Maximum score: 230 points.
Instructions: For this test, you work in teams to solve a multi-part, proof-oriented question. Problems that use the words "compute," "list," or "draw" require only an answer; no explanation or proof is needed. Unless otherwise stated, all other questions require explanation or proof. The problems are ordered by content, *not difficulty*. The difficulties of the problems are generally indicated by the point values assigned to them; it is to your advantage to attempt problems throughout the test. In your solution for a given problem, you may cite the statements of earlier problems (but not later ones) without additional justification, even if you haven't solved them. Footnotes are not necessary to understand the contents of the round. **Put your Team ID at the top of every page. Otherwise, we may be unable to grade your exam. No Calculators Allowed.**

1 Error Correcting Codes: A Tale of Bob and Alice (26 pts)

Consider a situation much too common in our modern world: a remote math contest participant, Alice wants to send her answers to Bob, a staff member of the math contest. There is a "channel" between them, which we can think of as something that Alice can send data across as a bunch of characters. Bob can then receive these characters. The set of possible characters has a name.

Definition 1.1. An **alphabet** is some set Σ of characters. Common alphabets include

- $\Sigma_2 = \{0, 1\}$
- $\Sigma_{10} = \{0, 1, 2, \ldots, 9\}$
- $\Sigma_{\alpha\beta} = \{\texttt{A}, \texttt{B}, \texttt{C}, \ldots, \texttt{Z}\}$

Before she can answer any questions, Alice has to send her name. Beforehand, the pair agree to use the alphabet $\Sigma_{\alpha\beta}$. Alice types in the **codeword**

A L I C E

and sends it through the channel.
Now, Bob is receiving Alice's answer. At the other end he sees:

A L _ _ E

Oh no! Our channel has "erased" the 3rd and 4th characters into underscores (_'s are not in our alphabet). This is a fact of life, as real-world channels often are not reliable enough to send every character perfectly. Let's say that Alice and Bob determine the following empirically: on any given transmission, we are guaranteed for at most two characters to be erased (but we don't know beforehand which ones). Luckily, Alice has a great idea that had she shared with Bob beforehand.

Scheme 1.1. Just like how at a loud party, you have to repeat your message for people to get it, Alice will duplicate her message 3 times, sending:

A L I C E A L I C E A L I C E

Now, Bob may receive

A L I _ E A L _ C E A L I C E

Then Bob does the following:

1. Bob receives a message of length n and divides n by 3 to get the length of the original word, which we denote $\ell = n/3$. In our example, this means he knows the message has length 5.
2. Bob places a marker at every ℓth letter. In our example he places markers in front of all 3 A's.
   (a) He compares all 3 of the letters at this position. He picks a letter that is not an underscore (all the non-underscore letters are necessarily the same).
   (b) He adds this letter to our draft message.
   (c) He advances to the next position and repeat this ℓ times.
3. At the end, our "draft message" is the message we recover.

If we run our scheme on the example:

1. We see 3 A's, so our first letter is A
2. We see 3 L's, so our second letter is L
3. We see 2 I's and 1 _, so our third letter is I
4. We see 2 C's and 1 _, so our fourth letter is C
5. We see 3 E's, so our fifth letter is E

This means Bob recovers A L I C E !
The scheme works because only at most 2 characters can be erased, so if you get to a position where 2 of the regions have an underscore, the third region will still have the letter.

Question 1.1 (3 pts)**.** Suppose now, instead of the channel "erasing" 2 characters, 2 characters are "corrupted" into random characters that are in the alphabet. In the example above, Bob may instead receive:

A L I X E A L Y C E A L I C E

Modify Scheme 1.1 such that step 2 (a) reads:

- He compares all 3 of the letters at this position, picking the one that is the "majority." For example, if the three letters are A, A, C, then Bob would pick A.

Does the scheme still work? If it does, give a proof. If not, give a counter-example.
No, suppose characters 1 and 6 were corrupted to B, then BLICE would be recovered.

Question 1.2 (4 pts)**.** Suppose that we had k characters erased and Alice's message has length g. What is the minimum amount of times t Alice needs to repeat her message to allow Scheme 1.1 to work (replacing 3 in the scheme with t)?

The worst-case scenario is when all of the erasures are at the same place. Thus, $t = k+1$ is necessary.

Let's define what has happened formally.

Definition 1.2. A **set** is a collection of elements with no duplicates. The following examples use the sets from Definition 1.1.

- To denote the size of any set, we use $|\cdot|$. For example, $\left|\Sigma_{\alpha\beta}\right| = 26$.
- To denote an element being a member of a set, we use $\in$. To denote an element not being a member of a set, we use $\notin$. For example, $3 \in \Sigma_{10}$ but $3 \notin \Sigma_2$.
- To denote subsets (i.e. every element of one set is also an element of the other), we use $\subseteq$. For example, $\Sigma_2 \subseteq \Sigma_{10}$.
- To denote sets of ordered tuples or sequences, we use $\times$. For example,

$$\Sigma_{10} \times \Sigma_2 = \{(0,0),(0,1),(1,0),(1,1),\ldots,(9,0),(9,1)\}$$

- To denote using the $\times$ on a set with itself, we use exponents. For example,

$$\Sigma_2^3 = \Sigma_2 \times \Sigma_2 \times \Sigma_2 = \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),\ldots,(1,1,0),(1,1,1)\}$$

Definition 1.3. A **code** C **of length** n is some subset of Σ^n. Each element of C is called a **codeword**. The elements of Σ^n are just called **words**.

Using the sets in Definition 1.1,

Question 1.3 (3 pts). Compute $\left|\Sigma_2^m\right|$ in terms of m.

$|\Sigma_2^m| = 2^m$.

Question 1.4 (4 pts). Compute the number of codes over alphabet Σ_{10} with codeword length n.

This is the total amount of subsets of Σ_{10}^n, which is: $2^{|\Sigma_{10}^n|} = 2^{10^n}$

Question 1.5 (5 pts). Compute the number of codes over alphabet $\Sigma_{\alpha\beta}$ with the following properties:

1. The length of each codeword is 10.
2. Each codeword has the substring 'SUSBUS' (where the letters are contiguous, in one block).

You may leave your answer in terms of exponents and binomial coefficients.

In a string of length 10, there are 5 positions where 'SUSBUS' can go and then 4 characters that can be anything (there isn't enough room in the string to have another 'SUSBUS' and worry about double-counting). Thus, the total amount of codewords that satisfy both these properties is $5 \cdot 26^4$, so the total amount of subsets is $2^{5 \cdot 26^4}$.

Definition 1.4. Consider an encoding setup with alphabet Σ and code $C \subseteq \Sigma^n$. The **message** m is the meaningful information we want to send (like Alice's name). From the time we have a message to the time the person at the other end of the channel receives it, we undergo three different stages.

- **Encoding** - Encoding function E takes a message m and gives its corresponding codeword $E(m)$.
- **Transmission** - The message passes through a transmission channel T, which takes a codeword c in C and returns a string c' in Σ^n.
- **Decoding** - Decoding function D takes an element w of Σ^n (not necessarily a valid codeword) and gives a corresponding message $D(w)$.

This means for Scheme 1.1:

- $C =$ the set of all words that are a message repeated three times.
  - So $'\texttt{ALICEALICEALICE}' \in C$ but $'\texttt{ALICEALICEBOBBY}' \notin C$.
- $E(m) = (m, m, m)$ (our "codeword" is our original message repeated 3 times)
- T takes c and returns c with 2 random characters erased
- $D(w) =$ the result of running Scheme 1.1 on w

Question 1.6 (2 pts). Give an expression for the final message received given an initial message m which passes through the encoding, transmission, and decoding stages in Definition 1.4. For this question, let $T(c)$ denote the output of the transmission channel on some codeword c.

$D(T(E(m)))$

Question 1.7 (5 pts). Suppose that T doesn't change the codeword at all (it takes codeword c and returns c) and we choose E and D such that our final message always matches our initial message m. Prove that if $x \neq y$, then $E(x) \neq E(y)$.

Proof by contraposition. By the given, we have $D(E(m)) = m$. Suppose $E(x) = E(y)$. Then $D(E(x)) = D(E(y))$ or $x = y$.

2 The Hamming Bound and Hamming Codes (59 pts)

2.1 Distances and the Hamming Bound

How can we compare the efficacy of schemes? Well, one answer is to look at a code's **distance**, which tells us how much wiggle room there is between codewords.

Definition 2.1. The **Hamming distance** between two words $x \in \Sigma^n$ and $y \in \Sigma^n$ is the number of places they differ. We will denote this by $\delta(x, y)$.
For a code C, its **distance** $d(C)$ is the minimum Hamming distance between any two distinct codewords. Stated mathematically,

$$d(C) = \min_{\substack{c,c' \in C \\ c \neq c'}} \delta(c, c')$$

In addition, the **Hamming Ball of radius** r **centered at** c, denoted $B_c(r)$, is the set of all words that are Hamming distance r or less from c (including c itself).

For example, if

$$x = \texttt{A L I C E} \qquad y = \texttt{A L I X Y} \qquad z = \texttt{A D I Y X}$$

then $\delta(x, y) = 2$ and $\delta(y, z) = 3$.

Question 2.1 (5 pts).

Show that Hamming distance satisfies the triangle inequality: for any three words x, y, and z,

$$\delta(x, y) + \delta(y, z) \geq \delta(x, z).$$

Let $I_{a,b}$ be the set of places where words a, b differ. The only possibilities for $I_{x,y}, I_{y,z}$ that could cause x and z to be the differ at a place are if at least one of $(x, y), (y, z)$ differs. Note that this is an upper bound on the number of difference; if they both differ x and y can still be the same. Furthermore, an upper bound on this quantity is $|I_{x,y}| + |I_{y,z}|$, when there is no overlap between sets. Thus $|I_{x,z}| \leq |I_{x,y}| + |I_{y,z}|$. Note that $|I_{a,b}| = \delta(a, b)$, so we have shown the result.

Question 2.2 (3 pts).

The **0-1 distance** between two words is:

$$\delta_{0-1}(x, y) = \begin{cases} 0 & x = y \\ 1 & x \neq y \end{cases}$$

For any two messages x, y, which of the following is always true? Explain your reasoning.

(a) $\delta_{0-1}(x, y) \leq \delta(x, y)$
(b) $\delta_{0-1}(x, y) = \delta(x, y)$
(c) $\delta_{0-1}(x, y) \geq \delta(x, y)$
(d) None of (a), (b), or (c)

(a) is always true. If two codewods differ and $\delta_{0-1}(x, y) = 1$, they must differ in at least one place so $\delta(x, y) \geq 1$. If two codewords are the same, both distances are 0, so the inequality still holds.

Now for Alice and Bob's code, what is $d(C)$? Well, since the message is repeated three times, you have to change all the letters at the same position for the message to be part of C. This means any two distinct codewords in C must differ by at least 3 characters. This tells us $\delta(c, c') \geq 3$ for all $c, c' \in C$ such that $c \neq c'$, so $d(C) \geq 3$ as well. On the other hand,

A L I C E A L I C E A L I C E and A L I C T A L I C T A L I C T

are code-words that differ in exactly three places, so the minimum distance over all code-words $d(C)$ must be at most 3. Thus, $d(C) = 3$.

Definition 2.2. An $[n, k]$ code is a code of length n, message length k. If we know the distance d, then we can also call it a $[n, k, d]$ code.

So the repetition code (Scheme 1.1) we have studied is a $[15, 5]$ or $[15, 5, 3]$ code. Note that with this distance, the code was able to fix 2 characters erased. This is no accident. In fact, we can show the following:

Consider a code C with odd minimum distance $d(C)$.

Question 2.3 (6 pts)**.** Show that we can always correct erasures of up to $d(C) - 1$ characters. (In other words, suppose we have a partially erased word $\tilde{c}$ that is the result of erasing up to $d(C) - 1$ bits from codeword c. Show that the only codeword such that erasing at most $d(C) - 1$ characters gives $\tilde{c}$ is c.) Explain why $d(C)$-character erasures cannot always be corrected.

Consider an original codeword c and a partially erased word $\tilde{c}$ which is the result of erasing $s \leq d(C) - 1$ characters. Suppose you could fill in the erased characters of $\tilde{c}$ in a different way to make a codeword $c' \neq c$. Then we must have $\delta(c, c') \leq s \leq d(C) - 1$. But this contradicts the minimum distance of C, so this must not be possible. Note that two codewords c_1, c_2 can differ by $d(C)$ and if you erase the positions they differ by then you end up with the same word and it's impossible to tell them apart.

Question 2.4 (7 pts)**.** Show that we can always correct corruptions of up to $\frac{d(C)-1}{2}$ characters. (*Hint*: Suppose a corrupted word $\tilde{c}$ is distance less than $\frac{d(C)-1}{2}$ from two distinct codewords c and c'. What would this mean?)

Consider an original codeword c and a corrupted word $\tilde{c}$ which is the result of erasing $e \leq \frac{d(C)-1}{2}$ characters. Suppose there was another codeword c' such that $\delta(\tilde{c}, c') \leq \frac{d(C)-1}{2}$ (which would mean the corrupted versions are indistinguishable). Then by the triangle inequality:

$$\delta(c, c') \leq \delta(c', \tilde{c}) + \delta(\tilde{c}, c) \leq \frac{d(C) - 1}{2} + e \leq \frac{d(C) - 1}{2} + \frac{d(C) - 1}{2} = d(C) - 1$$

which is a contradiction of minimum distance.

The question remains: if all that matters is the distance, what is the best code for the job? One answer may be the code with the most codewords, because then we can represent the most messages.

Consider a code C with length n and $d(C) = 3$ over the alphabet $\Sigma_2 = \{0, 1\}$.

Question 2.5 (2 pts)**.** How many possible words are there of length n over the alphabet Σ_2?

2^n

Question 2.6 (2 pts)**.** Consider some $c \in C$. What is $|B_c(1)|$? (Here, $B_c(1)$ denotes the Hamming ball of radius 1 centered at c; see Definition 2.1.)

The amount of distance 0 words from c is 1, only c itself. For distance 1 words, there are n places we can differ and there is $2 - 1 = 1$ characters we can place in that difference, so there are $n \cdot 1$ words here. In total, $n + 1$.

Question 2.7 (6 pts)**.** Question 2.4 shows that we can fix a 1-character corruption, i.e. the balls of radius 1 about every codeword in C will not overlap (the one corresponding to the codeword we are decoding to). Use this to show:

$$|C| \leq \frac{2^n}{n+1}.$$

Each $B_c(1)$ is the ball containing 1-character corrupted words from codeword c. They are disjoint by Question 2.4, since such a corruption is correctable. But there is one ball for each codeword $c \in C$ so there are $|C|$ balls, and each ball has $n + 1$ codewords. This means there are at least $(n + 1)|C|$ words overall. But we know there are exactly 2^n distinct words. So $2^n \geq (n + 1)|C|$ or $|C| \leq \frac{2^n}{n+1}$.

The bound we have derived is called the **Hamming bound**.

2.2 Hamming Codes

2.2.1 Motivating Hamming Codes

Due to the nice result from above, **for the rest of this section, we will only consider binary codes.**

Definition 2.3. A **binary code** is a code over the alphabet $\Sigma_2 = \{0, 1\}$. Each 0 or 1 character is called a **bit**.

We now have an idea of how efficient binary codes can be based on the Hamming bound: we know that $|C| \leq \frac{2^n}{n+1}$. The bigger the size of the code, the more messages we can send. The obvious question to ask here is:

$$\text{Are there any codes which achieve the Hamming bound; i.e, codes with } |C| = \frac{2^n}{n+1}?$$

The answer turns out to be **yes**: a family of codes known as **Hamming codes** achieve the Hamming bound of peak efficiency.

As done previously, let's fix distance $d = 3$ for simplicity. Suppose we want to send a 4-bit message (i.e. a length 4 binary message). The idea is that we need to introduce extra information in the form of "parity check" bits that watch over 3 out of 4 message bits each. We see that to account for the data from each of the message bits, we require 3 such parity bits[1]. A suitable operation to "account" for data, both computationally and intuitively, turns out to be the **exclusive or** of two bits, denoted **XOR**.

Definition 2.4. For any two bits x and y, their **XOR**, denoted $x \oplus y$, is simply $x + y \bmod 2$. For a series of bits $x_1 \dots x_n$, $y_1 \dots y_n$, we xor their bits individually to get $(x_1 \oplus y_1) \dots (x_n \oplus y_n)$.

[1] There are 3 "unknown" bits, so we need 3 parity "equations" to get a unique solution to the unknowns.

For instance, $1101 \oplus 1011 = 0110$, and $10011 \oplus 11001 = 01010$.
Immediately one may see why computationally **XOR** makes sense: it is one of the simplest operations to implement on a computer. Furthermore, take a look at the table of values for **XOR**:

a	b	$a \oplus b$
0	0	0
1	0	1
0	1	1
1	1	0

A result of 1 in the **XOR** captures the fact that there is exactly one error between a and b. This property generalizes to **XOR**s of multiple bits too: the value of **XOR** tells us something about errors. We will soon make this more precise.

Question 2.8 (1 pt each)**.** Compute the following:

a) $110011 \oplus 101101$

b) $10111011 \oplus 10111100$.

A quick computation using the table reveals

(a) $110011 \oplus 101101 = 011110$, or 11110.

(b) $10111011 \oplus 10111100 = 00000111$, or 111.

Question 2.9 (2 pts)**.** Compute a such that $a \oplus (1110111) = (1110110)$.

Set $a = a_1a_2a_3 \ldots a_7$. Then we need $a_1 \oplus 1 = 1$, $a_2 \oplus 1 = 1, \ldots, a_7 \oplus 1 = 0$. This gives $a = 0000001$, or simply $a = 1$.

2.2.2 $[7, 4]$ Hamming Code

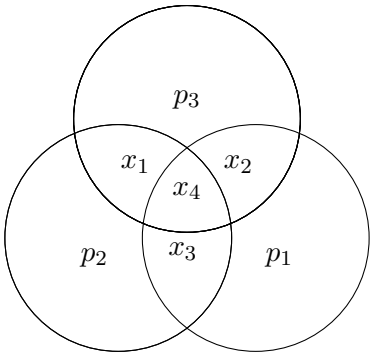
With this inspiration at hand, we can define the $[7, 4]$ Hamming Code.

Definition 2.5. Suppose you have a message $(x_1, x_2, x_3, x_4) \in \Sigma_2^4$. The encoding into [7, 4] Hamming codeword is:

$$E(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, p_1, p_2, p_3)$$

where $p_1 = x_2 \oplus x_3 \oplus x_4$, $p_2 = x_1 \oplus x_3 \oplus x_4$, $p_3 = x_1 \oplus x_2 \oplus x_4$.

The first four bits here are the message bits, and the next three are parity bits, where each parity bit encapsulates information about three message bits using **XOR**.
The goal is to now show that this code has distance 3.



.

Figure 1: Venn Diagram for $[7, 4]$ Hamming Code. Each p_i circle has three message bits within it that it will **XOR** together; intersections show messages bits that are common between the parity bits. For example, p_1 and p_3 share x_2 and x_4 and all of the parity bits contain x_4.

Question 2.10 (3 pts)**.** Show that if $\mathbf{x} = (x_1, \ldots x_7)$ and $\mathbf{y} = (y_1, \ldots y_7)$ are two codewords, then $\mathbf{x} + \mathbf{y} \mod 2$ is also a codeword under the Hamming code. This property is called "additivity."

We claim that the **XOR** of two Hamming codes $\mathbf{x}$ and $\mathbf{y}$ is the Hamming code for $(x_1 \oplus y_1, \ldots, x_4 \oplus y_4)$, the **XOR** of their message bits. Specifically,

$$\mathbf{x} + \mathbf{y} \mod 2 = (x_1 \oplus y_1, \ldots, x_4 \oplus y_4, r_1, r_2, r_3),$$

where $r_1 = (x_2 \oplus y_2) \oplus (x_3 \oplus y_3) \oplus (x_4 \oplus y_4) = p_1 \oplus q_1$, and similar expessions are obtained for r_2 and r_3. This is clear for the first four bits. For the next three bits, by commutativity and associativity,

$$p_1 \oplus q_1 = (x_2 \oplus x_3 \oplus x_4) \oplus (y_2 \oplus y_3 \oplus y_4) = (x_2 \oplus y_2) \oplus (x_3 \oplus y_3) \oplus (x_4 \oplus y_4).$$

Similar expressions can be verified for $p_2 \oplus q_2$ and $p_3 \oplus q_3$.

Question 2.11 (6 pts). Using additivity, prove that for the $[7, 4]$ Hamming code, the minimum distance between any two codewords is the same as the minimum distance from any one nonzero codeword to $\mathbf{0} = (0, 0, \ldots, 0)$. That is, show

$$d(C) = \min_{c \in C, c \neq \mathbf{0}} \delta(c, \mathbf{0}).$$

Hint: $\mathbf{x} - \mathbf{y} = \mathbf{x} + \mathbf{y}$ when adding $\bmod 2$.

We will need the following essential property: $\delta(c, c') = \delta(c - c' \bmod 2, 0)$ for all $c, c' \in C$. The idea here is that subtracting $\bmod 2$ returns 0 whenever two bits are the same, and returns 1 otherwise (since $-1 = 1 \bmod 2$). Thus counting the number of 1s in $c - c' \bmod 2$ tells us the number of places where characters were different. This, however, is identical to computing the Hamming distance between $c - c'$ and $\mathbf{0} = (0, 0, \ldots, 0)$. Since $c - c'$ and $c + c'$ are identical $\bmod 2$, we can conclude that $\delta(c, c') = \delta(c - c' \bmod 2, \mathbf{0}) = \delta(c \oplus c', \mathbf{0})$. Thus minimizing $\delta(c, c') = \delta(c \oplus c' \bmod 2, \mathbf{0})$ over all distinct pairs of code-words,

$$d(C) = \min_{\substack{c, c' \in C \\ c \neq c'}} \delta(c \oplus c', \mathbf{0})$$

Now it is clear that the set $\{c \oplus c' : c, c' \in C \text{ such that } c \neq c'\}$ of codewords is identical to the set of non-zero code words. Setting $c' = \mathbf{0}, c \neq \mathbf{0}$ in the first set gives all elements of the second set, while any **XOR** of unequal codewords obviously produces a non-zero codeword. Thus

$$d(C) = \min_{\substack{c, c' \in C \\ c \neq c'}} \delta(c \oplus c', \mathbf{0}) = \min_{c \in C, c \neq \mathbf{0}} \delta(c, \mathbf{0})$$

The above question reduces our job to showing that the minimum distance of any nonzero codeword from $\mathbf{0}$ is 3.

Let us set up a thought "experiment". Consider the encoded message as seven different lights. A light is on if its corresponding codeword bit is one, and is off otherwise. The lights all being off is the configuration corresponding to the zero codeword. We can only "flip" switches corresponding to the message bits, since the parity bits are derived from them.

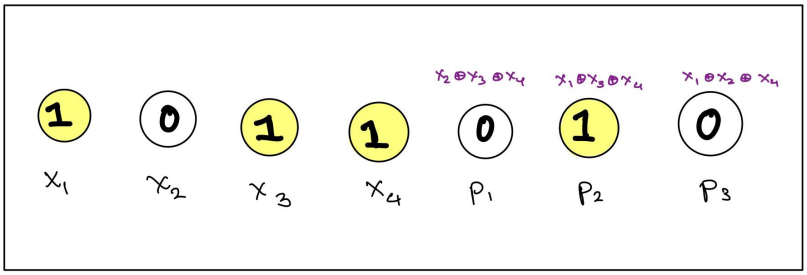


Figure 2: Bulb representation of the message 1011 with Hamming code 1011010

Question 2.12 (6 pts). Show that at least three codeword lights turn on anytime one or two message bits are flipped from the codeword **0** in the experiment.

We will have to consider the first three message bits and x_4 separately, since they are present in parity bits at different frequencies.
If one message bit is flipped, say x_1 without loss of generality, then the value for p_2 and p_3 will be flipped. Thus the lights for x_1, p_2, p_3 would light up. Generally, for the first three bits, given a flipped message bit, the two parity bits containing that message bit are also flipped, which results in three lights being on. The situation for x_4 is a little different: if x_4 is the only message bit flipped, then all three parity bits are flipped since they all contain x_4. Four lights are turned on.
If two message bits are flipped, say x_1 and x_2 without loss of generality, then $p_1 = x_2 \oplus x_3 \oplus x_4$ and $p_2 = x_1 \oplus x_3 \oplus x_4$ would both equal 1 since they are $1 \oplus 0 \oplus 0 = 1$. Thus their lights are turned on, resulting in four total lights, x_1, x_2, p_1, p_2, being turned on. Generally speaking, for any two bits of the first three bits flipped to 1, the two parity bits that contain exactly one of each are also flipped to 1, giving three lights turned on. If say x_1 and x_4 are involved, then x_1, x_4, and p_1 are flipped, giving three lights turned on.

Question 2.13 (4 pts). Prove that $d(C) = 3$.
Hint: First to show that $d(C) \geq 3$, separate into cases of one bit flip, two flips, and three or more flips from the codeword **0**. Then to establish $d(C) = 3$, find a codeword with distance exactly 3 from **0**.

Notice that we are allowed to compute only the distance of non-zero codewords from **0** by virtue of Question 2.11. As seen in the previous question, if one or two message bits are flipped, at-least three of the total message and parity bits are also flipped to 1, resulting in a distance ≥ 3 from **0**. If at least three message bits are flipped, it is obvious that these message bits cause the Hamming distance from **0** to be at least 3. Finally, the code word $(1, 0, 0, 0, 0, 1, 1)$ has distance exactly 3 from **0**, so we know that $d(C) = 3$ indeed.

Question 2.14 (4 pts). Prove that the $[7, 4]$ Hamming code achieves the Hamming bound.

The $[7, 4]$ Hamming code has $|C| = 2^4 = 16$ code words corresponding to 16 choices for the message bits, each generating a unique Hamming code $(x_1, x_2, x_3, x_4, p_1, p_2, p_3)$. Further, each code word has length $n = 7$. Further, as seen in Question 2.13, $d(C) = 3$. Thus we have

$$\frac{2^n}{n+1} = \frac{2^7}{7+1} = 16 = |C|.$$

3 The Singleton Bound and Reed-Solomon Codes (77 pts)

3.1 The Singleton Bound

We now turn back to codes over general alphabets. Now that we've introduced a few codes, a natural question to ask is if the codes are "optimal", or if a "better" code can be found. One way to measure optimality is by the trade-off between distance and *dimension*.

Definition 3.1. Consider a code C with alphabet Σ. Then, the **dimension**[a] of the code is $\log_{|\Sigma|}(|C|)$.

[a]It turns out that for a $[n, k, d]$ code, this is the same as the message length k from before (though you need not understand why for the purposes of the round).

Intuitively, this is the amount of information that the code can send with a single codeword. Our goal is to maximize k while fixing n and d. In general, higher values of d will correspond to lower values of k, as requiring codewords to be a larger distance from one another decreases the number of codewords we can have. The Singleton Bound formalizes this intuition.

Consider a $[n, k, d]$ code C over alphabet Σ.

Question 3.1 (5 pts). Let $c_1, c_2 \in C$ be two distinct codewords. Prove that the $(n - d + 1)$-prefixes of c_1 and c_2 are unique; that is, c_1 and c_2 do not share the same first $n - d + 1$ characters.

Suppose c_1 and c_2 are distinct share the same first $n-d+1$. Then they differ in at most $n-(n-d+1) = d-1$ characters. But $d(C) = d$, so $c_1 = c_2$. This is a contradiction, so c_1 and c_2 have unique $(n-d+1)$-prefixes.

Question 3.2 (5 pts). Prove the Singleton Bound[a], which states that $k \leq n - d + 1$.

Using the previous problem, we see that every codeword corresponds to a unique $(n-d+1)$-prefix. There are $|\Sigma|^{n-d+1}$ such prefixes, so $k = \log_{|\Sigma|}(|C|) \leq \log_{|\Sigma|}(|\Sigma|^{n-d+1}) = n - d + 1$.

[a]The Singleton Bound is not related to singleton sets; rather, it is named after Richard Singleton, a mathematician who proved this bound in 1964.

3.2 Reed-Solomon Codes

Reed-Solomon Codes are a class of codes widely used in communication that achieve the Singleton Bound, that is, we have $k = n - d + 1$ where n is the length of a codeword, k is the dimension of the code, and d is its minimum distance. The main idea of these codes is to treat our message as numbers modulo some prime p. We encode a message as values of a polynomial with integer coefficients, also taken modulo p.

Definition 3.2. A **polynomial with integer coefficients** $P(x)$ of **degree** D for a variable x is an expression written in the form $P(x) = \sum_{i=0}^{D} a_i x^i$, where $a_0, \ldots, a_D$ are integers and $a_D \neq 0$. We often denote $P(x)$ by P. We define the zero polynomial $P(x) = 0$ to have degree -1.

Definition 3.3. We have two natural operations on polynomials: addition and multiplication. Given polynomials $P(x) = \sum_{i=0}^{D_1} a_i x^i$, and $Q(x) = \sum_{i=0}^{D_2} a_i x^i$, we have $(P + Q)(x) = \sum_{i=0}^{D_1} a_i x^i + \sum_{i=0}^{D_2} a_i x^i$ and $(PQ)(x) = \left(\sum_{i=0}^{D_1} a_i x^i\right)\left(\sum_{i=0}^{D_2} a_i x^i\right)$

Definition 3.4. Polynomials A and B with integer coefficients are **equivalent modulo** p if $A - B = p \cdot C$, where C is another polynomial with integer coefficients.

Question 3.3 (4 pts)**.** Show that two polynomials with integer coefficients are equivalent modulo p if and only if for any i, the coefficients of x^i for each of the two polynomials are equivalent modulo p.

Let $A(t) = \sum_{i=0}^{n} a_i t^i$ and $B(t) = \sum_{i=0}^{n} b_i t_i$, where $n = \max(\deg(A), \deg(B))$ and we pad coefficients as 0 as necessary. Suppose that $a_i \equiv b_i (\mod q)$ for all $0 \leq i \leq n$, so $q | a_i - b_i$ and $a_i - b_i = q \cdot c_i$ for some c_i. Then taking $C(t) = \sum_{i=0}^{n} c_i t_i$, we see that $A(t), B(t)$ are equivalent modulo q.

Definition 3.5. Using the previous problem, we define $P(x)$ as a **polynomial of degree D modulo p** as $P(x) = \sum_{i=0}^{D} a_i x^i$, where $a_0, \ldots, a_D \in \{0, 1, \ldots, p-1\}$ and $a_D \neq 0$. Again, we often denote $P(x)$ by P. We can evaluate P at values $t \in \{0, 1, \ldots, p-1\}$, where the value of $P(t)$ is taken modulo p (i.e. in the range $\{0, 1, \ldots, p-1\}$).

Definition 3.6. An integer $t \in \{0, 1, \ldots, p-1\}$ is a **root** of $P(t)$ modulo p if $P(t) = 0$.

Definition 3.7. A polynomial P modulo p **divides** a polynomial Q modulo p if there exists a polynomial D modulo p such that $Q = P \cdot D$.

We will state the following theorem, which we will not prove, but you can use in the round:

Theorem 3.1. For any two nonzero polynomials $A(x)$ and $B(x)$ modulo p, there exist polynomials $Q(x)$ and $R(x)$ modulo p such that $A(x) = B(x)Q(x) + R(x)$ and $\deg R < \deg B$.

For the rest of this section, "polynomial" will refer to a polynomial with integer coefficients taken modulo p, and "roots" of polynomial refer to integers in the set $\{0, \ldots, p-1\}$.

First, let's prove some key properties of polynomials. In the next 4 problems, no facts about polynomials may be used other than the definitions and theorem stated above in Section 3.2 and any previous problems.

Consider an integer $D > 0$.

Question 3.4 (3 pts). Find a polynomial $P(x)$ of degree D with distinct roots $x_1, \ldots, x_D$.

$p(x) = \prod_{i=0}^{D}(x - x_i)$

Question 3.5 (3 pts). Show that for any two polynomials $P(x)$ and $Q(x)$, $\deg PQ = \deg P + \deg Q$.

Set $\deg P = d_1$ and $\deg Q = d_2$, with $P(x) = \sum_{i=0}^{D_1} a_i x^i$ and $Q(x) = \sum_{j=0}^{D_2} b_j x^j$ for $a_{D_1}, b_{D_2} \not\equiv 0 \bmod p$. Then

$$PQ(x) = \left(\sum_{i=0}^{D_1} a_i x^i\right) \cdot \left(\sum_{j=0}^{D_2} b_j x^j\right) = a_{D_1} b_{D_2} x^{D_1+D_2} + (\text{lower degree terms}).$$

Now since $a_{D_1}, b_{D_2} \not\equiv 0 \bmod p$, it follows that their product $a_{D_1} b_{D_2} \not\equiv 0 \bmod p$ as well. Thus $x^{D_1+D_2}$ is the highest degree monomial with a non-zero coefficient, making $\deg(PQ) = D_1 + D_2 = \deg(P) + \deg(Q)$.

Question 3.6 (5 pts). Show that if t is a root of polynomial $P(x)$, then $x - t$ divides $P(x)$.

By Theorem 3.1 (the Euclidean algorithm for polynomials), there exist polynomials $Q(x)$ and $R(x)$ modulo p such that $P(x) = (x - t)Q(x) + R(x) \bmod p$, such that $\deg R < \deg(x - t) = 1$. Then $\deg(R) = 0$ or $R = 0$ necessarily, so $R(x) = r$ for some constant r. Setting $x = t$ above, since t is a root of $P(x)$, we have $P(t) = 0 \bmod p$ and so $0 = (t - t)Q(t) + R(t) = r \bmod p$, so $r = 0 \bmod p$. Thus $P(x) = (x - t)Q(x) \bmod p$, meaning $(x - t) \mid P(x)$ as desired.

Question 3.7 (5 pts). Prove that a degree $D \geq 0$ polynomial cannot have more than D distinct roots.

We argue inductively. For $D = 0$, it is obvious that each non-zero constant[a] polynomial has no roots. Now suppose that for degree $D - 1$ each polynomial $R(x)$ has at-most $D - 1$ roots. Now suppose a polynomial of degree D has $D + 1$ distinct roots $t_1, \ldots, t_{D+1}$. Then by Question 3.6, there is a polynomial $Q(x)$ such that $P(x) = (x - t_{D+1})Q(x)$ modulo p. Clearly Q has degree $D - 1$. Now for each $t_i, 1 \leq i \leq D$, we have $(t_i - t_{D+1})Q(t_i) = P(t_i) = 0$, and since $t_i \neq t_{D+1}$ modulo p (roots are all distinct), we must have $Q(t_i) = 0$. Thus $t_1, \ldots t_D$ are all roots of $Q(x)$, contradicting the inductive hypothesis that Q has at-most $D - 1$ roots. We can then conclude that P also must have at most D roots.
Alternate solution: $P(x) = (x - t_1)Q_1(x), Q_1(x) = (x - t_2)Q_2(x), \ldots$ which is a descent either leading to contradiction (if assuming for contradiction $D + 1$ roots) or the conclusion.

Question 3.8 (6 pts). Prove that if we have distinct integers $x_1, \ldots, x_D$, and (not necessarily distinct) integers $y_1, \ldots, y_D$, there is at most 1 unique degree $D - 1$ polynomial $P(x)$ modulo a prime p such that $P(x_i) = y_i$ for all $1 \leq i \leq D$.

Suppose there are two distinct polynomials $P(x)$ and $Q(x)$ of degree $D - 1$ such that $P(x_i) = y_i$ for $1 \leq i \leq D$. Then $P(x) - Q(x)$ is a non-zero polynomial of degree $D - 1$ with D distinct roots $x_1, \ldots, x_D$, contradicting Question 3.7.

[a] We will not be too pedantic about things related to the zero polynomial.

Question 3.9 (8 pts). Reed-Solomon codes require working modulo a prime p to work, because the property in the previous problem only holds for prime numbers. Show that the result of Question 3.8 is false for any composite p.

Let $p = \prod_{i=1}^{f} p_i^{b_i}$. Take $P(x) = \prod_{i=1}^{f} \prod_{j=1}^{b_i} (x - p_i^j)$. Then $P(x)$ passes through the $\deg P + 1$ points $(0,0), (p_1, 0), \ldots, (p_1^{b_1}, 0), \ldots, (p_f^1, 0), \ldots (p_f^{b_f}, 0)$. However note that $2P(x)$ passes through these same points and $2P(x) \neq P(x)$ as $P(x)$ is monic while $2P(x)$ is not, so we have a contradiction.

We showed above that there is at most 1 unique degree $D-1$ polynomial modulo p passing through D points. In fact, such a polynomial always exists and can be easily found given the points using a computer[2]. For the rest of the round, you can use without proof that given D points, you can construct such a polynomial (just say "use a computer").
Using these results, we have a new scheme. We choose a prime number p and make a code over the alphabet $\Sigma_p = \{0, 1, 2, \ldots, p-1\}$ as follows.

Scheme 3.1. Suppose Alice has a length n message $m_1 m_2 \cdots m_n$, that she wants to send, where $m_1, \ldots, m_n$ are elements of Σ_p.

- She finds a polynomial P taken modulo p such that $P(i) = m_i$ for all $1 \leq i \leq n$. Note that degree of such a polynomial is $n-1$.
- Alice chooses a number $a \geq n$. Alice sends the a values $P(1)$, $P(2)$, $\ldots$, $P(a)$ along the channel to Bob.
- Bob receives a values, which we denote as $r_1, \cdots, r_a$, where r_i is the result of transmitting $P(i)$ over the channel. He can determine the polynomial $P(x)$ uniquely from these values using a computer.
- Bob plugs in $P(1), P(2), \ldots, P(n)$ to recover the message.

However, suppose we have a noisy channel, so k characters are erased. In other words, out of the $P(1), \ldots, P(a)$ that Alice sends, Bob only receives $b = a - k$ values he can read. How can we ensure that Bob can actually determine the polynomial? Well, since we need n points, we must have $b \geq n$ points received, or $a \geq n + k$ points sent.

Question 3.10 (3 pts). Suppose Alice wishes to send the message $(1, 10, 3)$ to Bob such that he will be able to correct up to 2 erasures. Alice and Bob work modulo $p = 11$. Find the encoded message that Alice sends.

By the Lagrange interpolation machine, a unique polynomial $P(x)$ modulo 11 of degree 2 passing through $(1,1), (2,10)$, and $(3,3)$ exists. The quadratic polynomial can easily be deterimend to be $P(x) = 3x^2 + 9$ modulo 11. Since we are correcting two erasures, Alice needs to send at-least 5 values $P(1) = 1, P(2) = 10, P(3) = 3, P(4) = 2, P(5) = 7$. Thus the message to be sent is $(1, 10, 3, 2, 7)$.

Now suppose that instead of characters being erased, we instead have corruptions, where some characters are replaced with other characters. Compared to erasures, Alice will need to send more bits to represent the message; if up to k bits are corrupted, sending $n + k$ of polynomial $P(x)$ is insufficient to guarantee

[2]We will not prove this in this Power Round, but it can be done using a technique known as Lagrange interpolation.

that we can correctly identify the original polynomial.

Question 3.11 (3 pts)**.** Find an example demonstrating the previous sentence for $n = 2$ and $k = 1$.

Say Alice wants to send the message $(1, 2)$. Then the unique polynomial needed is $P(x) = x$ modulo $p = 11$. She sends the encoded message $(1, 2, 3)$, which gets corrupted to $(1, 4, 3)$. Then suppose Bob finds the unique fitting polynomial through 3 points as $Q(x) = 9x^2 + 9x + 5 \bmod 11$. He then receives the message $(1, 4)$.

However, even if Alice sends more characters to Bob to represent her message, how can Bob decode it? Interpolation will not work, as he will be interpolating a polynomial through corrupted points, which do not lie on the actual polynomial we want to send. To do this, we first define the following:

Definition 3.8. If Alice sends a message m and Bob receives it with characters at positions $e_1, e_2, \ldots, e_k$ corrupted, then the **error-locating polynomial** is defined as $E(x) = (x - e_1)(x - e_2) \cdots (x - e_k)$. For example, if Alice sends the message $(19, 13, 20)$ to Bob, and Bob receives the message $(2, 13, 20)$, then $E(x) = x - 1$.

Bob does not know the error-locating polynomial, as otherwise, he knows the corrupted bits and can simply interpolate the polynomial through the remaining points. However, the error-locating polynomial has a useful property that allows us to decode a corrupted message:

Question 3.12 (5 pts)**.** Recall that r_i is the result of sending $P(i)$ over the channel, which Bob receives. Prove that $P(i)E(i) = r_iE(i)$ for all $1 \leq i \leq a$.

Suppose that an error occurs at position i. Then $E(i) = 0$, so $P(i)E(i) = 0 = r_iE(i)$. Otherwise, suppose that an error does not occur at position i. Then $P(i) = r_i$, so $P(i)E(i) = r_iE(i)$.

Let $Q(x) = P(x)E(x)$. If we have $a = n + 2k$, the previous problem shows that $Q(i) = r_iE(i)$ for all $1 \leq i \leq n + 2k$.

Question 3.13 (9 pts). Show that we can determine the polynomials Q and E from the $n+2k$ points received. *Hint*: Consider the degrees of the polynomials defined above.

Note that $\deg E = s$, so it has $s+1$ coefficients, but it is known to be monic so there are s unknown coefficients, and $\deg Q = \deg P + \deg E = k-1+s = k+s-1$, so it has $k+s$ coefficients, so we have a system of $k+2s$ equations with $k+2s$ unknowns, so we can solve the system of equations using standard methods. By Question 3.12, we know that $Q(i) = r_i E(i)$ for $1 \le i \le k+2s$, so the correct coefficients of Q and E are a valid solution to these equations, and thus this system does have a correct solution.

Question 3.14 (2 pts). Given Q and E, how can we recover P?

We have $Q = PE$, so $P = \frac{Q}{E}$.

Question 3.15 (4 pts). Suppose Alice sends a length-3 message (modulo 29) to Bob, but knowing that 1 corruption occurs, she sends an additional 2 characters. Bob receives the encoded message $(2, 6, 12, 17, 1)$. What is the message that Alice sent?

Since we have $s = 1$ errors, our error correcting polynomial is $E(x) = x - e$ for some e. We have $Q(i) = r_i E(i)$, which is a degree 3 polynomial, so $Q(i) = ai^3 + bi^2 + ci + d$. We plug in the equations for $Q(i) = r_i E(i)$ for $1 \le i \le 5$, to get the 5 equations modulo 19.

$$a + b + c + d = 2 - 2e \tag{1}$$
$$8a + 4b + 2c + d = 6 - 6e \tag{2}$$
$$8a + 9b + 3c + d = 12 - 12e \tag{3}$$
$$7a + 16b + 4c + d = 17 - 17e \tag{4}$$
$$11a + 6b + 5c + d = 1 - e \tag{5}$$

from which we get $(a, b, c, d, e) = (1, 26, 25, 0, 4)$, so $P(x) = \frac{Q(x)}{E(x)} = x^2 + x$. Thus Alice's message is $(P(1), P(2), P(3)) = (2, 6, 12)$.

The process described above is known as the Berlekamp-Welch algorithm, named after Elwyn Berlekamp and Lloyd Welch, who published this method in 1983. Finally, we prove that Reed-Solomon codes are optimal with regards to the tradeoff between codeword length and distance.

Question 3.16 (8 pts). Prove that Reed-Solomon codes achieve the Singleton Bound.
Hint: Prove that its minimum distance is $2s + 1$.

Let k be our message length. Consider these two messages which differ only in their last character: $A = m_1 m_2 \dots m_{k-1} x$ and $B = m_1 m_2 \dots m_{k-1} b$ where $a \neq b$. Then when encoding, we have $P_A(i) = m_i = P_B(i)$ for all $1 \le i \le k-1$. We send a total of $k+2s$ points, so the codes we send differ by at most $(k+2s) - (k-1) = 2s+1$. Thus we have $d \le 2s+1$.
Now suppose that $d \le 2s$. Then let c_A and c_B be distinct nonzero codewords with $\delta(c_A, c_B) = d$, corresponding to polynomials $P_A(x)$ and $P_B(x)$, with $\deg P_A \le k-1$ and $\deg P_B \le k-1$. Since our codewords are evaluations of P_A and P_B, $P_A(x) = P_B(x)$ for at least $k + 2s - 2s = k$ values. Therefore $P_A(x) - P_B(x) = 0$ has k solutions, and since $\deg(P_A - P_B) = k-1$, this means that $P_A - P_B = 0$. Thus $P_A = P_B$, so $c_A = c_B$, contradicting c_A and c_B being distinct. Thus we cannot have $d \le 2s$.
Combining with above, we see that $d = 2s + 1$.

4 More Bounds on ECCs (68 pts)

The Singleton Bound proved in the previous section shows that k cannot be too large for a given d. We can ask the opposite question: again given a fixed d, can we construct a code with a given k? The Gilbert-Varshamov Bound is a lower bound on k such that a code of dimension k is always constructible.

Question 4.1 (4 pts)**.** Find the number of binary codewords of length n with a Hamming distance of less than or equal to r from the length-n **0** codeword, $(0, \ldots, 0)$. You can write your answer as a summation.

Picking a codeword with Hamming distance less than or equal to r from the zero vector is equivalent to starting with a length-n zero vector, then picking some $0 \leq i \leq r$ bits and flipping those, so there are a total of $\sum_{i=0}^{r} \binom{n}{i}$ such codewords.

Question 4.2 (1 pts)**.** Let A be the answer to the previous question. Find the number of length n codewords with a Hamming distance of less than d from some given length n codeword x. Express your answer in terms of A.

This is the same as before, except we flip bits starting from x rather than the zero vector, so the answer is A.

Question 4.3 (7 pts)**.** Describe a method of finding a binary code C with length n and distance d such that $|C| \geq \frac{2^n}{A}$, where A is the answer to Question 4.1. This bound on the size of the code is known as the Gilbert-Vashamov Bound.

There are a total of 2^n length-n words. We can greedily pick codewords by simply picking a word not within distance d from our code. Note that each codeword has A codes with distance d from it, so everytime we add a codeword, the number of potential codewords decreases by at most A. Thus we can add at least $\frac{2^n}{A}$ codewords, so $|C| \geq \frac{2^n}{A}$.

The Gilbert-Varshamov Bound is currently the strongest known result for binary codes (in fact, for any $q-$ary code with $q < 49$[3]) and it is currently an open question as to whether it is tight or if codes with higher rate can be constructed. It can be shown, however, that while the GV Bound is true for all $d \leq n$, it is only meaningful when $\frac{d}{n} \leq \frac{1}{2}$, and it tells us nothing when $\frac{d}{n} > \frac{1}{2}$. Intuitively, for this case, we can see that if a code C has high distance, it will contain very few codewords. Thus, a more natural question to ask here is whether we can find an upper bound, ideally much tighter than the Singleton Bound, on the number of codewords such a code can contain. The Plotkin Bound below is such a bound.

[3]For $q \geq 49$, a type of code known as Algebraic Geometric Codes exceeds the GV Bound.

Question 4.4 (16 pts). Prove the Plotkin Bound, which states that for a binary code C of length n and distance d such that $d > \frac{n}{2}$, we have $|C| \leq \frac{2d}{2d-n}$.
Hint: Let r_i denote the number of codewords in C with a 1 in position i. Bound the quantity $\sum_{x,y \in C, x \neq y} \delta(x, y)$.

First a lower bound on the sum. For each distinct pair of code words $x, y \in C$, we obviously have $\delta(x, y) \geq d$. Then

$$\sum_{x,y \in C, x \neq y} \delta(x, y) \geq \sum_{x,y \in C, x \neq y} d = |C|(|C| - 1)d.$$

The upper bound is more work and requires the hint. The sum measures the total number of bit differences in all positions of all codewords. Instead of the current sum over codewords, our goal will be to consider an equivalent sum over positions.
Set r_i to be the number of codewords in C with a 1 in position i. Then there are $|C| - r_i$ codewords with 0 in position i. For a specific position i, each of the r_i codewords x differ from the $|C| - r_i$ code words y in 1 place in that column. Thus the column contributes 2 bit differences to the overall sum: one via $\delta(x, y)$ and one via $\delta(y, x)$. Thus from an entire position, by considering all codewords, we will have $2r_i(|C| - r_i)$ bit differences. Summing over all i, we have indeed all bit differences possible between codewords, so

$$\sum_{x,y \in C, x \neq y} \delta(x, y) = \sum_{i=1}^{n} 2r_i(|C| - r_i).$$

For each i, the quantity $2r_i(|C| - r_i)$ is maximized for $r_i = \frac{|C|}{2}$ and attains value $\frac{1}{2}|C|^2$. Hence the sum is bounded by $\frac{1}{2}n|C|^2$. With this, we get that

$$|C|(|C| - 1)d \leq \sum_{x,y \in C, x \neq y} \delta(x, y) \leq \frac{1}{2}n|C|^2.$$

Then $(|C| - 1) \cdot 2d \leq n \cdot |C|$, which upon rearranging gives $|C| \leq \frac{2d}{2d-n}$ for $n > 2d$.

Up to this point, we have dealt with codes where we can receive a corrupted message and correctly decode it. However, this is often very difficult, so one way to simplify this problem is by weakening our constraints to allow us to output more than one decoded message (a "list of messages"), and consider ourselves successful as long as one of the outputted messages is the correct message. This technique is called **list decoding**. Clearly, we need to limit the number of outputted messages, as we can technically list decode any codeword by simply outputting the entire code C. The Johnson Bound tackles this problem by upper bounding the number of codewords in a given radius.

Definition 4.1. The **Johnson radius** of a binary code C of length n and distance $d \leq \frac{n}{2}$ is $J(n, d) = \frac{n - \sqrt{n(n-2d)}}{2}$.

Question 4.5 (25 pts). (Johnson Bound) Show that for a binary code C of length n and distance $d \leq \frac{n}{2}$, there are at most $2n$ codewords in any Hamming ball of radius $J(n,d)-1$.
Hint: Let r_i denote the number of codewords in the Hamming ball with a 1 in position i. Bound an appropriate quantity and use Cauchy-Schwarz.

Note that it suffices to show that the result holds for Hamming balls of radius $J(n,d)$ centered at 0. Let S be the set of all codewords in this ball. We bound the sum $\sum_{x,y\in S, x\neq y}\delta(x,y)$. By Cauchy-Schwarz, $\frac{1}{n^2}\left(\sum_{i=1}^n \frac{r_i}{|S|}\right) \leq \frac{1}{n}\left(\sum_{i=1}^n \frac{r_i^2}{|S|}\right)$ so we get

$$|S|(|S|-1)d \leq \sum_{x,y\in S, x\neq y}\delta(x,y) \tag{6}$$

$$= \sum_{i=1}^n 2r_i(|S|-r_i) \tag{7}$$

$$= 2|S|^2\sum_{i=1}^n \frac{r_i}{|S|} - 2\sum_{i=1}^n r_i^2 \tag{8}$$

$$\leq 2|S|^2\sum_{i=1}^n \frac{r_i}{|S|} - \frac{2|S|^2}{n}(\sum_{i=1}^n \frac{r_i}{|S|})^2 \tag{9}$$

$$= 2|S|^2\sum_{i=1}^n \frac{r_i}{|S|}\left(1-\frac{1}{n}\sum_{i=1}^n \frac{r_i}{|S|}\right). \tag{10}$$

Let $M = \sum_{i=1}^n \frac{r_i}{|S|}$. Note that this is the sum of the average weights of all codewords in S, so $M \leq J(n,d)-1$. Also, rearranging the definition of Johnson radius and squaring gives $(n-2J(n,d))^2 = n(n-2d)$. The inequality above is equivalent to to $|S|dn - dn \leq 2|S|M(n-M)$, and rearranging gives

$$|S| \leq \frac{dn}{dn-2Mn+2M^2} \tag{11}$$

$$= \frac{2dn}{2dn-n^2+(n-2M)^2} \tag{12}$$

$$< \frac{2dn}{(n-2J(n,d)+2)^2-n(n-2d)} \tag{13}$$

$$\leq \frac{2dn}{(n-2J(n,d))^2+1-n(n-2d)} \leq 2dn. \tag{14}$$

as desired.

Using the Johnson bound, we can prove the Elias-Bassalygo Bound, which is a stronger upper bound on the size of codes than the Hamming Bound.

Question 4.6 (6 pts). (Lemma) Given a binary code C of length n, distance $d \leq \frac{n}{2}$, and some arbitrary $0 \leq r \leq n$, show that there exists a Hamming ball of radius r with at least $\frac{|C| \cdot A}{2^n}$, where A is the answer to Question 4.2.

Let $c \in C$ and pick a random length-n code y. Then letting $\mathbb{1}_c$ be the indicator that c is in $B_d(y)$, we have $\mathbb{E}[\mathbb{1}_c] = P(c \in B_d(y)) = \frac{|B_d(y)|}{2^n} = \frac{A}{2^n}$. By linearity of expectation, the expected number of codewords in C which are in $B_d(y)$ is $\frac{|C|A}{2^n}$, so there must exist some Hamming ball containing at least the expected $\frac{|C|A}{2^n}$, number of codewords.

Question 4.7 (9 pts). (Elias-Bassalygo Bound) Show that a code of length n and distance $d \leq \frac{n}{2}$ satisfies $|C| \leq \frac{n2^{n+1}}{\binom{n+1}{J(n,d)-1}}$.

By the previous problem, there exists a Hamming ball B of radius $J(n,d) - 1$ containing $\frac{|C|A}{2^n}$ codewords where by Pascal,

$$A = \sum_{i=0}^{J(n,d)-1} \binom{n}{i} \geq \binom{n}{J(n,d)-2} + \binom{n}{J(n,d)-1} = \binom{n+1}{J(n,d)-1}$$

. Let S be the set all codewords of C which are contained in B, so $|S| \geq \frac{|C|A}{2^n}$. Since $d \leq \frac{n}{2}$, the Johnson bound shows that $|S| \leq 2n$. Therefore we have $\frac{|C|A}{2^n} \leq 2n$ and rearranging gives $|C| \leq \frac{n2^{n+1}}{A} \leq \frac{n2^{n+1}}{\binom{n+1}{J(n,d)-1}}$.