DUNMAN HIGH SCHOOL
Preliminary Examination
Year 6

COMPUTING

9569 / 01

Paper 1 Written

22 Sep 2021

3 hours

READ THESE INSTRUCTIONS FIRST

An answer booklet will be provided with this question paper. You should follow all the instructions on the front cover of the answer booklet. If you need additional answer paper ask the invigilator for a continuation booklet.

Answer **all** questions.
Approved calculators are allowed.

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

This document consists of **8** printed pages.

1 A cosmic ray striking computer memory at just the right time can flip a bit, turning a 0 into a 1 or vice versa.

Bit flips had caused plane accidents, software glitches, and at times, the Blue Screen of Death (BSoD) on personal computers.

(a) The hexadecimal number `D4` had been changed to `9A`. Suggest the minimum number of times a bit flip could have affected it and explain why. [2]

4 times [1]

Show or explain the affected bits:
D4 = 1101 0100
9 = 1**0**01
A = **101**0
[1]

(b) Explain why Unicode characters, as compared to ASCII characters, are more likely to be altered after a cosmic ray strike. [2]

- Unicode is a superset of ASCII. ASCII defines 128 characters, which map to the numbers 0–127. Unicode defines about 2^{21} characters, which, similarly, map to numbers 0–2^{21}. [1]
- The more bits used to store the characters, the more exposure there is to a cosmic ray strike, so the more likelihood of alteration occurring. [1]

(c) Define what a backup and archive is, and explain which should be prioritised if a company is warned of a global cosmic ray strike occurring in a week. [4]

1. **Definition**
    1. **Backup**
        1. Is a duplicate redundant copy of current working files [1]
    2. **Archival**
        1. Stores data that's no longer in day-to-day use but still needs to be retained for historical purpose, future audit or legal requirements or ad-hoc reference eg past year students records (due to their their enduring cultural, historical, or evidentiary value) [1]
2. **Priority**
    1. **Backup [1]**
    2. Reason for backup [1]:
        1. To provide rapid recovery for operational data to resume normal operation in the event of the original file loss (e.g. could be a hard disk crash or data corruption due to bit flip).
        2. OR Prevent unintended data corruption or data loss
    3. Reason not to prioritise archival [1]
        1. An archive would not be designed for frequent access, and may not be up to date
        2. Archival doesn't backup all the files, so still will result in loss of data

(d) Explain how TCP/IP layers help to fulfil the purpose of the TCP/IP model. [2]

TCP/IP model ensures **data packets reach and are understood by the intended recipient** (OR communication protocol) [1].

This process is complex because it involves how data should be sent/received (i.e packetized, addressed, transmitted, routed)

This complexity is managed by br**eaking down its components into layers for specialization/efficiency** / Each layer provides functionality that depends solely on the layer directly below it [1].

(e) Suggest and explain a consequence on network communication when a data packet is affected by a bit flip occurring at the Internet Layer [2]

Packet will not take the correct route through the correct gateway in the network [1] and will not reach its destination [1].

(f) Explain why the consequence of a bit flip occurring at the Internet Layer and the Transport Layer would be the same as the consequence as a bit flip occurring at the Internet Layer only. [1]

The functionality of the Transport Layer would only be accessed if the packet was delivered to the destination device [1], which would not happen.

2 An array stores 16 powers of 2 integers in ascending order:

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768

(a) If the array is searched by means of a binary search, state which elements would be accessed, and in what order,

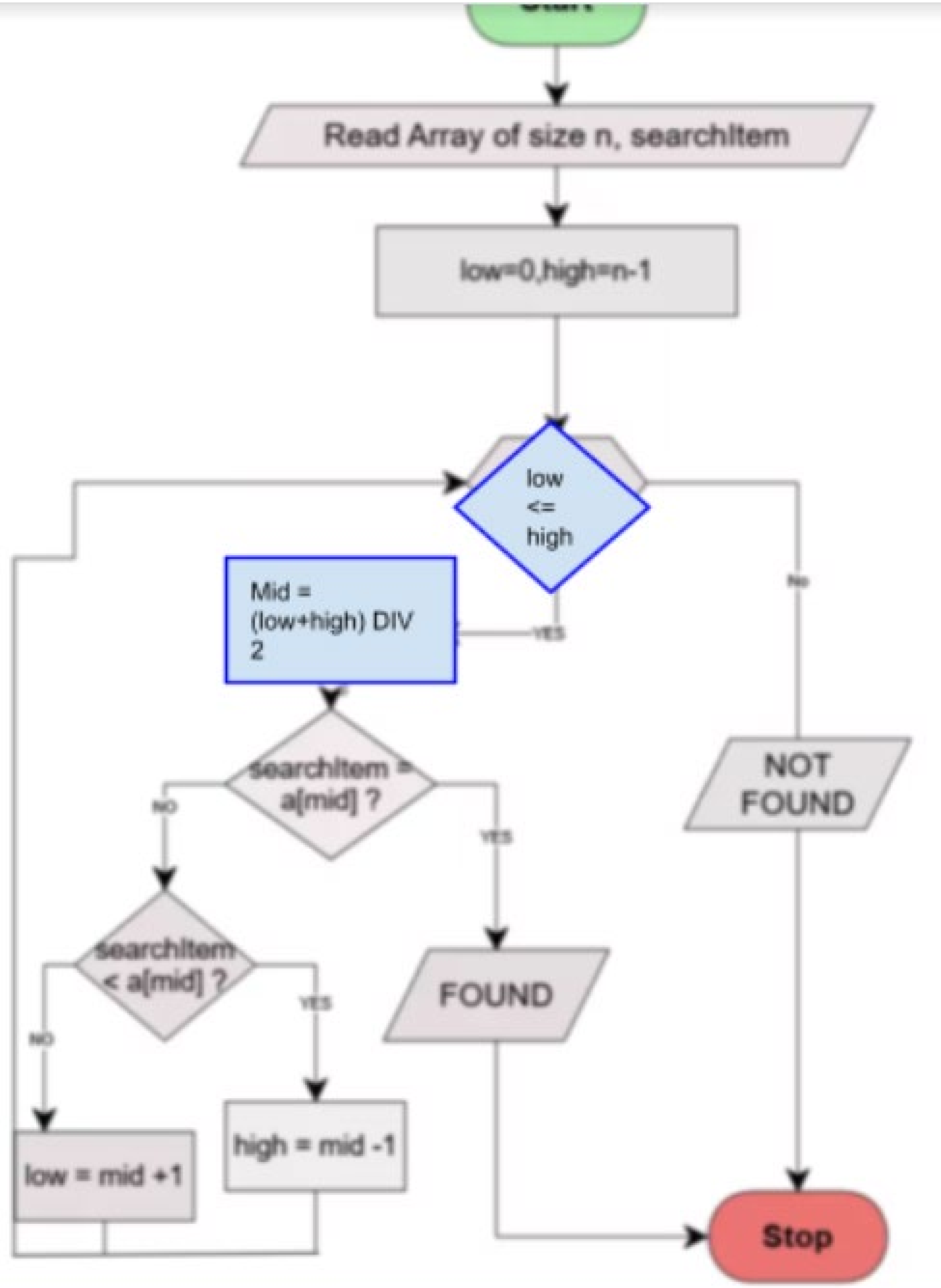
i) when searching for the number 4096 (which is present), and [1]

128, 2048, 8192, 4096

ii) when searching for the number 3 (which is not present). [1]

128, 8, 2, 4

(b) Draw a program flowchart of an iterative binary search algorithm. [6]



(c) Explain why binary search could be more efficient than linear search. [1]

Binary search reduces the problem/data size by half in every iteration.

(d) State the time complexities of hash table search and binary search and explain which is more efficient with reference to their time complexities in the above scenario. [4]

Hash table search: O(1) or O(n) if there are collisions in worst case [1]
Binary search: O(log(n)) [1]

Hash table search is more efficient [1] same number of operations done regardless of the number of elements, but for binary search increasing operations needed for increasing number of elements [1]

3 A mobile network provider's management of customer's overdue bills include an automated emailing and SMS system.

- If a mobile bill is overdue, a daily system generated reminder is emailed to the user indicating the overdue details.
- If the user has 1 mobile bill that is more than 6 days overdue, in place of the daily reminder email, a warning email and SMS will be sent to the user.
- If a user has more than 4 bills which are more than 14 days overdue, the user will incur a penalty fee for every additional week starting from the 14th day of being overdue.
- If a user has incurred more than 3 penalty fees, the user's mobile service will be terminated.
- Penalty fees could be incurred as a result of other actions such as accessing illegal websites.
- Each bill issued to a customer represents the past month's usage.

(a) Create a decision table showing all possible outcomes and results for bill(s) that are overdue. [4]

	Conditions								
1	One bill overdue by more than 6 days	Y	Y	Y	Y	N	N	N	N
2	More than 4 bills more than 14 days overdue	Y	Y	N	N	Y	Y	N	N
3	More than 3 penalty fees	Y	N	Y	N	Y	N	Y	N
	Actions								
1	Daily email reminder								X
2	Warning email and SMS				X				
3	Incur weekly penalty fee		X						
4	Terminate mobile service	X		X				X	

(b) Simplify the decision table by removing redundancies. [3]

	Conditions				
1	One bill overdue by more than 6 days	Y	Y	-	N
2	More than 4 bills more than 14 days overdue	Y	N	-	N
3	More than 3 penalty fees	N	N	Y	N
	Actions				
1	Daily email reminder				X
2	Warning email and SMS		X		
3	Incur weekly penalty fee	X			
4	Terminate mobile service			X	

(c) A user interface is designed for a program for customers to check their bills and overdue status. State a usability principle and describe how the user interface of the program should be designed to demonstrate this principle. [2]

Any valid pair in context of overdue scenario:

- #1: Visibility of system status
  - The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time.
- #2: Match between system and the real world

  - The design should speak the users' language. Use words, phrases, and concepts familiar to the user, rather than internal jargon. Follow real-world conventions, making information appear in a natural and logical order.
- #3: User control and freedom
  - Users often perform actions by mistake. They need a clearly marked ""emergency exit"" to leave the unwanted action without having to go through an extended process.
- #4: Consistency and standards
  - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.
- #5: Error prevention
  - Good error messages are important, but the best designs carefully prevent problems from occurring in the first place. Either eliminate error-prone conditions, or check for them and present users with a confirmation option before they commit to the action.
- #6: Recognition rather than recall.
  - Minimize the user's memory load by making elements, actions, and options visible. The user should not have to remember information from one part of the interface to another. Information required to use the design (e.g. field labels or menu items) should be visible or easily retrievable when needed.
- #7: Flexibility and efficiency of use
  - Shortcuts — hidden from novice users — may speed up the interaction for the expert user such that the design can cater to both inexperienced and experienced users. Allow users to tailor frequent actions
- .#8: Aesthetic and minimalist design
  - Interfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.
- #9: Help users recognize, diagnose, and recover from errors
  - Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution
- #10: Help and documentation
  - It's best if the system doesn't need any additional explanation. However, it may be necessary to provide documentation to help users understand how to complete their tasks.

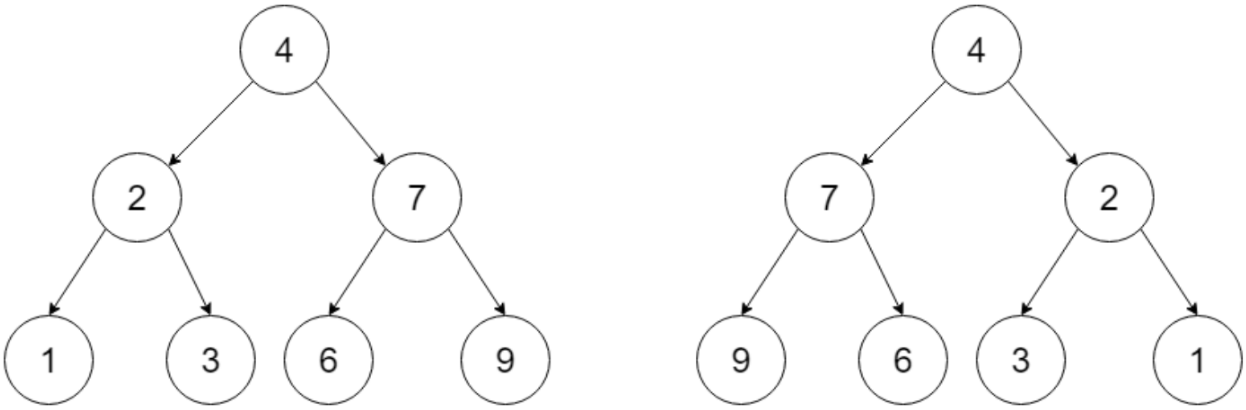
4 **(a)** Explain how a linked list data structure could be more suitable than an array data structure to implement a binary tree. [2]

Unpredictable expansion for growth OR difficult to predict or determine memory requirements at the start OR more efficient to have dynamic memory allocation. Expected high number of insertions and re-ordering, making reorganization for arrays expensive [1]

(b) Suggest and justify one circumstance where an array structure is more appropriate than a linked structure. [2]

- Memory allocation decision made at design time so memory can be freed up for other purposes OR the most often use case of tree is direct access to i-th element instead so no need for traversing tree to perform CRUD

The diagram shows a binary tree before and after an inversion.



Pre-inverted tree

Post-inverted tree

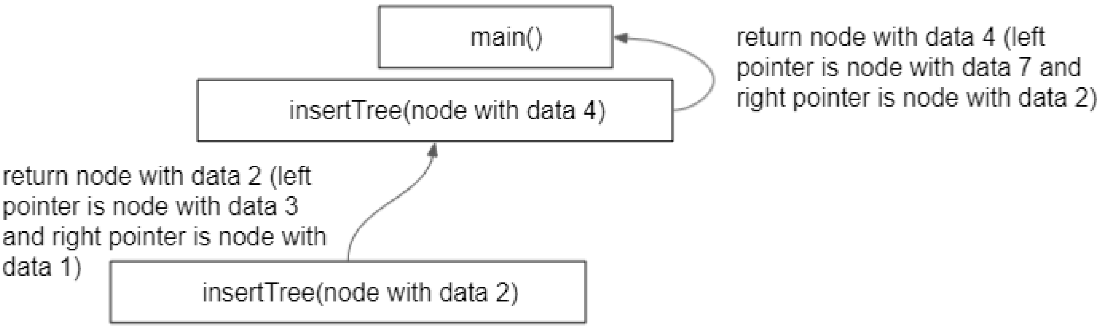
Each node has these attributes:

- `right` which is the pointer to right subtree
- `data` which is the value contained in each respective node displayed above
- `left` which is the pointer to the left subtree

(c) Write pseudocode for procedure `insert` which will add a node to the post-inverted tree (which may be empty) in such a way that if the new value of the node is **less** than the value at the current node, the new node will be added to the **right** subtree, or else it will be added to the left subtree. `insert` takes in values `node_data` and `root_node` which is the value to be added to the tree and the instance of the root node of the tree respectively. [6]

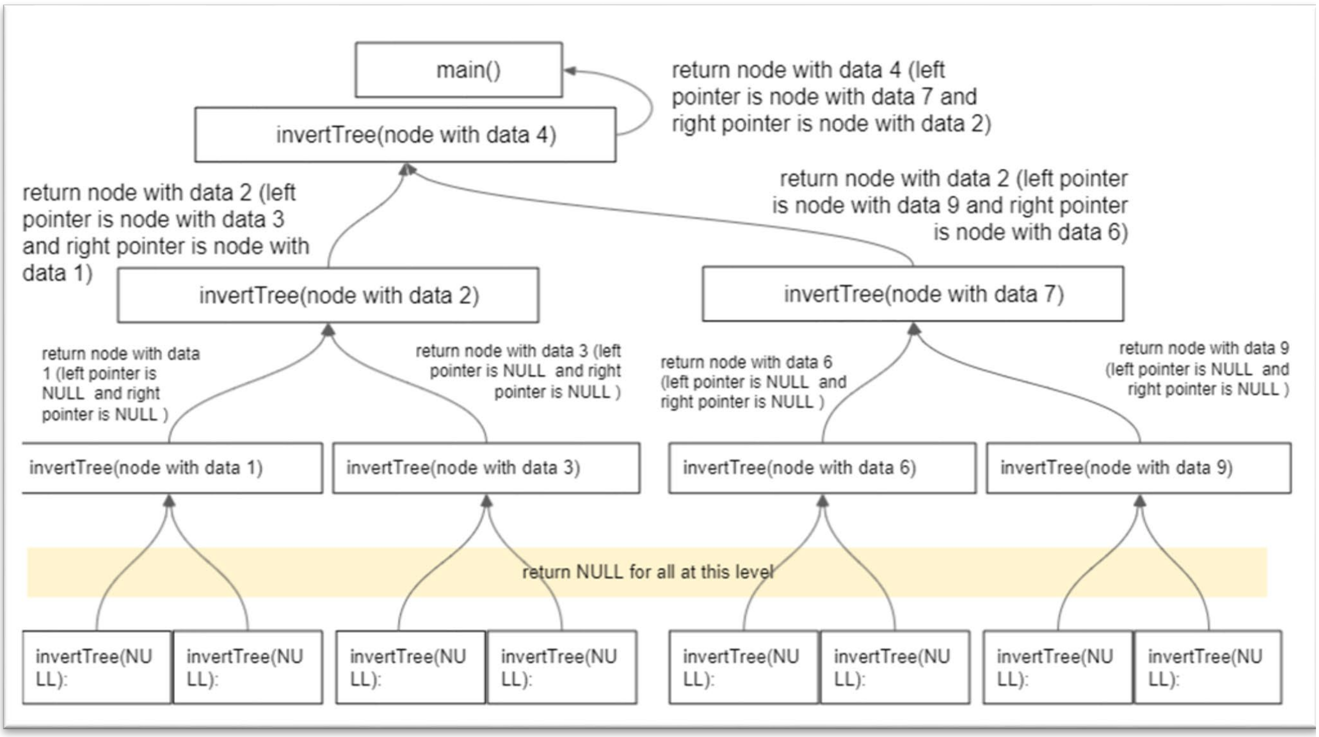
```
PROCEDURE insert(root_node: instance of root node, node_data: INTEGER) [1]
    IF root_node IS NULL THEN // if empty tree                        [1]
        root_node.data ← node_data // place node_data at root
        root_node.left ← NULL
        root_node.right ← NULL                         [1]

    ENDIF

    IF root_node.value < node_data THEN                       [1]
        root_node.left ← insert(root_node.left,node_data) // insert to left                [1]
    ELSE
        root_node.right ← insert(root_node.right,node_data) // insert to right              [1]
    ENDIF

ENDPROCEDURE
```

(d) A function `invertTree` takes in the root node of the above pre-inverted tree, uses **recursion** to invert it into the post-inverted tree, and returns the root node of the post-inverted tree. Write pseudocode for the function and visualise it in a trace diagram. An incomplete trace diagram is provided for you to begin with. Copy and complete it.


main()
return node with data 4 (left pointer is node with data 7 and right pointer is node with data 2)
insertTree(node with data 4)
return node with data 2 (left pointer is node with data 3 and right pointer is node with data 1)
insertTree(node with data 2)

(Note: Should be invertTree, not insertTree)

[6]


main()
return node with data 4 (left pointer is node with data 7 and right pointer is node with data 2)
invertTree(node with data 4)
return node with data 2 (left pointer is node with data 3 and right pointer is node with data 1)
return node with data 2 (left pointer is node with data 9 and right pointer is node with data 6)
invertTree(node with data 2)
invertTree(node with data 7)
return node with data 1 (left pointer is NULL and right pointer is NULL)
return node with data 3 (left pointer is NULL and right pointer is NULL)
return node with data 6 (left pointer is NULL and right pointer is NULL)
return node with data 9 (left pointer is NULL and right pointer is NULL)
invertTree(node with data 1)
invertTree(node with data 3)
invertTree(node with data 6)
invertTree(node with data 9)
return NULL for all at this level
invertTree(NULL):
invertTree(NULL):
invertTree(NULL):
invertTree(NULL):
invertTree(NULL):
invertTree(NULL):
invertTree(NULL):
invertTree(NULL):

Logic for algo used (should be written in pseudocode):

```
    def invertTree(root):
        if root:                  [1]
            root.left, root.right = invertTree(root.right),
    invertTree(root.left)                  [1]
            return root           [1]
```

5 Consider the following data which shows a single Civics Group record used in COVID19 vaccination tracking.

Civics Group Name: 6C35
Civics Tutor: Mr Tan
Chairperson: Steve Lim

Register Number	Student Name	Vaccine brand name	CCA Name	CCA Teacher
11	Steve Lim	Moderna	Choir	Mr Lee
4	Melody Tan	Moderna	Choir	Mr Lee
19	Paul Chang	Pfizer–BioNTech	Soccer	Mr Chan
3	Grace Lee	Pfizer–BioNTech	Dance	Ms Deepa
20	Alison Chong	Not Vaccinated	Infocomm Club	Mrs Ho
12	Vincent Eng	Moderna	Soccer	Mr Chan

(a) Derive a set of tables to show the above data in first, second and third normal form. [6]

1NF: No attribute in the tables can have multiple values.
CIVICSGROUP(**CivicsGroupName**, CivicsTutor, Chairperson)
[1]
STUDENT(**RegisterNumber**, **CivicsGroupName**, StudentName, VaccineBrandName, CCAName, CCATeacher)
[1]

2NF: Composite key needed for VaccineBrandName, CivicsGroupName or RegisterNumber cannot identify VaccineBrandName,
CIVICSGROUP(**CivicsGroupName**, CivicsTutor, Chairperson)
STUDENT(**RegisterNumber**, StudentName, CCAName, CCATeacher)
[1]
STUDENTVACCINE(**CivicsGroupName, RegisterNumber**, VaccineBrandName)
[1]

3NF: No functional dependency.
CCAName and CCATeacher are non-identifying attributes.
CCA(**CCAName**, CCATeacher) and
[1]
STUDENT(**RegisterNumber**, StudentName, CCAName)
[1]

Complete set:
CIVICSGROUP(**CivicsGroupName**, CivicsTutor, Chairperson)
STUDENT(**RegisterNumber**, StudentName, CCAName*)
STUDENTVACCINE(**CivicsGroupName***, **RegisterNumber***, VaccineBrandName)
CCA(**CCAName**, CCATeacher)
[max of 4 marks for complete set without differentiation of 1nf 2nf 3nf]

(b) Draw an ER diagram for a normalised database design. [2]

CIVICSGROUP -<- STUDENTVACCINE ---- STUDENT ->- CCA [1]
STUDENT ->- CCA [1]

(c) Using examples in the above context, explain the significance of the following terms:

i) primary key [2]

Column/field to uniquely identify each record in a database table
CivicsGroupName in CIVICSGROUP table, RegisterNumber in STUDENT table, CCAName in CCA table, CivicsGroupName and RegisterNumber in STUDENTVACCINE table

ii) foreign key [2]

Column/field to link two tables, not unique in current table but is a primary key in its reference table
CCAName in STUDENT table, CivicsGroupName or RegisterNumber in STUDENTVACCINE table

(d) With reference to the above context, describe or suggest a scenario where a NoSQL database would be more appropriate. [1]

eg multiple vaccines by 1 person. Not vaccinated can be replaced as an non-existing field/not optional.

6 Long-distance optical fibre lines and submarine cables which are a vital part of the global internet infrastructure are vulnerable to solar superstorms which happen once in a century. The last solar superstrom was in 1921.

(a) State and explain why websites would or would not be accessible by web browsers if a solar superstorm shuts down all DNS servers. [2]

Yes [1]. Type the IP address of website[1] into the web browser as URL.

(b) Explain *packet switching* and its importance in ensuring global internet connectivity when some parts of the earth are hit by a solar superstorm. [4]

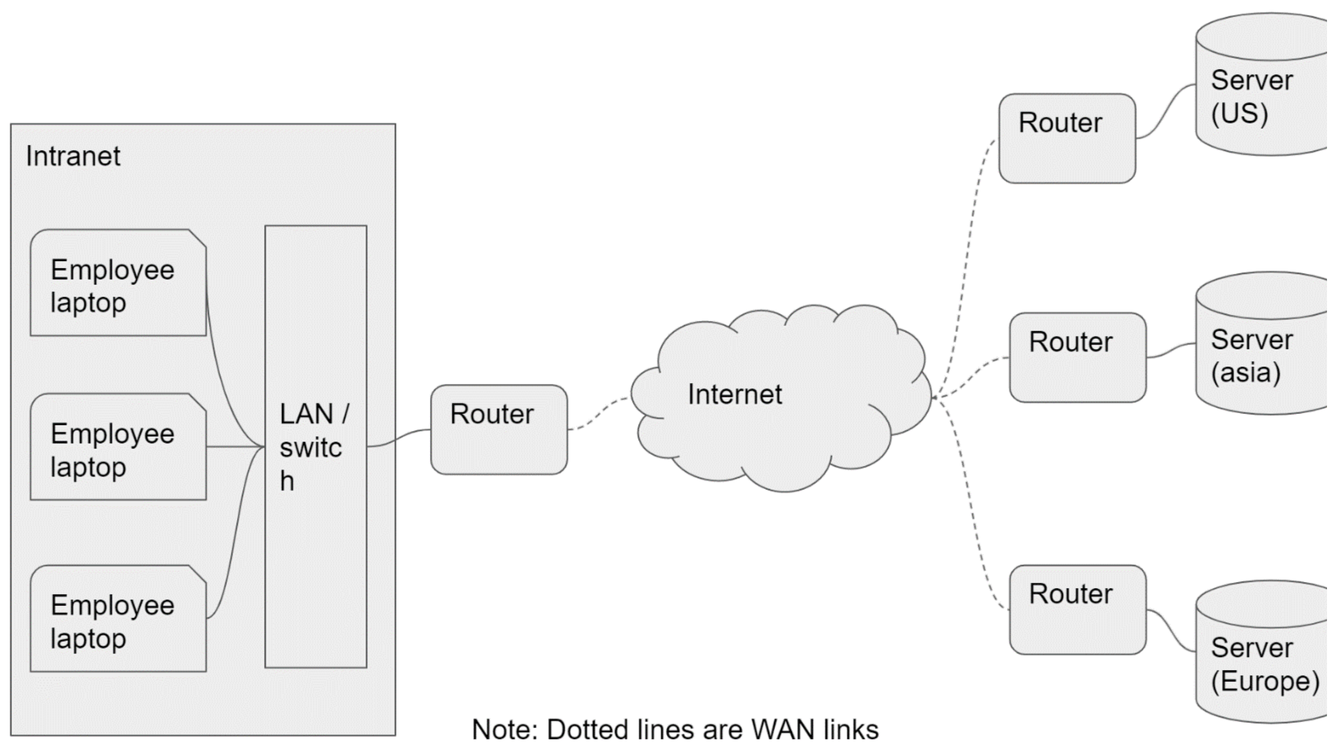
- Data is divided[1] into packets, each with its own ID number
- Each packet is sent from one node to another on the internet[1].
- At each point, each node decides where to forward the packet based on network traffic and node availability.[1]
- routers would not forward packets through routes with nodes affected by solar superstorm [1]

An international company based in many countries updates its network structure to ensure Internet connectivity during solar superstorms.

Employees can now access files on a shared virtual space which is made up of 3 servers located in the United States, Europe, and Asia. All servers hold identical information (any changes made on one server would update the other servers immediately) so even if one server is affected by the solar superstorm, employees still can access their files on the other servers.

Employees must be connected physically to the company's intranet network at each country's office to access the virtual space.

(c) Draw a network diagram of the above configuration and label the LAN, internet, router(s), WAN link(s), intranet, servers, and employee laptops. [5]



Note: Dotted lines are WAN links

(d) Why is version control vital when employees from different countries work in teams? [1]

- avoids the danger of *employee A* modifying a version and *employee B* modifying it in a different way. no correct current version then exists
- ensures that the latest correct edition of the file is being collaborated on by team members
- provides revision history or trace log to monitor progress status (who has done what at when)

7 A divide and conquer approach is used by merge sort to successively divide a list into half, forming two sublists, until each sublist is of length 1. The sublists are then sorted and merged into larger sublists until they are recombined into a single sorted list. An algorithm for merge sort to perform an ascending sort is given below. It will be used to sort large or small data sizes.

```
PROCEDURE mergesort(mergelist : ARRAY)

      IF LENGTH(mergelist) > 1 THEN

            mid ← LENGTH(mergelist) DIV 2

            FOR index ← 0 TO (mid - 1)
                  lefthalf[index] ← mergelist[index]
            NEXT index

            right_len ← LENGTH(mergelist) - mid

            FOR index ← 0  TO (right_len - 1)
                  righthalf[index] ← mergelist[right_len + index]
            NEXT index
```

```
16
17              mergesort(lefthalf)
18              mergesort(righthalf)
19
20              i ← 0
21              j ← 0
22              k ← 0
23              WHILE i < LENGTH(lefthalf) AND j < LENGTH(righthalf)
24                  IF lefthalf[i] > righthalf[j] THEN
25                      mergelist[k] ← lefthalf[i]
26                      i ← i + 1
27                  ELSE
28                      mergelist[k] ← righthalf[j]
29                      j ← j + 1
30                  ENDIF
31                  k ← k + 1
32              ENDWHILE
33
34              WHILE i < LENGTH(lefthalf)
35                  mergelist[k] ← lefthalf[i]
36                  i ← i + 1
37                  k ← k + 1
38              ENDWHILE
39
40              WHILE j < LENGTH(righthalf)
41                  mergelist(k) ← righthalf[j]
42                  j ← j + 1
43                  k ← k + 1
44              ENDWHILE
45          ENDIF
46      ENDPROCEDURE
```

(a) The following array of numbers is to be sorted using `mergesort`:

`mergelist = [2, 4, 2, 8, 2, 8, 9, 1, 3]`

What are the first two lists to be merged? [1]

[2], [4]

(b) Explain what a logic error is, give the line number for the logic error in the above code, and rewrite the line correctly. [2]

- a **mistake in program design** i.e. instructing the computer to do the wrong thing. **cannot usually be detected by the compiler/interpreter** eg use of incorrect formula (a+b instead of a-b), incorrect branch, else matched with wrong if
- Either
    - Line 24: `if lefthalf[i] > righthalf[j] THEN` should be `if lefthalf[i] < righthalf[j] THEN`
- Or
    - Line 14’: `right_len` should be `mid` else it will miss out on adding the `mid+1` element to the right array.
    - Note: Line 14 would also lead to a runtime error (list out of range) when implemented. Changing line 13 to `-1 TO right_len - 2` won’t work for lists with even number of elements.

The procedure `sorting_proc` uses an optimised bubble sort to sort an array `input_array` in an ascending order. It is used within `modified_mergesort` which is a modified version of `mergesort`.

```
PROCEDURE sorting_proc(input_array : ARRAY)

    length ← LENGTH(input_array)

    REPEAT
        swapped ← FALSE

        FOR curr_elem_index ← 1 to length – 1

            IF  input_array [curr_elem_index - 1] > input_array [curr_elem_index] THEN
                SWAP (input_array [curr_elem_index - 1], input_array [curr_elem_index])
                swapped ← TRUE
            ENDIF

        ENDFOR

        length ← length – 1

    UNTIL NOT swapped

ENDPROCEDURE
```

```
PROCEDURE modified_mergesort(mergelist : ARRAY)

    IF LENGTH(mergelist) > 1 THEN

        IF LENGTH(mergelist) < 5 THEN
            sorting_proc(mergelist)
            RETURN

        ELSE

            mid ← LENGTH(mergelist) DIV 2

            FOR index ← 0 TO (mid - 1)
                lefthalf[index] ← mergelist[index]
            NEXT index

            right_len ← LENGTH(mergelist) - mid

            FOR index ← 0  TO (right_len - 1)
                righthalf[index] ← mergelist[right_len + index]
            NEXT index

            mergesort(lefthalf)
            mergesort(righthalf)

            i ← 0
            j ← 0
            k ← 0
            WHILE i < LENGTH(lefthalf) AND j < LENGTH(righthalf)
                IF lefthalf[i] > righthalf[j] THEN
```

```
                    mergelist[k] ← lefthalf[i]
                    i ← i + 1
                ELSE
                    mergelist[k] ← righthalf[j]
                    j ← j + 1
                ENDIF
                k ← k + 1
            ENDWHILE

            WHILE i < LENGTH(lefthalf)
                mergelist[k] ← lefthalf[i]
                i ← i + 1
                k ← k + 1
            ENDWHILE

            WHILE j < LENGTH(righthalf)
                mergelist(k] ← righthalf[j]
                j ← j + 1
                k ← k + 1
            ENDWHILE
        ENDIF
    ENDIF
ENDPROCEDURE
```

(c) Would the above modification of `mergesort` improve the algorithm's overall efficiency? Support your answer with a description on how and explanation on why its efficiency is affected. [4]

1. Any 4 from below
    a. **Yes**, it would.
    b. Instead of repeatedly dividing the original array into subarrays until it reaches a subarray of 1 element, it **stops dividing when the subarray is less than 5 elements[**
    c. Which is considered
        i. **small data size** and
        ii. are **more likely to be already sorted**

Which are scenarios **more suitable to be sorted by the insertion sort** of `sorting_func` than merge sort

The procedure `insertionSort` is an algorithm which uses insertion sort.

```
PROCEDURE insertionSort(input_array: ARRAY)

    current_elem_index ← 0

    REPEAT
        current_elem_index ← current_elem_index + 1
        compared_item_index ← -1
        swapped ← FALSE

        REPEAT
            compared_item_index ← compared_item_index + 1

            IF input_array[current_elem_index] <
        input_array[compared_item_index] THEN
```

```
14
15                      temp ← input_array[current_elem_index]
16
17                      the value of each element of input_array from
                   compared_item_index to (current_elem_index – 1) is
                   sequentially assigned to each element of input_array
                   from (compared_item_index + 1) to current_elem_index
18
19                      input_array[compared_item_index] ← temp
20
21                      swapped ← TRUE
22                 ENDIF
23
24            UNTIL swapped ← TRUE
25
26       UNTIL current_elem_index = LENGTH(input_array) - 1
27
28  ENDPROCEDURE
```

(d) Modify `insertionSort` and `sorting_proc` to count and store the number of comparisons made in a variable named `comparisons`. Instead of copying all the pseudocode statements, state the line number(s) you want to modify or insert any pseudocode at, followed by the pseudocode statement(s) to be added/modified. [3]

```
PROCEDURE insertionSort(input_array: ARRAY)
     comparisons ← 0                 [1] for both declarations
     current_elem_index ← 0


     REPEAT
          current_elem_index ← current_elem_index + 1
          compared_item_index ← -1
          swapped ← FALSE

          REPEAT

               compared_item_index ← compared_item_index + 1
               comparisons ← comparisons + 1                 [1]
               IF input_array[current_elem_index] <
          input_array[compared_item_index] THEN

                    input_array[compared_item_index] ←
               input_array[current_elem_index]

                    input_array[compared_item_index+1:current_elem_i
               ndex+1] ←
               input_array[compared_item_index:current_elem_index]

                    swapped ← TRUE

               ENDIF

               compared_item_index ← compared_item_index + 1

          UNTIL swapped ← TRUE
```

```
    UNTIL current_elem_index = LENGTH(input_array) - 1

ENDPROCEDURE


PROCEDURE bubbleSort(input_array : ARRAY)
    arr_length ← LENGTH(input_array)
    comparisons ← 0

    REPEAT
        swapped ← false

        FOR curr_elem_index ← 1 TO arr_length - 1
            comparisons ← comparisons + 1                [1]

            IF input_array[curr_elem_index - 1] >
        input_array[curr_elem_index] THEN

                SWAP (input_array[curr_elem_index - 1],
            input_array[curr_elem_index])
                swapped ← TRUE

            ENDIF

        ENDFOR

        arr_length ← arr_length - 1

    UNTIL NOT swapped

ENDPROCEDURE
```

(e) Trace the modified algorithms `insertionSort` and `sorting_proc` for the array `[5, 2, 3, 4]` showing the value of all variables for each step by completing the following tables.

Trace table for `insertionSort`:

current_elem_index	compared_item_index	comparisons	input_array	swapped
1	0	1	[2,5,3,4]	TRUE
2	0	2	[2,5,3,4]	FALSE
...	...	...	...	...

Trace table for `sorting_proc`:

curr_elem_index	comparisons	input_array	arr_length	swapped

1	1	[2,3,5,4]	4	TRUE
2	2	[2,3,5,4]	4	TRUE
…	…	…	…	…

[7]

insertionSort

current_elem_index	compared_item_index	comparisons	input_array	swapd
1	0	1	[2,5,3,4]	TRUE
2	0	2	[2,5,3,4]	FALS
2	1	3	[2,3,5,4]	TRUE
3	0	4	[2,3,5,4]	FALS
3	1	5	[2,3,5,4]	FALS
3	2	6	[2,3,4,5]	TRUE

sorting_func

curr_elem_index	comparisons	input_array	arr_length	swappe
1	1	[5,2,3,4]	4	TRUE
2	2	[2,3,5,4]	4	TRUE
3	3	[2,3,4,5]	4	TRUE
			3	FALSE
1	4	[2,3,4,5]	3	FALSE
2	5	[2,3,4,5]	3	FALSE
			2	FALSE

(f) In the context of mergesort, suggest scenario(s) where using the current optimised bubble sort algorithm for sorting_proc would be better than using the insertionSort algorithm above. Support your answer by designing 3 test cases (normal and boundary) and comparing the number of comparisons made by each algorithm for each test case. Display your output. [7]

TEST 1 (Normal)
for Mostly/partial sorted list [5,2,3,4] [1]
Bubble sort comparisons: 5
[2, 3, 4, 5]
Insertion sort comparisons: 6

```
[2, 3, 4, 5]
```

[1]

TEST 2 (Boundary)
for Already sorted list [2,3,4,5] [1]

```
Bubble sort comparisons: 3
[2, 3, 4, 5]
Insertion sort comparisons: 6
[2, 3, 4, 5]
```

[1]

TEST 3 (Boundary)
for Reverse sorted list [5,4,3,2] [1]

```
Bubble sort comparisons: 6
[2, 3, 4, 5]
Insertion sort comparisons: 6
[2, 3, 4, 5]
```

[1]

Only better when data is mostly sorted or already sorted [1]

Note: Accept answers for empty list and single item list; The bubble sort algorithm is better because it works whereas the insertion sort algorithm produces list out of range error.