MONASH University

Office Use Only

Semester Two 2015
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT1008**
TITLE OF PAPER: **COMPUTER SCIENCE – PAPER 1**
EXAM DURATION: 3 hours writing time
READING TIME: 10 minutes

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

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

During an exam, you must not have in your possession, a book, notes, paper, electronic device/s, calculator, pencil case, mobile phone, smart watch/device 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, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

No exam paper or other exam materials are to be removed from the room.

AUTHORISED MATERIALS

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

if yes, items permitted are:

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

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

Page		Mark
3		7
5		7
7		4
9		5
11		12
13		7

Page		Mark
15		8
17		5
19		4
21		8
23		7
25		6
Total:		80

Blank Page

Question 1 (6 + 1 = 7 marks)

This question is about *sorting algorithms*.

(a) What is the best and worst case complexity in Big O of the three simple sorting algorithms: *Bubble Sort*, *Selection Sort*, and *Insertion Sort*? (*No explanation means no marks*).

(b) Which of the three sorting algorithms as mentioned above is not a *stable* sorting method? (*No explanation means no marks*).

7

Blank Page

Question 2 (5 + 2 = 7 marks)

This question is about *searching*.

(a) Write a Python function,

```
def find_sum(array, k)
```

which given a sorted array, **`array`**, and a value k, returns a tuple, **`(i,j)`**, of two distinct indexes i and j such that **`array[i] + array[j] = k`**. The algorithm should return **`None`** if no such indexes exist.

Examples:

`find_sum([1, 2, 3, 4, 5], 5)` returns either **`(0, 3)`** because 1 and 4 sum to 5, or **`(1, 2)`** because 2 and 3 sum to 5.

`find_sum([1, 2, 3, 4, 5], 8)` returns **`(2, 4)`** because 3 and 5 sum to 8.

`find_sum([1, 2, 3, 4, 5], 2)` returns **`None`**

(b) Explain the worst time complexity for the function you defined in part **(a)**. (*No explanation no marks*).

7

Blank Page

Question 3 (1 + 1 + 1 + 1 = 4 marks)

This question is about *time complexity*. For each of the given Python functions, explain the worst time complexity. (*No explanation means no marks.*)

(a)
```
def total_func(n):
    total = 0
    for k in range(n):
        for num in range(k):
            total += num
```

(b)
```
def division_func(n):
    product = 1
    while n > 1:
        product *= 2
        n //= 2
```

(c)
```
def another_total_func(n):
    total = 0
    for num in range(n):
        total += num
    for num in range(2*n):
        total += num
```

(d)
```
def mid_func(n):
    low = 0
    high = n
    while low < high:
        mid = (low + high) // 2
        low = mid + 1
```

4

Blank Page

Question 4 (5 marks)

This question is about *searching*. Write a Python function,

```
def max_repetitions(a_list)
```

which given a list, `a_list`, sorted in ascending order, finds an element that appears the maximum number of times. *Note there can be more than one element that appears the maximum number of times.*

Examples:

`max_repetitions([1, 1, 2, 2, 2, 3, 2])` returns `2`.
`max_repetitions([1, 1, 1, 1, 1, 1, 1])` returns `1`.

`max_repetitions([])` returns `None`

`max_repetitions([1, 2, 3, 4, 5, 6])` returns either `1`, `2`, `3`, `4`, `5`, or `6`.

5

Blank Page

Question 5 (10 + 2 = 12 marks)

(a) This question is about *MIPS programming and understanding function calls*. Translate the following Python code faithfully into MIPS assembly language. Make sure you follow the MIPS function calling and memory usage conventions.

Python Code	MIPS Code
`def func(n):`	
`    if n == 0:`	
`        return 0`	
`    else:` `        return n + func(n-1)`	

(b) Explain why a recursive function may use more memory than an iterative function.

12

Question 6 (5 + 2 = 7 marks)

(a) This question is about *recursion* and *binary trees*.

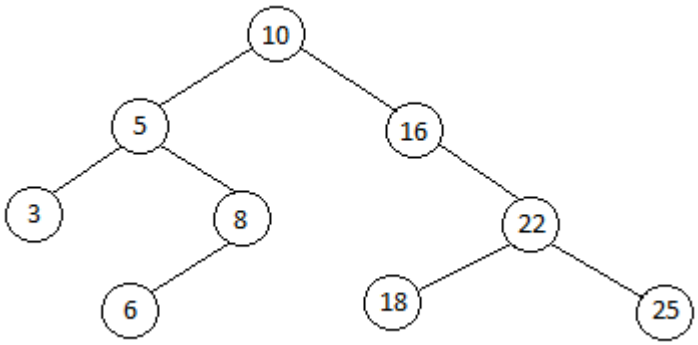
Consider the two classes **`BinaryTreeNode`** and **`BinaryTree`** which define a **Binary Data type** implemented using linked nodes, and which are defined as follows:

```
class BinaryTreeNode:
      def __init__ (self, value, left=None, right=None):
            self.value = value
            self.left = left
            self.right = right

class BinaryTree:
      def __init__ (self):
            self.root = None
```

Define a recursive method **`find_path_max_sum(self)`** inside the **`BinaryTree`** class that finds the maximum sum of values of the nodes within a path from the root to a leaf. The method returns the maximum sum.

For example, if the binary tree is:



there are 4 paths from the root to a leaf. The sum of the values in these paths are 18, 29, 66, and 73. So in this case the method should return **`73`**. If the binary tree is empty, the value return should be zero.

Write your answer of this part on the next page

Write your answer for Question 6(a) here

(b) Explain the best and worst time complexity of the function you defined in part **(a)**. (*No explanation means no marks*).

7

Blank Page

Question 7 (2 + 3 + 3 = 8 marks)

Consider the partial implementation of Circular Queue based on an array.

```
class CircularQueue:
    def __init__(self, size):
        assert size > 0, "Size should be positive"
        self.the_array = size*[None]
        self.count = 0
        self.rear = 0
        self.front = 0

    def is_empty(self):
        return self.count == 0

    def is_full(self):
        return self.count >= len(self.the_array)
```

(a) Define method reset, and state its worst case complexity

(b) Define method append, and state its worst case complexity

(c) Define method serve, and state its worst case complexity

8

Blank Page

Question 8 (5 marks)

This question is intended to test your skills at programming with *queues* and *stacks.*

Suppose you have a **`Queue`** class which implements a queue using some data structure (you do not need to know which one) and defines the following methods:

```
__init__()
append(item)
serve()
is_empty()
```

and a **`Stack`** class which implements a stack using some data structure (you do not need to know which one) and defines the following methods:

```
__init__()
push(item)
pop()
is_empty()
```

Define a function:

```
def reverse_k(a_queue, k)
```

which is given a queue, **`a_queue`**, and a non-negative integer **k**, reverses the first **k** elements of **`a_queue`** while keeping the rest of the elements in the given order. If **k = 0**, then **`a_queue`** should remain unchanged.

You must only use the methods defined for the above classes.

5

Blank Page

Question 9 (2 + 2 = 4 marks)

This question is about *recursive programming* and *linked structures*. Consider the two classes **`Node`** and **`List`** as seen in the lectures, which define a **list data type** implemented using a linked structure:

```
class Node:
    def __init__ (self, item = None, link = None):
        self.item = item
        self.next = link

class List:
    def __init__ (self):
        self.head = None

    def is_empty (self):
        return self.head is None
```

(a) Now consider the following code:

```
def mystery(a_list, node):
    if node is not None:
        right = node.next
        if a_list.head is not node:
            node.next = a_list.head
            a_list.head = node
        else:
            node.next = None
        mystery(a_list, right)

def my_function(a_list):
    mystery(a_list, a_list.head)
```

Describe what **`my_function`** does to a linked list?

(b) What is the worst-case complexity for **`my_function`**? (*No explanation means no marks*)

4

Question 10 (1 + 2 + 5 = 8 marks)

This question is about *hash tables*.

(a) Why should the table size and the constants defined in a hash function be *prime*? Explain (*no explanation means no marks*).

(b) Under which situation is Linear Probing the preferred solution for addressing collisions as compared to Double Hashing and Quadratic Probing? Explain (*no explanation means no marks*).

(c) Consider the class **`Hash`** which has the instance variables **`array`**, **`table_size`**, and **`count`**, and the following methods:

```
__init__()
hash(the_key)
```

Using Linear Probing, define a method **`__getitem__(self, the_key)`** which does the following:

- If there is an entry in the hash table with the key, **`the_key`**, then it returns the value associated with **`the_key`**.
- If there is no entry with the key, **`the_key`**, it raises a **`KeyError`** exception.

Write your answer of this part on the next page

Write your answer for Question 10(c) here

8

Blank Page

Question 11 (1 + 1 + 5 = 7 marks)

This question is about *iterators*. Define a **`PositiveIterator`** iterator class which is defined on a list. An instance of this class iterates through all the positive elements of the list.

For example:

```
>>> itr = PositiveIterator([3,-2,-1,5,11])
>>> next(itr)
3
>>> next(itr)
5
>>> next(itr)
11
```

Your class must have the three methods: **`__init__`**, **`__iter__`** and **`__next__`**.

7

Blank Page

Question 12 (5 + 1 = 6 marks)

This question is about *heaps.*

(a) Suppose a **max-heap** is represented using an array. Write a Python function,

```
def is_valid_heap(array)
```

which given an `array` and returns **True** if the **array** represents a **max-heap**, and **False** otherwise.

(b) Explain the worst time complexity of the function you defined in part **(a)**. (*No explanation means no marks*).

6

End of Exam