## Encoding and Decoding

<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 10, 2015
A **permutation** of \( r \) elements from a set of \( n \) distinct objects is an **ordered** arrangement of them. There are

\[
P(n,r) = n(n-1)(n-2) \cdots (n-r+1)
\]

many of these.

A **combination** of \( r \) elements from a set of \( n \) distinct objects is an **unordered** selection of them. There are

\[
C(n,r) = \frac{n!}{r!(n-r)!}
\]

many of these.
How many length $n$ binary strings contain $k$ ones?

**Objects**: all strings made up of $0_1, 0_2, 1_1, 1_2, 1_3, 1_4$  \[ n! \]

**Categories**: strings that agree except subscripts

**Size of each category**:  \[ k!(n-k)! \]

**# categories** = (# objects) / (size of each category)

\[ = n! / (k! (n-k)!) = C(n,k) = \binom{n}{k} \]
What's the smallest number of bits that we need to specify a binary string if we know it has \( k \) ones and \( n-k \) zeros?

A. \( n \)  
B. \( k \)  
C. \( \log_2(\binom{n}{k}) \)  
D. ??
Data Compression

Store / transmit information in as little space as possible
Data Compression: Video

**Video:** stored as sequence of still frames.

**Idea:** instead of storing each frame fully, record change from previous frame.
Data Compression: Run-Length Encoding

Image: described as grid of pixels, each with RED, GREEN, BLUE values.

Idea: instead of storing RGB value of each pixel, store run-length of run of same color.

When is this a good coding mechanism? Will there be any loss in this compression?
**Image**: described as grid of pixels, each with **RED**, **GREEN**, **BLUE** values.

**Idea**: use Linear Algebra to compress data to a fraction of its size, with minimal loss.
Complicated compression scheme

… save storage space
… may take a long time to encode / decode
Encoding: Binary Palindromes

**Palindrome:** string that reads the same forward and backward.

Which of these are binary palindromes?

A. The empty string.
B. 0101.
C. 0110.
D. 101.
E. All but one of the above.
Encoding: Binary Palindromes

**Palindrome**: string that reads the same forward and backward.

How many length n binary palindromes are there?

A. $2^n$
B. $n$
C. $n/2$
D. $\log_2 n$
E. None of the above
Palindromes: string that reads the same forward and backward.

How many bits are (optimally) required to encode length n binary palindromes?

A. n
B. n-1
C. n/2
D. \( \log_2 n \)
E. None of the above.

Is there an algorithm that achieves this?
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?
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?
**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.
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 ?
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 ?

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 $s = 01100000010$?  
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 = 01100000010 ? 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 = 01\textcolor{red}{10000000}10 ? 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 \( s = 011\underline{000}00010 \)?

Output: 0100

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 $s = 011\underline{000}00010$ ?

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?*
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: 0100011

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 $s = 011000000010$ ?
Output: 0100011

*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: 01000110.

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: 01000110.

Compressed to 8 bits!

*But can we recover the original string? Decoding …*
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:
**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 $s = 01\underline{1}0000000010$? Output:

What output corresponds to these first few bits?

A. 0  
B. 1  
C. 01  
D. 101  
E. None of the above.
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: 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 = 011000000010 ? 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: 101100

Interpret next bits as position of 1; this position is 00
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 $s = 011\underline{000}00010$ ? 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
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
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
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 \( s = 011000000010 \) ?  

Output: 10110001110

Compare to previous output: 01000110

**Output uses more bits than last time. Any redundancies?**
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

* After see the last 1, don't need to add 0s to indicate empty windows. *
procedure WindowEncode (input: b₁b₂...bₙ, with exactly k ones and n−k zeros)

1. w := 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?
Decoding: Fixed Density Strings

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 length($s$) < $n$
15.      $s += 0...0$ (n-length($s$) times )
16. Output $s$. 
Correctness?

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

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

*Well-defined functions?*
*Inverses?*

**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 long 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 ones.
- No bits representing 0s because all 0s are "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 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.

What's an upper bound on the number of these bits?
A. $n$
B. $n-k$
C. $k$
D. 1
E. 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.

**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.

\[
k \log_2(\frac{n}{k}) + k \leq |E(s)| \leq k \log_2(\frac{n}{k}) + 2k
\]

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.
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.
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 \( 2^{2k} \) many such strings.

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

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

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

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

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

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

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

Using evenly spread strings:

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

\textbf{Counting} helps us analyze our \textbf{compression algorithm}.

\textbf{Compression algorithms} help us \textbf{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.
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 …
String *a* comes **before** string *b* if the **first time they differ**, *a* is smaller.

I.e.

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

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  C. 1101  E. 0000
B. 0011  D. 1010
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 $\binom{n - 1}{k}$ and before $n$
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 $\binom{n-1}{k}$ and before $n$.

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 $\binom{n-1}{k} + \text{lexEncode}(b_2 \ldots b_n, n-1, k-1)$
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.

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)
**Victory!**

**Using lexEncode, lexDecode,** we can represent any fixed density length $n$ binary string with $k$ 1s as a number in the range $1 \ldots C(n,k)$.

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.
Reminders

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