For Exercises 1-6 (below), identify the order of growth of the runtime as a function of n. As an example, your answer might be $O(\sqrt{n})$... or even $O(n^5 \log n)$. Note that this code will also be on the slides, along with the solutions to all of these problems!

1. (2 points) Exercise 1

```
def mystery1(n):
    n, result = str(n), ''
    num_digits = len(n)
    for i in range(num_digits):
        result += n[num_digits - i - 1]
    return result


---------------
def mystery2(n):
    n, result = 5, 0
    while n <= 3000:
        result += mystery1(n // 2)
        n += 1
    return result


---------------
```

2. (3 points) Exercise 2

```
def mystery3(n):
    if n < 0 or n <= sqrt(n):
        return n
    return n + mystery3(n // 3)


---------------
def mystery4(n):
    if sqrt(n) <= 50:
        return 1
    return n * mystery4(n // 2)


---------------
def mystery5(n):
    for _ in range(int(sqrt(n))):
        n = 1 + 1
    return n


---------------
```

3. (2 points) Exercise 3

```
def mystery6(n):
    while n > 1:
        x = n
        while x > 1:
            print(n, x)
            x = x // 2
        n -= 1


---------------
def mystery7(n):
    result = 0
    for i in range(n // 10):
        result += 1
        for j in range(10):
            result += 1
            for k in range(10 // n):
                result += 1
    return result


---------------
```

4. (2 points) Exercise 4

```
def mystery8(n):
    if n == 0:
        return ''
    result, stringified = '', str(n)
    for digit in stringified:
        for _ in range(n):
            result += digit
    result += mystery8(n - 1)
    return result


---------------
def mystery9(n):
    total = 0
    for i in range(1, n):
        total *= 2
        if i % n == 0:
            total *= mystery9(n - 1)
            total *= mystery9(n - 2)
        elif i == n // 2:
            for j in range(1, n):
                total *= j
    return total


---------------
```

5. (2 points) Exercise 5

```
def mystery10(n):
    if n > 0:
        r1 = mystery10(-n)
        r2 = mystery10(n - 1)
        return r1 + r2
    return 1


---------------
def mystery11(n):
    if n < 1:
        return n
    def mystery12(n):
        i = 1
        while i < n:
            i *= 2
        return i
    return mystery11(n / 2) + mystery11(n / 2) + mystery12(n - 2)


---------------
```

6. (2 points) Exercise 6

The orders of growth should now be functions of m **and** n.

```
def mystery13(m, n):
    if n <= 1:
        return 0
    result = 0
    for i in range(3 ** m):
        result += i // n
    return result + mystery13(m - 5, n // 3)


---------------
def mystery14(m, n):
    result = 0
    for i in range(1, m):
        j = i * i
        while j <= n:
            result += j
            j += 1
    return result


---------------
```

7. (1 points) Exercise 7

Define n to be the length of the input list. How much memory does the following program use as a function of n?

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

----

8. (7 points) Exercise 8

Provide an algorithm that, given a sorted list A of distinct integers, determines whether there is an index i for which $A[i] = i$. Your algorithm should run in time $O(\log n)$, where n is the length of the list.

```
def index_exists(A):
    def helper(lower, upper):

        if _________________________________:

            return _________________________

        mid_idx = (lower + upper) // 2

        if _________________________________:
            return True

        elif _______________________________:

            return _________________________
        else:

            return _________________________

    return _________________________________
```

9. (3 points) Summer 2013 MT2 | Q2

(a) (1 pt) What is the order of growth for a call to `fizzle`(n)?

```
def fizzle(n):
    if n <= 0:
        return n
    elif n % 23 == 0:
        return n
    return fizzle(n - 1)
```

(b) (1 pt) What is the order of growth for a call to `explode`(n)?

```
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)
```

____

(c) (1 pt) What is the order of growth for a call to `dreams`(n)?

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

____

10. (4 points) Summer 2014 MT2 | Q6

Consider the following function (assume that parameter S is a list):

```
def umatches(S):
    result = set()
    for item in S:
        if item in result:
            result.remove(item)
        else:
            result.add(item)
    return result
```

(a) (1 pt) Fill in the blank: The function `umatches` returns the set of all

____.

(b) (1 pt) Let's assume that the operations of adding to, removing from, or checking containment in a set each take roughly constant time. Give an asymptotic bound (the tightest you can) on the worst-case time for `umatches` as a function of $N =$ `len`(S).

____

(c) (1 pt) Suppose that instead of having `result` be a set, we make it a list (so that it is initialized to `[]` and we use `.append` to add an item). What now is the worst-case time bound? You can assume that `.append` is a constant-time operation, and `.remove` and the `in` operator require time that is $\Theta(L)$ in the worst case, where L is the length of the list operated on. Since we never add an item that is already in the list, each value appears at most once, just as for a Python set.

____

(d) **(1 pt)** Now suppose that we consider only cases where the number of different values in list S is at most 100, and we again use a list for `result`. What is the worst-case time now?

----

11. (2 points) Summer 2015 MT2 | Q5(d)

```
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 LINK_1 so that all elements of LINK_2
    are added to the end of LINK_1.
    """
    while link2 is not Link.empty:
        append(link1, link2.first)
        link2 = link2.rest
```

(a) **(1 pt)** What order of growth describes the runtime of calling `append`? Give your function in terms of n, where n is the number of elements in the input `link`.

(b) **(1 pt)** Assuming the two input linked lists both contain n elements, what order of growth best describes the runtime of calling `extend`?

----

12. (2 points) Summer 2012 Final | Q2

(a) **(1 pt)** What is the order of growth in n of the runtime of `collide`, where n is its input?

```
def collide(n):
    lst = []
    for i in range(n):
        lst.append(i)
    if n <= 1:
        return 1
    if n <= 50:
        return collide(n - 1) + collide(n - 2)
    elif n > 50:
        return collide(50) + collide(49)
```

(b) **(1 pt)** What is the order of growth in n of the runtime of `into_me`, where n is its input?

```
def crash(n):
    if n < 1:
        return n
    return crash(n - 1) * n
```

```
def into_me(n):
    lst = []
    for i in range(n):
        lst.append(i)
    sum = 0
    for elem in lst:
        sum = sum + crash(n) + crash(n)
    return sum
```

----

13. (4 points) Spring 2014 Final | Q5(c)

Give worst-case asymptotic Θ bounds – you guys can write them as Big-O bounds – for the running time of the following code snippets. As a reminder, it is meaningful to write things with multiple arguments like $\Theta(a+b)$, which you can think of as "$\Theta(N)$ where $N = a + b$."

(a) (1 pt)

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

(b) (1 pt)

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

(c) (1 pt)

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

(d) (1 pt)

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