1. (String Matching)

(a) (Rabin-Karp algorithm) For $n = 26$, $val(n) = 4$. From the following table, there are 3 possible matches out of which one is successful; so there are 2 failed matches.

<table>
<thead>
<tr>
<th>text</th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>1</th>
<th>5</th>
<th>8</th>
<th>2</th>
<th>6</th>
<th>5</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>7</th>
<th>9</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>val(n)</td>
<td>9</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

(b) (KMP algorithm) Suppose $p$ occurs somewhere in $t$; consider the first occurrence. Then $t$ is of the form $xpy$, and $pt$ is of the form $pxpy$. The core of the prefix $pxp$ is $p$. That is, if $p$ is in $t$, there is a prefix of $pt$, of length at least $2 \times p$, whose core is exactly $p$. Conversely, suppose $pt$ has a prefix $z$ whose core is $p$. If $z$ is at least $2 \times p$ in length, its suffix which matches $p$ is entirely past $p$, i.e., within $t$. The additional condition on length is needed because if $p = aa$ and $t = ab$, then $pt = aaba$, the core of $aaa$ is $aa$, which is $p$, though $p$ is not in $t$.

(c) Yes. Suppose $v = \text{"ab"}$ and $v' = \text{"aba"}$. Then $c(v) = \epsilon$, and $u$ has been set to $c(v)$, i.e., $\epsilon$, before this portion of the code is executed. Because $p[\tilde{v}]$ and $p[\tilde{v}]$ are both “$a$”, $c(v')$ will be set “$a$”.

(d) (KMP Algorithm) This table shows various values of $l$. It is possible to terminate the algorithm when $l = 6$, because the text cannot possibly match the pattern at $l = 8$, from length considerations.

<table>
<thead>
<tr>
<th>$l$</th>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>pattern</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>a</td>
<td>b</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>a</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>a</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>a</td>
<td>b</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
2. (Data Parallel Programming)

(a) (Batcher Merge) First, we show that for any powerlist $v$, $v \downarrow v = v \uparrow v$.

\[
\begin{align*}
v \downarrow v &= \{ \text{definition of } \downarrow \} \\
&= \{ (v \ \min v) \bowtie (v \ \max v) \} \\
&= \{ (v \ \min v) = v \ \text{and} \ (v \ \max v) = v \}
\end{align*}
\]

From this result, it is sufficient to prove that $u \bowtie \uparrow u$. The proof is by induction on the structure of $u$.

- $(x) \bowtie \uparrow (x) = (x) \downarrow (x)$:

\[
(x) \bowtie \uparrow (x) = \{ \text{definition of } \bowtie \} (x) \downarrow (x)
\]

- Given that $p \bowtie q$ is sorted, show that $(p \bowtie q) \bowtie \uparrow (p \bowtie q) = (p \bowtie q) \downarrow (p \bowtie q)$:

\[
\begin{align*}
(p \bowtie q) \bowtie \uparrow (p \bowtie q) &= (p \bowtie q) \downarrow (p \bowtie q) \\
&= \{ \text{definition of } \bowtie \} (p \bowtie q) \downarrow (p \bowtie q) \\
&= \{ \text{since } p \text{ and } q \text{ are parts of } p \bowtie q \text{ and } p \bowtie q \text{ is sorted} \} (p \bowtie q) \downarrow (p \bowtie q)
\end{align*}
\]

(b) Suppose $p = \langle p_0 \cdots p_n \rangle$ and $q = \langle q_0 \cdots q_n \rangle$. Then the $i^{th}$ elements of $ps p$, $ps q$ and $ps(p + q)$ are, respectively,

\[
\begin{align*}
(ps p)_i &= (\oplus i : 0 \leq j \leq i : p_j) \\
(ps q)_i &= (\oplus i : 0 \leq j \leq i : q_j) \\
(ps(p + q))_i &= (\oplus i : 0 \leq j \leq i : p_j \oplus q_j)
\end{align*}
\]

Since $\oplus$ is commutative and associative, $(\oplus i : 0 \leq j \leq i : p_j \oplus q_j) = (\oplus i : 0 \leq j \leq i : p_j) \oplus (\oplus i : 0 \leq j \leq i : q_j)$, which is $(ps p)_i \oplus (ps q)_i$.

For the counterexample: let $\oplus$ be string concatenation, $p = \langle 0 \ 1 \rangle$, $q = \langle a \ b \rangle$. Then

\[
\begin{align*}
ps p &= \langle 0 \ 01 \rangle \\
ps q &= \langle a \ ab \rangle \\
(ps p) \oplus (ps q) &= \langle 0a \ 01ab \rangle \\
ps(p \oplus q) &= ps\langle 0a \ 1b \rangle = \langle 0a \ 01ab \rangle
\end{align*}
\]