## Encoding and Decoding, Lower Bounds

<table>
<thead>
<tr>
<th>Lecture A</th>
<th>Tiefenbruck</th>
<th>MWF 9-9:50am</th>
<th>Center 212</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture B</td>
<td>Jones</td>
<td>MWF 2-2:50pm</td>
<td>Center 214</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Tiefenbruck</td>
<td>MWF 11-11:50am</td>
<td>Center 212</td>
</tr>
</tbody>
</table>

http://cseweb.ucsd.edu/classes/wi16/cse21-abc/

February 22, 2016
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:
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 = 01\underline{1}000000010 ? Output: 101

Interpret next bits as position of 1; this position is 01
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 \( s = 01\underline{0000000}10 \) ? \hspace{1cm} Output: 101
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: 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 = 011\underline{0000}00010 ? 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 = 01100000010 ?

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
**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: $1011000111$

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 = 011000000010 ?  
Output: 1011000111

Now we can stop recording, since we have seen all three ones.
procedure WindowEncode (input: $b_1b_2...b_n$, with exactly $k$ ones and $n-k$ zeros)

1. $w := \text{floor} \ (n/k)$
2. $\text{count} := 0$
3. $\text{location} := 1$
4. While $\text{count} < k$:
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. \(s += 0...0\) (\(w\) 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. \(s += 0...0\) (\(n-\text{length}(s)\) times)
16. Output \(s\).
Correctness?

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

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

Well-defined functions?

Inverse?

Goal: For each s, \( D(E(s)) = s \).

Strong Induction!
Output size?

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

How many bits is \( E(s) \)?

A. \( n-1 \)
B. \( \log_2(n/k) \)
C. Depends on where 1s are located in \( s \)
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 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 1s.
- No bits representing 0s because all 0s occur 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 representing 0s 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 1s.

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

A. \( n \)        D. 1
B. \( n-k \)      E. None of the above.
C. \( k \)
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 representing 0s 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.

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

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

$$k \log_2(n/k) + k \leq |E(s)| \leq k \log_2(n/k) + 2k$$
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 $2^{(k \log_2(n/k)+2k)}$ many such strings.

\[
2^{(k \log_2(n/k)+2k)} = 2^{(k \log(n/k))} \cdot 2^{(2k)}
\]

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

\[
= (n/k)^k \cdot 4^k = (4n/k)^k
\]
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 \( 2^{k \log_2(n/k) + 2k} \) many such strings.

\[
2^{(k \log_2(n/k) + 2k)} = 2^{(k \log_2(n/k))} \cdot 2^{2k}
\]

\[
= \left(2^{\log_2(n/k)}\right)^k \cdot 2^{2k}
\]

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

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

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

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 $\log_2 \binom{n}{k}$ 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.
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 lexicographical (dictionary) ordering ...
Lex Order

String $a$ comes before string $b$ if the 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 last in lex order?

A. 1001        C. 1101        E. 0000
B. 0011        D. 1010
Lex Order

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 at or 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 at or after $\binom{n-1}{k}$

procedure lexEncode ($b_1b_2...b_n$, n, k)

1. If $n = 1$,
2. return 0.
3. If $s_1 = 0$,
4. return lexEncode ($b_2...b_n$, n-1, k)
5. Else
6. return $C(n-1,k) +$ lexEncode($b_2...b_n$, 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 at or after $\binom{n-1}{k}$ must correspond to string that starts with 1.

```
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 0 through $C(n,k) - 1$.

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.
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.

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 number of comparisons for algorithm is **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.
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 $h+1$. 
**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.  
- *Leaves correspond to possible input arrangements.*

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

i.e. fastest possible worst case performance of sorting is **\( \log_2(n!) \)**
What's the **fastest possible worst case** for any sorting algorithm? \( \log_2(n!) \)

**How big is that?**

**Lemma:** For \( n > 1 \),

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

**Proof:**

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

\[
n! = (n)(n-1)(n-2) \ldots (3)(2)(1)
\]
\[
< (n)(n)(n) \ldots (n)(n)(n)
\]
\[
= n^n
\]
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.

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

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

How big is that?

**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 **Wednesday 11:59pm via Gradescope.**

**Midterm 2:** one week from today, Monday, February 29 in class

* Practice midterm on website/Piazza.
* Review sessions Thursday & Saturday: see website/Piazza.
* **Seating chart on website/Piazza.**
* One double-sided handwritten note sheet allowed.
* If you have AFA letter, see me as soon as possible.