Started on	Sunday, 31 January 2021, 9:05 AM
State	Finished
Completed on	Sunday, 31 January 2021, 10:12 AM
Time taken	1 hour 6 mins
Grade	Not yet graded

Question **1**

Complete

Marked out of 3.00

*A sparse data structure is one where we presume most of the elements have a common value (e.g. 0). Write the **SparseList** class, which implements a sparse list data structure, so that the following code works as specified by the comments. Each comment refers to the line(s) below it. Elements not explicitly set will have the default value 0. The list's length is given by the element set using the largest index (hint: use the **__setitem__** and **__getitem__** methods). Do not represent 0 values in memory! Specifications or tests are not required **[3p]**.*

```
# Initialise the sparse list
data = SparseList()
# Add elements to the sparse list
data[1] = "a"
data[4] = "b"
data[9] = "c"
# Element at index 14 is 'd'
data[14] = "d"
# append adds the element after the last
# initialised index
data.append("z")
# Prints:
# 0 a 0 0 b 0 0 0 0 c 0 0 0 0 d z
for i in range(0, len(data)):
    print(data[i])
```

```
class SparseList():
    def __init__(self):
        self.__data = {}
        self.__last = 0

    def __setitem__(self, key, value):
        self.__data[key]=value
        if key>self.__last:
            self.__last = key

    def __getitem__(self, item):
        if item in self.__data:
            return self.__data[item]
        else:
            return 0

    def append(self,value):
        self.__last+=1
        self.__data[self.__last]=value

    def __len__(self):
        if self.__last == 0:
            return 0
        else:
            return self.__last+1
```

Question **2**

Complete

Marked out of 2.00

Analyse the time and extra-space complexity of the following function **[2p]**.

```
def f(n):
    s = 0
    for i in range(1, n * n + 1):
        while i > 0:
            s = s + i
            i -= 1
    return s
```

The function consists of a for loop (which repeats n^2 times) in which we perform a while that repeats i times. We deduce that the time complexity is:
T(n) = sum from i=1 to n^2 of i.
T(n) = 1 + 2 + 3 +... + n^2 = n^2(n^2+1)/2= 1/2(n^4+n^2) which is Θ(n^4)

Since the memory remains constant (i and s are the only variables), the extra memory complexity is Θ(1)

Question **3**

Complete

Marked out of 4.00

Write the specification, Python code and test cases for a ***recursive function*** that calculates the greatest common divisor of the numbers found on odd positions of a list, using the ***divide and conquer*** method. For the input [2, **35**, 5, **65**, 10, **20**, 5] the greatest common divisor is between 35, 65 and 20 and its value is 5. You may divide the implementation into several functions according to best practices. Specify and test all functions **[4p]**.

```
def gcd(a,b):
    """
    calculates the greatest common divisor of a and b using Euclid's algorithm
    :param a, b: positive integers
    :return: the gcd of a and b
    """
    if b > a:
        b, a = a, b
    if b == 0:
        return a
    else:
        return gcd(b,a%b)

def arr_gcd(arr,even = 1):
    """
    computes the gcd of the elements of an array using divide and conquer
    :param arr: list of integers
    :param even: used internally, if even==1  the slice begins with an even index
    :return : the gcd
    """
    if len(arr)<2: return 0
    elif len(arr)==2:
        return arr[even]
    else:
        m = len(arr)//2
        return gcd(arr_gcd(arr[:m],even),arr_gcd(arr[m:],(even+m%2)%2))
```

```
class arr_gcdTest(unittest.TestCase):
    def testA(self):
        v = [2, 35, 5, 65, 10, 20, 5]
        self.assertEqual(arr_gcd(v),5)
    def testB(self):
        v = [1,2,1,2,1,2]
        self.assertEqual(arr_gcd(v),2)
    def testC(self):
        v = [1,2,1,2,1,2,1]
        self.assertEqual(arr_gcd(v),2)
    def testD(self):
        v = [0,0,0,0,0]
        self.assertEqual(arr_gcd(v),0)
```

◄ Practice Quiz

Jump to...