Algorithms from IB Examinations

Previous examination paper questions, even those that apply to an earlier syllabus, can be a valuable resource because they can be used:
- to illustrate algorithms that relate either directly to the syllabus details
- to strengthen the problem-solving skills referred to in the introduction to Section B
- as a source of data and information for school-based tests.

This section consists of examples of questions that required candidates to trace and/or construct algorithms and their solutions. Some are complete questions, and others are parts of questions; they both refer to algorithm tracing or construction. In some cases solutions have been provided because they include an algorithm as part of the answer.

All questions and solutions have been rewritten in PURE.

The questions are categorized according to year, level, paper, question number, the action on the algorithm required, and the content area of the question. At the time of compilation the M98 papers had not yet been taken by candidates.

<table>
<thead>
<tr>
<th>No.</th>
<th>Year</th>
<th>Level &amp; Paper</th>
<th>Qu.</th>
<th>Action on Algorithm</th>
<th>Content of Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M96</td>
<td>SL P1</td>
<td>1</td>
<td>Trace</td>
<td>Summing and Averaging</td>
</tr>
<tr>
<td>2</td>
<td>M96</td>
<td>SL P1</td>
<td>3</td>
<td>Trace</td>
<td>Count Loops</td>
</tr>
<tr>
<td>3</td>
<td>M96</td>
<td>SL P2</td>
<td>1</td>
<td>Trace and construct</td>
<td>Determining Frequencies</td>
</tr>
<tr>
<td>4</td>
<td>M96</td>
<td>SL P2</td>
<td>5</td>
<td>Construct</td>
<td>If-then-else Statements</td>
</tr>
<tr>
<td>5</td>
<td>N96</td>
<td>SL P1</td>
<td>1</td>
<td>Trace</td>
<td>Dividing and Truncating to Manipulate Single-decimal Digits</td>
</tr>
<tr>
<td>6</td>
<td>N96</td>
<td>SL P1</td>
<td>3</td>
<td>Trace and construct</td>
<td>Nonsense with Absolute Value</td>
</tr>
<tr>
<td>7</td>
<td>N96</td>
<td>SL P2</td>
<td>5</td>
<td>Trace and construct</td>
<td>Calculating Typing Fees</td>
</tr>
<tr>
<td>8</td>
<td>M97</td>
<td>SL P1</td>
<td>1</td>
<td>Trace and analyse</td>
<td>Golf Scores in Arrays</td>
</tr>
<tr>
<td>9</td>
<td>M97</td>
<td>SL P1</td>
<td>3</td>
<td>Trace</td>
<td>Summing Loop</td>
</tr>
<tr>
<td>10</td>
<td>M97</td>
<td>SL P2</td>
<td>1</td>
<td>Construct</td>
<td>One-dimensional Array Manipulation (Hashing)</td>
</tr>
<tr>
<td>11</td>
<td>N97</td>
<td>SL P1</td>
<td>1</td>
<td>Trace</td>
<td>Finding the Largest Value in an Array</td>
</tr>
<tr>
<td>12</td>
<td>N97</td>
<td>SL P1</td>
<td>3</td>
<td>Trace and modify</td>
<td>Binary Search</td>
</tr>
<tr>
<td>13</td>
<td>N97</td>
<td>SL P2</td>
<td>5</td>
<td>Construct</td>
<td>Simulation to Count Fish</td>
</tr>
<tr>
<td>14</td>
<td>M96</td>
<td>HL P1</td>
<td>10</td>
<td>Trace and construct</td>
<td>Highest Values in a Two-dimensional Array</td>
</tr>
<tr>
<td>15</td>
<td>M96</td>
<td>HL P2</td>
<td>4</td>
<td>Trace and construct</td>
<td>Fibonacci Sequence (recursive)</td>
</tr>
<tr>
<td>16</td>
<td>N96</td>
<td>HL P2</td>
<td>1</td>
<td>Trace and construct</td>
<td>Binary Search</td>
</tr>
<tr>
<td>17</td>
<td>N96</td>
<td>HL P2</td>
<td>7</td>
<td>Trace and modify</td>
<td>Binary Tree</td>
</tr>
<tr>
<td>18</td>
<td>N97</td>
<td>HL P1</td>
<td>10</td>
<td>Trace and construct</td>
<td>Summing in a Two-dimensional Array</td>
</tr>
<tr>
<td>19</td>
<td>N97</td>
<td>HL P2</td>
<td>1</td>
<td>Trace and construct</td>
<td>Random Numbers</td>
</tr>
</tbody>
</table>
Summing and Averaging (M96 SL P1 Q1)

The following algorithm fragment has been designed to analyse the temperatures at a tourist resort.

```
COUNT, TOTAL, TEMP, BIG, AVERAGE  // variables
1   COUNT = 0
2   TOTAL = 0
3   input TEMP
4   BIG = TEMP
5   loop while not(TEMP = 0)
6       TOTAL = TOTAL + TEMP
7       COUNT = COUNT + 1
8       input TEMP
9       if TEMP > BIG then
10          BIG = TEMP
11       endif
12   end loop
13   AVERAGE = TOTAL / COUNT
14   output AVERAGE, BIG
```

Copy and complete the following trace table for the data: 15, 7, 23, 9, 0

<table>
<thead>
<tr>
<th>Line</th>
<th>COUNT</th>
<th>TOTAL</th>
<th>TEMP</th>
<th>BIG</th>
<th>TEMP ≠ 0</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(a) Consider the algorithm fragment below.

\[
X = 1 \\
\text{loop while } X < 6 \\
\quad Y = 1 \\
\quad \text{loop while } Y < 5 \\
\quad \quad Y = Y + 1 \\
\quad \text{end loop} \\
\quad \text{output } \text{"The product of X and Y is ", } X \times Y \\
\quad \text{output } \text{"The values of X and Y are ", } X, Y \\
\quad X = X + 1 \\
\text{end loop}
\]

Complete the trace through the fragment. The first two lines are provided and your output should be similar to that below:

The product of X and Y is 5
The values of X and Y are 1 5

(b) Consider the algorithm fragment below.

\[
X = 1 \\
\text{loop while } X < 6 \\
\quad Y = 1 \\
\quad \text{loop while } Y < 6 \\
\quad \quad \text{output } \text{"The product of X and Y is ", } X \times Y \\
\quad \quad Y \leftarrow Y + 2 \\
\quad \text{end loop} \\
\quad \text{output } \text{"The values of X and Y are ", } X, Y \\
\quad X = X + 2 \\
\text{end loop}
\]

Complete the trace through the fragment. The first line is provided and your output should be similar to that below:

The product of X and Y is 1

.
Determining Frequencies (M96 SL P2 Q1)

In your school, you must determine the number of students taking various examinations. As part of the input process, a student enters the code of a subject into a computer. For example, 21 could represent computer science. These are stored in an array and the frequency (number of times) with which each code occurs is to be determined.

(a) Consider the following subject code data:


Copy and complete the tables below to show the unique subject codes and their corresponding frequency. Use the subject code data above.

<table>
<thead>
<tr>
<th>CODES</th>
<th>FREQS</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Consider the following variables:

CODES single-dimension array of integer values which will eventually contain the subject codes without duplications.
(Initialized for a maximum of 750 values.)

FREQS single-dimension array of integer values which will eventually contain the frequency of the subject codes.
(Initialized for a maximum of 750 values.)

SIZE Integer variable that indicates the current number of valid entries in CODES and FREQS.

CODE Integer variable that contains an input subject code.

FOUND Boolean variable that indicates if the current value in CODE is already stored in CODES.

POSITION Integer variable that indicates the position that CODE should be placed in the CODES array.

(This question continues on the following page)
The algorithm below (FREQUENCIES) uses these variables to compute the subject code frequencies.

\[
\text{SIZE} = 0
\]

\textbf{input} CODE // the input stream is terminated by -99

\textbf{loop while} (SIZE <= 750) and (CODE <> -99)

\begin{verbatim}
    SEARCH(CODES, SIZE, CODE, FOUND, POSITION)
    /* CODES, FOUND and POSITION are
       pass-by-reference parameters;
       all others are pass-by-value  */

    if not FOUND then
        . . .
        . . .
        . . .
\end{verbatim}

(i) The subalgorithm SEARCH tests if the current subject code (stored in CODE) is already in array CODES. Explain how the parameters POSITION and FOUND will be effected by SEARCH if:

- CODE is already in array CODES
- CODE is not already in array CODES.

(ii) Construct subalgorithm SEARCH.

(iii) Explain how the contents of FREQS and CODES will be updated by the algorithm FREQUENCIES if:

- CODE is already in array CODES
- CODE is not already in array CODES.

(iv) Complete the algorithm FREQUENCIES.

(This question continues on the following page)
A possible solution to (b) (ii)

```
procedure SEARCH(CODES integer array, SIZE integer,
                 CODE integer, FOUND boolean,
                 POSITION integer)

    FOUND = false
    X = 0
    loop while (X < SIZE) and not FOUND
        X = X + 1
        if CODES[X] = CODE then
            FOUND = true
        endif
    end loop
    POSITION = X
endprocedure SEARCH
```

A possible solution to (b) (iv)

```
SIZE = 0
input CODE    // the input stream is terminated by -99
while (SIZE <= 750) and (CODE <> -99) do
    SEARCH(CODES, SIZE, CODE, FOUND, POSITION)
    // CODES, FOUND and POSITION are
    // pass-by-reference parameters; all others
    // are pass-by-value
    if not FOUND then
        SIZE = SIZE + 1
        CODES[SIZE] = CODE
        FREQS[SIZE] = 1
    else
        FREQS[POSITION] = FREQS[POSITION] + 1
    end if
endwhile
```
If-then-else Statements (M96 SL P2 Q5)

When evaluating the Boolean operators, and and or, in some circumstances the evaluation of the entire Boolean expression can be determined by the value of the first operand.

For example, the statement “S1 and S2” is false if S1 is false. Similarly, “S1 or S2” is true if S1 is true. In both of these cases, the value of S2 is irrelevant.

Rewrite the following algorithm fragments to use this strategy, testing only one operand at a time (e.g. if S1 then if not S1 then).

(a) Implement the statement below. No Boolean operators are necessary.

```plaintext
if S1 or S2 then
  ACTION1
else
  ACTION2
endif
```

(b) Implement the statement below. The Boolean operator not is required.

```plaintext
if S1 and S2 then
  ACTION1
else
  ACTION2
endif
```

A possible solution for (a):

```plaintext
if S1 then
  ACTION1
else
  if S2 then
    ACTION1
  else
    ACTION2
  endif
endif
```

Two possible solutions for part (b):

```plaintext
if S1 then
  if S2 then
    ACTION1
  else
    ACTION2
  endif
else
  if not S2 then
    ACTION2
  else
    ACTION1
  endif
endif
```
Consider the procedure below.

```
  input NUMBER    // assume 123 is entered
1   NEWVALUE = 0
2 loop while NUMBER > 0
3     DIGIT = NUMBER - truncate(NUMBER / 10) * 10
4     NEWVALUE = NEWVALUE * 10 + DIGIT
5     NUMBER = truncate(NUMBER / 10)
6   end loop
7   output NEWVALUE
```

Remember that \texttt{truncate}(769.84) returns 769, rounding DOWN to the nearest integer.

(a) Copy and complete the following trace table for the call \texttt{ALTER}(123).

<table>
<thead>
<tr>
<th>Line</th>
<th>NUMBER</th>
<th>NEWVALUE</th>
<th>NUMBER &gt; 0</th>
<th>DIGIT</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

(b) Describe the purpose of the procedure \texttt{ALTER}.

(c) State the output for \texttt{NUMBER} = -123

(d) Explain the difference if \texttt{ALTER} used a pass-by-reference parameter rather than a pass-by-value parameter.
Consider the algorithm fragment below. Remember that abs(-23.4) returns +23.4 and abs(1051.0) returns +1051.0.

```
input NUMBER1
input NUMBER2

if NUMBER1 >= 0.0 then
    if NUMBER1 < 1000.0 then
        NUMBER2 <-- 2 * NUMBER1
    endif
    if NUMBER1 <= 500.0 then
        NUMBER1 <-- NUMBER1 / 10.0
    endif
else
    NUMBER2 <-- 3 * NUMBER1
endif
else
    NUMBER2 <-- abs(NUMBER1)
endif
```

(a) State the values of NUMBER1 and NUMBER2 after this algorithm fragment is evaluated, given that the initial value of NUMBER1 is:

(i) 381.5
(ii) -21.0
(iii) 1200.0

(b) Complete the following algorithm for the function abs.

```
function abs(val NUMBER real)
result real
    declare ANSWER real

    . . . . . .
    . . . . . .
    . . . . . .

    return ANSWER
endfunction abs
```
Calculating Typing Fees (N96 SL P2 Q5)

In order to earn extra money, a student types extended essays for a fee. The amount charged depends on the number of pages in the document. The student charges:

- 5.00 (of some monetary unit) minimum fee for one to three pages
- 1.50 per page for each page over three pages
- an additional 3.75 if the number of pages exceeds 10.

Assuming that 200 words fit on a single typed page, a 1300 word extended essay would produce a fee of 11.00. That is, 1300 + 200 = 6.5 actual pages, for which the student charges 7 whole pages. The calculation is 5.00 (for the first 3 pages) + 1.50 x 4 pages (7 - 3) to produce a fee of 11.00.

(a) Calculate the fees, showing all working, for the following extended essays of length:

(i) 1000 words
(ii) 2425 words.

(b) Construct the algorithm fragment which a student can use to calculate the fee.

The algorithm fragment must prompt the student to give the number of words in the extended essay. The desired output will be:

- actual number of pages
- whole number of pages
- typing fee.

Remember that \textbf{truncate}(769.84) returns 769.

The output should be clearly labelled and all variables defined.

**Answers:**

(a)(i) 8.00  
(a)(ii) 23.75

(This question continues on the following page)
A possible solution for (b):

```
input NUMWORDS
TYPING = NUMWORDS / 200
if truncate(TYPING) = TYPING then
    PAGES = TYPING
else
    PAGES = truncate(TYPING) + 1
end if

FEE <-- 5
if PAGES > 3 then
    FEE <-- FEE +(PAGES - 3) * 1.5
end if
if PAGES > 10 then
    FEE <-- FEE + 3.75
end if

output "The actual number of pages is ", TYPING
output "A number of pages to be charged for is ", PAGES
output "The amount to be paid is ", FEE
```
The best weekly scores of golfers at a golf club are stored in two arrays as follows:

<table>
<thead>
<tr>
<th>NAME</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jenkins</td>
<td>82</td>
</tr>
<tr>
<td>Zendra</td>
<td>77</td>
</tr>
<tr>
<td>Lirmin</td>
<td>78</td>
</tr>
<tr>
<td>Jenkins</td>
<td>76</td>
</tr>
<tr>
<td>Furniss</td>
<td>81</td>
</tr>
<tr>
<td>Jenkins</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>

The array subscripts indicate the week of the score. For example, the best score at the club in week 3 was 78 which was made by Lirmin.

Consider the algorithm below.

```plaintext
LOC = 0
SUM = 0
NUMBER = 0

output "Enter a golfer’s name"
input PERSON
loop while (LOC <> 53) and (SCORE[LOC] <> -1) do
  if NAME[LOC] = PERSON then
    SUM = SUM + SCORE[LOC]
    NUMBER = NUMBER + 1
  end if
  LOC = LOC + 1
end loop

ANSWER = SUM / NUMBER
output ANSWER
```

(a) State what should be the dimension of the arrays.
(b) State the sentinel value and in what array it is to be found.
(c) The variable ANSWER has not been declared. State its data type and justify your answer.
(d) Trace the algorithm with the input data "Jenkins" for the array entries given.
(e) Describe the purpose of the algorithm.
(f) Some inputs will cause the algorithm to fail. Explain when this will happen and give an outline solution to stop the error.
The following algorithms are designed to calculate the sum (total) of a series of integers:

```plaintext
procedure SUM1
  declare SUM, VALUE integer
  SUM = 0
  input VALUE
  loop while VALUE > 0
    SUM = SUM + VALUE
    input VALUE
  end loop
  output SUM
endprocedure SUM1

procedure SUM2
  declare SUM, VALUE integer
  SUM = 0
  input VALUE
  loop
    SUM = SUM + VALUE
    input VALUE
  while VALUE >= 0
  output SUM
endprocedure SUM2

procedure SUM3
  declare SUM, VALUE, LENGTH, COUNT integer
  input LENGTH
  SUM = 0
  loop COUNT from 1 to LENGTH
    input VALUE
    SUM = SUM + VALUE
  end loop
  output SUM
endprocedure SUM3
```

For each algorithm (SUM1, SUM2 and SUM3):

(a) Explain what kind of integers (negative and/or positive) the loops are designed to accept and what effect they have on the loops.

(b) State how many times the loop is executed.
Binary Search (N97 SL P1 Q3)

The following procedure searches for a number.

```plaintext
procedure SEARCH(val MARKS integer array, val MAXLENGTH integer, val NUMBER integer)

// Assume that the array MARKS contains MAXLENGTH elements

FOUND = false
LOWER = 0
LENGTH = MAXLENGTH + 1
loop
  MIDDLE = LOWER + (LENGTH - LOWER) div 2
  if NUMBER = MARKS[MIDDLE] then
    FOUND = true
  else
    if NUMBER < MARKS[MIDDLE] then
      LENGTH = MIDDLE
    else
      LOWER = MIDDLE
    endif
  endif
while not FOUND
if FOUND then
  output MIDDLE
else
  output “Not found”
endif
endprocedure SEARCH
```

(a) Trace the call `SEARCH(GRADES, 11, 24)` by means of a trace table. Show the changes to the variables.

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>15</td>
<td>20</td>
<td>24</td>
<td>26</td>
<td>28</td>
</tr>
</tbody>
</table>
```

(b) State the output of the call `SEARCH(GRADES, 7, 24)`.

(c) There is a problem when searching for the number 21.

(i) Identify the problem.

(ii) State what can be added to the algorithm in order to solve this problem and indicate where it must be added.
Simulation to Count Fish (N97 SL P2 Q5)

A computer is used to monitor the number of fish passing under a sensor in a lake. The sensor has a timing device which sends the character “T” to the processor every minute. If a fish is detected, the sensor sends the character “F” to the processor. At the end of the timing period, the “T” is immediately followed by an “E”.

A typical data stream could be:

T F T T F F T F F T F E

This means that the first fish detected was between the first and second minute. Two fish passed under the sensor between the fourth and fifth minute, etc.

Construct an algorithm that inputs the data from the sensor and produces the following output:

- the number of minutes that the survey was in operation
- the total number of fish that passed under the sensor
- the largest number of fish that passed under the sensor in any one-minute interval.

The data that would be the output, for the above example, would be:

- the survey lasted 7 minutes
- a total of 8 fish passed under the sensor
- the largest number of fish in a one-minute time interval was 4.

(The pseudo-code instruction that reads a character from the SENSOR into the character variable SIGNAL is:

```
input(SENSOR, SIGNAL)
```
Simulation to Count Fish (N97 SL P2 Q5) (cont.)

A possible solution:

```
main FISH

FISHCOUNT = 0
MINUTES = 0
MAXFISH = 0
THISFISH = 0

input (SENSOR, SIGNAL)

loop while SIGNAL <> "E"
    if SIGNAL = "F" then
        FISHCOUNT = FISHCOUNT + 1
        THISFISH = THISFISH + 1
    else
        if SIGNAL = "T" then
            MINUTES = MINUTES + 1
            if THISFISH > MAXFISH then
                MAXFISH = THISFISH
            endif
            THISFISH = 0
        endif
    endif

    input (SENSOR, SIGNAL)

end loop
output "The survey lasted ", MINUTES, " minutes"
output "A total of ", FISHCOUNT, " fish passed under the sensor"
output "The Largest number of fish in a one-minute time interval was ", MAXFISH

endmain FISH
```
Highest Values in a Two-dimensional Array (M96 HL P1 Q10)

The two-dimensional array of real values TEMPS outlined below contains the maximum daily temperatures over a period of one year.

<table>
<thead>
<tr>
<th>DAYS</th>
<th>(WEEKS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[1]</td>
</tr>
<tr>
<td>[1]</td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>[7]</td>
<td>X</td>
</tr>
</tbody>
</table>

The cell indicated by the X, TEMPS (7, 2), would store the maximum temperature for Saturday in week 2.

By referring to this array, construct an algorithm that computes the highest and lowest maximum daily temperatures of the year (that is, the largest and smallest values in the array).

In addition to the temperature, the algorithm should clearly output the week of the year and the day of the week (specifying Sunday, Monday, etc.) on which these extreme temperatures occurred. (Assume that day 1 is Sunday, day 2 is Monday, etc.)

Define all variables used. (You may assume that valid real temperatures are already in the array TEMPS.)
A possible solution:

```
HIGH = TEMPS[1,1]
LOW = TEMPS[1,1]
LWEEK = 1
LDAY = 1
HWEEK = 1
HDAY = 1
loop X from 1 to 7
  loop Y from 1 to 52 do
    if TEMPS[X,Y] > HIGH then
      HIGH = TEMPS[X,Y]
      HWEEK = Y
      HDAY = X
    end if
    if TEMPS[X,Y] < LOW then
      LOW = TEMPS[X,Y]
      LWEEK = Y
      LDAY = X
    end if
  end loop
end loop
output "The highest temperature for the year is ", HIGH,
  " and it occurred in week ", HWEEK, " on a ">
  if HDAY = 1 then
    output "Sunday"
  else if HDAY = 2 then
    output "Monday"
  else if HDAY = 3 then
    ...
  else if HDAY = 7 then
    output "Saturday"
  end if
output "The lowest temperature for the year is ", LOW,
  " and it occurred in week ", LWEEK, " on a ">
  if LDAY = 1 then
    output "Sunday"
  else if LDAY = 2 then
    output "Monday"
  else if LDAY = 3 then
    ...
  else if LDAY = 7 then
    output "Saturday"
  end if
```
The algorithm fragment below generates values that are called the Fibonacci sequence.

```
P1 = 1
NEXT = 1
output P1
output NEXT
loop x from 1 to 10
  P2 = P1
  P1 = NEXT
  NEXT = P2 + NEXT
  output NEXT
end loop
```

(a) What are the Fibonacci numbers generated by this algorithm?

(b) Given the algorithm below (FIBSEQ), construct the recursive subalgorithm for the function FIBONACI (used near the end of the algorithm) that will generate the Nth Fibonacci number (when N >= 2).

Define and describe all parameters and variables used in the recursive algorithm.

**Hint:** the nth Fibonacci number [or FIBONACI (N)] is given by the (N - 1)th + the (N - 2)th Fibonacci number, for N > 2.

```
main FIBSEQ
  declare N, FIB integer
  output "Enter the number, n, greater than 2, for the n<sup>th</sup> Fibonacci number to be generated."
  input N
  loop while N <= 2
    output "The number, n, must be greater than 2. Please re-enter."
    input N
  end loop
  FIB = FIBONACI(N)
  output "The n<sup>th</sup> Fibonacci number is ", FIB
endmain FIBSEQ
```

(This question continues on the following page)
A possible solution for (b):

```pascal
function FIBONNACI(val N integer)
result integer

    if N = 1 then
        return 1
    else
        if N = 2 then
            return 1
        else
            return FIBONNACI(N - 1) + FIBONNACI(N - 2)
        endif
    endif
endfunction FIBONNACI
```
**Binary Search (N96 HL P2 Q1)**

(a) To solve a particular computer problem, data has to be stored in sorted order.

If an array is used, a sorted order can be obtained as the data is entered initially, for example, if the input data is:

Grapefruit, Apple, Banana, Pear, Orange, Grape, Kiwi.

If the array to store them is FRUIT, after the first read it would store:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Grapefruit</td>
</tr>
</tbody>
</table>

After the second read, it would store:

<table>
<thead>
<tr>
<th>Apple</th>
<th>Grapefruit</th>
</tr>
</thead>
</table>

(i) Draw the array to show how it would look after every subsequent read.

(ii) Draw and describe a data structure which would not require as much data rearrangement, when a new item is added, to maintain a sorted order. Give brief details of its organization.

(b) A binary search can be used to locate a data item in a sorted array of size N. It does this by comparing the middle item of a list with the required input value. If the input is equal to the middle item of the list the search stops, otherwise the half of the list which cannot contain the item is ignored by reassigning the current value for the top or bottom of the list.

For example, using the final array from (a)(i) above, where $N = 7$ and the item to find is Kiwi:

Initially bottom is 1 and top is 7 (that is, $N$).

<table>
<thead>
<tr>
<th>Apple</th>
<th>Banana</th>
<th>Grape</th>
<th>Grapefruit</th>
<th>Kiwi</th>
<th>Orange</th>
<th>Pear</th>
</tr>
</thead>
</table>

Thus $\text{MIDDLE} = (\text{TOP} + \text{BOTTOM}) / 2 = 4$

Since the item at location 4 (Grapefruit) is not equal to the input value (Kiwi) the search does not stop and since Grapefruit is less than Kiwi, the input value cannot be in the lower half of the list, so bottom is reassigned to $\text{MIDDLE} + 1$ and the search continues.

(i) Continue the trace of the binary search, using a suitable layout.

(ii) Construct the algorithm fragment to perform the binary search. (Remember to give an error message if the item is not found.)
(c) The binary search can be expressed as a recursive routine.

(i) Explain the term recursion and state one necessary condition required by any recursive routine.

(ii) A call to a recursive binary search routine when looking in array FRUIT for item SEEK could be:

\[ \text{BSEARCH} (\text{SEEK}, \text{BOTTOM}, \text{TOP}, \text{FRUIT}). \]

Construct the recursive algorithm fragment that would carry out this call. No loop is used, instead the routine is used recursively with suitable parameters.

A possible solution for (b)(ii):

```
procedure BINSEARCH(val SEEK string, val BOTTOM integer, 
val TOP integer, val FRUIT string array)

/* Looking for the occurrence of SEEK within FRUIT */
declare MIDDLE integer
repeat
    MIDDLE <-- (BOTTOM + TOP) div 2
    if FRUIT[MIDDLE] = SEEK then
        output "Found"
    else
        if FRUIT[MIDDLE] > SEEK then
            TOP <-- MIDDLE - 1
        else
            BOTTOM <-- MIDDLE + 1
        endif
    endif
until (FRUIT[MIDDLE] = SEEK) or (TOP < BOTTOM)
if TOP < BOTTOM then
    output "Error"
endif
endprocedure BINSEARCH
```
A possible solution for (c) (ii):

```plaintext
procedure BSEARCH(val SEEK string, val START integer, 
        val FINISH integer, val FRUIT string array)

/* Looking for the occurrence of SEEK within FRUIT recursively */

declare MIDDLE integer

if FINISH < START then
    output "Error"
else
    MIDDLE <-- (START + FINISH) div 2
    if FRUIT[MIDDLE] = SEEK then
        output "Found"
    else
        if FRUIT[MIDDLE] > SEEK then
            BSEARCH(FRUIT, START, MIDDLE - 1)
        else
            BSEARCH(FRUIT, MIDDLE + 1, FINISH)
        endif
    endif
endif
endif
endprocedure BSEARCH
```

**Binary Tree (N96 HL P2 Q7)**

A binary tree is stored in a dynamic data structure as follows. Each node is given in the format:

```
+  /  B
327 350 -1
```

- Problem continues on the following page.
(a) State what the values associated with left_link and right_link represent.

(b) Draw the binary tree presented above if the root node is 380.

(c) A recursive routine is used to traverse this tree. Assume the new type NODE has been defined as given below.

```
newtype NODE record
   NODE_ENTRY character
   LEFT_LINK pointer->NODE
   RIGHT_LINK pointer->NODE
endrecord
```

```
procedure TRAVERSE(val VALUE pointer->NODE)
   if VALUE->LEFT_LINK ≠ -1 then
      TRAVERSE(VALUE->LEFT_LINK)
   endif
   if VALUE->RIGHT_LINK ≠ -1 then
      TRAVERSE(VALUE->RIGHT_LINK)
   endif
   output VALUE->NODE_ENTRY
endprocedure TRAVERSE
```

Trace the algorithm with the initial call TRAVERSE(380).

(d) Give the in-order traversal of the tree given in (b).

(e) State the changes needed to be made to the algorithm in (c) to generate an in-order traversal of a tree.

**Summing in a Two-dimensional Array (N97 HL P1 Q10)**
The two-dimensional array, called GRID, has the following values.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>12</td>
<td>19</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>15</td>
<td>21</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>17</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>12</td>
<td>18</td>
<td>9</td>
</tr>
</tbody>
</table>

For example, GRID[3, 2] = 14.

Consider the algorithm fragment below:

1. for ROW <-- 1 upto 4 do
   2. for COL <-- 2 upto 5 do
   3. if GRID[ROW, COL] < GRID[ROW, COL-1] then
   4. GRID[ROW, COL] <-- GRID[ROW, COL-1]
   5. endif
   6. endfor
   7. endfor

(a) Redraw the array and its new contents after tracing the algorithm above.

(b) Construct the algorithm that displays both the average (mean value) of each column and the overall average.

_A possible solution for (b):_

```plaintext
procedure AVERAGE(ref GRID integer array [1..4,1..5])
declare ROW, COL, COLSUM, ALLSUM integer
declare COLAVG, ALLAVG real
ALLSUM <-- 0
for COL <-- 1 upto 5 do
   COLSUM <-- 0
   for ROW <-- 1 upto 4 do
      COLSUM <-- COLSUM + GRID[ROW, COL]
      ALLSUM <-- ALLSUM + GRID[ROW, COL]
   endfor
   COLAVG <-- COLSUM / 4
   output "In column ", COL, " the average is ", COLAVG
endfor
ALLAVG <-- ALLSUM / 20
output "The overall average is", ALLAVG
endprocedure AVERAGE
```

**Random Numbers (N97 HL P2 Q1)**

One method of generating pseudo-random numbers is to start with two values (seeds), multiply them together and then extract the middle digits. This value can then be used as a seed to generate the next value.

For example, if 85 and 65 are used as the seeds, the following is obtained:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Seed 1</th>
<th>Seed 2</th>
<th>Product</th>
<th>Pseudo-random Number</th>
</tr>
</thead>
</table>

---

Teacher Support Material - Computer Science © IBO, 4/2/2015  25
(a) Complete the table above to show up to iteration 6.

(b) Construct the algorithm $\text{PSEUD}$ that will calculate and display the first six pseudo-random numbers. You may assume that there is a function $\text{EXTRACT}$ which returns the middle two digits from a four digit number. For example, $\text{EXTRACT}(5525)$ would return 52.

(c) Construct the algorithm for $\text{EXTRACT}$ as used in part (b). Remember that $\text{truncate}(43.8)$ returns 43.

(d) Explain what would happen if the product was 2008.

(e) Another way to generate pseudo-random numbers is to use modulo arithmetic. For example, in a sequence of values the new number is calculated by multiplying the last generated random value by a set number and using the remainder after dividing by another set value.

For example, with an initial seed of 4, a multiplier of 3, and a modulus of 7, the following is obtained:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Pseudo-random Number ($R_n$)</th>
<th>$3R_n$</th>
<th>$R_{n+1} = (3R_n) \mod 7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

(i) Construct the recursive algorithm $\text{PSEUD2}(\text{ITER, NUM})$ which outputs any value in the sequence. For example, the call $\text{PSEUD2}(2, 4)$ would output 5, where the first parameter represents the sequence number required and the second parameter represents the seed.

(ii) Construct $\text{PSEUD2}$ as an iterative, rather than a recursive, algorithm.

(iii) State one advantage and one disadvantage of recursive routines when compared to iterative routines.

---

A possible solution for (b):

```main PSEUD
declare SEED1, SEED2, X, PRODUCT, RANDOM integer
SEED1 <-- 85
SEED2 <-- 65
for X <-- 1 upto 6 do
    PRODUCT <-- SEED1 * SEED2
```
RANDOM <-- EXTRACT(PRODUCT)
output RANDOM
SEED1 <-- SEED2
SEED2 <-- RANDOM

endfor
endmain PSEUD

A possible solution for (c):

function EXTRACT(val VALUE integer)
result integer
declare HUN, LEFT integer
VALUE <-- truncate(VALUE / 10)
HUN <-- truncate(VALUE / 100)
HUN <-- HUN * 100
LEFT <-- VALUE - HUN
return LEFT
endfunction EXTRACT

A possible solution for (e) (i):

procedure PSEUD2(val ITER integer,val NUM integer)
    if ITER = 1 then
        output NUM
    else
        PSEUD2(ITER - 1, (3 * NUM) mod 7)
    endif
endprocedure PSEUD2

A possible solution for (e) (ii):

procedure PSEUD2(val ITER integer,val NUM integer)
declare RES, TIMES integer
RES <-- NUM
if ITER > 1 then
    for TIMES <-- 2 upto ITER do
        RES <-- RES * 3 mod 7
    endfor
endprocedure PSEUD2
endfor
endif
output RES