National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester II, 2022/2023

Midterm Practice Questions Part 2

The questions are of varying difficulty and are meant to reinforce your fundamentals. Please make sure that you try work on the questions in IDLE, and not just try to write code on paper (or worse, in your head). This is important. **Simply thinking or writing out the solution and concepts involved on paper is not likely to be sufficient to help prepare you for the midterm and practical exams!**

We hope that this set of questions will be helpful for your revision on the material for the first half of the course. Good luck!

Higher-order Functions

1. **[Basic, Concrete Abstraction]** Write a higher-order function `exception_function` which will return a function with exceptions. `exception_function` should take in a function f(x), an integer input and an integer output, and return another function g(x). g(x) should be same to f(x), except that when x is the same as the input integer, the output will be returned.

   For example, in the code below, given that we have a function `usually_sqrt` which returns the square root of the argument. Using `exception_function` we obtain `new_sqrt`, which behaves similarly to `usually_sqrt` except when called `new_sqrt(7)`, where the value of `2` will be returned.

   The test case is given below, and if the program is implemented correctly in IDLE, you will expect the outputs stated.

   ```
   new_sqrt = exception_function(usually_sqrt, 7, 2)
   new_sqrt(9) => 3.0
   new_sqrt(16) => 4.0
   new_sqrt(7) => 2
   ```

2. Write a function `usually_double` which doubles its argument. Using the functions `usually_double` and `exception_function`, create a function `new_double`, which returns double the argument except when given the argument `4`, `7`, and `11`, in which it returns the value `0`.

Order of Growth

3. **[Intermediate, Lecture Notes]** Compute the time and space complexity ie. O(n) order of growth for `move_tower` to solve the Towers of Hanoi problem as discussed in lecture. The code is attached below for your reference. *Hint: Try to compute the complexity for 2 discs, then 3 discs, and try to generalise it for n to n+1.*

```
def move_tower(size, src, dest, aux):
    if size == 1:
        print_move(src, dest)
    else:
        move_tower(size-1, src, aux, dest)
        print_move(src, dest)
        move_tower(size-1, aux, dest, src)

def print_move(src, dest):
    print("move top disk from ", src," to ", dest)
```

4. **[Basic]** Give the simplified big O notations for the five expressions below, and arrange them in increasing order of growth.
   (a) O($5n^4 + 2n^2 - n$)
   (b) O($3^n + ln(n)$)
   (c) O($n! + 8n^2$)
   (d) O($n\sqrt{n}$)
   (e) O($log_2 20n$)

5. **[Basic]** What are the time and space complexity of `foo1` and `foo2`? Explain, succinctly, what both functions do. Can you write a function that is better than `foo1` and `foo2`? What will be its time and space complexity?

```
def foo1(n):
    counter = 2
    while counter < n:
        if n % counter == 0:
            return True
        counter = counter + 1
    return False

def foo2(n):
    for i in range(2,(n//2+1)):
        if n % i == 0:
            return True
    return False
```

Another Higher-order Functions

6. **[Basic]** Consider the following function-generating functions:

```
from operator import *

def make_multiplier(scaling_factor):
    return lambda x : mul(x, scaling_factor)

def make_exponentiator(exponent):
    return lambda x : pow(x, exponent)
```

`mul` takes two variables `x` , `y` and returns `x * y`. `pow` takes two variables `x` , `y` and returns x^y. Notice that these two functions are quite similar. We could abstract out the commonality into an even more general function `make_generator` such that we could then just write:

```
make_multiplier = make_generator(mul)
make_exponentiater = make_generator(pow)
```

Write the function `make_generator` in IDLE. If you have seen this question before, try it out without referring to your notes.