1. (3 points) What, if anything, is wrong with the following method that attempts to determine the number of items in \( L \) that are multiples of \( n \):

```python
def multiple(L, n):
    if len(L) == 0:
        return 0
    if L[0] % n == 0:
        return 1 + multiple(L[1:], n)
    return multiple(L[1:], n)
```

Answer: Nothing is wrong

2. (3 points) What, if anything, is wrong with the following method that attempts to determine if list \( L \) is sorted:

```python
def isSorted(L):
    if len(L) < 2:
        return True
    if L[0] > L[1]:
        return False
    return isSorted(L[1:])
```

Answer: Nothing is wrong

3. (3 points) What, if anything, is wrong with the following method that attempts to determine if string \( S \) is a palindrome:

```python
def palindrome(S):
    if len(S) < 2:
        return True
    if S[0] != S[-1]:
        return False
    palindrome(S[1:-1])
```

Answer: It breaks the design rule: even if the recursive calls work properly, the original call does not work.

4. (4 points) Consider the program below.

```python
1 def binary_numbers(n, string_so_far):
2     if n==0:
3         print(string_so_far)
4     else:
5         binary_numbers(n-1, string_so_far+'0')
6         binary_numbers(n-1, string_so_far+'1')
7     binary_numbers(3, '')
```

How many activation records will be in the stack (including the one corresponding to the main program) when the first print statement is produced?

Answer: 5

5. (4 points) Consider the program below.
1 def binary_numbers(n, string_so_far):
2     if n==0:
3         print(string_so_far)
4     else:
5         binary_numbers(n-1, string_so_far+'0')
6         binary_numbers(n-1, string_so_far+'1')
7     binary_numbers(3, '')

At some point in execution, the top of the stack contains the following activation record: [binary_numbers, n=1, string_so_far = '10', ip=1] What is the next output that will be produced?

Answer: 100

6. (6 points) Consider the program below.

1 def binary_numbers(n, string_so_far):
2     if n==0:
3         print(string_so_far)
4     else:
5         binary_numbers(n-1, string_so_far+'0')
6         binary_numbers(n-1, string_so_far+'1')
7     binary_numbers(3, '')

Which of the following stack configurations will occur at some point during the program’s execution? Mark all that apply.

Answer:

a) [binary_numbers, n=0, string_so_far = '000', ip=1]
   [binary_numbers, n=1, string_so_far = '00', ip=6]
   [binary_numbers, n=2, string_so_far = '0', ip=6]
   [binary_numbers, n=3, string_so_far = '', ip=6]
   [main, ip =9]

c) [binary_numbers, n=0, string_so_far = '001', ip=1]
   [binary_numbers, n=1, string_so_far = '00', ip=7]
   [binary_numbers, n=2, string_so_far = '0', ip=6]
   [binary_numbers, n=3, string_so_far = '', ip=6]
   [main, ip =9]

d) [binary_numbers, n=1, string_so_far = '10', ip=1]
   [binary_numbers, n=2, string_so_far = '1', ip=6]
   [binary_numbers, n=3, string_so_far = '', ip=7]
   [main, ip =9]

7. (4 points) The running time of a function that returns smallest element in an unsorted list is (mark all that apply)

Answer: \( \Omega(1), \Omega(\log n), \Omega(n), O(n), \Theta(n) \).

8. (4 points) The running time of binary search is (mark all that apply)

\( O(n), \Omega(1), \Theta(\log n), O(\log n), \Omega(\log n) \)

9. (2 points) Determine the big-O running time of the following function:

```python
def p1(A):
    for a in A:
        print(a)
    for b in A:
        for c in A:
            print(b, c)
```
Answer: $O(n^2)$

10. (2 points) Determine the big-O running time of the following function:

```python
def p2(A):
    for a in A:
        print(a)
    i = len(A)-1
    while i>0:
        print(A[i])
        i = i//2
```

Answer: $O(n)$

11. (2 points) Determine the big-O running time of the following function:

```python
def p3(A):
    for a in A:
        print(a)
    i = len(A)-1
    while i>0:
        print(A[i])
        i = i//2
```

Answer: $O(n \log n)$

12. (2 points) Determine the big-O running time of the following function:

```python
def p8(a):
    i = len(a)-1
    while i>=0:
        print(a[i])
        i -= 2302
```

Answer: $O(n)$

13. (2 points) Determine the big-O running time of the following function:

```python
def p10(a):
    i = 1
    while i<len(a):
        for t in a:
            print(t)
        i = i*2
```

Answer: $O(n \log n)$

14. (2 points) Determine the big-O running time of the following function:

```python
def p10(a):
    i = 1024
    while i>0:
        print(a[0])
        i = i//2
```

Answer: $O(1)$

15. (2 points) What is the recurrence equation that describes the running time of the following function?

```python
def r0(a):
    if len(a)>0:
        print(a[0])
        r0(a[1:])
```
Answer: $T(n) = T(n-1) + 1$

16. (2 points) What is the recurrence equation that describes the running time of the following function?

```python
def r2(a):
    if len(a)<2:
        print(a)
    else:
        mid = len(a)//2
        r2(a[mid:])
        for i in range(len(a)):
            print(a[i], end=' ')
        r2(a[:mid])
```

Answer: $T(n) = 2T(n/2) + n$

17. (2 points) What is the recurrence equation that describes the running time of the following function?

```python
def r3(a):
    if len(a)<2:
        print(a)
    else:
        mid = len(a)//2
        for i in range(len(a)):
            print(a[i], end=' ')
        for i in range(4):
            r3(a[mid:])
```

Answer: $T(n) = 4T(n/2) + n$

18. (2 points) What is the recurrence equation that describes the running time of the following function?

```python
def r4(a):
    if len(a)<2:
        print(a)
    else:
        mid = len(a)//2
        for i in range(len(a)):
            for j in range(len(a)):
                print(a[i], end=' ')
        for i in range(2):
            r4(a[mid:])
```

Answer: $T(n) = 2T(n/2) + n^2$

19. (2 points) What is the recurrence equation that describes the running time of the following function?

```python
def r5(a):
    if len(a)>0:
        print(a[0])
        r5(a[1:])
        r5(a[::-1])
```

Answer: $T(n) = 2T(n-1) + 1$

20. (2 points) What is the solution to the recurrence $T(n) = 2T(n-1) + 1$?

Answer: $T(n) = O(2^n)$

21. (2 points) What is the solution to the recurrence $T(n) = T(n-1) + n$?

Answer: $T(n) = O(n^2)$
22. (2 points) What is the solution to the recurrence $T(n) = 2T(n/2) + n$?
   Answer: $T(n) = O(n \log n)$

23. (2 points) What is the solution to the recurrence $T(n) = 2T(n/2) + n^2$?
   Answer: $T(n) = O(n^2)$

24. (2 points) What is the solution to the recurrence $T(n) = 8T(n/2) + n$?
   Answer: $T(n) = O(n^3)$

25. (10 points) Write the recursive function $SumUntil(L,i)$ that receives a (native) list $L$ and an integer $i$ and returns the sum of the items that appear before $i$ in $L$. If $i$ does not appear in $L$, the function should return the sum of the whole list.

26. (10 points) Write the recursive function $LastOccurrence(L,i)$ that receives a (native) list $L$ and an integer $i$ and returns the position of the LAST occurrence of $i$ in $L$, or -1 if $i$ is not in $L$.

27. (10 points) Write the function $SumLast2(self)$ to be included in the $List$ class that returns the sum of the last two elements in an object of class $List$. If the list has less that two elements, the function should return the sum of the whole list.

28. (10 points) Write the function $SumFirstAndLast(self)$ to be included in the $List$ class that returns the sum of the first and last elements in an object of class $List$. If the list is empty, the function should return 0.

29. (12 points) Write the function $KeepFirstAndLast(self)$ to be included in the $List$ class that removes all the items, except for the first and last, from an object of class $List$. If the list has less that two elements, the function should do nothing (hint: this can be done in $O(1)$ time; there's no need to traverse the list).

Answers 25-29:

class Node(object):
    # Constructor
    def __init__(self, data, next=None):
        self.data = data
        self.next = next

#List Functions
class List(object):
    # Constructor
    def __init__(self, head = None, tail = None):
        self.head = head
        self.tail = tail

    def Print(self):
        t = self.head
        while t is not None:
            print(t.data, end=' ')
            t = t.next
        print()

    def Append(self, x):
        if self.head is None:
            self.head = Node(x)
            self.tail = self.head
        else:
            self.tail.next = Node(x)
            self.tail = self.tail.next

    def AppendList(self, python_list):
        for d in python_list:
            self.Append(d)
```python
def SumLast2(self):
    next_to_last, last = 0, 0
    t = self.head
    while t is not None:
        next_to_last, last = last, t.data
        t = t.next
    return next_to_last+last

def SumFirstAndLast(self):
    if self.head == None:
        return 0
    return self.head.data + self.tail.data

def KeepFirstAndLast(self):
    if self.head != None and self.head.next != None:
        self.head.next = self.tail

def SumUntil(L, i):
    if L == [] or L[0] == i:
        return 0
    return L[0] + SumUntil(L[1:], i)

def LastOccurrence(L, i):
    if L == []:
        return -1
    if L[-1] == i:
        return len(L)-1
    return LastOccurrence(L[:-1], i)

if __name__ == '__main__':
    print('SumUntil(L, i)')
    L = [2, 5, 1, 7, 4, 8, 3, 4, 9, 6]
    print(SumUntil(L, 2))  # 0
    print(SumUntil(L, 7))  # 8
    print(SumUntil(L, 4))  # 15
    print(SumUntil(L, -4))  # 49

    print('LastOccurrence(L, i)')
    L = [3, 4, 2, 5, 1, 4, 3, 4, 1]
    print(LastOccurrence(L, 0))  # -1
    print(LastOccurrence(L, 1))  # 8
    print(LastOccurrence(L, 2))  # 2
    print(LastOccurrence(L, 3))  # 6
    print(LastOccurrence(L, 4))  # 7
    print(LastOccurrence(L, 5))  # 3

    print('SumLast2()')
    L = List()
    L.AppendList([])
    print(L.SumLast2())  # 0

    L = List()
    L.AppendList([2302])
    print(L.SumLast2())  # 2302

    L = List()
    L.AppendList([5, 7, 8, 2, 1, 9, 2])
    print(L.SumLast2())  # 11

    L = List()
    L.AppendList([12, 5, 7, 8])
```
print(L.SumLast2())  # 15

print('SumFirstAndLast()')
L = List()
L.AppendList([])
print(L.SumFirstAndLast())  # 0

L = List()
L.AppendList([2302])  # 4604
print(L.SumFirstAndLast())

L = List()
L.AppendList([5,7,8,2,1,9,2])
print(L.SumFirstAndLast())  # 7

L = List()
L.AppendList([12,5,7,8,1,9,2])
print(L.SumFirstAndLast())  # 14

print('KeepFirstAndLast()')
L = List()
L.AppendList([])
L.KeepFirstAndLast()
L.Print()  # No output

L = List()
L.AppendList([2302])
L.KeepFirstAndLast()
L.Print()  # 2302

L = List()
L.AppendList([2401,2302])
L.KeepFirstAndLast()  # 2401 2302
L.Print()

L = List()
L.AppendList([5,7,8,2,1,9,2])
L.KeepFirstAndLast()
L.Print()  # 5 2

L = List()
L.AppendList([12,5,7,8,1,9,2])
L.KeepFirstAndLast()
L.Print()  # 12 2