In [11]:
```
#Assignment 01-1.Flowchart
#Define a function
def Print_values():
    #Input a b c
    a=input("Please input a number of a: ")
    b=input("Please input another number of b: ")
    c=input("Please input another number of c: ")
    if int(a)>int(b):
        if int(b)>int(c):
            i=int(a)+int(b)-10*int(c)
            print(a+","+b+","+c)
        else:
            if int(a)>int(c):
                i=int(a)+int(c)-10*int(b)
                print(a+","+c+","+b)
            else:
                i=int(c)+int(a)-10*int(b)
                print(c+","+a+","+b)
    else:
        if int(b)>int(c):
            print("nothing")
        else:
            i=int(c)+int(b)-10*int(a)
            print(c+","+b+","+a)
    print("x+y-10z="+str(i))
Print_values()
```

10,5,2
x+y-10z=-5

In [12]:
```
# Assignment 01-2. Continuous celing function
#用input传入多个值并储存到一个list
#参考https://blog.csdn.net/qq_45261963/article/details/107971245
from math import *
x=input("please input a list of N positive integers, the numbers are separated by co
x_list=x.split(",")  #以“,”为分隔符分隔
x_list=[int(x_list[i]) for i in range(len(x_list))] #将x_list转变为整数类型
print(x_list)
def F_ceil(X):
    if X==1:
        return 1  #return思路来自焦小乔同学
    else:
        return F_ceil(ceil(X/3))+2*X
for n in x_list:
    print(str(F_ceil(n))+",",end="")
```

[1, 2, 3, 4, 5, 6, 21]
1,5,7,13,15,17,63,

In [13]:
```
#Assignment 01-3. Dice rolling-3.1
def Find_number_of_ways(N,X):
    #总和小于骰子数或大于骰子数*6，不可能
    if X<N or X>6*N:
        return 0
    #一个骰子，总和在1~6之间，一种情况
    elif N==1:
        return 1
    #迭代求和
    else:
        return Find_number_of_ways(N-1,X-1)+Find_number_of_ways(N-1,X-2)+Find_numbe
print(Find_number_of_ways(10,11))
```

```
print(Find_number_of_ways(10,10))
print(Find_number_of_ways(10,61))
#迭代思路来自焦小乔同学与https://www.geeksforgeeks.org/dice-throw-dp-30/
```

```
10
1
0
```

In [14]:
```
#Assignment 01-3. Dice rolling-3.2
#给定一个空的list
Number_of_ways=[]
for i in range(10,61):
        Number_of_ways.append(Find_number_of_ways(10,i))
print(Number_of_ways)
#找出总和在10~60之间对应的最大方法数
M=max(Number_of_ways)
print(M)
#返回最大方法数对应的总和X
d_M=Number_of_ways.index(max(Number_of_ways))
print(int(d_M)+10)
```

```
[1, 10, 55, 220, 715, 2002, 4995, 11340, 23760, 46420, 85228, 147940, 243925, 38347
0, 576565, 831204, 1151370, 1535040, 1972630, 2446300, 2930455, 3393610, 3801535, 41
21260, 4325310, 4395456, 4325310, 4121260, 3801535, 3393610, 2930455, 2446300, 19726
30, 1535040, 1151370, 831204, 576565, 383470, 243925, 147940, 85228, 46420, 23760, 1
1340, 4995, 2002, 715, 220, 55, 10, 1]
4395456
35
```

In [15]:
```
#Assignment 01-4. Dynamic programming-4.1
import numpy as np
def Random_integer(N):
    arr1=np.random.randint(0,11,int(N))
    return arr1
Random_integer(4)
```

Out[15]:
```
array([7, 6, 7, 5])
```

In [17]:
```
#Assignment 01-4. Dynamic programming-4.2
#利用Random_integer函数给定一个含有n个元素的在0~10之间的随机整数数组
import math
n=input("please input a number representing the number of elements,n:")
arr2=Random_integer(n)
print(arr2)
#定义Sum_averages函数
def Sum_averages(n):
    #先把最大子集的平均和所有最小子集（除空集）的平均相加
    S=np.mean(arr2)+arr2.sum()
    #再算其他子集的平均
    for i in range(1,int(n)-1):
        S+=(math.factorial(int(n)-1)/math.factorial(i)/math.factorial(int(n)-1-i))*
    return S
Sum_averages(n)
#解题思路来自https://www.geeksforgeeks.org/sum-average-subsets/
#举例解释：arr= [a0,a1, a2, a3], sum of average = a0/1 + a1/1 + a2/1 + a3/1 +(a0+a1)
#接上条注释：A = (a0+a1+a2+a3), then above expression can be rearranged as below,sum
```

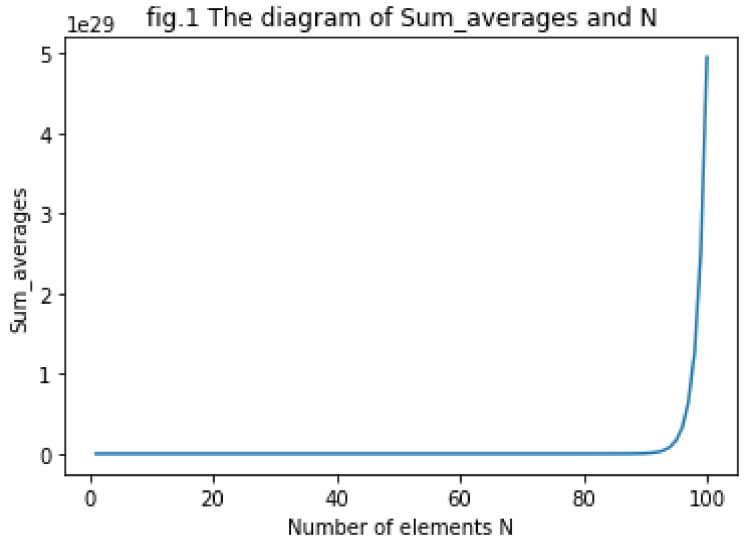
Out[17]:
```
[ 8  7  8  6 10]
241.8
```

In [18]:
```
#Assignment 01-4. Dynamic programming-4.3
import matplotlib.pyplot as plt
#assign the output to a list called Total_sum_averages
```

```
Total_sum_averages=[]
for i in range(1,101):
    Total_sum_averages.append(Sum_averages(i))
print(Total_sum_averages)
#作图
x=np.linspace(1,100,100)
y=Total_sum_averages
plt.xlabel("Number of elements N")
plt.ylabel("Sum_averages")
plt.title("fig.1 The diagram of Sum_averages and N")
plt.plot(x,y)
```

```
[46.8, 46.8, 85.8, 144.3, 241.8, 410.8, 709.8, 1246.05, 2217.8, 3993.6, 7261.8, 13313.3, 24577.8, 45643.37142857143, 85199.40000000001, 159746.925, 300697.8, 567982.1333333333, 1076173.8, 2044727.1, 3894714.942857143, 7435361.345454546, 14224165.8, 27262980.55, 52344918.6, 100663300.8, 193870056.46666667, 373892247.3, 721998817.8, 1395864376.3999999, 2701672981.8, 5234491397.362499, 10151740886.89091, 19706320540.094116, 38286565615.62858, 74446099802.96666, 144868086097.80002, 282111536079.85266, 549755813893.8, 1072023837087.4501, 2091753828457.8, 4083900331745.3716, 7977851810845.8, 15593073993920.209, 30493122476993.8, 59660457020199.5, 116782171188469.8, 228698418577414.16, 448062207825132.06, 878201927337253.0, 1721964563406372.5, 3377699720527878.5, 6627939074243378.0, 1.301039892351478e+16, 2.5547692431629004e+16, 5.0182967276414104e+16, 9.860512868348035e+16, 1.9381008051580614e+17, 3.810503277937884e+17, 7.493989779944506e+17, 1.4742274976940006e+18, 2.900899269655936e+18, 5.709706499005335e+18, 1.1240984669916758e+19, 2.2136092888451457e+19, 4.360139508331349e+19, 8.590125598503551e+19, 1.6927600444109937e+20, 3.336454580288336e+20, 6.57758188685415e+20, 1.29698797768955e+21, 2.557948511554391e+21, 5.045816241970307e+21, 9.955259071995465e+21, 1.9645044568737712e+22, 3.877311428040339e+22, 7.653913468339374e+22, 1.5111572745182865e+23, 2.9840574028462364e+23, 5.893513370621318e+23, 1.1641507892585313e+24, 2.299907656827831e+24, 4.5443958520453537e+24, 8.980591802851533e+24, 1.7749875563283023e+25, 3.508696332276877e+25, 6.936732978754284e+25, 1.3715812935264155e+26, 2.7123405355129126e+26, 5.364406836903313e+26, 1.0610914622446112e+27, 2.0991157187882527e+27, 4.1530891640541786e+27, 8.217814728873161e+27, 1.6262622831875316e+28, 3.2186441021419883e+28, 6.370924408363526e+28, 1.2611829951250238e+29, 2.4968875459040885e+29, 4.943837340890097e+29]
```

Out[18]: [<matplotlib.lines.Line2D at 0x1c58876b3a0>]



In [2]:
```
#Assignment 01-5.Path counting-5.1
import numpy as np
def creat_matrix(N,M):
    a1=np.ones((N,M))
    for i in range(N):
        for j in range(M):
            a1[i][j]=np.random.randint(0,2)
    #限定左上角、右小角元素为1
    a1[0][0]=1
```

```
    a1[int(N)-1][int(M)-1]=1
    return a1
creat_matrix(3,4)
```

Out[2]:
```
array([[1., 0., 1., 0.],
       [0., 1., 0., 0.],
       [1., 1., 1., 1.]])
```

In [3]:
```
#Assignment 01-5.Path counting-5.2
def Count_path(N,M):
    #利用creat_matrix函数得到一个矩阵
    a2=creat_matrix(N,M)
    #利用判断语句返回-1、0到原先为0、1的位置，即将原矩阵0替换为-1，1替换为0
    #题目转化为：0 is good to go，-1 as a blockage or dead-end
    a2[a2==0]=-1
    a2[a2==1]=0
    #print(a2)
    #将矩阵第一列可以通过的元素（即可以产生路径的元素，0）赋值为1
    for n in range(N):
        if a2[n][0]==0:
            a2[n][0]=1
    #将矩阵第一行可以通过的元素（即可以产生路径的元素，0）赋值为1
    for m in range(M):
        if a2[0][m]==0:
            a2[0][m]=1
    #当前元素可以继续前进的话，并且元素左边或（和）上面相邻元素大于0，就加上该元素左
    #赋值后当前元素值表示：从左上角前进到该元素位置可能的路径数
    for i in range(1,N):
        for j in range(1,M):
            if a2[i][j]==-1:
                continue
            if a2[i-1][j]>0:
                a2[i][j]+=a2[i-1][j]
            if a2[i][j-1]>0:
                a2[i][j]+=a2[i][j-1]
    #如果矩阵右下角元素值大于0，返回该元素值，否则返回0
    #返回值即代表从左上角到右下角可能的路径数
    if a2[N-1][M-1]>0:
        return a2[N-1][M-1]
    else:
        return 0
Count_path(10,8)
#解题思路来自https://www.geeksforgeeks.org/count-number-ways-reach-destination-maze/
#矩阵内元素转化思路来自赵望超同学
```

Out[3]:
```
0
```

In [5]:
```
#Assignment 01-5.Path counting-5.3
ave_path=[]
for i in range(1000):
    ave_path.append(Count_path(10,8))
print(ave_path)
print("Average number of paths:"+str(np.mean(ave_path)))
```

```
[2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 58.0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1.0, 36.0, 0, 0, 4.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2.0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24.0, 0, 27.0, 0, 0, 0, 2.0, 0, 9.0, 0, 0, 0,
2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 48.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 8.0, 0, 0, 0, 0, 0, 4.0, 0, 0, 0, 0, 0, 0, 0, 2.0, 12.0,
0, 0, 0, 9.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2.0, 0, 1.0, 0, 0, 7.0, 0, 0, 0, 0, 0, 0, 0, 18.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28.0, 0, 0, 0, 0, 0, 0, 12.0, 0, 0, 0, 0, 0,
4.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 36.0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 3.0, 0, 0, 0, 0, 0, 4.0, 0, 0, 0, 3.
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 3.0, 0, 0, 0, 0, 0, 0, 0, 0, 4.0, 0, 0, 1.0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1.0, 0, 0, 21.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 1.0, 1.0, 0, 0,
0, 0, 0, 0, 0, 12.0, 0, 0, 0, 0, 72.0, 0, 0, 0, 0, 0, 0, 0, 6.0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 3.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 9.0, 0, 0, 6.0, 0,
0, 0, 0, 0, 0, 0, 0, 24.0, 2.0, 0, 13.0, 0, 0, 0, 96.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 3.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4.0, 0, 0, 0, 0, 7.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 14.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0,
0, 0, 0, 0, 0, 0, 0, 14.0, 6.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 6.0, 0,
0, 0, 0, 0, 0, 0, 0, 1.0, 2.0, 0, 0, 0, 0, 0, 0, 0, 0, 8.0, 0, 0, 0, 0, 9.0, 45.0,
0, 0, 2.0, 12.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 7.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6.0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 4.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4.0, 0, 0, 1.0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13.0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 9.0,
0, 0, 0, 4.0, 2.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12.0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9.0, 0,
0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 9.0, 0, 1.0, 0, 0, 0, 0,
0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2.0, 0, 1.0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 24.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1.0, 0, 1.0, 0, 0, 11.0, 0, 0, 0, 3.0]
Average number of paths:0.972
```

In [ ]: