest possible worst case* for any sorting algorithm?*

**Tree diagram** represents possible comparisons we might have to do, based on relative sizes of elements.
Another application of counting … lower bounds

Tree diagram represents possible comparisons we might have to do, based on relative sizes of elements.
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements.

What's the *fastest possible worst case* for any sorting algorithm?

Maximum number of comparisons for algorithm is *height* of its tree diagram.
What's the height of decision tree diagram above, in terms of n, the number of elements to sort?

A. $2^n$
B. $\log n$
C. $n \log n$
D. $n^2$
E. None of the above.
Another application of counting … lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements.

What's the **fastest possible worst case** for any sorting algorithm?

Maximum number of comparisons for algorithm is **height** of its tree diagram.

For any algorithm, what would be **smallest possible height**?

What do we know about the tree?

* Internal nodes correspond to comparisons.
* Leaves correspond to possible input arrangements.
Another application of counting ... lower bounds

**Sorting algorithm:** performance was measured in terms of number of comparisons between list elements.

What's the **fastest possible worst case** for any sorting algorithm?

Maximum number of comparisons for algorithm is **height** of its tree diagram.

For any algorithm, what would be **smallest possible height**?

What do we know about the tree?
- Internal nodes correspond to comparisons. Depends on algorithm.
- Leaves correspond to possible input arrangements. n!
How does height relate to number of leaves?

**Theorem**: There are at most $2^h$ leaves in a binary tree with height $h$.

Which of the following is true?

A. It's possible to have a binary tree with height 3 and 1 leaf.
B. It's possible to have a binary tree with height 1 and 3 leaves.
C. Every binary tree with height 3 has 1 leaf.
D. Every binary tree with height 1 has 3 leaves.
E. None of the above.
How does height relate to number of leaves?

**Theorem:** There are at most $2^h$ leaves in a binary tree with height $h$.

**Proof:** By *induction* on the height $h \geq 0$.

*Base case* WTS that there are at most $2^0$ leaves in a binary tree with height $h=0$.

*What trees have height 0?*
How does height relate to number of leaves?

**Theorem:** There are at most \(2^h\) leaves in a binary tree with height \(h\).

**Proof:** By induction on the height \(h \geq 0\).

**Base case** WTS that there are at most \(2^0\) leaves in a binary tree with height \(h=0\).

If a binary tree has height 0, its only node is the root. In this case the root is also a (and the only) leaf node. So, the number of leaves is \(1 = 2^0\) in the only possible tree with \(h=0\). 😊
Another application of counting … lower bounds

How does height relate to number of leaves?

**Theorem**: There are at most $2^h$ leaves in a binary tree with height $h$.

**Proof**: By *induction* on the height $h \geq 0$.

**Induction Step** Let $h$ be some integer $\geq 0$ and assume (as the *IH*) that

There are at most $2^h$ leaves in a binary tree with height $h$.

WTS that there are at most $2^{h+1}$ leaves in a binary tree with height $h+1$. 
Another application of counting … lower bounds

*Induction Step* Let h be some integer $\geq 0$ and assume (as the **IH**) that

There are at most $2^h$ leaves in a binary tree with height h.

WTS that there are at most $2^{h+1}$ leaves in a binary tree with height h+1. Consider a tree U with height h+1. *How can we relate it to trees of height h so that we can use IH?*
Another application of counting ... lower bounds

*Induction Step* Let h be some integer \( \geq 0 \) and assume (as the **IH**) that

There are at most \( 2^h \) leaves in a binary tree with height h.

WTS that there are at most \( 2^{h+1} \) leaves in a binary tree with height \( h+1 \).

Consider a tree U with height \( h+1 \).

*How can we relate it to trees of height h so that we can use IH?*

**Remove** all the leaves of U. This gives a new tree, T, of height h.

By the **IH** the tree T has at most \( 2^h \) leaves.

To get from T to U, we need to add back the leaves of U.  *How many are there?*
**Another application of counting … lower bounds**

*Induction Step* Let $h$ be some integer $\geq 0$ and assume (as the IH) that

There are at most $2^h$ leaves in a binary tree with height $h$.

WTS that there are at most $2^{h+1}$ leaves in a binary tree with height $h+1$.

Consider a tree $U$ with height $h+1$. *How can we relate it to trees of height $h$ so that we can use IH?*

**Remove** all the leaves of $U$. This gives a new tree, $T$, of height $h$.

By the IH the tree $T$ has at most $2^h$ leaves.

To get from $T$ to $U$, we need to add back the leaves of $U$. *How many are there?*

For each leaf of $T$, there are **at most 2** leaves in $U$. 
**Induction Step** Let $h$ be some integer $\geq 0$ and assume (as the **IH**) that

There are at most $2^h$ leaves in a binary tree with height $h$.

WTS that there are at most $2^{h+1}$ leaves in a binary tree with height $h+1$.
Consider a tree $U$ with height $h+1$.  How can we relate it to trees of height $h$ so that we can use IH?

**Remove** all the leaves of $U$.  This gives a new tree, $T$, of height $h$.
By the **IH** the tree $T$ has at most $2^h$ leaves.

To get from $T$ to $U$, we need to add back the leaves of $U$.  How many are there?
For each leaf of $T$, there are **at most 2** leaves in $U$.

$$\# \text{ leaves in } U \leq 2(\# \text{ leaves in } T) \leq 2(2^h) = 2^{h+1}$$
Another application of counting … lower bounds

What's the **fastest possible worst case** for any sorting algorithm?

Maximum number of comparisons for algorithm is **height** of its tree diagram.

For any algorithm, what would be **smallest possible height**?

What do we know about the tree?

* Internal nodes correspond to comparisons.  
* **Leaves correspond to possible input arrangements.**  

Each tree diagram must have at least **n! leaves**, so its height must be at least \( \log_2(n!) \).
Another application of counting … lower bounds

What's the fastest possible worst case for any sorting algorithm?

Maximum number of comparisons for algorithm is height of its tree diagram.

For any algorithm, what would be smallest possible height?

What do we know about the tree?

* Internal nodes correspond to comparisons.  
  Depends on algorithm.
* Leaves correspond to possible input arrangements.  
  $n!$

Each tree diagram must have at least $n!$ leaves, so its height must be at least $\log_2(n!)$.  

i.e. fastest possible worst case performance of sorting is $\log_2(n!)$
Another application of counting ... lower bounds

What's the **fastest possible worst case** for any sorting algorithm? $\log_2(n!)$

**How big is that?**

**Lemma**: For $n>1$,

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

**Proof**:

$$n! = (n)(n-1)(n-2)\ldots\left(\frac{n}{2}\right)\ldots(3)(2)(1) > \left(\frac{n}{2}\right)\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)\ldots\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$n! < (n)(n)(n)\ldots(n)(n)(n) = n^n$$
Another application of counting … lower bounds

What's the fastest possible worst case for any sorting algorithm? \( \log_2(n!) \)

How big is that?

**Lemma:** for \( n>1 \), \( \left( \frac{n}{2} \right)^{\frac{n}{2}} < n! < n^n \)

**Theorem:** \( \log_2(n!) \) is in \( \Theta(n \log n) \)

**Proof:** For \( n>1 \), taking logarithm of both sides in lemma gives

\[
\frac{n}{2} \log \left( \frac{n}{2} \right) < \log_2(n!) < n \log n
\]

i.e.

\[
\frac{1}{2} (n \log n - n \log 2) < \log_2(n!) < n \log n
\]
What's the **fastest possible worst case** for any sorting algorithm? \( \log_2(n!) \)

How big is that?

**Lemma:** for \( n > 1 \), \( \left( \frac{n}{2} \right)^{\frac{n}{2}} < n! < n^n \)

**Theorem:** \( \log_2(n!) \) is in \( \Theta(n \log n) \)

**Therefore,**
the best sorting algorithms will need \( \Theta(n \log n) \) comparisons in the worst case.

i.e. it's impossible to have a comparison-based algorithm that does better than **Merge Sort** (in the worst case).
Reminders

HW 7 due **Friday 11:59pm via Gradescope**.

Exam 2 on Tuesday.
- Check the seating chart.
- Practice exam & review sessions available.