Office Use Only

MONASH University

Semester Two 2019
Examination Period

Faculty of Information Technology

EXAM CODES:	**FIT2004**
TITLE OF PAPER:	Algorithms and Data Structures
EXAM DURATION:	2 hours

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Caulfield [✓ Clayton] ☐ Parkville ☐ Peninsula
☐ Monash Extension ☐ Off Campus Learning [✓ Malaysia] ☐ Sth Africa
☐ Other (specify)

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations, or a breach of instructions under Part 3 of the Monash University (Academic Board) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	☐ **YES**	[✓ **NO**]
CALCULATORS	☐ **YES**	[✓ **NO**]
SPECIFICALLY PERMITTED ITEMS **if yes, items permitted are:**	☐ **YES**	[✓ **NO**]

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ DESK NUMBER: __ __ __ __ __

This page is intentionally left blank. If needed, you can use this space to write answers.

0

INSTRUCTIONS

- You must answer ALL the questions.
- Answers to each question should be in the space DIRECTLY BELOW the questions and (if required) on the blank page overleaf of each question.

==

Best of Luck!

==

Do not write anything in this table. It is for office use only.

Question	Points	Score
1	5	
2	3	
3	6	
4	5	
5	7	
6	4	
7	5	
8	7	
9	4	
10	3	
11	11	
Total:	60	

0

This page is intentionally left blank. If needed, you can use this space to write answers.

0

1. Consider the function `mystery(N, M)` shown below and answer the following questions.

```
def mystery(N, M):
  if N == 1:
    return M
  else:
    x = 3 * mystery(N//3, M)
  return x
```

(a) (1 mark) Write the recurrence relation for the time complexity of the `mystery` function.

Recurrence relation:
$T(1) = b$
$T(N) = T(N//3) + a$

- 1 mark for correct recurrence relation (deduct 0.5 mark for minor mistakes)

1

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(b) (4 marks) What is the time complexity of `mystery` in Big-O notation? You need to show appropriate working out, or you will receive no marks.

Solution:
$T(N) = T(N//3) + a$
$T(N) = T(N//9) + 2a$
$T(N) = T(N//27) + 3a$
$T(N) = T(N//3^k) + ka$
Base when $N//3^k = 1$; $k = log_3 N$
$T(N) = T(1) + log_3 N * a$
Time complexity in Big-O notation is $O(log_3 N)$ or just $O(log(N))$ since the base is a constant factor. Either is correct.

- 3 marks for correctly solving it (deduct 1 mark for each mistake).
  If the recurrence relation above is wrong, this part should be marked based on the recurrence relation they wrote except that the maximum achievable mark for this part should be 1.5.
- 1 mark for time complexity. The answer should be based on their solution to the recurrence relation

- Note: This question is made somewhat easier but still maintain the trickiness to realize the $3 * mystery(N//3)$ is still a single call. Also having an additional argument M as the distraction for the student.

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

2. Suppose we wish to “stably” sort an array of size N, containing integers in the range $[1, M]$, using count sort.

   (a) (2 marks) During count sort, a non-constant amount of auxiliary space is allocated, in the form of several arrays. Describe what each of these arrays contains.

First method: Position and count arrays
Count array: counts the number of times each key occurs in the input
Pos array: records, for each key, the position in the output where the next instance of that key belongs
Output array: An array which holds the sorted values

Second method: Using linked lists
Create an array of size M, which will contain at each index a linked list of all the elements whose keys are that index
Output array: An array which holds the sorted values

First method: Using position and count arrays

- Note: Names can be anything, but the descriptions need to be correct to obtain the marks.
- 0.5 mark for count array
- 1 mark for pos array
- 0.5 marks for mentioning output array

Second method: Using linked lists

- Note: Names can be anything, but the descriptions need to be correct to obtain the marks.
- 0.5 marks for mentioning the size M
- 1 marks for clearly explaining what will be stored at each index
- 0.5 marks for mentioning output array or updating the input array for output.

2

(b) (1 mark) What is the auxiliary space complexity of stable count sort? Justify in terms of your answer to part (a)

First method: $O(N+M)$. Both pos and count are $O(M)$, but the output array will be $O(N)$. Second method: $O(N+M)$. The array is size M, and the total number of elements in all the linked lists will be N, so the total size is $O(N+M)$.

- 0.5 mark for correct complexity.
- 0.5 mark for correct justification

1

This page is intentionally left blank. If needed, you can use this space to write answers.

0

3. Consider a partition algorithm for quicksort which places all elements less than the pivot on the left of the pivot, and all elements greater than or equal to the pivot on the right of the pivot. This algorithms runs in $O(n)$ time, where n is the length of the list to be partitioned.

   (a) (1 mark) What is the worst case time complexity of quicksort using the above partition algorithm, assuming that the median element is always chosen as the pivot?

   $O(n^2)$. This occurs when every item is the same

   - 1 mark for correct answer (no justification required)

   (b) (1 mark) Define what it means for an algorithm to be "in-place".

   "In-place" means uses O(1) auxiliary space.

   - 1 mark for "O(1) auxiliary space" definition of in place (or similar)

2

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(c) (2 marks) Is it possible to implement a partition algorithm which behaves as described above, and which is in-place? Briefly explain your answer.

It is possible, since there are partition algorithms which just swap elements in the list, meaning that O(1) auxiliary space is allocated by the algorithm.

- 1 mark for "yes" answer
- 1 mark for explaining why partition uses O(1) aux space

(d) (2 marks) Is it possible to devise an in-place algorithm for quicksort? Justify your answer.

No, because the recursive calls take up at least log(n) space on the stack (whether we mean the system call stack or a stack data structure inside the algorithm)

- 1 mark for "no" answer.
- 1 mark for saying recusrive calls take up space (or other good explanation)

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

4. (a) (2 marks) Consider the problem of calculating the n^{th} Fibonacci number using dynamic programming. State the overlapping subproblems and the optimal substructure for this problem.

Overlapping subproblems: The values of Fibonacci numbers 0...n. Alternatively, students may express this as DP[i] = {The value of the i^{th} Fibonacci number}
Optimal substructure: Fib(i) = Fib(i-1) + Fib(i-2)

- 1 mark for exrpressing the subproblems
- 1 mark for expressing the recurrence

(b) (3 marks) Consider the following dynamic programming problem: Given a list of numbers `L[1..n]`, you wish to select numbers so that the sum of the numbers you select is maximised. However, if you select a number in position i, you cannot select any of the numbers from positions $i-2, i-1, i+1, i+2$.

Our memo array `memo[1..n]` is defined as `memo[i] = {The maximum value that can be obtained when considering numbers in L[1..i]}`. Write down the recurrence relation for the values in this memo array.

Recurrence relation:

```
memo[1] = L[1]
memo[2] = max(L[1], L2])
memo[3] = max(memo[2], L[3])

memo[i] = max(memo[i-3] + L[i], memo[i-2], memo[i-1]); i > 3
```

- 1 mark for base case
- 2 mark for recursive case

5

This page is intentionally left blank. If needed, you can use this space to write answers.

0

5. (a) (1 mark) Define a perfect hash function.

A perfect hash function is one which never results in collisions. Alternatively, a perfect hash function is one which maps each key to a unique index.

- 1 mark for no collisions/unique indices.

(b) (1 mark) Describe the circumstances which are required for a perfect hash function to be possible.

We need to know the universe of keys in advance, and it needs to be finite

- 0.5 marks for knowing keys in advance
- 0.5 marks for mentioning there must be finitely many keys

(c) (2 marks) Suppose we implement a hash table which uses separate chaining, using AVL trees instead of linked lists as the chains. What is the worst case time complexity of a lookup in such a hash table, if the hash table currently contains n elements and the table is size m? Briefly justify your answer.

$O(log(n))$ where n is the number of elements in the hash table. This occurs when all the elements are hashed to the same position, so we have an AVL tree of size n, which has worst case lookup of $O(log(n))$

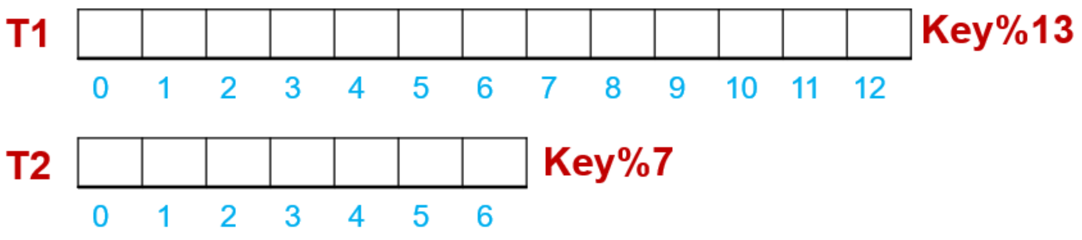
- 1 mark for O(log(n))
- 1 mark stating that all n elements are in one AVL tree

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(d) (2 marks) Assume that cuckoo hashing uses two tables as shown below. The hash function for the table T1 is *key*%13 and the hash function for T2 is *key*%7. Insert the values 21, 125 and 47 in that order to the tables below. You only need to show the final positions of the three values.



T1 has 47 at position 8; T2 has 21 at position 0 and 125 at position 6

- 1 mark for values corrected inserted into T1
- 1 mark for values corrected inserted into T2

(e) (1 mark) What is the worst-case time complexity to search for a key in a cuckoo hashing scheme? Briefly explain.

$O(1)$ because the key could only exist in its hashed position for T1 or T2.

- 1 mark for correct complexity with explanation.
- Note: 0 marks if there is no explanation.

3

This page is intentionally left blank. If needed, you can use this space to write answers.

0

6. Consider a suffix **trie** created by repeatedly inserting the suffixes of a given string of length N into an initially empty trie. Each node in this trie represents one character.

   (a) (1 mark) What is the worst-case space complexity of such a suffix trie? Briefly justify your answer.

   Suffix tree: $O(N^2)$
   There are N suffixes, of lengths N, $N-1$ ... 1, so the total space complexity is $O(N^2)$

   - 0.5 marks for correct complexity
   - 0.5 mark for arguing that the total space taken up by the suffixes is $O(N^2)$

   (b) (3 marks) Argue that an optimized suffix trie (i.e. a suffix tree) has $O(N)$ space complexity.

   There are $O(N)$ leaves, since each leaf corresponds to a suffix. The number of nodes in each level above is at most half the number of the previous level (since each internal node has at least 2 children). So the total number of nodes is bounded by $O(N + N/2 + N/4 + ... + 1) \leq 2N$ which is $O(N)$. Each node takes constant space, since it only needs to store a 2 element tuple.

   - 1 mark for saying there are N or $O(N)$ leaves
   - 1 mark for saying that internal nodes have at least 2 children
   - 0.5 mark for correct summation of series
   - 0.5 mark for explaining why each node is constant space

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

7. (a) (1 mark) Write down the suffix array for the string ABBAAB$, assuming that $ is the terminator character. Use 1-indexing in your answer.

[7,4,5,1,6,3,2]

- 1 mark for a correct suffix array. 0.5 marks if the sorted suffixes are written instead of indices.

(b) (1 mark) Describe how the value at position i of a suffix array of a string S can be used to obtain the i^{th} character of the Burrows Wheeler Transform (BWT) of S.

```
if SA[i] = 1:
BWT[i] = '$'
else:
BWT[i] = S[SA[i]-1]
```

- 0.5 marks for correctly stating that it is the character at SA[i] - 1. This can be in words instead of pseudocode. Note that it needs to be a character, since the BWT is not just indices. So saying something like BWT[i] = SA[i] - 1 is not correct.
- 0.5 marks for identifying the edge case of '$'

(c) (1 mark) Briefly state why BWT is useful for compression.

In the BWT of a string, similar characters tend to be grouped together.

- 1 mark for stating the similar characters tend to be grouped together.

3

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(d) (2 marks) Assume that we are constructing suffix array of `MISSISSIPPI$` using prefix doubling approach and we have already sorted the suffixes on their first 2 characters with the corresponding rank array shown below.

Index/ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
String	**M**	**I**	**S**	**S**	**I**	**S**	**S**	**I**	**P**	**P**	**I**	**$**
Rank	**5**	**4**	**9**	**8**	**4**	**9**	**8**	**3**	**7**	**6**	**2**	**1**

Describe how will you compare the two suffixes with ID 2 and ID 5 on their first 4 characters in $O(1)$.

First, we compare the rank of both suffixes. Both have the same rank; rank[2] = rank[5] = 4. The same rank means the first 2 characters of the suffixes are the same. Thus, we compare them on the next 2 character, rank[2 + 2] = 8 and rank[5 + 2] = 8. The rank remains the same; this means that the first 4 characters of the suffixes are still the same.

- 1 mark for comparing ranks of suffix 2 and 4; stating it to be similar on first 2 character.
- 1 mark for comparing ranks+2 and stating that the first 4 characters are the same without comparing another level of +2. If there is another level of +2 for the comparison, then 0 mark because it can't be resolved in the current iteration.

2

This page is intentionally left blank. If needed, you can use this space to write answers.

0

8. (a) (2 marks) The Graph data structure can be implemented using an **adjacency matrix** representation with the benefit of $O(1)$ random access.

Briefly describe the 2 possible scenarios where the **adjacency list** representation is more suitable than the **adjacency matrix**.

1. When the graph is sparse, there will be no wasted space.
2. For algorithms where all edges are traversed, adjacency list would avoid the need to check if an edge exist.

- 1 mark for each scenario.

(b) (2 marks) Given a unweighted graph G and a vertex s, we want to find the longest path (i.e. most edges) starting with s in G which does not contain a cycle. Describe an algorithm which solves this problem. You do not need to write code or psuedocode.

A backtracking algorithm which explores all paths and finds the longest one is required. Simply running BFS and finding the farthest vertex from the source will not work, and running DFS will also not work as the DFS may take a path which prevents it from finding the longest path.

- 1 mark for using a modified version of DFS to allow backtracking (or just backtracking)
- 1 mark for correctly finding the longest path (there are many ways this can be done)
- some marks awarded for using brute force or other less efficient methods

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(c) (3 marks) Consider a directed graph $G(V, E)$ with some negatively weighted edges but with no negative cycle. We wish to find the shortest distance between all pairs of vertices in G. For each of the algorithms listed below, state whether it is possible to use them to solve this problem; and if it is possible, explain how they could be used, and the time complexity of the approach in terms of the number of edges and number of vertices.

1. Dijkstra
2. Bellman-Ford
3. Floyd-Warshall

1. Dijkstra is not possible because there are negative edges that breaks the greediness of the algorithm.
2. Bellman-Ford needs to be ran from all source vertex with a total complexity of $O(V.EV)$.
3. Floyd-Warshall is ran once with a total complexity of $O(V^3)$.

- 1 mark for saying dijkstra cannot be used
- 0.5 marks for saying Bellman Ford needs to be run from each vertex
- 0.5 marks for saying O(V.EV) for bellman ford complexity
- 0.5 marks for saying floyd warshall can be used
- 0.5 marks for correct complexity for floyd warshall

3

This page is intentionally left blank. If needed, you can use this space to write answers.

0

9. (a) (4 marks) Consider a directed graph G with some negatively weighted edges and with negative weight cycle. Can Prim's algorithm be used to find the minimum-spanning tree T of the graph? Provide a strong justification for your answer.

A possible answer: Choose the most negative edge weight, w. Add w to every edge, giving a graph with no negative edges. Run prim's which we know works correctly. Then subtract w from every edge, to obtain the MST of the original graph.

Another possible answer: Give the proof of prim's algorithm shown in lectures (no where does that proof use the non-negativity of edges)

Another possible answer: Prim's algorithm simply chooses the lightest edge adjacent to the tree which connects a vertex in the tree and a vertex not in the tree. Whether this edge is positive or negative is irrelevant, since negative weight edges only cause problems when they a) break the greedy metric used in dijkstras or b) create negative cycles, which are not present in trees

- 1 mark for mentioning it is possible.
- 3 marks for a correct presentation of either of the first 2 arguements

OR

- 1 mark for mentioning it is possible.
- 2 mark for mentioning that we grow the tree by adding edges to the tree if they do not create a cycle; prioritizing the smallest edges and negative edges are the smallest.
- 1 mark for mentioning that the cycle does not matter, especially tree doesn't have a cycle.
- Note: The student can draw to illustrate their point

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

10. (a) (3 marks) In Kahn's algorithm for topological sort in a Directed Acyclic Graph (DAG), we need to determine when a vertex has no incoming edges. Describe how this information could be stored so that it could be updated and accessed efficiently, and how it is updated and used during a topological sort.

Data structure
EITHER The vertex object v would have an attribute which keeps count of the number of incoming edges.
OR An array indexed by vertex IDs stores the number of incoming edges for each vertex
initialisation
EITHER values is initialized to 0. Whenever an edge $e =< u, v >$ is added to the graph G, the value of the attribute will increase by 1.
OR It is intialised to the number of incoming edges of v.
update
On edge relaxation during traversal through edge $e =< u, v >$, this value for vertex v would decrease by 1. If this value reaches 0, then the vertex v would be added to the discovery queue.

OPTIONAL As the topological sort can be ran multiple time, this value would be stored with 2 variables.

- 1 mark for each point.

3

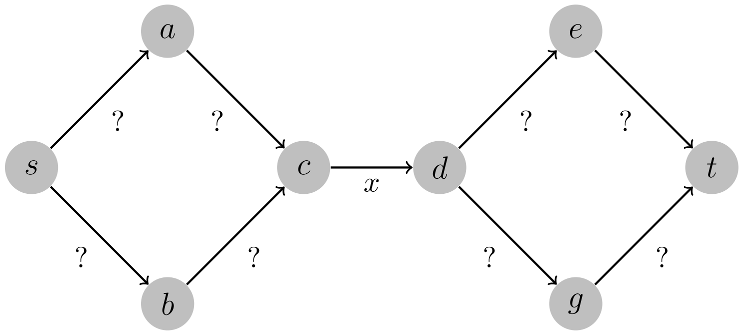
This page is intentionally left blank. If needed, you can use this space to write answers.

0

11. (a) (3 marks) Consider the network flow diagram shown below. The maximum flow in this network is f. None of the capacities are known.

For each of the three statements below, either state that it must be true in the given graph, or give an example of a situation in which it is false.

1. $x \leq f$
2. $x = f$
3. $x > f$



None of the statements need be true. The capacity of an edge can be greater than the flow, so 1 could be false, and same with 2. However, it is also possible that $x = f$, so 3 does not need to be true either.

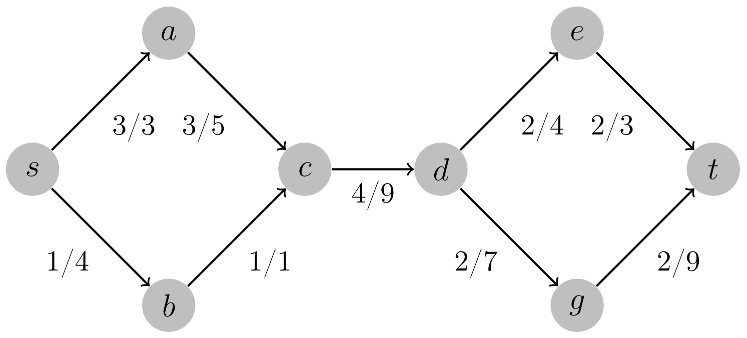
- 0.5 marks for each "false" claim
- 0.5 marks for an explanation/counterexample for each of the three statements.

3

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(b) (2 marks) Consider the network flow diagram shown below. Produce its corresponding residual network.



For every edge a/b, there should be a forward edge with weight b-a (unless b-a = 0) and a backwards edge with weight a.

- 1 mark if all forward edges are correct.
- 1 mark if all reverse edges are correct.

(c) (2 marks) Is the flow through the network depicted in part (b) a maximum flow? Justify your answer.

The flow across every cut in the network flow is the same and is the flow of the network; but the capacity across each cut differs. The flow across a cut is bounded by the capacity across the cut.

A min-cut is a cut where the flow across the cut is the same as the capacity across the cut. Thus, the it is not possible to have a greater flow in the network.

- 0.5 mark for saying that the flow through every cut is the same as the flow of the network
- 1 mark for saying that the flow is maximum when some cut is saturated (0.5 marks if they do not cite the max-flow min-cut theorem)
- 0.5 marks for identifying the saturated cut

4

This page is intentionally left blank. If needed, you can use this space to write answers.

0

(d) (4 marks) Consider a timetabling system where the students are allowed to provide 3 preferences for FIT2004 tutorials. Each student will only be allocated to 1 tutorial. Each tutorial could however only accommodate 20 students. Describe in detail how would you model this scenario as a flow network and use it to satisfy as many students as possible.

Answer is as follows: Create a node for each student. Create a node for each tutorial. From each student node, create 3 edges which lead to the tutorials that student has preferenced. Each edge should have capacity 1. Create a single source, and create and edge from the source to each student with capacity 1. Create a single sink, and create an edge from each tutorial with capacity 20.

Run Ford Fulkerson on this graph to obtain the maximum flow. The edges from students to tutorials which have a flow through them represent which student should be allocated to which tutorial, in order to satisfy the largest number of students.

- 1 mark for creating a single source and sink.
- 0.5 mark for connecting the source to each student with a capacity of 1.
- 0.5 mark for connecting the each class to the sink with a capacity of 20.
- 1 mark for connecting each student to their 3 preferred class with a capacity of 1.
- 1 mark for mentioning using existing maximum flow approaches such as the Ford-Fulkerson method to solve it.

This is the end of the test.

4