Monash University
Faculty of Information Technology

Semester Two 2017 – Mid-Semester Test

EXAM CODES:	**FIT2085**
TITLE OF PAPER:	**INTRODUCTION TO COMPUTER SCIENCE FOR ENGINEERS**

THIS PAPER IS FOR STUDENTS STUDYING AT: (office use only - tick where applicable)

Berwick ☐	Clayton ✓	Peninsula ☐	Distance Education ☐	Open Learning ☐
Caulfield ☐	Gippsland ☐	Malaysia ☐	Enhancement Studies ☐	Other (specify) ☐

Candidates must complete this section

STUDENT ID __ __ __ __ __ __ __ __

SURNAME ....SIGNATURE...

OTHER NAMES (in full) ....

Candidates are reminded that they should have no material on their desks unless their use has been specifically permitted by the following instructions.

AUTHORISED MATERIALS

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

if yes, items permitted are:

INSTRUCTIONS TO CANDIDATES
1. Print your name and ID number in the section above.
2. Answer all questions in the space provided.
3. The duration of the test is **50 minutes.**
4. Total marks for this test are 40.
5. Individual marks are indicated for each question.
6. Calculators are **not** permitted.
7. **Candidates must NOT remove this paper from the examination room.**

Do not open this paper until you are instructed to do so.

Official use only

Question		Marks
1		8
2		7
3		8
4		10
5		7
Total:		40

Question 1 (8 marks)

This question is about Big-O time complexity. For each of the following fragments of code, give the **worst** time complexity using Big-O notation, assuming `the_list` has length `n` and its elements are strings of size `m`. Provide an explanation (no explanation, no marks). Note that function `range(n)` generates a sequence of integers from `0` to `n-1`, while function `range(k,n)` generates a sequence of integers from `k` to `n-1`.

```
a) def code1(the_list):
       for i in range(len(the_list)):
           for j in range(i+1, len(the_list)):
               if the_list[i] == the_list[j]:
                   print(i,j)
```

The outer loop always executes `n` times. The inner loop executes first `n` times, then n-2, then n-3, and so on until it executes once, so $(n^2-n)/2$ times in total. Printing integers `i` and `j` is assumed to be constant, but the complexity of `==` usually depends on the size of the strings. Thus, it is not just O(n^2). Assuming the big O complexity of == is O(`m`), we have a big O for `code1` of O(n^2*m). The loops always execute the same amount of times regardless of the elements in the list, so best = worst.

```
b) def code2(the_list, string):
       i = 0
       while i < len(the_list) and string < the_list[i]:
           i = i + 1
       return i
```

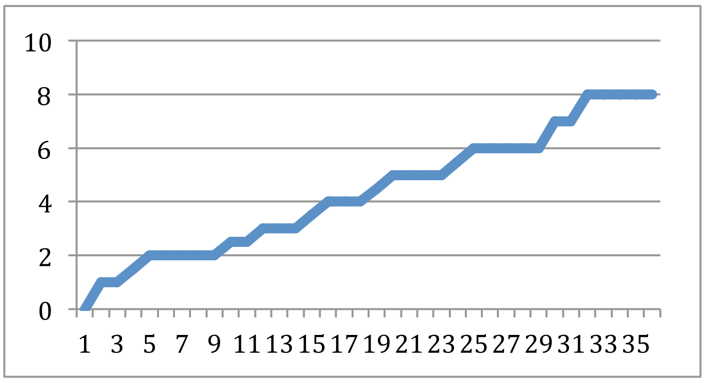
The only loop executes at most `n` times, less if it finds a string in `the_list` that is less or equal than `string`. All operations are constant except the comparison between `string` and the string elements in `the_list`, which in general depends on the size of the strings (the comparison with `i` is a comparison among integers and, thus, assumed to be constant). Thus, the best complexity is O(`m`) when the first string is already less or equal than `string`, and O(`n*m`) if all strings are greater than `string`.

```
c) def code3(the_list):
       n = len(the_list)
       return (n > 0 and n%2==0 and the_list[0] == the_list[n-1])
```

All operations are constant except for the comparison between the two strings, which will usually depend on `m`. Assuming a big `n` (as we always do for bigO complexity) `n> 0` will always be true but `n%2==0` might fail when the length of the list is odd. If so, the comparison will not be executed yielding a best case of O(1). When the length is even (`n%2 !=0)` and worst is O(`m`),

This question was not done well. As the chart shows, 15 out of 36 students (41.7%) failed it. Given it is a well-known question that only needs proper reasoning, it is clear many students need to practice this. Coming to consultation would be a good thing…

.



Question 2 (7 marks)

This question is about *exceptions* and *assertions*. The program below can be improved to check its precondition and take care of input errors (by asking the user to re-input the number until it is correct). We ask that you do so by using appropriate assertion(s) and exception(s) handling. Note however that you do not need to check the type of the parameter of `inverse()`. Recall that the function `int()` throws an exception of the type `ValueError` if an improper parameter is passed.

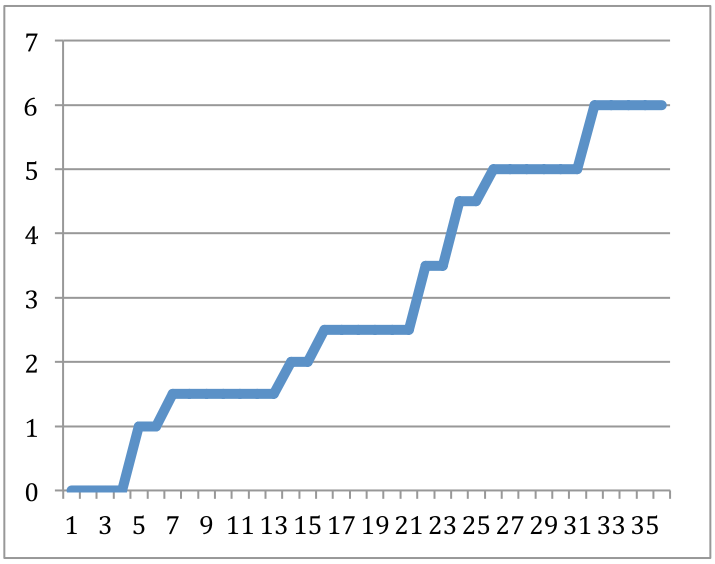
```
    #Takes a non-zero number as parameter and returns the inverse value
    def inverse(a):
        return 1/a

    #Reads an integer number n>0 and prints the result of inverse(n)
    def main():
        n = int(input("Input a non-negative integer number: "))
        print(inverse(n))
```

```
def inverse(a):
    assert a != 0, "inverse: argument must be non-zero"
    return 1/a

def main():
    correct_input = False
    while not correct_input:
        try:
            n = int(input("Input a non-negative integer number: "))
            if n > 0:
                correct_input = True
            else:
                print("The integer you entered was not > 0")
        except ValueError:
            print("The input you entered was not an integer")
    print(inverse(n))
```

This was the question where students performed the worst. As the chart shows, 23 out of 36 students (63.9%) failed it. Given it was the only question brand new (a question of this kind had never appeared in a test before), it suggests students are studying to pass the test (by simply studying previous tests), not to learn (that is, going through the material ensuring they understand everything and can comfortably answer questions about it). The exam will include many brand new questions, so I would suggest changing the approach.



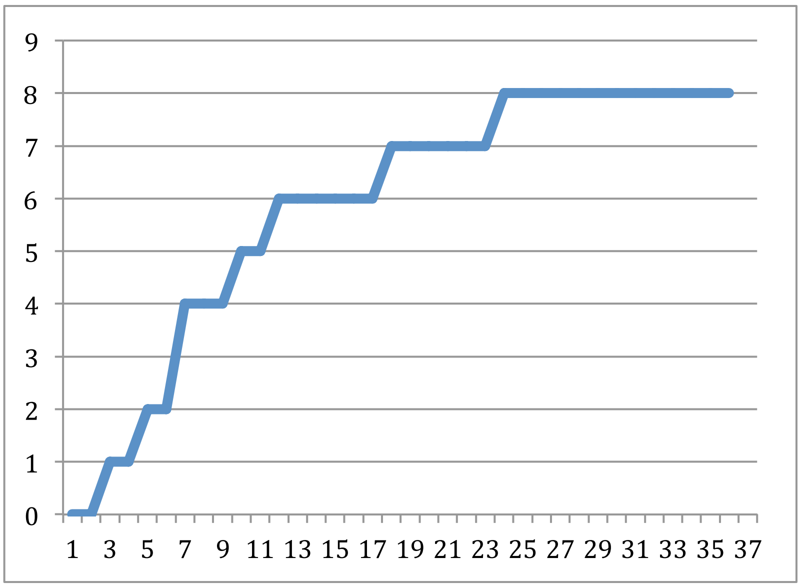
Question 3 (8 marks)

Consider the following memory diagram of the layout of the stack and the heap at a point in the life of function **`f`**, which is about to call function **`g(a, b)`**, where **`a`** and **`b`** are local variables of **`f`**. Assume function **`g(a, b)`** receives in **`a`** an array of integers, in **`b`** and integer, and has a local integer variable **`temp`** which is initialized to **`a`**.

Use the empty spaces for the heap, stack and registers appearing on the right hand side to draw the memory diagram at the point once **`g(a, b)`** has been called and the local variable **`temp`** has been allocated and initialized. Make sure you name the registers and memory cells appropriately and provide adequate values for its contents. Assume also that the return address to **`f`** is **`0x00401234`**.

	Name	Contents	Address		Contents	Address
Heap	a.length	1	0x10014F20	a.length	1	0x10014F20
	a[0]	4	0x10014F24	a[0]	4	0x10014F24
			0x10014F28			0x10014F28
Stack			0x7FFEFFE4	temp	0x10014F20	0x7FFEFFE4
			0x7FFEFFE8	$fp	0x7FFF0000	0x7FFEFFE8
			0x7FFEFFEC	$ra	0x00401234	0x7FFEFFEC
			0x7FFEFFF0	arg1(a)	0x10014F20	0x7FFEFFF0
			0x7FFEFFF4	arg2(b)	5	0x7FFEFFF4
	b	5	0x7FFEFFF8	b	5	0x7FFEFFF8
	a	0x10014F20	0x7FFEFFFC	a	0x10014F20	0x7FFEFFFC
		■	0x7FFF0000		■	0x7FFF0000
Regs	$fp	0x7FFF0000		$fp	0x7FFEFFE8	
	$sp	0x7FFEFFF8		$sp	0x7FFEFFE4	

This question was well answered, as shown by the chart. Only 6 students (16.7%) failed. However, it is not surprising, given how easy the question is and the fact a similar question was in last year's Mid Semester test.

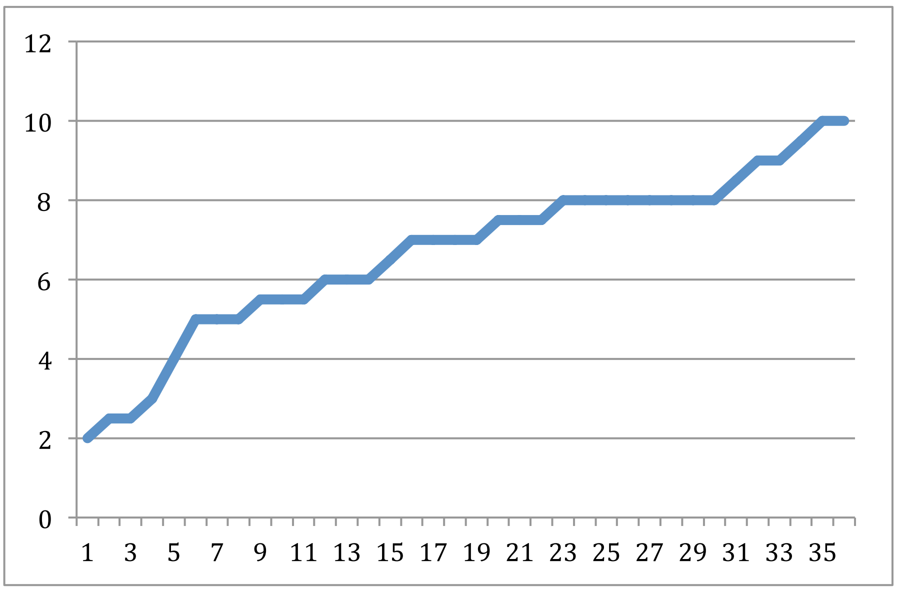


Question 4 (10 marks)

This question is about *MIPS programming*. Assume **`i`** and **`a`** are local variables located at **`-4($fp)`** and **`-8($fp)`**, respectively, with **`a`** being an array of integers. In the space next to this part of a Python code, complete the MIPS program so that it constitutes a ***faithful*** translation of the original Python program (no need to write a full MIPS program). **Remember to comment the code and use only the instruction provided in the MIPS reference sheet provided at the end of the test.**

Python code	MIPS code	MIPS comments
`if  i <= 0:`	`lw   $t0, -4($fp)`	`# load i`
	`slt  $t1, $0, $t0`	`# $t1=1 if 0<i,and=0 if i<=0`
	`bne  $t0, $0, else`	`# if i>0 go to else, if i<=0`
		`cont`
	`lw  $t0, -8($fp)`	
	`lw  $t1, -4($fp)`	`# load a's address`
	`sll $t1, $t1, 2`	`# load i`
`    print(a[i])`	`add $t1, $t1, $t0`	`# i*4 bytes`
	`lw  $a0, 4($t1)`	`# address of a[i]-4`
		`# load a[i] into $a0 to print`
	`addi $v0, $0, 1`	
	`syscall`	`#print integer i`
	`j endif`	
		`# jumper over else`
`else:`	`else:`	
	`lw   $a0, -4($fp)`	
	`addi $v0, $0, 1`	`#load i`
	`syscall`	`#print integer i`
`    print(i)`		
	`endif:`	
		`# finish`

This question was also well answered, as shown by the chart. Again, only 6 students (16.7%) failed, although this time not many students got everything right. Again, the relative good performance is not surprising, given the fact a similar question was in last year's Mid Semester test.

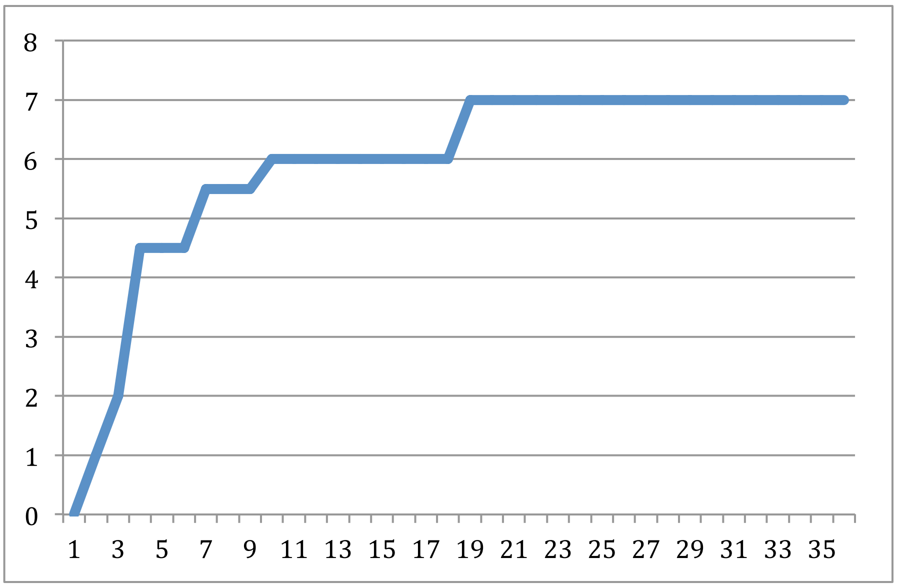


Question 5 (7 marks)

This question is about *MIPS programming.* In the space next to the MIPS code, provide the Python code associated to this (uncommented) code. As we have done in the lectures, assume you have a **`read(i)`** function in python that reads an integer and assigns it to variable **`i`**. I have left space for MIPS comments; they are not needed to get full marks, but might help you to add them.

MIPS code	MIPS comments	Python code
`    .data`		
`g:  .word 0`	`# assign 0 to global g`	`g = 0`
`    .text`		
`main:`	`# start main function`	`def main():`
`    addi $v0, $0, 5`	`# read integer`	`    read(g)`
`    syscall`		`    while g > 0:`
`    sw $v0, g`	`# store it in g`	`        print(g)`
		`        g -= 1`
`loop:`	`# while loop`	
`    lw $t0, g`	`# load g into $t0`	
`    slt $t1, $0, $t0`	`# if g > 0, $t1=1`	
`    beq $t1, $0, end`	`# if not g>0 go to end`	
`    addi $v0, $0, 1`	`# print(g)`	
`    add $a0, $t0, $0`		
`    syscall`		
`    addi $t0, $t0, -1`	`# $t0 = $t0 -1`	
`    sw $t0, g`	`# g = $t0`	
`    j loop`	`# back to while`	
`end:`		
`    addi $v0, $0, 10`		
`    syscall`		

As shown by the associated chart, this was the question students did best, with only 3 students (8.4%) failing. The most common mistake was not defining main (I ended up not counting this as a mistake), and defining the loop condition wrongly.



END OF TEST