MONASH University

ter Two 2019

nation Period

Faculty of Information Technology

EXAM CODES:	FIT1008
TITLE OF PAPER:	Introduction to computer 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.

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, or a breach of instructions under Part 3 of the Monash University (Academic Board) Regulations.

<u>Authorised Materials</u>

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

if yes, items permitted are:

Instructions

Please answer all questions.

This exam is worth 60% of your overall mark.

To answer a question that requires a code response use 2 spaces to represent each indentation level. Do not use the Tab key to indent, as this will not indent and instead move you to the next field.

Multiple Choice

Question 1

This question is about hash tables with open addressing.

A hash table of size 10 uses hash function **hash(key) = key % 10** and linear probing to handle collisions. After inserting 6 keys into an empty hash table, the cor[...]s as follows: [None, None, 32, 13, 54, 22, 46, 53, None, None]

Which one of the following choic[...]er in which the key values could have been inserted in the table?

Select one:

- a. 46, 32, 54, 22, 13, 53
- b. 54, 32, 13, 22, 53, 46
- c. 46, 54, 32, 13, 22, 53
- d. 32, 46, 53, 13, 54, 22
- e. 53, 46, 22, 54, 13, 32
- f. None of the above

Only c is a possible ordering. For 22 to end up in slot 5, 32, 13 and 54 must already have been inserted. Similarly, for 53 to end up in slot 6, it must have been inserted last.

Question 2

Consider the array representation of **Heap** class as seen in the lectures. Which one of the following arrays does **not** represent a **max heap**.

Select one:

- a. [None,9,5,6,3,4,1]
- b. [None,11,9,3,5,6,1]
- c. [None,8,6,5,3,2,2]
- d. [None,15,6,10,2,7,8]
- e. [None,11,7,9,5,6,8]
- f. None of the above

d is not a valid max-heap. Node 2 (with priority 6) has a child with priority 7, which violates the heap condition

Question 3

This question is about MIPS. Assume you want to translate to MIPS the Python condition

```
if x <= y:
```

where both **x** and **y** are global variables. Which of the following pieces of MIPS code correctly translates the Python condition? Indicate what each line of the selected code does, or (if "None of the above") provide the correct code.

a)

```
lw  $t0,  x
lw  $t1,  y
slt  $t2,  $t0,
$t1 beq $t2,  $0,  endif
```

b)

```
lw $t0,  x
lw  $t1,  y
slt  $t2,  $t0,  $t1
bneq $t2,  $0,  endif
```

c)

```
lw $t0,  x
lw  $t1,  y
slt  $t2,  $t1,  $t0
bne $t2,  $0,  endif
```

d)

```
lw  $t0,  x
lw  $t1,  y
slt  $t2,  $t1,  $t0
beq $t2,  $0,  endif
```

e)

```
slt  $t2,  x,  y
beq $t2,  $0,  endif
```

f)

```
slt  $t2,  x,  y
bne $t2,  $0,  endif
```

g)

```
slt  $t2,  y,  x
beq $t2,  $0,  endif
```

h)

```
slt  $t2,  y,  x
bne $t2,  $0,  endif
```

i)

None of the above.

The correct solution is c. The first two instructions load x into $t0 and y into $t1, respectively. The next instruction tests whether y < x, setting $t2 to 1 if true, and 0 if false. The last instruction exists the if then else if y<x is true (i.e., if not x <=y) by checking if $t2 is 1, which means not equal to 0.

Marking guide:

- For a, b, c, and d,
  - 0.5 for correctly saying what lw does in the first two lines
  - 0.5 for correctly saying what the slt does
  - 0.5 for correctly saying what the last instruction does
  - 1.5 for correctly selecting c

Question 4

This question is about MIPS. The following piece of MIPS code translates the Python array access

x = the_list[i-1]

where **x**, **the_list** and **i** are all global variables. The code is correct if **i>0**. This might seem strange, as the MIPS code does not include an instruction to subtract 1 from index **i**. Explain why this is not needed when no marks).

```
lw $t0, i
lw $t1,the_list
addi $t2, $0,
4
mult $t0, $t2
mflo $t0
add $t0,
$t0,$t1
lw $t0, ($t0)
SW $t0, X
```


We do not need to subtract 1 from index `i` because the penultimate instruction (`lw $t0, ($t0)`) did not add 4 to the address of the element to jump over the length, that is, it was not the usual `lw $t0, 4($t0)`. Therefore the code is accessing one element less than `i`. This is only a problem when the index `i` is 0, since then it will be accessing the length of the array. But in such case, the python code should not have been called either.

Marking guide:

2/2 marks for talking about usually adding 4 to jump over the length of the array
(or explicitly mentioning size is first element and skipping size of array)

-0.5 for saying [address+0] is the address, and not the size of the list

1/1 mark to point to the instruction that adds the 4 usually
(or explaining the shifting of indices due to no +4 offset ($t0) and concluding subtract 1 is not needed for i)

Question 5

This question is about sorting. Consider the following piece of code:


```
def some_sort(the_array):
    n = len(the_array)
    for p in range(n)
       tmp = the_arra
       q=0
       while the_arra
           q += 1
       while q <= p:
            r = the_a
            the_array[q] = tmp
            tmp = r
        q += 1
```


Is this a correct sorting algorithm? If so, explain the invariant that ensures correctness; if not, give an example where it fails.

Yes, it is a correct sorting algorithm. The invariant it maintains is that after each iteration, the first p elements of the list are in sorted order (like insertion sort). The while loop identifies the position where the_array[p] should be inserted: everything left of q is strictly less than the item, and everything between q and p is greater than or equal to. Then the second loop shuffles all the elements in range [p, q) one step to the right, inserting the new item into the empty space.

Marking guide: (8 marks)

- 1 mark for saying it is a sorting algorithm
- 2 marks for correctly identifying the invariant
- 2 marks for explaining how the destination is found
- 3 marks for explaining the insertion

2 mark for saying it is a sorting algorithm (1.5 if not given, but assume so)
3 marks for correctly identifying the invariant
1 marks for explaining how the destination is found
2 marks for explaining the second while loop (the insertion)

Common issues:

- state that it is not a sorting lgorithm
- For those saying YES:
    - do not understand what an invariant is
    - invariant is not complete, fairly vague.
    - fail to explain what the first and second while loops do?

Suggestion:

To ask students to explain each task in the code if saying TRUE

Question 6

Given the following implementation of Quick Sort, implement the partition function, choosing the last element of the list as pivot.


```
def quick_sort(array):
  start = 0
  end = len(array) -
  quick_sort_aux(arra

def quick_sort_aux(ar                    d):
  if start < end:
        boundary =                     y, start,
     end)
        quick_sort_aux(array, start,
     boundary - 1)
        quick_sort_aux(array, boundary +
     1, end)
```



```
def partition(array, start, end):
    mid = end
    pivot = array[mid]
    array[start], array[mid] = array[mid], array[start]
    index = start

    for k in range(start + 1, end + 1):
        if array[k] < pivot:
            index += 1
            array[k], array[index] = array[index], array[k]

    array[start], array[index] = array[index], array[start]
    return index
```

Marking guide (7 marks):

- 1 mark for choosing the correct pivot
- 1 mark for walking over the array, (total 4)
  - +1 for swapping when < pivot
  - +1 for updating index
  - +1 for handling boundary cases
- 1 mark for swapping the pivot into position
- 1 mark for returning pivot index

IF sorting not done in place (MAX 5 marks)

Alternate version (starting from both sides) is also fine; same marking scheme, but with the corresponding condition for swapping.

Not doing in place SWAPS (MAX 5)
1/1 correct pivot selection
1/1 walking over the array
1/1 for handling boundary cases
1/1 updating the original array
1/1 returning pivot index

Data Structures

Question 7

This question is about Data Structures.

Consider a **List** class implemented using arrays, whose partial implementation is as follows:

```
def __init__(self, si
    if size <= 0:
        raise ValueEr                 d be positive")
    self.array = [Non
    self.count = 0

def size(self):
    return self.count

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

def is_full(self):
    return self.size() >= len(self.array)
```



```
def __contains__(self, item):
    for i in range(self.count):
        if self.array[i] == item:
            return True
    return False

## ALSO OKAY:
def __contains__(self, item):
    return item in self.array[:self.count]
```

Marking guide (4 marks):

- 2 marks for iterating over the array, comparing elements
- 1 mark for correct boundary condition (self.count not len(self.array))
- 1 mark for correct return (returning False at the end, not e.g. inside the loop)

/2 iterating through the array and comparing
-0.5 for not returning from infinite loop
-0.5 for more than 1 indentation error

/1 boundary for using self.count or self.size() (*self.size is ok)
-0.5 for using self.count-1 or self.size()-1
-0.5 for using count and not self.count
-0.5 for using size and not self.size()

/1 return, setting it outside loop

Question 8

Consider a **List** class implemented using arrays, whose partial implementation is as follows:

```
def __init__(self, size):
    if size <= 0:
        raise ValueError("Size should be positive")
```

```
        self.array = [None] * size self.count
    = 0

    def size(self):
        return self.count

    def is_empty(self):
        return self.

    def is_full(self):
        return self.                    array)
```

Add also to the **List** class an im   d

`remove_first(self)`

which removes and returns the element at index 0, ensuring all other elements are correctly swapped to the left. You should appropriately account for any errors.

```
def remove_first(self):
    if self.count == 0:
        raise IndexError("Cannot remove first from empty list.")
    item = self.array[0]
    self.count -= 1
    for i in range(self.count):
        self.array[i] = self.array[i+1]
    return item
```

Marking guide (7 marks):

- 1 mark for updating count
- 1 mark for returning correct item
- 4 marks for correctly shuffling elements to the left (swap is okay)
    - o -1 mark for missing first/last item
- 1 mark for correctly handling empty list

Marks

Question 9

The queue consists of items on stack.left (in stack order), followed by items on stack.right in reverse order.

Append pushes items onto the top of the self.right. For serve, if self.left is empty but self.right is not, it pops each element off self.right, and [illegible] (reversing the order). In either [illegible] top off self.left. Less efficient: T[illegible] element from self.left and push [illegible] push the new item, and reverse [illegible] serve, we pop the top of self.lef[illegible] where append pushes, and serve does move-pop-move).

Marking guide (6 marks):

- 2 marks for explaining representation (how the queue elements are stored)
- 2 marks for explaining operation of append
    - -1 for poor explanation, or missing boundary cases
- 2 marks for explaining operations of pop
    - -1 for poor explanation, or missing boundary cases

Linked Lists

Question 10

D, E, F and any linked list with non negative values are all possible outputs after running mystery. The first loop drops any nodes at the start of the list [illegible] g self.head. Then the second loop steps through the remaining nodes, unlinking any containing [illegible] h, only lists containing non-negative values can be returned.

Alternative answer: Assuming t [illegible] npty before calling mystery function, the correct answer is option F, the list remains empty after call [illegible] both while and if statement wont be executed since self.head is None.

Marking guide (8 marks):

- 2 marks for identifying one or more correct results
  - -1 mark for identifying one or more correct results but also stating the other correct results as wrong
- 1 mark for talking about negative values (not including zero) as the final results
- 2 marks for talking dropping initial values (total 3)
  - + 1 for talking about updating head
  - -1 for incomplete/unclear explanation
- 2 marks for talking about second loop removing following non negative items
  - -1 mark for incomplete/unclear explanation.
- 0 marks for totally wrong explanation even with correct result option.

For alternative answer:

- 8 marks for correct and valid reasoning that no effect to the list because the list is empty
  - -4 marks for incorrect reasoning.

Recursion

Question 11

This question is about Recursion.

You are visiting a friend, who has 1008/FIT2085, for the weekend. But when you arrive, your friend is nowhere to be found; and instead of their door ou a pair of numbers, and the following clue:

```
def clue(x, y):
    if y == 0:
        return 1
    elif y % 2 ==
        y = clue(x, y-1)
        return x * y else:
    y = clue(x, y//2)
        return y * y
```


You may assume all inputs are **positive integers**.

Write the result of the function **clue** for the input values

x = 100, y = 1

100

Question 12

16

Question 13

8

Question 14

32

Question 15

You are visiting a friend, who has reviously completed FIT1008/FIT2085 for the weekend. But when you arrive, your friend is nowhere to be found; and instead of their door code, they [illegible] a pair of numbers, and t[illegible]:


```
def clue(x, y):
  if y == 0:
    return 1
  elif y % 2 == 1:
    y = clue(x, y-1)
    return x * y
  else:
    y = clue(x, y//2)
    return y * y
```


You may assume all inputs are **positive integers**.

What is the worst-case time complexity of **clue** (using big-O time complexity)? Explain your answer (no explanation, no marks).

Worst-case complexity of clue(x, y) is O(log(y)).
Complexity of a call body is O(1). At each step, y halves if y is even. If y is odd, it decreases by 1, so y is even in the next iteration.

Marking guide (3 marks):

- 1 mark for log(y), if an explanation is given
- 1 mark for talking about y being halved
- 1 mark for including the odd case

Binary Search Trees

Question 16

Binary Search Trees

Question 16

9 Marks

This question is about Bina

Complete the missing expre um_leq, which recursively computes the sum of all elements in a BST tree whic al to **item**.

```
def sum_leq(self, it
    return self.sum_                    item)

def sum_leq_aux(self, current, item):
    if tree is None:
        return #1
    else:
        low_sum = #2
        if tree.item == item:
            return #3
        elif #4
          return low_sum
        else:
          return #5
```


1. 0
2. sum_leq(current.left, item)
3. low + current.item
4. item < current.item
5. low + current.item + sum_leq(current.right, item)

Marking guide (9 marks):

1. 1 mark for 0
2. 2 marks for correct recursive call
3. 1 mark for correct addition
4. 2 marks for correct inequality
5. 2 marks for recursive call, +1 for correct additions

Question 17

Question 17

2
Marks

The following method either returns the sum of the items in the tree that are **ancestors** of the node whose item is **value** in ... ne if **value** is not present.

```
def sum_parents(nod
    if node is None
        return None
    elif node.item
        return 0
    elif value < node.item:
        below = sum_parents(node.left, value)
    else:
        below = sum_parents(node.right, value)
    if below is None:
        return None
    else:
        return below + node.item
```


What is the worst-case complexity of **sum_parents**? (Remember to define any variables you use). For what kind of trees does this occur?

O(n), where n is the number of nodes in the tree. This happens when the tree is very unbalanced (a stick) so has depth O(k*n), and value is at the deepest node. (Alternatively, it is O(k*d), where d is the depth of the tree.)

Marking guide (2 marks): 1 mark for correct complexity (0.5 for missing the compare), 1 for mentioning unbalanced trees (or any tree, if depth).

Question 18

Complete the tail-recursive function **sum_parents_tail**, which is computes the same value as **sum_parents**


```
def sum_parents_tail(node, value):
    return sum_parents_tail_aux(node, value, 0)
def sum_parents_tail_aux(node, value, acc):
    if node is None:
        return #1
    elif node.item ==
        return #2
    else:
        #3
```

1. None
2. acc
3. return sum_parents_tail_aux(node.left if value < node.item else node.right, value, acc + node.item)

For #3, it is also acceptable to add use an if-then-else, rather than a ternary if:

```
if value < node.item:
   return sum_parents_tail_aux(node.left, value, acc + node.item)
else
   return sum_parents_tail_aux(node.right, value, acc + node.item)
```


Marking guide (5marks):

- 1 mark for None
- 1 mark for acc
- 1 mark for being correctly tail recursive, 1 mark for correct child, 1 for correct accumulator in call(s).
- -0.5 for missing return.
- -0.5 if the statement appears in wrong order for example #2 written as an answer for #1.
- -0.5 if the condition is flipped, example (if value > node.item then go left) which should go right.

Heaps

Question 19

This question is about heaps.

Consider an implementation of a **MaxHeap**, which provides the following methods:

__init__(self), which creates
add(self, item), which adds a
get_max(self), which remove t value item from the Heap.

Write a function **find_kth_sma** the **kth smallest** item in array **alist** (with 1 being the smallest). You must start your implementation with the fo


```
def find_kth_smallest(alist, k):
    mx = MaxHeap()
    n = len(alist)
    # TODO
```


You cannot use any additional MaxHeaps or other data structures.


```
find_kth_smallest(alist, k):
    index = n - k
    if(index < 0)
        raise IndexError("Tried to get the kth smallest in a list with fewer than k elements.")

    for(item in alist):
        mx.add(item)
    for(_ in range(index)):
        mx.get_max()
    return mx.get_max()
```

Marking guide (8 marks):

- 1 mark for raising an exception on k > n.
- 2 marks for adding all items into the max heap
- 2 marks for removing some number of items from the heap
    - +2 for removing (n-k), only +1 if some error (off by one, etc.)
- 1 for returning the last item

Question 20

It will print “a should be greater or equal to b”. The mystery2(3, 4) calls mystery1(3, 4), and the assert statement fails. At that point, the assertion fails [illegible]r(“a should be greater or equal to b”). Control immediately passes to the catch block in mystery2. [illegible]ssertions of type MyError, so control continues up to the top level. The AssertionError matches the [illegible] catch block, and the attached message is printed.

Marking guide (4 marks):

- 1 for ‘a should be great[illegible]lanation given
    - 0 for saying it c[illegible]n error *and* prints ‘a should be greater or equal to b’
    - 0.5 for referring to the correct error in mystery1 but mistakenly using the wrong error message
- 1 for talking about the raised assertion
- 1 for talking about skipping the inner catch
    - 0.5 for not explaining why the inner catch is skipped
    - 0.5 for recognising inner catch can’t handle an assertion error (but thinking it crashes)
    - 0 for not explicitly mentioning that it is skipped
- 1 for talking about matching the outer catch
    - 0.5 for not specifying the exception in the outer catch