Quick Review
N bits represent $2^N$ things:
How many bits do you need to represent 768 things?
10 bits
Kind men give terminal pets extra zebra yolk:
$2^{67} = 128$ exbi

With 8 bits, what are the bit patterns for the following? For the last row, what is the decimal value of the given bit pattern?

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>Sign &amp; Magnitude</th>
<th>One’s Complement</th>
<th>Two’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>No can do</td>
<td>1000 0001</td>
<td>1111 1110</td>
<td>1111 1111</td>
</tr>
<tr>
<td>MAX</td>
<td>1111 1111</td>
<td>0111 1111</td>
<td>0111 1111</td>
<td>0111 1111</td>
</tr>
<tr>
<td>MIN</td>
<td>0000 0000</td>
<td>1111 1111</td>
<td>1000 0000</td>
<td>1000 0000</td>
</tr>
<tr>
<td>0x83</td>
<td>131</td>
<td>-3</td>
<td>-124</td>
<td>-125</td>
</tr>
</tbody>
</table>

In general, with N bits the max/min for unsigned is $2^N - 1/0$, and for two’s complement the max/min is $2^{N-1} - 1/-2^{N-1}$.

What are the advantages and disadvantages of each integer representation?
Unsigned can represent about twice as the others in terms of magnitude, but no negatives. =( S&M (lol) is easier for humans to read, but has two zeroes and the problem of going in the opposite direction after overflow.
One’s Complement fixes above flaw, but still has two zeroes.
Two’s Complement has one extra negative number, but is otherwise perfect.

Complete the following function convert() that takes an unsigned integer as an argument, and returns it’s value when interpreted as a sign and magnitude number:

```c
int convert(unsigned int signMag)
{
    return -(signMag >> 31) * (signMag & 0x7fffffff);
    /* So the >> right shifts the number’s bits by 31 places and leaves
     * Only the topmost bit. The & makes the topmost bit 0. We hardcoded the
     * 31 and the 0x7fffffff mask; later on we’ll learn about sizeof and can
     * Dynamically adjust to the data size. */
}
```

C details
int* p1, p2, p3, p4;
Did I just declare four pointers?
No, that would be int *p1, *p2, *p3, *p4. The spacing around the * doesn’t matter.

```c
if ((5/4) * 100 == 125) printf(“C can do math!\n”);
Did it print?
No, integer division 5/4 is equal to 1, not 1.25. To get correct behavior, cast them or do (5.0/4.0).```
Pointers

Writing the function swap and complete its call.

```c
void swap (int *x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

Alternatively:

```c
void swap (int *x, int* y) {
    *x ^= *y ^= *x ^= *y;
}
```

What is the output of the following program given this snapshot of memory?

<table>
<thead>
<tr>
<th>Variable (if any)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>p</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>...</td>
<td>171</td>
<td>172</td>
<td>173</td>
<td>174</td>
<td>175</td>
</tr>
<tr>
<td>Initial Value</td>
<td>15</td>
<td>19</td>
<td>-5</td>
<td>171</td>
<td>0</td>
<td>255</td>
</tr>
</tbody>
</table>

- `int main(int argc, char * argv[]){`
  - `int main(int argc, char * argv[]){`
    - `int a = 3, b = 144, c = 170;`
    - `int *p;`
    - `printf("%d, %d, %d\n", *p, p, &p);`
    - `p = (int *) foo(a,&c);`
    - `printf("%d, %d, %d\n", *p, p, &p);`
    - `bar(&a, &b);`
    - `printf("%d, %d, %d\n", a, b, c);`
  - `bar(&a, &b);`
  - `printf("%d, %d, %d\n", a, b, c);`
  - `return 0;`
  - `return 0;`

- `int foo (int x, int * y){`
  - `int foo (int x, int * y){`
    - `*y = -12;`
    - `return x + (int) y;`
  - `return x + (int) y;`

- `void bar (int * x, int * y){`
  - `void bar (int * x, int * y){`
    - `*x = *y;`
    - `*y = (int) &y;`
  - `*y = (int) &y;`

- `int mystery (unsigned int n) {`
  - `int mystery (unsigned int n) {`
    - `int count = 8 * sizeof(int) ;`
    - `n ^= (unsigned int) - 1 ;`
    - `while (n) {`
        - `count--;`
        - `n &= (n - 1) ;`
    - `while (n) {`
    - `return count ;`
    - `return count ;`

Do your homework