Announcements

**HW**
- HW6 due tomorrow
- HW7 due Sunday

**No class Friday**

**Midterm**
- Weds 11/16
- Practice Exam Review Sessions
  See website!

**Office Hours**
- Mine are Friday 10-12 this week (no Saturday.)
- Lots on the course calendar!
**Goal:** encode a length $n$ binary string that we know has $k$ ones (and $n-k$ zeros).

*How would you represent such a string with $n-1$ bits? Can we do better?*

**Idea:** give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.
Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?
Encoding: Fixed Density Strings

**Idea:** give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

**Example** n=12, k=3, window size n/k = 4.

How do we encode s = 01100000010 ?

*There's a 1! What's its position?*

Output:

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.
   - Fix window size.
   - If there is a 1 in the current "window" in the string, record its position and move the window over.
   - Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 01

There's a 1! What's its position?
Idea: give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 01

There's a 1! What's its position?
Encoding: Fixed Density Strings

Idea: give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000010 ? Output: 0100

There's a 1! What's its position?
Idea: give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 0100

No 1s in this window.
**Idea:** give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

**Example** n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?

Output: 01000

*No 1s in this window.*
Encoding: Fixed Density Strings

**Idea:** give positions of 1s in the string within some smaller window.
- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

**Example** n=12, k=3, window size n/k = 4.

How do we encode $s = 011000000010$ ?

Output: 01000

*There's a 1! What's its position?*
**Idea:** give positions of 1s in the string within some smaller window.

- Fix window size.
- If there is a 1 in the current "window" in the string, record its position and move the window over.
- Otherwise, record a 0 and move the window over.

**Example** \( n=12, \ k=3, \text{ window size } n/k = 4. \)

How do we encode \( s = 011000000010 \)?

Output: \( 0100011 \)

*There's a 1! What's its position?*
Idea: give positions of 1s in the string within some smaller window.
   - Fix window size.
   - If there is a 1 in the current "window" in the string, record its position and move the window over.
   - Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010? Output: 0100011

No 1s in this window.
Idea: give positions of 1s in the string within some smaller window.
   - Fix window size.
   - If there is a 1 in the current "window" in the string, record its position and move the window over.
   - Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010__ ? Output: 01000110.

No 1s in this window.
Idea: give positions of 1s in the string within some smaller window.
   - Fix window size.
   - If there is a 1 in the current "window" in the string, record its position and move the window over.
   - Otherwise, record a 0 and move the window over.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 01000110.

Compressed to 8 bits!

But can we recover the original string? Decoding …
Encoding: Fixed Density Strings

**With** $n=12$, $k=3$, window size $n/k = 4$. **Output:** `01000110`

Can be parsed as the (intended) input: $s = 011000000010$?

*But also:*

01: one in position 1
0: no ones
00: one in position 0
11: one in position 3
0: no ones

$s' = 010000100010$

**Problem:** two different inputs with same output. Can't uniquely decode.
A **valid compression algorithm** must:

- Have outputs of shorter (or same) length as input.
- Be uniquely decodable.
Can we modify this algorithm to get unique decodability?

**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.
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.

Example \( n=12, k=3 \), window size \( n/k = 4 \).

How do we encode \( s = 011000000010 \) ? Output:
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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?

Output:
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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?

What output corresponds to these first few bits?
A. 0  
B. 1  
C. 01  
D. 01  
E. None of the above.
Encoding: Fixed Density Strings

**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.

**Example** n=12, k=3, window size n/k = 4.

How do we encode s = \textcolor{red}{01}1000000010 ? Output: 101

Interpret next bits as position of 1; this position is 01
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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000010 ? Output: 101
Encoding: Fixed Density Strings

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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000010 ? Output: 101

A. 101000110  C. 1011000110
B. 1011000110 D. 10110000111
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.

Example $n=12$, $k=3$, window size $n/k = 4$.

How do we encode $s = 01\underline{1000000010}$ ? Output: 101100

Interpret next bits as position of 1; this position is 00
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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 101100
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.

Example n=12, k=3, window size n/k = 4.

How do we encode $s = 011\underline{000}000010$ ? Output: 1011000

No 1s in this window.
Encoding: Fixed Density Strings

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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ? Output: 1011000
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.

Example \( n=12, k=3, \text{ window size } n/k = 4. \)

How do we encode \( s = 011000000010 \) ? Output: 1011000111

Interpret next bits as position of 1; this position is 11
Encoding: Fixed Density Strings

**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.

**Example** $n=12$, $k=3$, window size $n/k = 4$.

How do we encode $s = 01100000001$? Output: 1011000111
Encoding: Fixed Density Strings

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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010__ ? Output: 10110001110

No 1s in this window.
Encoding: Fixed Density Strings

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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 01100000001\_? Output: 10110001110

Compare to previous output: 01000110

Output uses more bits than last time. Any redundancies?
Encoding: Fixed Density Strings

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.

Example n=12, k=3, window size n/k = 4.

How do we encode s = 011000000010 ?

Output: 10110001110

Compare to previous output: 01000110

* Since k is known, after we see the last 1, we can stop since the rest are 0s. *
procedure WindowEncode (input: $b_1 b_2 \ldots b_n$, with exactly $k$ ones and $n-k$ zeros)

1. $w := \text{floor} \ (n/k)$
2. count := 0
3. location := 1
4. While count < $k$:
5. If there is a 1 in the window starting at current location
6. Output 1 as a marker, then output position of first 1 in window.
7. Increment count.
8. Update location to immediately after first 1 in this window.
9. Else
10. Output 0.
11. Update location to next index after current window.

Uniquely decodable?
procedure WindowDecode (input: $x_1x_2...x_m$, target is exactly $k$ ones and $n-k$ zeros)

1. $w := \text{floor} \left( \frac{n}{k} \right)$
2. $b := \text{floor} \left( \log_2(w) \right)$
3. $s := \text{empty string}$
4. $i := 0$
5. While $i < m$
6.     If $x_i = 0$
7.         $s += 0...0$ (j times)
8.         $i += 1$
9.     Else
10.        $p := \text{decimal value of the bits} \ x_{i+1}...x_{i+b}$
11.        $s += 0...0$ (p times)
12.        $s += 1$
13.        $i := i+b+1$
14.        If $\text{length}(s) < n$
15.         $s += 0...0$ (n-length(s) times )
16. Output s.
Encoding/Decoding: Fixed Density Strings

Correctness?

\[ E(s) = \text{result of encoding string } s \text{ of length } n \text{ with } k \text{ 1s, using WindowEncode.} \]

\[ D(t) = \text{result of decoding string } t \text{ to create a string of length } n \text{ with } k \text{ 1s, using WindowDecode.} \]

Well-defined functions?
Inverses?

Can show that for each \( s \), \( D(E(s)) = s \).
Proof uses Strong Induction!
Output size?

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

How long is \( E(s) \)?

A. \( n-1 \)
B. \( \log_2(n/k) \)
C. Depends on where 1s are located in \( s \).
D. None of the above.
Output size?

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

For which strings is E(s) shortest?

A. More 1s toward the beginning.
B. More 1s toward the end.
C. 1s spread about evenly throughout.
Output size?

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

Best case: 1s toward the beginning of the string. \( E(s) \) has
- One bit for each 1 in \( s \) to indicate that next bits denote positions in window.
- \( \log_2(n/k) \) bits for each 1 in \( s \) to specify position of that 1 in a window.
- \( k \) such ones.
- No bits representing empty windows because all 0s are either "caught" in windows with 1s or after the last 1.

Total \( |E(s)| = k \log_2(n/k) + k \)
Output size?

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

**Worst case**: 1s toward the end of the string. $E(s)$ has
- Some bits for empty windows since there are no 1s in first several windows.
- One bit for each 1 in $s$ to indicate that next bits denote positions in window.
- $\log_2(n/k)$ bits for each 1 in $s$ to specify position of that 1 in a window.
- $k$ such ones.

What's an upper bound on the number of these bits?

A. $n$
B. $n-k$
C. 1
D. $k$
**Output size?**

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

**Worst case**: 1s toward the end of the string. \( E(s) \) has
- At most \( k \) bits for empty windows since at most \( k \) nonoverlapping windows of length \( n/k \) will fit in a string of length \( n \).
- One bit for each 1 in \( s \) to indicate that next bits denote positions in window.
- \( \log_2(n/k) \) bits for each 1 in \( s \) to specify position of that 1 in a window.
- \( k \) such ones.

**Total** \(|E(s)| \leq k \log_2(n/k) + 2k\)
Encoding/Decoding: Fixed Density Strings

Output size?

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

\[ k \log_2(n/k) + k \leq |E(s)| \leq k \log_2(n/k) + 2k \]

Using this inequality, there are at most ____ length $n$ strings with $k$ 1s.

A. $2^n$  
B. $n$  
C. $(n/k)^2$  
D. $(n/k)^k$  
E. None of the above.

See next slide.
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 $2^{(k \log(n/k)+2k)}$ many such strings.

\[
2^{(k \log(n/k)+2k)} = 2^{(k \log(n/k))} \cdot 2^{(2k)}
\]

\[
= \left(2^{(\log(n/k))}\right)^k \cdot 2^{(2k)}
\]

\[
= (n/k)^k \cdot 4^k = (4n/k)^k
\]
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 \( 2^{2(k \log(n/k) + 2k)} \) many such strings.

\[
2^{(k \log(n/k) + 2k)} = 2^{(k \log(n/k))} \cdot 2^{(2k)}
\]

\[
= \left(2^{(\log(n/k))}\right)^k \cdot 2^{(2k)}
\]

\[
= (n/k)^k \cdot 4^k = (4n/k)^k
\]

\[
C(n,k) = \# \text{ Length } n \text{ binary strings with } k \text{ 1s } \leq (4n/k)^k
\]
Bounds for Binomial Coefficients

Using `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. \( C(n,k)^k \)  
E. None of the above.
Bounds for Binomial Coefficients

Using `windowEncode()`:

\[
\binom{n}{k} \leq (4n/k)^k
\]

Using evenly spread strings:

\[
(n/k)^k \leq \binom{n}{k}
\]

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 $\log_2 \binom{n}{k}$ 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 …
Lex Order

String *a* comes **before** string *b* if the **first time they differ**, *a* is smaller.

I.e.

\[ a_1 a_2 \ldots a_n <_{\text{lex}} b_1 b_2 \ldots b_n \]

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 **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>

\[
\binom{n-1}{k} = \binom{n}{k} - \binom{n}{k-1}
\]
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.

Key insight: In lex order, strings that start with 0 come before strings that start with 1.
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 at or after \( \binom{n - 1}{k} \).

Length \( n-1 \) binary strings with \( k \) 1s

Length \( n-1 \) binary strings with \( k-1 \) 1s
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 at or after $\binom{n-1}{k}$.

procedure lexEncode ($b_1 b_2 \ldots b_n$, 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 $C(n-1,k) + \text{lexEncode}(b_2 \ldots b_n, n-1, k-1)$

Recursive
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)

For D(s,n,k):
• Any position before \( \binom{n-1}{k} \) must correspond to string that starts with 0.
• Any position at or after \( \binom{n-1}{k} \) must correspond to string that starts with 1.
Using \textit{lexEncode}, \textit{lexDecode}, we can represent any fixed density length $n$ binary string with $k$ 1s as one of $C(n,k)$ positions.

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

\textbf{Theoretical lower bound}: $\log_2( C(n,k) )$.

Same! So this encoding algorithm is optimal.
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 for a sorting algorithm** represents possible comparisons we might have to do, based on relative sizes of elements.

Sometimes called a decision tree
Another application of counting … lower bounds

Tree diagram for a sorting algorithm

Based on the result of comparisons, travel from root to leaf (ex.)

A, B, C distinct integers

Rosen p. 761
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 (worst-case) number of comparisons for a sorting algorithm is the **height** of its tree diagram.
Another application of counting … lower bounds

How many leaves will there be in a decision tree that sorts n elements?

A. $2^n$
B. $\log n$
C. $n!$
D. $C(n,2)$
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?

max # of comparisons = **height** of 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?

\[
\text{max # of comparisons} = \text{height of tree diagram}
\]

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.

\[n!\]
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 $h$.

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.
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 $h$.

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.

Proof by induction on $h$
Another application of counting … lower bounds

What's the \textbf{fastest possible worst case} for any sorting algorithm?

max # of comparisons = \textbf{height} of tree diagram

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:

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

$$> \left(\frac{n}{2}\right)\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)\ldots\left(\frac{n}{2}\right)$$

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

$n! = (n)(n-1)(n-2)\ldots(3)(2)(1)$

$$< (n)(n)(n)\ldots(n)(n)(n)$$

$$= n^n$$

replace each term with a smaller term

replace each term with a bigger term
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 \( \Theta(n \log n) \) \( \leftarrow \) bounded on both sides by some factor of \( n \log n \)

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

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

\[\frac{1}{2} (n \log n - n \log 2) < \log_2(n!) < n \log n\]
Another application of counting … lower bounds

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

How big is that? \( \Theta(n \log n) \)

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

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

quicksort

tree sorting alg. from HW5 \( \Rightarrow \) also \( n \log n \) sorts
**Announcements**

**HW**
- HW6 due tomorrow
- HW7 due Sunday

**No class Friday**

**Midterm**
- Weds 11/16
- Practice Exam
- Review Sessions
  - See website!

**Office Hours**
- Mine are Friday 10-12 this week (no Saturday.)
- Lots on the course calendar!