1. (a) Any of the two mistakes identified with supporting descriptions, 1 pt each [2]
   - Line 04 `Middle ⟵ Lower + Upper` should be `Middle ⟵ ( Lower + Upper ) DIV 2`, as this is a binary search algorithm and we are finding the elements by splitting the array in the middle and searching in the smaller lower and upper halves of the array
   - Line 07 `Lower ⟵ Middle + 1` should be `Upper ⟵ Middle - 1` as when `SearchItem`, if it exists, is less than the value in the middle should belong in the lower half of the array. So we move the upper boundary down.
   - Line 09 `Upper ⟵ Middle - 1` should be `Lower ⟵ Middle + 1` as when `SearchItem`, if it exists, is greater or equal than the value in the middle should belong in the upper half of the array. So we move the lower boundary up.
   - *Program code doesn't handle the case that `SearchItem` doesn't belong in the array

   Fix proposed corrects the output of the proram for all the values in `Value` [1]

   (b) Logic Error [1]

2. (a)
   - 3 classes, `WORKER`, `NANNY`, `CLEANER` [1]
   - Correct inheritance shown with upward pointing arrows [1]
   - Correct distribution of attributes for at least one class.. [1]
     - ... for at least 2 classes [1]
     - ...for all 3 classes [1]
   - Identification of appropriate `get`, `set` methods for at least one class... [1]
     - ... for at least 2 classes [1]
     - ... for all 3 classes [1]

   (b) We see that
   - cleaners and janitors can both clean [1]
   - however, janitor is able to do minor plumbing and electrical repairs which cleaners can't [1]
   - so new class `JANITOR` with parent class `CLEANER` [1]
   - Janitor class has property of `MaxWorks` [1]
   - has methods for setting and getting `MaxWorks` [1]

   (c) i. Attributes and methods to manipulate attributes held in the same structure. So, it allows for easier manipulation and troubleshooting [1]

   ii. Data/Information Hiding. Implementation details hidden and data only accessible via class interface, i.e get and set methods, which prevents accidental modification of data [1]

   iii. Validity check on values when setting attributes can be easily handled without alteration of method calls [1]

   (d) Polymorphism is the technique where methods behaves differently for different classes in the hierarchy.

3. (a)

   - All entities shown `Participant`, `ParticipantID`, `Result` [1]
   - All relationships with correct cardinality [2]

   (b) One such tables description is

```
Participant(NRIC, Name, Gender, Age, Contact Number)
ParticipantID(ID*,NRIC*,Year )
Result(ID*, ScoreAIML, ScoreCC, ScoreDS, Overall, Rank)
```

   - All relations depicted in ERD shown with correct PK [1]
   - All correct foreign keys shown [2]

   (c) One such SQL code is

```
SELECT ID
From ParticipantID                    //Correct table and column[1]
WHERE Year = 2021                     //Correct WHERE [1]
```

(d) Use the SQL code

```
SELECT ScoreDS
From Result
WHERE ID = 1234                                                    //Correct SQL [1]
```

If the return of the `SELECT` statement is -1, then the competitior is disqualified for the DS event//Correct Check [1]

(e) Assuming the Year is `year`. Use the SQL code

```
SELECT ID, Overall
WHERE Year = year
ORDER BY Overall ASC                                               //Correct SQL [1]
```

This will return records of the participants in the year `year` sorted in ascending order by their overall marks.

Next, we will need to update the table

```
DECLARE Record :  RECORD
DECLARE List :  LIST OF RECORD
DECLARE Count :  INTEGER

Count ⟵ 1
FOR EACH Record in List DO
    IF Record.Overall <> -1 THEN                          //Check for disqualified entry[1]
        UPDATE Result SET Rank = Count                          //Correct UPDATE[1]
        Count ⟵ Count + 1
    ELSE
        UPDATE Result SET Rank = -1
    ENDIF
ENDFOR                                                    //Correct pseudocode[1]
```

4. (a) i. The ideal pivot is the median of the values in the array as it will allow the array to be split roughly evenly. [1]

   ii. Array must be already sorted to find the median. [1]

   iii. Random selection of pivot reduces the probability of choosing a pivot that gives the worst case time complexity .. [1]
   ...e.g., when the array is already sorted/partially sored sorted or even reverse in order [1]

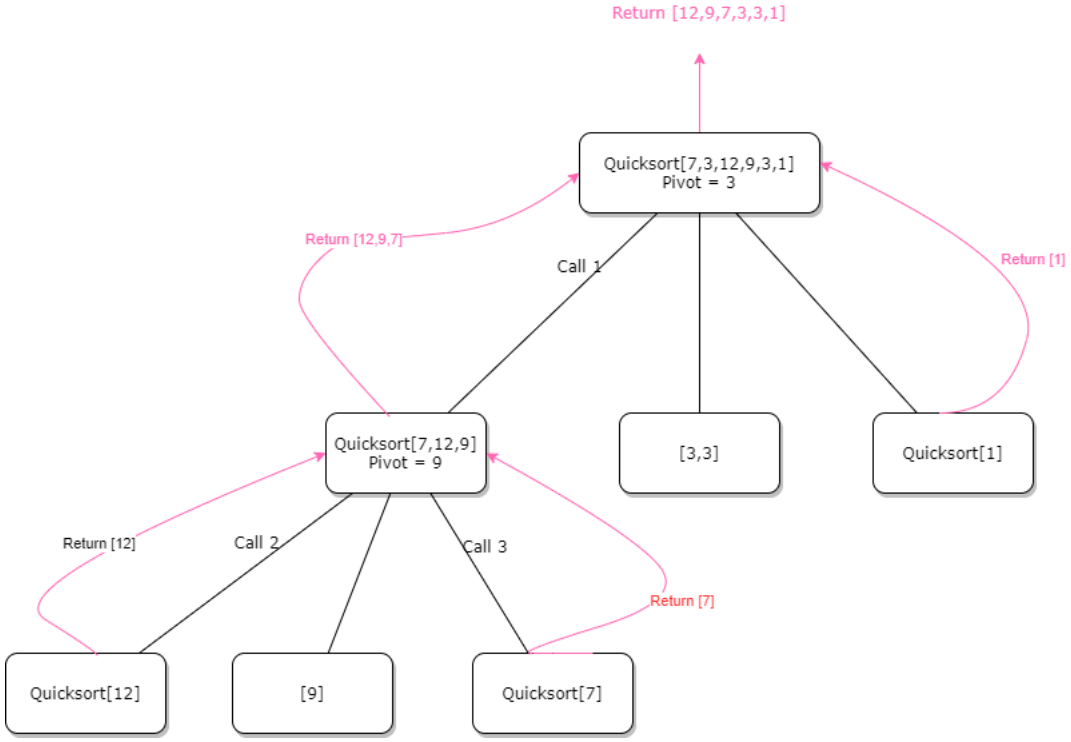
   iv. One such pseudocode is

```
FUNCTION QS(L: List )
DECLARE
    Lesser :  List
    Equal :  List
    Greater :  List
    Pivot:  OBJECT
    Lesser ⟵ CREATE_LIST(0)
    Equal ⟵ CREATE_LIST(0)
    Greater ⟵ CREATE_LIST(0)                    //Creation of Lesser, Greater lists[1]
    IF L.Size = 0 THEN
        RETURN CREATE_LIST(0)                                  //Handling of base case[1]
    ENDIF
    Pivot ⟵ L[1]
    FOR i = 2 TO LEN(L)
        IF L[i] < Pivot THEN
            APPEND(Lesser, L[i])
        ELSE IF L[i] > Pivot THEN
            APPEND(Greater, L[i])       // Adding the correct elements into Lesser and
Greater [1]
        ELSE IF L[i] = Pivot THEN
            APPEND(Equal, L[i])
        ENDIF
    ENDFOR
    RETURN CONCATENATE( QS(Higher), Equal ,QS(Lower))   // Correct Recursive call with
correct types and all operators used[1]
ENDFUNCTION
```

(b) Any one of Insertion Sort, Bubble Sort, Merge Sort or any other valid sort. [1]

(c) i.

Function call	Middle	Right	Left	Return
f([7,3,12,9,3,1])	[12]	[]	[7,3,9,3,1]	
f([7,3,9,3,1])	[3,3]	[7,9]	[1]	
f([7,9])	[7]	[9]	[]	
f([9])	[9]	[]	[]	[]+[9]=[9]
f([7,9])	[7]	[9]	[]	[9]+[7]=[9,7]
f([1])	[1]	[]	[]	[]+[1]=[1]
f([7,3,9,3,1])	[3,3]	[7,9]	[1]	[9,7]+[3,3]+[1]=[9,7,3,3,1]
f([7,3,12,9,3,1])	[12]	[]	[7,3,9,3,1]	[]+[12]+[9,7,3,3,1]=[12,9,7,3,3,1]



- Frame or table or tree diagram observed [1]
- At least 2 recursive function calls observed [1]
- All calls and return values are correct [1]

ii. Without loss of generality, we will assume the sorting is ascending (but actually, descending could be shorter). Underlined elements indicate that they're already sorted.
Bubble sort trace:

```
[7,3,12,9,3,1]
[3,7,12,9,3,1] //Swap
[3,7,12,9,3,1] //NoSwap
[3,7,12,9,3,1]
[3,7,9,12,3,1] //Swap
[3,7,9,12,3,1]
[3,7,9,3,12,1] //Swap
[3,7,9,3,12,1]
[3,7,9,3,1,12] //Swap
[3,7,9,3,1,12]
[3,7,9,3,1,12] //NoSwap
[3,7,9,3,1,12]
[3,7,3,9,1,12] //Swap
[3,7,3,9,1,12]
[3,7,3,1,9,12] //Swap
[3,7,3,1,9,12]
[3,7,3,1,9,12] //NoSwap
[3,3,7,1,9,12] //Swap
[3,3,7,1,9,12]
[3,3,1,7,9,12] //Swap
[3,3,1,7,9,12]
[3,3,1,7,9,12] //NoSwap
[3,1,3,7,9,12] //Swap
[3,1,3,7,9,12]
[1,3,3,7,9,12] //Swap
[1,3,3,7,9,12]
```

- demonstrate swapping [1]
- at least 3 correct traces [1]
- all correct traces [1]

Underlined elements indicate the elements being compared with the starred element before the insertion. Insertion sort trace:

```
[7,3*,12,9,3,1]
[3*,7,12,9,3,1]
[3,7,12*,9,3,1]
[3,7,12,9*,3,1]
[3,7,12,9*,3,1]
[3,7,9*,12,3,1]
[3,7,9,12,3*,1]
[3,7,9,12,3*,1]
[3,7,9,12,3*,1]
[3,7,9,12,3*,1]
[3,3*,7,9,12,1]
[3,3,7,9,12,1*]
[3,3,7,9,12,1*]
[3,3,7,9,12,1*]
[3,3,7,9,12,1*]
[3,3,7,9,12,1*]
[1*,3,3,7,12,9]
```

- demonstrate comparison with previous elements [1]
- at least 3 correct traces [1]
- all correct traces [1]

iii. A. The approach used in two algorithms are different. At each iteration in Insertion sort, one element is removed from the input list and get inserted into its correct position in a partially sorted output list. It repeats until no input elements remain. On the other hand, quicksort algorithm repeatedly partitions the input list, based on pivot elements at each partitioning, into 3 sublists that contain elements smaller, equal and greater than the pivot elements respectively until there are only at most 1 element in the smaller and greater sublists. The sublists were then combined together to form a sorted list. [1]

B. Insertion sort has average time complexity $O\left(n^2\right)$ while Quicksort's average time complexity is $O\left(n \log n\right)$. [1]

(d) Insertion sort will be better for a largely sorted data set...

... as for insertion sort to insert element in largely sorted list, it mostly only need to do one comparison (the ith and $i+1$th element in the list) to determine that it only need to append the $i+1$th element and the end of the partially sorted list. On the other hand, quicksort algorithm doesn't benefit from this and will still need to do the repeated partitioning. [1]

5. (a) Recursion is the ability of a function or procedure to call itself. [1]

(b) Line 05 indicate that `f` is recursive. [1]

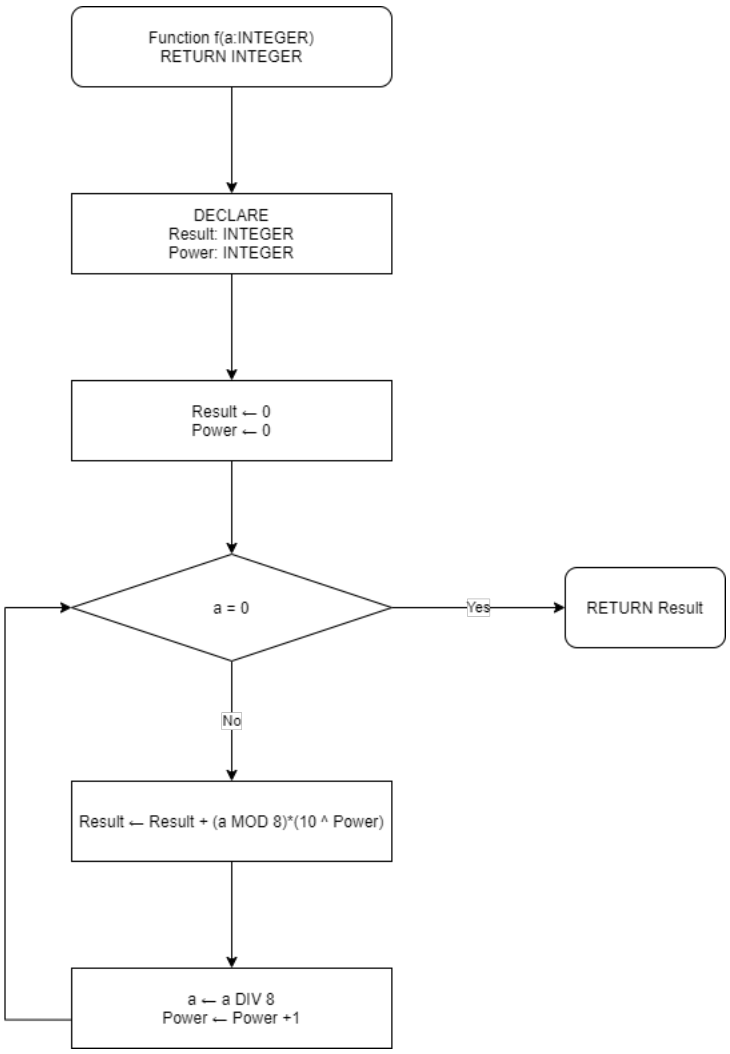
(c)

Function call	a	Return
f(2534)	2534	6+10(474)=4746
f(316)	316	4+10(47)=474
f(39)	39	7+10(4)=47
f(4)	4	4+10(0)=4
f(0)	0	0

- All correct function call [1]
- All correct `a` [1]
- At least 3 correct return values seen [1]
- 4746 returned as the final one. [1]

(d) The function `f` converts the denary integer input into its base 8 representation. [1]

(e)



- Selection (Diamond) `a = 0` or `a > 0` with correct false case [1]
- Loop with correct body (rectangle) [1]
- All shapes and output correct [1]

(f) The advantage is that recursive solution is usually more elegant as the problem space can be reduced to a smaller problem space and use less program code. [1]

However, the disadvantage is that repeated recursive calls can carry large amounts of memory usage and processor time. [1]