

SOLUTIONS

ALGORITHMICS UNIT 3 & 4

Trial Exam 2: 2021

Reading Time: 15 minutes
Writing time: 120 minutes (2 hours)

QUESTION AND ANSWER BOOK

<i>Section</i>	<i>Number of questions</i>	<i>Number of questions to be answered</i>	<i>Number of marks</i>
A	20	20	20
B	8		80

- Students are permitted to bring into the examination room: pens, pencils, highlighters, erasers, sharpeners, rulers and one scientific calculator.
- Students are NOT permitted to bring into the examination room: blank sheets of paper and/or correction fluid/tape

Materials supplied

- Question and answer book of 25 pages
- Answer sheet for multiple-choice questions

Instructions

- Write your student number in the space provided above on this page.
- Check that your name and student number as printed on your answer sheet for multiple-choice questions are correct, and sign your name in the space provided to verify this.
- All written responses must be in English, point form is preferred.

Students are NOT permitted to bring mobile phones and/or any other unauthorised electronic devices into the test room.

The VCAA Exam will include the Master Theorem in this form.

Use the Master Theorem to solve recurrence relations of the form shown below.

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + kn^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases} \quad \text{where } a > 0, b > 1, c \geq 0, d \geq 0, k > 0$$

$$\text{and its solution } T(n) = \begin{cases} O(n^c) & \text{if } \log_b a < c \\ O(n^c \log n) & \text{if } \log_b a = c \\ O(n^{\log_b a}) & \text{if } \log_b a > c \end{cases}$$

The VCAA form of Master Theorem is equivalent to the form of Master Theorem taught in our class by consideration of log laws.

$$\log_b a = c \Leftrightarrow a = b^c \Leftrightarrow \frac{a}{b^c} = 1$$

$$\log_b a < c \Leftrightarrow a < b^c \Leftrightarrow \frac{a}{b^c} < 1$$

$$\log_b a > c \Leftrightarrow a > b^c \Leftrightarrow \frac{a}{b^c} > 1$$

$$T(n) = aT\left(\frac{n}{b}\right) + f(n^k)$$

- $\frac{a}{b^k} < 1$ then $O(n^k)$
- $\frac{a}{b^k} = 1$ then $O(n^k \log_b n)$
- $\frac{a}{b^k} > 1$ then $O(n^{\log_b a})$

SECTION A – Multiple Choice – select one option only

Question 1

Scratch	Python
	<pre>shopping=[] shopping.append('milk') shopping.append('bread') shopping.append('butter') shopping.append('eggs') shopping.sort() print('quinoa' in shopping) shopping.remove('eggs') shopping[shopping.index('butter')]='low fat spread' print (len(shopping))</pre>

What is the printed output from these ADT operations shown above in Scratch and in Python?

<p>A. false, 3</p> <p>B. quinoa, 3</p> <p>C. false, 4</p> <p>D. true, 3</p>	<p>Answer is A</p> <p>Print('quinoa' in shopping) prints False</p> <p>Print(len(shopping)) prints 3</p>
---	---

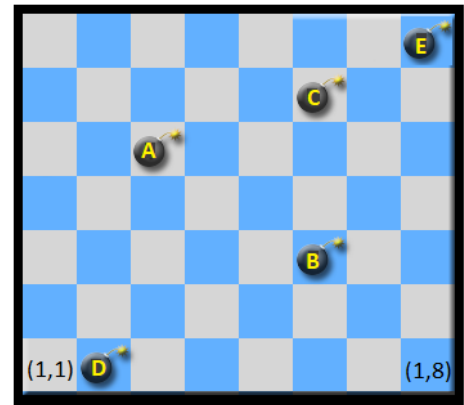
Question 2

<pre>name Dictionary; import key, value; ops newDictionary : →dictionary; insertDictionary : key × value × dictionary → dictionary; removeDictionary : key × dictionary → dictionary; lookupDictionary : key × dictionary →value;</pre>	
<p>The missing terms for defining the Dictionary ADT in order are:</p>	
<p>A. Dictionary, value, value</p> <p>B. Dictionary, key, key</p> <p>C. Dictionary, key, dictionary</p> <p>D. Dictionary, key, value</p>	<p>Answer is D</p>

Question 3

The following algorithm Countit counts how many bombs on a game board of n by m cells, as shown in the diagram, where cell (1,1) and cell (1,8) are shown on the **same row**.

```
Algorithm Countit(n,m)
Set numBombs to 0
For row = 1 to n
  For column = 1 to m
    If (gameBoard(row, column) is a bomb) then
      Add 1 to numBombs
    End if
  End for
End for
Print numBombs
End Algorithm
```



In what order will the bombs be discovered by **Countit(7,8)**?

- A. E, C, A, B, D
- B. D, B, A, C, E
- C. D, A, B, C, E
- D. A, B, C, D, E

Answer is B

Question 4

The time complexity of the Countit above in Question 3 is best described by:

- A. $O(n)$
- B. $O(n+m)$
- C. $O(nm)$
- D. $O(n^2)$

Answer is C

Inner loop runs m times

Question 5

The Halting problem is an example of:

- A. A decidable problem
- B. An undecidable problem
- C. A complete problem
- D. A trackable problem

Answer is B

Question 6

Dijkstra's algorithm is based on which design pattern?

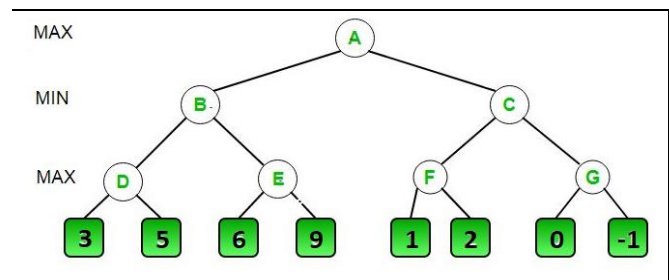
- A. Greedy design pattern
- B. Backtracking design pattern
- C. Dynamic Programming design pattern
- D. Divide and Conquer design pattern

Answer : A
Dijkstra relates to the greedy approach since we select the node with the shortest distance from the set of unvisited nodes.

Question 7

Running the minimax algorithm on the game tree shown at the right will result in a score for the maximising player at node A of:

- A. 6
- B. 9
- C. 8
- D. 5



Answer is D

Question 8

Consider the following statements:

Statement 1: The basic idea of dynamic programming considers all possible cases and is essentially the opposite of a greedy strategy.

Statement 2: When dynamic programming is applied to a problem, it takes far less time as compared to other methods that don't take advantage of overlapping sub-problems.

Then

- A. Only Statement 1 is true
- B. Only Statement 2 is true
- C. Both Statements are true
- D. Both Statements are false

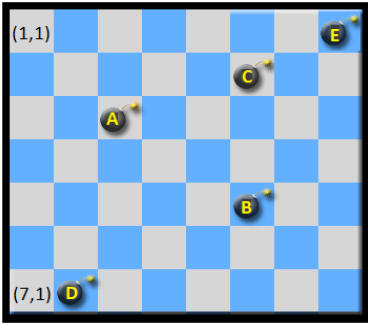

The answer is C

The following algorithm CountBombs is to be used to answer Question 9 and Question 10

```

Algorithm CountBombs(Board,n,m)
// Input Board grid of n rows and m columns
New Array cell:=[1,1] //[row,col]
NewQueue Q
Enqueue cell to Q
While Q is not empty do
  Set cell to Dequeue Q
  row := cell[1]
  col := cell[2]
  Set Board[row,col] as visited
  If ((row+1)≤n) AND NOT (Board[row+1,col] visited) then
    cell := [row+1,col]
    Enqueue cell to Q
  End if
  If (((row+1)≤n) AND ((col+1)≤m) AND NOT (Board[row+1,col+1] visited) then
    cell := [row+1,col+1]
    Enqueue cell to Q
  End if
  If ((col+1)≤m) AND NOT (Board[row,col+1] visited) then
    cell := [row,col+1]
    Enqueue cell to Q
  End if
End do
End Algorithm
  
```

Question 9

<p>Consider the game board shown of rows by columns, cell (1,1) and cell (7,1) are shown in the same column. Several bombs labelled A, B, C, D, E are located on the game board in distinct cells of the board.</p> <p>In what order will the bombs be discovered by the algorithm defined above Algorithm CountBombs(Board,7,8)?</p>	
<p>A. A, B, C, D, E</p> <p>B. E, C, A, B, D</p> <p>C. D, A, B, C, E</p> <p>D. E, C, B, A, D</p>	 <p style="text-align: right; color: red; font-weight: bold;">Answer is A</p>

Question 10

<p>Which graph traversal algorithm does CountBombs most closely follow?</p>	
<p>A. Depth First Search</p> <p>B. Best First Search</p> <p>C. Random Search</p> <p>D. Breadth First Search</p>	<p style="color: red; font-weight: bold;">Answer is D</p>

Question 11

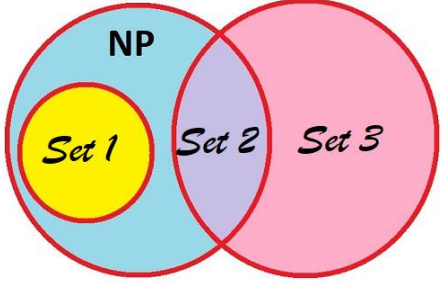
Consider the following recursive algorithm for Depth First Search defined in pseudocode as Function DFS below.

First of all, the visited array is initialised with false values. The use of the visited array determines which nodes have been visited to prevent the algorithm from visiting the same node more than once.


Which are the missing parts of the Function DFS?

<p>Algorithm 1: Recursive DFS</p> <hr/> <p>Data: G: The graph root: The starting node</p> <p>Result: Prints all nodes inside the graph in the <i>DFS</i> order <i>visited</i> \leftarrow {false}; DFS(<i>root</i>);</p> <p>Function <i>DFS</i>(<i>u</i>): if <i>visited</i>[<i>u</i>] = <input type="checkbox"/> then return; end print(<i>u</i>); <i>visited</i>[<i>u</i>] \leftarrow true; for <i>v</i> \in <i>G</i>[<i>u</i>].neighbors() do <input type="text"/> end end</p>	<p>Solution</p> <p>Function <i>DFS</i>(<i>u</i>): if <i>visited</i>[<i>u</i>] = true then return; end print(<i>u</i>); <i>visited</i>[<i>u</i>] \leftarrow true; for <i>v</i> \in <i>G</i>[<i>u</i>].neighbors() do DFS(<i>v</i>); end end</p> <p>Answer is A</p>
<p>A. true, DFS(<i>v</i>)</p> <p>B. false, DFS(<i>u</i>)</p> <p>C. true, DFS(<i>u</i>)</p> <p>D. false, DFS(<i>v</i>)</p>	

Question 12

<p>A partial diagram of the time complexity classes relationship is shown.</p> <p>Set 1, Set 2 and Set 3 in consecutive order are:</p>	
<p>A. NP-Complete, NP, NP-Hard</p> <p>B. NP-Complete, NP-Hard, P</p> <p>C. P, NP-Hard, NP-Complete</p> <p>D. P, NP-Complete, NP-Hard</p>	<p>Answer is D</p>

Question 13

<p>The problem of opening the lock shown at the right in the diagram belongs to the complexity class:</p>	
<p>A. P B. NP C. NP-Hard D. Undecidable</p>	<p>Answer is B</p>

Question 14

Which of the following best describes the position of Strong AI?

<p>A. The principal value of computers is that they are powerful tools for studying the mind B. Having a mind is a matter of having the right outputs C. Computers cannot be minds D. An appropriately programmed computer is a mind, in the sense that it can understand</p>	<p>Answer is D</p>
---	--------------------

Question 15

Which of the following best describes the “Systems reply” to Searle's thought experiment?

<p>A. We only attribute understanding to people because of their behaviour, so we should for machines too B. While the individual in the room doesn't understand the story, the system she's a part of does C. There would be understanding if we put the system into a mechanism that walked around, perceiving D. None of the above</p>	<p>The Answer is B</p>
---	------------------------

Question 16

Consider the following algorithm X, what is the time complexity?

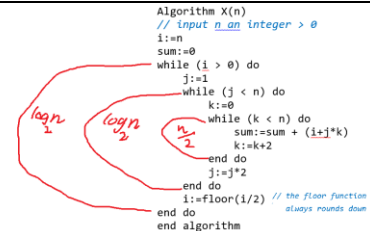
```

Algorithm X(n)
// input n an integer > 0
i:=n
sum:=0
while (i > 0) do
  j:=1
  while (j < n) do
    k:=0
    while (k < n) do
      sum:=sum + (i+j*k)
      k:=k+2
    end do
    j:=j*2
  end do
  i:=floor(i/2) // the floor function always rounds down
end do
end algorithm

```

- A. $O(n^3)$
- B. $O(n \log n)$
- C. $O(n^2 \log n)$
- D. $O(n[\log n]^2)$

Answer is D



Question 17

The time complexity of a recursive algorithm with the number of actions based on the input size of n is given by $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$, $T(1) = 1$. What is the time complexity represented as a function of n ?

- A. $O(n)$
- B. $O(n^2)$
- C. $O(n \log n)$
- D. $O(n^2 \log n)$

Answer C:
 $O(n \log n)$

Question 18

Let X be a problem that belongs to the class NP. Then which one of the following is TRUE?

- A. There is no polynomial time algorithm for X .
- B. If X can be solved deterministically in polynomial time, then $P=NP$.
- C. If X is NP-Hard, then it is NP-Complete
- D. X may be undecidable

Answer is C

Question 19

Let G be a complete undirected graph on 6 nodes. If the nodes of G are labelled, then the number of distinct cycles of length 4 in G is equal to:

A. 15	Answer is C	(a, c, b, d,a)
B. 30	Explanation: There can be total 6C_4 ways to pick 4 nodes from 6. The value of 6C_4 is 15.	(a, c, d, b,a)
C. 45	Note that the given graph is complete so any 4 vertices can form a cycle.	(a, d, b, c,a)
D. 360	There can be 6 different cycle with 4 nodes.	(a, d, c, b,a)
	For example, consider 4 vertices as a, b, c and d. The three distinct cycles are	and
	cycles should be like this	(a, b, c, d,a) and (a, d, c, b,a)
	(a, b, c, d,a)	(a, b, d, c,a) and (a, c, d, b,a)
	(a, b, d, c,a)	(a, c, b, d,a) and (a, d, b, c,a)
		So total number of distinct cycles is $(15*3) = 45$.

Question 20

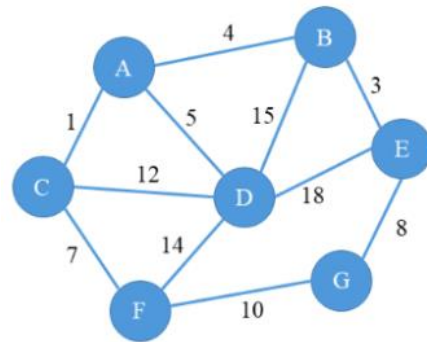
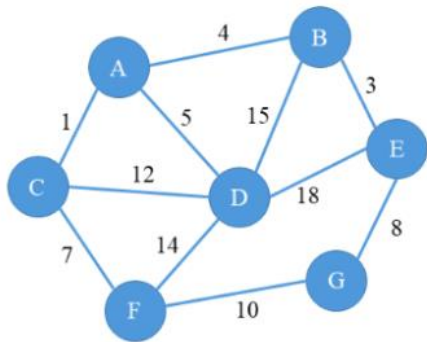
What does the following **mystery** algorithm defined in pseudocode below print when called as **mystery(3,4,5)**?

<pre> Algorithm mystery(n,a,b) // Input n an integer // input a an integer // input b an integer if (n > 0) then mystery(n-1, a, b+n) // 1st recursion print n,a,b //prints out variables mystery(n-1, b, a+n) // 2nd recursion end if end algorithm </pre>		<p>Answer is A,</p> <p>Using a call Trace eliminates many options early</p> <p>Mystery1(3,4,5)</p> <p> Mystery1(2,4,8)</p> <p> Mystery1(1,4,10)</p> <p> Prints 1,4,10</p> <p> Mystery2(0,10,5)</p> <p> Prints 2,4,8</p> <p> Mystery2(2,8,6)</p> <p> Mystery1(1,8,7)</p> <p> Prints 1,8,6</p> <p> Mystery2(0,.....</p> <p> Mystery(1,7,9)</p> <p> Prints 3,4,5</p> <p> Mystery2(2,5,7)</p> <p> Mystery(1,5,9)</p> <p> Prints 1,5,9</p> <p> Mystery2(0,.....</p> <p> Prints 2,5,7 ???</p> <p> Mystery2(2,7,7)</p> <p> Mystery(1,7,8)</p> <p> Prints(1,7,7)</p>
<p>A</p> <pre> 1 4 10 2 4 8 1 8 6 3 4 5 1 5 9 2 5 7 1 7 7 </pre>	<p>C</p> <pre> 3 4 5 1 4 10 2 4 8 1 8 6 1 5 9 2 5 7 1 7 7 </pre>	
<p>B</p> <pre> 1 4 10 2 4 8 1 8 6 3 4 5 </pre>	<p>D</p> <pre> 3 4 5 1 5 9 2 5 7 1 7 7 </pre>	

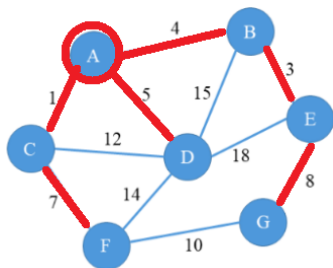
SECTION B – Extended Response Questions Answer all questions in the space provided.

Question 1 (10 marks)

Consider the graph below shown in duplicate to answer parts a) and b)

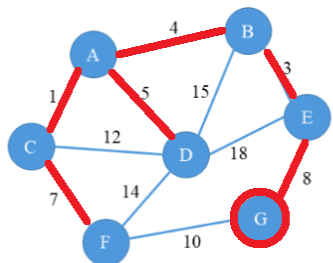


- a) List the order of edges that are added to the minimum spanning tree that Prim's algorithm would find if it used vertex A as the starting node. (2 marks)



- A-C
- A-B
- B-E
- A-D
- C-F
- E-G

- b) List the order of edges that are added to the minimum spanning tree that Prim's algorithm would find if it used vertex G as the starting node. (2 marks)



- G-E
- E-B
- B-A
- A-C
- A-D
- C-F

- c) In what circumstances is the minimum spanning tree of any undirected weighted graph unique? Justify your claims using a logical contradiction argument. (3 marks)

If all the weights on the edges are distinct then the minimum spanning tree will be unique.
 Say by contradiction we have a graph G with distinct edge weights with two different minimum spanning trees T and T'. If T is different to T' then T has an edge that T' omits.
 If we cut T into two subtree sets S and T-S, the cheapest possible distinct weighted edge e that does not make a cycle will connect S and T-S, where the nodes in S+T-S are equal to the nodes in G.
 If we cut T' into the same two subtree sets S and T'-S, where the nodes in S+T'-S are equal to the nodes in G. The cheapest possible distinct weighted edge e that does not make a cycle will connect S and T'-S, which is the same edge e as for S and T-S.
 Hence T and T' have identical edges and are the same MST, a contradiction.

Question 1 (Continued)

d) In what circumstances are there multiple minimum spanning trees in an undirected weighted graph? Justify your claims. (2 marks)

If there are edges in cycles of a graph with the same weight then the minimum spanning tree will not be unique, as the cut property of a MST T into subtrees S and T-S may have several possible choices for connecting the two subtrees together if there are many edges with the same weights.

e) What changes can be made to the graph above to make it have a non-unique minimum spanning tree? (1 mark)

Changing C-D, B-D, E-D, F-D to have the weight of 5 will give multiple possible spanning trees.

Question 2 (6 marks)

A directed graph of 4 nodes and 6 directed edges is shown in Figure 1.

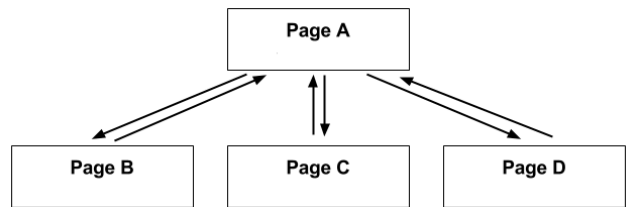
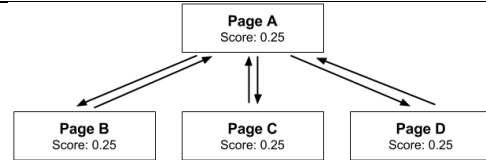


Figure 1

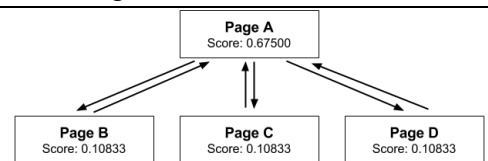
a) Calculate the Page ranks after initialisation for the directed graph in Figure 1. (1 mark)

Page A 0.25
Page B 0.25
Page C 0.25
Page D 0.25



b) Calculate the Page ranks for Page A and Page D after iteration 1 for Figure 1. (2 marks)

- Page A new rank: $\underbrace{d \cdot 0.25/1}_{\text{from page B}} + \underbrace{d \cdot 0.25/1}_{\text{from page C}} + \underbrace{d \cdot 0.25/1}_{\text{from page D}} + \underbrace{(1 - d)/4}_{\text{new surfers}} = 0.67500$
- Page B new rank: $\underbrace{d \cdot 0.25/3}_{\text{from page A}} + \underbrace{(1 - d)/4}_{\text{new surfers}} = 0.10833$
- Page C new rank: $\underbrace{d \cdot 0.25/3}_{\text{from page A}} + \underbrace{(1 - d)/4}_{\text{new surfers}} = 0.10833$
- Page D new rank: $\underbrace{d \cdot 0.25/3}_{\text{from page A}} + \underbrace{(1 - d)/4}_{\text{new surfers}} = 0.10833$



Question 2 (Continued)

The navigation *is changed* for Page D in the directed graph as shown in Figure 2.

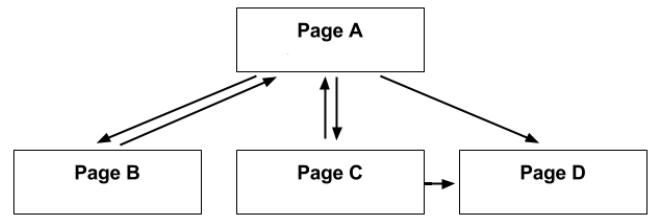


Figure 2

c) Explain how this change affects the Page rank calculations for Figure 2. (1 mark)

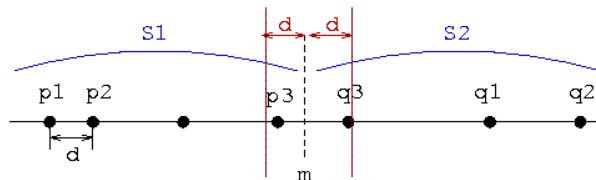
Page D becomes a sink node, and its importance needs to be re-distributed evenly to every other page and itself.

d) Show the Page Rank recurrence relation for Page A and Page B for the new graph shown in Figure 2 (2 marks)

$\Pr(A_{i+1}) = \frac{1-d}{N} + d \left(\frac{\Pr(A_i)}{L(A)} + \frac{\Pr(B_i)}{L(B)} + \frac{\Pr(C_i)}{L(C)} + \frac{\Pr(D_i)}{L(D)} \right)$ $\Pr(A_{i+1}) = \frac{0.15}{4} + 0.85 \left(\frac{\Pr(B_i)}{1} + \frac{\Pr(C_i)}{2} + \frac{\Pr(D_i)}{4} \right)$ $\Pr(D_{i+1}) = \frac{1-d}{N} + d \left(\frac{\Pr(A_i)}{L(A)} + \frac{\Pr(B_i)}{L(B)} + \frac{\Pr(C_i)}{L(C)} + \frac{\Pr(D_i)}{L(D)} \right)$ $\Pr(D_{i+1}) = \frac{0.15}{4} + 0.85 \left(\frac{\Pr(A_i)}{3} + \frac{\Pr(C_i)}{2} + \frac{\Pr(D_i)}{4} \right)$	
---	--

Question 3 (12 marks)

Consider a set of points S on a number line, our goal is to determine which two of these points are minimally distant from each other.



a) Outline a Naïve Brute Force approach for solving this problem. What would be the time complexity of the Naïve Brute Force approach? (2 marks)

A Naïve approach would be $n(n-1)$ comparisons, as each point is compared to every other point ($n-1$) comparisons, this is repeated for each of the n points. Time complexity of this Brute Force approach is $O(n^2)$

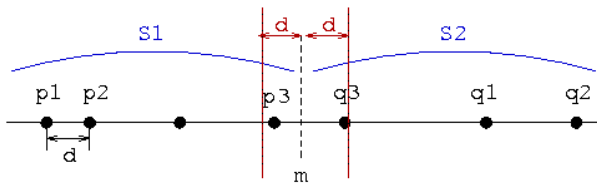
b) What design pattern could be used to improve the time complexity of a Brute Force approach for this problem? (HINT: Is given in the diagram above. Justify your response, and state the improved time complexity. (3 marks)

Divide and Conquer recursively partition this set into two sets by some point m that splits the data into two sets S_1 is to the left of m and S_2 to the right, where the number of points in S_1 is approximately equal to the number of points in S_2 . The closest points will be either in the left partition $\{p_1, p_2\}$ or the right partition $\{q_1, q_2\}$ or the values closest to the median on either side, these solutions were calculated using our recursive solution to the smaller sub-problem. The Divide and Conquer design pattern will avoid comparing each point to every other and will require fewer actions. The recurrence would be $T(n) = 2T(\frac{n}{2}) + O(n)$, where $T(1) = 1$, resulting in $T(n) = O(n \log n)$ time complexity.

Question 3 (continued)

- c) Outline an algorithm in plain English for solving this problem using the design pattern you have selected in part b) (3 marks)

Recursively partition the number line into two sets by some point m the median value. Set S_1 is to the left of m and S_2 to the right. Assume that $\{p_1, p_2\}$ are the points defining the closest pair in S_1 , and that $\{q_1, q_2\}$ are the points defining the closest pair in S_2 , these solutions were calculated using our recursive solution to the smaller sub-problem. Assume that $\{p_1, p_2\}$ are the points defining the closest pair in S_1 , and that $\{q_1, q_2\}$ are the points defining the closest pair in S_2 , and that points $\{p_3, q_3\}$ are the closest points either side of the median value m , recur to find the smallest of these solutions were calculated using our recursive solution to the smaller sub-problem.



- d) Outline an algorithm in structured pseudocode for solving the closest pair of points 1D problem using the design pattern you have selected in part b) (4 marks)

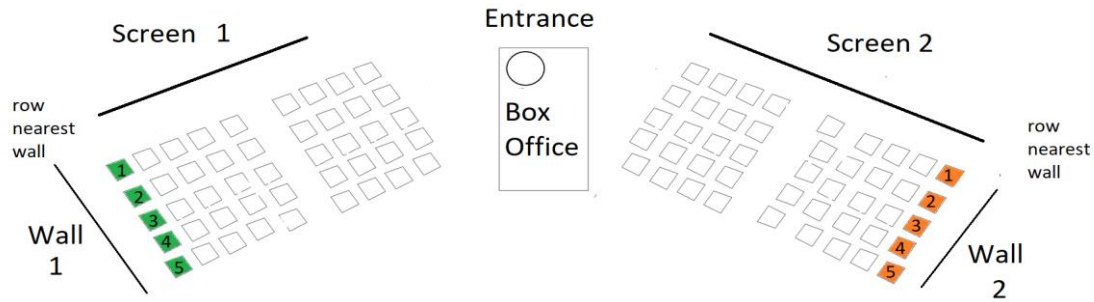
```

Function Closest-Pair(S)
  If |S|=1 then
    Distance = infinity
  Else If |S|=2 then
    Distance = |p2-p1|
  Else
    Let m=median(S)
    Divide S into S1 and S2 at median m
    Distance1 = Closest-Pair(S1)
    Distance2 = Closest-Pair(S2)
    Distance3 = minimum distance across the cut
    Distance=minimum(Distance1, Distance2, Distance3)
  End if
  Return Distance
End Function
  
```

1 mark structured algorithm
 1 mark uses Divide and Conquer across median partition
 1 mark correctly selects closest distance from partitions
 1 reports correct output

Question 4 (14 marks)

Funkytown has a 2 Screen Drive-In where people watch movies on huge screens from their cars.



Car Entry Rules of Funkytown Drive-In (2 Screens, 2 Walls, 2 Parking Bays (each of 5×9 car spots))

- Cars come into the entrance and are served at the box office in a first come first served order for a double session.
- Depending on their movie selection if there are still parking spots available then, each car is directed to the single lane entry for the parking spaces of either Screen 1 or Screen 2 (see diagram above).
- Cars are offered the other movie subject to availability if their first choice is not available, otherwise they are turned away.
- As cars make their way single file into the screen parking area, they must go to the empty rows nearest the wall filling the 5 spots in order from nearest the screen to the back before starting a new row, as shown in the diagram above.

- a) Explain and justify what specific Abstract Data Types ADT(s) can be used to model the parking of vehicles for a double screening session according to the Funkytown Drive-In rules above. (4 marks)

EntranceQ – is a queue, since FIFO processing, dequeues cars and then assigns vehicles to FIFO queues Screen1Q, Screen2Q depending on movie selection

Screen1Q – is a queue for Screen1, since FIFO processing, dequeues a car and assigns it sequentially to an unfilled parking row list of 5 elements **List1**, which is then pushed onto LIFO stack **Wall1S**, then List1 is emptied.

Screen2Q – is a queue for Screen2, since FIFO single line, dequeues a car and assigns it sequentially to an unfilled parking row list of 5 elements **List2**, which is then pushed onto LIFO stack **Wall2S**, then List2 is emptied.

Capacity1 – is variable counter showing available capacity for Screen1

Capacity2 – is a variable counter showing available capacity for Screen2

- b) Show the ADT operations to create, add/remove elements for the ADT(s) you have specified in part a) according to ADT signature definitions. (4 marks)

Create empty queues NewQueue EntranceQ, Screen1Q, Screen2Q, add Queue elements enqueue, remove Queue elements dequeue

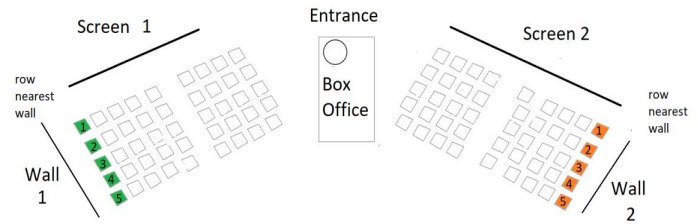
Create empty stacks, NewStack Wall1S, Wall2S, add Stack element push, remove Stack element pop

Create empty Lists, NewList List1, NewListList2, add List element append, first element observe, delete elements

Create variable Capacity1 and Capacity2 initialise them to 45 car spots

Question 4 (continued)

Refer to the Entry Rules of Funkytown Drive-In to answer this question.



- c) Implement the car parking allocation **from the cars already queued at the Entrance gate** following the given rules using a **modular** and well structured algorithm written in pseudocode, using your ADT(s) from parts a) and assuming the ADT(s) from part b) operations have already been created. (6 marks)

```

// EntranceQ contains enqueued cars
// Screen1Q, Screen2Q, Wall1S, Wall2S, List1, List2 are initially empty
// Capacity1=Capacity2=45
Algorithm DriveInArrivalParking(EntranceQ, Open)
  While (Box Office is open for session) AND (EntranceQ is not empty) do
    Dequeue a car from EntranceQ

    If ((Car wants Movie1) AND (Capacity1>0)) then
      DirectCarScreen(Screen1Q, Capacity1)
    Else
      If ((Car wants Movie2) AND (Capacity2>0)) then // Offer other movie if capacity
        DirectCarScreen(Screen2Q, Capacity2)
      Else
        Turn away car
      End if
    End if

    If ((Car wants Movie2) AND (Capacity2>0)) then
      DirectCarScreen(Screen2Q, Capacity2)
    Else
      If ((Car wants Movie1) AND (Capacity1>0)) then // Offer other movie if capacity
        DirectCarScreen(Screen1Q, Capacity1)
      Else
        Turn away car
      End if
    Endif

    While (Screen1Q is not empty OR Screen2Q is not empty) do
      Parking(Screen1Q, Wall1S,List1,5)
      Parking(Screen2Q, Wall2S, List2,5)
    End do
  End do
End Algorithm

```

```

Function DirectCarScreen(Q, C)
  Process a car request and take their money
  C := C - 1
  Enqueue car to Q
End function

```

```

Function Parking(Q, S, L, Limit)
  Dequeue a Car from Q
  Append Car to L
  If (length of L=Limit) then
    Push L onto S
    Empty L
  End if
End Function

```


Question 5 (continued)

Neural Networks computers/algorithms can learn how to read handwritten texts through training.



- d) Describe in your own words how a Neural Network could display “strong AI” and “weak AI” through learning how to read handwritten texts. (2 marks)

Strong AI is shown when the NN understands the handwritten texts and works out their semantic meaning. Weak AI is shown when the NN simply translates the handwritten symbols to another set of symbols without any interpretation.

- e) Briefly describe the “Searle’s Chinese room thought experiment” and outline how it could pass the Turing test. (2 marks)

Searle's thought experiment takes Chinese characters in the form of questions as input into a room and, by following the instructions of a computer program a person or computer in the room, produces other Chinese characters that answer the questions, which it presents as output to the outside of the room. Passing the Turing Test is when the computer performs its task so convincingly that it convinces a human Chinese speaker that the computer is itself a live Chinese speaker, it makes appropriate responses, such that any Chinese speaker would be convinced that they are talking to another Chinese-speaking human being. Failing the Turing Test is when the responses do not convince they are talking to another Chinese speaker.

	<p>Searle concluded that programs/computers/algorithms are neither constitutive of nor sufficient for minds:</p> <ul style="list-style-type: none"> • programs don't have semantics, programs only have syntax, syntax is insufficient for semantics, every mind has semantics, therefore no programs are minds • minds have intentions and “causal powers” which capture the probability of cause and effect, programs only run formal programs and do not have intentions and “causal powers” 	
--	---	--

- f) Explain the Virtual mind replies to Searle’s conclusions and how they address the concept of the mind. What are Searle’s responses to those replies? (2 marks)

Some argue that a computer may contain a “Virtual mind” in the sense that virtual machines such as calculators exist and virtual worlds such as those in gaming exist. They argue that just because the man in the room doesn’t understand Chinese, this does not imply there is nothing in the room that understands Chinese if the output responses provided are convincing.

Searle says that such a virtual mind is, at best, a simulation, as it has no intentions or causal inferences and gives analogy as argument: "No one supposes that computer simulations of a five-alarm fire will burn the neighbourhood down or that a computer simulation of a rainstorm will leave us all drenched”.

Question 6 (6 marks)

A Film Festival has a program of films being screened each marked by a start time (s_i) and finish time (f_i). The problem is to select the maximum number of films that can be viewed by a single person, assuming that a person can only watch one film at a time.

Non-Conflicting films: Let's consider there are N films in a program and for each film, there is a start s time and finish time f for that film: $[s, f]$. Two films i and j are said to be non-conflicting if $s_i \geq f_j$ or $s_j \geq f_i$.



Example 1	Example 2	Example 3
Given Program of Films: $\{[1, 3], [2, 5], [0, 7], [6, 8], [9, 11], [10, 12]\}$ Selected Films: $[1, 3] [6, 8] [9, 11]$	Given Program of Films: $\{[1, 3], [2, 5]\}$ Selected Films: $[1, 3]$	Given Program of Films: $\{[0, 7], [1, 3], [4, 5]\}$ Selected Films: $[1, 3] [4, 5]$

Write a greedy algorithm in commented structured pseudocode that tries to select the maximum number of non-conflicting films that can be viewed by a single person at the Film Festival (assuming that the person can only watch one film at a time and stays until the finish time of a film).

Algorithm GreedyFilmPicker(program)

```

# Input program a list of [start-time,end-time] times for each film
1. Initialise the watchlist
2. Sort all the films in program according to their finish time. #Greedy bit
3. Select the first film in sorted program and set current_finish to the finish time.
4. # Now iterate through the rest of the films.
5. For each film in program
    1. If start time of film > current_finish
        1. add film to watchlist
        2. Update current_finish = finish time of film.
    2. Else
        1. Ignore the film.
6. Report watchlist
7. End Algorithm
    
```

- 1 mark structured algorithm
- 1 mark inputs/ADTS clearly defined
- 1 mark uses a Greedy method supported by sorting/priority queue
- 1 mark uses Greedy logic selection
- 1 mark correctly selects non-overlapping films
- 1 reports correct output

Question 7 (13 marks)

- a) What is a heuristic? How is it used for problem solving? (2 marks)

A short cut or quick polynomial time process for finding an approximate solution when it is not feasible to explore the total solution space for a particular problem.

- b) What is the hill-climbing heuristic? In general terms explain how it works. (2 marks)

A hill-climbing algorithm is a local search algorithm that moves continuously upward, by repetitively checking its immediate neighbourhood for which way to step to go up (a best first search where increasing gradient informs the next step) until the best solution is attained according to constraints.

The process of continuous improvement of the current state of iteration can be termed as climbing. This explains why the algorithm is termed as a hill-climbing algorithm.

- c) What are the main features of a hill-climbing heuristic algorithm? (3 marks)

It employs a greedy approach: This means that it moves in a direction in which the cost function is optimized. The greedy approach enables the algorithm to establish local maxima or minima.

No Backtracking: A hill-climbing algorithm only works on the current state and succeeding states (future). It does not look at the previous states.

Feedback mechanism: The algorithm has a feedback mechanism that helps it decide on the direction of movement (whether up or down the hill). The feedback mechanism is enhanced through the generate-and-test technique.

Incremental change: The algorithm improves the current solution by incremental changes.

- d) What are the problems that can be encountered with hill-climbing heuristic for finding maximums? Explain the possible implications for finding a solution. (2 marks)

There are three regions in which a hill-climbing algorithm cannot attain a global maximum or the optimal solution: local maximum, ridge, and plateau.

- e) Explain how the hill-climbing heuristic is used to solve practical problems such as the optimal Travelling Salesman Problem. (2 marks)

This algorithm is widely used in solving Traveling-Salesman problems, where the solution space is imagined to be a three dimensional surface of choices, at each iteration always head upwards. It can help by optimizing the distance covered and improving the travel time of sales team members. The algorithm helps establish the local minima efficiently in polynomial time.

- f) How would the hill-climbing heuristic need to be altered to solve the decision Travelling Salesman Problem? (2 marks)

Since a threshold is known for making a decision the hill climbing heuristic could stop as soon as that threshold was reached.

Question 8 (10 marks)

- a) Complete the missing parts of the pseudocode below for the Floyd Warshall Shortest Path algorithm. (4 marks)

```
Algorithm Floyd-Warshall(G)
# Input graph G a directed or undirected graph of nodes V(G) and edges E(G)
# The graph G may have negative weighted edges, but no negative weight cycles
# initialise an adjacency matrix distance of |V| rows by |V| columns for graph G
For i = 1 to |V| do
  For j = 1 to |V| do
    If (i equals j) then
      distance[i][i] := 0 # zero distance from node i to itself
    Else If (edge i-j exists) then
      distance[i][j] := edge.weight(i-j) # edge weight from i-j
    Else
      distance[i][j] := ∞ # no direct edge between node i and node j
  For k = 1 to |V| do
    For i =1 to |V| do
      For j = 1 to |V| do
        If distance[i][k] + distance[k][j] < distance[i][j] then
          distance[i][j] := distance[i][k] + distance[k][j]
```

- b) State the time complexity of the Floyd Warshall Shortest Path algorithm with justification. (2 marks)
The triple nested loop in algorithm does the most work and each nested loop runs $|V|$ times therefore $O(n^3)$ where $n=|V|$ in the graph G.
- c) Give a proof of correctness by Contradiction of the Floyd Warshall Shortest Path algorithm, stating any assumptions made. (4 marks)

By contradiction we can state that Floyd Warshall's algorithm does not return the shortest path between two nodes given an input graph G with the assumption that G does not contain directed negative cycles.

The algorithm numbers the nodes 1 to n and initialises a matrix ($n \times n$) of distances $\text{distance}[][]$, when $k=0$ (before the triple nested loop starts) the process of initialisation sets the $\text{distance}[i][i]$ to zero as zero distance between node i and itself, if the edge (i-j) exists in E then the $\text{distance}[i][j]$ set to distance from node i to node j, otherwise there is no edge (i-j) set $\text{distance}[i][j]$ to ∞ , indicating no direct edge exists between node i and j.

Proceed by induction

Base case: $k=1$, the $\text{distance}[][]$ is correctly set up for the shortest path from node i to j using node set $k=\{1\}$
Assume $\text{distance}[][]$ matrix has been correctly updated for $k-1$ iterations for node set $\{1,2,\dots,(k-1)\}$

By induction in the k -th iteration of the triple nested loop, there are two cases:

Case 1: the shortest way from node i to node j with internal nodes from the set $\{1,2,\dots,k\}$ is the same as the shortest path with internal nodes from the set $\{1,2,\dots,(k-1)\}$, in this case $\text{distance}[i][j]$ is unchanged

Case 2: the shortest way with internal vertices from $\{1,2,\dots,k\}$ is shorter, the new shorter path passes through the node k (edges i-k, k-j). The shortest path exists between node i and k, and the path k and j via edge k-j. Shortest distance to nodes i and node j have already been computed with internal nodes from set $\{1,2,\dots,(k-1)\}$ so $\text{distance}[i][j]$ is updated as $\text{distance}[i][k]+\text{distance}[k][j]$ if it is shorter, hence the path is always shortened if possible through node k where edges i-k, k-j exist in G.

As a result when $k=n=|V|$ then the algorithm has iterated over all nodes the value in $\text{distance}[i][j]$ is the shortest path between nodes i and j, or is ∞ if a path between node i and j does not exist.

So $\text{distance}[][]$ holds all shortest path pairs between all nodes, contradicting the original statement.

END OF TRIAL EXAM 2