Ex.1 若将y ← uniform(0, 1) 改为 y ← x, 则上 述的算法估计的值是什么?

当y=x时,随机点分散在线段 $y=x, (-1 \le x \le 1)$ 上,当满足 $x^2+y^2 \le 1$ 时,显然当且仅当 $x \in [-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}]$, 此时若随机选取满足条件的点,那么该算法估算的结果趋近于 $\frac{2}{\sqrt{2}}$

```
import random
import math
tot =1000000
cnt = 0;
for i in range (0,tot):
    x = random.random()
    y = x
    if(x*x+y*y<1) : cnt += 1;
print("估计值:"+str(1.0*cnt/tot))
print("1/sqrt(2):"+str(2/2.0/math.sqrt(2)))
```

```
估计值:0.707156
1/sqrt(2):0.7071067811865475
```

Ex2. 在机器上用 $4\int_0^1 \sqrt{1-x^2}\, dx$ 估计π值，给出不 同的n值及精度。

```
tot = 100.0
for i in range(0,5):
    tot *= 10
    cnt = 0;
    for j in range(0,int(tot)):
        x = random.random()
        y = random.random()
        if(1-x*x>=y*y ) : cnt+=1;
    print("重复"+str(tot)+"次时估计的π为:"+str(4*cnt/tot)+"   相对误差
为:"+str(abs(4*cnt/tot-math.pi)/math.pi))
```

```
重复1000.0次时估计的π为:3.22   相对误差为:0.024957833511806065
重复10000.0次时估计的π为:3.1248   相对误差为:0.005345267652890864
重复100000.0次时估计的π为:3.14208   相对误差为:0.00015512718036502713
重复1000000.0次时估计的π为:3.141604   相对误差为:3.61167454156795e-06
重复10000000.0次时估计的π为:3.1407948   相对误差为:0.00025396468535834096
```

Ex3. 设a, b, c和d是实数，且a ≤ b, c ≤ d, f:[a, b] → [c, d]是一个连续函数，写一概率算法计算积分：

我们不妨设函数为$y = sin(x), a = 0, b = 1, c = 0, d = 1$ 因此其表达式可以表示为:

$$\int_0^1 sin(x)dx$$

```
def func(x,y):
    if(math.sin(x)>=y) : return 1
    return 0

tot = 100000
cnt = 0
for i in range(0,tot):
    x = random.random()
    y = random.random()
    cnt += func(x,y)
print("估算的积分值为:"+str(cnt/tot))
```

```
估算的积分值为:0.45925
```

EX. 用上述算法(setCount)，估计整数子集1~n的大小，并分析n对估计值的影响

```
import numpy as np
def SetCount(X):
    k = 0
    S = {}
    a = np.random.choice(X,size = 1)[0]
    while True :
        k += 1;
        S[a] = 1;
        a = np.random.choice(X,size = 1)[0]
        if(a in S.keys()) : break
    #print(k)
    return k
n = 10
tot=1000
for i in range (0,6):
    n *= 2
    X = []
    for j in range(0,n) : X.append(j)
#     X = random.shuffle(X)
    ans = 0;
    for k in range(0,1):
        cnt = 0;
        for j in range(0,tot):
            cnt += SetCount(X)
        print(cnt/tot)
        cnt /= tot
        cnt = 2*cnt*cnt/math.pi
        ans += cnt/1

    print("当n="+str(n)+"时估计的n为:"+str(ans)+"  相对误差为:"+str(abs(ans-
n)/n))
```

```
5.291
当n=20时估计的n为:17.821967445723057  相对误差为:0.10890162771384712
7.542
当n=40时估计的n为:36.212055649546485  相对误差为:0.09469860876133787
10.885
当n=80时估计的n为:75.42876372887692  相对误差为:0.057140453389038545
15.526
当n=160时估计的n为:153.4614462028058  相对误差为:0.040865961232463735
22.166
当n=320时估计的n为:312.7913833377296  相对误差为:0.022526927069595006
31.599
当n=640时估计的n为:635.6628061623782  相对误差为:0.006776865371284124
```

上面代码实现了SetCount的功能,对于相同的n,进行10组实验,每组实验重复计数1000次,k取1000次的平均值,最后结果为这10组实验平均值,我们不难看出随着n的增大器相对误差也逐渐减小

Ex. 分析dlogRH的工作原理，指出该算法相应的u和v

使用一个随机生成的随机数与a进行运算,保证生成的数据的均匀性,计算这个新的结果通常情况下能保证其平均性能

之后对生成的数据进行拟操作使得还原原先a的结果.

对于dlogRH其对应的u,v如下:

$$u: a \rightarrow [(g^r, mod, p) * a], mod, p, (r \in [0, p-2]) \qquad v: (r+x), mod(p-1) \rightarrow x$$

Ex. 写一Sherwood算法C，与算法A, B, D比较， 给出实验结果。

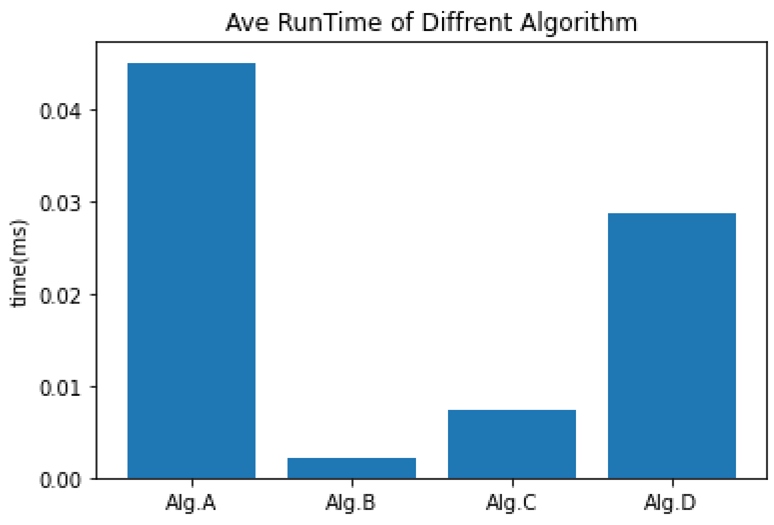
```
## 初始数据生成
import math
import random as rd
def RandData(n,val,ptr):
    pre = []
    val = rd.sample(range(0,n*10),n) # 生成无重复的序列
    val.append(-1)
    val.sort()                      # 排序
    for i in range (0,n):          # 生成指针数组ptr为后继元素 pre为前驱元素
        ptr.append(i+1)
        pre.append(i-1)
    ptr.append(0)
    pre.append(n-1)

    for i in range (0,n) :          # 随机打乱
        while(True) :             # 随机挑选a,b位置元素进行交换
            a = rd.randint(1,n)
            b = rd.randint(1,n)
            if val[a] > val[b] : a,b = b,a
            if(val[ptr[a]]<val[pre[b]]): break  #要求交换元素间距大于3(否则会出
现指针错误)

        pre[ptr[a]],pre[ptr[b]]= pre[ptr[b]],pre[ptr[a]] # 前驱/后继节点指针交
换
        ptr[pre[a]],ptr[pre[b]]=ptr[pre[b]],ptr[pre[a]]

        pre[a],pre[b]=pre[b],pre[a]                      # 节点前驱后继指针交
换
        ptr[a],ptr[b]=ptr[b],ptr[a]

        val[a],val[b]=val[b],val[a]                      # 值交换
    return (val,ptr)
```

```
## 算法实现
def search(x,i,val,ptr):  # 基础算法
    cnt = 0;
    while(x > val[i] and i!=0):
        i=ptr[i]
        cnt += 1
    return (i,cnt);

def A(x,n,val,ptr): #朴素算法A,时间复杂度O(n)
    return search(x,ptr[0],val,ptr)

def B(x,n,val,ptr): #算法B,时间复杂度O(sqrt(n))
    i = ptr[0]
    maxi = val[i]
    sqrt = int(math.sqrt(n))
    for j in range(0,sqrt):
        y = val[j]
        if maxi < y and y<= x:
            i = j
            maxi = y
    return search(x,i,val,ptr)

def C(x,n,val,ptr): #Sherwood算法C,时间复杂度O(sqrt(n))
    i = ptr[0]
    maxi = val[i]
    sqrt = int(math.sqrt(n))
    sample = rd.sample(range(1,n),sqrt)
    for j in sample:
        y = val[j]
```

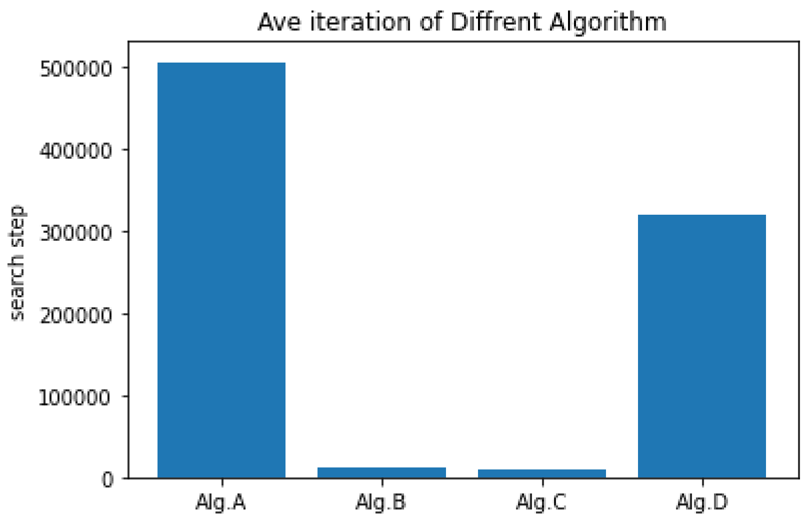
```
            if maxi < y and y<= x:
                i = j
                maxi = y
        return search(x,i,val,ptr)

    def D(x,n,val,ptr): # 朴素算法A优化,时间复杂度O(n/2)
        i = rd.randint(1,n)
        y = val[i]
        if x < y : return search(x,ptr[0],val,ptr)
        elif x > y : return search(x,ptr[i],val,ptr)
        return (i,0)
```

```
## 实验对比
import time
import matplotlib.pyplot as plt

it = [0,0,0,0]  #迭代时间
ave = [0,0,0,0] #搜索次数
init_time = 0
for i in range (0,100): # 测试程序
    val = []
    ptr = []
    n = 10000
    begin = time.perf_counter()
    (val,ptr)=RandData(n,val,ptr)
    end = time.perf_counter()
    init_time += end-begin
    tar = val[rd.randint(1,n)]

    begin = time.perf_counter()
    ave[0] += A(tar,n,val,ptr)[1]
    end = time.perf_counter()
    it[0] += end-begin

    begin = time.perf_counter()
    ave[1] += B(tar,n,val,ptr)[1]
    end = time.perf_counter()
    it[1] += end-begin

    begin = time.perf_counter()
    ave[2] += C(tar,n,val,ptr)[1]
    end = time.perf_counter()
    it[2] += end-begin

    begin = time.perf_counter()
    ave[3] += D(tar,n,val,ptr)[1]
    end = time.perf_counter()
    it[3] += end-begin
```

```
labels = ["Alg.A","Alg.B","Alg.C","Alg.D"]
plt.bar(range(4),it,tick_label=labels)
plt.ylabel('time(ms)')
plt.title("Ave RunTime of Diffrent Algorithm")
plt.show()
print("Ave RunTime of Diffrent Algorithm:"+str(it))

plt.bar(range(4),ave,tick_label=labels)
plt.ylabel('search step')
plt.title("Ave iteration of Diffrent Algorithm")
plt.show()
print("Ave iteration of Diffrent Algorithm:"+str(ave))
```



```
Ave RunTime of Diffrent Algorithm:[0.045119399999999637,
0.002061100000005034, 0.007289000000000101, 0.028673099999994456]
```



```
Ave iteration of Diffrent Algorithm:[505127, 12299, 10478, 319520]
```

Ex.证明：当放置（k+1)th皇后时，若有多个位置是开放的,则算法 QueensLV选中其中任一位置的概率相等

原命题:对于放置（k+1)th皇后第i轮时($1 \le i \le k+1$),算法选择[1,i]中间任意位置概率相等,为$\frac{1}{i}$

归纳基础:对于i=1时显然选择[1,1]概率相等,均为1

归纳假设:假设当i=n时原命题成立

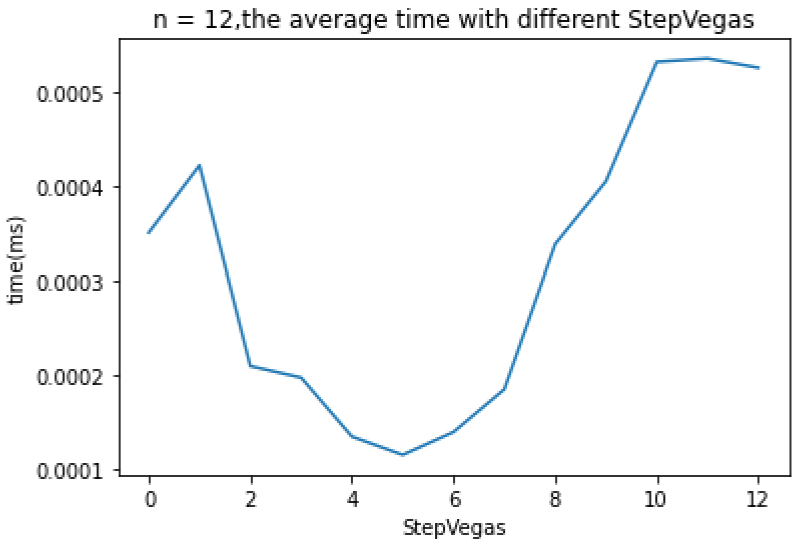
归纳递推:当i=n+1时,首先显然对于第n+1个元素其被选中的概率为$\frac{1}{n+1}$,对于其他元素被选中的概率之和为$\frac{n}{n+1}$,由于当i=n时各个元素被选取的可能性均相等为$\frac{1}{n}$因此对于单个元素,在第n+1轮时其被选中的概率为: $\frac{n}{n+1} * \frac{1}{n} = \frac{1}{n+1}$,因此对于i=n+1时也成立,因此对于所有i<=k+1均有上述结论

Ex. 写一算法，求n=12~20时最优的StepVegas值。

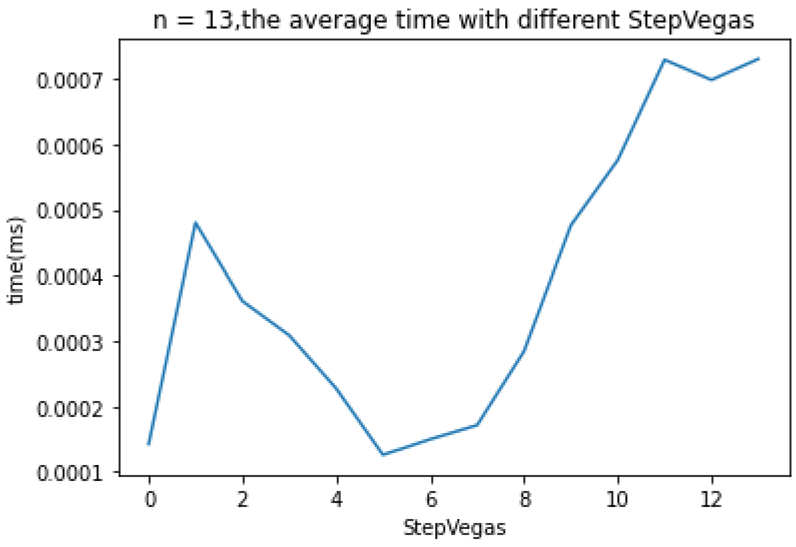
```python
import random as rd
import matplotlib.pyplot as plt
import time
def dfs(depth,col,dia45,dia135,n,ans):
    if(depth == 0) :
        return True,col,dia45,dia135,ans

    for j in range(1,n+1):
        if(j in col or depth+j in dia45 or depth-j in dia135): continue
        col.add(j)
        dia45.add(depth+j)
        dia135.add(depth-j)
        ans.append((depth,j))

        a = dfs(depth-1,col,dia45,dia135,n,ans)
        if a[0] == True : return a
        ans.pop(-1)
        col.remove(j)
        dia45.remove(depth+j)
        dia135.remove(depth-j)
    return False,col,dia45,dia135,ans

def QueensLv(col,dia45,dia135,n,StepVegas,ans):
    col = set()
    dia45 = set()
    dia135 = set()
    ans = []
    for i in range(n,n-StepVegas,-1):
        nb = []
        for j in range(1,n+1):
            if(j in col or (i+j) in dia45 or (i-j) in dia135) : continue;
            nb.append(j)
        if(len(nb)==0) : return False,col,dia45,dia135,ans;
        j = rd.sample(nb,1)[0]
        col.add(j)
        dia45.add(i+j)
        dia135.add(i-j)
```

```
            ans.append((i,j))

    return (True,col,dia45,dia135,ans)

cntF = cntT = 0
record = []
for n in range(12,20):
    record.append([])
    for setp in range (0,n+1):
        round = 100
        begin = time.perf_counter()
        for i in range(0,round):
            col = set()
            dia45 = set()
            dia135 = set()
            ans = []
            while True:
                while True :
                    a = QueensLv(col,dia45,dia135,n,setp,[])
                    flag,col,dia45,dia135,ans = a
                    if flag == True : break;
                flag,col,dia45,dia135,ans=dfs(n-
setp,col,dia45,dia135,n,ans)
                if flag == False : continue
                break
        end = time.perf_counter()
        record[-1].append((end-begin)/round)
```

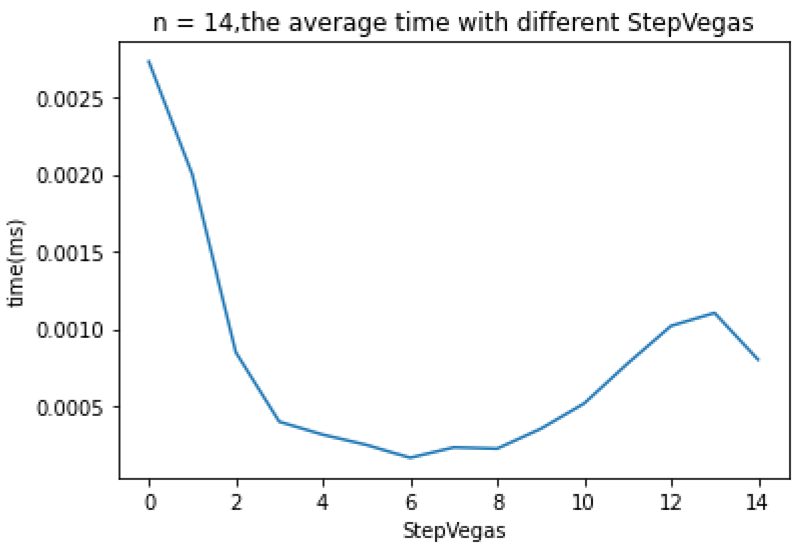
```
import matplotlib.pyplot as plt
for i in record:
    xx = [i for i in range(len(i))]
    plt.title("n = "+str(len(i)-1)+",the average time with different
StepVegas")
    plt.xlabel('StepVegas')
    plt.ylabel('time(ms)')
    plt.plot(xx,i)
    plt.show()
    mini = 0
    for j in range(len(i)):
        if(i[j]<i[mini]): mini = j
    print("When n="+str(len(i))+",the Best SetpVegas:"+str(mini))
```



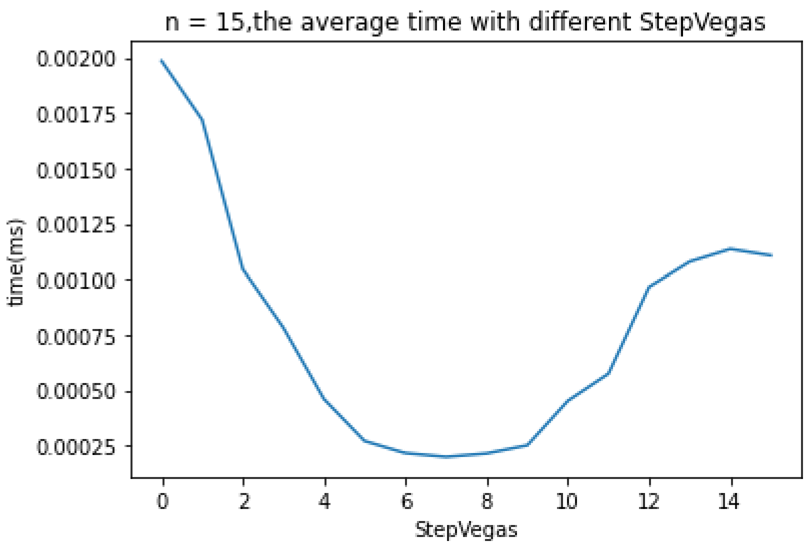
```
When n=13,the Best SetpVegas:5
```



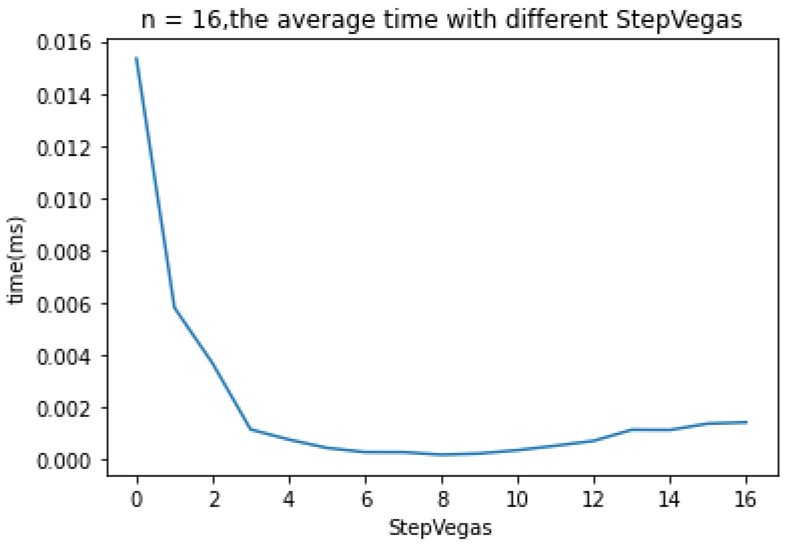
```
When n=14,the Best SetpVegas:5
```



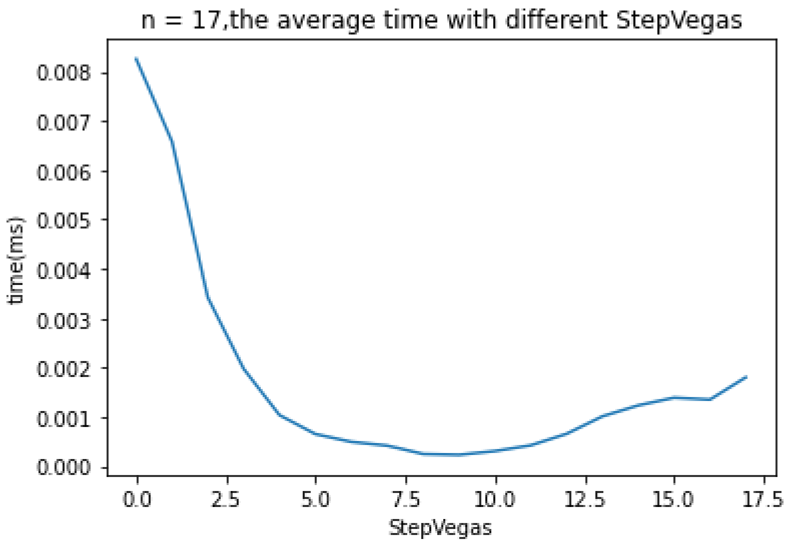
```
When n=15,the Best SetpVegas:6
```



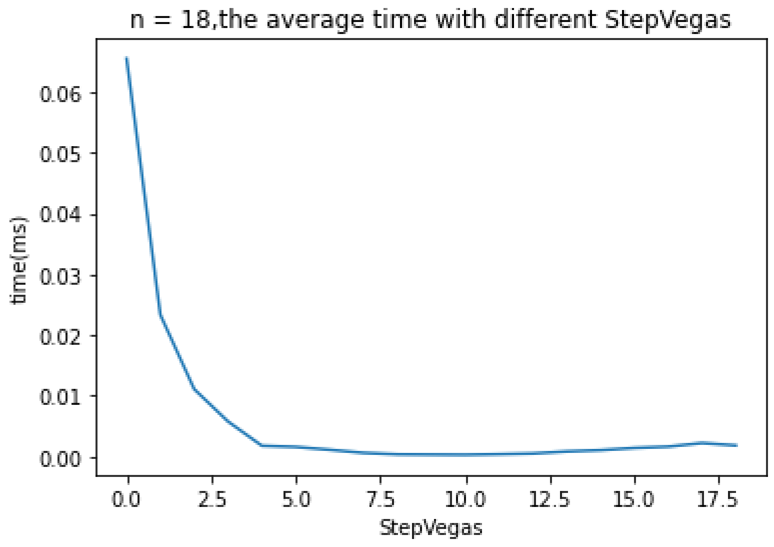
```
When n=16,the Best SetpVegas:7
```



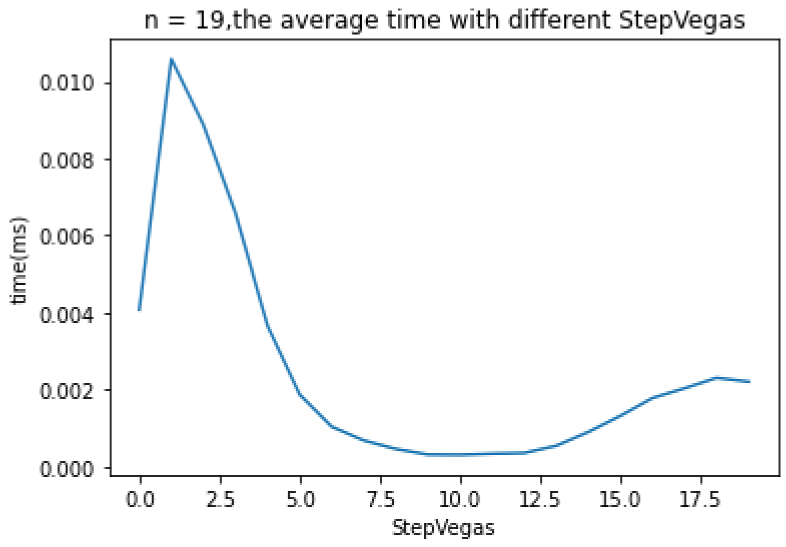
```
When n=17,the Best SetpVegas:8
```



```
When n=18,the Best SetpVegas:9
```



```
When n=19,the Best SetpVegas:10
```



```
When n=20,the Best SetpVegas:10
```

习题: 使用Miller-Rabin并与确定性算法相比较，并给出100~10000以内错误的比例。

```
def Btest(a,n):
    s = 0
    t = n-1
    while True:
        if t %2 == 1 : break
        s += 1
        t /= 2
    x = 1
    for i in range(int(t)):
        x = x*a%n
    if x == 1 or x == n-1 : return True
    for i in range(1,s):
        x = x*x%n
        if x == n-1 : return True
    return False

def MillRab(n):
    a = random.randint(2,n-2)
    return Btest(a,n)

def RepeatMillRab(n,k):
    for i in range(k):
        if MillRab(n) == False: return False
    return True

def printPrimes():
    ans = [2,3]
    for n in range(5,10000,2):
        k = math.floor(math.log(n))
        if RepeatMillRab(n,k) : ans.append(n)
    print(ans)
    return ans

def shai():
    ans = [2,3]
    for n in range(5,10000,2):
        sqrt = math.ceil(math.sqrt(n))
        flag = 0
        for i in ans:
            if i>sqrt : break;
            if n%i == 0 :
                flag = 1
                break
        if flag == 0 : ans.append(n)
    return ans


print("由Miller-Rabin算法求出的指数为:")
ans1 = len(printPrimes());
ans2 = len(shai());
print("Miller-Rabin算法求出的10000内质数共:%d个,实际存在:%d个,错误率为:%f"%
(ans1,ans2,(ans1-ans2)/10000))
```

```
由Miller-Rabin算法求出的指数为:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229,
233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409,
419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499,
503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601,
607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809,
811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907,
911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009,
1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087,
1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171,
1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259,
1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327,
1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447,
1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523,
1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607,
1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697,
1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787,
1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879,
1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993,
1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081,
2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153,
2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269,
2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351,
2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437,
2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549,
2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659,
2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719,
2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803,
2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909,
2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019,
3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121,
3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229,
3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329,
3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433,
3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529,
3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613,
3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701,
3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803,
3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911,
3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007,
4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099,
4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217,
4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289,
4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421,
4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517,
4519, 4523, 4547, 4549, 4561, 4567, 4583, 4591, 4597, 4603, 4621, 4637,
4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723,
4729, 4733, 4751, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817,
4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943,
4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021,
5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107, 5113, 5119,
5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233, 5237,
5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381,
5387, 5393, 5399, 5407, 5413, 5417, 5419, 5431, 5437, 5441, 5443, 5449,
5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531, 5557,
5563, 5569, 5573, 5581, 5591, 5623, 5639, 5641, 5647, 5651, 5653, 5657,
5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717, 5737, 5741, 5743, 5749,
5779, 5783, 5791, 5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851,
5857, 5861, 5867, 5869, 5879, 5881, 5897, 5903, 5923, 5927, 5939, 5953,
5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073, 6079,
6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173, 6197,
6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269, 6271, 6277,
6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361,
6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451, 6469, 6473, 6481,
6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581, 6599,
6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703,
6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793, 6803, 6823,
6827, 6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911,
6917, 6947, 6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001,
7013, 7019, 7027, 7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121, 7127,
7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219, 7229, 7237,
7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333, 7349, 7351,
7369, 7393, 7411, 7417, 7433, 7451, 7457, 7459, 7477, 7481, 7487, 7489,
7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561, 7573,
7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639, 7643, 7649, 7669, 7673,
7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741, 7753, 7757, 7759,
7789, 7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877, 7879, 7883,
7901, 7907, 7919, 7927, 7933, 7937, 7949, 7951, 7963, 7993, 8009, 8011,
8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089, 8093, 8101, 8111, 8117,
8123, 8147, 8161, 8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231, 8233,
8237, 8243, 8263, 8269, 8273, 8287, 8291, 8293, 8297, 8311, 8317, 8329,
8353, 8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443, 8447,
8461, 8467, 8501, 8513, 8521, 8527, 8537, 8539, 8543, 8563, 8573, 8581,
8597, 8599, 8609, 8623, 8627, 8629, 8641, 8647, 8663, 8669, 8677, 8681,
8689, 8693, 8699, 8707, 8713, 8719, 8731, 8737, 8741, 8747, 8753, 8761,
8779, 8783, 8803, 8807, 8819, 8821, 8831, 8837, 8839, 8849, 8861, 8863,
8867, 8887, 8893, 8923, 8929, 8933, 8941, 8951, 8963, 8969, 8971, 8999,
9001, 9007, 9011, 9013, 9029, 9041, 9043, 9049, 9059, 9067, 9091, 9103,
9109, 9127, 9133, 9137, 9151, 9157, 9161, 9173, 9181, 9187, 9199, 9203,
9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283, 9293, 9311, 9319,
9323, 9337, 9341, 9343, 9349, 9371, 9377, 9391, 9397, 9403, 9413, 9419,
9421, 9431, 9433, 9437, 9439, 9461, 9463, 9467, 9473, 9479, 9491, 9497,
9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629,
9631, 9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733, 9739,
9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817, 9829, 9833,
9839, 9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931,
9941, 9949, 9967, 9973]
Miller-Rabin算法求出的10000内质数共:1229个,实际存在:1229个,错误率为:0.000000
```

```
```