CSEE 3827 Homework 1 Solutions

1. a) In unsigned binary, the first number is 252 and the second is 112, so the first, \(11111100\), is clearly larger.

b) In two's complement, the first number is -4 and the second is 112, so the second, \(01110000\), is larger.

c) In signed magnitude, the first number is -124 and the second is 112, so the second number is larger.

2. a) In one's complement, the smallest negative number (that is, closest to minus infinity) that can be represented in 6 bits is 100000 = -31, and the largest positive number that can be represented is 011111 = 31.

b) In two's complement, the smallest negative number is 100000 = -32 and the largest positive number is 011111 = 31.

3. a) A quick observation will reveal that the first and third terms of this expression, \(xyz\) and \((xyz)'\), cover all possible input combinations, so the expression evaluates to 1 regardless of the values of \(x, y,\) and \(z\). The truth table is therefore:

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>(z)</th>
<th>(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

b) You may choose to construct the truth table directly from the expression as it is given, but I prefer to simplify and convert into sum-of-products form:

\[
(x + y)(x + z)(x' + z) = (xx + xz + xy + yz)(x' + z) = xx' + xz + xz' + xz + xyx' + xy + x'y + yz + x'y'z + yz = 0 + xz + 0 + xyz + x'y'z = xz + (x + x')yz = xz + yz
\]
The truth table is:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4. There are several ways to solve this problem, and any method is valid as long as it yields the correct result. I prefer to do it algebraically. Begin by simplifying F:

\[ F = xyz'(y'z + x)' + (w'yz + x') \]
\[ = xyz'((y'z)'(x))' + (w'yz + x') \]
\[ = xyz'(y + z')x' + (w'yz + x') \]
\[ = x'xyz'(y + z') + (w'yz + x') \]
\[ = 0 + (w'yz + x') \]
\[ = w'yz + x' \]

Now compute the inverse using DeMorgan's law:

\[ F' = (w'yz + x')' \]
\[ = (w'yz)'(x')' \]
\[ = (w + y' + z')(x) \]

or, in sum-of-products form: \[ F' = wx + xy' + xz' \]

5. The trick with this problem is to substitute a literal value for one of the variables and show that the equation will always be true.

\[ (xy)' = x' + y' \]

Substitute \( x = 1 \):
\[ (1y)' = 1' + y' \]
\[ (y)' = 0 + y' \]
\[ y' = y' \] This is clearly true, so the equation holds for \( x = 1 \).

Now we substitute \( x = 0 \):
\[ (0y)' = 0' + y' \]
\[ (0)' = 1 + y' \]
\[ 1 = 1 \] This is also clearly true, so the equation holds for \( x = 0 \).
(x + y)′ = xy′
Substitute x = 1:
(1 + y)′ = 1′y′
(1)′ = 0y′
0 = 0 This is always true.
Now substitute x = 0:
(0 + y)′ = 0′y′
(y)′ = 1y′
y′ = y′ This is always true.

So we've proved algebraically that DeMorgan's laws are valid.

6. There are several ways to go about doing this. My favorite method appears below:

xy + x′z + yz
= xy + x′z + yz(x + x′)
= xy + x′z + xyz + x′yz
= xy + xyz + x′z + x′yz
= xy(1 + z) + x′z(1 + z)
= xy + x′z

7. The easiest way to construct a sum-of-products form is to read the minterms right off of the truth table:

F = xy′z + x′yz′ + xy′z + xyz′

The product-of-sums form could be constructed using maxterms, but I prefer to start with F′ and then use DeMorgan's law to negate it. The method isn't as important as the answer.

F′ = xy′z′ + x′yz + xy′z + xyz
F″ = F = (xy′z′ + x′yz + xy′z + xyz)′
    = (xy′z′)′(x′yz)′(xyz)′
    = (x + y + z)(x + y′ + z′)(x′ + y + z)(x′ + y′ + z)′

8. We begin with the truth table for an XOR gate:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a XOR b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

From the table, we can see that

a XOR b = a'b + ab′
a) This can't be simplified, so we translate that directly into a circuit:

```
    a
   /\  
  /   \ 
 b -\ / - a \ or \ b
     \ /      
      b
```

b) We can start from our first circuit and convert it into a NAND-only circuit: A NAND gate can simulate a NOT gate if its inputs are tied together. Also, any sum-of-products arrangement of AND and OR gates will keep its functionality if every gate is swapped out for a NAND gate. So the NAND-gate implementation is as follows:

```
    a
   /\  
  /   \ 
 b -\ / - a \ or \ b
     \ /      
      b
```

9. a) \[ F = x'y'z' + x'y'z + xy'z' + xyz' \]
   This equation is easy to simplify algebraically, so the question specifies using a K-map.

```
    xy
   z   00 01 11 10
   0 1 1 1 1 1 1 1
   1 0 0 0 0 0 0 0
```

\[ F = z' \]

b) \[ F = y'z' + y'z + xyz' \]

```
    xy
   z   00 01 11 10
   0 1 0 1 0 1 0 1
   1 1 0 0 0 0 0 1
```

\[ F = y' + xz' \]

10. The problem must be done algebraically. Pulling the minterms off of the k-map, we get

\[
F = a'bc' + abc' + a'bc + abc \\
= bc'(a + a') + bc(a + a') \\
= bc' + bc \\
= b(c' + c) \\
= b
\]