Q1:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c</td>
<td>6</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>7</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>i</td>
<td>8</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>h</td>
<td>9</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>10</td>
<td>j</td>
<td></td>
</tr>
</tbody>
</table>

Q2:

<table>
<thead>
<tr>
<th></th>
<th>UMA</th>
<th>NUMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td><img src="a" alt="Diag UMA" /></td>
<td><img src="b" alt="Diag NUMA" /></td>
</tr>
<tr>
<td><strong>Definition</strong></td>
<td>the time taken by a processor to access any memory word in the system is identical</td>
<td>Access time to local memory is much less than shared-address-space memory</td>
</tr>
<tr>
<td><strong>Pros/Cons</strong></td>
<td>Contention problem especially in the case of no caches</td>
<td>Low cost, Good scalability, require locality to improve performance</td>
</tr>
</tbody>
</table>

Q3:
Q4:

\[ \frac{1}{(1 - 0.5) + \frac{0.5}{15}} = \frac{1}{0.5 + 0.033} = 1.876 \]

\[ \frac{1}{(1 - 0.75) + \frac{0.75}{15}} = \frac{1}{0.25 + 0.05} = 3.33 \]

\[ F = \frac{1}{(1 - F) + \frac{F}{15}} = \frac{15}{15 - 15F + F} = \frac{15}{15 - 14F} \]

\[ 2.25(15 - 14F) = 15 \]

\[ 33.75 - 31.5F = 15 \]

\[ 31.5F = 18.75 \]

\[ F = \frac{18.75}{31.5} = 0.595 \text{ or } 60\% \]

Q5:

>> each row of the matrix occupies 4000*4 = 16 KB
The matrix is too large to fit in the cache. Hence, only the vector gets cached, where each element actually occupies one word

>> The vector get cached in \( \frac{4000}{4} * 100 * 10^{-9} = 100 \) \( \mu \text{sec} \)
However, at that time no computation can be done at all.
>> the matrix get cahed in:
\[
\frac{4 * 10^3 * 4 * 10^3}{4} * 100 * 10^{-9} = 0.4 \text{ seconds}
\]
By the time the rows of the matrix is loaded into cache the computation can proceed regularly.

>> The total number of computations = 2n²
Hence the effective peak computation rate can be computed as:

\[
\frac{\text{number of FLOPs}}{\text{data access time}} = \frac{0}{100 \times 10^{-6}} + \frac{2 \times 4000 \times 4000}{0.4} = 80 \times 10^6
\]

This corresponds to a peak computation rate of 80 MFLOPS

**Another solution:**

>> each element actually occupies one word
>> the matrix is too large to fit in the cache. Hence, only the vector gets cached.
In this case, 8 FLOPS can be performed on 1 cache line for the matrix.

\[
\frac{8}{100 \times 10^{-9}} = 80 \times 10^6
\]

This corresponds to a peak computation rate of 80 MFLOPS