Orders of Growth and Linked Lists

March 12 to March 15, 2018

1 Orders of Growth

1. The Weakest Link (Su15 Midterm 2 Q5d)

```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
```

Consider the following linked list functions: Circle the order of growth that best describes the runtime of calling `append`, where \( n \) is the number of elements in the input link.

\[
O(1) \quad O(\log n) \quad O(n) \quad O(n^2) \quad O(2^n)
\]

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`.

\[
O(1) \quad O(\log n) \quad O(n) \quad O(n^2) \quad O(2^n)
\]
2. 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 \( g(n) \):

\[ \Theta(1) \quad \Theta(\log n) \quad \Theta(n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(b^n) \]

3. Not with a fizzle, but with a bang (Su13 Midterm 2 Q2b) Consider the following linked list functions:

```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 \( \text{explode}(n) \):

\[ \Theta(1) \quad \Theta(\log n) \quad \Theta(n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(2^n) \]

4. Not with a fizzle, but with a bang (Su13 Midterm 2 Q2c) Consider the following linked list functions:

```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 \( \text{dreams}(n) \):

\[ \Theta(1) \quad \Theta(\log n) \quad \Theta(n) \quad \Theta(n^2) \quad \Theta(n^3) \quad \Theta(2^n) \]
5. **Various Programs (Sp14 Final Q5c)** Give worst-case asymptotic bounds, in terms of \( m \) and \( n \), for the running time of the following functions.

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

Bound:

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

Bound:

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

Bound:
```

6. **OOG Potpourri** What is the order of growth of each of the following functions?

a. **Weighted**

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

Order of Growth:

b. **Iceskate**

```python
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)
```

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

Order of Growth:

d. Palindrome
   ```python
def is_palindrome(s):
    if len(s) <= 1:
        return True
    return s[0] == s[-1] and is_palindrome(s[1:-1])
```

Order of Growth:

e. More Palindrome
   ```python
def is_palindrome2(s):
    for i in range(len(s) // 2):
        if s[i] != s[-i-1]:
            return False
    return True
```

Order of Growth:

f. Havana
   ```python
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)
```

Order of Growth:

g. Barbados
   ```python
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)
```

Order of Growth:
1. **Conserve Links (Challenge Linked List problem)** Implement `conserve_links`, as described below.

   ```python
   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.
       >>> x = Link(1, Link(2, Link(3, Link(4, Link(5, Link(6))))))
       >>> y = Link(1, Link(9, Link(3, Link(4, Link(9, Link(6))))))
       >>> z = conserve_links(x, y)
       >>> curr_x, curr_z = x, z
       >>> while curr_z is not Link.empty:
       >>>     assert curr_z.first == curr_x.first
       >>>     curr_x, curr_z = curr_x.rest, curr_z.rest
       >>> assert z == y
       >>> assert z.rest.rest == y.rest.rest
       >>> assert z.rest.rest.rest.rest.rest == y.rest.rest.rest.rest.rest
       >>> """
   ```
2. **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.

You must use mutation; solutions which call the `Link` constructor will not receive credit. The `Link` class reference is provided below.

```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 = ______________________________________________________
    for _____________________________________________________________:
        start = ___________________________________________________
    reverse = Link.empty
    current = _____________________________________________________
    for _____________________________________________________________:
        ___________________________________________________________
        current.rest = _____________________________________________
        reverse = ________________________________________________
        current = _________________________________________________
```