NATIONAL JUNIOR COLLEGE
SENIOR HIGH 2 COMMON TEST
HIGHER 2

COMPUTING **9569/02**

Paper 2 (Lab-based) **25 May 2021**

3 hours

Additional Materials: Electronic version of `FILE_LIST.CSV` data file
Electronic version of `TIER_3_UPDATE.CSV` data file
Electronic version of `IGP.CSV` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Note that up to **6 marks out of 100** will be awarded for the use of common coding standards for programming style.

The number of marks is given in the brackets [ ] at the end of each question or part question. The total number of marks for this paper is 100.

This document consists of **17** printed pages.

[Turn Over

Instructions to candidates:

Copy the folder from the thumb drive to the PC's desktop and rename the folder on the desktop to `<your name>_<NRIC number>`. (For example, `TanKengHan_T0123456A`). All the resource files are found in the folder and you should work on the folder in the desktop.

Your program code and output for each of Task 1 to 5 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as `Task_1_<your name>_<NRIC number>.ipynb`.

You should have a total of **five** `.ipynb` files to submit at the end of the paper. At the end of the exam, copy the working folder on your desktop to the thumb drive.

Task 1

A hierarchical file storage management is to be implented which allows files on a computer to be physically stored in 3 tier of storage:

- Tier-1 is an online hard disk drive attached to the computer.
- Tier-2 is a online cloud storage provided by a cloud storage provider.
- Tier-3 is an offline DVD storage.

A file listings of all the files stored on the computer is given in the file, `FILE_LIST.CSV`.
Each line contains the **filename**, **time-stamp** and the **size** of the file in bytes.
The time-stamp is in the following format : `"dd/mm/yy HH:MM"` , where `dd` is the 2 digit day, `mm` is the 2 digit month, `yy` is the 2 digit year and `HH:MM` is the hour and minute in 24 hour format.
For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1] : #Task 1.1
         Program Code
         Output:
```

The files provided are `FILE_LIST.CSV`, `TIER_3_UPDATE.CSV`.

Task 1.1

Write Python code to read the contents of the file `FILE_LIST.CSV` and group the file listings into 3 **Python Lists**, tier-1, tier-2 and tier-3 for processing and then output into 3 files according to the following requirements:

Group	Description	Output
Tier 1 files	The last-modified time-stamps on these files must be less than 100 days from the current time-stamp.	The file listings in this group is to be sorted in descending order of the last-modifed time-stamp (latest modified files will appear first in the list) and saved in a file named `TIER-1.CSV`
Tier 2 files	The last-modified time-stamps on these files must be greater or equal to 100 days but less than 365 days.	The file listings in this group is to be sorted in descending order of the last-modifed time-stamp (latest modified files will appear first in the list) and saved in a file named `TIER-2.CSV`
Tier 3 files	The last-modified time-stamps on these files must be equal or greater then 365 days from the current time-stamp.	The file listings in this group is to be sorted in descending order of the last-modifed time-stamp (latest modified files will appear first in the list) and saved in a file named `TIER-3.CSV`

The sorting of the file-listings must be done using a **quick-sort** algorithm. You **cannot** use any build-in sorting functions. The datetime string format should be in the form `"dd/mm/yy HH:MM"` , where `dd` is the 2 digit day, `mm` is the 2 digit month, `yy` is the 2 digit year and `HH:MM` is the hour and minute in 24 hour format.

For example the contents of file `TIER-3.CSV` should look like this:

```
WMSysPr9.prx,07/12/19 17:53,316640
DtcInstall.log,07/12/19 17:17,776
Professional.xml,07/12/19 17:10,30831
⋮
pyshellext.amd64.dll,05/12/17 20:42,55456
system.ini,29/09/17 21:44,219
CoreSingleLanguage.xml,29/09/17 21:42,35138
```

[15]

Task 1.2

The file names in the Windows operating system have two parts; the file's name, then a period followed by the extension (suffix). The extension is an abbreviation that signifies the file type. Extensions are important because they tell your computer what icon to use for the file, and what application can open the file.
The following examples show the filenames and their extensions:

File name	File extension
`bootstat.dat`	`dat`
`pyshellext.amd64.dll`	`dll`
`Prog.is.fun.pdf`	`pdf`
`..txt`	`txt`
`Modules`	'' (empty string)
`.`	'' (empty string)
`..`	'' (empty string)

Implement the following function using Python code to extract the file extension from the given file name:

```
FUNCTION get_extension(s:  STRING)RETURNS STRING
```

The function must support the test cases in the example file names shown above. [4]

Task 1.3

A MongoDB database named `file_stats` is to be used to store statistics about the file system. A MongoDB collection named `storage` is to be created to store documents based on the storage tier level. An example of a document in the `storage` collection looks like this:

```
{"storage_tier":1,
"file_count":14,
"file_names":["WindowsUpdate.log","bootstat.dat", "setupact.log","PFRO.log","explorer.exe",
"notepad.exe", "HelpPane.exe","regedit.exe","bfsvc.exe","splwow64.exe", "diagerr.xml",
"diagwrn.xml","comsetup.log","setuperr.log"],
"file_types":{"log":  5, "dat":  1, "exe":  6, "xml":  2}
}
```

The description of the attributes used in the document is as follow:

Attribute name	Value description
`storage_tier`	Storage tier number
`file_count`	Number of files in the storage tier
`file_names`	List of filenames in the storage tier
`file_types`	Document object containing the file extensions and the number of files with the file extension in the storage tier. Example in the document shown above, storage_tier 1 has 5 files with extension log

Using the 3 files, `TIER-1.CSV`, `TIER-2.CSV` and `TIER-3.CSV` created in Task 1.1 and the function `get_extension` implemented in Task 1.2 write Python code to :

- create the MongoDB database and collection.
- create the documents based on the required statistics given above. [5]

Task 1.4

Assuming that the file `TIER_3_UPDATE.CSV` is generated using the code implementation in **Task 1.1** at a future date from the current date and time. It contains the file listings for Tier-3 storage.

Write Python code to update the MongoDB database that you have created in **Task 1.3**. You may re-use your code in **Task 1.1** to **Task 1.3**. [2]

Name and save your Jupyter Notebook as `Task1_<your name>_<NRIC number>.ipynb`.

Task 2

A **max-heap** is a binary tree **(NOT BST)** such that the value in each node is greater than or equal to the values of the left and right child of that node. The root node is therefore the node with the highest value in the tree.

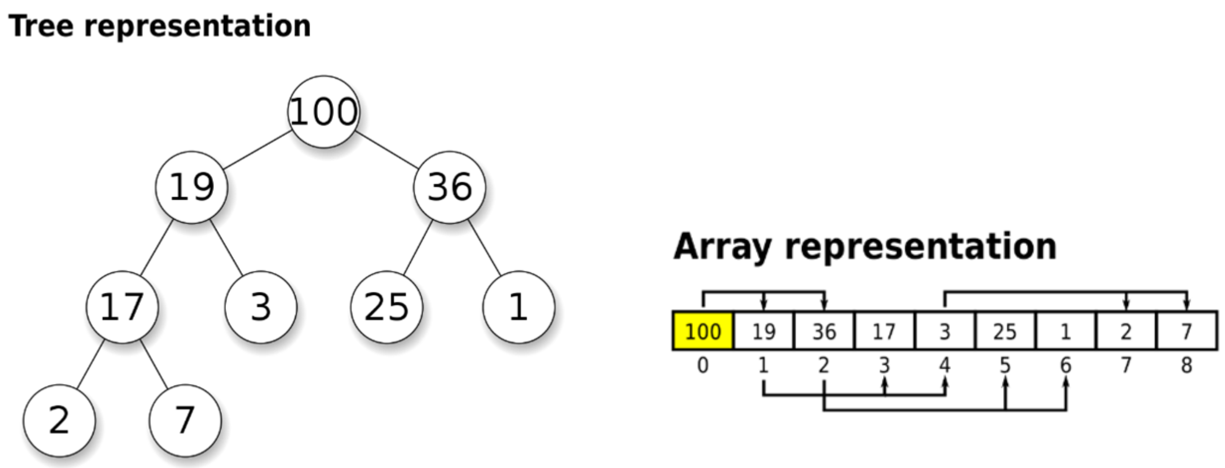
The class `Max_Heap` is implemented using a fixed-size 1D array with each child node index calculated using the following formula:

```
left_child_index   =  current_node_index*2+1,
right_child_index  =  current_node_index*2+2.
```

and the parent node index of an internal node (i.e., not root node) is calculated as follows:

```
parent_index  =  (current_node_index-1) DIV 2.
```

An example of a max-heap is shown below:



class `Max_Heap` attributes	Description
`count: INTEGER`	It stores the number of data item currently in `Max_Heap`.
`size: INTEGER`	It stores the maximum number of data item minHeap can take.
`tree: ARRAY[0:N-1] OF Node`	It is a 1D array that stores the data items as Nodes in `Max_Heap`. If a data item doesn't exist it is represented by `None`. The value attribute in the `Node` object will determine the index position of the `Node` object in the array.

class `Max_Heap` methods	Description
`constructor(N) RETURNS None`	`N`, the number of nodes in the tree, is passed as an argument in the constructor of the class.
`__str__()`	Converts the `Max_Heap` object into a string representation in the form "`[<Node>,<Node>,<Node>]`" where `<Node>` is the string representation of the `Node` objects in the tree array. `None` elements should not be displayed.

class `Node` attributes	Description
`name: STRING`	A sring to represent the `Node` object.
`value: INTEGER`	The data value of the `Node` object. This value is used to determine the position where the `Node` object is stored in the `Max_Heap` tree.
class `Node` methods	
`constructor(n, v)`	The `n` and `v` arguments are passsed in to initialise the `Node` attributes `name` and `value` respectively.
`__str__()`	Converts the `Node` object into a string `"<name>:<value>"` where `<name>` and `<value>` are the `Node` attribues name and value respectively.

For each sub-task, add a comment statement, at the beginning of the code using the hash symbol "#", to indicate the sub-task the program code belongs to, for example:

```
In [1] :  #Task 2.1
          Program Code
          Output:
```

Task 2.1

Write the Python code to implement the class `Max_Heap` and the class `Node` according to the requirements above. [6]

Task 2.2

Write the Python code to implement the `insert` and `get_max` methods in the class `Max_Heap`.

class `Max_Heap` methods	Description
`insert(Node) RETURNS Boolean`	Inserts the `Node` object into the `Max_Heap` tree. Returns `True` if insert is successful and `False` if insert fails. This is an in-place operation.
`get_max() RETURNS Node`	Returns the `Node` object with the highest data value. This should be a $O(1)$ operation.

[6]

Task 2.3

Write the Python code to implement the `in_order` and `all_paths` methods in the class `Max_Heap`. Note that if you fail to implement insert(Node) in **Task 2.2**, you may hard-code an instance of the `Max_Heap` object to implement the rest of the code in **Task 2.3** to **Task 2.4**.

class `Max_Heap` methods	Description
`in_order() RETURNS None`	Displays all the nodes in the tree using an in-order traversal.
`all_paths() RETURNS None`	Displays all paths from the root of the tree to all its leaves. The format of the path string is shown in the example in **Task 2.4**.

[10]

Task 2.4

Use the test cases in the code snippet below to test the implementation of your code in Task 2.1 to Task 2.3:

```
heap = Max_Heap(7)
heap.insert(Node("alpha",19))
heap.insert(Node("beta",17))
heap.insert(Node("charlie",100))
heap.insert(Node("delta",50))
heap.insert(Node("echo",65))
heap.insert(Node("fox",16))
```

The output for `heap.in_order()` should look like this:

```
charlie:100
echo:65
beta:17
delta:50
alpha:19
fox:16
```

The output for `heap.all_paths()` should look like this:

```
charlie:100,echo:65,beta:17
charlie:100,echo:65,delta:50
charlie:100,alpha:19,fox:16
```

Name and save your Jupyter Notebook as `Task2_<your name>_<NRIC number>.ipynb`. [2]

Task 3

A **fix-sized** stack is to be implemented to store characters.

Task 3.1

Write the Python code to implement the stack and the operations specified below.

Your code should allow operations to:

- push an item on to the stack only if the stack is not full.
- pop an item off the stack only if the stack is not empty.
- determine the number of items currently in the stack. [5]

The stack is to be used to identify if an expression is balanced.

An expression is balanced if each opening bracket has a corresponding closing bracket. Different pairs of brackets can be used. These are `()`, `[]` or `{}`.

Example of a balanced expression:

```
([9-1]/2)+{3*7}
```

Example of unbalanced expressions:

```
[(9-1)+{3*7}
```

```
(9-1])+{3*7}
```

Task 3.2

Implement a Python function that makes use of the stack implemented in **Task 3.1** to check if a given expression is balanced :

```
FUNCTION is_balanced(exp :  STRING) RETURNS BOOLEAN
```

where `exp` is the expression to be evaluated.

The function returns `TRUE` if the expression is balanced and `FALSE` if it is not balanced. [5]

Task 3.3

Write Python code to test your function using the following test cases:

Input	Expected return value
`([9-1]/2)+{3*7}`	`TRUE`
`[(9-1)+{3*7}`	`FALSE`
`[(9-1])+{3*7}`	`FALSE`

Add one boundary test case and one invalid test case to test your function.

[2]

Name and save your Jupyter Notebook as `Task3_<your name>_<NRIC number>.ipynb`.

Task 4

In mathematics, a $m \times n$ matrix is a rectangular array of numbers arranged in m rows and n columns, e.g., the following is a 3×3 matrix

$$\begin{pmatrix} 1 & 2 & 6 \\ -1 & 3 & 0 \\ 2 & 5 & 13 \end{pmatrix}.$$

It is customary to denote a matrix using a capital letters in bold, e.g., $\mathbf{A}$ and the entry in row i and column j in a matrix $\mathbf{A}$ is as the lower case of the letter representing the matrix followed with the subscript ij, e.g., in this case, it is a_{ij}. Furthermore, we take the example matrix above : $a_{11} = 1$, $a_{23} = 0$ and $a_{32} = 5$.

Let $\mathbf{A}$ be an $m \times n$ matrix and $\mathbf{B}$ be an $n \times p$ matrix, i.e.,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}.$$

The **product** of the matrices, denoted as $\mathbf{C} = \mathbf{AB}$, is the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}.$$

The **determinant** of $\mathbf{A}$, denoted as $|\mathbf{A}|$, is a scalar value dependent on the dimension of the matrix.

- if $\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ is a 2×2 matrix, $|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$.
- if $\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ is a 3×3 matrix, $|\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}$ and

$$|\mathbf{A}| = a_{11}\left(a_{22}a_{33} - a_{23}a_{32}\right) - a_{12}\left(a_{21}a_{33} - a_{23}a_{31}\right) + a_{13}\left(a_{21}a_{32} - a_{22}a_{31}\right).$$

The **adjugate** of $\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$, denoted as adj $(\mathbf{A})$, is the 3×3 matrix

$$\text{adj}\,(\mathbf{A}) = \begin{pmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ -\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{pmatrix}.$$

A system of 3 linear equations with unknown variables x_1, x_2, x_3 and real coefficients a_{ij}, $1 \leq i, j \leq 3$ has the following form:

$$\begin{array}{cccc} a_{11}x_1 & +a_{12}x_2 & +a_{13}x_3 & = & -1 \\ a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & = & 5 \\ a_{31}x_1 & +a_{32}x_2 & +a_{33}x_3 & = & 6. \end{array}$$

Consequently, the equations can be represented with a matrix equation

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ 5 \\ 6 \end{pmatrix}.$$

A linear system has a unique solution if the the determinant of the matrix of the coefficients of the linear expressions is nonzero and it is given by

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \frac{1}{|\mathbf{A}|}\text{adj}\,(\mathbf{A}) \begin{pmatrix} -1 \\ 5 \\ 6 \end{pmatrix}.$$

Otherwise, the system could either have no solution or infinite number of solutions.

A program is required to determine if the unique solution to the linear system above exists.

Task 4.1

Without using `numpy` or any similar modules, write a program code with the following specification:

- Input a matrix as a nested list
- Validate the input
- Calculate the product of two matrices (write this code as a function)
- Calculate the unique solution to the linear system if it exists (write this code as a function)
- Output the unique solution.
- Output the string "Unique solution does not exist" if the unique solution does not exist. [11]

Task 4.2

Draw up a list of **three** suitable test cases. Complete a table with the following headings:

Matrix	Purpose of the test	Expected output

[5]

Name and save your Jupyter Notebook as `Task4_<your name>_<NRIC number>.ipynb`.

Task 5

When you apply for local universities in Singapore (NUS/NTU/SMU/SUTD/SIT), you will need to compute a score based on your A level results known as the University Admission Score (UAS).
You will need to provide the following inputs to compute your UAS.

- Grade for H1 General Paper(GP)
- Grade for H1 Project Work(PW)
- Grade for 3 H2 Subjects
- Grade for 1 H1 Subject (If you take 4 H2 subjects, the lowest H2 subject grade will be used as a H1 grade)

The following table shows the corresponding points for the grades you obtained for H1 and H2 Subjects in the Cambridge A level examination.

Grade	H2 Equivalent	H1 Equivalent
A	20	10
B	17.5	8.75
C	15	7.5
D	12.5	6.25
E	10	5
S	5	2.5
U	0	0

A perfect score of 90 points is obtained when you obtained A for H1 GP, H1 PW and 4 H2 subjects or 3 H2 and 1 H1 subjects:

$$10(\text{GP}) + 10(\text{PW}) + 60(3\text{H2}) + 10(1\text{H1}) = 90.$$

You can assume that all the A level candidates took a minimum of 3 H2 subjects and 1 H1 subject and the compulsory H1 GP and H1 PW subjects.
The file provided is `IGP.CSV`.

Task 5.1

Create a Web Application using Python/Flask code for the users to enter their A level grades from a web browser and display the computed UAS.

Save your program as `Task5_1_<your name>_<NRIC number>.py` with any additional files/sub-folders as needed in a folder named `Task5_<your name>_<NRIC number>`. [6]

Task 5.2

Instead of using a web browser to enter the A level grades. A terminal-based client program is written to use socket-based communication to send the user input A level grades to a server program to compute the UAS score.

Write Python code to implement an iterative server that computes and return UAS score to be displayed on the client-program. You may re-use the code in **Task 5.1**.

The Python code for the client program is given below:

```
gp_grade = input("GP Grade:")
pw_grade = input("PW Grade:")
H2_1_grade = input("1st H2 subject grade:")
H2_2_grade = input("2nd H2 subject grade:")
H2_3_grade = input("3rd H2 subject grade:")
H1_grade = input("H1 Grade/Lowest H2 subject grade:")
s = socket.socket()
s.connect(("127.0.0.10",9999))
s.sendall( f"{gp_grade},{pw_grade},{H2_1_grade},{H2_2_grade},{H2_3_grade},{H1_grade}".encode() +
b"\n")
UAS = s.recv(1024)
print(f"UAS:{UAS.decode()}")
s.close()
```

Save your program as `Task5_2_<your name>_<NRIC number>.py`. [3]

Task 5.3

The file `IGP.CSV` contains a partial list of the universities' courses and their minimum UAS points (cut-off points). The cut-off points in the file do not include the GP and PW subjects.

Write Python code to load the contents of the file into a database table. The database name should be named `uni.db` and the table name should be UAS.

Save your program as `Task5_3_<your name>_<NRIC number>.py` in the same folder that you have created in **Task 5.1**. [3]

Task 5.4

Modify your Python code in part **Task 5.1** so that a list of the courses that meet the minimum cut-off points based on your computed UAS is displayed. You should use the data from the database table that you created in **Task 5.3**. Note that the cut-off points provided in the file `IGP.CSV` only use 3 H2 and 1 H1 subjects for computation.
An example of the web display is as follows:

```
UAS Score (GP,PW,3H2,1H1):  82.5

Available Courses:
```

NUS	Engineering
NUS	Architecture
NUS	Business Administration
NTU	Engineering
NTU	Computer Engineering
NTU	Aerospace
NTU	Physics
NTU	Sports Science
SMU	Computer Science
SMU	Economics

Save your program as `Task5_4_<your name>_<NRIC number>.py` in the same folder that you have created in **Task 5.1**. [4]