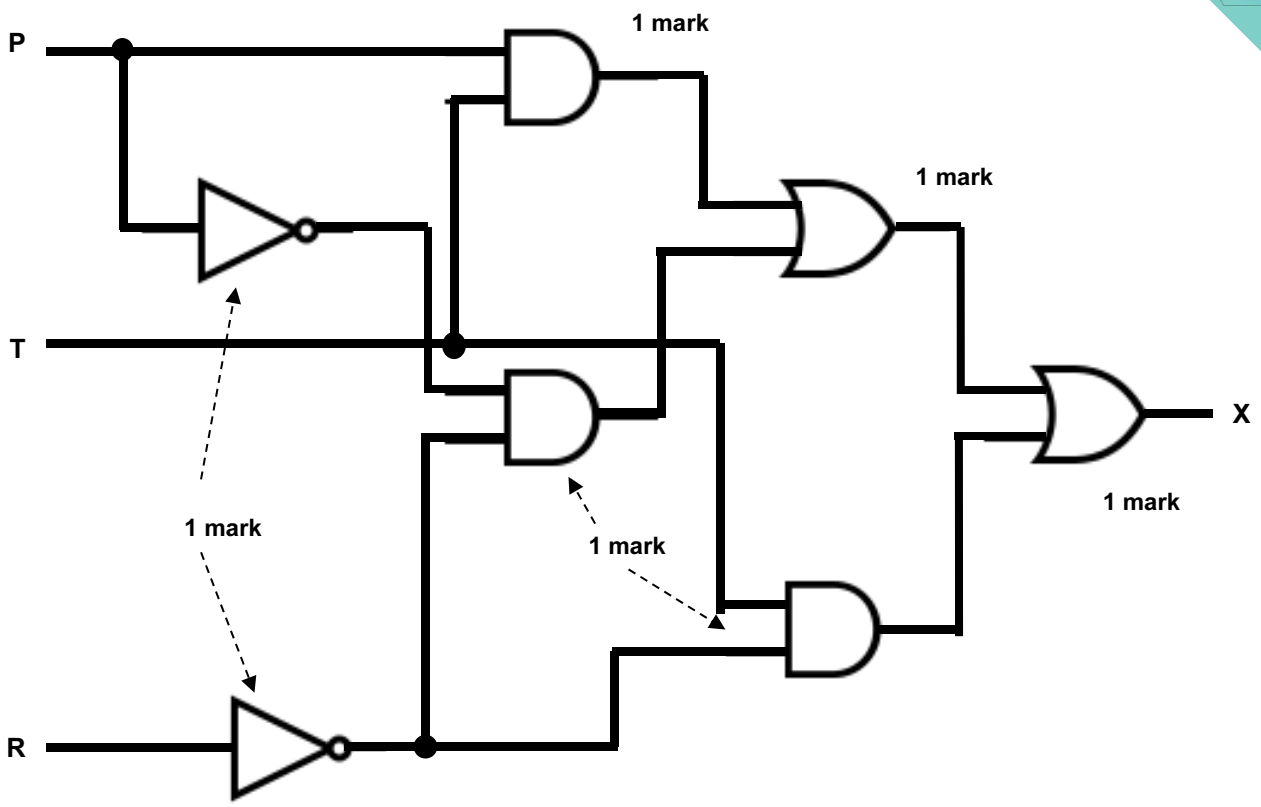
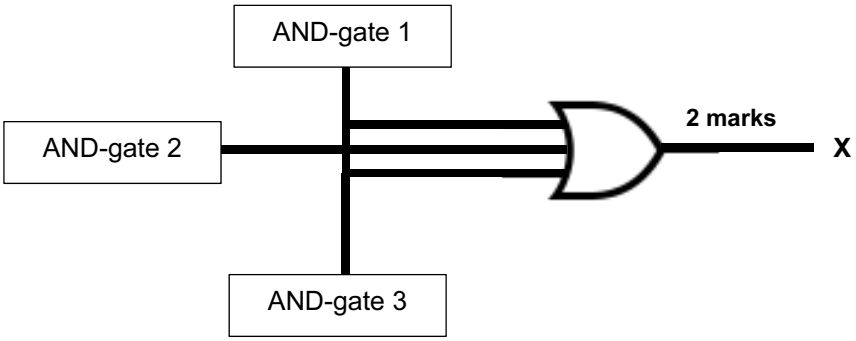
QUESTION 1.

7 (a) Since it is possible to simplify the original conditions, at least 3 possible answe
logic circuit.

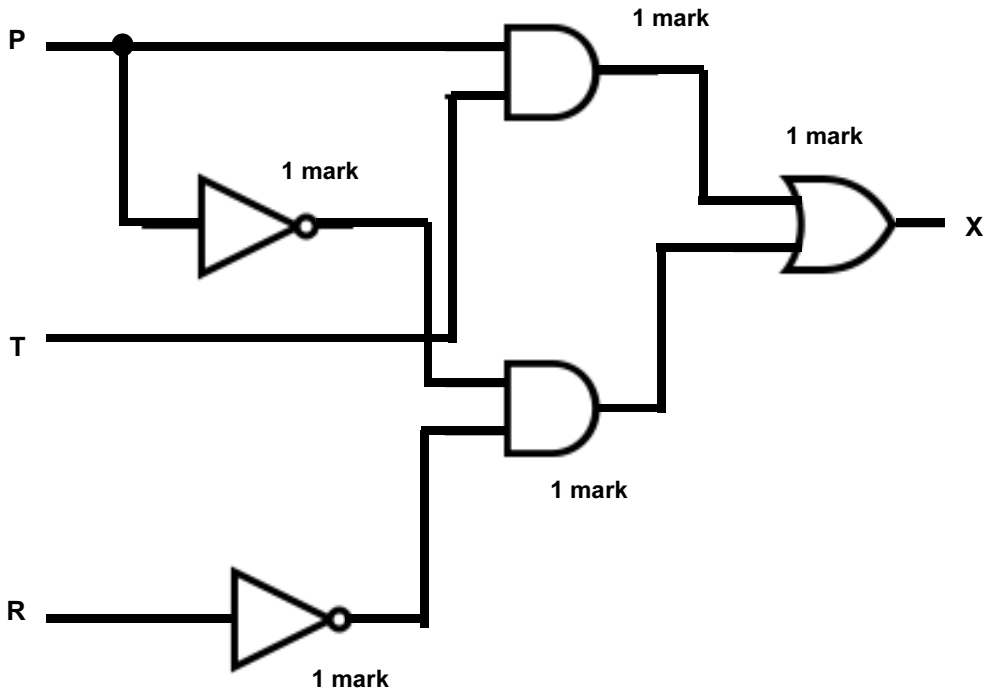


Note: input T has 2 cross overs that should not be connections

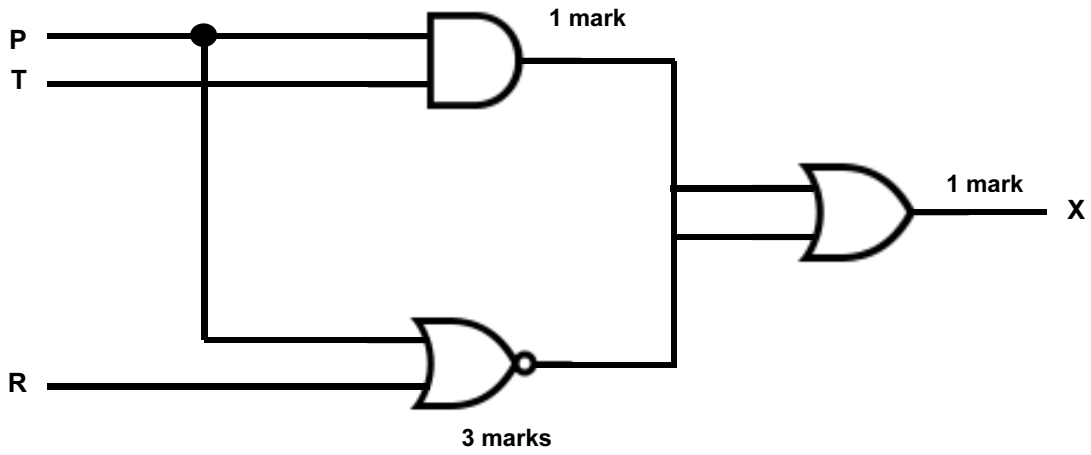
Note: it is possible to use a 3-input OR gate rather than the two 2-input OR gates on the top right:



Alternative solution 1:



Alternative solution 2:



[5]

Note: other solutions may be possible depending on how simplification of the original statement is done

(b)

P	T	R	Workspace	X	
0	0	0		1	} 1 mark
0	0	1		0	
0	1	0		1	} 1 mark
0	1	1		0	
1	0	0		0	} 1 mark
1	0	1		0	
1	1	0		1	} 1 mark
1	1	1		1	

QUESTION 2.

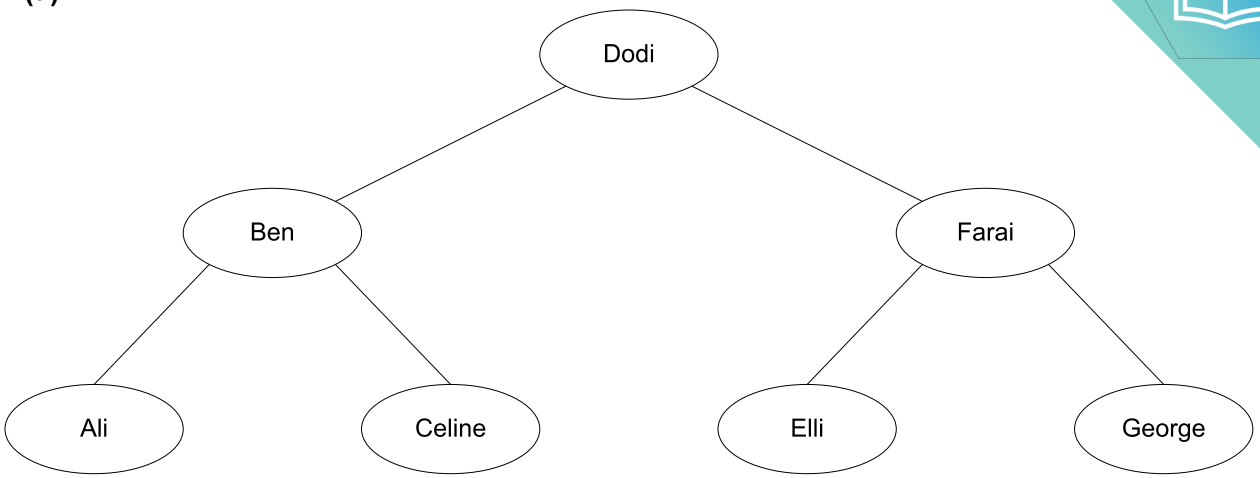
2

Activity	First pass or second pass
any symbolic address is replaced by an absolute address	**2**
any directives are acted upon	**1**
any symbolic address is added to the symbolic address table	**1**
data items are converted into their binary equivalent	**1**
forward references are resolved	**2**

[5]

QUESTION 1.

4 (a)



[4]

(b)

RootPointer: 1

FreePointer: 8

Tree

	Name	LeftPointer	RightPointer
[1]	Dodi	5	2
[2]	Farai	3	4
[3]	Elli	0	0
[4]	George	0	0
[5]	Ben	7	6
[6]	Celine	0	0
[7]	Ali	0	0
[8]		9	0
[9]		10	0
[10]		0	0

[7]

(c) (i)

```
01 PROCEDURE TraverseTree(BYVALUE Root : INTEGER)
02     IF Tree[Root].LeftPointer < > 0
03         THEN
04             TraverseTree(Tree[Root].LeftPointer)
05     ENDIF
06     OUTPUT Tree[Root].Name
07     IF Tree[Root].RightPointer < > 0
08         THEN
09             TraverseTree(Tree[Root].RightPointer)
10     ENDIF
11 ENDPROCEDURE
```

[5]

(ii) A procedure that calls itself // is defined in terms of itself
Line number: 04/09

[2]

QUESTION 2.

3

START:	LDR	#0	// initialise index register to zero	
	LDM	#0	// initialise COUNT to zero	[1]
	STO	COUNT		
LOOP1:	LDX	NAME	// load character from indexed address NAME	[1]
	OUT		// output character to screen	[1]
	INC	IX	// increment index register	[1]
	LDD	COUNT	// increment COUNT starts here	
	INC	ACC		[1]
	STO	COUNT		
	CMP	MAX	// is COUNT = MAX?	[1]
	JPN	LOOP1	// if FALSE, jump to LOOP1	[1]
REVERSE:	DEC	IX	// decrement index register	[1]
	LDM	#0	// set ACC to zero	[1]
	STO	COUNT	// store in COUNT	
LOOP2:	LDX	NAME	// load character from indexed address NAME	[1]
	OUT		// output character to screen	
	DEC	IX	// decrement index register	[1]
	LDD	COUNT	// increment COUNT starts here	
	INC	ACC	//	[1]
	STO	COUNT	//	
	CMP	MAX	// is COUNT = MAX?	[1]
	JPN	LOOP2	// if FALSE, jump to LOOP2	
	END		// end of program	[1]
COUNT:				
MAX:	4			
NAME:	B01000110		// ASCII code in binary for 'F'	
	B01010010		// ASCII code in binary for 'R'	
	B01000101		// ASCII code in binary for 'E'	
	B01000100		// ASCII code in binary for 'D'	

[Max 15]

QUESTION 3.

3 (a) (i) 1 mark per bullet to max 2: [2]

- `11011111`
- `AND`

(ii) 1 mark per bullet to max 2: [2]

- `00100000`
- `OR`

(b) 1 mark per line

START:	LDR	#0	// initialise index register to zero	
	LDX	WORD	// get first character of WORD	1
	AND	MASK1	// ensure it is in upper case using MASK1	1
	OUT		// output character to screen	
	INC	IX	// increment index register	1
	LDM	#1	// load 1 into ACC	1
	STO	COUNT	// store in COUNT	1
LOOP:	LDX	WORD	// load next character from indexed address WORD	1
	OR	MASK2	// make lower case using MASK2	1
	OUT		// output character to screen	
	LDD	COUNT	// increment COUNT	1
	INC	ACC	//	
	STO	COUNT	//	
	CMP	LENGTH	// is COUNT = LENGTH?	1
	JPN	LOOP	// if FALSE – jump to LOOP	1
	END		// end of program	1
COUNT:	0			
MASK1:	B11011111		// bit pattern for upper case	1
MASK2:	B00100000		// bit pattern for lower case	
LENGTH:	4			
WORD:	B01100110		//ASCII code in binary for 'f'	
	B01101000		//ASCII code in binary for 'r'	
	B01000101		//ASCII code in binary for 'E'	
	B01000100		//ASCII code in binary for 'D'	

[max 12]

QUESTION 4.

Question	Answer	

3	FUNCTION Find(BYVAL Name : STRING,
	BYVAL Start : INTEGER,
	BYVAL Finish : INTEGER) RETURNS INTEGER
	// base case
	IF **Finish < Start** 1
	THEN
	RETURN -1
	ELSE
	Middle ← **(Start + Finish) DIV 2** 1
	IF **NameList[Middle] = Name** 1
	THEN
	RETURN **Middle** 1
	ELSE // general case
	IF SearchItem > **NameList[Middle]** 1
	THEN
	Find(Name, Middle + 1, Finish) 1
	ELSE
	Find(Name, Start, Middle - 1) 1
	ENDIF
	ENDIF
	ENDIF
	ENDFUNCTION	

Question	Answer	Marks

QUESTION 5.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>• A pointer that doesn't point to another node/other data/address // indicates the end of the branch</td><td>1</td></tr>
<tr><td>2(b)</td><td>one mark per bullet
• node with 'Athens' linked to left pointer of Berlin (ignore null pointer)
• null pointers in left and right pointers of Athens</td><td>2</td></tr>
<tr><td>2(c)(i)</td><td>
RootPointer
0

FreePointer
7
1 mark
<table>
<tr><th></th><th>LeftPointer</th><th>Tree Data</th><th>RightPointer</th></tr>
<tr><td>[0]</td><td>2</td><td>Dublin</td><td>1</td></tr>
<tr><td>[1]</td><td>-1/∅</td><td>London</td><td>3</td></tr>
<tr><td>[2]</td><td>6</td><td>Berlin</td><td>5</td></tr>
<tr><td>[3]</td><td>4</td><td>Paris</td><td>-1/∅</td></tr>
<tr><td>[4]</td><td>-1/∅</td><td>Madrid</td><td>-1/∅</td></tr>
<tr><td>[5]</td><td>-1/∅</td><td>Copenhagen</td><td>-1/∅</td></tr>
<tr><td>[6]</td><td>-1/∅</td><td>Athens</td><td>-1/∅</td></tr>
<tr><td>[7]</td><td>8</td><td></td><td>-1/∅</td></tr>
<tr><td>[8]</td><td>9</td><td></td><td>-1/∅</td></tr>
<tr><td>[9]</td><td>-1/∅</td><td></td><td>-1/∅</td></tr>
</table>
</td><td>5</td></tr>
<tr><td>2(c)(ii)</td><td>• –1
• It is not the number for any node.</td><td>2</td></tr>
</table>

Question	Answer	Marks
2(d)(i)	TYPE Node **LeftPointer : INTEGER** **RightPointer : INTEGER** 1 **Data : STRING** ENDTYPE DECLARE Tree : ARRAY[0 : 9] **OF Node** 1 DECLARE FreePointer : INTEGER DECLARE RootPointer : INTEGER PROCEDURE CreateTree() DECLARE Index : INTEGER **RootPointer ← -1** 1 **FreePointer ← 0** 1 FOR Index ← 0 TO 9 // link nodes **Tree[Index].LeftPointer ← Index + 1** 1 **Tree[Index].RightPointer ← -1** 1 ENDFOR **Tree[9].LeftPointer ← -1** 1 ENDPROCEDURE	**7**

Question	Answer	Marks

2(d)(ii)	PROCEDURE AddToTree(ByVal NewDataItem : STRING)
	// if no free node report an error
	IF FreePointer **= -1** 1
	THEN
	ERROR("No free space left")
	ELSE // add new data item to first node in the free list
	NewNodePointer ← FreePointer
	Tree[NewNodePointer].Data ← NewDataItem 1
	// adjust free pointer
	FreePointer ← **Tree[FreePointer].LeftPointer** 1
	// clear left pointer
	Tree[NewNodePointer].LeftPointer ← **-1** 1
	// is tree currently empty ?
	IF **RootPointer = -1** 1
	THEN // make new node the root node
	RootPointer ← NewNodePointer 1
	ELSE // find position where new node is to be added
	Index ← RootPointer
	CALL FindInsertionPoint(NewDataItem, Index, Direction)
		8

Question	Answer	Marks
	IF Direction = "Left" THEN // add new node on left **Tree[Index].LeftPointer ← NewNodePointer** 1 ELSE // add new node on right **Tree[Index].RightPointer ← NewNodePointer** **1** ENDIF ENDIF ENDIF ENDPROCEDURE	
2(e)	**1 mark per bullet** • test for base case (null/-1) • recursive call for left pointer • output data • recursive call for right pointer • order, visit left, output, visit right IF Pointer <> NULL 1 THEN TraverseTree(Tree[Pointer].LeftPointer) 1 OUTPUT Tree[Pointer].Data 1 + 1 TraverseTree(Tree[Pointer].RightPointer) 1 ENDIF ENDPROCEDURE	**5**

QUESTION 6.

2(b)	1 mark for each completed section
	
	`FUNCTION FindElement(Item : INTEGER) RETURNS INTEGER`
	`  CurrentPointer ← RootPointer`
	`  WHILE CurrentPointer <> NullPointer`
	`  IF List[CurrentPointer].Data <> Item`
	`  THEN`
	`  CurrentPointer ← List[CurrentPointer].Pointer`
	`  ELSE`
	`  RETURN CurrentPointer`
	`  ENDIF`
	`  ENDWHILE`
	`  CurrentPointer ← NullPointer`
	`  RETURN CurrentPointer`
	`ENDFUNCTION`
		6

2(c)(i)	1 mark per bullet point to max 3 e.g. • A sequence of steps that change the state of the program • The steps are in the order they should be carried out • e.g. procedural programming/language • Groups code into self-contained blocks // split the program into modules • … which are subroutines // by example	**3**

Question	Answer	
2(c)(ii)	1 mark per bullet point to max 3 e.g. • Creates classes • …as a blueprint for an object // objects are instances of classes • …that have properties/attributes **and** methods • … that can be private to the class // properties can **only** be accessed by the class's methods // encapsulation • **Subclasses** can inherit from **superclasses** (child and parent) • A **subclass** can inherit the methods and properties from the **superclass** • A **subclass** can change the methods from the **superclass** // **subclass** can use polymorphism • Objects can interact with each other	

2(d)(i)	1 mark per bullet point
	
	• Method header and close (where appropriate)
	• …with `InputPlayerID` parameter
	• Initialise `Score` to 0
	• Initialise Category to `"Not Qualified"`
	• Initialise `PlayerID` to parameter
	
	PYTHON
	`def__init__(self, InputPlayerID):`
	`  self.__Score = 0`
	`  self.__Category = "Not Qualified"`
	`  self.__PlayerID = InputPlayerID`
	PASCAL
	`Constructor Player.Create(InputPlayerID);`
	`begin`
	`  Score := 0 ;`
	`  Category := 'Not Qualified' ;`
	`  PlayerID := InputPlayerID;`
	`end;`
	VB
	`Public Sub New (InputPlayerID)`
	`  Score = 0`
	`  Category = "Not Qualified"`
	`  PlayerID = InputPlayerID`
	`End Sub`
		5

Question	Answer

2(d)(ii)	1 mark per bullet point
	
	• 1 get Method header without parameter (returning correct data type if given)
	• …returning the property
	• A second working Get
	• A third working Get
	
	PYTHON
	`def GetScore():`
	`  return (Score)`
	`def GetCategory():`
	`  return (Category)`
	`def GetPlayerID():`
	`  return (PlayerID)`
	
	PASCAL
	`function GetScore():Integer;`
	`begin`
	`  GetScore:= Score;`
	`end;`
	`function GetCategory():String;`
	`begin`
	`  GetCategory:= Category;`
	`end;`
	`function GetPlayerID():String;`
	`begin`
	`  GetPlayerID:= PlayerID;`
	`end;`
	
	VB
	`Public Function GetScore() As Integer`
	`  Return Score`
	`End Function`
	`Public Function GetCategory() As String`
	`  Return Category`
	`End Function`
	`Public Function GetPlayerID() As String`
	`  Return PlayerID`
	`End Function`

Question	Answer	

2(d)(iii)	1 mark per bullet point
	
	• Set method header and close (where appropriate)
	• Input value
	• Looping until input value is correct length …
	• … storing **valid** input value in `PlayerID`
	
	PYTHON
	`def SetPlayerID(self)`
	`  PlayerID = input("Enter your player ID")`
	`  while len(PlayerID) > 15 and len(PlayerId) < 4`
	`  PlayerID = input("Must be <=15 AND >=4 characters long. Enter your player ID")`
	
	PASCAL
	`Procedure SetPlayerID ()`
	`  WriteLn ('Enter your player ID');`
	`  ReadLn(PlayerID);`
	`  while length(PlayerID) > 15 and length(PlayerID) < 4 do`
	`  begin`
	`  WriteLn('Must be <=15 AND >=4 characters long. Enter your player ID');`
	`  ReadLn(PlayerID);`
	`  end;`
	
	VB
	`Public Sub SetPlayerID()`
	`  Console.WriteLine ("Enter your player ID")`
	`  PlayerID = Console.ReadLine()`
	`  While Len(PlayerID) > 15 and Len(PlayerID) < 4`
	`  Console.WriteLine ("Must be <=15 AND >=4 characters long. Enter your player ID")`
	`  PlayerID = Console.ReadLine()`
	`  End While`
	`End Sub`	

Question	Answer

2(d)(iv)	1 mark per bullet point
	
	• Function header and close (where appropriate) **and** takes `ScoreInput` as parameter
	• Check if 0 <= `ScoreInput` <= 150
	• …if valid, set `Score` to parameter
	• …if not valid, output error
	• Returns TRUE if valid **and** returns FALSE if not valid
	
	PYTHON
	`def __SetScore(ScoreInput):`
	`  if ScoreInput >=0 and ScoreInput <=150:`
	`  IsValid = True`
	`  self__Score = ScoreInput`
	`  else:`
	`  print("Error")`
	`  IsValid = False`
	`Return(IsValid)`
	
	PASCAL
	`function Player.SetScore(ScoreInput: Integer) : Boolean;`
	`begin`
	`  If (ScoreInput >=0) AND (ScoreInput <=150) Then`
	`  IsValid := True;`
	`  result := ScoreInput;`
	`  Else`
	`  WriteLn('Error')`
	`  result := False;`
	`end;`
	
	VB
	`Public Function SetScore(ByVal ScoreInput As Integer) As Boolean`
	`  If (ScoreInput >=0) And (ScoreInput <=150) Then`
	`  Return True`
	`  Score = ScoreInput`
	`  Else`
	`  Console.Writeline("Error")`
	`  Return False`
	`  End If`
	`End Function`

Question	Answer

2(d)(v)	1 mark per bullet point
	
	• Procedure header and close (where appropriate)
	• Accessing Score attribute
	• Correct selection to assign each category
	• … storing in Category attribute
	
	PYTHON
	`def SetCategory()`
	`  if self.__Score >120:`
	`  self.__Category = "Advanced"`
	`  elif self.__Score >80:`
	`  self.__Category = "Intermediate"`
	`  elif self.__Score>=50:`
	`  self.__Category = "Beginner"`
	`  else:`
	`  self.__Category = "Not Qualified"`
	
	PASCAL
	`procedure player.SetCategory()`
	`begin`
	`  If Score >120 Then`
	`  Category := "Advanced";`
	`  Else If Score >80 Then`
	`  Category := "Intermediate";`
	`  Else If Score >= 50 Then`
	`  Category := "Beginner";`
	`  Else`
	`  Category := "Not Qualified";`
	`end;`
	
	VB
	`Public Sub SetCategory()`
	`  If Score >120 Then`
	`  Category = "Advanced"`
	`  ElseIf Score >80 Then`
	`  Category = "Intermediate"`
	`  ElseIf Score >=50 Then`
	`  Category = "Beginner"`
	`  Else`
	`  Category = "Not Qualified"`
	`  End If`
	`End Sub`

Question	Answer

2(d)(vi)	1 mark per bullet point
	
	• `CreatePlayer()` header and close (where appropriate)
	• Input of `score` and `PlayerID` with suitable prompts
	• Create instance of `Player` named `JoannePlayer` …
	• …with `PlayerID` as parameter
	• Call method `SetScore` for `JoannePlayer` with parameter `Score`
	• …storing return value
	• …outputting appropriate message for not valid
	• Call `SetCategory` for `JoannePlayer`
	• Output Category for `JoannePlayer` …
	• … using `GetCategory` for object Joanne
	
	PYTHON
	`def CreatePlayer():`
	`  InputPlayerID = input("Enter your chosen ID")`
	`  Score = int(input("Please enter the score"))`
	`  JoannePlayer = Player(InputPlayerID)`
	`  if JoannePlayer.SetScore(Score) == false:`
	`  print("Invalid score")`
	`  else:`
	`  JoannePlayer.SetCategory()`
	`  print(JoannePlayer.GetCategory)`
	
	PASCAL
	`procedure CreatePlayer();`
	`  var`
	`  playerID : String;`
	`  isValid : boolean;`
	`  JoannePlayer : Player;`
	`  score : integer;`
	`  begin`
	`  Writeln(Enter Player ID: ');`
	`  Readln(playerID);`
	`  Writeln('Enter score: ');`
	`  Readln(score);`
	`  JoannePlayer := Player.Create(PlayerID);`
	`  isValid := JoannePlayer.SetScore(Score);`
	`  if isValid = true:`
	`  JoannePlayer.SetCategory();`
	`  Writeln(JoannePlayer.GetCategory());`
	`  else:`
	`  Writeln("Invalid score")`
	`end;`

<table>
<tr><th>Question</th><th>Answer</th><th></th></tr>
<tr><td>2(d)(vi)</td><td>VB
<pre>
Sub CreatePlayer()
    Dim Score As Integer, InputPlayerID As String
    Console.WriteLine("Please enter your chosen ID")
    InputPlayerID = Console.ReadLine()
    Console.WriteLine("Please enter the score")
    Score = Console.ReadLine()

    Dim JoannePlayer As New Player(InputPlayerID)
    if JoannePlayer.SetScore(Score) = True then
        JoannePlayer.SetCategory()
        Console.WriteLine(JoannePlayer.GetCategory())
    else
        Console.Writeline("Invalid score")
    endif
End Sub
</pre></td><td></td></tr>
<tr><td>2(e)</td><td>1 mark per bullet point

• 3 correct Normal test data
• 3 correct Abnormal test data
• 3 correct Boundary test data

<table>
<tr><th>Category</th><th>Type of test data</th><th>Example test data</th></tr>
<tr><td rowspan="3">Beginner</td><td>Normal</td><td>e.g. 75</td></tr>
<tr><td>Abnormal</td><td>e.g. 85 / bob</td></tr>
<tr><td>Boundary</td><td>80, 50</td></tr>
<tr><td rowspan="3">Intermediate</td><td>Normal</td><td>e.g. 95</td></tr>
<tr><td>Abnormal</td><td>e.g. 70 / bob</td></tr>
<tr><td>Boundary</td><td>81, 120</td></tr>
<tr><td rowspan="3">Advanced</td><td>Normal</td><td>e.g. 125</td></tr>
<tr><td>Abnormal</td><td>e.g. 115 / bob</td></tr>
<tr><td>Boundary</td><td>121, 150</td></tr>
</table></td><td>3</td></tr>
<tr><td>2(f)(i)</td><td>Insertion sort</td><td>1</td></tr>
<tr><td>2(f)(ii)</td><td>One from:
• Bubble sort
• Merge sort</td><td>1</td></tr>
</table>

Question	Answer
2(f)(iii)	1 mark per shaded section

Item	NumberOfScores	InsertScore	Index	ArrayData				
				0	1	2	3	4
				99	125	121	109	115
1	5	125	0		(125)			
2		121	1			125		
			0		121			
3		109	2				125	
			1			121		
			0		109			
4		115	3					125
			2				121	
			1			115		

QUESTION 7.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>4(a)(i)</td><td>1 mark for error and correction

Error 1 – <code>IF List[LowerBound] = SearchValue</code>
Correction – <code>IF List[MidPoint] = SearchValue</code>

Error 2 – <code>UpperBound ← MidPoint + 1</code>
Correction – <code>LowerBound ← MidPoint + 1</code>

Error 3 – <code>IF LowerBound > MidPoint</code>
Correction - <code>IF LowerBound > UpperBound</code>

Error 4 – <code>IF ValueFound = FALSE</code>
Correction – <code>IF ValueFound = TRUE</code></td><td>4</td></tr>
<tr><td>4(a)(ii)</td><td>Linear search</td><td>1</td></tr>
<tr><td>4(b)(i)</td><td>1 mark per shaded section

<table>
<tr><th rowspan="2">Count</th><th rowspan="2">TempValue</th><th rowspan="2">Sorted</th><th colspan="6">ArrayData</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>0</td><td>""</td><td>TRUE</td><td>5</td><td>20</td><td>12</td><td>25</td><td>32</td><td>29</td></tr>
<tr><td>1</td><td>12</td><td>FALSE</td><td></td><td>12</td><td>20</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>29</td><td>(FALSE)</td><td></td><td></td><td></td><td></td><td>29</td><td>32</td></tr>
<tr><td>0</td><td></td><td>TRUE</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></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></td><td></td><td></td></tr>
</table>
</td><td>4</td></tr>
</table>

Question	Answer	Marks

4(b)(ii)	1 mark per bullet point
	
	• Initialising a counter variable to 0 **and** must be the same variable used to access array elements
	• While loop checking counter is < 5 or <= 4
	• Incrementing counter inside the loop and outside IF, **and** remainder of algorithm completed (The IF to ENDIF)
	
	e.g.
	
	`Count ← 0`
	`WHILE Count < 5`
	`  IF ArrayData[Count] > ArrayData[Count + 1]`
	`  THEN`
	`  TempValue ← ArrayData[Count + 1]`
	`  ArrayData[Count + 1] ← ArrayData[Count]`
	`  ArrayData[Count] ← TempValue`
	`  Sorted ← False`
	`  ENDIF`
	`  Count ← Count + 1`
	`ENDWHILE`
		3

4(b)(iii)	Bubble sort	**1**
4(b)(iv)	One from: • Insertion sort • Merge sort • Quick sort	**1**

QUESTION 8.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>5(a)</td><td>1 mark for each shaded section

LDD NUMBER
LSL
#2
STO NUMBER
END

<table>
<tr><th>Label</th><th>Op code</th><th>Operand</th><th>Comment</th></tr>
<tr><td></td><td>LDD</td><td>NUMBER</td><td>// load contents of NUMBER</td></tr>
<tr><td></td><td>LSL</td><td>#2</td><td>// perform shift to multiply by 4</td></tr>
<tr><td></td><td>STO</td><td>NUMBER</td><td>// store contents of ACC in NUMBER</td></tr>
<tr><td></td><td>END</td><td></td><td>// end program</td></tr>
<tr><td>NUMBER:</td><td colspan="2">B00110110</td><td></td></tr>
</table></td><td>5</td></tr>
</table>

Question	Answer	Marks
5(b)		**8**

Label	Op code	Operand	Comment	
	LDR	#0	// initialise index register to 0	
START:	**LDX**	**STRING**	// load the next value from STRING	**1**
	AND	**MASK**	// bitwise AND operation with MASK	**1**
	CMP	**MASK**	// check if result equals MASK	**1**
	JPN	**UPPER**	// if FALSE jump to UPPER	**1**
	LDD	**COUNT**	// increment COUNT	**1**
	INC	**ACC**		
	STO	**COUNT**		
UPPER:	INC	IX	// increment Index Register	**1**
	LDD	**LENGTH**	// decrement LENGTH	**1**
	DEC	**ACC**		
	STO	**LENGTH**		
	CMP	**#0**	// is LENGTH = 0 ?	**1**
	JPN	**START**	// if FALSE, jump to START	**1**
	END		// end program	

Question	Answer	Marks
5(b)		

MASK:	B00100000	// if bit 5 is 1, letter is lower case	
COUNT:	0		
LENGTH:	5		
STRING:	B01001000	// The ASCII code for 'H'	
	B01100001	// The ASCII code for 'a'	
	B01110000	// The ASCII code for 'p'	
	B01110000	// The ASCII code for 'p'	
	B01011001	// The ASCII code for 'Y'	