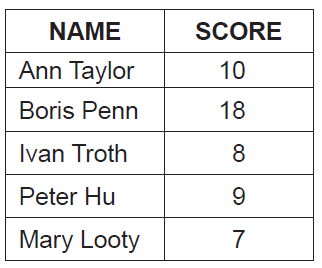
**HL Unit 5** **– Abstract Data Structures**  
Test 1

1. The table below holds student names and scores, from a class test.
2. Draw a diagram to show how the data given in the table could be stored in a binary tree

in the order of scores. Data should be inserted into the binary tree in the order given in

the table (ie data about Ann Taylor is to be inserted first). [3]

*Award marks as follows up to* ***[3 max]****.*

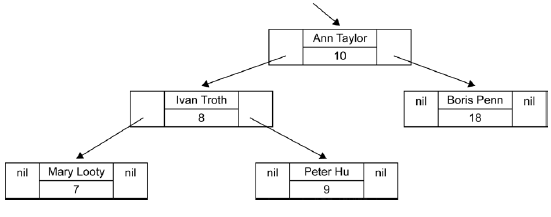
*Award* ***[1]*** *for correct nodes (2 data fields and two pointers)*.

*Award* ***[1]*** *for correct root*.

*Award* ***[1]*** *for correct left subtree*.

*Award* ***[1]*** *for correct right subtree*.

Award ***[1]*** for trees that display either only names or only numbers, provided that they are structurally correct.



1. The same data could be inserted into a singly linked list in descending order of scores.

Draw a diagram of this singly linked list. [3]

*Award marks as follows up to* ***[3 max]****.*

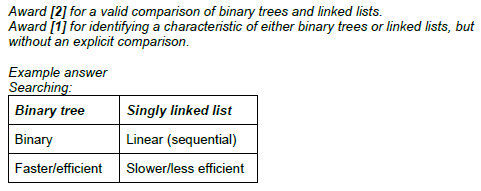
*Award* ***[1]*** *for correct nodes (2 data fields and one pointer field)*.

*Award* ***[1]*** *for correct order (links shown)*.

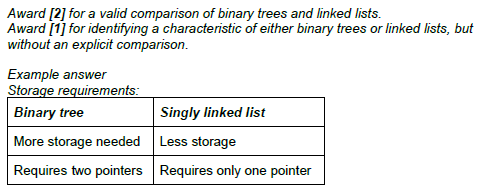
*Award* ***[1]*** *for the external pointer to list and showing the end of the list (nil)*.

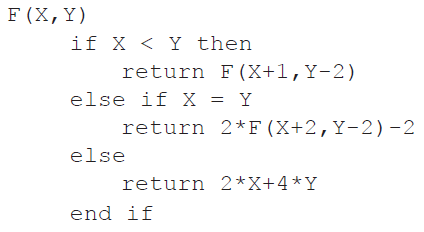


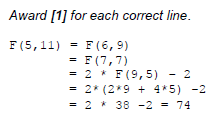
1. Compare the data structures in part (a) and part (b) in terms of:  
     
   (i) searching [2]

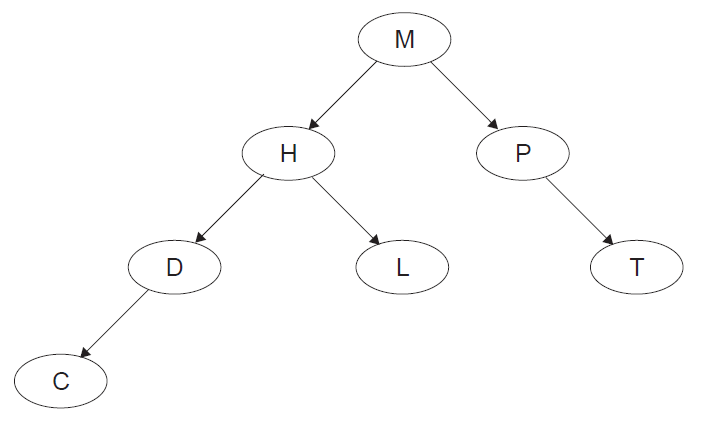


(ii) storage requirements. [2]



1. Consider the following **recursive** algorithm, in which X and Y are parameters in the  
   method F. The return statement gives the value that the method generates.  
     
     
     
     
     
     
     
     
     
   Determine the value of F(5,11). [5]

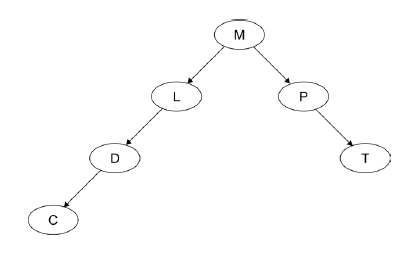


1. Consider the following binary tree.
2. State the order that the nodes will be listed using the postorder tree traversal. [1]

CDLHTPM

1. The node H is deleted so that the postorder traversal of the **remaining** nodes is

preserved from part (a). Sketch the updated binary tree following this deletion. [2]



1. A laptop computer supplements its primary memory by making use of virtual memory.  
   The laptop has 1GB of random access memory (RAM) and a single processor.  
   The laptop is using one of the latest operating systems to run multimedia gaming programs.

One of the laptop’s game applications stores the data relating to the different actions of one  
of its characters in a stack.

1. Suggest **one** reason why the character’s actions might be stored in a stack. [2]

A stack is a LIFO data structure;

*In the game it allows the “undo” function to work / movements/steps can be retraced;*

***Note****: Allow any reasonable answer for the second marking point.*

At specific moments during the game this data is read from the stack (S) into a queue (Q).

1. Using appropriate access methods for stacks and queues, construct an algorithm that

reads the data from the stack and enters it into the queue. You should assume that the

queue structure exists and that both structures are of a fixed size. [6]

*Award up to* ***[6 max]*** *as follows.*

*Example answer 1:*

*Award* ***[1]*** *for while loop;*

*Award* ***[2]*** *for the correct condition in while statement (stack is not empty* ***and*** *queue is not full) award* ***[1]*** *in case of the minor error;*

*Award* ***[1]*** *for pop. (****Note:*** *Do not allow alternatives)*

*Award* ***[1]*** *for enqueue; (****Note****: Allow alternatives)*

*Award* ***[1]*** *for the output only if stack is not empty after the loop;*

loop while NOT S.isEmpty() and NOT Q.isFull()

N = S.pop()

Q.enqueue(N)

end loop

if NOT S.isEmpty() *//accept NOT S.isEmpty()and Q.isFull()*

output “incomplete process”

end if

*Example answer 2:*

*Award* ***[1]*** *for the correct use of flag throughout.*

*Award* ***[1]*** *for the while loop*

*Award* ***[1]*** *for checking !(S.isEmpty())*

*Award* ***[1]*** *for pop. (****Note:*** *Do not allow alternatives.)*

*Award* ***[1]*** *for checking if Q is full*

*Award* ***[1]*** *for enqueue; (****Note****: Allow alternatives.)*

*Award* ***[1]*** *for output*

FLAG = true

loop while !(S.isEmpty())

N = S.pop()

if !(Q.isFull()

Q.enqueue(N)

else

FLAG = false

break

// can be missed out or alternative used

end if

end loop

if (FLAG = false)

output “incomplete process”

end if

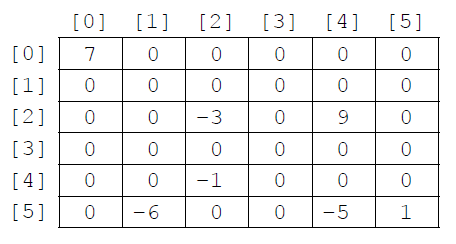
1. Outline **one** advantage of making the queue dynamic. [2]

*Award* ***[2]*** *marks,* ***[1]*** *for advantage,* ***[1]*** *for expansion.*

A dynamic queue would not have predetermined/fixed size;

Memory is allocated as required/efficient use of memory/flexible size//there is always sufficient memory to accommodate the queue;

1. Consider the following two-dimensional array, MAT, with dimensions 6 × 6.



The value −1 is stored in MAT at position [4][2]. The position [4][2] means row 4  
 and column 2.

1. State the total number of elements stored in MAT. [1]

36

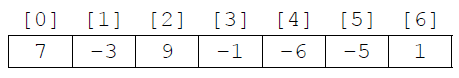
1. State the number of non-zero elements in MAT. [1]

7

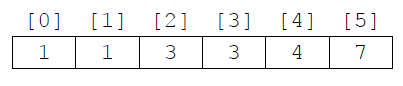
A two-dimensional array in which most of the elements are zero is called a **sparse matrix**. A sparse matrix can be compressed by storing only non-zero elements using three one‑dimensional arrays.

The **first array**, VALUES, stores all non-zero elements taken from the sparse matrix in row‑major order (left-to-right then top-to-bottom order).

The length of the array VALUES is equal to the number of non-zero elements in the sparse matrix. For the sparse matrix above, MAT, the array VALUES is:

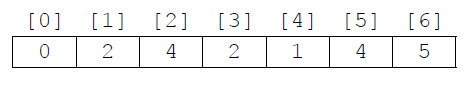


The **second array** is ROWC. ROWC[i] stores the number of non-zero elements, from row 0 to row i of the sparse matrix, **inclusive**.

The length of ROWC is equal to the number of rows in the sparse matrix. For MAT the array ROWC is:

For example, ROWC[2] stores 3 because in MAT there are three non-zero elements from row 0 to row 2, inclusive.

The **third array**, COL, stores the column index for each non-zero element in the sparse matrix. COL[i] stores   
the sparse matrix column index for the non-zero element stored in VALUES[i]. For MAT the array COL is:



1. Construct an algorithm that compresses a 6 × 6 two-dimensional array, such as MAT,   
   into the three one-dimensional arrays described on page 8. You may assume that the   
   6 × 6 array is inputted and all three one-dimensional arrays are initialized. [6]

*Award marks as follows up to* ***[6 max]****. (There are 7 marking points)*

*Award* ***[1]*** *for initialization and correct changes of K (index/position in arrays*

*VALUES and COL);*

*Award* ***[1]*** *for initialization and correct changes of COUNT (counts non-zero*

*elements);*

*Award* ***[1]*** *for correct conditions in “row” loop;*

*Award* ***[1]*** *for correct conditions in “column” loop;*

*Award* ***[1]*** *for placing non-zero element at correct position in array VALUES;*

*Award* ***[1]*** *for placing the “column” index of the non-zero element at correct*

*position in array COL;*

*Award* ***[1]*** *for placing COUNT at correct position in array ROWC;*

COUNT = 0

K = 0

loop for I from 0 to 5

loop for J from 0 to 5

if MAT[I][J]! = 0 then

VALUES[K] = MAT[I][J]

COL[K] = J

K = K + 1

COUNT = COUNT + 1

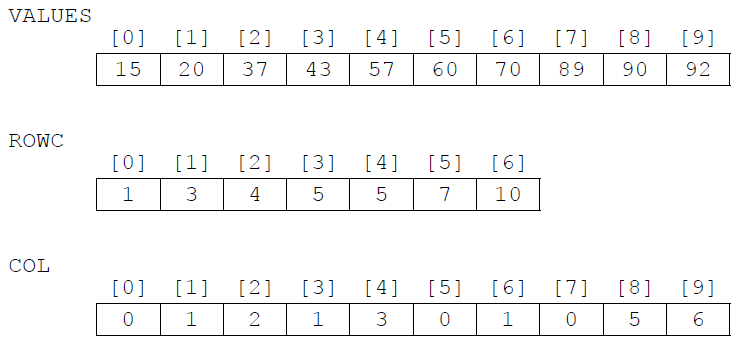
end if

end loop

ROWC[I] = COUNT

end loop

Consider the following three arrays. They hold the compressed contents of a 7 × 7 sparse   
 matrix, BIGMAT.



1. For a given column, C, in BIGMAT, outline how it could be determined that this column contains no non-zero elements. [2]

*Award up to* ***[2 max]****.*

***Example answer 1***

Array COL should be searched for (value) C;

If (value) C is not found in array COL then this column (the column whose index in

BIGMAT is C) holds only zeros;

***Example answer 2***

If the number of occurrences of (value) C in array COL;

Equals zero then this column holds only zeros;

***Example answer 3****: Award* ***[1 only]****.*

If COL[C] = COL[C–1], then the column with index C in BIGMAT contains no

non-zero-elements;

*(Accept words to that effect: “if the difference between COL[C] and COL[C-1] is*

*zero, then...”);*

1. State how many rows in BIGMAT contain only zeros. [1]

1

1. (i) State the index in VALUES of the first non-zero element in row 5 of BIGMAT. [1]

5

(ii) For a given row, R, in BIGMAT, determine the range of indexes in VALUES where  
 non-zero elements in row R of BIGMAT are placed. You may assume that there  
 is at least one non-zero element in row R. [3]

*Award marks as follows up to* ***[3 max]****.*

*Award* ***[1]*** *for realizing that the range should be determined differently for*

*the first row (when row index R is 0) OR correct range when row index is 0;*

*Award* ***[1]*** *for correct first index in range (when row index R is not 0);*

*Award* ***[1]*** *for correct last index in range;*

If row index R is equal to 0 then the range is from 0 to ROWC[0]-1 ;

If row index R is not equal to 0 then the range is from ROWC[R-1];

To ROWC[R]-1;

***Note****: Award* ***[2 max]*** *for a correct calculation of the indexes, but no*

*unifying expression showing how they have been calculated is given).*