In [4]:

```
#流程图：将数字从大到小排序
def print_values():
    a=float(input("please enter your number a=: "))
    b=float(input("please enter your number b=: "))
    c=float(input("please enter your number c=: "))
#首先两两比较，取其中较小值与另一个数比较，如果大于，则可以得到按大小排序，如果不能则再进行一次比较
    if a>b:
        if b>c:
            print(a,b,c,a+b-10*c)
        else:
            if a>c:
                print(a,c,b,a+c-10*b)
            else:
                print(c,a,b,c+a-10*b)
    else:
        if b>c:
            if a>c:
                print(b,a,c,b+a-10*c)
            else:
                print(b,c,a,b+c-10*a)
        else:
             print(c,b,a,c+b-10*a)
print_values()
#运行结果
#please enter your number a=: 10
#please enter your number b=: 5
#please enter your number c=: 1
#10.0 5.0 1.0 5.0
```

```
please enter your number a=: 10
please enter your number b=: 5
please enter your number c=: 1
10.0 5.0 1.0 5.0
```

In [35]:

```
#编写一个方程
from  math import*
x=input("Please input a list of N positive integers:")
#将输入的值都编入x_list中
x_list=x.split(",")
x_list=[int(x_list[i])for i in range(len(x_list))]
print(x_list)
#list部分借鉴袁文婷同学
#定义F函数
def F(z):
    if z==1:
        return 1
    else:
        return F(ceil(z/3))+2*z
#打出所包含的数及其对应的函数值
for m in x_list:
    print(m,str(F(m)))
#运行结果
#Please input a list of N positive integers:12,23,45
#[12, 23, 45]
#12 37
#23 69
#45 135
```

```
Please input a list of N positive integers:12,23,45
[12, 23, 45]
12 37
23 69
45 135
```

In [6]:

```
#骰子
dice=[1,2,3,4,5,6]
def Find_number_of_ways():
    x=int(input("Please enter your number:"))
    if x<10 or x>60:
        print("0")
    else:
        i=0#i用来存储次数，每次有一种可以达到x值就加上1
        for a in range(6):
            for b in range(6):
                for c in range(6):
                    for d in range(6):
                        for e in range(6):
                            for f in range(6):
                                for g in range(6):
                                    for h in range(6):
                                        for j in range(6):
                                            for k in range(6):
                                                if dice[a]+dice[b]+dice[c]+dice[d]+dice[e]+dice[f]+
                                                    i+=1
        print(i)
#写一个方程，便于赋值，原理与上述一样，此方法缺点为运算量过大，需要运行很久
def Count_number(x):
    if x<10 or x>60:
        print("0")
    else:
        i=0
        for a in range(6):
            for b in range(6):
                for c in range(6):
                    for d in range(6):
                        for e in range(6):
                            for f in range(6):
                                for g in range(6):
                                    for h in range(6):
                                        for j in range(6):
                                            for k in range(6):
                                                if dice[a]+dice[b]+dice[c]+dice[d]+dice[e]+dice[f]+
                                                    i+=1
        return i
Find_number_of_ways()
Number_of_ways=[]

for x in range(10,61):
    l=Count_number(x)
    Number_of_ways.append(x)
    Number_of_ways.append(l)
#通过print，找到最大值
print("次数最多为",max(Number_of_ways))
y=Number_of_ways.index(max(Number_of_ways))
print("次数最多时x的值为",Number_of_ways[y-1])
#输出结果
#Please enter your number:12
#55
#次数最多为 4395456
#次数最多时x的值为 35
```

```
Please enter your number:12
55
次数最多为 4395456
次数最多时x的值为 35
```

In [1]:

```
#随机列表
import random
def Random_integer(a):
    if a<1:
        print("error")
    else:
        inte=[]
        for i in range(a):
            b=random.randint(0,10)
            inte.append(b)
    return(inte)
#求列表中所有子集平均值的和
#感谢焦小乔同学给我指出翻译错误且提供平均数的和为总值的和的排列组合倍数的思路
def jc(x):#上10.13周四课之前不知道可以使用math中的factorial函数所以自己定义了一个阶乘函数
    c=1
    for i in range(x):
        c=c*(i+1)
    return c
def Sum_averages(b):
    #当有n个数时，取i个数平均数的子集的总和应该为，C(n-1,i-1)*列的总和/i
    Number=[]
    Number.extend(Random_integer(b))#定义一个空list将random_integer函数放入其中
    sum_0=0 #list中求得所有数的和
    total=jc(b-1) #a-1的阶乘
    sum_1=[]#不同取值时的总和放入此list
    sum_2=0#方便后续求和
    for i in range(b):
        sum_0=Number[i]+sum_0
    for i in range(b):
        if i==0:
            sum_1.append(sum_0)
            continue
        elif i==b-1:
            sum_1.append(sum_0/b)
            break
        else:
            d=jc(i)#i的阶乘
            e=jc(b-1-i)#b-1-i的阶乘
            sum_1.append(sum_0*total/((i+1)*d*e))
    for i in range(b):
        sum_2+=sum_1[i]
    return sum_2
Sum_averages(100)
#5.590339147006491e+30
```

Out[1]:

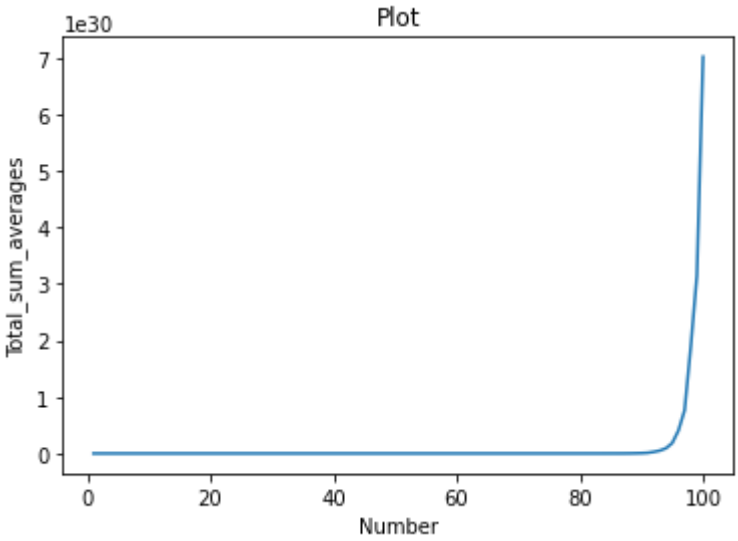
```
5.590339147006491e+30
```

In [17]:

```
#作图
Total_sum_averages=[]#做y轴
x_axis=range(1,101)#做x轴
for i in range(1,101):
    Total_sum_averages.append(Sum_averages(i))
import matplotlib.pyplot as plt
plt.plot(x_axis,Total_sum_averages)
plt.xlabel("Number")#x轴名字
plt.ylabel("Total_sum_averages")#y轴名字
plt.title("Plot")#表头
plt.show
#图为一个指数函数前面几乎是平的后续接近100时斜率非常大接近直线上升
```

Out[17]:

<function matplotlib.pyplot.show(close=None, block=None)>



In [2]:

```
#创建一个矩阵
import numpy as np
import random
def Creat_array(N,M):
    arr0=np.random.randint(0,2,N*M)
    arr2=arr0.reshape(N,M)
    arr2[[0,N-1],[0,M-1]]=1
    return(arr2)
#矩阵迷宫
#参考https://www.geeksforgeeks.org/count-number-ways-reach-destination-maze/
def Count_path(N,M):
    arr1=Creat_array(N,M)
#倒置0和1，方便计算总数，如果产生一条路径，就能将这条路径的1值带到arr1[N-1,M-1]上视为一条路径
    arr1=np.where(arr1==0,-1,arr1)
    arr1=np.where(arr1==1,0,arr1)
    for i in range (N):
        if arr1[i,0]==0:
            arr1[i,0]=1#将与arr1[0，0]相连的可行的ceil全部变为1视为起始点
        else:
            break
    for i in range(1,M):
        if arr1[0,i]==0:
            arr1[0,i]=1#与上述同理
        else:
            break
    for i in range(1,N):
        for j in range(1,M):
            if arr1[i,j]==-1:#不能移动的ceil就找下一块找到可以移动的ceil查看其左边以及上边的ceil是否
                continue
            elif arr1[i,j-1]>0:
                arr1[i,j]=arr1[i,j-1]+arr1[i,j]#当可以移动时寻找有上边ceil可以达到这个ceil
            elif arr1[i-1,j]>0:
                arr1[i,j]=arr1[i-1,j]+arr1[i,j]#当可以移动时寻找有坐边ceil可以达到这个ceil
                #即每次可以存储有几条路径达到这个ceil，当下次有两条路径时可以将这个值乘以2，达到一个
    return arr1[N-1,M-1]
#求路径总数的平均值
Average_path=[]#存1000个迷宫路径的list
sum_0=0
for i in range(1000):
    Average_path.append(Count_path(10,8))
print(Average_path)
for i in range(1000):
    sum_0=Average_path[i]+sum_0
Average=sum_0/1000
print(Average)
#运算结果
#[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,
#0.026
```

```
[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, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 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, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 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, 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, 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, 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, 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, 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, 1,
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, 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, 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, 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, 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, 1, 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, 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, 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, 1, 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, 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, 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, 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, 1,
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, 0,
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,
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, 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, 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, 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, 1,
0]
0.026
```

In [ ]: