Started on	Sunday, 31 January 2021, 9:06 AM
State	Finished
Completed on	Sunday, 31 January 2021, 10:16 AM
Time taken	1 hour 9 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 a sparse list
data1 = SparseList()
# Add elements
data1[1] = "a"
data1[3] = "b"
data1[5] = "c"

# Initialise another sparse list
data2 = SparseList()
# Add elements
data2[0] = "x"
data2[3] = "z"
# List concatenation
data = data1 + data2
# Prints:
# 0 a 0 b 0 c x 0 0 z
for i in range(0, len(data)):
    print(data[i])
```

```
class SparseList():


    def __init__(self):
        self.list=list()


    def __setitem__(self, index, value):

        if index + 1 > 0:
            list2 = [0] * (index+1)
            for x in range(len(self.list)):
                list2[x] = self.list[x]
            self.list = list2

        self.list.__setitem__(index, value)

    def __getitem__(self, index):

        try:
            return list.__getitem__(self, index)
        except IndexError:
            return 0

    def __add__(self, other):
        return self.list + other.list
```

Question **2**

Complete

Marked out of 2.00

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

```
def f(a,b):
    if (a!=1):
        f(a-1,b-1)
        print (a,b)
        f(a-1,b-1)
    else:
        print (a,b)
```

IMG_1034.JPG

Question **3**

Complete

Marked out of 4.00

Write the specification, Python code and test cases for a **recursive function** which uses the **divide and conquer** programming technique to calculate the sum of the numbers found on prime positions in a list of natural numbers. Calling the function for [5, 6, **7**, **8**, 9, **10**, 11, **12**, 13, 14, 15, **16**, 17, **18**, 19] will return 7 + 8 + 10 + 12 + 16 + 18 = **71**. Divide the implementation into several functions according to best practices. Specify and test all functions **[4p]**.

```
#######FUNCTIONS#######

def check_prime(n):
    """
    Checks if a number is prime
    :param n: integer number to be checked
    :return: True if number is prime and False if number is not prime
    """
    if n <= 1:
        return False
    else:
        for i in range(2, n):
            if (n % i) == 0:
                return False
        return True


def get_sum_divide_and_conquer(array):
    """
    Recursive function implemented with divide and conqer
    The function calculates the sum of the numbers found on prime positions in a list of
natural numbers
    :param array: list of natural numbers
    :return: sum of the numbers found on prime positions
    """
    if len(array) <= 1:
        if check_prime(global_array.index(array[0])):
            return array[0]

    middle = len(array) // 2
    sum_left = get_sum_divide_and_conquer(array[:middle])
    sum_right = get_sum_divide_and_conquer(array[middle:])

    return get_sum_divide_and_conquer(sum_left) + get_sum_divide_and_conquer(sum_right)


global_array = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

#######TESTS#######


def test_check_prime():
    assert check_prime(4) == False
    assert check_prime(15) == False
    assert check_prime(1) == False
    assert check_prime(2) == True
    assert check_prime(11) == True


def test_get_sum_divide_and_conquer():
    assert get_sum_divide_and_conquer([5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
== 71
    assert get_sum_divide_and_conquer([5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]) == 53
    assert get_sum_divide_and_conquer([5, 6, 7, 8, 9, 10, 11, 12]) == 37


test_check_prime()
test_get_sum_divide_and_conquer()
```

◀ Practice Quiz Jump to...