<table>
<tr><th>Question</th><th>Solution</th><th>Marks Allocation</th></tr>
<tr><td>1(a)</td><td>String data type.
Patient number has always got to be 4 digits in length, where some patients may have patient numbers with leading zeros eg. 0031.</td><td>[1]
[1]</td></tr>
<tr><td>1(b)</td><td>A collision happens when **a patient record is hashed by the hashing algorithm** to a **memory location that is already full**.

OR

A collision happens when **a patient record is hashed by the hashing algorithm** to a **memory location that already stores two patient records** stored within.</td><td>[1]
[1]
SR:
-1m if student gets answer correct but suggests in any ways the memory location stores one record and it is full.
-1m if context is not given in response.</td></tr>
<tr><td>1(c)</td><td><table>
<tr><td>…</td><td></td><td></td></tr>
<tr><td>212</td><td>1425 (1m)</td><td>3423 (collision) (1m)</td></tr>
<tr><td>(1m) 211</td><td>1422</td><td>2422</td></tr>
<tr><td>210</td><td></td><td></td></tr>
<tr><td>……</td><td></td><td></td></tr>
<tr><td>Memory location</td><td colspan="2">Memory</td></tr>
</table></td><td>1m : location 211 both correct
1m: location 212 stores 1425
1m: 3423 stored in location 212 as a result of collision</td></tr>
<tr><td>1(d)</td><td>- The data of the patient record that is to be **deleted will be marked by a tombstone value** to indicate that it has been deleted.
- The patient record deleted is **intentionally left in position within the memory location for the purpose of allowing future search** to be performed</td><td>[1]

[1]</td></tr>
<tr><td>1(e)</td><td>- **Apply the rightmost three digits** of the patient number to the **hash algorithm** to **obtain the hashed memory location** to store the patient record.
- Go to the hashed memory location, if there is an empty item in the memory location, insert patient record to the empty item of the memory location, and break out. Else
- Repeat
-     Check if memory location is full
-         Check if first item of memory location is tombstone marked.
            If yes, overwrite new patient record in to the first item in memory location
            Break
        Check if second item of memory location is tombstone marked
            If yes, store new patient record in to the second item in memory location
            Break
        If both items in memory location are occupied, move to</td><td>**[1]** mention of **hashing right most 3 digits** of patient number to get the hashed memory location
[1] Mention of **going to the memory location to check if there are any empty slots**. If there is store record in the empty slot
[1] if memory location is</td></tr>
</table>

<table>
<tr>
<td></td>
<td>the following memory location
• Until patient record gets stored.</td>
<td><b>entirely full, check if any of the items is tombstone marked</b> and store record over the existing deleted record <b>[1] Go to the next memory location and repeat the steps above</b></td>
</tr>
<tr>
<td>1(f)</td>
<td>
<b>Patient A:</b>
Patient Number = 1246, National Identity Number = S7654301Z

H(1246) = 123

<table>
<tr><td>Pos</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td></tr>
<tr><td>ID</td><td>1</td><td>2</td><td>3</td><td>4</td><td>3</td><td>0</td><td>1</td><td>C</td></tr>
<tr><td>Product</td><td>8</td><td>14</td><td>18</td><td>20</td><td>12</td><td>0</td><td>2</td><td>C</td></tr>
</table>

Product sum = 8 + 14 + 19 + 20 + 12 + 0 + 2 = 74
74 mod 11 = 8
C = (11 – 8) mod 11 = 3

Therefore, Patient A new Patient ID is 12343013

<b>Patient B:</b>
Patient Number = 5323, National Identity Number = S8563222B.
H(323) = 161

<table>
<tr><td>Pos</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td></tr>
<tr><td>ID</td><td>1</td><td>6</td><td>1</td><td>3</td><td>2</td><td>2</td><td>2</td><td>C</td></tr>
<tr><td>Product</td><td>8</td><td>42</td><td>6</td><td>15</td><td>8</td><td>6</td><td>4</td><td>C</td></tr>
</table>

Product sum = 8 + 42 + 6 + 15 + 8 + 6 + 4 = 89
89 mod 11 = 1
C = (11 – 1) mod 11 = 10

Therefore, Patient B new Patient ID is 1313222X (where X = 10)
</td>
<td>[1] - correct product sum

[1] - correct answer

[1] - correct product sum

[1] - correct answer

[1] - state where X = 10</td>
</tr>
</table>

<table>
<tr>
<td>1(g)</td>
<td>New Patient ID of Patient C = 21167892

<table>
<tr><td>Pos</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td></tr>
<tr><td>ID</td><td>2</td><td>1</td><td>1</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2</td></tr>
<tr><td>Product</td><td>16</td><td>7</td><td>6</td><td>30</td><td>28</td><td>24</td><td>18</td><td>2</td></tr>
</table>

Product sum = 16 + 7 + 6 + 30 + 28 + 24 + 18 + 2 = 131
Since 131 mod 11 <> 0
Therefore, the new patient ID of Patient C is invalid.</td>
<td>[1] - workings in attempt to obtain product sum
[1] use 131 mod 11 <> 0 as reasoning leading to invalid conclusion.</td>
</tr>
<tr>
<td>2(a)</td>
<td>A Distributed Denial-of-Service (DoS) attack is carried out by one or a group cybercriminals that controls a network of Internet-connected machines known as botnet.

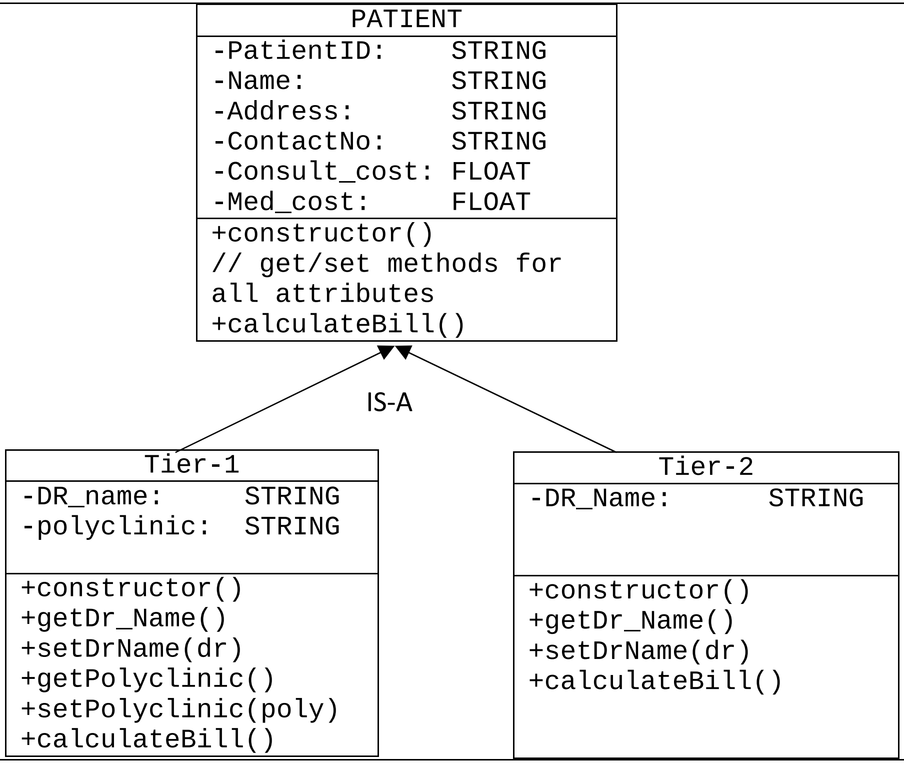
The attacker then sends remote instructions to each bot in the botnet to send voluminous bogus requests to the server that stored the medical records to overwhelm the resource and capacity of the server.

When the server direct its resources to handle the bogus requests and gets overwhelmed, it results in depriving the genuine users of the medical system from accessing the information stored in the system.</td>
<td>[1]: mention of <b>botnet or network of devices</b> controlled by cybercriminals.

[1] <b>bots send repeated bogus requests to the server</b> to overwhelm server.

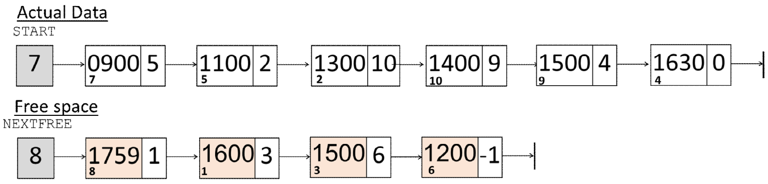
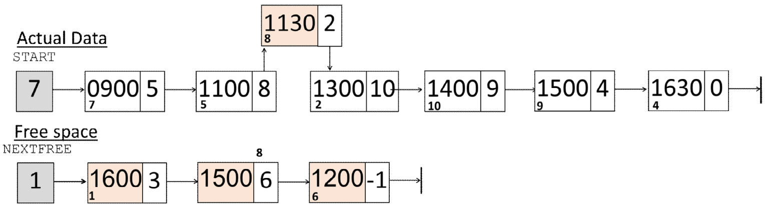
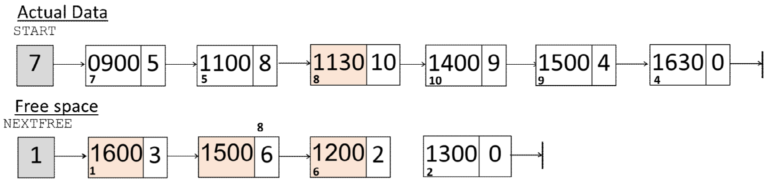
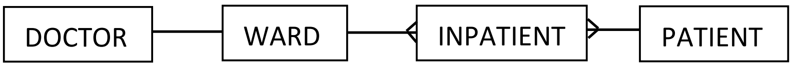
[1] with a purpose of <b>depriving genuine users from accessing the server</b>.</td>
</tr>
<tr>
<td>2(b)</td>
<td>Any one of the methods below with elaboration:
1. Man – in – the middle attack – Intercepting communication between two parties to obtain personal information.
2. Malicious Software – device could be infected with malware, such as keyloggers and packet sniffers, to monitor and steal private data.
3. SQL Injection – hacker inserts their own code into a website to breach its security measures and access protected data. Once inside, they can control the website's database and hijack user information.</td>
<td>[1] method
[1] explanation of how the method is used to steal data.</td>
</tr>
<tr>
<td>2(ci)</td>
<td>• <b>The sender applies the message it wishes to send to a hash algorithm in order to obtain a digest of the message (also known as message hash).</b>
• <b>Sender then encrypts digest using his/her private keys. The encrypted digest is also known a small digital representation of the message that is unique to the message.</b>
• <b>Sender then attaches the digital signature together with the original message it wishes to send and have it sent to the receiver.</b>
(The reason for encrypting the hash instead of the entire message is because a hash function can convert an arbitrary input into a fixed-length value, which is usually much shorter. This saves time, as hashing is much faster than signing).</td>
<td>[1] mark for each point.

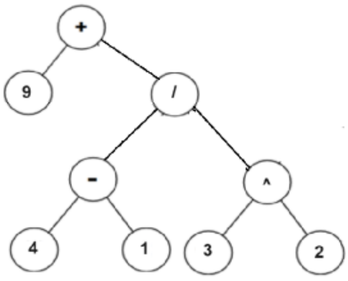
[Total 3m]</td>
</tr>
</table>

2(cii)	• Since the message is encrypted using sender’s private key, **the only way to decrypt the message is by use of sender’s public key**. Recipient will use the public key of the sender to decrypt the digest. **If the recipient can successfully decrypt the message it means the sender is authenticated as the sender’s public key is the only key to decrypt the encrypted message**, given that the sender keeps its private key only to itself. • **Independently, the recipient would also put the message received to the same hash algorithm to obtain a digest of its own.** • Recipient will then compare the decrypted digest it receives to the digest it generates. • **If the two digest are exactly similar it means that the received message is authentic (ie. from the sender) as the message received has not been changed since the sender sends it out**.	[1] sender’s public key can only be used to decrypt [1] reasons to explain
2(d)	Backup file: a copy of data that can be restored in case of data loss, corruption, or a system failure. Archive file: To store data that is no longer actively used but needs to be retained for long-term reference, compliance, or historical purposes	[1] [1]
2(e)	Data validation is the process that ensures that the data entered is accurate, meaningful, sensible and meets certain predefined criteria. Data verification is the process that ensures that the data entered is consistent and matches exactly to its original source.	[1] [1]
2(f)	Data validation: Presence check, type check, length check on any relevant specific input of data related to patient. Data verification: Double entry or getting another staff to verify the patient record entered by the data entry clerk	
3(a)	 PATIENT -PatientID: STRING -Name: STRING -Address: STRING -ContactNo: STRING -Consult_cost: FLOAT -Med_cost: FLOAT +constructor() // get/set methods for all attributes +calculateBill() IS-A Tier-1 -DR_name: STRING -polyclinic: STRING +constructor() +getDr_Name() +setDrName(dr) +getPolyclinic() +setPolyclinic(poly) +calculateBill() Tier-2 -DR_Name: STRING +constructor() +getDr_Name() +setDrName(dr) +calculateBill()	

<table>
<tr><td></td><td></td><td></td></tr>
<tr><td>3(bi)</td><td>Public, private, protected</td><td></td></tr>
<tr><td>3(bii)</td><td>Encapsulation promotes information hiding:
The internal implementation details of a class eg. attributes of a patient, are made private and inaccessible to the outside world, exposing only necessary parts eg. attributes get and set accessor methods through public methods to the outside world.
This protects the internal state of the object from unintended or harmful modifications and reduces the risk of errors.
OR
Encapsulation promotes modularity:
Encapsulation allows the internal implementation of any of the classes to change without affecting other parts of the program, as long as the public methods exposed to other parts of the code remains consistent.</td><td></td></tr>
<tr><td>3(ci)</td><td>
<table>
<tr><th></th><th colspan="6">Outcomes</th><th></th><th></th></tr>
<tr><td><b>Conditions</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td><td><b>4</b></td><td><b>5</b></td><td><b>6</b></td><td><b>7</b></td><td><b>8</b></td></tr>
<tr><td>Rerferred by Polyclinic</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>Annual Value of HDB < $25000</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>Y</td><td>Y</td><td>N</td><td>N</td></tr>
<tr><td>Monthly income per person <= $1200</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td></tr>
<tr><td><b>Actions</b></td><td colspan="6"></td><td></td><td></td></tr>
<tr><td>Award full subsidy to patient</td><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Award partial susidy to patient</td><td></td><td>X</td><td>X</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>Do not award subsidy to patient</td><td></td><td></td><td></td><td>X</td><td></td><td>X</td><td>X</td><td>X</td></tr>
</table>
</td><td></td></tr>
<tr><td>3(cii)</td><td>
<table>
<tr><th></th><th colspan="7">Outcomes</th></tr>
<tr><td><b>Conditions</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td><td><b>4&8</b></td><td><b>5</b></td><td><b>6</b></td><td><b>7</b></td></tr>
<tr><td>Rerferred by Polyclinic</td><td>Y</td><td>Y</td><td>Y</td><td>-</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>Annual Value of HDB < $25000</td><td>Y</td><td>Y</td><td>N</td><td>N</td><td>Y</td><td>Y</td><td>N</td></tr>
<tr><td>Monthly income per person <= $1200</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td><td>N</td><td>Y</td></tr>
<tr><td><b>Actions</b></td><td colspan="6"></td><td></td></tr>
<tr><td>Award full subsidy to patient</td><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Award partial susidy to patient</td><td></td><td>X</td><td>X</td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>Do not award subsidy to patient</td><td></td><td></td><td></td><td>X</td><td></td><td>X</td><td>X</td></tr>
</table>
</td><td></td></tr>
<tr><td>4(a)</td><td>The elements of a linked list are not stored in contiguous memory locations in the computer’s memory.

In the case of a linked list, the location of the current node can only be determined at its preceding node. Hence, the only way to find a node is to follow the chain of pointers.</td><td></td></tr>
<tr><td>4(b)</td><td>Free space list memory locations: 8, 1, 3, 6
A = 10
B = 0
C = 2
D = 5
E = 4
F = 9
G = 7</td><td></td></tr>
</table>

4(c)	 Actual Data START 7 0900 5 7 1100 2 5 1300 10 2 1400 9 10 1500 4 9 1630 0 4 Free space NEXTFREE 8 1759 1 8 1600 3 1 1500 6 3 1200 -1 6	
4(di)	 1130 2 8 Actual Data START 7 0900 5 7 1100 8 5 1300 10 2 1400 9 10 1500 4 9 1630 0 4 Free space NEXTFREE 8 1 1600 3 1 1500 6 1200 -1 6	
4(dii)	 Actual Data START 7 0900 5 7 1100 8 5 1130 10 8 1400 9 10 1500 4 9 1630 0 4 Free space NEXTFREE 8 1 1600 3 1 1500 6 1200 2 6 1300 0 2	
4(e)	Array elements are fixed in size Array can only store elements of the same data type Array elements need to be stored in contiguous memory locations. Hence, any insertion or deletion operation will likely require much resources to move the existing elements of the array from one memory location to another.	
5(a)	Normalisation is the process of organising data to - Reduce data redundancy by eliminating duplicate data in order to utilise storage space and reduces the potential for data inconsistencies. - Improve data integrity by ensuring that the data remains consistent and accurate across all operations.	1m either point or any other valid points
5(b)	Data redundancy Increased cost: - Takes up more storage space, which increases costs, especially in large-scale databases like a healthcare data management system. - This can also affect database performance due to the larger volume of patient, doctor, ward and inpatient data that needs to be managed Update anomalies: - When hospital data is stored redundantly, any update to that data must be applied in all the places where it exists. If this is not done consistently, it leads to **update anomalies**, where different copies of the same data have different values.	1m for stating either point 1m for reasonable explanation to the point mentioned. SR: 0m for responses like “more efficient”, “faster execution”, etc
5(c)		1m – DOCTOR, PATIENT, WARD entities seen

		1m – INPATIENT entity between WARD and PATIENT 1m – Correct relationships.
5(d)	PATIENT(**patientID**, name, sex, address, contactNo, med_history) DOCTOR(**staffID**, name, sex, department, appointment) WARD(**wardID**, speciality, capacity, bedsOccupied, drIC*) INPATIENT(**patientID***, **wardID***, **admissionDate**, dischargeDate) *denotes foreign key drID of WARD table is a foreign key that references the primary key staffID of DOCTOR table patientID and wardID of INPATIENT table are foreign keys that reference the primary keys patientID of PATIENT table and wardID of WARD table respectively.	1m – correct tables for PATIENT, WARD and DOCTOR with PK correctly identified 1m – for INPATIENT table with correct composite key identified 1m – for correct identification of all FKs. 1m – legend to indicate and describe of FK stated.
6(a)		1m – correct root node. 2m – correct BT SR: -1 m for each mistake made up to 2 marks
6(b)	Postorder traversal: 941-32^/+ Preorder traversal: +9/-41^32	1m – correct postorder 1m - correct preorder
6(c)	In postfix notation, the order of operations is performed by the position of the operators and operands, ie. left to right. There is no need for parentheses to indicate precedence or grouping, which simplifies both the expression and the evaluation process.	1m 1m
6(d)	To evaluate the postfix expression 941-32^/+ using a stack, we'll follow these steps:	1m – describes and shows

	1. **Push operands onto the stack** when encountered. 2. **Pop operands off the stack**, apply the operator, and **push the result back onto the stack** when an operator is encountered. 3. Continue this process until the entire expression is evaluated. • **Initial Expression**: 941-32^/+ • **Stack**: Empty 1. **9**: Push 9 onto the stack. o **Stack**: [9] 2. **4**: Push 4 onto the stack. o **Stack**: [9, 4] 3. **1**: Push 1 onto the stack. o **Stack**: [9, 4, 1] 4. **-**: Pop the top two elements (1 and 4) from the stack, subtract them (4 - 1 = 3), and push the result (3) back onto the stack. o **Stack**: [9, 3] 5. **3**: Push 3 onto the stack. o **Stack**: [9, 3, 3] 6. **2**: Push 2 onto the stack. o **Stack**: [9, 3, 3, 2] 7. **^**: Pop the top two elements (2 and 3) from the stack, apply the exponentiation operation (3 ^ 2 = 9), and push the result (9) back onto the stack. o **Stack**: [9, 3, 9] 8. **/**: Pop the top two elements (9 and 3) from the stack, divide them (3 / 9 = 0.3333...), and push the result (0.3333...) back onto the stack. o **Stack**: [9, 0.3333...] 9. **+**: Pop the top two elements (0.3333... and 9) from the stack, add them (9 + 0.3333... = 9.3333...), and push the result (9.3333...) back onto the stack. o **Stack**: [9.3333...] 10. Pop 9.3333…from stack and return as answer	pushing of item into the stack when it is an operand 1m – when an operator is encountered, pop 2 items from the stack and apply the operator on both items in the correct order, and push the result back into the stack. 1m – if all items of the expression has been visited, pop the final value out from stack and return as the result.
7(a)	Sequence, selection, iteration	1m Note: 0m if either word is incorrect
7(bi)	Use of white space and line breaks	1m for each point
7(bii)	Line 2	1m
7(c)	A **recursively defined procedure** is a function or algorithm that solves a problem by calling itself with a smaller or simpler version of the original problem. Recursive procedures typically have two main components: 1. **Base Case (Termination Condition)**: This is the condition that stops the recursion. It defines the simplest possible scenario, which can be solved directly without further recursion. When the base case is reached, the recursion ends. 2. **Recursive Case**: This is where the procedure calls itself with a modified input that gradually reduces the problem's complexity. The recursive calls continue until the base case is met.	1m for both points with reasonable elaborations to the context.
7(d)	Line 6	1m

<table>
<tr><td>7(e)</td><td>A <b>stack</b> is used to execute recursive routines because recursion inherently requires a last-in, first-out (LIFO) approach to manage the sequence of function calls.

It allows the system to keep track of active function calls, local variables, and return addresses during recursive execution.

Each time a function calls itself (recursively), a new stack frame is pushed onto the stack. This stack frame holds the function’s local state, enabling the program to return to the correct point after the recursive call finishes. When the function completes, its stack frame is popped off the stack, and control returns to the previous function call</td><td>1m for each point with reasonable elaborations to the context.</td></tr>
<tr><td>7(f)</td><td>
<table>
<tr><th>Line</th><th>Call Procedure</th><th>x=0 or x=1?</th><th>x DIV 2</th><th>x MOD 2</th><th>x</th><th>Output(x)</th></tr>
<tr><td>1</td><td>A(43)</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>FALSE</td><td></td><td></td><td></td><td></td></tr>
<tr><td>6/1</td><td>A(21)</td><td></td><td>21</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>FALSE</td><td></td><td></td><td></td><td></td></tr>
<tr><td>6/1</td><td>A(10)</td><td></td><td>10</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>FALSE</td><td></td><td></td><td></td><td></td></tr>
<tr><td>6/1</td><td>A(5)</td><td></td><td>5</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>FALSE</td><td></td><td></td><td></td><td></td></tr>
<tr><td>6/1</td><td>A(2)</td><td></td><td>2</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>FALSE</td><td></td><td></td><td></td><td></td></tr>
<tr><td>6/1</td><td>A(1)</td><td></td><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>TRUE</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr>
<tr><td>7</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td></tr>
<tr><td>8</td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
<tr><td>7</td><td></td><td></td><td></td><td>1</td><td>1</td><td></td></tr>
<tr><td>8</td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr>
<tr><td>7</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td></tr>
<tr><td>8</td><td></td><td></td><td></td><td></td><td></td><td>0</td></tr>
<tr><td>7</td><td></td><td></td><td></td><td>1</td><td>1</td><td></td></tr>
<tr><td>8</td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr>
<tr><td>7</td><td></td><td></td><td></td><td>1</td><td>1</td><td></td></tr>
<tr><td>8</td><td></td><td></td><td></td><td></td><td></td><td>1</td></tr>
</table>
</td><td></td></tr>
<tr><td>7(g)</td><td>It is a function that takes in an input denary number and covert it into its binary equivalent.</td><td>1m</td></tr>
<tr><td>8(a)</td><td>
<table>
<tr><td>sorted</td><td colspan="6">unsorted</td></tr>
<tr><td>ar[1]</td><td>ar[2]</td><td>ar[3]</td><td>ar[4]</td><td>ar[5]</td><td>ar[6]</td><td>ar[7]</td></tr>
<tr><td>18</td><td>39</td><td>6</td><td>44</td><td>41</td><td>5</td><td>30</td></tr>
</table>

First item is taken to be sorted. Hence insertion sort starts with the second element.

<table>
<tr><td colspan="2">sorted</td><td colspan="5">unsorted</td></tr>
<tr><td>ar[1]</td><td>ar[2]</td><td>ar[3]</td><td>ar[4]</td><td>ar[5]</td><td>ar[6]</td><td>ar[7]</td></tr>
</table>
</td><td>1m – mention about first item is taken to be sorted at the end of the first pass.</td></tr>
</table>

18	39	6	44	41	5	30

Compare 39 with 18. Since 39 > 18 no change needed.

sorted			unsorted			
ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]	ar[7]
6	18	39	44	41	5	30

Compare 6 with 39. Since 6 < 39, swap 6 and 39
Compare 6 with 18. Since 6 < 18, swap 6 and 18.

sorted				unsorted		
ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]	ar[7]
6	18	39	44	41	5	30

Compare 44 with 39, Since 44 > 39, no change needed.

sorted					unsorted	
ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]	ar[7]
6	18	39	41	44	5	30

Compare 41 with 44. Since 41 < 44, swap 41 and 44
Compare 41 with 39. Since 41 > 39, no change needed.

sorted						unsorted
ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]	ar[7]
5	6	18	39	41	44	30

Compare 5 with 44. Since 5 < 44, swap 5 and 44
Compare 5 with 41. Since 5 < 41, swap 5 and 41
Compare 5 with 39. Since 5 < 39, swap 5 and 39
Compare 5 with 18. Since 5 < 18, swap 5 and 18
Compare 5 with 6. Since 5 < 6, swap 5 and 6

sorted						
ar[1]	ar[2]	ar[3]	ar[4]	ar[5]	ar[6]	ar[7]
5	6	18	30	39	41	44

Compare 30 with 44. Since 30 < 44, swap 30 and 44
Compare 30 with 41. Since 30 < 41, swap 30 and 41
Compare 30 with 39. Since 30 < 39, swap 30 and 39
Compare 30 with 18. Since 30 > 18, no change.

Insertion sort completes and terminates.

8(b)	O(n^2) time complexity. Happens when every item in the pre-sort data set is in a complete reverse order as opposed to the algorithm's sort order	1m – time complexity 1m – reason
8(c)	30, 6, 18	2m
8(d)	O($\log_2 n$)	1m