CS1010X — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

25 March 2023

Time allowed: 2 hours

Student No: | A | 0 | 2 | 6 | 9 | 6 | 4 | 5 | M |

Instructions (please read carefully):

1. Write down your student number on the **question paper**. DO NOT WRITE YOUR NAME ON THE QUESTION PAPER!
2. This is **an open-book test**.
3. Please switch off your mobile phone, and no laptop and no calculator. If found using any of these equipment any moment from now till your scripts have been collected, you will be given a zero mark.
4. This paper comprises **FIVE (5) questions** and **TWELVE (12) pages**. The time allowed for solving this test is **2 hours**.
5. The maximum score of this test is **32 marks**. The weight of each question is given in square brackets beside the question number.
6. All questions must be answered in the space provided in this question paper. If you write something important as your answer outside the space provided, please make clear with some marking. At the end of the test, please submit your question paper only. No extra sheets will be accepted as answers.
7. You are allowed to use pencils. Please be sure your handwriting is legible and neat, and you have proper indentation as needed for your Python codes to avoid misinterpretation.

GOOD LUCK!

Question	Marks
Q1	6
Q2	4
Q3	2.5
Q4	2
Q5	4+3
Total	21.5

Question 1: Warming Up [6 marks]

A. For the following Python code, write the output of the `print` statement. [2 marks] 2

```
def f(a, b):
    a = 10
    def g(a, c):
        return a + b + c
    return g

print( f(11,9)(1,8) )
```

f(11, 9) → g(1, 8) : 1 + 9 + 8 = 18

Ans:

18 ✓

B. For the following Python code, write the outputs (2 lines in total) of the `print` statements. [2 marks] 2

```
t = (1, 2, 3)
s = ("a", "b", "c")
def fn(t, s):
    a = t
    b = s
    t = ( a, b )
    s = [ t ]
    return s

t = fn(s, t)
print( t )
print( s )
```

fn(("a","b","c"), (1,2,3))

a = ("a","b","c")

b = (1,2,3)

t = (("a","b","c"), (1,2,3))

s = [(("a","b","c"), (1,2,3))]

return s

Ans:

[(("a","b","c"), (1,2,3))] ✓

("a","b","c") ✓

C. Study the following codes. [2 marks] 2

```
def g(n, m):
    if n > m:
        return n * m
    else:
        return f(m, n)

def f(n, m):
    if n < m or m == 0:
        return m * n
    else:
        return f(n%m, m)
```

m >= n

now n >= m

```
def gcd(a,b)
  if b==0:
    return a
  else:
    return gcd(b, a%.b)
```

when n<m

What can you claim about the outputs of the function `g(n, m)` when the inputs `n, m` are positive integers with `0 < n < m`? Please first give your output in terms of the original inputs `n` and `m`, and then explain your reasoning.

Ans:

When $0 < n < m$ and n, m are positive integers for $g(n,m)$, $g(n,m)$ will call $f(m,n)$, which will call $f(m\%.n, n)$ in turn. In the call $f(m\%.n, n)$, the base case will be called as $(m\%.n) < n$, hence the final answer is $n * (m\%.n)$.

Question 2: Something familiar [6 marks]

A. The following function `bigger_sum` is supposed to take in three numbers and return the sum of the squares of the two larger numbers. [2 marks]

```
def sum_of_squares(a, b):
    return a * a + b * b

def bigger_sum(a, b, c):
    if a < b and a < c:
        return sum_of_squares(b, c)
    elif b < a and b < c:
        return sum_of_squares(a, c)
    else:
        return sum_of_squares(a, b)
```

should also apply if a=b, a<c

Returns the sum of the larger 2 nums among a, b, and c.

c<a and c<b

check equality

Suppose the inputs `a`, `b`, `c` to the function `bigger_sum` are valid (and there is no syntax error for the codes), state whether the above implementation actually can or cannot produce the correct output for the stated objective. If can, please explain the logic of the program; if cannot, please provide a simple counter-example to show that the output is wrong.

Ans:

It cannot produce the output.

If a=b and a<c, ~~the~~ bigger_sum should return sum_of_squares(b,c) as c is the greatest number and the value of a and b would be the 2nd largest number. However, ~~this if branch~~ [or elif] this will not fulfil the condition in the if branch, hence the else branch will be taken and sum_of_squares(a,b) will be returned.

2

B. Given the below codes, you are to write out the output of each given `print` statement. you do not need to calculate out the number explicitly. That is, you can leave your answer as the power of some other number (that we can enter into a simple calculator to get the answer too).

```
def addF(f, g):
    return lambda x: f(x) + g(x)

def compose(f, g):
    return lambda x: f(g(x))

x = lambda x: x**x
y = lambda y: y**y

print( compose(x,y)(3) )
```

(lambda x: (g(x))^2)(3) →

[2 marks]

Ans:

$(3^3)^2$

2

```
print( compose( addF(x, y), addF(addF(y, x), x)) (2) )
```

f(2) + g(2)

[2 marks]

Ans:

20

0

f(2) + g(2) where f → addF(x,y), g → addF(addF(y,x),x)

↓

addF(x,y)(2) + addF(addF(y,x),x)(2)

↓ ↓

(2*2)+(2**2) + addF(2^2+2+2, 2^2)(2)

8 + 12

Question 3: Not really unfamiliar [7 marks]

Study the following codes carefully.

```
def create(t, m):
    n = len(t)
    while n > 1:
        s = ()
        for i in range(0, n, m):
            minX = t[i]
            for j in range(i+1, min(i+m, n)):
                if t[j] < minX:
                    minX = t[j]
            s = s + (minX,)
        n = len(s)
        t = s
        print(t)
    return t
```

Give the output of the following `print` statement. Your answer should have 4 separate lines. [2 marks]

```
print(create( (11, 9, 10, 6, 7, 8, 5, 4, 3, 2, 1), 3 ))
```

Ans:

(9, 7, 3, 1) ✗

(9, 7, 3) ✗

(9, 7) ✗

(9, 1), 3, 1) ✗

With the above tracing of the `print` statement, we want to understand the time and space complexity of the codes in terms of the length `n` of the input tuple `t`. Note that the input `m` is always a positive integer between 2 and 9, and `n` is much larger than `m`.

What is the space requirement of the codes? State your answer first, and then provide your reasoning. [2 marks]

Ans:

Space requirement is O(n). ✓

In the worst case, the tuple s will have the same number of elements as the number of iterations in the first for loop (for i in range(0, n, m):). Since the for loop runs $\frac{n}{m}$ times in the worst case, s will be $\frac{n}{m}+1$ elements long in the worst case. Other operations in the function occupy O(1) space. Hence, space complexity is $O(\frac{n}{m}+1) \Rightarrow O(n)$.

What is the time requirement of the codes? State your answer first, and then provide your reasoning. [3 marks]

Ans:

Time requirement is $O(n^3)$. X (~~for j in range...~~)

In the worst case, the innermost for loop will run in $O(n)$ time (worst case: i=0, min(i+m, n) = n)

The enclosing for loop (for i in range...) runs in $O(n)$ time in the worst case as it iterates $\frac{n}{m}+1$ times at most, hence it runs in $O(\frac{n}{m}+1) \rightarrow O(n)$ time.

The enclosing while loop, however, runs in $O(n)$ time in the worst case (when m=1).

0.5

no mention of length / magnitude addition

Hence, overall time complexity is $O(n*n*n) = O(n^3)$

Question 4: The Familiar Counting of Squares? [3 marks]

There was a counting squares in last year's midterm; we are going to use it to solve this year's question. Below is the repeat of what appeared in last year's midterm together with its one possible solution. You can skip the following to read directly our question after the program `num_sq_given_grid_of(n)`.

Given a square grid of size n by n, we want to count the total number of squares of size 1 by 1 (there are $n \times n$ of them), size 2 by 2 (depending on n), and so on until size of n by n (just 1 such big square).

For a 1 by 1 grid, there is just 1 square in the grid.

For a 2 by 2 grid, there are 4 squares of size 1 by 1, and 1 square of size 2 by 2, for a total of 5 squares in the grid.

For a 3 by 3 grid, there are 9 squares of size 1 by 1, 4 squares of size 2 by 2, and 1 square of size 3 by 3, for a total of 14 squares in the grid.

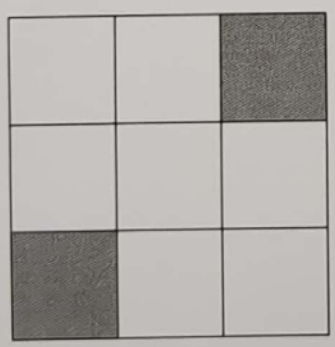
For a 4 by 4 grid, there are in total 30 squares we can find from the following Python codes.

```
def num_sq_given_grid_of(n):
    if n==1:
        return 1
    else:
        undercounted = 0
        for i in range(1, n+1):
            undercounted = undercounted + 2*(n-i)+1
        return undercounted + num_sq_given_grid_of(n-1)
```

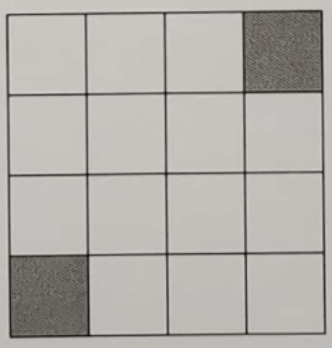
total	n	1×1 sq	2×2 sq	3×3	4×
1	1	1			
5	2	4	1		
14	3	9	4	1	
30	4	16	9	4	1
55	5	25	16	9	4

Our question this year is to count the number of squares for a $n \times n$ grid with a missing 1×1 grid at the lower left hand corner, and a missing 1×1 grid at the upper right hand corner. So, our grid is of size at least 3 by 3, then 4 by 4 and so on as follows:

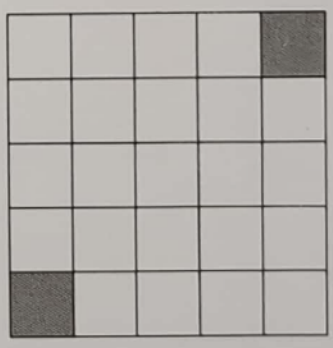
2 lesser 1×1



3 × 3 grid



4 × 4 grid



5 × 5 grid

	Total (n^2-2)	n	1×1	2×2	3×3	4×4
1	0	1	0			
5	2	2	2_4			
14	9	3	7_9	2_4		
30	23	4	14_{16}	7_9	2_4	
55	46	5	23_{25}	14_{16}	7_9	2_4

For a 3 by 3 grid with the two missing corners, there are 7 squares of size 1 by 1, 2 squares of size 2 by 2, and no square of larger sizes due to the missing corners.

For a 4 by 4 grid with the two missing corners, there are now 14 squares of size 1 by 1, 7 squares of size 2 by 2, and 2 squares of size 3 by 3.

For a 5 by 5 grid with the two missing corners, what is the total number of squares we can find? [1 marks]

Ans: 34 ✗ ①

Now you can write a program to do the count on the total number of squares for a grid of size n by n with the mentioned missing corners. Your program MUST make call(s) to the program `num_sq_given_grid_of` given in the above. That is, no credit will be given if you write a completely new (and correct) codes to solve the problem without using `num_sq_given_grid_of`. [2 marks]

```
def num_sq_given_grid_with_missing_corners (n):
#
# make sure you call num_sq_given_grid_of(n) in your answer
#
    if n == 1:
        return 0
    else:
        return num_sq_given_grid_of(n) \
               -(2*(n-1)+1)
```

55-(

✓ ②

Question 5: A (slight) Different in Higher Order Function [10 marks]

You will be working with the following higher-order function `sumN` which is different from those you have encountered so far in our course material (such as `sum`).

```
def sumN(t, term, next, i, default):
    if t==() or i >= len(t) or i < 0:
        return default
    else:
        return term(t, i) + sumN(t, term, next, next(i), default)
```

only this parameter changes after each recursive call.

A. The function `alt_sum_sq` takes an input tuple `t` (of numbers) to compute the alternating sum of the square of each term in `t`. That is, the output is:

$$\sum_{i=0}^{len(t)-1} (t[i])^2 \cdot (-1)^i$$

Example execution:

```
>>> alt_sum_sq( (6,7,8) )   # = 6*6 - 7*7 + 8*8
51
>>> alt_sum_sq( (6,7,8,9) )   # = 6*6 - 7*7 + 8*8 - 9*9
-30
```

We can define `alt_sum_sq` with `sumN` as follows:

```
def alt_sum_sq( t ):
    return sumN( t, <T1>, <T2>, <T3>, <T4> )
```

Please provide possible lambda functions for `<T1>` and `<T2>`, and constants for `<T3>` and `<T4>`. We reserve the right to give no credit if your `<T1>` and `<T2>` do not match up to give a correct thinking to solve the problem. For example, no credit could be given when either `<T1>` or `<T2>` is left unanswered. [3 marks]

term: (Tup[int], int) -> int

`<T1>`: [1 marks]

```
lambda t,i : (t[i]**2)-(t[i+1]**2) if len(t)-1>=2 else
```

t[i] ** 2 ✗

next: (int) -> int

`<T2>`: [1 marks]

```
lambda i : i+2
```

✓

i: int

`<T3>`: [0.5 marks]

```
0
```

✓

default: int

`<T4>`: [0.5 marks]

```
0
```

✓

(base)

2

What are the time and space complexities of your version of `alt_sum_sq`? State your answers, and then provide your reasoning. No credit can be given if your `<T1>`, `<T2>`, `<T3>`, `<T4>` are far from correct. Also, we reserve the right to deduct some small credit should your time and space complexities are not matching up to the best that one can do with the given `sumN`; that is, your `<T1>`, `<T2>`, `<T3>`, `<T4>` may be correct but not as good. [2 marks]

Ans:
Time: O(n), Space: O(n)

Taking n to be the length of the tuple, a single call of sumN within alt_sum_sq would take O(1) time to run. The depth of the recursion is n calls, hence the total time complexity is O(1*n) = O(n).

Space complexity is O(n) since the maximum recursion depth is n calls.

2

B. Here is your familiar `reverse(t)` function:

```
def reverse(t):
    if t == ():
        return ()
    else:
        return (t[-1],) + reverse(t[:-1])
```

We can actually define `reverse(t)` with `sumN` as follows:

```
def reverse( t ):
    return sumN( t, <T5>, <T6>, <T7>, <T8> )
```

Please provide possible lambda functions for `<T5>` and `<T6>`, and constants for `<T7>` and `<T8>`. We reserve the right to give no credit if your `<T5>` and `<T6>` do not match up to give a correct thinking to solve the problem. For example, no credit could be given when either `<T5>` or `<T6>` is left unanswered. [3 marks]

term: Tup['a'] → Tup['a']

<T5>: [1 marks]

```
lambda t, i : (t[-i],)
```

X

next: (int) → int

<T6>: [1 marks]

```
lambda i : i+1
```

i : int

<T7>: [0.5 marks]

```
1
```

X

default; base

<T8>: [0.5 marks]

```
( )
```

1.5

What are the time and space complexities of your version of `reverse(t)`? State your answers, and then provide your reasoning. No credit can be given if your `<T5>`, `<T6>`, `<T7>`, `<T8>` are far from correct. Also, we reserve the right to deduct some small credit should your time and space complexities are not matching up to the best that one can do with the given `sumN`; that is, your `<T5>`, `<T6>`, `<T7>`, `<T8>` may be correct but not as good. [2 marks]

Ans:

Time: $O(n^2)$ ✓ Space: $O(n^2)$ X

Taking n to be the length of the tuple, each single call of sumN within reverse will take O(n) time, and space in each call. term(t,i) will produce a tuple and will be concatenated to the result of sumN(t, term, next, next(i), default) which is also a tuple. Since tuple concatenation takes O(n) time and space, each call of sumN also takes up $O(n^2)$ time and space. Since the recursion depth is n calls, the total work done will be n^2 and thus the total time complexity will be $O(n^2)$. Since space accumulates in the same way, the total space complexity will also be $O(n^2)$.

1.5

— END OF PAPER —