Data Structures - CSCI-SHU 210

Mock Midterm Exam

First Name				
Last Name				
NetID				
Section	☐ Wed. (11:15)	☐ Wed. (15:45)	☐ Thursday	☐ Friday

Details

- **Duration:** 75 minutes
- **Max Score:** none
- Computers, calculators, phones, textbooks, or notebooks are not allowed

Instructions

- Fill the table with your information as it appears on Brightspace and Albert.
- Mark the section that aligns with the day (and time, if applicable) of your recitation.
- Show your NYU ID card to the proctor before signing the attendance sheet.
- You can use the blank space to make notes for yourself. Your notes are not evaluated.
- Turn off and place your mobile devices with your belongings at the front of the room.
- Please do not damage the exam paper.
- Do not discuss this quiz with anyone until the test is handed back.
- If you need to ask questions, please raise your hand and wait for the instructor.
- Stay seated until the instructor has collected all exams.

Please sign the statement below:

I, __________________, confirm that I am a student in this class and I will complete this exam without accessing any unauthorized information during the exam. I have read and understood the exam instructions above.

Multiple Choices Questions

After answering all knowledge questions, transfer your solution letter to the table below. Only one solution is correct for each answer option. Please mark your solution clearly:

Questions 1.1 to 1.10

	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10
Answer	C	A	A	A	B	D	A	A	B	B
Points										

Question 1.1. | Time Complexity

If $f(n) = \sqrt{n} + 2^{log_2 n}$ then $f(n)$ is:

□ A: $O(1)$

□ B: $O(\sqrt{n})$

□ C: $O(n)$

□ D: $O(log_2 n)$

Question 1.2. | Time Complexity

Analyze the worst-case time complexity of the following Python function:

```
def method1(n):
    i = 1
    total = 0
    while i*i < n:
        total += 1
        i += 1
    return total
```

□ A: $O(\sqrt{n})$

□ B: $O(n^2)$

□ C: $O(n)$

□ D: $O(log_2 n)$

Question 1.3. | FIFO

Assume that you start with an initially empty FIFO data structure. Considering a letter means enqueue and an asterisk (*) means dequeue, give the sequence of values returned by the dequeue operations for the following input: D * A T * A * * N Y * U * *

□ A: DATANYU

□ B: AN

□ C: DTY

□ D: ATANY

Question 1.4. | Time Complexity

Analyze the worst-case time complexity of the following Python function:

```
def fun(n,m):
    arr=[[0] * m for i in range(n)]
    for i in range(n):
        for j in range(m):
            k = 1
            while k < n * m:
                k *= 2
```

- ☐ A: $O(n * m * log_2 (n * m))$
- ☐ B: $O(n * m)$
- ☐ C: $O(n * m))$
- ☐ D: $O(n^2)$

Question 1.5. | Recursive Calculation

What does the function `fMystery(a)` calculate?

```
def fMystery(a):
   if a == 0:
      return 0
   else:
      return fMystery(a-1) + 3*a**2 – 3*a + 1
```

- ☐ A: a
- ☐ B: a^3
- ☐ C: a^4
- ☐ D: a^2

Question 1.6. | Recurrence Relation

Solve the following recurrence relation:

```
T(n) = T(n / 2) + 1
```

☐ A: $O(n^2)$

☐ B: $O(n)$

☐ C: $O(nlog_2n)$

☐ D: $O(log_2n)$

Question 1.7. | Recursive Reading

What is the output of the following Python function?

```
def f(n):
    if n <= 0:
        return
    print("n")
    f(n // 20)
    print("m")
    f(n // 20)

f(400)
```

☐ A: n n n m m n m m n n m m n m

☐ B: n n n m n m n m n m n m n m

☐ C: n m n m n m n m n m n m n m

☐ D: m n m n m n m n m n m n m n

Question 1.8. | Armortised Time Complexity

What is the amortized time complexity for the array append operation using the doubling strategy (if the array is full, we resize the array into an array with double the capacity)?

☐ A: $O(1)$ *amortised*

☐ B: $O(n)$ *amortised*

☐ C: $O(log_2n)$ *amortised*

☐ D: $O(n^2)$ *amortised*

Question 1.9. | Code Reading

What will be the output of the following code?

```
def g(s):
    a = s.split(" ")
    b = []
    for i in range(len(a)):
        if a[i] == '(':
            b.append('*')
        if a[i] == ')':
            if len(b) > 0:
                b.pop()
    return len(b)
print(g("( ( ) ) )"))
```

- □ A: 1
- □ B: 0
- □ C: 2
- □ D: 3

Question 1.10. | References

Select the correct representation for the following functions and operations:

```
def fun1(c, r):
    return [[0] * c] * r
x = fun1(3, 3)
x[0][0] = 2
print(x)

def fun2(c, r):
    return [[0] * c for j in range(r)]
y = fun2(3, 3)
y[0][0] = 4
print(y)
```

- □ A: `x=[[2,0,0],[2,0,0],[2,0,0]] y=[[4,0,0],[4,0,0],[4,0,0]]`
- □ B: `x=[[2,0,0],[2,0,0],[2,0,0]] y=[[4,0,0],[0,0,0],[0,0,0]]`
- □ C: `x=[[0,0,0],[0,0,0],[0,0,0]] y=[[4,0,0],[4,0,0],[4,0,0]]`
- □ D: `x=[[2,0,0],[0,0,0],[0,0,0]] y=[[4,0,0],[0,0,0],[0,0,0]]`

2. Programming Question

Question 2.1. - Sum of Nested Lists

Write the recursive Python function `summer(lis)` that takes a list of nested lists *L* and returns the sum of all values. Your function has to be recursive.

Example 1:

```
L = [ [1], [2, 3], [4], [3, [2, 4] ] ]
res = summer(L)
print(res)  # should print: 19
```

Example 2:

```
L = [ [ [ [1] ], [2] ], [3] ]
res = summer(L)
print(res)  # should print: 6
```

```
def summer(lis):
       total = 0
           for i in L:
               if isinstance(i, list):
                   total += sum(i)
               else:
                   total += i
           return total
```

Question 2.2. - Rearrange Even-Odd

Implement the member `rearrange_even_odd(self)` to sort a SinglyLinkedList so that nodes with even elements appear first and odd nodes with odd elements appear last. You can find the SinglyLinkedList implementation for your reference in Appendix 1.

Requirements:

- You can not use nonlocal or global variables or use any library.
- You are not allowed just to swap node elements.
- Your function has to work in place.
- You can not create a new SinglyLinkedList instance.
- You can not delete nodes or create new nodes.
- Your function has to connect all nodes correctly.
- You can only use the provided class members.
- You can write your own functions.

Example 1:

```
sll = SingleLinkedList()
sll.insertAtFirst(8)
sll.insertAtFirst(7)
sll.insertAtFirst(6)
sll.insertAtFirst(5)
sll.insertAtFirst(4)
sll.insertAtFirst(3)
sll.insertAtFirst(2)
sll.insertAtFirst(1)

print(sll)  # should print: Head-->1-->2-->3-->4-->5-->6-->7-->8-->None
sll.rearrange_even_odd()
print(sll)  # should print: Head-->2-->4-->6-->8-->1-->3-->5-->7-->None
```

Example 2:

```
sll = SingleLinkedList()
sll.insertAtFirst(6)
sll.insertAtFirst(4)
sll.insertAtFirst(1)
sll.insertAtFirst(2)

print(sll)  # should print: Head-->2-->1-->4-->6-->None
sll.rearrange_even_odd()
print(sll)  # should print: Head-->2-->4-->6-->1-->None
```

```
def rearrange_even_odd(self):
       if self._head is None:
                   return

               # Separate even and odd nodes into separate lists
               even_head = even_tail = None
               odd_head = odd_tail = None
               curr = self._head
               while curr:
                   if curr._element % 2 == 0:  # even node
                       if not even_head:
                           even_head = even_tail = curr
                       else:
                           even_tail._next = curr
                           even_tail = curr
                   else:  # odd node
                       if not odd_head:
                           odd_head = odd_tail = curr
                       else:
                           odd_tail._next = curr
                           odd_tail = curr
                   curr = curr._next

               # Connect even and odd lists
               if even_tail:
                   even_tail._next = odd_head
               else:
                   even_head = odd_head
               if odd_tail:
                   odd_tail._next = None

               self._head = even_head
```

Appendix 1 - SinglyLinkedList Implementation

```
class Node:
    def __init__(self, element=None, next=None):
        self._element = element
        self._next = next

class SingleLinkedList:
    def __init__(self):
        self._head = None
        self._size = 0

    def __len__(self):
        return self._size

    def is_empty(self):
        return self._size == 0

    def insertAtFirst(self, e):
        newNode = Node(e, self._head)
        self._head = newNode
        self._size += 1
        return newNode

    def __str__(self):
        result = "Head-->"
        currNode = self._head
        while currNode is not None:
            result += str(currNode._element) + "-->"
            currNode = currNode._next
```

You can tear off this page and use it as scratch paper.