This test consists of 5 questions on 10 pages (including this one). When you receive the signal to start, please make sure that your copy of the test is complete.

Please answer questions in the space provided. You will earn 20% for any question you leave blank or write “I cannot answer this question,” on. You will earn substantial part marks for writing down the outline of a solution and indicating which steps are missing.

Good Luck!
QUESTION 1. [5 marks]

Why is there a gap of hundreds of years between the namesake of algorithms (Al-Khwarizimi), and today, when algorithms are indispensable in everyday life? Why did it take so long for algorithms to become such an everyday phenomenon?

In the intervening centuries there have been machines built that can run algorithms automatically (computers), more everyday problems have algorithms for them (graphics, text), the scale of society and economics exceed manual methods, and governments invested huge sums to make large scale computing possible.

These changes - industrial techniques, algorithm development, government policy, population explosion - don't happen overnight, they take centuries.
QUESTION 2. [3 marks]

Are there any examples of algorithms that do not use a machine? Explain.

Sure, a sequence of simple steps that you can carry out without any external aid, say counting on your fingers.
QUESTION 3.  [8 MARKS]

Complete the table below. In each row, assume \(a\), \(b\), and \(c\) have the given values, and then evaluate the expression at the end of the row — it will be either true or false.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>(not (or (and (a) (b) (c)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
QUESTION 4.  [11 MARKS]

Assume the expressions below have been typed into the definitions pane of DrRacket. Below each parenthesized expression write, draw, or describe its effect when the “Run” button is clicked. pic: hacker and pic: calendar are images supplied by picturing-programs.

(require picturing-programs)

(provide tools for working with images)

(scale
  (length
    (list "seven" 3 false (circle 10 "solid" "green"))) pic: hacker)

(4x enlargement of pic: hacker)

(/ (* 2 (- 15 (+ 4 5))) 4)

(12 6 9)

(3)

(and (not (equal? 7 (+ 2 3))) (> (+ 2 3) 4))

(true)

(map rotate-cw
  (rest (list pic: calendar (triangle 10 "solid" "blue") pic: hacker)))

(list with (triangle 10 "solid" blue) and pic: hacker each rotated 90° clockwise)

(beside (above pic: hacker pic: calendar)
  (above pic: calendar pic: hacker))
QUESTION 5.  [10 MARKS]

For each of the two functions `product>sum` and `little-faceoff`, I have provided a summary sentence, one check-expect expression, started the define statement, and written an incomplete contract comment (I left out what is consumed and what is produced, after the ";"). You should complete both functions by:

(i) Adding another appropriate check-expect expression for each function

(ii) Completing the define statement with the body of the definition for each function

(iii) Completing the contract comment, saying what is consumed and what is produced for each function

(require picturing-programs)

; product>sum : number number number number number → boolean
; Is ab > c+d?
(define (product>sum? a b c d)
  ; write the body of the definition below here
  (>
    (* a b)
    (+ c d))
)

; write your check-expect below this one
(check-expect (product>sum? 2 3 1 4) true)
(check-expect (product>sum? 1 4 2 3) false)
; little-faceoff : image -> image
; Half-size im beside half-size horizontally-flipped im.
(define (little-faceoff im)
  ; write the body of the definition below here
  (beside (scale 1/2 im)
    (scale (flip-horizontal im))))

; write your check-expect below this one
(check-expect (little-faceoff pic:calendar)
  (beside (scale 1/2 pic:calendar)
    (flip-horizontal (scale 1/2 pic:calendar)))))

(check-expect (little-faceoff pic:hacker)
  (beside (scale 1/2 pic:hacker)
    (flip-horizontal (scale 1/2 pic:hacker)))))
# 1: _____/ 5
# 2: _____/ 3
# 3: _____/ 8
# 4: _____/11
# 5: _____/10

TOTAL: _____/37