Started on	Sunday, 31 January 2021, 9:05 AM
State	Finished
Completed on	Sunday, 31 January 2021, 10:15 AM
Time taken	1 hour 10 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.__sparse_list = {}

    def __setitem__(self, index, value):
        self.__sparse_list[index] = value

    def __getitem__(self, index):
        try:
            return self.__sparse_list[index]
        except KeyError:
            return 0

    def append(self, value):
        try:
            self.__sparse_list[max(self.__sparse_list.keys())+1] = value
        except ValueError:
            self.__sparse_list[0] = value

    def __len__(self):
        try:
            return max(self.__sparse_list.keys())+1
        except ValueError:
            return 0


# 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], end=" ")
```

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)
```

Ex2_Truta_David.pdf

Question **3**

Complete

Marked out of 4.00

Write the specification, Python code and test cases for a ***recursive function*** that calculates the product of prime numbers found on multiple of 3 positions in a list, using the ***divide and conquer*** method. Return ***None*** if no suitable prime number is found. For the input [0, 1, 2, **31**, 4, 5, **2**, 7, 8, **9**, 10] the product is 31 * 2 = 62. Divide the implementation into several functions according to best practices. Specify and test all functions **[4p]**.

```
def prime(n):
    if n % 2 == 0 and n > 2 or n < 2:
        return False
    for d in range(3, n // 2 + 1, 2):
        if n % d == 0:
            return False
    return True
```

```
def divide_and_conquer(list, left, right):
    if left % 3 == 0:
        if prime(list[left]):
            return list[left]
    if right % 3 == 0:
        if prime(list[right]):
            return list[right]
    return divide_and_conquer(list, left + 3, right - right % 3)
```

◄ Practice Quiz

Jump to...