Question 4:

a. We want to store a binary encoding of the 150 original Pokemon. How many bits do we need to use?

b. What is the encoding for Pikachu (#25)?

Question 2: *Flippin' Fo' Fun* (10 points, 14 minutes)

Assume that the most significant bit (MSB) of x is a 0. We store the result of flipping x's bits into y. Interpreted in the following number representations, how large is the magnitude of y relative to the magnitude of x? Circle ONE choice per row.

Unsigned	$\|y\| < \|x\|$	$\|y\| = \|x\|$	$\|y\| > \|x\|$	Can't Tell
One's Complement	$\|y\| < \|x\|$	$\|y\| = \|x\|$	$\|y\| > \|x\|$	Can't Tell
Two's Complement	$\|y\| < \|x\|$	$\|y\| = \|x\|$	$\|y\| > \|x\|$	Can't Tell
Sign and Magnitude	$\|y\| < \|x\|$	$\|y\| = \|x\|$	$\|y\| > \|x\|$	Can't Tell
Biased Notation (e.g. FP exponent)	$\|y\| < \|x\|$	$\|y\| = \|x\|$	$\|y\| > \|x\|$	Can't Tell

Question 2: *This Problem is Like a Box of Chocolates* (27 points, 48 minutes)

a) In C, `char` is in fact a *signed* variable type. Assume we have 4 `chars` != 0x00 loaded into a 32-bit `int`, one in each byte. Complete the function below that will negate the `char` in byte `i` of the `int` "in place", with byte 0 being the least significant and byte 3 being the most significant.

```
void negByte(int *data, char i) {

   _____________________________________________________________________;

}
```

Question 1: *Reppin' Yo Numbas* (10 points, 16 minutes)

For this question, we are using 16-bit numerals. For Floating Point, use 1 sign bit, 5 exponent bits, and 10 mantissa bits. For Biased use a bias of $-2^{15}+1$.

a) Indicate in which representation(s) the numeral is **closest to zero**: Two's Complement (T), Floating Point (F), or Biased (B). The first one has been done for you. NaN is not valid in comparisons.

	Numeral:	**Closest to zero:**
1)	`0x0000`	__TF__
2)	`0xFFFF`	_____
3)	`0x0001`	_____
4)	`0xFFFE`	_____
5)	`0x8000`	_____
6)	`0x7FFF`	_____

b) We now wish to **add** the numerals from **top to bottom (1 to 6)**. However, it is possible that we encounter an error when performing these addition operations. For each number representation, state the FIRST error that is encountered and which numeral causes it; if no error is encountered, answer "no error."

Possible arithmetic errors are: OVERFLOW, UNDERFLOW, NaN, and ROUNDING (assume we are rounding using a truncating scheme).

Representation:	**Arithmetic Error:**	**Numeral #:**
Two's Complement	_____	_____
Floating Point	_____	_____
Biased	_____	_____

SID: _____

Question 1: Number Representation (15 pts)

a) Complete the tables below:

Convert unsigned integers:

Base 8	Hexadecimal
115_8	
	0x1A

Convert to and from IEC prefixes:

Standard	IEC Prefixes
	16 Pebi-bits
2048 students	

b) Due to limitations in storage space, we are using only 4 bits to represent integers.

1) What is the *most negative 2's complement* signed integer (decimal) we can represent?

2) What is the value (decimal) of the 2's complement number `0b1010`?

3) Write a number (binary) that, when added to `0b0100`, will cause *signed overflow*.

c) An amino acid is defined by a set of 3 consecutive nucleotides (A, C, G, or T). For example, ATG is Methionine. All combinations are unique (e.g. ATG ≠ AGT ≠ GTA).

1) How many total *possible* amino acids are there?

2) In reality, there are 21 amino acids found in the human body. How many bits would it take to encode these amino acids in binary?

3) Scientists also use single-digit encodings for amino acids (e.g. 'A' for Alanine). In a single sentence, explain why it is okay that we use A for the amino acid Alanine, the nucleotide adanine, and the hex representation of the decimal number 10.

4) We wish to encode the 21 amino acids in base 2, 3, or 5. Which of these choices allows for the MOST new amino acids discoveries before needing to increase the number of digits and how many new discoveries are allowed in this choice?

Base:	Possible New Discoveries:

SID: _____

Question 2: C Potpourri (12 pts)

a) Given the library function `rand()` that returns a random number between `0` and `(2^32)-1` when called, write a valid C expression that uses *bit operations* `(^, ~, |, &)` to initialize the variable `r` with a random integer between `0` and `n`, which is some power of 2 less than `(2^32)-1`.

```
int n = 8; // In this case, we want r to contain one of {0,1,2,3,4,5,6,7}.

int r = _____________________________________;
```