NAME:

ALGORITHMICS UNIT 3 & 4

Trial Exam 1: 2020
SOLUTIONS

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

QUESTION AND ANSWER BOOK

Section	*Number of questions*	*Number of questions to be answered*	*Number of marks*
A	20	20	20
B	7	7	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 22 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 you 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.

IMPORTANT NOTE:
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

If the actions are completed in sequence for the abstract data type **MQ** as shown below:

1. Create minimum priority queue MQ
2. Enqueue into MQ item=”Blue”, rank=5
3. Enqueue into MQ item=”Red”, rank=3
4. Enqueue into MQ item=”Orange”, rank=2
5. Enqueue into MQ Item=”Green”, rank=4

The contents of **MQ** after these actions could be:

A. Blue, Red, Orange, Green

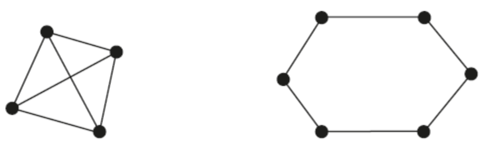
B. Orange, Red, Green, Blue

C. Blue, Green, Orange, Red

D. Blue, Orange, Red, Green

Question 2

Two graphs, labelled Graph 1 and Graph 2, are shown below.



Graph 1 Graph 2

The sum of the degrees of the vertices of Graph 1 is:

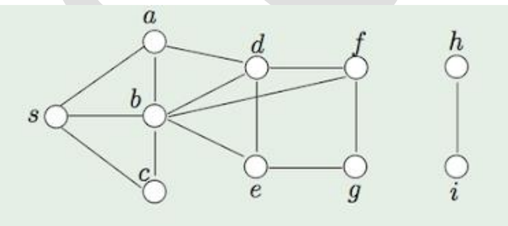
A. More than the sum of the degrees of the vertices of Graph 2

B. Two less than the sum of the degrees of the vertices of Graph 2

C. One less than the sum of the degrees of the vertices of Graph 2

D. Equal to the sum of the degrees of the vertices of Graph 2

Question 3



A Breadth First Search node traversal order starting at **node “s”** for the graph shown above could be:

A. s, c, a, b, e, d, g, f

B. s, a, d, f, g, e, b, c

C. s, a, b, c, d, e, f, g, h, i

D. s, b, e, c, a, d, f, g

Consider the **Algorithm Check** shown in pseudocode below to answer **Question 4** and **Question 5**:

```
Algorithm Check (Items, Key)
// Input: Items, a list of items
// Input: Key a variable
If (there are no items in Items) then
    Return False
Else
    If (the 1st item of Items is equal to Key) then
        Return True
    Else
        Check(all but the first item of Items, Key)
    End if
End if
End Algorithm
```

Question 4

Which is the **equivalent iterative algorithm** for **Algorithm Check** in pseudocode?

A.

```
Algorithm A (Items, Key)
KeyFound:=FALSE
For i=1 to length of Items do
    If (ith item of Items equals Key) then
        KeyFound:=TRUE
    End if
End do
Return KeyFound
End Algorithm
```

B.

```
Algorithm B (Items, Key)
For i=1 to length of Items do
    If (ith item of Items equals Key)
then
        Return i
    End if
End do
End Algorithm
```

C. ☺

```
Algorithm C (Items, Key)
KeyFound:= FALSE
While (there are items in the List) AND
NOT(KeyFound) do
    If (first item of List equals Key) then
        KeyFound:= TRUE
    Else
        Remove first item of Items
    End if
End do
Return KeyFound
End Algorithm
```

D.

```
Algorithm D (Items, Key)
Repeat until (first item of Items equals
Key) do
    If (first item of list equals Key)
        Return first item of list
    Else
        Remove first item of Items
    End if
End Repeat
End Algorithm
```


Question 5

The design pattern of **Algorithm Check** is:

- **A.** Greedy Recursion
- **B.** Transform and Conquer
- **C.** Divide and Conquer
- **D.** Decrease and Conquer

Question 6

Which of the following statements is true?

A. A greedy algorithm can be used to solve the 0-1 knapsack optimization problem.

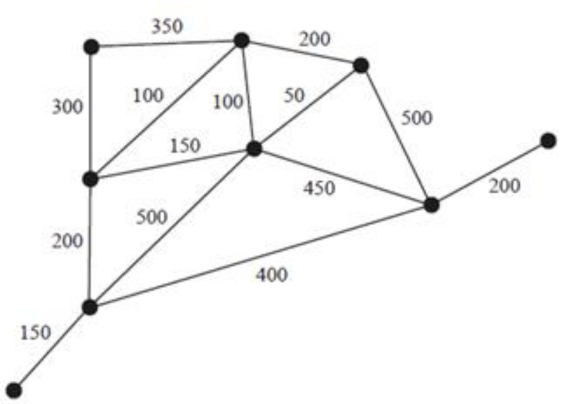
B. Dynamic programming can be used to solve optimization problems where the size of the space of possible solutions is exponentially large.

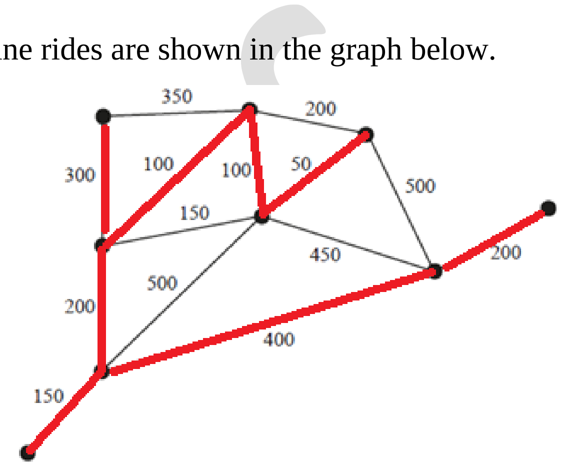
C. Dynamic programming can be used to find an approximate solution to an optimization problem, but cannot be used to find a solution that is guaranteed to be optimal.

D. Decision trees are always binary.

Question 7

A theme park has nine rides and the position of the nine rides are shown in the graph below.





150+200+100+100+50+400+200+300=1500

The numbers on the edges represent the distances in metres between the rides. Electrical cables are required to power the rides. These cables will form a connected graph. The shortest total length of cable that can be used to connect all the rides is:

A. 1500 metres

B. 1550 metres

C. 1250 metres

D. 1350 metres

Question 8

The correct ascending order of the time complexities shown in each option below is:

A.
$O(n^2)$
$O(2^n)$
$O(n!)$
$O(logn)$
$O(n)$

B.
$O(1)$
$O(nlogn)$
$O(logn)$
$O(n)$
$O(n^2)$

C.
$O(1)$
$O(nlogn)$
$O(n)$
$O(n^2)$
$O(2^n)$

D. ☺
$O(logn)$
$O(n)$
$O(nlogn)$
$O(n^2)$
$O(n!)$

Consider the Algorithm Mystery defined in pseudocode.

```
Algorithm Mystery (A, L, H, K)
If (H ≥ L) then
    M:=L + round((H-L)/2)
    If A[M]=K then
        Return M
    Else
        If (K < A[M]) then
            Mystery(A,L,M-1,K)
        Else
            Mystery(A,M+1,H,K)
        End if
    End if
Else
    Return -1
End if
End Algorithm
```

Use the **Algorithm Mystery** shown above to answer **Question 9 Question 10** and **Question 11.**

Question 9

The Algorithm Mystery can most closely be described as having the algorithmic design pattern of:

A. Brute Force

B. Greedy

C. Divide and Conquer

D. Backtracking

Question 10

The time complexity of the Algorithm Mystery is closest to:

A. O(nlogn)

B. O(n^2)

C. O(2^n)

D. O(logn)

Question 11

If the Algorithm Mystery is called as follows:

```
Create Array Hue
Hue[1]=128
Hue[2]=178
Hue[3]=209
Hue[4]=221
Mystery (Hue, 1, 4, 221)
```

The **next** recursive call to Algorithm Mystery will be:

A. `Mystery (Hue, 2, 4, 221)`

B. `Mystery (Hue, 4, 4, 221)`

C. `Mystery (Hue, 3, 4, 221)`

D. `Mystery (Hue, 4, 4, 209)`

Question 12

```
name Digraph;
import node, edge, set;
ops     newDigraph   :   →Digraph;
        insertNode : Digraph × node→ ________
        removeNode : Digraph ×  node →Digraph;
        insertEdge     : Digraph × edge →Digraph;
        removeEdge     : Digraph × ________  →Digraph;
        startNode : edge→node;
        endNode  :  edge→node;
        incident  :  Digraph×node→set;
```

The missing terms to complete the Abstract Data Type signature shown above are:

A. Node, node

B. Digraph, node

C. Node, edge

D. Digraph, edge

Question 12

A big-*O* estimate for the number of operations, where an operation is an addition or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the **while** loop).

$i := 1$
$t := 0$
while $i \leq n$
  $t := t + i$
  $i := 2i$

A. O(logn)

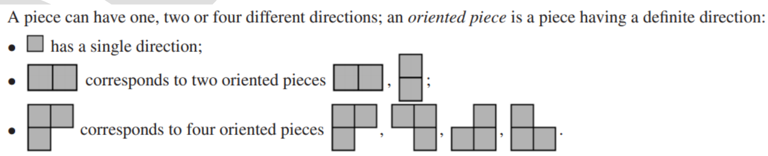
B. O(n)

C. O(n^2)

D. O(nlogn)

Question 13

A Tetris-like game has regular shaped pieces that can be fitted closely (tessellated) together in different orientations to form a flat surface.



The problem of finding a way of tessellating a large irregular flat area such as the map of Australia with regular shaped tetris pieces so that there are **minimal** gaps is classified as:

A. NP

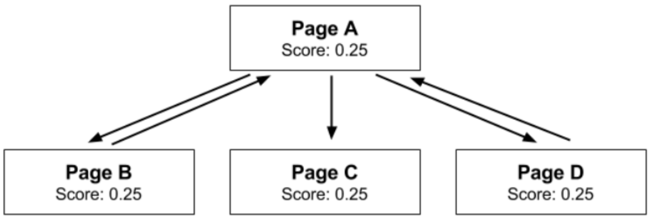
B. P

C. NP-Complete

D. NP-Hard

Question 14

The PageRank Algorithm is run on a directed graph with four nodes A, B, C and D as shown below. After the initialize step runs, each page has an initial rank of 0.25:



If d=0.85 which is the probability of links being followed, then on the next iteration the rank of page B will be found by the following mathematical expression:

A. $d\left(\frac{0.25}{3}\right) + \left(\frac{1-d}{4}\right)$

B. $d\left(\frac{0.25}{1}\right) + d\left(\frac{0.25}{1}\right) + d\left(\frac{0.25}{4}\right) + \left(\frac{1-d}{4}\right)$

C. $d\left(\frac{0.25}{3}\right) + d\left(\frac{0.25}{4}\right) + \left(\frac{1-d}{4}\right)$ next value for Page B: due to redistribution of sink node C

D. $d\left(\frac{0.25}{4}\right) + \left(\frac{1-d}{4}\right)$

Question 15

Which of the following heuristics uses randomised approaches to solve hard problems?

A. Hill Climbing

B. Simulated Annealing

C. Nearest Neighbour

D. Generate and Test

Question 16

If the fastest known algorithm to solve a problem X can run in $O(2^n)$ time, then the problem X is classified as:

A. P class problem

B. Not enough information given.

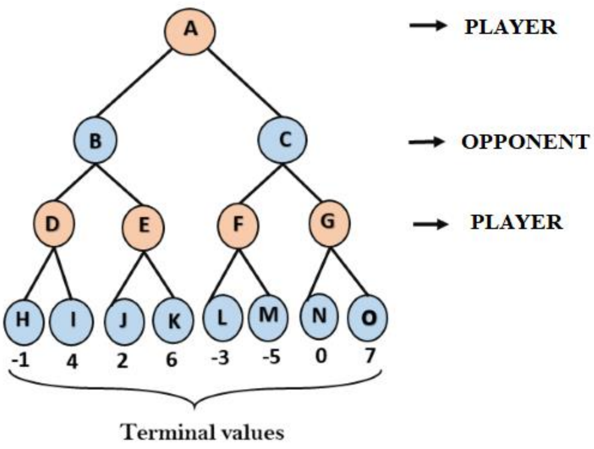
C. NP Complete problem

D. NP problem

Question 17

A game tree of several levels is shown where the PLAYER and OPPONENT levels are indicated.

The values that will be calculated by the Minimax Algorithm for nodes A, B and C will be:



A. Node A=4, Node B=4, Node C=-3

B. Node A=0, Node B=2, Node C=0

C. Node A=7, Node B=6, Node C=7

D. Node A=4, Node B=4, Node C=0

Question 18

Which of the following is **not** an Intractable problem?

A. The decision version of the Travelling Salesman problem in a weighted connected graph.

B. Finding the shortest path between two locations in a weighted connected graph.

C. Finding the cheapest Hamiltonian path in a weighted connected graph.

D. Finding the maximum path length between two locations in a weighted connected graph.

Question 19

The recurrence relation $T(n) = 3T\left(\frac{n}{3}\right) + \sqrt{n}, where\ T(1) = O(1)$ for a recursive algorithm, can be expressed by the following equivalent function:

A. $O(n^3)$

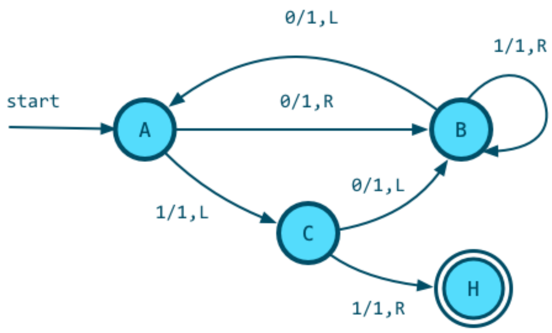
B. $O(\sqrt{n})$

C. O(n)

D. $O(3^n)$

Question 20

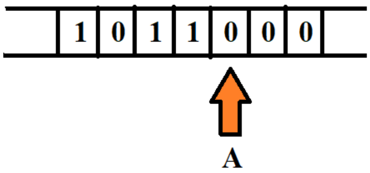
A Turing Machine (TM) of four states A, B, C and H is defined as follows:



TM Symbols: {0,1}
TM Movement:

- L=Left one cell
- R=Right one cell

TM Edge transitions
<Input Symbol>/<Output Symbol>, <Movement>



The **next four** actions for this TM for the memory tape shown above with the current position and state indicated will be:

A. 0/1,L => State A, 1/1,L => State C, 1/1,R => State H, no action

B. 0/1,R => State B, 1/1,L => State B, 1/1,L => State B, 0/1,R => State A

C. 0/1,R => State C, 0/1,L => State B, 1/1,L => State B, 1/1,R => State B

D. 0/1,R => State B, 0/1,L => State A, 1/1,L => State C, 1/1,R => State H

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

Question 1 (15 marks)

a) Complete the missing actions in pseudocode of the following Quicksort Algorithm and the partition module shown below when it is called as `quicksort(A, 0, length(A)-1)`. **(3 marks)**

```
algorithm quicksort(A, low, high)
// input A: an array zero referenced
// input low: an integer index into the Array
// input high: an integer index into the Array
  if low < high then
      p := partition(A, low, high)
      quicksort(A, low, p - 1)
      quicksort(A, p + 1, high)
  end if
end algorithm
```

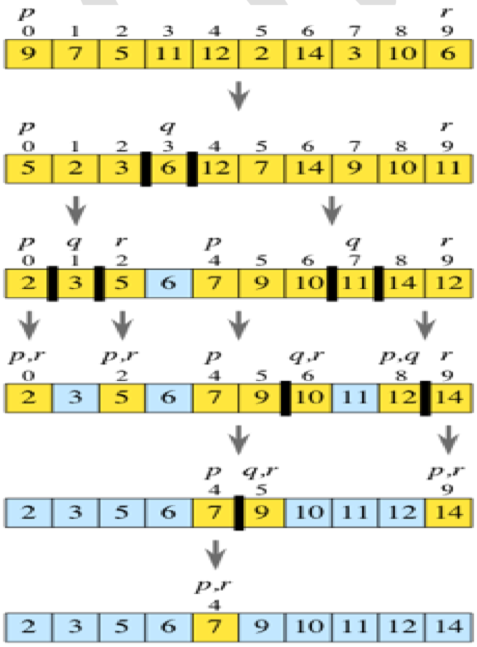
```
module partition(A, low, high)
// input A: an array zero referenced
// input low: an integer index into the Array
// input high: an integer index into the Array
  pivot := A[high]
  i := low
  for j := low to high do
     if A[j] < pivot then
        swap A[i] with A[j]
        i := i + 1
     end if
  end do
  swap A[i] with A[high]
  return i
end module
```

b) What are the main features and design pattern of the Quicksort algorithm? **(2 marks)**

The divide and conquer recursion that places the pivot in the correct place, and then recurs on the lower and upper partition about the pivot.

Use the array of unsorted numeric values [9, 7, 5, 11, 12, 2, 14, 3, 10, 6] to answer part c)

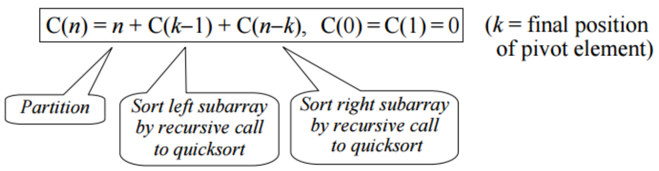
c) If we run Quicksort using the rightmost element as the pivot value, show the partitioning steps that quicksort would take at each level, and indicate the pivot value for each sub-array partition. **(3 marks)**



Question 1 (continued)

d) If we run Quicksort using the rightmost element as the pivot value, what could be the best case and worst case time complexity for execution? Justify your analysis using recurrences or other mathematical constructions and provide examples to justify your conclusions. **(4 marks)**

<table>
<tr><th>Quicksort Worst case</th><th>Quicksort Best case</th></tr>
<tr><td>
If $C(n)$ is the number of comparisons performed by quicksort in sorting an array of size n, partition performs (n-1) comparisons as every element other than the pivot must be compared to the pivot; to determine whether it goes left or right of the pivot.

So our recurrence for C(n) is:


In the worst case at every step, partition splits the array as unequally as possible ($k = 1$ or $k = n$). Then our recurrence becomes
$C(n) = n + C(n-1),\ C(0) = C(1) = 0$

$$\begin{aligned} C(n) &= n + C(n-1) \\ &= n + n-1 + C(n-2) \\ &= n + n-1 + n-2 + C(n-3) \\ &= n + n-1 + n-2 + \ldots + 3 + 2 + \overset{0}{\cancel{C(1)}} \\ &= (n + n-1 + n-2 + \ldots + 3 + 2 + 1) - 1 \\ &= n(n+1)/2 - 1 \\ &\approx n^2/2 \end{aligned}$$
Which is $O(n^2)$
</td><td>
At every step, partition splits the array as equally as possible ($k = (n+1)/2$; the left and right subarrays each have size $(n-1)/2$)).

In this case the recurrence becomes
$$C(n) = n + 2C\left(\frac{(n-1)}{2}\right),$$
$$C(0) = C(1) = 0$$
which we approximate by
$C(n) = n + 2C(n/2), C(1) = 0$

By the Master Theorem or other this can be shown to be: $C(n) = n\log(n)$.

Which is $\Omega(n\log n)$
</td></tr>
</table>

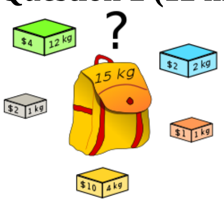
e) What strategies could be used for Quicksort to avoid the worst case time complexity? **(1 mark)**

The pivot value could be selected randomly in each partition to try to avoid splitting 1, (n-1)

f) What are the advantages of using Quicksort compared to Mergesort? Compare the time complexity and space complexity of the two sorting algorithms. **(2 marks)**

Quicksort is an in-place **sorting** algorithm.
In-place **sorting** means no additional storage space is needed to perform **sorting**.
Mergesort requires a temporary array to **merge** the **sorted** arrays and hence it is not in-place giving **Quicksort** the **advantage** of space.

Question 2 (12 marks)



a) Describe the 01 Knapsack problem, and the two main versions of it. **(2 marks)**

- Optimal packing of Knapsack of a limited capacity, given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
- Decision version of Knapsack problem, pack items into Knapsack of limited capacity so that value of items $\geq$ threshold or $\leq$ threshold.

b) What complexity class do the two main versions of 01 Knapsack belong to and why? **(3 marks)**

- Both versions of the problem result in a combinatorial explosion, of possible solutions, if there are n items there will be 2^n combinations that can be selected for the knapsack, which puts the problem in the NP class.
- Optimal packing of Knapsack is NP-Hard as the solution should it be found cannot be verified in polynomial time.
- Decision version of the Knapsack problem is still a difficult problem and is accepted as NP-Complete, since a solution if found can be verified in polynomial time.

c) What are real life applications of each of these two versions of the Knapsack problem? **(2 mark)**

- Optimal: Which shares to invest your limited money to maximise your earnings.
- Decision: Which shares to invest your limited money to earn at least $20000 per year.

d) Outline a brute force approach for finding a solution, outlining the benefits and limitations. **(2 marks)**

- A brute force algorithm for n items can generate and test each of the 2^n combinations that can be put in the knapsack and find the optimal solution.
- This strategy will guarantee a correct optimal solution, however it may not be able to find the solution this way in a reasonable time due to the exponential time complexity of this approach.

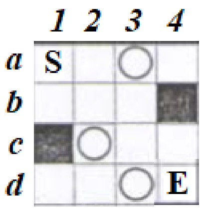
e) What other kind of algorithm(s) or algorithmic approaches can be used to find solutions for these problems? Describe in detail two algorithmic strategies for finding a solution, outlining the benefits and limitations. **(3 marks)**

- Greedy Heuristic approach, work out the value/weight ratio and pick the most favourable first.
- Simulated Heuristic annealing, a combination of greedy with some randomisation of choices that decrease over time.
- Both these heuristics will run in polynomial time and will give an acceptable “good enough” solution in a reasonable time, however the proposed solution may not be the optimal solution.

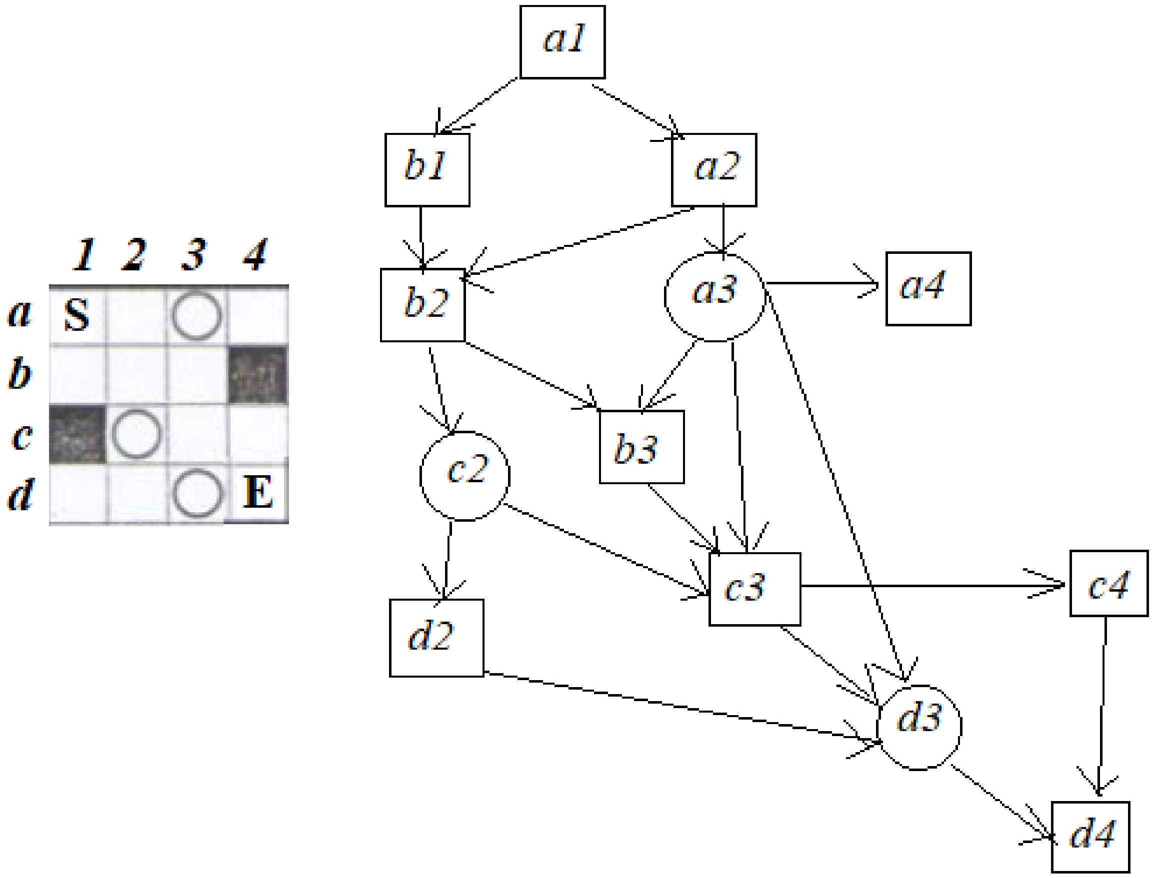
Knapsack problem and heuristics. See page 125 135 of textbook for solutions.

Question 3 (13 marks)

Consider the following board puzzle.

 S=a1 E=d4 Circle={a3, c2, d3} Black={b4,c1}	In this board puzzle there are five types of cells; white, black, circle, S and E. The task is to find a way how to move from the point S (Start) to the point E (End) using the smallest number of steps as possible keeping to the following rules: • You can only move horizontally right or vertically down on the board • If you are on a white cell you can move only 1 cell • If you are on an S you can move 1 cells • If you are on a circle, you can move 1, 2 or 3 cells • You cannot enter nor go through black cells.

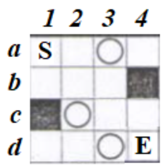
a) Using the cell row (*a*, *b*, *c*, *d*) and column references (*1*, *2*, *3*, *4*) for the board, show how the solution to this puzzle be modelled as using a **graph** Abstract Data Type. **(4 marks)**



b) What are the main attributes of the model you have created in part a)? **(1 marks)**

Graph attributes: directed, unweighted, sparse, low outgoing degree on most nodes.

Question 3 (Continued)



c) For each of the following graph algorithms complete the following table with the main Abstract Data Type (ADT) used by the algorithm in controlling the information of the graph G={V,E}, and its time complexity for a graph G={V,E} in terms of |V| and |E|, and the main actions done. **(5 marks)**

Algorithm for G={V,E}	Main ADT used	Time complexity in terms of |V| and |E|	Main action(s) of algorithm
Dijkstra's Shortest Path	Priority queue	$O(\|V\| + \|E\|\log_2\|V\|)$	While unvisited nodes loop, inside for each outgoing edge consider the neighbours put into a priority queue
Bellman-Ford Shortest Path	Graph itself	$O(\|V\|\|E\|)$	Fixed nested loop outer for each node, inner for each edge
Floyd Warshall Shortest Path	array/matrix	$O(\|V\|^3)$	Fixed triple nested loops for each node
Breadth First Search	queue	$O(\|V\|+\|E\|)$	each node and edge is visited and actioned on once only by storing in a queue
Depth First Search	stack	$O(\|V\|+\|E\|)$	each node and edge is visited and actioned on once only by storing in a stack

d) Considering the algorithms shown in the table in part c), **which of these algorithms** is best to use to find the shortest path from node "S" to node "E" using the graph model from part a)? **Justify** your choice of algorithm based on the graph attributes that you have identified from part b), and **time complexity**. **(3 marks)**

- The algorithms Dijkstra's, Bellman-Ford, Floyd Warshall, Breadth First Search could be used to find the shortest path from node S to node E correctly if every edge is given the weight of 1.
- Depth First Search is not guaranteed to find the shortest path.
- With these considerations Breadth First Search would be the most efficient solution in this case.

Question 4 (10 marks)

Consider the recursive function eMaze defined in pseudocode below for an 8x8 cell matrix representation of a maze.

Blocked cells are filled with an “ * ” or an “ + ” symbol.

An example of the input matrix for the function eMaze is shown to the right.

	A	B	C	D	E	F	G	H
1	*	*	*	*	*			
2	*				*			
3	*		*	*	*			
4	*				*	*	*	*
5	*		*		S			*
6	*				*			*
7	*	*	*	*	*		E	*
8					*	*	*	*

```
function eMaze(maze[][],x,y)
//Input maze: a 2 dimensional matrix, maze[x][y] is a cell in the matrix
//Input x: a column of the maze
//Input y: a row of the maze

if (y>8 OR y<1
  OR x <'A' OR x>'H') then
    //position is outside the matrix
    return false
end if
if (maze[x][y] is '*' or '+') then
    print (blocked) //blocked cell
    return false
end if
if (maze[x][y] equals 'E' then
    print (found E) //reached end
    return true
end if
//mark cell x,y as part of path
print (x,y)
If (maze[x][y] is empty) then
    maze[x][y]='+'
end if
if eMaze(maze,x,y+1) then
    return true
end if
if eMaze(maze,x,y-1) then
    return true
end if
if eMaze(maze,x+1,y) then
    return true
end if
if eMaze(maze,x-1,y) then
    return true
end if
//unmark cell x,y as part of path
If (maze[x][y]='+') then
    maze[x][y]=' '
end if
return false
End function
```

Question 4 (continued)

Use the following 8x8 matrix as input for the **function eMaze** to answer the questions.
The starting cell is labelled with an "S" and has the row $x = E$ and the column $y = 5$.

	A	B	C	D	E	F	G	H
1	*	*	*	*	*			
2	*				*			
3	*		*	*	*			
4	*				*	*	*	*
5	*		*		S			*
6	*				*			*
7	*	*	*	*	*		E	*
8					*	*	*	*

a) Identify all the base cases for the function eMaze. **(3 marks)**

Base cases:
dimension checking,
blocked equals "*" or '+" or
found cell that equals "E"

b) Complete the table below and show the output that will result from the recursive **function eMaze** in the table below when the **function is called** `eMaze(maze,E,5)` **(4 marks)**

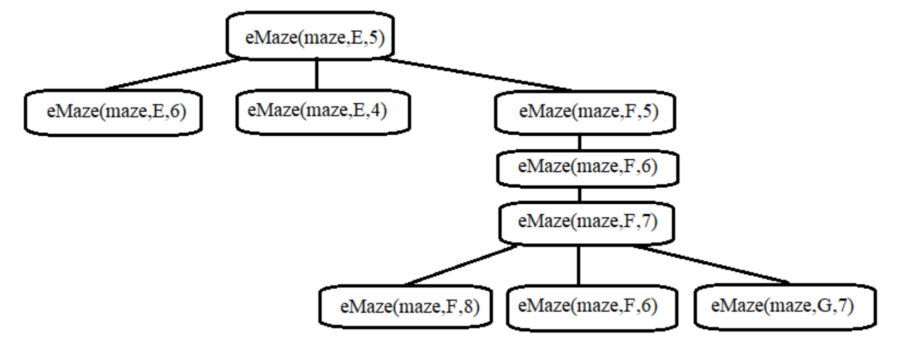
starting at cell x=E,y=5	**Show the order that cells are visited**	**Show the Main Actions trace starting from Cell x=E,y=5**

function eMaze(maze[][],x,y)
if (y>8 OR y<1
OR x <'A' OR x>'H') then
// outside the matrix
return false
end if
if (maze[x][y] is '*' or '+') then
print (blocked) //blocked cell
return false
end if
if (maze[x][y] equals 'E' then
print (found E) //reached end
return true
end if
//mark cell x,y as part of path
print (x,y)
If (maze[x][y] is empty) then
maze[x][y]='+'
end if
if **eMaze(maze,x,y+1)** then
return true
end if
if **eMaze(maze,x,y-1)** then
return true
end if
if **eMaze(maze,x+1,y)** then
return true
end if
if **eMaze(maze,x-1,y)** then
return true
end if
//unmark cell x,y as part of path
If (maze[x][y]='+') then
maze[x][y]=' '
end if
return false
End function
	E,5
	E,6
	blocked "*"
	E,4
	Blocked "*"
	…F,5
	F,6
	F,7
	F,8
	Blocked "*"
	F,4
	Blocked "+"
	G,7
	Found E
		Cell x=E,y=5, down
		eMaze(maze,E,6)
		maze[E][6]='*' blocked
		base case reached
		false return
		Cell x=E,y=5, up
		eMaze(maze,E,4)
		maze[E][4]='*' blocked
		base case reached
		false return
		Cell x=E,y=5, right
		eMaze(maze,F,5)
		eMaze(maze,F,6)
		eMaze(maze,F,7)
		eMaze(maze,F,8)
		maze[F][8]='*' blocked
		base case reached
		false return
		eMaze(maze,F,6)
		maze[F][6]='*' blocked
		base case reached
		false return
		eMaze(maze,G,7)
		maze[G][7]='E' found
		base case reached
		true return

Question 4 (continued)

c) Using the trace from part b) Show the call tree for `eMaze(maze,E,5)`. **(3 marks)**

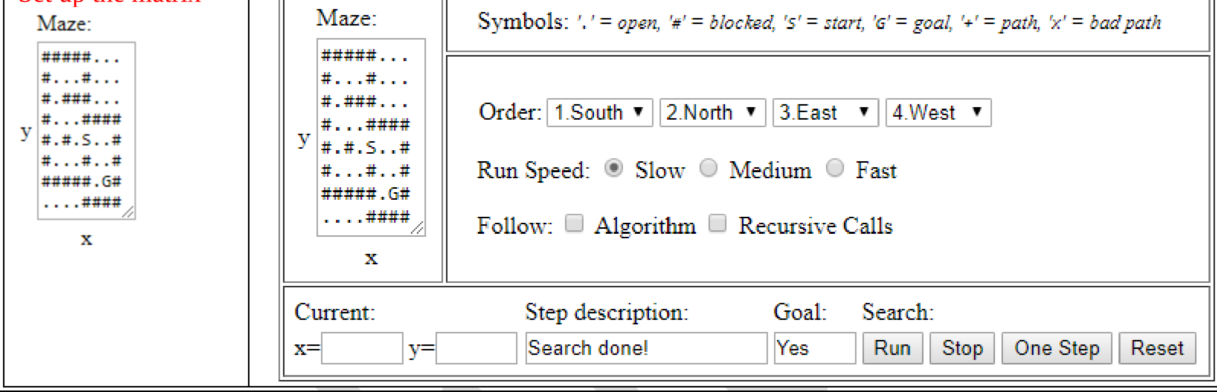
```
eMaze(maze,E,5)
    eMaze(maze,E,6)
    eMaze(maze,E,4)
    eMaze(maze,F,5)
        eMaze(maze,F,6)
          eMaze(maze,F,7)
            eMaze(maze,F,8)
            eMaze(maze,F,6)
            eMaze(maze,G,7)
```



Extra space provided on this page to answer question 4.

solution can be interactively checked at https://www.cs.bu.edu/teaching/alg/maze/

Set up the matrix



Question 5 (10 marks)

At the **Super Big Rooster** restaurant you can only buy chicken nuggets in packages containing 6, 9 or 20 pieces.

a) Write a brute force naive algorithm in structured pseudocode that accepts an integer, N, as an argument and **finds all the possible ways that it is or isn't** possible to buy N nuggets at the Super Big Rooster restaurant. **(5 marks)**

```
Algorithm BuyNuggets (N)
// Input N, the number of nuggets to buy
// Output Countways, BuyList
Create Variable Countways:=0
Create List BuyList
Create Array abc
// Using exhaustive enumeration for guess-and-check to find all the ways
// the floor function is called and always rounds down

for a = 0 to floor(N/6) do
   for b = 0 to floor(N/9) do
       for c = 0 to floor(N/20) do
            if (6×a + 9×b + 20×c) equals N) then
                  Countways := Countways + 1
                  abc[1]:= a
                  abc[2]:= b
                  abc[3]:= c
                  Append array abc to BuyList
            end if
       end do
    end do
end do
If Countways is equal to zero ) then
    Return “Not possible to buy N nuggets”
Else
    Return Countways, BuyList
End if
end Algorithm
```

Question 5 (continued)

At the **Super Big Rooster** restaurant you can only buy chicken nuggets in packages containing 6, 9 or 20 pieces.

For 1 to 10 nuggets the minimum package purchase possibilities are:

1	2	3	4	5	6	7	8	9	10
F	F	F	F	F	1	F	F	1	F

For 11 to 20 nuggets the minimum package purchase possibilities are:

11	12	13	14	15	16	17	18	19	20
F	2 (6+6)	F	F	2 (9+6)	F	F	2 (9+9)	F	1 (20)

For 21 to 30 nuggets the minimum package purchase possibilities are:

21	22	23	24	25	26	27	28	29	30
3 (15+6)	F	F	3 (15+9)	F	2 (20+6)	F	F	2 (20+9)	4 (24+6)

b) Using an Advanced Algorithmic Pattern such as Divide and Conquer, or Dynamic Programming, or Backtracking, create an **efficient algorithm** in structured pseudocode that accepts an integer, N, as an argument and **finds the minimum packages that it is or isn't** possible to buy from 1 to N nuggets at the Super Big Rooster restaurant. **(5 marks)**


```
Algorithm NuggetsDP (N)
// Input N, the exact number of nuggets to buy
// Output Nuggets – an array of minimum Nuggets packages
Create Array Nuggets [1..N]

For i = 1 to N do
    // Initialise the Nuggets array
    Nuggets[i] := ∞
    For j in (6, 9, 20) do
       If ((i > j) AND (Nuggets[i-j] is not ∞) then
          Nuggets[i] := minimum (Nuggets[i], Nuggets[i-j] + 1)
       Else if (i equals j) then
          Nuggets[i] := 1  // Nuggets[6]=1, Nuggets[9]=1. Nuggets[20]=1
       End if
    End do
End do

Return Nuggets
End Algorithm
```

Question 6 (8 marks)

a) Show that for all natural numbers n it is the case that n is divisible by 4 if and only if the number given by the last two digits of n is divisible by 4. *(Hint: One may write any natural number $n = 100k + l$, with $k \in N \cup \{0\}$ and $0 \leq l \leq 99$, so l is the number given by the last two digits of n. Eg. 115 =100+15, 3921=3900+21)* **(4 marks)**

Since we can write any natural number $n = 100k + l$, with $k \in N \cup \{0\}$ *and* $0 \leq l \leq 99$, so l is the number given by the last two digits of n.

We can use statements about divisibility (by 4) and mathematical induction to show the claimed result.

- Let integer $i = \frac{l}{4}$, $4i = l$
- Base case $i = 1, l = 4, \frac{n}{4} = \frac{100k+4}{4} = 25k + 1$ which is true
- Assume this is true for $l = 4i$ where $0 \leq l \leq 95$
- By induction we show that this is true for $l = 4(i+1)$
- By induction $\frac{n}{4} = \frac{100k+4(i+1)}{4} = 25k + i + 1$ which is true

Consider the following recursive algorithm to compute n!

```
Algorithm Factorial
INPUT: n, a positive integer.
OUTPUT: n!.
if n = 1 then
    return 1;
else
    return n × Factorial(n - 1);
fi;
end;
```

b) Prove the correctness of the given factorial algorithm. **(4 marks)**

Induction is the most common device used to prove the correctness of a recursive algorithm.

Proposition 17 *The algorithm* `Factorial` *is correct.*

Proof: By induction of the input n.

(*Induction basis.*) If $n = 1$, then `Factorial` terminates and returns the correct result 1.

(*Induction step.*) Let k be an arbitrary, but fixed positive integer, and assume that algorithm `Factorial` works correctly with input $n = k$. Then if $n = k + 1$, the algorithm returns

$$\begin{aligned}(k+1)\cdot \texttt{Factorial}(k) &= (k+1)\cdot k! \\ &= (k+1)\cdot k\,(k-1)\cdots 2\cdot 1 \\ &= (k+1)!\end{aligned}$$

which is correct.' ■

Question 7 (12 marks)

a) Complete the following table of Computer Science terminology with a short description. **(6 marks)**

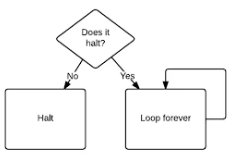
Computer Science terminology	**Description of terminology**
P complexity class	Groups together problems that can be solved correctly (determined) with worst case polynomial time algorithms.
NP complexity class	Groups together problems that cannot be solved/determined with polynomial time algorithms, but their solutions should they be found non-deterministically (randomly) can be verified for correctness in polynomial time.
NP-Complete problem	Groups together the hardest problems in NP, these problems can be reduced/transformed to each other in polynomial time. Therefore should a polynomial solution be found for one problem in NP-Complete, then theoretically they can all become P class.
Intractable problem	Problems that do not have polynomial time algorithms for finding the correct solution, and are in one of the classes of {NP, NP-Complete, NP-Hard}.
Decidable problem, give an example	Are problems for which the output or answer is either yes or no. The decision 01 knapsack problem, can a knapsack of capacity C be filled with items whose value exceeds $k.
Computable problem, give an example	Are problems that can be solved correctly with a proven algorithm. Finding a minimum spanning tree in a connected weighted graph is a computable problem, solved with Prim's algorithm and others.

b) What is Hilbert's program? What were the aims of the program? Explain how Hilbert's program can be classified using Computer Science terminology and give reasons for your answer. **(3 marks)**

- Hilbert proposed to ground all existing Mathematical theories to a finite, complete set of axioms and theorems and provide a proof that these axioms and theorems were consistent.
- Hilbert's program was unattainable for key areas of mathematics involving non-finitary (real numbers) arithmetic as demonstrated by Godel's incompleteness theorem.
- It could be argued that Hilbert's program was uncomputable.

c) What is the Halting problem? What is its classification using Computer Science terminology and give reasons for your answer. **(3 marks)**

- the halting problem is the problem of determining, from a description of an arbitrary algorithm/computer program and its input, whether the program will finish running, or continue to run forever.
- the halting problem is undecidable over Turing machines as a yes or no answer cannot be determined
- the reason this is undecidable is that any algorithm written to determine the halting of another algorithm cannot verify the halting of itself.



END OF TRIAL EXAM