Python “Review”: the `xrange` function can be used with a `for` loop to generate a sequence of values one at a time for each iteration of the loop. For example:

```python
n = input("Enter # of iterations? ")
for count in xrange(n):
    print count, "  ",
print "\nDone"
```

(NOTE: a comma at the end of a print statement causes the next print to occur on the same line)

Terminology:

**problem** - question we seek an answer for, e.g., "what is the largest item in a list/array?"

**parameters** - variables with unspecified values

**problem instance** - assignment of values to parameters, i.e., the specific input to the problem

```
myList:  0  1  2  3  4  5  6
      5 10  2 15 20  1 11
      n:  7
  (number of elements)
```

**algorithm** - step-by-step procedure for producing a solution

**basic operation** - fundamental operation in the algorithm (i.e., operation done the most) Generally, we want to derive a function for the number of times that the basic operation is performed related to the **problem size**.

**problem size** - input size. For algorithms involving lists/arrays, the problem size is the number of elements (“n”).

**Big-oh notation** ($O()$) - As the size of a problem grows (i.e., more data), how will our program’s run-time grow.

1. Consider the following `sumList` function.

```python
import time
def main():
    n = input("Enter size of list: ")
    aList = range(1, n+1)
    start = time.clock()  # resolution better than one microsecond, better than time.time
    sum = sumList(aList)
    end = time.clock()
    print "Time to sum the list was %.5f seconds" % (end-start)

def sumList(myList):
    """Returns the sum of all items in myList""
    total = 0
    for item in myList:
        total = total + item
    return total

main()
```

a) What is the basic operation of `sumList` (i.e., operation done the most) ?

b) What is the problem size of `sumList`?

c) If we input n of 10000 and `sumList` takes 0.00144 seconds, how long would you expect `sumList` to take for n of 20000?

d) What is the big-oh notation for `sumList`?
2. Consider the following `sumSomeListItems` function.

```python
import time

def main():
    n = input("Enter size of list: ")
    aList = range(1, n+1)
    start = time.clock()
    sum = sumSomeListItems(aList)
    end = time.clock()
    print "Time to sum the list was %.9f seconds" % (end-start)

def sumSomeListItems(myList):
    """Returns the sum of some items in myList"""
    total = 0
    index = len(myList) - 1
    while index > 0:
        total = total + myList[index]
        index = index / 2
    return total

main()
```

a) What is the problem size of `sumSomeListItems`?

b) If we input `n` of 1,000,000 and `sumSomeListItems` takes 0.000021998 seconds, how long would you expect `sumSomeListItems` to take for `n` of 2,000,000?

c) What is the big-oh notation for `sumSomeListItems`?
3. Consider the following `someLoops` function.

```python
import time

def main():
    n = input("Enter n: ")
    start = time.clock()
    sum = someLoops(n)
    end = time.clock()
    print "Time of someLoops was %.9f seconds" % (end-start)

def someLoops(n):
    """Returns the sum of values""
    total = 0
    for i in xrange(n):
        for j in xrange(n):
            total = total + i + j
    return total

main()
```

a) What is the basic operation of `someLoops` (i.e., operation done the most)?

b) How many times will the basic operation execute as a function of `n`?

c) What is the big-oh notation for `someLoops`?

d) If we input `n` of 10000 and `someLoops` takes 18.7 seconds, how long would you expect `someLoops` to take for `n` of 20000?

4. Analyze the below algorithm to determine its big-oh notation, $O(\cdot)$.

```python
i = 1
while i <= n:
    for j in xrange(n):
        # something of $O(1)$
        # end for
    i = i * 2
# end while
```

Execution flow

- $i = 0$ loops $n$ times
- $i = 1$ loops $n$ times
- $i = 2$ loops $n$ times
- $i = n-1$ loops $n$ times

- $j = 0$ to $n-1$ loops $n$ times
- $j = 0$ to $n-1$ loops $n$ times
- $j = 0$ to $n-1$ loops $n$ times
- $j = 0$ to $n-1$ loops $n$ times

```
```