2020 Semester Two (November-December 2020) Examination Period

Faculty of Information Technology

EXAM CODES:	FIT1008-FIT2085
TITLE OF PAPER:	Intro to comp science
EXAM DURATION:	3 hours 10 mins

Rules

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.

You must not retain, copy, memorise or note down any exam content for personal use or to share with any other person by any means following your exam.

You must comply with any instructions given to you by an exam supervisor.

As a student, and under Monash University's Student Academic Integrity procedure, you must undertake your in-semester tasks, and end-of-semester tasks, including exams, with honesty and integrity. In exams, you must not allow anyone else to do work for you and you must not do any work for others. You must not contact, or attempt to contact, another person in an attempt to gain unfair advantage during your exam session. Assessors may take reasonable steps to check that your work displays the expected standards of academic integrity.

Failure to comply with the above instructions, or attempting to cheat or cheating in an exam may constitute a breach of instructions under regulation 23 of the Monash University (Academic Board) Regulations or may constitute an act of academic misconduct under Part 7 of the Monash University (Council) Regulations.

<u>Authorised Materials</u>

OPEN BOOK	☐ YES	☑ NO
CALCULATORS	☐ YES	☑ NO
DICTIONARIES	☐ YES	☑ NO
NOTES	☐ YES	☑ NO
SPECIFICALLY PERMITTED ITEMS	☐ YES	☑ NO

if yes, items permitted are:

nstructions

Marks

There are 100 marks in this exam. The exam is worth 60% of the unit mark.

MIPS code:

- All translations from Python to MIPS must be faithful.
- Only the instructions in the MIPS reference sheet (in appendix) are allowed.
- The conventions given in the MIPS reference sheet must be followed.

Python code:

- Unless otherwise specified, do not write type hinting, documentation, assertions or exceptions.
- Write comments if necessary for understanding the code.

Complexity:

- By default, "runtime complexity" refers to worst-case runtime complexity, which we ask you to express using the O( ) notation.

Instructions

Information

Marks

There are 100 marks in this exam. The exam is worth 60% of the unit mark.

MIPS code:

- All translations from Python to MIPS must be faithful.
- Only the instructions in the MIPS reference sheet (in appendix) are allowed.
- The conventions given in the MIPS reference sheet must be followed.

Python code:

- Unless otherwise specified, do not write type hinting, documentation, assertions or exceptions.
- Write comments if necessary for understanding the code.

Complexity:

- By default, "runtime complexity" refers to worst-case runtime complexity, which we ask you to express using the O( ) notation.

Implementations of the Set ADT

Information

We consider the Set ADT studied in the workshop:

```
class Set(ABC, Generic[T]):
    """ Abstract class for a generic Set. """

    def __init__(self) -> None:
        """ Initialization. """
        self.clear()

    @abstractmethod
    def __len__(self) -> int:
        """ Returns the number of elements in the set. """
        pass

    @abstractmethod
    def is_empty(self) -> bool:
        """ True if the set is empty. """
        pass

    @abstractmethod
    def clear(self) -> None:
        """ Makes the set empty. """
        pass

    @abstractmethod
    def __contains__(self, item: T) -> bool:
        """ True if the set contains the item. """
        pass

    @abstractmethod
    def add(self, item: T) -> None:
        """ Adds an element to the set. Note that an element
already
        present in the set should not be added.
        """
        pass

    @abstractmethod
    def remove(self, item: T) -> None:
        """ Removes an element from the set. An exception should be
        raised if the element to remove is not present in the set.
        """
        pass
```

We are interested in two different implementations of this ADT and their complexities.

Question 1

Suppose that we use an implementation of the Set ADT based on a**linked list** (which is not ordered). Give the runtime complexities of each of the class methods of Set for this implementation. No explanation, no marks.

6
Marks

Question 2

Suppose that we use an implementation of the Set ADT based on an**ordered array**, which means that the internal array is kept ordered at all times:

6
Marks

```python
class ASet(Set[T]):
    """Implementation of the set ADT using an ordered array.

    Attributes:
         size (int): number of elements in the set
         array (ArrayR[T]): array storing the elements of the set

    ArrayR cannot create empty arrays. So default capacity value 1
    is used to avoid this.
    """

    def __init__(self, capacity: int = 1) -> None:
        """ Initialization. """
        Set.__init__(self)
        self.array = ArrayR(max(1, capacity))
```

Give the runtime complexities of each of the class methods of Set for this implementation. No explanation, no marks.

Hash Tables

Question 3

Consider a hash table of size **tablesize**=11, with the hash function:

5
Marks

hash(key) = key % tablesize

Starting from an empty hash table, the keys **11,9, 7, 63, 13, 40, 33, 5, 39, 50** are inserted in the table in that order. Using Linear Probing as the method of collision resolution, and the hash function shown above, write the content of the hash table as a list. Separate the keys using commas and denote an empty slot using quotation marks (""), for instance [x, "", y, ...]. You must also explain each insertion step by step. **No explanation, no marks.**

Sorting

Question 4

Describe Selection Sort in a few sentences. How can we make Selection Sort stable? Give a precise description. How does this affect the worst-case runtime complexity? Give a proof. **No proof, no marks.**

6
Marks

Question 5

Describe Quicksort in a few sentences. What is the best and worst case runtime complexity of Quicksort? Give a proof of both of these. **No proof, no marks.**

6
Marks

Heaps
Question 6

What are the advantages and disadvantages, if any, of an implementation of a heap that uses an array, rather than a binary tree made of linked nodes?

4 Marks

Question 7

Consider the partial implementation of a max heap, as seen in the lessons:

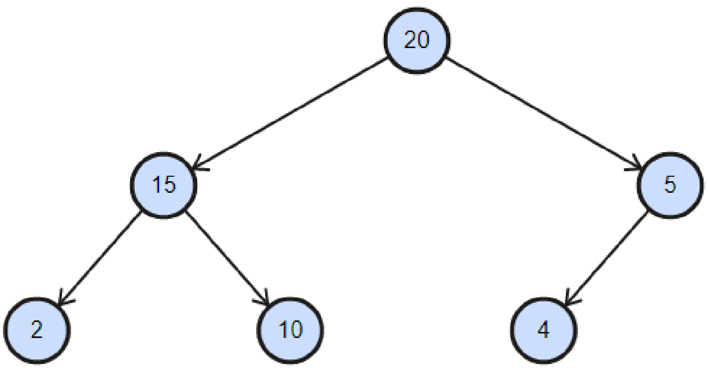
6 Marks

```python
from typing import Generic
from referential_array import ArrayR, T


class Heap(Generic[T]):
    MIN_CAPACITY = 1

    def __init__(self, max_size: int) -> None:
        self.length = 0
        self.the_array = ArrayR(max(self.MIN_CAPACITY, max_size) + 1)

    def __len__(self) -> int:
        return self.length
```

Write a recursive method called **postorder_list**() inside the **Heap** class above that returns a list of the content of the heap binary tree in postorder traversal. Recall that in a postoder traversal of a binary tree,

- First, the left subtree is traversed recursively in postorder
- Second, the right subtree is traversed recursively in postorder
- Third, the current node is processed (in our case, this simply means that the value at the node is printed).

For example, it will return [2, 10, 15, 4, 5, 20] for the following **Heap** instance:



Iterators

Question 8

Write an **iterator** to generate the Fibonacci sequence. Recall that the Fibonacci sequence is 1,1,2,3,5,8,13,21, ...

8

Marks

Use the template:

```
class FibIterator:
    def __init__(self) -> None:

    def __iter__(self) -> FibIterator:

    def __next__(self) -> T:
```

Scoping

Question 9

In this question you are tasked to determine what is printed by a code. Each **print** instruction prints a Python variable that refers to a function. For example, the code

5
Marks

```
1 class Mystery:
2     def f(self):
3         print(f)
4
5 f = Mystery()
6 f.f()
```

prints 1 Python object, which is the function defined at line 2 (remember that in Python, functions are objects!). For the code below, write which functions (referring to them by the line at which they are defined) are printed, in the correct order. Justify each answer. No justification, no mark.

```
 1 class Mystery:
 2     def f(self):
 3         def g():
 4             print(f)
 5         def h():
 6             def f():
 7                 print(f)
 8             f()
 9         g()
10         h()
11
12 f = Mystery()
13 f.f()
```

Stack ADT and sorting

Question 10

8
Marks

The problem consists in sorting a stack with the help of an auxiliary stack, and no other container. Recall that the ADT of a Stack is:

```python
class StackADT(ABC, Generic[T]):
    def __init__(self) -> None:
        self.length = 0

    @abstractmethod
    def push(self, item: T) -> None:
        """ Pushes an element to the top of the stack."""
        pass

    @abstractmethod
    def pop(self) -> T:
        """ Pops an element from the top of the stack."""
        pass

    @abstractmethod
    def peek(self) -> T:
        """ Pops the element at the top of the stack."""
        pass

    def __len__(self) -> int:
        """ Returns the number of elements in the stack."""
        return self.length

    def is_empty(self) -> bool:
        """ True if the stack is empty. """
        return len(self) == 0

    @abstractmethod
    def is_full(self) -> bool:
        """ True if the stack is full and no element can be pushed. """
        pass

    def clear(self):
        """ Clears all elements from the stack. """
        self.length = 0
```

Given one input stack and one temporary stack, write a Python program that sorts the input stack, using the temporary stack (no other containers) and without calling Python's sorting methods in any way. Comment your code. For reference, there is a solution that has fewer than 12 lines (excluding comments).

The Bisection Algorithm

Information

We now attempt to answer a few questions related to the bisection method for finding the square root of a number x, which we denote x½. We provide the description of the recursive version of this algorithm here.

The input of this algorithm is:

- x, a real number >= 0 that we must find the root of.
- l, a lower bound on x½, i.e. l <= x½. Also l >= 0.
- u, an upper bound on x½, i.e. x½ <= u. Also l <= u.
- e, a numerical tolerance, i.e. the output y of the algorithm should satisfy |y-x½| <= e.

The bisection algorithm relies on the assumption that the root that we are looking for, x½, is in the interval [l, u] at the start of the algorithm. It recursively divides the interval by 2, and selects the half in which x½ is located by adjusting the values of l and u, until the interval is small enough to satisfy |u-l| <=e, at which point u can be output with the guarantee that |u-x| <= e.

A Python implementation of the bisection algorithm that uses recursion to compute x½ is:

```python
def bisection_rec(x, l, u, e):
    # base case
    if u - l <= e:
        return u

    # compute the middle point of the interval [l,u]
    m = (u+l)/2
    # compute its square
    s = m*m

    # check how to divide the interval
    if s >= x:
        u = m
    else:
        l = m

    # recurse
    return bisection_rec(x, l, u, e)
```

Make sure that you understand this algorithm as it will be used in a few questions. The questions are independent of each other and can be attempted in any order.

Examples of calls and returned values are given below:
bisection_rec(2, 0, 2, 0.0001) -> 1.41424560546875
bisection_rec(2, 0, 2, 0.1) -> 1.4375
bisection_rec(4.0, 0, 4.0, 0.0001) -> 2.0

Question 11

The Python code we have provided does not have type hinting, documentation or assertions (in this question we ignore exceptions). We provide the original code again for convenience:

8
Marks

```python
def bisection_rec(x, l, u, e):
    # base case
    if u - l <= e:
        return u

    # compute the middle point of the interval [l,u]
    m = (u+l)/2
    # compute its square
    s = m*m

    # check how to divide the interval
    if s >= x:
        u = m
    else:
        l = m

    # recurse
    return bisection_rec(x, l, u, e)
```

Based on the description of the algorithm and the code itself, add**type hinting**, **documentation** (description, pre- and post-conditions) and **assertions** to match. Do not add exceptions, but for the purpose of this question add assertions where you may normally add an exception.

Question 12

Extend the function we have provided to handle an extra argument n and to compute and return the n^th root of x (rather than the square root). We provide the original code again for convenience:

3
Marks

```python
def bisection_rec(x, l, u, e):
    # base case
    if u - l <= e:
        return u

    # compute the middle point of the interval [l,u]
    m = (u+l)/2
    # compute its square
    s = m*m

    # check how to divide the interval
    if s >= x:
        u = m
    else:
        l = m

    # recurse
    return bisection_rec(x, l, u, e)
```

Question 13

In this question we want to determine the runtime complexity of the bisection algorithm. We provide the original code again for convenience:

5
Marks

```python
def bisection_rec(x, l, u, e):
    # base case
    if u - l <= e:
        return u

    # compute the middle point of the interval [l,u]
    m = (u+l)/2
    # compute its square
    s = m*m

    # check how to divide the interval
    if s >= x:
        u = m
    else:
        l = m

    # recurse
    return bisection_rec(x, l, u, e)
```

We denote L = (u - l) the length of the search interval and N = L/e the number of intervals corresponding to the base case.

Express the worst-case runtime complexity of this algorithm as a function of N. Justify your answer.

Question 14

In this question we ask that you write the iterative version of the recursive bisection. We provide the original code again for convenience:

8
Marks

```python
def bisection_rec(x, l, u, e):
    # base case
    if u - l <= e:
        return u

    # compute the middle point of the interval [l,u]
    m = (u+l)/2
    # compute its square
    s = m*m

    # check how to divide the interval
    if s >= x:
        u = m
    else:
        l = m

    # recurse
    return bisection_rec(x, l, u, e)
```

Give a direct translation using the template provided.

Question 15

Faithfully translate into MIPS the (modified) bisection function. Do not translate the body of the original function. Only translate the code below:

16
Marks

```
1 def bisection_rec(x, l, u, e):
2 
3     #the body is empty. Nothing to translate here.
4 
5     # recurse
6     return bisection_rec(x, l, u, e)
```

Recall that in MIPS, a recursive call can be translated as any other function call.**Start the translation with a stack diagram** written as comments at the point of line 3's execution (the translation assumes no body). The clarity of the MIPS code you write will be assessed together with its correctness and faithfulness.

Appendix

Information

MIPS reference sheet for FIT1008 and FIT2085

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 of uninitialized, unaligned space
.ascii "string"	allocate ASCII string, do not terminate
.asciiz "string"	allocate ASCII string, terminate with '\0'

Table 4: Function calling convention

On function call:	**Caller:** saves temporary registers on stack passes arguments on stack calls function using `jal fn_label`	**Callee:** saves value of \$ra on stack saves value of \$fp on stack copies \$sp to \$fp allocates local variables on stack
On function return:	**Callee:** 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`	**Caller:** clears arguments off stack restores temporary registers off stack uses return value in \$v0

Table 5: A partial instruction set is provided below. The following conventions apply.

Instruction Format column
Rsrc, Rsrc1, Rsrc2: register source operand(s) - must be the name of a register
Rdest: register destination - must be the name of a register
Addr: address in the form `offset(Rsrc)`, that is, absolute address = `Rsrc + offset`
label: label of an instruction
******: pseudoinstruction
Immediate Form column
Associated instruction where **Rsrc2** is an immediate. Symbol - appears if there is no immediate form.
Unsigned or overflow column
Associated unsigned (or overflow) instruction for the values of **Rsrc1** and **Rsrc2**. Symbol - if no such form.

Table 6: Allowed MIPS instruction (and pseudoinstruction) set

<table>
<tr><th>Instruction format</th><th>Meaning</th><th>Operation</th><th>Immediate</th><th>Unsigned or Overflow</th></tr>
<tr><td>add Rdest, Rsrc1, Rsrc2</td><td>Add</td><td>Rdest = Rsrc1 + Rsrc2</td><td>addi</td><td>addu (no overflow trap)</td></tr>
<tr><td>sub Rdest, Rsrc1, Rsrc2</td><td>Subtract</td><td>Rdest = Rsrc1 - Rsrc2</td><td>-</td><td>subu (no overflow trap)</td></tr>
<tr><td>mult Rsrc1, Rsrc2</td><td>Multiply</td><td>Hi:Lo = Rsrc1 * Rsrc2</td><td>-</td><td>mulu</td></tr>
<tr><td>div Rsrc1, Rsrc2</td><td>Divide</td><td>Lo = Rsrc1/Rsrc2;
Hi = Rsrc1 % Rsrc2</td><td>-</td><td>divu</td></tr>
<tr><td>and Rdest, Rsrc1, Rsrc2</td><td>Bitwise AND</td><td>Rdest = Rsrc1 & Rsrc2</td><td>andi</td><td>-</td></tr>
<tr><td>or Rdest, Rsrc1, Rsrc2</td><td>Bitwise OR</td><td>Rdest = Rsrc1 | Rsrc2</td><td>ori</td><td>-</td></tr>
<tr><td>xor Rdest, Rsrc1, Rsrc2</td><td>Bitwise XOR</td><td>Rdest = Rsrc1 ∧ Rsrc2</td><td>xori</td><td>-</td></tr>
<tr><td>nor Rdest, Rsrc1, Rsrc2</td><td>Bitwise NOR</td><td>Rdest = ~(Rsrc1 | Rsrc2)</td><td>-</td><td>-</td></tr>
<tr><td>sllv Rdest, Rsrc1, Rsrc2</td><td>Shift Left Logical</td><td>Rdest = Rsrc1 << Rsrc2</td><td>sll</td><td>-</td></tr>
<tr><td>srlv Rdest, Rsrc1, Rsrc2</td><td>Shift Right Logical</td><td>Rdest = Rsrc1 >> Rsrc2
(MSB=0)</td><td>srl</td><td>-</td></tr>
<tr><td>srav Rdest, Rsrc1, Rsrc2</td><td>Shift Right Arithmet.</td><td>Rdest = Rsrc1 >> Rsrc2
(MSB preserved)</td><td>sra</td><td>-</td></tr>
<tr><td>mfhi Rdest</td><td>Move from Hi</td><td>Rdest = Hi</td><td>-</td><td>-</td></tr>
<tr><td>mflo Rdest</td><td>Move from Lo</td><td>Rdest = Lo</td><td>-</td><td>-</td></tr>
<tr><td>lw Rdest, Addr</td><td>Load word</td><td>Rdest = mem32[Addr]</td><td>-</td><td>-</td></tr>
<tr><td>sw Rsrc, Addr</td><td>Store word</td><td>mem32[Addr] = Rsrc</td><td>-</td><td>-</td></tr>
<tr><td>la Rdest, Addr(or label) **</td><td>Load Address (for printing strings)</td><td>Rdest=Addr (or Rdest=label)</td><td>-</td><td>-</td></tr>
<tr><td>beq Rsrc1, Rsrc2, label</td><td>Branch if equal</td><td>if (Rsrc1 == Rsrc2)
PC = label</td><td>-</td><td>-</td></tr>
<tr><td>bne Rsrc1, Rsrc2, label</td><td>Branch if not equal</td><td>if (Rsrc1 != Rsrc2)
PC = label</td><td>-</td><td>-</td></tr>
<tr><td>slt Rdest, Rsrc1, Rsrc2</td><td>Set if less than</td><td>if (Rsrc1 < Rsrc2)
Rdest = 1
else Rdest = 0</td><td>slti</td><td>sltu , sltiu</td></tr>
<tr><td>j label</td><td>Jump</td><td>PC = label</td><td>-</td><td>-</td></tr>
<tr><td>jal label</td><td>Jump and link</td><td>$ra = PC + 4;
PC = label</td><td>-</td><td>-</td></tr>
<tr><td>jr Rsrc</td><td>Jump register</td><td>PC = Rsrc</td><td>-</td><td>-</td></tr>
<tr><td>jalr Rsrc</td><td>Jump and link register</td><td>$ra = PC + 4;
PC = Rsrc</td><td>-</td><td>-</td></tr>
<tr><td>syscall</td><td>system call</td><td>depends on the value of <code>$v0</code></td><td>-</td><td>-</td></tr>
</table>