MONASH University

Office Use Only

Semester One 2014 Examination Period

Faculty of Information Technology

EXAM CODES:	**FIT1008**
TITLE OF PAPER:	**COMPUTER SCIENCE**
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 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 Monash Statute 4.1.

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

<u>**AUTHORISED MATERIALS**</u>

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		**6**
5		**8**
7		**5**
9		**10**
11		**10**
13		**8**

Page		**Mark**
15		**8**
17		**15**
19		**6**
21		**6**

Total:		**82**

Blank Page

Question 1 (2 marks)

This question is about *sorting*. Describe a list of ***N*** items that would be a worst-case input for Bubble Sort. Explain why this is the worst case.

A list with all the items in reverse order.
Every comparison will always result in a swap.

1 mark for example
1 mark for explanation.

Question 2 (2 + 2 = 4 marks)

This question is about *lists and complexity*.

Background definition: The *median* of a list is an item *x* such that half the items are ≤ *x* and half the items are ≥ *x*. So the median of the list 3, 4, 7, 7, 9, 18 is 7. (Do not confuse the *median* with the *average*. The two can be very different.)

Consider a *sorted list* of *N* integers. Explain what is the worst-case complexity of finding the median of the list, in each of the following cases.

a) the list is implemented using an *array*.

The median is half-way along a sorted list. So, if the array has n elements, then the median is the element at index n//2. This requires an int division followed by an array lookup, which takes time O(1) in every case.

1 mark for complexity
1 mark for explanation.

b) the list is implemented as a *linked* list.

To find the element half-way along a linked list of n elements, you need a loop to follow the links from node to node until you have visited n//2 of them. This requires n//2 iterations, each of which takes constant time (an int equality test, following an address to the next node, and controlling the loop counter). So time taken is O(n).

1 mark for complexity
1 mark for explanation.

6

Blank Page

Question 3 (4+2+2 = 8 marks)

This question is about *understanding code and its complexity in BigO*. Consider a stack data type provided by the `Stack` class which is implemented using some data structure (you do not need to know which one) and defines the following methods:

```
pop(self)
push(self, item)
__init__(self)
```

You may assume that the data structure is automatically resizable and therefore *never runs out of space.*

```
def mystery(a_list):
     the_stack = Stack()

     for k in range(len(a_list)):
         the_stack.push(a_list[k])

     for k in range(len(a_list)):
        # HERE
        if (a_list[k] != the_stack.pop()):
            return False
     return True
```

a) Consider the list `a_list = {1,2,3,5,8,6,3,2,1}.` Use the sequence of stacks below to draw the elements on the stack every time the line `# HERE` is reached during the execution of `mystery(a_list)`

1								
2	2							
3	3	3						
6	6	6	6					
8	8	8	8					
5	5	5	5					
3	3	3	3					
2	2	2	2					
1	1	1	1					

1 mark for each stack.

b) Give `a_list` which would cause our `mystery(a_list)` method to return `True`. Explain why (no explanation means no marks).

a_list = [1, 1]. As the method returns True if the list is the same as when it is reversed, and returns False otherwise.

1 mark for example. 1 mark for explanation.

c) Explain the complexity in Big O of our `mystery` method.

O(N), as the first loop iterates through all the items in the list.

1 mark for complexity. 1 mark for explanation.

8

Blank Page

Question 4 (5 marks)

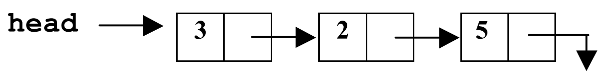
This question is about *basic programming with linked lists*. Consider a **`Node`** class which defines a node for a linked data structure, and which is defined as follows:

```
class Node:
    def __init__(self, item, link):
        self.item = item
        self.next = link
```

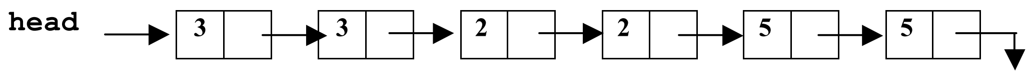
Suppose you have a **`List`** class that implements a linked list using the Node class above and uses the instance variable **`head`** to point to the head of the list. Add the following method to the **`List`** class:

```
    def doubleList(self)
```

which modifies the list pointed to by **`head`**, by duplicating every node in the list. For example, consider the following Linked List:



Then, after calling **`doubleList()`** the list looks like:



Important: the only method you can assume you have access to is **`Node(item, link).`** You will need to write any other Python code you need.

```
def doubleList(self):
    current = self.head

    while current is not None:
      new_node = Node(current.item, current.next)
      current.next = new_node
      current = new_node.next
```

1 mark for working for empty lists.
2 marks for iterating through all the list correctly.
2 marks for duplicating all the nodes, and joining them onto the list correctly.

5

Blank Page

Question 5 (2 marks)

This question is about *Big O complexity*. Consider an algorithm that performs a certain operation on a list of length **N**. Explain why is it wrong to consider the case **N = 0** when determining the best case complexity.
.

Because Big O complexity is all about scalability (i.e., how the time or memory grows as the input grows), thus it always has to assume a sufficiently large (i.e., big) N. Being empty (N=0) is not sufficiently large.

2 marks for explanation.

Question 6 (4 marks)

This question is about *choosing data structures*. Several data types (such as stacks, queues, and lists) can be implemented using arrays or using linked nodes. Explain an advantage and a disadvantage of the array implementation compared to a linked node implementation.

Advantage of array (only need one):

- less memory if nearly full, because for items in an array, you don't need to store links to other items.
- Some operations faster, e.g., search in a sorted list (because you can do binary search for an array, but must use linear search for linked list), or pop a stack, or serve a queue (because there is no need to manipulate and follow links).

Disadvantage of array (only need one):

- more memory if not very full, because max size must be set at creation and cannot be changed later [so array will always occupy same space no matter how much, or how little, is stored].
- some operations slower, e.g., deleting an element from sorted list, since for an array you have to move many other elements to one place away, whereas for linked lists you just have to do a constant number of local changes to links.

2 marks for an advantage and an explanation.
2 marks for a disadvantage and an explanation.

Question 7 (2 + 2 = 4 marks)

Suppose you want to search a list using *binary search*.

a) Explain what data structure would you use.

You would use an array, because you can always use array indices to find the middle element of the current subarray. With linked lists, there is no way of going straight to the middle [or even knowing when you are at the middle] without going through all the nodes before it.

2 marks for an explanation

b) Explain what property the elements of the list must satisfy.

The elements of the list must be in order --- i.e., the list must be sorted. If you compare your target with the middle element, the result needs to tell you which half of the current subarray your element is in --- and this can only be guaranteed if the array is sorted.

2 marks for an explanation

10

Blank Page

Question 8 (5 + 4 + 1 = 10 marks)

This question is about *Binary Search Trees*. Consider the following class for **`TreeNode`**.

```
class TreeNode:
     def __init__(self, item, left, right):
        self.item = item
        self.left = left
        self.right = right
```

a) Suppose you have a **`BinarySearchTree`** class that implements a Binary Search Tree using the **`TreeNode`** class above and uses the instance variable **`root`** to point to the root of the tree. Add to the **`BinarySearchTree`** class a method **`getMin()`** which returns the *minimum* element in the binary search tree.

```
def getMin(self):
    if self.root is None:
        raise Exception("Empty tree")

    current = self.root
    while current.left is not None:
        current = current.left

    return current.left
```

2 marks for dealing with an empty list
2 marks for find the minimum value.
1 mark for returning the value.

b) Explain what is the worst- and best-case complexity of **`getMin()`** in terms of ***N***, the number of items in the tree.

The worst case is that every node only has a left child, so that the nodes form a long chain going down to the left from the root. In this case, every node is visited, with a constant amount of work done at each. For N nodes, this takes time const * N. So complexity is O(N).

Best case is when the root has no left node. Then, we do a constant number of operations: test if root is null, one method call, test if left child is null, if so return the item at the root node. So complexity is O(1).

For each case: 1 mark for complexity. 1 mark for explanation.

c) What would these complexities be if they were expressed in terms of the *Depth* of the tree, rather than ***N*** ?

Worst case: O(Depth)

Best case: O(1)

For each case: 1 mark for complexity.

10

Blank Page

Question 9 (6 + 2 = 8 marks)

A Heap is represented using an array. Suppose the contents of the array are as follows.

	17	12	11	3	9	4	
0	1	2	3	4	5	6	7

a) Use the diagrams below to show the contents of the array at each step of the method for getting the maximum element in the heap (**`getMax`**). You should first show how the array looks just after the maximum element is first replaced. Then show how the array looks after each swap. You may or may not need all the diagrams given.

	4	12	11	3	9		
0	1	2	3	4	5	6	7

	12	4	11	3	9		
0	1	2	3	4	5	6	7

	12	9	11	3	4		
0	1	2	3	4	5	6	7

0	1	2	3	4	5	6	7

2 marks for each correct array.

b) Draw the heap-ordered binary tree that results from applying the **`getMax`** method to our initial heap.

```
        12
       /  \
      9    11
     / \
    3   4
```

2 marks for a correct heap.

8

Blank Page

Question 10 (3+3+2 = 8 marks)

This question is about *hash tables*.

Consider a Hash function which returns the following values for the elements

Game	*Hash Value*
`StarCr`	4
`Thief`	6
`HalfL`	1
`Splinter`	4
`DeusEx`	1
`Doom`	4
`Diablo`	5

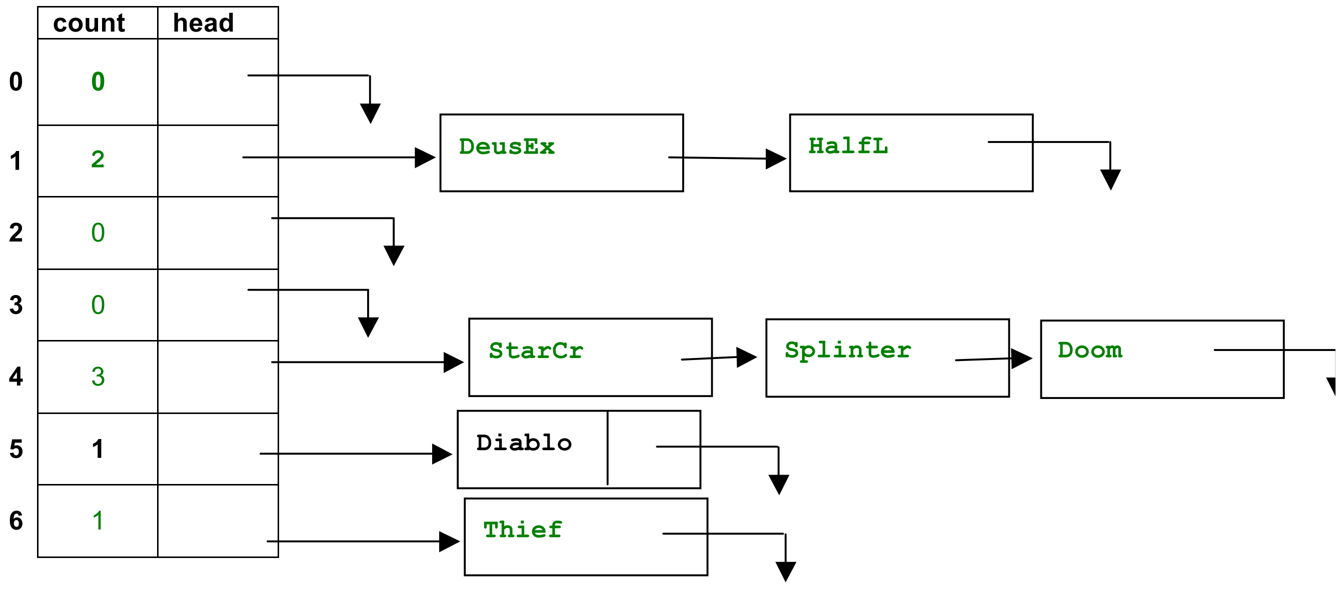
a) Using *linear probing*, insert the above games into the following hash table in the order:

`StarCr, Thief, HalfL, Splinter, DeusEx, Doom, Diablo`

0	1	2	3	4	5	6
`Doom`	`HalfL`	`DeusEx`	`Diablo`	`StarCr`	`Splinter`	`Thief`

0.5 marks for each item in a correct position.

b) Complete the following representation of a hash table that uses *chaining*. Insert the names in the same order as in part **a**.



0.5 marks for each correct chain.

c) Give two characteristics of a good hash function. Explain why they are needed for the function to be good.

Properties: (Only need two)

Returns a value within the table range, so each value corresponds to an index in the table.
Fast, so not to degrade performance.
Minimizes collisions, so that items get spread uniformly throughout the table.

1 mark for each characteristic.

8

Question 11 [10 + 5 = 15 marks]

This question is about *MIPS programming and understanding function calls*.

a) 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 gcd(x, y):`	gcd:
	    addi $sp, $sp, -8
	    sw $fp, 0($sp)
	    sw $ra, 4($sp)
	    add $fp, $0, $sp

`    val = 0`	addi $sp, $sp, -4 sw $0, -4($fp)
`    if (y == 0):`	lw $t0, 12($fp) bne $t0, $0, endif
`        return x`	lw $v0, 8($fp) addi $sp, $sp, 4 lw $ra, 4($fp) lw $fp, 0($fp) addi $sp, $sp, 8 jr $ra

`    val = gcd(y, x % y)`	endif:
	    addi $sp, $sp, -8
	    lw $t0, 12($fp)
	    sw $t0, 0($sp)
	    lw $t1, 8($fp)
	    div $t1, $t0
	    mfhi $t0
	    sw $t0, 4($sp)
	    jal gcd
	    addi $sp, $sp, 8
	    sw $v0, -4($fp)

`    return val`	lw $v0, -4($fp) addi $sp, $sp, 4 lw $ra, 4($fp) lw $fp, 0($fp) addi $sp, $sp, 8 jr $ra

b) Imagine that the `gcd` method defined in part (a) of this question is called from a `main` method with the line `gcd(15, 3)`.

Assume that the `gcd` method is located at 0x00400AC, and that at the time the method is run, `$sp` is 0x7FFE004C and `$fp` is 0x7FFE0050.

Draw what the stack, `$sp`, and `$fp` will look like:

i. the first time the method is entered, immediately after the declaration of `val`;
ii. immediately before the return from the base case.

You do not have to fill in the stack frame for `main`.

i.

Name	Value	Address
		0x7FFE0010
		0x7FFE0014
		0x7FFE0018
		0x7FFE001C
		0x7FFE0020
		0x7FFE0024
		0x7FFE0028
		0x7FFE002C
		0x7FFE0030
		0x7FFE0034
val	0	0x7FFE0038
old $fp	0x7FFE0050	0x7FFE003C
$ra	?	0x7FFE0040
x	15	0x7FFE0044
y	3	0x7FFE0048
Main's local var		0x7FFE004C
Main's stored $fp		0x7FFE0050

$sp	0x7FFE0038
$fp	0x7FFE003C

ii.

Name	Value	Address
		0x7FFE0010
		0x7FFE0014
		0x7FFE0018
		0x7FFE001C
		0x7FFE0020
val	0	0x7FFE0024
old $fp	0x7FFE003C	0x7FFE0028
$ra	?	0x7FFE002C
x	3	0x7FFE0030
y	0	0x7FFE0034
val	0	0x7FFE0038
old $fp	0x7FFE0050	0x7FFE003C
old $ra	?	0x7FFE0040
x	15	0x7FFE0044
y	3	0x7FFE0048
Main's local		0x7FFE004C
Main stored $fp		0x7FFE0050

$sp	0x7FFE0024
$fp	0x7FFE0028

15

Blank Page

Question 12 (4 marks)

Explain what the following fragment of MIPS code does.

```
            addi  $s0, $0, 1
            sw    $s0, x
ALABEL:     lw    $s0, x
            lw    $s1, y
            slt   $t0, $s0, $s1
            beq   $t0, $0, BLABEL
            sll   $s0, $s0, 1
            sw    $s0, x
            j ALABEL
BLABEL:     lw    $s0, x
            addi  $v0, $0, 1
            add   $a0, $s0, $0
            syscall
```

Prints the smallest positive integer power of 2 greater than or equal to y.

4 marks for explanation.

Question 13 (2 marks)

Give two reasons why a programmer might choose to store variables in main memory rather than in registers.

You might need more variables than you have general-purpose registers.
Registers don't persist across function calls unless you save their values in main memory.
Registers are limited in size and so can't store objects such as strings or arrays – the contents of these kinds of objects need to live in main memory.

1 mark for each reason.

6

Blank Page

Question 14 (2+2=4 marks)

This question is about *Binary Search Trees.* Binary Search Trees were designed as an alternative to sorted lists.

a) Explain what is the main advantage of a Binary Search Tree over a sorted list implemented using *arrays*?

You can insert and delete with no shuffling. Thus, the complexity of these operations is O(log N) if the tree is balanced, instead of O(N) for the sorted array (worst case).

1 mark for advantage. 1 mark for explanation.

b) Explain what is the main advantage of a Binary Search Tree with respect to a sorted list implemented using *linked lists*?

You can perform binary search, thus O(logN) for search, add and delete (if the tree is balanced), as opposed to O(N) for linked lists (worse case).

1 mark for advantage. 1 mark for explanation.

Question 15 (2 marks)

This question is about *recursive sorting algorithms.* Consider a version of the *quicksort* algorithm in which the pivot is chosen as the last element of the array. Provide an array of 6 integers which would have worst case complexity in big O notation for this version of quicksort. Explain why your choice would give the worst case complexity (no explanation means no marks).

1 2 3 4 5 6

In fact, you could use any array where the last element is always either bigger than all its predecessors, or smaller than all its predecessors.

Each time the list is partitioned, one of the partitions will be empty.

1 mark for example. 1 mark for explanation.

6

End of Exam