A1Q1

In [1]:

```
# Algorithm and code independently designed by Shao Shi

def Print_values(a, b, c):
    if a > b:
        if b > c:
            return compute(a, b, c)
        else:
            if a > c:
                return compute(a, c, b)
            else:
                return compute(c, a, b)
    else:  # a<=b
        if not b > c:
            return compute(c, b, a)


def compute(x, y, z):
    return x + y - 10 * z
```

In [2]:

```
print(Print_values(10, 5, 1))
```

```
5
```

A1Q2

In [3]:

```
from math import ceil


# Algorithm and code independently designed by Shao Shi
# This algorithm is an implementation of recursion with a memo
# @parameters
# x is the index of the desired term

def my_list(x):
    memo = list(range(-1, -x - 1, -1))
    return core(x, memo)


def core(x, memo):
    if memo[x - 1] >= 0:
        return memo[x - 1]
    else:
        if x == 1:
            memo[0] = 1
            return 1
        else:
            memo[x - 1] = core(ceil(x / 3), memo) + 2 * x
            return memo[x - 1]
```

In [4]:

```
# this is a demo of the function of my_list()
from A1Q2 import my_list

# Algorithm and code independently designed by Shao Shi
```

```
N = 100
listN = list(range(-1, -N-1, -1))
# print(listN)
for i in range(0, N, 1):
    listN[i] = my_list(i+1)

print(listN)
```

```
[1, 5, 7, 13, 15, 17, 21, 23, 25, 33, 35, 37, 41, 43, 45, 49, 51, 53, 59, 61, 63, 67, 69, 71, 7
5, 77, 79, 89, 91, 93, 97, 99, 101, 105, 107, 109, 115, 117, 119, 123, 125, 127, 131, 133, 135,
141, 143, 145, 149, 151, 153, 157, 159, 161, 169, 171, 173, 177, 179, 181, 185, 187, 189, 195, 1
97, 199, 203, 205, 207, 211, 213, 215, 221, 223, 225, 229, 231, 233, 237, 239, 241, 253, 255, 25
7, 261, 263, 265, 269, 271, 273, 279, 281, 283, 287, 289, 291, 295, 297, 299, 305]
```

A1Q3

In [5]:

```
# A1Q3_1 this is a function design problem with no output, please check the code
import numpy as np


# Algorithm and code independently designed by Shao Shi
# This algorithm is an implementation of recursion with a memo
# @parameters
# x is the target number of sum
# diceCount is the number of dices
# faceCount is the number of faces of each dice, each face have an integer value from 1 to faceC

def Find_number_of_ways(x, diceCount, faceCount):
    core_memo = np.zeros((x + 1, diceCount + 1), dtype=np.int64) - 1  # create a memo to store re
    return core(x, diceCount, faceCount, core_memo)


def core(x, diceCount, faceCount, core_memo):
    pathCount = 0
    if diceCount < 0 or x < 0:
        pathCount = 0
    elif diceCount == 0 and x == 0:
        pathCount = 1
    else:
        if core_memo[x][diceCount] != -1:  # search for result in memo
            return core_memo[x][diceCount]
        else:
            for i in range(1, faceCount + 1):
                pathCount += core(x - i, diceCount - 1, faceCount, core_memo)
        core_memo[x][diceCount] = pathCount
    return pathCount
```

In [6]:

```
# A1Q3_2
import A1Q3_1 as dice

# Algorithm and code independently designed by Shao Shi

Number_of_ways = list(range(10, 61, 1))
for x in range(10, 61, 1):
    Number_of_ways[x-10] = dice.Find_number_of_ways(x, 10, 6)

print(Number_of_ways)
```

```
[1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147940, 243925, 383470, 576565, 83
1204, 1151370, 1535040, 1972630, 2446300, 2930455, 3393610, 3801535, 4121260, 4325310, 4395456,
4325310, 4121260, 3801535, 3393610, 2930455, 2446300, 1972630, 1535040, 1151370, 831204, 576565,
383470, 243925, 147940, 85228, 46420, 23760, 11340, 4995, 2002, 715, 220, 55, 10, 1]
```

A1Q4

In [7]:

```
# A1Q4_1
import random


# Algorithm and code independently designed by Shao Shi

def Random_integer(N):
    rand_list = [0 for i in range(N)]
    for i in range(N):
        rand_list[i] = random.randint(0, 10)

    return rand_list
```

In [8]:

```
# A1Q4_2_method1
# Algorithm and code independently designed by Shao Shi
# in my algorithm, subsets was found firstly, and the average is calculated based on them
# the subset finding algorithm is O(n*2^n)


def Sum_averages(inputList):
    subsets = Find_subsets(inputList)
    averageSum = 0
    for i in range(1, len(subsets), 1):
        averageSum += sum(subsets[i]) / len(subsets[i])
    return averageSum


def Find_subsets(inputList):
    if not inputList:
        return [[]]
    else:
        # lastSetList = Find_subsets(inputList[:-1])
        # a bug in recursive python code have occurred
        return Find_subsets(inputList[:-1]) + Setlist_append_term(Find_subsets(inputList[:-1]), i


def Recursive_function(inputList):
    return Find_subsets(inputList[:-1]) + Setlist_append_term(Find_subsets(inputList[:-1]), inpu


def Setlist_append_term(inputSetList, term):
    if not inputSetList:
        return [[term]]
    else:
        out = inputSetList.copy()
        for i in range(len(inputSetList)):
            out[i] += [term]
        return out
```

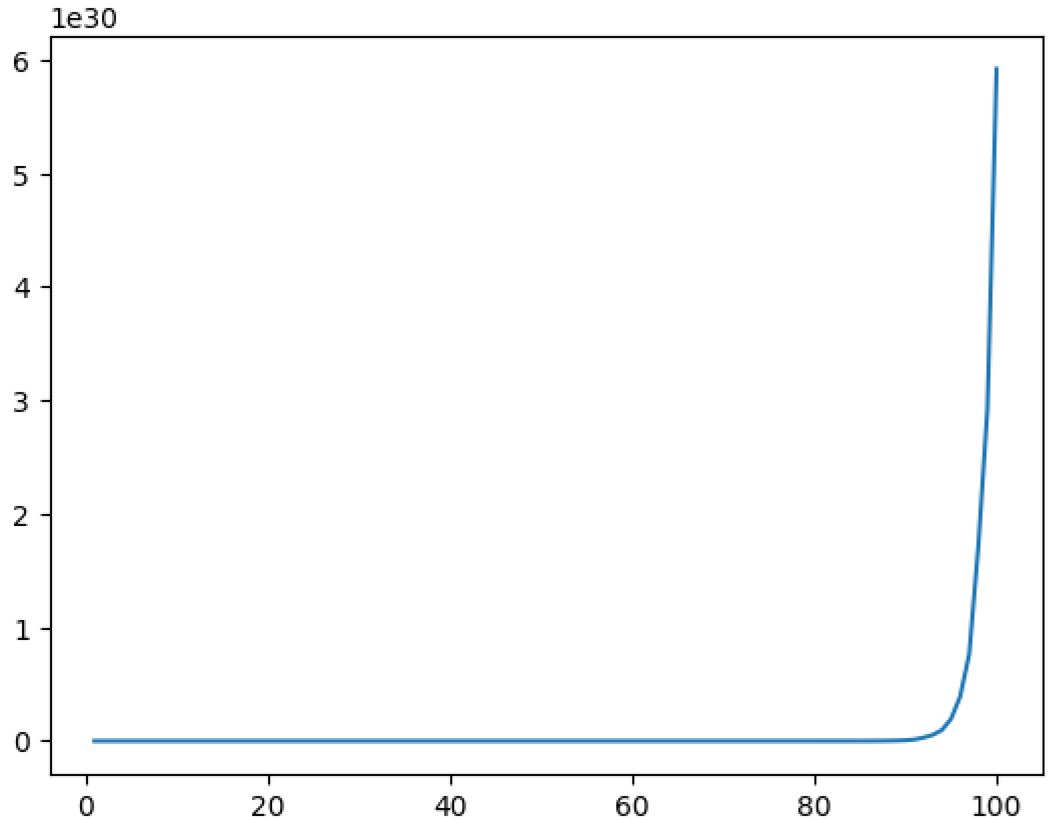
In [9]:

```
# A1Q4_2_method2
from scipy.special import comb
```

```
# Algorithm and code independently designed by Shao Shi

def Combination(n, m):
    # a very slow implementation of combination computation
    # not used in this question
    # choose m element in a set of n numbers
    if m == n:
        return 1
    elif m == 1:
        return n
    else:
        return Combination(n - 1, m - 1) + Combination(n - 1, m)


def Sum_averages(inputList):
    averageSum = 0
    n = len(inputList)
    listSum_over_n = sum(inputList) / n
    for i in range(n):
        # averageSum = Combination(n, i + 1) * listSum_over_n
        averageSum += comb(n, i + 1) * listSum_over_n
    return averageSum
```

In [10]:

```
# A1Q4_3
from matplotlib import pyplot as plt
from A1Q4_1 import Random_integer
import A1Q4_2_method1
import A1Q4_2_method2
# Algorithm and code independently designed by Shao Shi
# in method1, calculating loop = 100 have an intensive running time requirement, but it is correc
# the reason is that, in my algorithm, subsets was found firstly, and the average is calculated b
# the subset finding algorithm is O(n*2^n)
# method2 used some tricky mathematical simplification to avoid the problem in method1

loop = 100
xList = list(range(1, loop + 1, 1))
yList = list(range(1, loop + 1, 1))
'''
# method1: find subsets and get sum
for N in range(1, loop + 1, 1):
    randList = Random_integer(N)
    Total_sum_averages = A1Q4_2.Sum_averages(randList)
    yList[N-1] = Total_sum_averages
plt.plot(xList, yList)
plt.show()
'''
# method2: find subsets and get sum
for N in range(1, loop + 1, 1):
    randList = Random_integer(N)
    Total_sum_averages = A1Q4_2_method2.Sum_averages(randList)
    yList[N-1] = Total_sum_averages
plt.plot(xList, yList)
plt.show()
```



A1Q5

In [11]:
```
# A1Q5_1
import numpy as np


# Algorithm and code independently designed by Shao Shi

def Matrix_generator(N, M):
    matrix_out = np.random.randint(2, size=(N, M))
    matrix_out[[0, -1], [0, -1]] = 1
    return matrix_out
```

In [12]:
```
print(Matrix_generator(10, 20))
```

```
[[1 0 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 0]
 [1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1]
 [0 1 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0]
 [1 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0]
 [0 0 1 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 0 1]
 [0 1 1 1 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1]
 [0 1 1 0 1 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0]
 [0 1 1 0 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 0]
 [0 1 0 1 0 1 1 0 1 1 0 1 0 0 0 0 1 1 1 1]]
```

In [13]:
```
#A1Q5_2 this problem requires no output, only the function design
import numpy as np


# Algorithm and code independently designed by Shao Shi
# This algorithm is an implementation of recursion with a memo, and is O(n)
# @parameters
```

```python
# randMatrix is the random matrix created in A1Q5_1
# x is the row number of the term in the randMatrix
# y is the column number of the term in the randMatrix
# matrix_compute is the matrix for path computation
# memo is the matrix storing recursive results

def Count_path_core(x, y, matrix_compute, memo):
    if memo[x, y] != -1:
        return memo[x, y]
    else:
        if x == 0 or y == 0:
            memo[x, y] = 0
            return 0
        elif x == 1 and y == 1:
            memo[x, y] = 1
            return 1
        else:
            memo[x, y] = matrix_compute[x, y] * (
                        Count_path_core(x - 1, y, matrix_compute, memo) + Count_path_core(x, y -
                                                                                        memo))
            return memo[x, y]


def Count_path(randMatrix):
    randMatrix_shape = randMatrix.shape
    memo = np.zeros((randMatrix_shape[0] + 1, randMatrix_shape[1] + 1), dtype=np.int64) - 1
    matrix_compute = np.zeros((randMatrix_shape[0] + 1, randMatrix_shape[1] + 1), dtype=np.int64
    matrix_compute[1:, 1:] = randMatrix
    return Count_path_core(randMatrix_shape[0], randMatrix_shape[1], matrix_compute, memo)
```

In [14]:

```python
#A1Q5_3
from A1Q5_1 import Matrix_generator
from A1Q5_2 import Count_path

# Algorithm and code independently designed by Shao Shi

N = 10
M = 8
loop = 1000
path_count = 0

for i in range(loop):
    path_count += Count_path(Matrix_generator(N, M))
print(path_count/loop)
```

0.334