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

Extra Practice on Orders of Growth - Part 1

This extra practice is meant to test your understanding of orders of growth (OOG) which is useful especially on your future modules such as CS2040 (Data Structures and Algorithms) and its variant or even CS1010S exams themselves. The questions will vary in difficulty in order to reinforce your fundamentals, so try to think about how does one come up with the answer to these questions.

Note: You may assume that doing an integer operation and printing an integer will take $O(1)$ time, and there won't be any potential recursion error or stack overflow. You may also assume that the initial integer inputs, if any, will be positive integers.

Deja Vu (6 marks)

For each of these expressions, determine the simplified big-O notation of it.

1. $n^2 \log n + n(\log n)^2$
2. $\log \sqrt{n} + (\log n)^2$
3. 3^{n^2+3n+6}
4. $\log \log(n^{2.5})$
5. $\ln n + \log_2 n$
6. $5^{7 \log_5 n} + n^5$

Analysis (19 marks)

For each **outermost** function, determine respectively the time and space complexity in terms of the simplified big-O notation. **Side Quest 2.4** might be useful in this practice. Have fun!

1.
```
def foo(n):
    for i in range(n):
        for j in range(n // 2):
            print(i, j)
```

2.
```
def bar(n):
    for i in range(n):
        for j in range(i):
            print(i, j)
```

3.
```
def baz(n):
    i = 1
    while i < n:
        i *= 2
        print("*" * i)
```

4.
```
def qux(n):
    i = 1
    while i < n:
        i *= 2
        print(i)
```

5.
```
def quux(n):
    if n >= 1:
        return
    for i in range(n):
        print(i)
    quux(n // 2)
    quux(n // 2)
```

6.
```
def quuz(n):
    if n <= 1:
        return
    for i in range(n):
        print(i)
    quuz(n // 2)
    quuz(n // 2)
```

7.
```
def corge(n):
    a, b = '', ''
    for i in range(n):
        a += 'a'
        b += 'b'
    return a + b
```

8.
```
def grault(n, m):
    if n <= 1:
        for i in range(m):
            print('@')
        return
    grault(n - 1, m)
```

9.
```
def garply(n, m):
    if n <= 1:
        for i in range(m):
            print('@')
        return
    garply(n - 1, m)
    garply(n - 1, m)
```

10.
```
def waldo(n, m):
    if n <= 1:
        for i in range(m):
            print('@')
        return
    waldo(n // 2, m)
    waldo(n // 2, m)
```

11.
```python
def fred(n):
    m = 1
    while m <= 1e9:
        m *= 2
        print(n)
```

12.
```python
def plugh(n):
    if n <= 0:
        return 1
    return plugh(n - 1) + plugh(n - 1)
```

13.
```python
def xyzzy(n):
    if n <= 0:
        return 1
    return 2 * xyzzy(n - 1)
```

14.
```python
def thud(n):
    if n != 0:
        for i in range(n // 2):
            for j in range(i):
                print(None)
        thud(n // 2)
```

15.
```python
def wobble(m, n):
    if m > n:
        return n
    return wobble(m + n, n) + n
```

16.
```python
def wubble(m, n):
    if m > n:
        return n
    return wubble(m + 1, n) + wubble(m + 3, n)
```

17.
```python
def explode(n):
    def boom(x):
        print("bow")
        bow(x - 1)

    def bow(x):
        if x < 0:
            return
        print("boom")
        boom(x - 1)

    boom(n)
```

18.
```
def spudow(n):
    def boom(x):
        for _ in range(x):
            print("bow")
        bow(x - 1)

    def bow(x):
        if x < 0:
            return
        boom(x - 1)
        while x > 0:
            print("boom")
            x //= 3

    boom(n)
```

19.
```
def fum(n):
    def fee(n):
        if n < 0:
            print('fum!')
            return
        print('fee')
        fi(n - 1)

    def fi(n):
        fo(n - 1)
        while n > 0:
            print('fi')
            n //= 2

    def fo(n):
        fee(n - 1)
        for _ in range(8):
            print('fo')

    for n in range(n):
        print('fum?')
        fee(n)
```