Official Use Only

Monash University

Semester One Mid Semester Test 2018

Faculty of Information Technology

EXAM CODES: **FIT2004 (Mid-semester Test T2)**
TITLE OF PAPER: **Algorithms and Data Structures**
TEST DURATION: 45 minutes
READING TIME: 5 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT:

☐ Berwick ✓ Clayton ✓ Malaysia ☐ Off Campus Learning ☐ Open Learning
☐ Caulfield ☐ Gippsland ☐ Peninsula ☐ Enhancement Studies ☐ Sth Africa
☐ Pharmacy ☐ Other (specify)

During an exam, you must not have in your possession, a book, notes, paper, electronic device/s, calculator, pencil case, mobile phone or other material/item which has not been authorised for the exam or specifically permitted as noted below. Any material or item on your desk, chair or person will be deemed to be in your possession. You are reminded that possession of unauthorised materials in an exam is a discipline offence under Monash Statute 4.1.

No examination papers are to be removed from the room.

AUTHORISED MATERIALS

CALCULATORS	**☐ YES**	**✓ NO**
OPEN BOOK	**☐ YES**	**✓ NO**
SPECIFICALLY PERMITTED ITEMS	**☐ YES**	**✓ NO**

STUDENT ID __ __ __ __ __ __ __ __ __

Office use only

This page is intentionally left blank. You may write your answers here if more space is needed.

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.

General exam technique

Do not throw marks away by **not** attempting all questions. Suppose you get 7/10 on a question for a 20 minutes effort. Spending another half hour on the same question gets *at most* 3 more marks. On the other hand, were you to spend that time on a new question, you might get another 10 marks.

Answer the question that is asked of you. If the question asks for Insertion sort, do not write Quick-sort – this only wastes your time.

Do not write un-necessarily long answers. This wastes your valuable exam time. The question will specifically ask for the information required. Therefore, do not include the information that is not specifically asked for. If asked to justify your answer, provide a clear, logical and concise reasoning.

You do not have to attempt the questions in order. Some questions require less work but may be worth more marks. Carefully read the paper to decide the order in which you should attempt the questions based on the marks associated with each question and whether you know the answer or not.

Best of Luck!

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

Question	Points	Score
1	8	
2	6	
3	4	
4	6	
Total:	24	

This page is intentionally left blank. You may write your answers here if more space is needed.

0

1. This question is composed of short questions. Write your answers to each of these questions in no more than a few lines.

   (a) (2 marks) What are the space complexity and auxiliary space complexity of insertion sort? Give brief reasoning.

   > Space complexity is $O(N)$ which is the total space taken by it. Auxiliary space complexity of insertion sort is $O(1)$ because it only takes constant amount of space in addition to the input array.

   (b) (2 marks) Consider the Python function shown below which returns the number of times `item` appears in `myList`. Write a loop invariant which holds at the end of each iteration of the while loop (write next to `#INVARIANT`). Using this loop invariant, show that the algorithm returns correct count of `item` in `myList`.

```
# this function returns how many times the item appears in myList
def countItem(myList, item):
    count = 0
    i = 0
    while i < len(myList):
        if myList[i] == item:
            count += 1
        i += 1

        #INVARIANT:


    return count
```

   > INVARIANT: count represents the number of times item appears in myList[:i]
   > $i = len(myList)$ when the while loop terminates. Thus, count represents the number of times item appears in myList[:len(myList)] (i.e., in myList[:]).

This page is intentionally left blank. You may write your answers here if more space is needed.

0

(c) (2 marks) What is the worst-case time complexity of searching in Cuckoo hashing? Give brief reasoning.

$O(1)$ because the item (if present) must be in one of the two tables at its hash value.

(d) (2 marks) Define balance factor of a node in an AVL Tree. What must be the balance factor for every node in a binary search tree so that it can be called an AVL tree?

Balance factor of a node is the difference between the height of its left subtree and the height of its right subtree. The balance factor of each node must be 0, 1 or -1.

This page is intentionally left blank. You may write your answers here if more space is needed.

0

2. (a) (3 marks) Assume that you have already implemented an efficient Quickselect algorithm which, given a value k, finds k-th smallest number in an unsorted array in $O(N)$ worst-case time complexity. Assume that you also have an algorithm for partitioning (as shown in class) which partitions the array using a pivot p such that all numbers smaller or equal to p are on its left in the array and all numbers greater than p are on its right. Using the Quickselect and partitioning algorithm, write a Quicksort algorithm that runs in $O(N \log N)$ in the **worst-case**.

```
Quicksort:
    num = len(array)//2
    median = Call Quickselect to get num-th smallest number
    Call partitioning using median as the pivot
    Recursively call Quicksort on left_array
    Recursively call Quicksort on right_array
```

(b) (3 marks) Justify why your Quicksort algorithm in part(a) has $O(N \log N)$ time complexity in the worst-case. If you were unable to write Quicksort algorithm above, you may assume that a correct implementation of Quicksort in part(a) is available for your analysis.

Since Quickselect always returns median as a pivot in O(N), the height of the recursion tree is $O(logN)$. The total cost at each level is $O(N)$ because both partitioning and Quickselect take time linear to the size of the array. Therefore, the worst-case cost is $O(NlogN)$

This page is intentionally left blank. You may write your answers here if more space is needed.

0

3. (a) (3 marks) Consider the Python function shown below. Write its recurrence relation for time complexity and solve it. Also, write its time complexity based on the solution to recurrence relation.

```
def mystery(n):
    if n == 0:
        return 0
    else:
        value = 0
        for i in range(n):
            value += i
        return value + mystery(n-1)
```

$T(0) = b$
$T(n) = T(n-1) + a * n$
$T(n-1) = T(n-2) + a * (n-1)$;
So $T(n) = T(n-2) + a * (n + (n-1))$.
$T(n-2) = T(n-3) + a * (n-2)$.
So, $T(n) = T(n-3) + a * (n + (n-1) + (n-2))$.
$T(n) = T(n-k) + a * (n + (n-1) + \cdots + (n-k+1))$.
Setting $k = n$
$T(n) = T(0) + a * (1 + 2 + 3 + \cdots + n) = b + a * n * (n+1)/2$
Thus, the time complexity is $O(n^2)$.

(b) (1 mark) What is the space complexity of the `mystery(n)` function. Give a brief reasoning.

The space complexity is $O(n)$ which is the space taken by recursion calls.

4

This page is intentionally left blank. You may write your answers here if more space is needed.

0

4. (a) (2 marks) Assume that you have a suffix trie for a string. Briefly describe how would you find the longest repeated substring in it. For example, the longest repeated substring in `MISSISSIPPI` is `ISSI`. What is the worst-case time complexity of your approach and why?

Traverse the suffix trie (e.g., using BFS or DFS) and find the deepest node containing at least two children. The time complexity is $O(N^2)$ as the trie has $O(N^2)$ nodes.

(b) (2 marks) Assume that you have a suffix trie for a string. Briefly describe how can you use it to find the most frequent character. E.g., if the string is `REFERRER`, the most frequent character is `R` which appears 4 times in the string. What is the time complexity and why?

Look at the children of the root node and count the number of dollar signs below each node. The child node with maximum number is the one that is repeated maximum number of times. If the suffix trie maintains, for each node, the number of dollar signs below it then the time complexity is $O(1)$ assuming that the alphabet size is constant (26). If the suffix trie does not maintain the count of dollar signs, we may need to traverse the whole tree to do the counting which takes $O(N^2)$.

(c) (2 marks) What is the space complexity of a suffix trie and why?

The space complexity of a suffix trie is $O(N^2)$ because it stores N suffixes each of size at most N.

This is the end of the test.

0