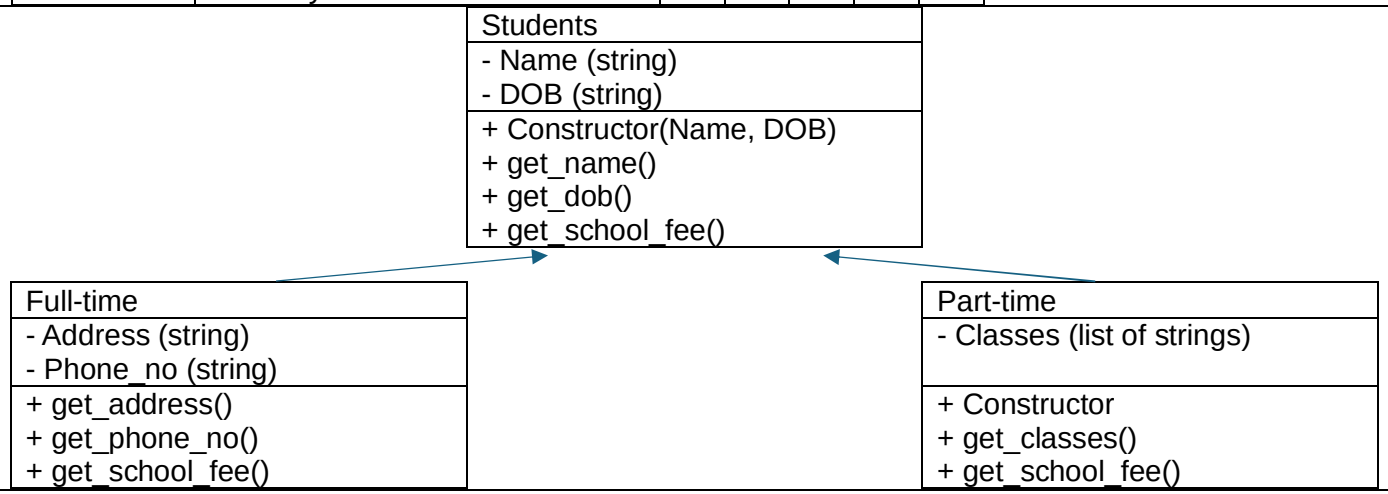
1(i)

Conditions	Run 2.4 km in at most 12 min	T	T	T	T	F	F	F	F
	Do at least 50 push ups in 1 min	T	T	F	F	T	T	F	F
	Do at least 40 sit ups in 1 min	T	F	T	F	T	F	T	F
Actions	Gold member	X							
	Silver member		X	X					
	Ordinary member				X	X	X	X	X

1(ii)

Conditions	Run 2.4 km in at most 12 min	T	T	T	T	F
	Do at least 50 push ups in 1 min	T	T	F	F	-
	Do at least 40 sit ups in 1 min	T	F	T	F	-
Actions	Gold member	X				
	Silver member		X	X		
	Ordinary member				X	X

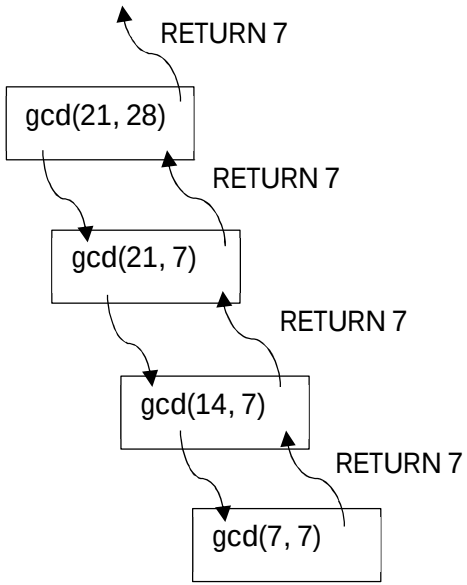
2(a)

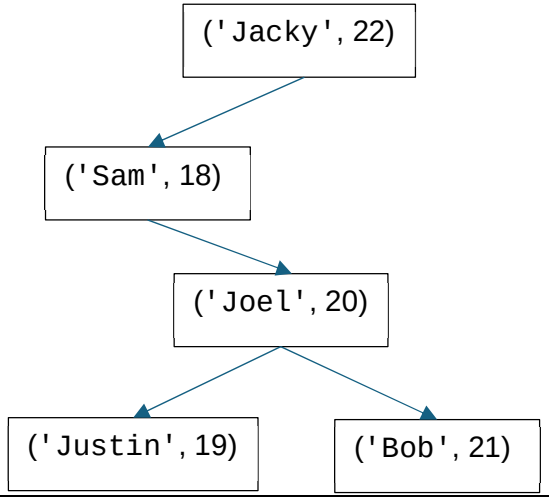


2(b) Full-time and part-time classes inherit attributes (Name and DOB) and methods (accessor for Name and DOB) from Students parent class

2(c) The calculation of school fees is different for full-time and part-time students, but the get_school_fee() method can be used for both. This streamlines code that uses the school fee for other purposes.

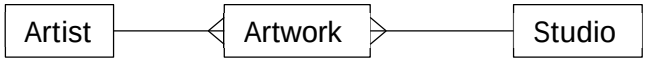
2(d)(i) Data should be backed up as it is updated so that a copy of the data exists in case the main database is damaged or deleted by accident or due to malicious action. The data can then be recovered from the backup copy.

2(d)(ii)	Data that is no longer needed on a day-to-day basis (e.g. students who have already graduated) should be archived. It can still be looked up if necessary, but the data does not clog up space in the current database.	
3(a)	21	
3(b)	06: IF m MOD i = 0 AND n MOD i = 0	
3(c)	The function has a base case (line 02) and for other cases, it calls itself with smaller values (lines 06 and 08). The function would thus keep calling itself until the base case is reached.	
3(d)		
3(e)	Since *m* > *n*, when the function calls itself, *m* will be replaced by *m* – *n*, which is larger than *m* and is still positive. As a result *m* will keep getting larger with each function call. Hence, the function will end up calling itself infinitely many times (until the computer runs out of memory or the program terminates it)	
3(f)	00 FUNCTION gcd(m, n : INTEGERS) RETURNS INTEGER 01 02 WHILE m != n 03 IF m > n THEN 04 m ← m – n 05 ELSE 06 n ← n – m 07 ENDIF 08 ENDWHILE	

	09 10 RETURN m 11 ENDFUNCTION	
4(i)	[1] [2] [3] ('Sam', 18) [4] ('Joel', 20) [5] ('Jacky', 22) [6] ('Bob', 21) [7] ('Justin', 19) [8] [9] [10]	
4(ii)	Hashing 'Bob' gives a value of 3. Go to index 3 at the hash table and check the name (first item of the tuple). If it is 'Bob', we return the age (second item of the tuple). If it is not 'Bob', or the entry is empty, we perform a linear search through the hash table starting form index 3 and looping around to the beginning of the table again if necessary. If we find a tuple whose name is 'Bob', we return the age. (In this case, it is at index 6) If we come back to index 3 without having found such a tuple, then 'Bob' does not exist in the hash table.	
4(iii)		
4(iv)	Justin, Bob, Joel, Sam, Jacky	

4(v)	Since the root has no right child, the oldest person is the root. The second oldest person is the oldest person in the left subtree. Starting from the left child of the root, keep going right until you reach a node with no right child (Bob). This is the second oldest person in the entire tree. (If the root has a right child there are two cases. Firstly, keep going right. The oldest person is the rightmost node. If this node does not have a left child, then the second oldest person is the parent of the rightmost node. If this node has a left child, then start from the left child and keep going right until you reach a node with no right child.)	
5(a)	**(A)**: `Arr[j] > Arr[j+1]` **(B)**: `Arr[j]` **(C)**: `Temp`	
5(b)	First way: Line 6: Iterate `j` from `1` to `N-i` Second way: Introduce flag variable between lines 5 and 6 that changes value if a swap in lines 8-10 happened. If the entire loop for `j` in line 6 runs without a swap then the list is already sorted. We can then break the loop for `i` and end the function there.	
5(c)	$O(n^2)$	
5(d)	**(D)**: `i <= N1 AND j <= N2` **(E)**: `Arr1[i] < Arr2[j]` **(F)**: `i <= N1` (alternatively, `j > N2`)	

5(e)	`FUNCTION Mergesort(Arr : ARRAY OF INTEGER) RETURNS ARRAY OF INTEGER`
	`    DECLARE N : INTEGER`
	`    DECLARE Result : ARRAY OF INTEGER`
	`    N ← LENGTH(Arr)`
	`    IF N <= 1 THEN`
	`        Result ← Arr`
	`    ELSE`
	`        DECLARE Mid : INTEGER`
	`        DECLARE Left, Right : ARRAYS OF INTEGER`
	`        Mid ← N DIV 2`
	`        Left ← Mergesort(Arr[1:Mid])`
	`        Right ← Mergesort(Arr[Mid+1:N])`
	`        Result ← Combine(Left, Right)`	

	ENDIF RETURN Result ENDFUNCTION	
6(a)	Data validation: Ensuring that data fits a particular format suitable for the algorithm Data verification: Ensuring that the data inputted / transmitted is the same as what was intended	
6(b)	Data validation can be carried out via length check (8 characters) and format check (contains required characters)	
6(c)	Data verification can be carried out by asking user to enter password twice, and comparing to make sure both entered password strings are identical	
6(d)	The data is encrypted using the server's public key. It can only be decrypted and read using the server's private key, which is unknown to a third party.	
6(e)	The three-way handshake takes place **before** the data transfer to ensure that the connection is reliable. (1) The user first sends a synchronization packet to the server to check that the server is ready to receive. (2) The server sends an acknowledgement back to the user and sends its own synchronization packet to the user. (3) The user acknowledges the server's synchronization packet. After this, the actual data packets are transmitted.	
6(f)	Since a hash is irreversible, it is impossible to recover the original password from the hashed form. This ensures that even if the server's database is compromised, it is not possible to recover the list of passwords. When a user logs in, the password entered by the user is hashed and this is compared to the hashed password stored under the user's name in the database. If they do not match, then the user has logged in with an incorrect password and will be prompted to retry.	
6(g)	The server can use 2-factor authentication to ensure that the user is who he claims to be.	
6(h)	The message is hashed and then encrypted with the sender's (server's) private key to generate the digital signature.	
6(i)	The user hashes the message. The user also decrypts the digital signature using the sender's public key. If the decrypted signature matches the hashed message, then the message is authentic. If they do not match, then the message has been edited.	
7(a)	There exist transitive dependencies in the table (e.g. studio address depends on studio name which depends on artwork title)	
7(b)	Artworks(Title, Medium, Price, ArtistName, StudioName) Artists(Name, Contact) Studios(Name, Address)	
7(c)		

7(d)	Remove redundancies in the data, so that when data is updated, only one entry needs to be changed. It also lowers the risk of errors and inconsistencies in the data.	
7(e)	SELECT ArtistName FROM Artworks WHERE StudioName = 'La Galeria' AND Price <= 3500	
7(f)	Personal data such as contact details should be stored with reasonable security. Use of artists' data should be carried out with consent of the artists. Personal data should be deleted if no longer in use, e.g. artist is no longer working with the company Etc.	