CMPSC 9 F21
Intermediate Python Programming

Final Examination

Please state your answers as clearly as possible. This exam not only tests your understanding of the material, but also how well you can convey your understanding to us. Remember that you are solely responsible for answers to the questions, therefore please refrain from consulting with your class peers.

Please write all your answers **LEGIBLY and CLEARLY**. If we cannot decipher your answers, you will not receive credit. No electronic devices are allowed during the exam (calculators, cell phones, laptops, etc).

READ all questions carefully before attempting to answer them. If there are any ambiguities in the statement of questions, please ask us. **Clearly** label your answers if you use extra space in each question as scratch work.

THE GRADE IN THIS EXAM IS A TOTAL OF 84 POINTS.

Name (as it would appear on the official course roster)	**Umail Address**
	@umail.ucsb.edu

Question 1 (10 points): Write whether each statement is TRUE or FALSE**.** If FALSE state why **in 1 sentence.**

a. The underlying list in a Python dictionary does not shift items in order to make room for an item to be inserted.

b. The best case running time for Insertion Sort is O(n).

c. When **mutable** types are passed into a function, a copy of that variable is made and used within the function.

d. The order of inserting elements in a Binary Heap may affect whether or not the binary heap tree is balanced.

e. A subclass will inherit all fields and methods of its parent class (including the parent constructor).

f. The Binary Search Tree property must maintain a complete tree structure.

Question 2 (6 points):
Write the output next to each `print` statement for each subquestion below (treat each print statement independent of each other). If a runtime error occurs for a specific print statement, write **ERROR.**

```
class A:
   def x(self):
      return "A.x"

class B(A):
   def y(self):
      return "B.y"

class C(B):
   def z(self):
      return "C.z"

class D(A):
   def z(self):
      return "D.z"

b = B()
d = D()
```

```
print(b.x())
print(b.y())
print(b.z())
print(d.x())
print(d.y())
print(d.z())
```

_____
_____
_____
_____
_____
_____

Question 3 (10 points):
Write the Big-O notation next to the function for each subquestion below.

a.
```
# Assume s is a string with n characters
def f(s):
    if s == "":
        return ""
    print(s[:len(s)//2])
    return f(s[:len(s)//2])
```

b.
```
# Assume s is a string with n characters
def f(s):
    x = len(s)
    y = s
    while x > 0:
        s = y
        while len(s) > 0:
            s = s[:len(s)//2]
        x = x - 1
```

c.
```
# Assume n is a positive integer
def f(n):
   for i in range(n):
      print(i)
   for j in range(n):
      print(j)
   for k in range(10):
      print(k)
```

d.
```
# Assume aList is list containing n items
def f(aList):
    for i in range(len(aList)-1, 0, -2):
        for j in range(i):
            print(i, j)
```

e.
```
# Assume n is a positive integer
def f(n):
    if n < 500:
        return f(n*3)
    else:
        return 0
```

Question 4 (18 points):

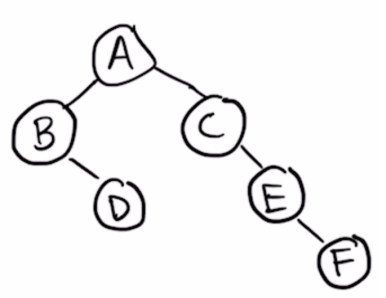
a. Construct a min heap (`minH`) object using the code below. Draw the state of the conceptual `minH` tree **after** the following code is executed.

```
minH.insert(30)
minH.insert(1)
minH.insert(50)
minH.insert(40)
minH.insert(60)
minH.insert(20)
minH.delMin()
minH.insert(10)
minH.insert(70)
minH.delMin()
```

b. Construct a binary tree (`bst`) object using the code below. Draw the state of the `bst` tree **after** the following code is executed.

```
bst.put(40)
bst.put(25)
bst.put(35)
bst.put(65)
bst.put(75)
bst.put(55)
bst.put(45)
bst.put(50)
bst.delete(40)
bst.delete(25)
```

c. Using the binary tree diagram below, write the order of nodes visited in an **in-order,** and **post-order** traversal of the binary tree starting at the root node.



In-order:

Post-order:

Question 5 (10 points):

a. In the space below, write a **recursive** Python function `countOddDigits(s)` that takes in a string `s` as its parameter and returns an integer representing the number of odd digits (`1,3,5,7,9`) in `s`. Your solution must be a complete function definition with appropriate syntax and indentation. Your solution must be recursive (and follow the three laws of recursion). You **may not** use the string's `.find` or `.replace` methods (hint: think about using the `in` operator), or use any helper functions. The following assert statements illustrate the correct functionality of `countOddDigits`:

```
assert countOddDigits("") == 0
assert countOddDigits("A") == 0
assert countOddDigits("A1B3") == 2
assert countOddDigits("123456") == 3
```

b. In the space below, illustrate the entire call stack you implemented in part a. for the following function call:

```
countOddDigits("A36")
```

You can illustrate this in various ways, but your illustration **must**:

- Show the function's input parameter state for each layer in the stack as it grows.
- Show the return expression that will (eventually) return to the caller of the function for each layer in the stack.
- Illustrate the specific values that are returned through **each** layer in the stack and the **final** result.

Question 6 (10 points):

a. In lecture and the textbook, a Deque data structure was implemented using a Python List where the rear was located in index `0` and the front was located in index `n-1` (assuming the Deque contains n elements).

   If we instead implemented the Deque data structure with a Linked List where the rear was located at the head of the Linked List and the front was located at the end of the Linked List, state what the Big-O notation for each of the following Deque operations: `addFront`, `addRear`, `removeFront`, and `removeRear`.

b. **Briefly (in 1 - 2 sentences)** state what will be printed if the `__str__` method is not defined in a class definition, and an instance of this object is used in a `print` statement.

c. **Briefly (in 2 - 4 sentences)** explain both the time and space tradeoff between Mergesort and Quicksort in the best and worst case scenarios.

Question 7 (10 points):
Write the output for the following code segment below.

Answer:

```
class A(Exception):
    pass

class B(A):
    pass

class C(A):
    pass

class D(C):
    pass


# Assume n is an integer
def f(n):

    try:
        if n < 0 and n >= -5:
            raise D()
        elif n >= 0 and n < 5:
            raise C()
        elif n >= 5 and n < 10:
            raise B()
        elif n > 10:
            raise A()
        print("0")
    except B:
        print("1")
    except D:
        print("2")
    except C:
        print("3")
    except A:
        print("4")
    except Exception:
        print("5")


f(15)
f(8)
f(-15)
f(1)
f(-3)
```

Question 8 (10 points):

Complete the recursive `binarySearch` and `mergeSort` algorithms below as covered in the textbook / lecture by filling in the blanks with the proper Python expression, operator, variable, or value.

a.

```
def mergeSort(aList):
    if len(aList) > 1:
        mid = len(aList) // 2
        lefthalf = aList[_____________________________]
        righthalf = aList[_____________________________]
        mergeSort(lefthalf)
        mergeSort(_____________________________)
        i = 0
        j = 0
        k = 0

        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] <= _____________________________[j]:
                aList[k] = _____________________________[i]
                i = i + 1
            else:
                aList[k] = _____________________________[j]
                j = j + 1
            k = _____________________________

        while i < len(_____________________________):
            aList[_____________________________] = lefthalf[i]
            i = i + 1
            k = k + 1
        while _____________________________ < len(_____________________________):
            aList[_____________________________] = righthalf[j]
            _____________________________ = j + 1
            k = k + 1
```

b.

```
# intList is a list of integers. item is the element to be searched for in intList.
# Return True if item is found in intList, otherwise return False
def binarySearch(intList, item):
    if len(intList) == _____________________________:
        return _____________________________

    mid = len(intList) // 2
    if intList[_____________________________] == item:
        return _____________________________
    elif _____________________________ < intList[mid]:
        return binarySearch(_____________________________, item)
    else:
        return binarySearch(_____________________________, item)
```

Scratch Page (do not detach)