Data Structures (810:052) Lecture 1 Name:_____

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

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

```
Enter # of iterations? 6
0  1  2  3  4  5
Done
```

(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

	0	1	2	3	4	5	6
myList:	5	10	2	15	20	1	11

largest: ?

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.

```
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 10 seconds, how long would you expect `sumList` to take for n of 20000?

d) What is the big-oh notation for `sumList`?

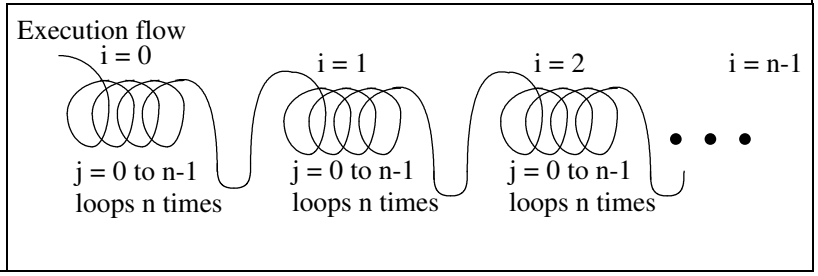
2. Consider the following `someLoops` function.

```
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()
```

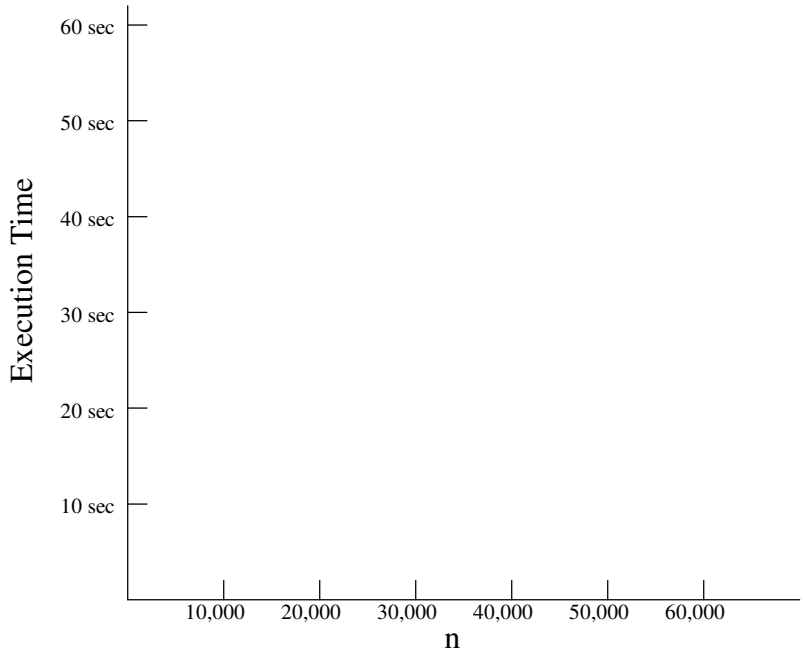


a) What is the basic operation of `someLoops` (i.e., operation done the most)?

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 10 seconds, how long would you expect `someLoops` to take for n of 20000?



e) Draw the graph for `someLoops` and `sumList`

3. Consider the following `sumSomeListItems` function.

```
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 10,000 and `sumSomeListItems` takes 10 seconds, how long would you expect `sumSomeListItems` to take for n of 20,000?

(Hint: For n of 20,000, how many more times would the loop execute than for n of 10,000?)

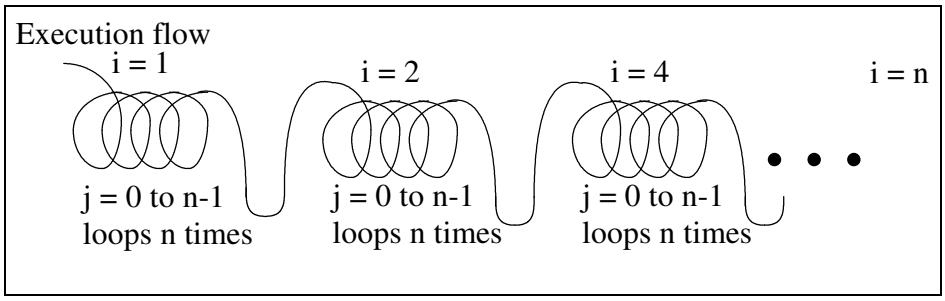
c) What is the big-oh notation for `sumSomeListItems`?

d) Add the execution-time graph for `sumSomeListItems` to the graph.

4.

```
i = 1
while i <= n:
     for j in xrange(n):
         # something of O(1)

     # end for
     i = i * 2
# end while
```



a) Analyze the above algorithm to determine its big-oh notation, *O*( ).

b) If n of 10,000, takes 10 seconds, how long would you expect the above code to take for n of 20,000?

c) Add the execution-time graph for the above code to the graph.