Two more days!

HW6 extended

Now due Tuesday at 10pm

HW7 due Sunday 11/13.
Midterm 2 on Weds 11/16.
Recall: Fixed-density Binary Strings

How many length $n$ binary strings contain $k$ ones?

Density is number of ones

**Objects**: all strings made up of $0_1$, $0_2$, $1_1$, $1_2$, $1_3$, $1_4$  

**Categories**: strings that agree except subscripts

**Size of each category**: $k!(n-k)!$

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

= $n! / (k! (n-k) !) = C(n,k) = \binom{n}{k}$
Binomial Coefficient Identities

What's an identity? An equation that is always true.

To prove

\[(x + 1)^2 = x^2 + 2x + 1\]

LHS = RHS

• Use algebraic manipulations of formulas.

OR

• Interpret each side as counting some collection of strings, and then prove a statements about those sets of strings.
Theorem: \[ \binom{n}{k} = \binom{n}{n-k} \]
Symmetry Identity

Theorem: \( \binom{n}{k} = \binom{n}{n-k} \)

Proof 1: Use formula

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}
\]
Symmetry Identity

Theorem: \( \binom{n}{k} = \binom{n}{n-k} \)

Proof 1: Use formula

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}
\]

Proof 2: Combinatorial interpretation?

**LHS** counts number of binary strings of length \( n \) with \( k \) ones

**RHS** counts number of binary strings of length \( n \) with \( n-k \) ones

Rosen p. 411
Symmetry Identity

Theorem: \( \binom{n}{k} = \binom{n}{n-k} \)

Proof 1: Use formula

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k}
\]

Proof 2: Combinatorial interpretation?

- **LHS** counts number of binary strings of length n with k ones and n-k zeros
- **RHS** counts number of binary strings of length n with n-k ones and k zeros

*Rosen p. 411*
Theorem: \[ \binom{n}{k} = \binom{n}{n-k} \]

Proof 1: Use formula
\[ \binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n!}{(n-k)!k!} = \binom{n}{n-k} \]

Proof 2: Combinatorial interpretation?

**LHS** counts number of binary strings of length n with k ones and n-k zeros

**RHS** counts number of binary strings of length n with n-k ones and k zeros

Can match up these two sets by pairing each string with another where 0s, 1s are flipped. This *bijection* means the two sets have the same size. So **LHS = RHS**.
Pascal's Identity

Theorem: \[ \binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k} \]

Proof 1: Use formula

Proof 2: Combinatorial interpretation?

LHS counts number of binary strings ???
RHS counts number of binary strings ???
Pascal's Identity

Theorem: \[ \binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k} \]

Proof 2: Combinatorial interpretation?

**LHS** counts number of binary strings of length n+1 that have k ones.

**RHS** counts number of binary strings ???

Length n+1 binary strings with k ones
Theorem: \[ \binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k} \]

Proof 2: Combinatorial interpretation?

**LHS** counts number of binary strings of length $n+1$ that have $k$ ones.

**RHS** counts number of binary strings ???

Rosen p. 418

Start with 1
Start with 0
How many length $n+1$ strings start with 1 and have $k$ ones in total?

A. $C(n+1, k+1)$
B. $C(n, k)$
C. $C(n, k+1)$
D. $C(n, k-1)$
E. None of the above.

Pascal's Identity

Rosen p. 418
How many length \( n+1 \) strings start with 0 and have \( k \) ones in total?

A. \( \binom{n+1}{k+1} \)

B. \( \binom{n}{k} \)

C. \( \binom{n}{k+1} \)

D. \( \binom{n}{k-1} \)

E. None of the above.

Pascal's Identity

Rosen p. 418
Pascal's Identity

Theorem: \( \binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k} \)

Proof 2: Combinatorial interpretation?

**LHS** counts number of binary strings of length n+1 that have k ones.
**RHS** counts number of binary strings of length n+1 that have k ones, split in two.

Start with 1
Start with 0

Rosen p. 418
Sum Identity

Theorem: \[ \sum_{k=0}^{n} \binom{n}{k} = 2^n \]

What set does the LHS count?

A. Binary strings of length n that have k ones.
B. Binary strings of length n that start with 1.
C. Binary strings of length n that have any number of ones.
D. None of the above.
Theorem:
\[ \sum_{k=0}^{n} \binom{n}{k} = 2^n \]

Proof: Combinatorial interpretation?

**LHS** counts number of binary strings of length n that have any number of 1s.

By sum rule, we can break up the set of binary strings of length n into disjoint sets based on how many 1s they have, then add their sizes.

**RHS** counts number of binary strings of length n.

This is the same set so **LHS = RHS**.
What's the **smallest** number of **bits** that we need to specify a binary string of length **n** if we know it has **k ones** and **n-k zeros**?

\[ \# \text{ bits} = \log_2 \left( \binom{n}{k} \right) \]

A. \( n \)
B. \( k \)
C. \( \log_2(\binom{n}{k}) \)
D. \( n-k \)
E. None of the above
Data Compression

Store / transmit information in as little space as possible
Video: stored as sequence of still frames.

Idea: instead of storing each frame fully, record change from previous frame.
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?
**Lossy Compression: Singular Value Decomposition**

**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.

\[ \text{e.g., } n = 3, \ 0 \ 1 \ 0, \ 0 \ 1 \ 0, \ 1 \ 1 \ 1, \ 1 \ 1 \ 1 \]

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
**Encoding: Binary Palindromes**

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

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

A. \( n \)
B. \( n-1 \)
C. \( \lceil n/2 \rceil \)
D. \( \log_2 n/2 \)

Is there an algorithm that achieves this?
Encoding: Fixed Density Strings

**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?*

*Ex.:* $n = 7$, $k = 4$

Store first 6 bits can figure out last bit from knowing $k$ (total # of ones)
Encoding: Fixed Density Strings

**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? What if we know \( k \) is much less than \( n \)?*

- **Example:** \( n = 15, \ k = 3 = \# \text{ of ones} \)
  - For each 1, give its position as a 4-bit number
  - \( \log_2 15 \) = 4 bits needed
  - \( 3 \times 4 = 12 \) bits total

---

**Note:**
- Only store positions of ones.
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? What if we know k is only slightly less than n?

opposite - store positions of 0s
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? What if k is about half of n?

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 ?
**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:

```
0 1 1 0
00 01 10 11
```
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?
**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 $s = 01\underline{1}00000010$? 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\underline{1}00000010 ? Output: 0100

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: 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 = 01100000010 ? 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.
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.
**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:
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:
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 $s = \underline{011000000010}$? 

**Output:**

What output corresponds to these first few bits?

<table>
<thead>
<tr>
<th>Option</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. 0</td>
<td>C. 01</td>
</tr>
<tr>
<td>B. 1</td>
<td>D. 101</td>
</tr>
</tbody>
</table>

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
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 = 011000000010 ? 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: 101

A. 101000110
B. 10110001110
C. 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 \( s = 01100000010 \) ? 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 = 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 = 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 = 011000000101__ ? Output: 1011000111
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: 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: $b_1b_2\ldots b_n$, 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: \(x_1 \ldots x_m\), target is exactly \(k\) ones and \(n-k\) zeros)

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

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

D(t) = result of decoding string t to create a string of length n with k 1s, using \texttt{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 \( \frac{n}{k} \) 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 \( \frac{n}{k} \) will fit in a string of length \( n \).
- One bit for each 1 in \( s \) to indicate that next bits denote positions in window.
- \( \log_2(\frac{n}{k}) \) bits for each 1 in \( s \) to specify position of that 1 in a window.
- \( k \) such ones.

**Total** \( |E(s)| \leq k \log_2(\frac{n}{k}) + 2k \)
Output size?

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

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

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.
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.

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

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

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

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

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

HW6 extended
Now due Tues 10pm
HW7 due Sunday 11/13.
Midterm 2 on Weds 11/16.

Office Hours
Mine are today 10-11
and tomorrow 3:30-5:30.
Lots on the course calendar!