Scores

- c and s stand for “correctness” and “style”
- Each item marked ** counts as two items

<table>
<thead>
<tr>
<th>Score</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness errors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Style errors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0. General

(s0a) Variable names are appropriately chosen
(s0b) Code lines are properly indented
(s0c) All extraneous output is suppressed with semi-colons
(s0d) Reasonable line lengths, no horizontal scrolling
(s0e) Appropriate header comment in each script
(s0f) No superfluous code
(s0g) Table headers are printed and align with table data, where appropriate

1. Roots of a Cubic

(c1a) for or while loop that runs for at least 100000 steps
(c1b) Running count of “nice” cubics is initialized to 0 before the start of the loop
(c1c) Coefficients b, c, and d are generated at every iteration of the loop; a is either not used, or set to 1
(c1d) The determinant is computed for \( q'(x) = 3x^2 + 2bx + c \), using correct coefficients
(c1e) Roots \( r_1 \) and \( r_2 \) are computed for \( q'(x) \), using correct coefficients
(c1f) \( q(r_1) \) and \( q(r_2) \) are correctly computed
(c1g) Running count of “nice” cubics is updated whenever the determinant > 0 and \( q(r_1) \times q(r_2) < 0 \)
(c1h) Probability table is printed as specified: one entry per 1000 iterations for the first 10000 iterations, and one entry per 10000 iterations afterwards

(s1a) The determinant is computed only once, and is used in computing the square roots
(s1b) \( r_1 \), \( r_2 \), \( q(r_1) \), and \( q(r_2) \) are only computed if the determinant is greater than 0
(s1c) Concise logic determines when to print table entries; multiple “||” expressions should not be used

2. \((n, k, r)\)-stars

(c2a) \( \theta \) is computed as \( 2\pi/n \); a variable for \( \theta \) does not need to be assigned explicitly
(c2b) for or while loop runs over each of the \( n \) points
(c2c) The points \( V_j = (x_j, y_j) \) are computed so that \( x_j = \cos(j\theta) \) and \( y_j = \sin(j\theta) \)
(c2d) (**For each \( V'_j \), \( W_{j-k} = (x_{j-k}, y_{j-k}) \) and \( W_{j+k} = (x_{j+k}, y_{j+k}) \) are computed so that:
\[
\begin{align*}
x_{j-k} &= r \cos((j-k)\theta) \\
y_{j-k} &= r \sin((j-k)\theta) \\
x_{j+k} &= r \cos((j+k)\theta) \\
y_{j+k} &= r \sin((j+k)\theta)
\end{align*}
\]
(c2e) Segments from \( V_j \) to \( W_{j-k} \) and \( W_{j-k} \) are drawn, using appropriate endpoints
(c2f) Segments which cross an axis – meaning that the \( x \) or \( y \) coordinates of their endpoints have different signs – are drawn in red, otherwise they are drawn in green. Alternatively, if at least one of the segments \( (V_j, W_{j-k}) \) or \( (V_j, W_{j+k}) \) crosses an axis, both are red, otherwise both are green

(s2a) \( \theta \) is computed once and stored in a variable, not re-computed every time in the loop
(s2b) The loop does not go through more iterations than necessary
(s2c) Concise logic tests whether a segment crosses an axis: should not consist of multiple “&”s and “||”s
3. \((n, k, r)\)-stars

(c3a) Variables for counting the number of terms and storing the running sum are properly initialized

(c3b) A while loop is set up to run for as long as the relative error exceeds the tolerance \(10^{-5}\)

(c3c) The term count is incremented by 1 inside the loop

(c3d) The running sum is incremented inside the loop according to the update formula

(c3e) The relative error is computed correctly.

(c3f) (**The minimum number of terms is \(N = 31831\). Count as one error if \(N\) is off by just 1.

(s3a) The number of steps is output as an integer, not a floating-point number.

4. Searching DNA

(c4a) Variables start and slice are initialized correctly (\(\text{start} = 34124; \text{slice} = 41242;\))

(c4b) (**A while loop is set up to run until slice is equal to start (this does not need to be explicit). If the loop also terminates after a set number of steps, count this as one error (i.e. the loop should only terminate once the matching sub-sequence is found)

(c4c) (**slice is updated at each step so that its first digit is dropped, and the next digit is appended

(c4d) The correct sequence length is printed out after the loop terminates

(s4a) The sequence length is printed on a new line

(s4b) The update formula for slice is simple, using either mod or rem with addition and multiplication by 10, without logarithms, exponentiation, etc.