Only at most six friends on display, compare directly. $O(n)$ complexity.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong></td>
</tr>
<tr>
<td>4 2 8</td>
</tr>
<tr>
<td>300 950 500 200</td>
</tr>
<tr>
<td>1 3</td>
</tr>
<tr>
<td>2 4</td>
</tr>
<tr>
<td>2 3</td>
</tr>
<tr>
<td>1 1</td>
</tr>
<tr>
<td>1 2</td>
</tr>
<tr>
<td>2 1</td>
</tr>
<tr>
<td>2 2</td>
</tr>
<tr>
<td>2 3</td>
</tr>
<tr>
<td><strong>output</strong></td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>
Use a list \( \mathbf{li} \) to keep track of the gender of the dancers. Use a list \( \mathbf{dan} \) to keep track of the dancer’s left and right neighbours. Use a heap \( \mathbf{h} \) to keep track of the neighbouring couples ever appeared. Use a set \( \mathbf{notthere} \) to log the neighbouring couples that are not currently in the line.

\[
\text{heapify}(\mathbf{h})
\]

\[
\text{while } \mathbf{h}:
\]

\[
d = \text{heappop}(\mathbf{h})
\]

\[
\text{if not}(d[1] \text{ in } \mathbf{notthere}):
\]

\[
1, r = d[1][0], d[1][1]
\]

\[
l = \text{dan}[1][0] ; ri = \text{dan}[r][1]
\]

\[
L = R = \text{False}
\]

\[
\text{if } le != \text{ None}:
\]

\[
\text{if } li[le-1] != li[1-1]:
\]

\[
\text{notthere}.\text{add}((le, li))
\]

\[
\text{dan}[le][1] = ri
\]

\[
L = \text{True}
\]

\[
\text{if } ri != \text{ None}:
\]

\[
\text{if } li[ri-1] != li[r-1]:
\]

\[
\text{notthere}.\text{add}((r, ri))
\]

\[
\text{dan}[ri][0] = le
\]

\[
R = \text{ True}
\]

\[
\text{if } L \text{ and } R:
\]

\[
\text{if } li[le-1] != li[ri-1]:
\]

\[
\text{heappush}(\mathbf{h}, \text{abs}(a[le-1]-a[ri-1]), (le, ri)))
\]

\[
\text{seq.append(' ' .join((\text{str}(l), \text{str}(r))))}
\]
The result is the greatest common divisor of all the numbers

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1 2</td>
<td>2</td>
</tr>
<tr>
<td>3 2 4 6</td>
<td>6</td>
</tr>
<tr>
<td>2 12 18</td>
<td>12</td>
</tr>
<tr>
<td>5 45 12 27 30 18</td>
<td>15</td>
</tr>
</tbody>
</table>
We simulate the process. Use a heap $h$ to keep track of the current first substrings of the string $s$. First, we heapify all the characters in the string, the top of the heap is to be written down. Second, each time we have written down a substring $a$ on the top of the heap, we add to the heap the substring made by $a + s[\text{last_of}_a]$

```python
for i in range(k):
    t = heap.pop(h)
    last_of_a, a = t[1], t[0]
    ans = a
    if last_of_a < n:
        a += s[last_of_a]
        heap.push(h, (a, last_of_a+1))
print(ans)
```