Practice Midterm 2

1. What is the complexity of the following pieces of code expressed in big-O notation?

```
def loop(n, lis):
     n = n ** 2
     for i in range (n):
          for j in range(len(lis)):
               print (lis)
```

$O(n^3)$

```
def foo():
     n = input()
     m = (ord('a') % 256) ** 2
     alist = LinkedList()
     for char in n:
          alist.add_to_head(n)
          for i in range(m):
               alist.add_to_tail("hello")
     return alist
```

$O(n^2)$

```
def print_list_multiple_times(list_of_string):
     For i in range(10):
          For element in list_of_strings:
               print(element)
```

$O(n)$

```
def bar(n):
     m = random.randint(0,512)
     if m * 2 == 1024:
          for i in range(n):
               for j in range(n):
                    for k in range(n):
                         for l in range(n):
                              if i + j == k + l:
                                   print(i, j, k, l)
     else:
          for i in range(n):
               print(i, i, i, i)
```

O(n) O(n) O(n) O(n) O(n⁴) + O(n) = O(n⁴)

2. What is the difference between white-box and black-box testing?

Answer: Black-box testing is a software testing method where the structure or design of the code is not known to the tester, so the tester can only see the input/output of the program. White-box testing is a method which the design of the code IS known to the tester, so different sections of the code can be experimented upon.

3. Provide the formal definition of Big-O notation.

Answer:
Let f(n) and g(n) be functions mapping positive integers to positive real numbers.
Then, f(n) is O(g(n)) if there is a real constant c and an integer constant n0 ≥ 1 such that f(n) ≤ cg(n) for all n > n0

4. Write two blackbox tests that would detect error(s) in the following code and state what Error each one would detect

Files that exist in the current directory are:
average.py
ThisIsAFileWithInput.txt
ClickThisToGetAnA.exe

```
"""
This function reads in a file, one integer per line, and then
finds the average and returns it
"""
def average():
    file=open(input())
    file=file.readlines()
    sum=0
    for line in file:
        sum+=int(line)

    return sum/len(file)
```

give a nonexistant file
nofile.txt

give file with unexpected lines
ClickThisToGetAnA.exe

Answer: File is empty (Produces ZeroDivisionError)
Input a file that is not in the current directory (FileNotFoundError)

5. Note what lines of the following code could produce an error and write down what the error is for each line noted (Ex: ZeroDivisionError, ValueError, etc)

```
def print_elements(some_list):
    fname=input()
    file=open(fname).readlines()

    for i in range(len(file)):
        pos=int(file[i])
        element=some_list[pos]

        for temp in element:
            print(temp)
```

FileNotFoundError
ValueError
IndexError
TypeError

6. Given the following Python code, finish the exception handlers so that each one catches a single exception, prints out an appropriate error message, and ends the program.

```
def errorFunction(string, num, dictionary):
    try:
        val = int(string)
        quotient = dictionary[string] / num
        key = string + str(quotient)
        dictionary[key] = num
    except _________:


    except _________:


    except _________:


    except _________:
```

ValueError
print("Invalid type for int")

DivisionByZeroError
print("Can't divide by zero")

TypeError
print("Can't divide string by num")

KeyError
print("Key not present in dictionary")

7. Write a recursive function *get_last(node)* that returns the last node in a linked list, that takes the first node as a parameter.

Ex:

```
L = LinkedList()
L.add(Node(1))
L.add(Node(2))
node = L.get_head()
get_last(Node)
#Should return node with 1 for value
```

Code that you have access to

```
class Node:
    def __init__(self,value):
        self._value = value
        self._next = None

    def get_next(self):
        return self._next

class LinkedList:

    def __init__(self):
        self._head = None

    def get_head(self):
        return self._head

    def add(self,new):
        new._next = self._head
        self._head = new
```

8. Write a recursive function `is_prime(n)` that returns a boolean as to whether the number is prime or not. `n` will be greater than or equal to 2. Hint. Introduce a helper function that takes another variable.

9. Write a recursive function `recursive_primes(n)` that will print out all the primes up-to and including `n` in ascending order. `n` will be greater than or equal to 2.

```
>>> recursive_primes(11)
2
3
5
7
11
```

10. Write a recursive function `is_in_string(s, c)` that takes string s and char c as parameters and returns True or False if c is in the string s.

```
Ex.
s= 'hello' c='e'. Is_in_string returns True
s= 'hello' c='a'. Is_in_string returns False
```

11. Write a method, `r_sum(curr)`, that takes the head of a linked list as a parameter and recursively returns the sum of a linked list. Assume that the LinkedList class and Node class are provided. Do NOT create a new linked list. You must traverse your original one. If the linked list is empty, return 0.

Example:

```
>>> LL = LinkedList()
>>> LL.add(Node(7))
>>> LL.add(Node(6))
>>> LL.add(Node(5))
>>> print(r_sum(LL._head))
18
```

12. Given our past project fake-news where it takes in a csv that contains data representing a particular new post, identify if the following tests are regular, error, or edge test cases.

Input File:	N:	Type of test case
nonexistantfile.csv	2	Error
in100.csv	3	Regular
emptyFile.csv	2	edge
in100.csv	0	edge
in100.csv	-1	Error

13. Using a Stack class with the `push(item)`, `pop()`, `peek()`, `length()`, and `is_empty()` methods, write a Python function `reverse_pairs` that returns a new string, where every pair of letters in the string has been swapped. If the string has an odd length, the position of the last character will not change.
Examples:

- `reverse_pairs("test") --> "etts"`
- `reverse_pairs("") --> ""`
- `reverse_pairs("abc") --> "bac"`

14. Write a method for the class LinkedList `keep_only_duplicates(alist)` that takes a linked list `alist` as a parameter and modifies `alist` in-place such that only nodes with values found more than once in the list remain. Duplicated values do not need to be adjacent. You may not construct another linked list, and your function must operate in O(n) time. You may assume that `len(alist) > 0`.

Example:

```
>> print(alist)
1 -> 2 -> 3 -> 3 -> 1 -> hello -> world -> hello -> None
>> keep_only_duplicates(alist)
>> print(alist)
1 -> 3 -> 3 -> 1 -> hello -> hello -> None
```

Given this simple Node class for problems 15 and 16.

```
class Node:
    def __init__(self,value):
        self._value = value
        self._next = None

    def get_value(self):
        return self._value

    def get_next(self):
        return self._next

    def set_value(self, value):
        self._value = value

    def set_next(self, next):
        self._next = next
```

15. Implement a Stack class using a Linked List Implementation.
Your class should have these methods:

push(item) - pushes the item to the top of the stack (item is not a Node)

pop() - returns and removes the item from the top of the stack. If the stack is empty, then do nothing.

(Don't forget about the init function!)

16. Implement a Queue class using a Linked List Implementation.. Suggestion: Maintain a Tail pointer.

enqueue(item) - adds the item to the back of the queue (item is not a Node)

dequeue() - adds the item to the front of the queue. If the queue is empty, then do nothing.

(Don’t forget about the init function!)

17. Write a Linked List class that contains the method `random_list()`, which will create a new linked list object. Remove the head from the linked list attribute (in the Linked List class) and insert it randomly either at the beginning or the end of the new list. Use a random number (1 or 0) where 0 is the front of the list and 1 is at the end. The new list you created should now be the list attribute that is saved in the linked list class.

```
import random
class Node:
    def __init__(self,value):
        self._value = value
        self._next = None

class LinkedList:
    def __init__(self):
        self._head = None

    def add_to_head(self,new):
        new._next = self._head
        self._head = new

    def remove_first(self):
        if self._head == None:
            return None
        else:
            n = self._head
            self._head = n._next
            n._next = None
            return n

    def add_to_end(self, new):
        if self._head == None:
            self._head = new
        else:
            current = self._head
            prev = current
            while current != None:
                prev = current
                current = current._next
            prev._next = new
```

```
def random_list(self):
#SOLUTION GOES HERE
```