Started on	Sunday, 31 January 2021, 9:06 AM
State	Finished
Completed on	Sunday, 31 January 2021, 10:15 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.__data = []

    def __setitem__(self, key, value):
        self.__data.append((key, value))

    def __getitem__(self, item):
        for elem in self.__data:
            if elem[0] == item:
                return elem[1]
        return 0

    def __len__(self):
        max_index = 0
        for elem in self.__data:
            max_index = max(max_index, elem[0])
        return max_index + 1

    def __add__(self, other):
        concat_data = SparseList()
        for elem in self.__data:
            concat_data[elem[0]] = elem[1]
        len_1 = len(self)
        for elem in other.__data:
            concat_data[len_1 + elem[0]] = elem[1]
        return concat_data


data1 = SparseList()
data1[1] = 'a'
data1[3] = 'b'
data1[5] = 'c'

data2 = SparseList()
data2[0] = 'x'
data2[3] = 'z'

data = data1 + data2

for i in range(0, len(data)):
    print(data[i])
```

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)
```

2nd_exercise.pdf

Question **3**

Complete

Marked out of 4.00

Write the specification, Python code and test cases for a **recursive function** which returns the largest prime number found on an even position within a list of integers using the **divide and conquer** programming technique**.** If the list does not contain at least one prime number on an even position, return **None**. Calling the function for [5, 6, 7, 8, 9, 10, 4, 12, 13, 14, 15, 16, 17, 25, **19**] will return 19. Calling the function for [21, 3, 50] will return **None**. Divide the implementation into several functions according to best practices. Specify and test all functions **[4p]**.

```
import math


def is_prime(n):
    """
    Checks if a given number is prime or not.
    :param n: The number to be checked for primality; integer
    :return: True if the number is prime; False otherwise
    """
    if n <= 1:
        return False
    elif n == 2:
        return True
    elif n % 2 == 0:
        return False
    for d in range(3, int(math.sqrt(n)) + 1):
        if n % d == 0:
            return False
    return True


def largest_prime_on_even(some_list):
    """
    Returns the largest prime number in a list
    :param some_list: Find the largest prime from this list; list or integers
    :return: the largest prime if there is one; -1 otherwise
    """
    if len(some_list) == 0:
        return -1
    elif len(some_list) == 2 or len(some_list) == 1:
        if is_prime(some_list[0]):
            return some_list[0]
        else:
            return -1
    elif len(some_list) > 2:
        if is_prime(some_list[0]):
            return max(some_list[0], largest_prime_on_even(some_list[2:]))
        else:
            return max(-1, largest_prime_on_even(some_list[2:]))


def caller_largest_prime_on_even(some_list):
    """
    Function that just calls the largest <largest_prime_on_even> function. If the return of that function is
    -1 this function returns None, otherwise it just returns the return of <largest_prime_on_even>.
    :param some_list: A list of integers to find the largest prime from
    :return: None if <largest_prime_on_even> returns -1 on <some_list>, otherwise it returns the return value
    of <largest_prime_on_even>.
    """
    largest_prime = largest_prime_on_even(some_list)
    if largest_prime == -1:
        return None
    else:
        return largest_prime


def test_is_prime():
    assert is_prime(1) is False
    assert is_prime(2) is True
    assert is_prime(20) is False
    assert is_prime(17) is True
    assert is_prime(19) is True
    assert is_prime(121) is False


def test_caller_largest_prime_on_even():
    assert caller_largest_prime_on_even([5, 6, 7, 8, 9, 10, 4, 12, 13, 14, 15, 16, 17, 25, 19]) == 19
    assert caller_largest_prime_on_even([8, 7, 10, 3]) is None
    assert caller_largest_prime_on_even([]) is None
```

```
    assert caller_largest_prime_on_even([17]) == 17
    assert caller_largest_prime_on_even([17, 10, 23]) == 23
    assert caller_largest_prime_on_even([4, 6, 8, 10, 12]) is None


test_is_prime()
test_caller_largest_prime_on_even()
```

◄ Practice Quiz

Jump to...