Data Structures (CS 1520) Lecture 2 Name:____________________

1. An alternative to functional-decomposition design is to use object-oriented design(OOD). For the following program, what objects would be useful and what methods (operations on the objects) should each support?

"Write a program to roll two 6-sided dice 1,000 times to determine the percentage of each outcome (i.e., sum boths dice). Report the outcome(s) with the highest percentage." (You only need consider the program's OOD)

Objects: Die: data attributes: currentRoll
methods/operations: roll, getRoll

Tally Sheet: data attributes
tally Dictionary
incrementATally
addLabel

2. Consider the Die and AdvancedDie classes from the Python Summary handout.

a) What data attributes of AdvancedDie are inherited from the parent Die class?

-currentRoll

b) What new data attributes are added as part of the subclass AdvancedDie?

-numSide

c) Which Die class methods are used directly for an AdvancedDie object?

getRoll

d) Which Die class methods are redefined/overridden by the AdvancedDie object?

__init__, __str__, roll

e) Which methods are new to the AdvancedDie class and not in the Die class?

__eq__, __lt__, __gt__, __add__, getSides

f) If `die1` and `die2` are AdvancedDie objects, then the statement "`if die1 == die2:`" invokes the `__eq__` method of AdvancedDie with `die1` "passed" as `self` and `die2` passed as `rhs_Die`.

```
def __eq__(self, rhs_Die):
    """Overrides default '__eq__' operator to allow for deep comparison of dice"""
    return self._currentRoll == rhs_Die._currentRoll
```

What would the code be for AdvancedDie `__le__` method to allow for the "`if die1 <= die2:`" statement?

def __le__(self, rhs_Die):
return self._currentRoll <= rhs_Die._currentRoll

g) Good software engineering practice is to include *precondition* and *postcondition* comments on each method/function where the:

- *precondition* - indicates what must be true for the method to work correctly. Typically, the precondition describes the valid values of the parameters. If the precondition is not satisfied, the method does not need to work correctly!
- *postcondition* - describes the expected state after the method has executed

Consider the AdvancedDie constructor:

```
class AdvancedDie(Die):
    """Advanced die class that allows for any number of sides"""

    def __init__(self, sides = 6):
        """Constructor for any sided Die that takes an the number of sides
        as a parameter; if no parameter given then default is 6-sided."""

        Die.__init__(self)  # call Die parent class constructor
        self._numSides = sides
        self._currentRoll = randint(1, self._numSides)
```

What precondition and postcondition comments should we add?

precondition: sides is positive integer

postcond: current roll is random value between 1 and # sides

h) If a method/function has a precondition that is not met when invoked (e.g., `die1 = AdvancedDie("six")`), why should the method raise an error?

To raise an error immediately when it occurs

```
class AdvancedDie(Die):
    def __init__(self, sides=6):
        """ precondition: sides is a positive integer
            postcond: current roll is a random value
            between 1 and # sides. """

        if not isinstance(sides, int):
            raise(TypeError, "AdvancedDie sides
                  must be an integer.")
        if sides <= 0:
            raise (ValueError, "AdvancedDie sides
                  must be a positive integer.")
        self._numSides = sides
        self._currentRoll = randint(1, sides)
```

Data Structures (CS 1520) Lecture 2 Name:________________

3. **General "Algorithmic-Complexity Analysis" terminology:**

problem - question we seek an answer for, e.g., "What is the sum of all the items 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

sum: ?

(number of elements) n: 7

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.

Consider the following sumList function.

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

addition

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

b) What is the problem size of sumList? length of myList = "n"

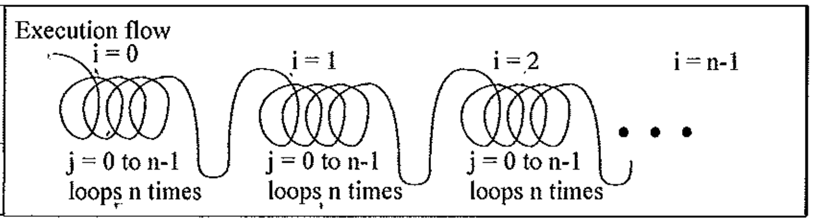
c) If n is 10000 and sumList takes 10 seconds, how long would you expect sumList to take for n of 20000? 20 sec

d) What is the big-oh notation for sumList? $O(n)$ linear time

4. Consider the following someLoops function.

```
def someLoops(n):
    total = 0
    for i in range(n):
        for j in range(n):
            total = total + i + j
    return total
```

once
n times
n^2



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?

$$\underbrace{n + n + n + \dots + n}_{n \text{ times}} = n^2$$

c) What is the big-oh notation for someLoops? $O(n^2)$

d) If we input n of 10000 (= n) and someLoops takes 10 seconds, how long would you expect someLoops to take for n of 20000? (= 2n)

$O(n^2)$ $T(n) \doteq c\,n^2 + \cancel{dn + e}$ — time to do inner most loop once

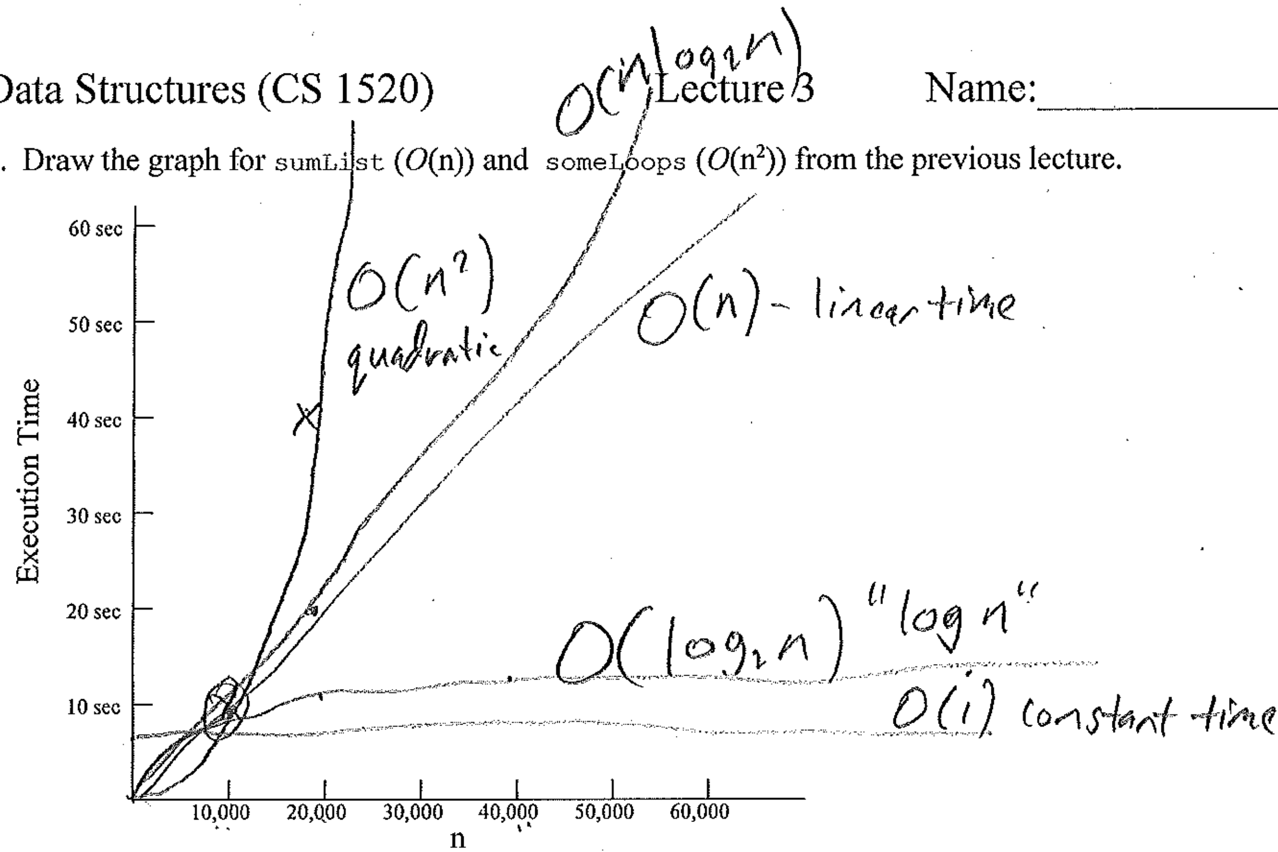
$T(10000) = c\, 10000^2 = 10\text{ sec}$

$c = \left(\frac{10\text{ sec}}{10000^2}\right)$

$T(20000) = c\, 20000^2 = \left(\frac{10\text{sec}}{10000^2}\right) 20000^2$ ($2^2 \cdot 10000^2$)

$= 40\text{ sec}$

1. Draw the graph for `sumList` (O(n)) and `someLoops` (O(n^2)) from the previous lecture.



2. Consider the following `sumSomeListItems` function.

0 1 2 4 8 16 32 64

```
import time

def main():
    n = eval(input("Enter size of list: "))
    aList = list(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`? len(myList) = "n"

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?)

index: 64, 32, 16, 8, 4, 2, 1

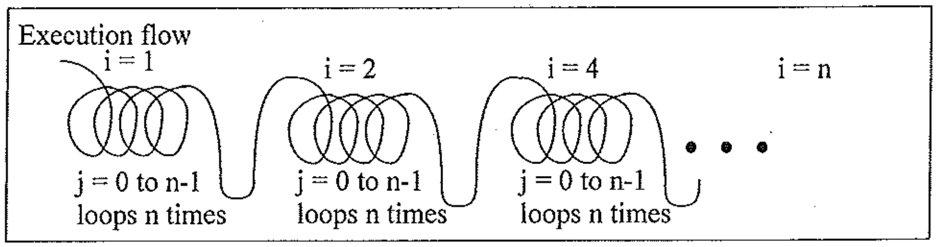
2^6 2^5 2^4 2^3 2^2 2^1 2^0 $O(\log_2(n)+1)$ ($\log_2 n = x$ iff $2^x = n$)

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

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

3.
```
i = 1
while i <= n:
     for j in range(n):
          # something of O(1)
     # end for
     i = i * 2
# end while
```
$\log_2 n$

$\leftarrow n$ times



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

$O(n \log_2 n)$

$\log_2 x = \dfrac{\log_{10} x}{\log_{10} 2}$

32K
16K 2^{14} $\leftarrow 10000$
8K 2^{13} 1K 2^{10}
4096 2048 1024 512 256 128 64 32 16 8 4 2 1
2^2 2^1 2^0

b) If n of 10,000, takes 10 seconds, how long would you expect the above code to take for n of 20,000? 14.x

$T(n) = c\, n \log_2 n$

$T(10000) = c\, 10000 \log_2 10000 = 10 sec$

$c = \dfrac{10 sec}{10000 \log_2 10000}$ 13.x

$T(20000) = c\, 20000 \log_2 20000$

$= \dfrac{10 sec}{10000 \log_2 10000} 20000 \log_2 20000$

$= \dfrac{10 sec\ 20000 (14.x)}{10000 (13.x)} =$

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

4. Most programming languages have a built-in array data structure to store a collection of same-type items. Arrays are implemented in RAM memory as a contiguous block of memory locations. Consider an array X that contains the odd integers:

address	Memory	
4000	1	X[0]
4004	3	X[1]
4008	5	X[2]
4012	7	X[3]
4016	9	X[4]
4020	11	X[5]
4024	13	X[6]
⋮		

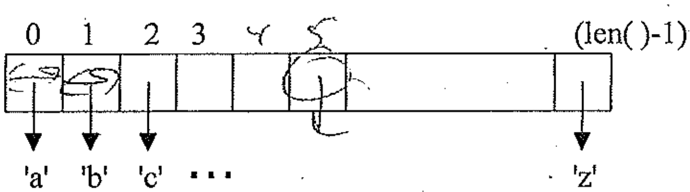
a) Any array element can be accessed randomly by calculating its address. For example, address of X[5] = 4000 + 5 * 4 = 4020. What is the general formula for calculating the address of the ith element in an array?

addr X[i] = (starting addr. of array) + i × (element size in bytes)

b) What is the big-oh notation for accessing the ith element?

O(1) constant time

c) A Python list uses an array of references (pointers) to list items in their implementation of a list. For example, a list of strings containing the alphabet:



Since a Python list can contain heterogeneous data, how does storing references in the list aid implementation?

O(1) constant time access to any index

5. Arrays in most HLLs are static in size (i.e., cannot grow at run-time), so arrays are constructed to hold the "maximum" number of items. For example, an array with 1,000 slots might only contain 3 items:

size: ~~3~~ 4

	0	1	2	3				999
scores:	20	10	30	25				

a) The *physical size* of the array is the number of slots in the array. What is the physical size of scores? 1000

b) The *logical size* of the array is the number of items actually in the array. What is the logical size of scores? 3

c) The *load factor* is faction of the array being used. What is the load factor of scores? 3/1000

d) What is the $O(\,)$ for "appending" a new score to the "right end" of the array? $O(1)$

e) What is the $O(\,)$ for adding a new score to the "left end" of the array? $O(n)$ $n \equiv$ logical size

f) What is the *average* $O(\,)$ for adding a new score to the array? $O(\frac{n}{2}) = O(n)$ $T(n) = c\frac{n}{2} = d\,n$

g) During run-time if an array fills up and we want to add another item, the program can usually:
- Create a bigger array than the one that filled up
- Copy all the items from the old array to the bigger array
- Add the new item
- Delete the smaller array to free up its memory

When creating the bigger array, how much bigger than the old array should it be?
double size over the old array

h) What is the $O(\,)$ of moving to a larger array? $O(n)$

6. Consider the following list methods in Python:

Method	Usage	Average $O(\,)$ for myList containing n items
index []	`itemValue = myList[i]`	
	`myList[i] = newValue`	
append	`myList.append(item)`	
extend	`myList.extend(otherList)`	
insert	`myList.insert(i, item)`	
pop	`myList.pop( )`	
pop(i)	`myList.pop(i)`	
del	`del myList[i]`	
remove	`myList.remove(item)`	
index	`myList.index(item)`	
iteration	`for item in myList:`	
reverse	`myList.reverse( )`	

Dictionary Operations:

Method	Usage	Explanation	Average $O(\,)$ for n keys
get item	`myDictionary.get(myKey)` `value = myDictionary[myKey]`	Returns the value associated with myKey; otherwise *None*	$O(1)$
set item	`myDictionary[myKey]=value`	Change or add myKey:value pair	$O(1)$
in	`myKey in myDictionary`	Returns True if myKey is in myDictionary; otherwise False	$O(1)$
del	`del myDictionary[myKey]`	Deletes the mykey:value pair	$O(1)$