1. The following table describes various spiders. Given the first seven characteristics, build a
decision tree to determine if a spider is poisonous. Use entropy to select the branching order
as described in the text and in lecture.

<table>
<thead>
<tr>
<th>Big</th>
<th>Fat</th>
<th>Hairy</th>
<th>Slimy</th>
<th>Bloated</th>
<th>Beady-eyed</th>
<th>Gnashing</th>
<th>Vampiric</th>
<th>Poisonous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

2. Build the smallest decision tree to represent the following clause:

\[(A \land B) \lor (C \land D \land E)\]

You can assume that the variables \(A, B, C, D, E\) take the Boolean values 0 (representing
false) and 1 (representing true). Comment on why the decision tree seems large relative to
the clause.

3. The rest of this exercise compares decision trees to perceptrons, which are single layer neural
nets. I will specify two Boolean functions. For each function you will design a perceptron
that computes this function. In each case the function will take \(N\) Boolean inputs \(x_1, \ldots, x_N\)
and output a Boolean value, as in the previous problem. The first function, which I will call
\(f\), is the majority function. It takes the value 1 if most of its inputs are 1, otherwise it takes
the value 0. Thus it is defined by

\[
f(x_1, \ldots, x_N) = \begin{cases} 1 & \text{if } N/2 \text{ or more of the } x_i's \text{ are 1} \\ 0 & \text{otherwise} \end{cases}
\]

a) Draw a picture of a perceptron that implements the function \(f\) for three inputs, indicating
the weights and threshold values used.

b) Build a decision tree that implements the function \(f\) for three inputs.

4. The second function, which I call \(h\), is the palindromic recognition function. It is defined for
even \(N\) by

\[
h(x_1, \ldots, x_N) = \begin{cases} 1 & \text{if } x_1 = x_N \text{ and } x_2 = x_{N-1} \text{ and, } \ldots, \text{ and } x_{N/2} = x_{N/2+1} \\ 0 & \text{otherwise} \end{cases}
\]

a) Even for \(N = 2\), this function cannot be implemented by a simple perceptron. Prove this
by picture (i.e. show that it is not linearly separable.)

b) Build a decision tree that implements the function \(h\) for \(N = 2\) and \(N = 4\).

5. (EXTRA CREDIT) The next function, which I will call \(g\), is known as Muroga’s function. It
is defined using the following nested if-then-else expression:
if $x_1 = 1$ then $g(x_1, \ldots, x_N) = 1$

else if $x_2 = 1$ then $g(x_1, \ldots, x_N) = 0$

else if $x_3 = 1$ then $g(x_1, \ldots, x_N) = 1$

else if $x_N = 1$ then $g(x_1, \ldots, x_N) = 1$

else $g(x_1, \ldots, x_N) = 0$

if $N$ is odd, otherwise the sign of $g$ is flipped in the last two cases, so that the alternation between 1 and 0 is maintained. You must show that this function can be implemented by a perceptron. The curious fact about this function is that even though it can be represented exactly by a perceptron, if you start from all weights equal to zero, the perceptron learning algorithm requires exponentially many updates (w.r.t. $N$) to learn this function. Double extra credit if you can prove this.