Worksheet 6: Orders of Growth & Linked Lists

1. Interpretation (Fa14 Mock Final Q5e)

```python
def g(n):
    if n % 2 == 0 and g(n + 1) == 0:
        return 0
    return 5
```

Circle the correct order of growth for a call to g(n):

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(b^n)$

**Solution: $\Theta(1)$**

2. The weakest link (Su15 Midterm 2 Q5d)

(2 pt) Consider the following linked list functions:

```python
def append(link, value):
    """Mutates link by adding value to the end of link.""
    if link.rest is Link.empty:
        link.rest = Link(value)
    else:
        append(link.rest, value)

def extend(link1, link2):
    """Mutates link1 so that all elements of link2 are added to the end of link1."
    while link2 is not Link.empty:
        append(link1, link2.first)
        link2 = link2.rest
```

Circle the order of growth that best describes the runtime of calling `append`, where $n$ is the number of elements in the input `link`.

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n^2)$
- $O(2^n)$

Assuming the two input linked lists to `extend` both contain $n$ elements, circle the order of growth that best describes the runtime of calling `extend`.

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n^2)$
- $O(2^n)$

**Solution 1: $O(n)$**

**Solution 2: $O(n^2)$**
3. Not with a fizzle, but with a bang (Su13 Midterm 2 Q2b)

(2 pt) Now consider the following function definitions.

```python
def boom(n):
    if n == 0:
        return "BOOM!"
    return boom(n - 1)

def explode(n):
    if n == 0:
        return boom(n)
    i = 0
    while i < n:
        boom(n)
        i += 1
    return boom(n)
```

Circle the correct order of growth for a call to `explode(n)`:

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$

**Solution:** $\Theta(n^2)$

4. Not with a fizzle, but with a bang (Su13 Midterm 2 Q2c)

(2 pt) Now consider the following function definition.

```python
def dreams(n):
    if n <= 0:
        return n
    if n > 0:
        return n + dreams(n // 2)
```

Circle the correct order of growth for a call to `dreams(n)`:

- $\Theta(1)$
- $\Theta(\log n)$
- $\Theta(\sqrt{n})$
- $\Theta(n)$
- $\Theta(n^2)$
- $\Theta(n^3)$
- $\Theta(2^n)$

**Solution:** $\Theta(\log n)$
5. Various Programs (Sp14 Final Q5c)

(2 points) Give worst-case asymptotic $\Theta()$ bounds for the running time of the following code snippets. (Note: although we haven’t explicitly talked about it, it is meaningful to write things with multiple arguments like $\Theta(a + b)$, which you can think of as “$\Theta(N)$ where $N = a + b$."

```python
def a(m, n):
    for i in range(m):
        for j in range(n // 100):
            print("hi")

def b(m, n):
    for i in range(m // 3):
        print("hi")
    for j in range(n * 5):
        print("bye")

def d(m, n):
    for i in range(m):
        j = 0
        while j < i:
            print("hi")
            j = j + 100

def f(m):
    i = 1
    while i < m:
        i = i * 2
    return i
```

Solution: $\Theta(mn)$
Solution: $\Theta(m+n)$
Solution: $\Theta(m^2)$
Solution: $\Theta(\log m)$
6. OOG Potpourri
What is the order of growth of each of the following functions?

a. Weighted
def weighted_random_choice(lst):
    temp = []
    for i in range(len(lst)):
        temp.extend([lst[i]] * (i + 1))
    return random.choice(temp)

Solution: Θ(n^2)

b. Iceskate
def ice(n):
    skate = n
    def rink(n):
        nonlocal skate
        print(n)
        if skate > 0:
            skate -= 1
            rink(skate)
        return skate
    return rink(n//2)

Solution: Θ(n)

c. Olympics
def olym(pics):
    total, counter = 0, 0
    for i in range(pics):
        while counter == 0:
            total += (i + counter)
            counter += 1
    return total

Solution: Θ(1)
d. Palindrome

def is_palindrome(s):
    if len(s) <= 1:
        return True
    return s[0] == s[-1] and is_palindrome(s[1:-1])

Solution: Θ(n^2)

e. More Palindrome

def is_palindrome2(s):
    for i in range(len(s) // 2):
        if s[i] != s[-i-1]:
            return False
    return True

Solution: Θ(n)

f. Havana

def camila(m, n):
    if n <= 1:
        return 0
    cabello = 0
    for i in range(3 ** m):
        cabello += i // n
    return cabello + camila(m - 5, n // 3)

Solution: Θ(3^m log n)

g. Barbados

def ri(na):
    if na < 1:
        return na
    def han(na):
        i = 1
        while i < na:
            i *= 2
        return i
    return ri(na / 2) + ri(na / 2) + han(na - 2)

Solution: O(nlogn)
7. Conserve Links (Challenge Linked List problem)
Implement conserve_links, as described below.

def conserve_links(a, b):
    """Makes Linked List a share as many Link instances as possible
    with Linked List b. a can use b's i-th Link instance as its i-th
    Link instance if a and b have the same element at position i.

    Should mutate a. b is allowed to be destroyed. Returns the new
    first Link instance of a.
    """
    if a.first == b.first:
        b.rest = conserve_links(a.rest, b.rest)
        return b
    else:
        return a
8. Slice Reverse (Challenge Linked List problem)

Implement `slice_reverse` which takes a linked list `s` and mutatively reverses the elements on the interval, `[i, j)` (including `i` but excluding `j`). Assume `s` is zero-indexed, `i > 0`, `i < j`, and that `s` has at least `j` elements.

```python
def slice_reverse(s, i, j):
    
    >>> s = Link(1, Link(2, Link(3)))
    >>> slice_reverse(s, 1, 2)
    >>> s
    Link(1, Link(2, Link(3)))
    >>> s = Link(1, Link(2, Link(3, Link(4, Link(5)))))
    >>> slice_reverse(s, 2, 4)
    >>> s
    Link(1, Link(2, Link(4, Link(3, Link(5)))))
    
    start = s

    for _ in range(i - 1):
        
        start = start.rest

    reverse = Link.empty

    current = start.rest

    for _ in range(j - i):
        
        rest = current.rest

        current.rest = reverse

        reverse = current

        current = rest

    start.rest.rest = current

    start.rest = reverse
```