Office Use Only

MONASH University

Semester Two 2018
Examination Period
SAMPLE PAPER
Faculty of Information Technology

EXAM CODES:	FIT1008
TITLE OF PAPER:	Introduction to computer science PAPER 1
EXAM DURATION:	3 hours writing time
READING TIME:	10 minutes

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

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 intentionally left blank, use if needed but it will not be marked.

Important Information

When writing python code, ensure you follow conventions for Python 3.x and avoid using any in built python functions with greater than $O(1)$ complexity.
Any MIPS code you write for this exam must satisfy the following requirements:

- use only instructions and commands listed in the MIPS reference sheet (provided at the end of this exam)
- be a faithful translation of equivalent python code (unless told otherwise)

Write down any assumptions you make.

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

Page	Points	Score
5	6	
7	6	
9	6	
11	4	
13	4	
15	4	
17	6	
19	6	
23	6	
25	6	
27	6	
Total:	60	

This page intentionally left blank, use if needed but it will not be marked.

Question 1: [3 marks]

What must be true of a list to perform binary search? Why?

Question 2: [1 marks]

Why is it not valid to treat $n = 0$ as a best case scenario for some an algorithm's time complexity?

Question 3: [2 marks]

What would happen if a recursive function didn't have a base case? Why?

6

This page intentionally left blank, use if needed but it will not be marked.

Question 4: [2 marks]

What are the ***best*** and ***worst*** case time complexities for:

i Heap sort
ii Quick sort

Question 5: [4 marks]

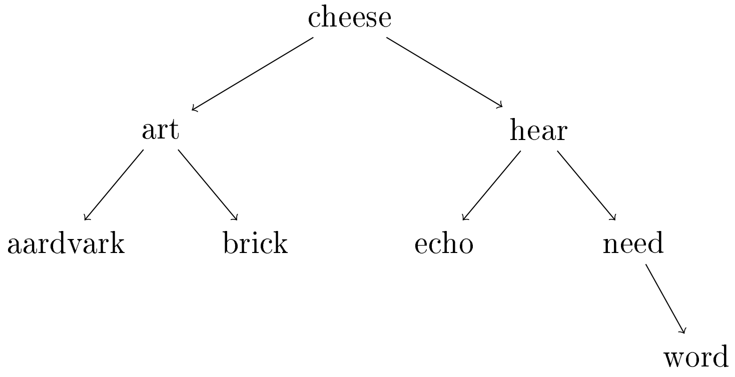
Implement the following faithfully in MIPS:

```
num = 100
while num > 0:
        num = num - 1
print(num)
```

This page intentionally left blank, use if needed but it will not be marked.

Question 6: [2 marks]

Show what would be printed if we output a **post-order** traversal of the binary search tree below:



In the tree above, 'cheese' is the root with left child 'art' and right child 'hear'.
'art' has a left child of 'aardvark' and a right of 'brick' 'hear' has 'echo' and 'need' as its left and right children.
'need' has a right child of 'word'

Question 7: [2 marks]

Why should function arguments be stored in a stack-frame rather than held in registers (i.e. in MIPS)?

Question 8: [2 marks]

*Students of **FIT1008** should answer the following:*

Why is it that when performing a jal command, $ra gets set to PC+4?

This page intentionally left blank, use if needed but it will not be marked.

Question 9: [2 marks]

*Students of **FIT1008** should answer the following:*

Provide a useful test case (input state, expected output and reason) for the __len__ method of an arrayList class.

*Students of **FIT1008** should answer the following:*

Question 10: [2 marks]

Consider the idea of a class to represent a line on the x-y plane. Give an example of a useful attribute and method for this class.

4

This page intentionally left blank, use if needed but it will not be marked.

Question 11: [4 marks]

Consider the ***Mystery*** method (of the *linkedList* class) as defined below.

```
def Mystery(self, k):
        self._mystery_aux(self.head)

def _mystery_aux(self, current):
        S = None
        while not current is None:
                S = current.next
                if not S is None and not S.next is None:
                        S = S.next
                current.next = S
                current = current.next
```

Given the linked list below:

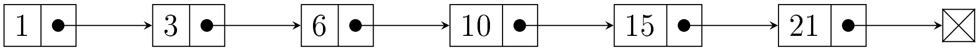


Figure 1: a linked list containing the elements 1,3,6,10,15,21,28,36

Show or describe the effect of ***Mystery*** on this list. What does ***Mystery*** do in general?

4

This page intentionally left blank, use if needed but it will not be marked.

Question 12: [10 marks]

Consider the sequence represented by the following recurrence relation:

$$A(n) = \begin{cases} A(n-1) + 2 \times A(n-2); & n > 1 \\ 1; & n = 0, 1 \end{cases}$$

(a) (4 marks) Create an iterator (A_iter) capable of generating the first k elements of this sequence
For instance, if we ran the following

```
for val in A_iter(6):
    print(val)
```

We would expect the output of `1,1,3,5,11,21`

4

This page intentionally left blank, use if needed but it will not be marked.

(b) (6 marks) Define a recursive **dynamic programming** solution to this problem (*dp_A*) which returns an array *A_table* where A_table[i] = A(i). What benefits does this have over a **naive** recursive implementation?

If you aren't sure how to apply dynamic programming in code, you should explain the approach with a written description (in enough detail that it can be followed) or as pseudocode

6

This page intentionally left blank, use if needed but it will not be marked.

Question 13: [6 marks]

Abby Sträctip is developing an online taxi manager for the hotel chain *RoomE*. The purpose is to allow hotel guests to request a taxi from their hotel room (with time and destination). In the system Abby wants a structure to hold each guest organised so that they can be easily retrieved in order of departure time (with no particular order for taxis departing at the same time). Guests are then added to the structure **when** they make a booking and removed from the structure at the time of their taxi's departure.

Abby is planning for at most 3400 guests to be in the system at once (the maximum capacity of the hotel) but it's likely there will be significantly fewer than this in actually (perhaps a few hundred at once).

(a) (3 marks) What kind of ADT is appropriate in this situation? Justify your answer.

(b) (3 marks) Should this ADT be implemented using an array, nodes, or would either be equally appropriate? Justify you answer.

6

This page intentionally left blank, use if needed but it will not be marked.

Question 14: [6 marks]

Consider the task of computing the ‘width’ of nodes in a binary search tree. In this case, the ‘width’ refers to the difference between the minimum and maximum values in a given subtree. For instance, given a tree as below

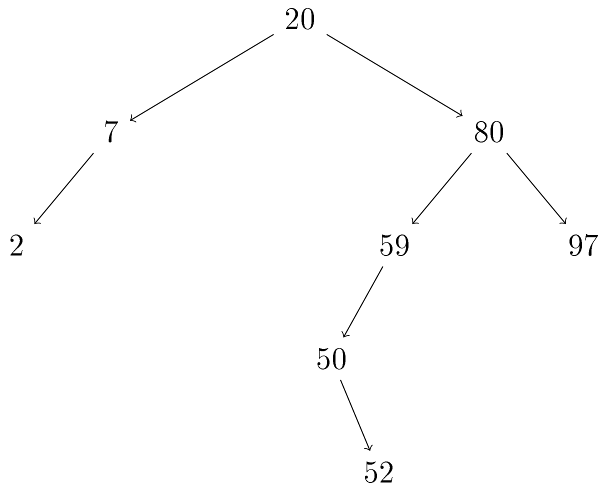


Figure 2: binary search tree containing $20, 7, 80, 2, 59, 97, 50, 52$

The root node (20) would have a ‘width’ of 95 as 2 is the minimum element that is a descendent of 20 and 97 is the maximum element which is a descendent of 20. In the same way, 7 has a width of 5 as the minimum element is 2 but 7 has no larger children so we use the 7 itself. Any nodes with no children have a ‘width’ of 0 as the minimum and maximum element in their subtree is the node itself.

This page intentionally left blank, use if needed but it will not be marked.

(a) (4 marks) Prepare a recursive method ***findWidth*** of the ***Binary Search Tree*** class which determines the 'width' of each node of the tree and assigns this as a property to that node when computed.

(b) (2 marks) What is the **worst case** complexity of the recursive function you prepared in 14(a)? Justify your answer. If you make any assumptions about the shape of the tree, you should mention these in your answer. ***No marks without explanation***

6

This page intentionally left blank, use if needed but it will not be marked.

Question 15: [6 marks]

In managing hash tables, we often refer to both

1. primary clustering and
2. secondary clustering

What are these? Is it possible to reduce the impact of them? Justify your answer.

6

This page intentionally left blank, use if needed but it will not be marked.

Students of ***FIT1008*** *should answer the following:*

Question 16: [6 marks]

Explain in words and/or with the aid of a diagram the best and worst case time complexities of merge sort.

End of Exam

6

This page intentionally left blank.
Answers on this page will not be marked.

MIPS reference sheet for FIT1008

Table 1: System calls

Call code (`$v0`)	Service	Arguments	Returns	Notes
1	Print integer	`$a0` = value to print	-	value is signed
4	Print string	`$a0` = address of string to print	-	string must be terminated with `'\0'`
5	Input integer	-	`$v0` = entered integer	value is signed
8	Input string	`$a0` = address at which the string will be stored `$a1` = maximum number of characters in the string	–	returns if `$a1-1` characters or Enter typed, the string is terminated with `'\0'`
9	Allocate memory	`$a0` = number of bytes	`$v0` = address of first byte	-
10	Exit	-	-	ends simulation

Table 2: General-purpose registers

Number	Name	Purpose
R00	`$zero`	provides constant zero
R01	`$at`	reserved for assembler
R02, R03	`$v0, $v1`	system call code, return value
R04–R07	`$a0--$a3`	system call and function arguments
R08–R15	`$t0--$t7`	temporary storage (caller-saved)
R16–R23	`$s0--$s7`	temporary storage (callee-saved)
R24, R25	`$t8, $t9`	temporary storage (caller-saved)
R28	`$gp`	pointer to global area
R29	`$sp`	stack pointer
R30	`$fp`	frame pointer
R31	`$ra`	return address

Table 3: Assembler directives

.data	assemble into data segment
.text	assemble into text (code) segment
.word w1[, w2, ...]	allocate word(s) with initial value(s)
.space n	allocate n bytes uninitialized, unaligned space
.ascii ”string”	allocate ASCII string, do not terminate
.asciiz ”string”	allocate ASCII string, terminate with `'\0'`

Table 4: Function calling convention

	Caller:	**Callee:**
On function call:	saves temporary registers on stack passes arguments on stack calls function using `jal fn_label`	saves value of `$ra` on stack saves value of `$fp` on stack copies `$sp` to `$fp` allocates local variables on stack

	Callee:	**Caller:**
On function return:	sets `$v0` to return value clears local variables off stack restores saved `$fp` off stack restores saved `$ra` off stack returns to caller with `jr $ra`	clears arguments off stack restores temporary registers off stack uses return value in `$v0`

A partial instruction set is provided below. The following conventions apply.

Instruction Format
Rsrc, Rsrc1, Rsrc2: source operand(s), - must be a register value(s)
Src2; source operand - may be an immediate value or a register value
Rdest: destination, must be a register
Imm: Immediate value, may be 32 or 16 bits (**Imm16**: only 16-bit value)
Addr: Address in the form: offset(Rsrc) ie. absolute address = Rsrc + offset
label: label of an instruction
⋆: pseudoinstruction
Immediate Form -: no immediate form, or this is the immediate form (⋆: immediate form is a pseduoinstruction)
Unsigned form (append 'u' to instruction name):(**-** : no unsigned form, or this is the unsigned form)

Table 5: MIPS instruction set

Instruction format	Meaning	Operation	Immediate	Unsigned
add Rdest, Rsrc1, Src2	Add	Rdest = Rsrc1 + Src2	addi	addu (no overflow trap)
sub Rdest, Rsrc1, Src2	Subtract	Rdest = Rsrc1 - Src2	-	subu (no overflow trap)
mult Rsrc1, Src2	Multiply	Hi:Lo = Rsrc1 * Src2	-	mulu

div Rsrc1, Src2	Divide	Lo = Rsrc1/Src2;
		Hi = Rsrc1 % Src2
			-	divu

and Rdest, Rsrc1, Src2	Bitwise AND	Rdest = Rsrc1 & Src2	andi	-
or Rdest, Rsrc1, Src2	Bitwise OR	Rdest = Rsrc1 | Src2	ori	-
xor Rdest, Rsrc1, Src2	Bitwise XOR	Rdest = Rsrc1 ∧ Src2	xori	-
nor Rdest, Rsrc1, Src2	Bitwise NOR	Rdest = ∼(Rsrc1 | Src2)	-	-
not Rsrc1, Src2 ⋆	Bitwise NOT	Rdest = ∼(Rsrc1)	-	-
sllv Rdest, Rsrc1, Src2	Shift Left Logical	Rdest = Rsrc1 << Src2	sll	-
srlv Rdest, Rsrc1, Src2	Shift Right Logical	Rdest = Rsrc1 >> Src2 (MSB=0)	srl	-
srav Rdest, Rsrc1, Src2	Shift Right Arithmetic	Rdest = Rsrc1 >> Src2 (MSB preserved)	sra	-
move Rsrc1, Src2 ⋆	Move	Rdest = Rsrc	-	-
mfhi Rdest	Move from Hi	Rdest = Hi	-	-
mflo Rdest	Move from Lo	Rdest = Lo	-	-
li Rdest, Imm ⋆	Load Immediate	Rdest = Imm	-	-
la Rdest, Addr (or label) ⋆	Load Address	Rdest = Addr (or label)	-	-
lw Rdest, Addr	Load word	Rdest = mem32[Addr]	-	-
sw Rsrc, Addr	Store word	mem32[Addr] = Rsrc	-	-

beq Rsrc1, Rsrc2, label	Branch if equal	if (Rsrc1 == Rsrc2)
		PC = label
			⋆	-

bne Rsrc1, Rsrc2, label	Branch if not equal	if (Rsrc1 != Rsrc2)
		PC = label
			⋆	-

blt Rsrc1, Rsrc2, label ⋆	Branch if less than	if (Rsrc1 < Rsrc2)
		PC = label
			⋆	unsigned operands

ble Rsrc1, Rsrc2, label ⋆	Branch if less or equal to	if (Rsrc1 ≤ Rsrc2)
		PC = label
			⋆	unsigned operands

bgt Rsrc1, Rsrc2, label ⋆	Branch if greater than	if (Rsrc1 > Rsrc2)
		PC = label
			⋆	unsigned operands

bge Rsrc1, Rsrc2, label ⋆	Branch if greater or equal to	if (Rsrc1 ≥ Rsrc2)
		PC = label
			⋆	unsigned operands

slt Rdest, Rsrc1, Src2	Set if less than	if (Rsrc1 < Src2)
		Rdest = 1
		else Rdest = 0
			slti	sltu

j label	Jump	PC = label	-	-
jal label	Jump and link	$ra = PC + 4; PC = label	-	-
jr Rsrc	Jump register	PC = Rsrc	-	-
jalr Rsrc	Jump and link register	$ra = PC + 4; PC = Rsrc	-	-
syscall	system call	depends on call code in $v0	-	-