Algorithm A:

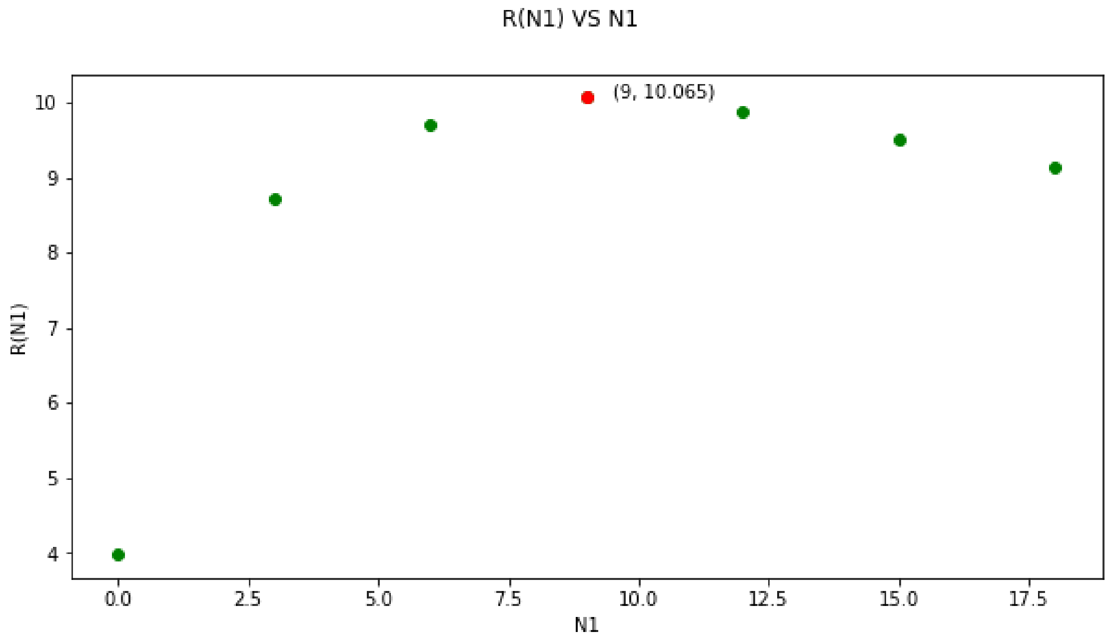
In this we have taken N1 as multiple of 3, coz if we dont do that then some of the coin will get tossed more than other coins therefore it is not fair if N1 is not a multiple of 3.

```
from tkinter import Y
import matplotlib.pyplot as plt;
import numpy as np;
import random;
plt.rcParams['figure.figsize']= [10,5]
import csv

N =5000 #int(input("enter N:"))
Pa =0.2 #float(input("enter Pa:"))
Pb = 0.4 #float(input("enter Pb:"))
Pc = 0.7 #float(input("enter Pc:"))

Rn1F = [0 for y in range(1000)]
Rn1 = [0 for z in range(N)]


remainder = N%3
j=N-remainder
g = int(j/3)

ntcc = [0 for c in range(g+1)]

for q in range(0,g+1):
  N1=q*3

  for r in range(0,1000):
    ma=0
    for i in range(0,q):
      xa = random.uniform(0,1)
      if xa<Pa:
        ma=ma+1          #number of heads for coin a in n1/3 tosses

    mb=0
    for i in range(0,q):
      xb = random.uniform(0,1)
      if xb<Pb:
        mb=mb+1         #number of heads for coin a in n1/3 tosses

    mc=0
    for i in range(0,q):
      xc = random.uniform(0,1)
      if xc<Pc:
        mc=mc+1          #number of heads for coin a in n1/3 tosses
```

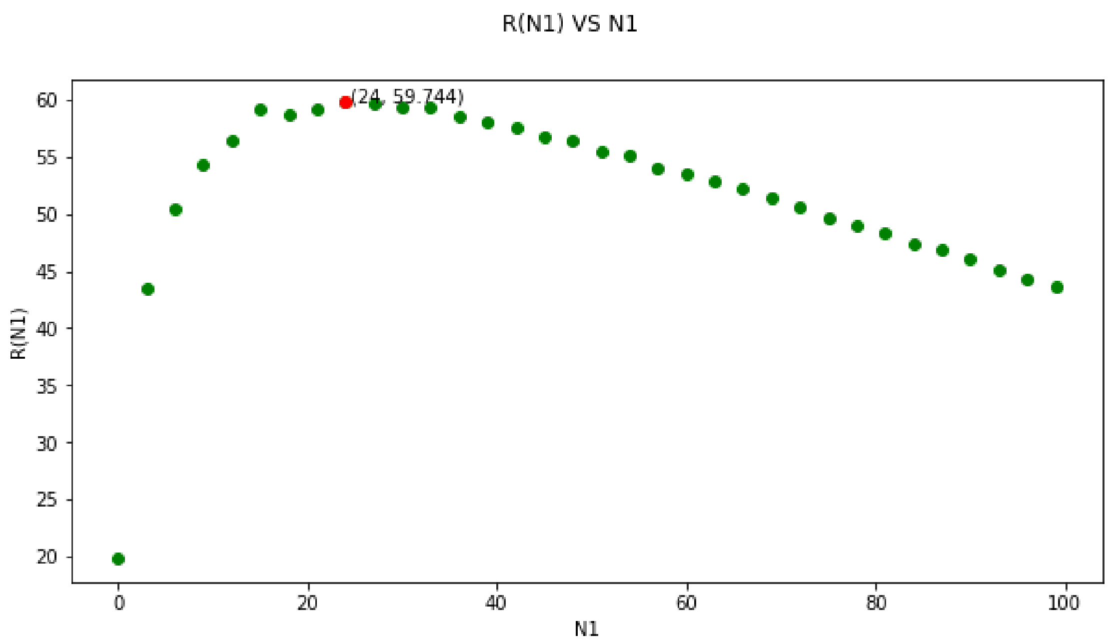
```
    m = [ma,mb,mc]
    m_max = max(m)
    m_sup = 0

    if m_max == mc and m_max > ma and m_max > ma:    #ntcc bro
      ntcc[q]=ntcc[q]+1

    for i in range(N1,N):
      if m_max == ma:
        x = random.uniform(0,1)
        if x<Pa:
          m_sup = m_sup+1

      elif m_max == mb:
        x = random.uniform(0,1)
        if x<Pb:
          m_sup = m_sup+1

      elif m_max == mc:
        x = random.uniform(0,1)
        if x<Pc:
           m_sup = m_sup+1

    Rn1F[r] = m_sup+ma+mb+mc
  for h in range(0,1000):
     Rn1[N1] = Rn1F[h] + Rn1[N1]
  Rn1[N1] = Rn1[N1]/1000


  plt.scatter(N1,Rn1[N1],color='green')
  plt.xlabel('N1')
  plt.ylabel('R(N1)')
  plt.suptitle('R(N1) VS N1')
R = max(Rn1)
print(R)
N1max = 0
for i in range(1,g+1):
  if (Rn1[i*3]>Rn1[(i-1)*3]):
    N1max = i
plt.scatter((Rn1.index(R)),R,color='red')
plt.text((Rn1.index(R))+1,R+0.5,((Rn1.index(R)),R))
```

FOR Pa=0.2, Pb=0.4, Pc=0.7

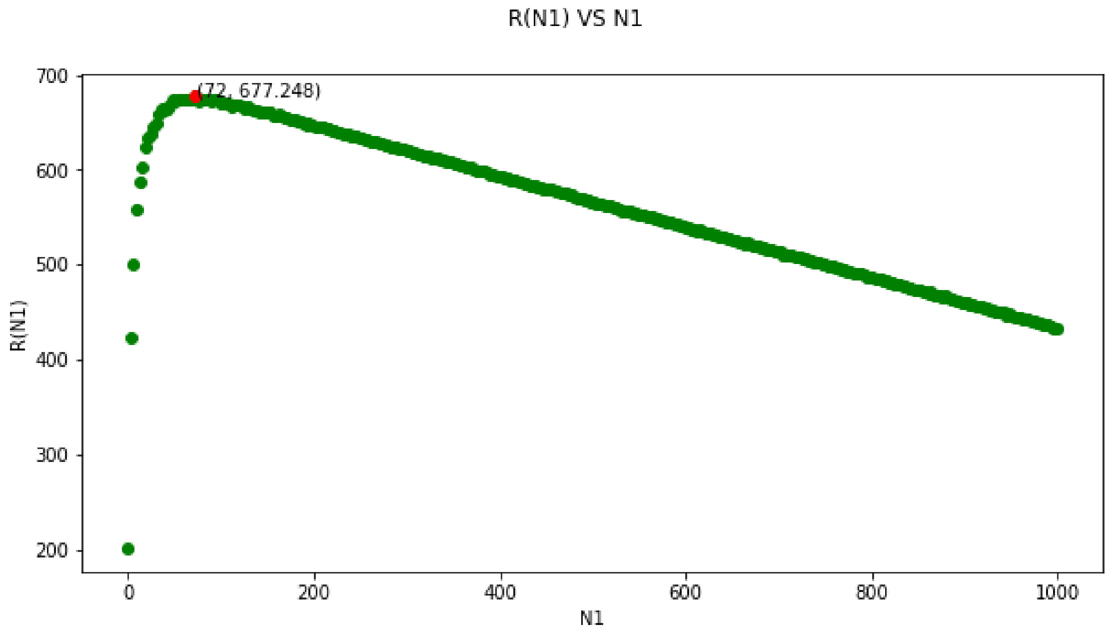
N1 = 20



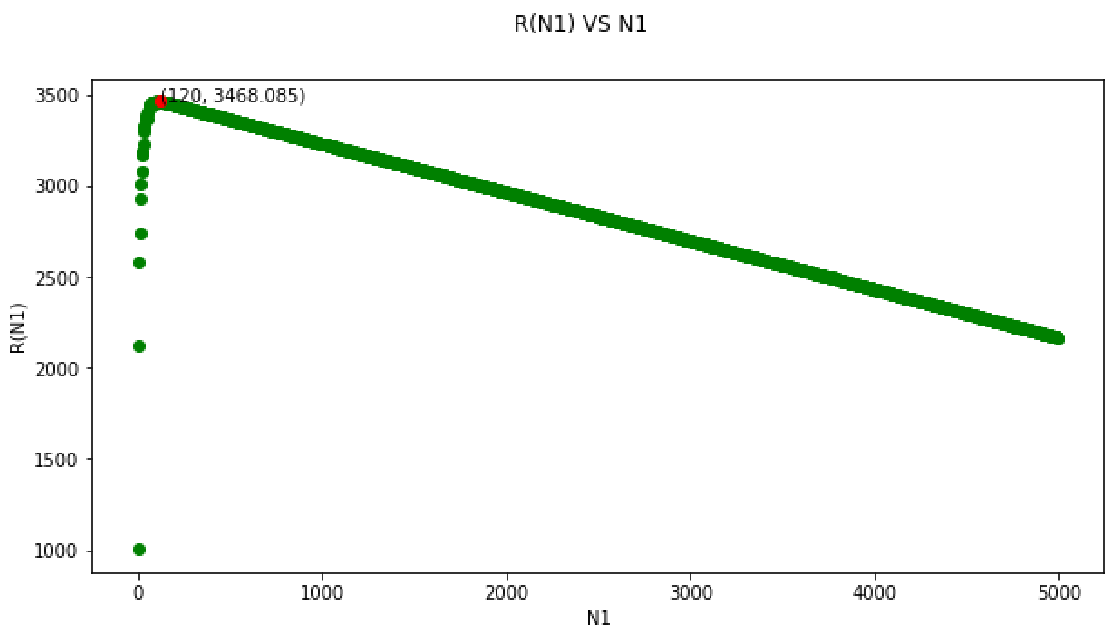
N1 = 100



N1 = 1000

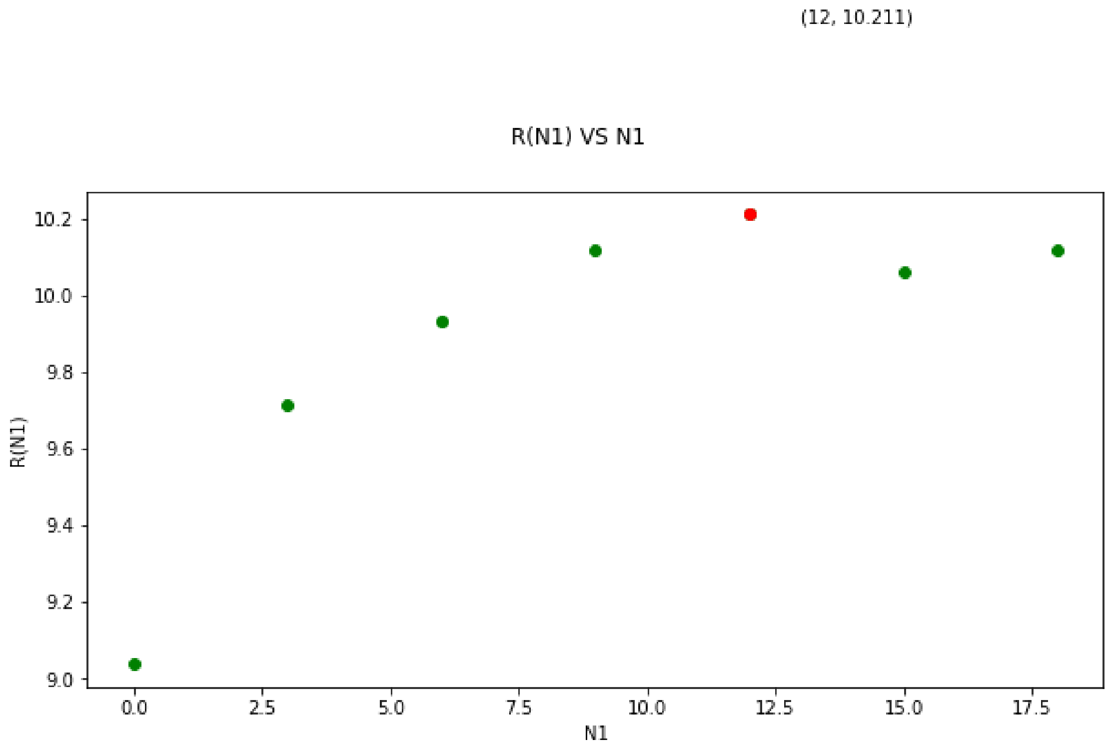


N1 = 5000

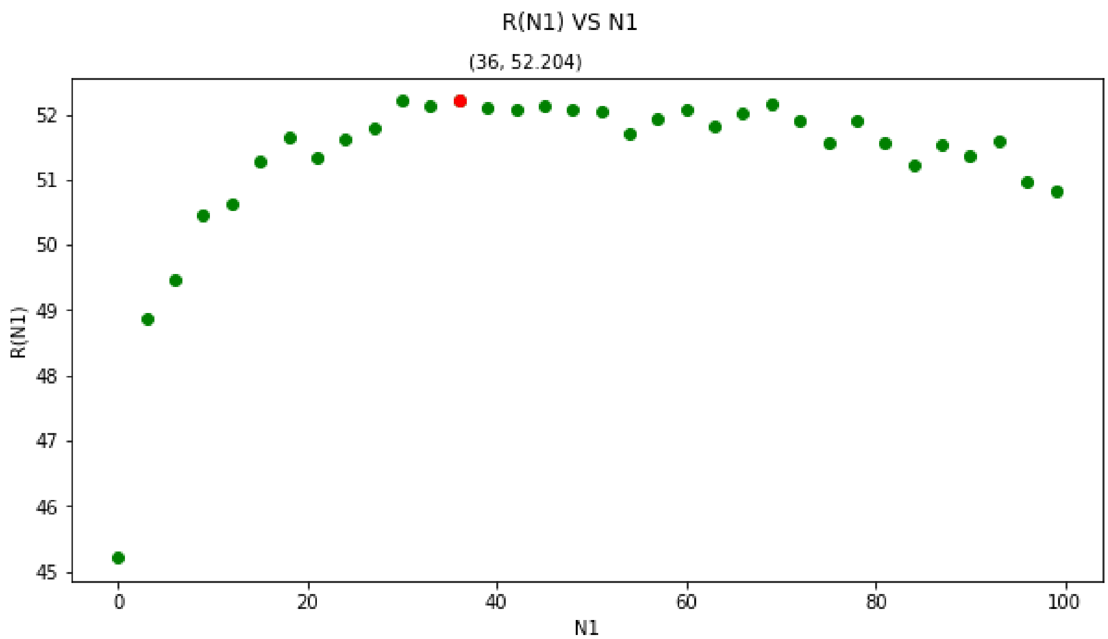


FOR Pa=0.45, Pb=0.5, Pc=0.58

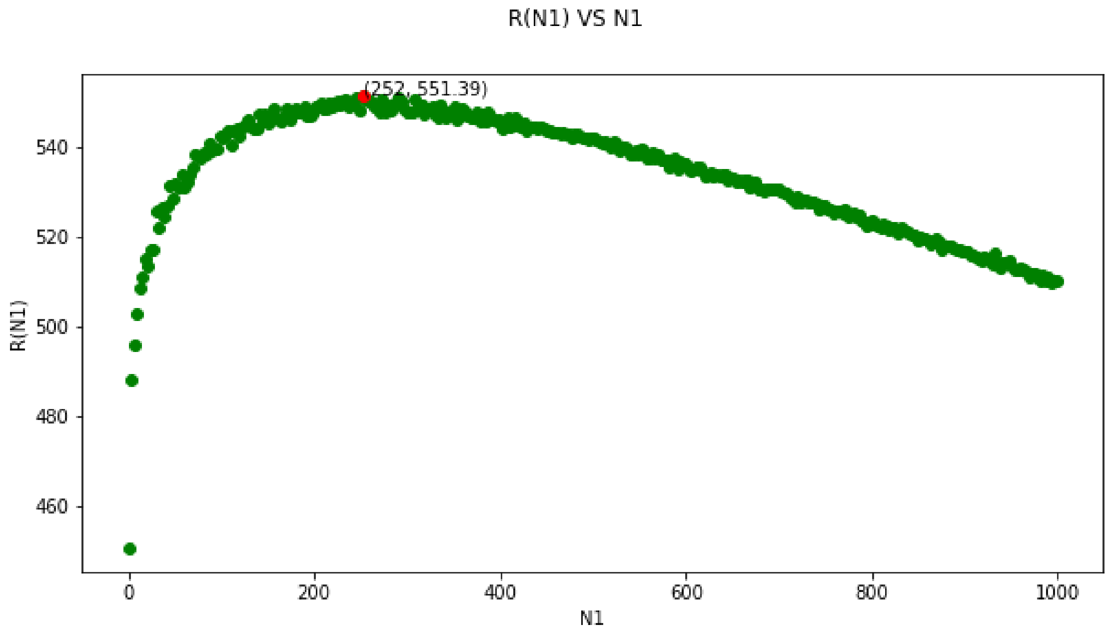
N1 = 20



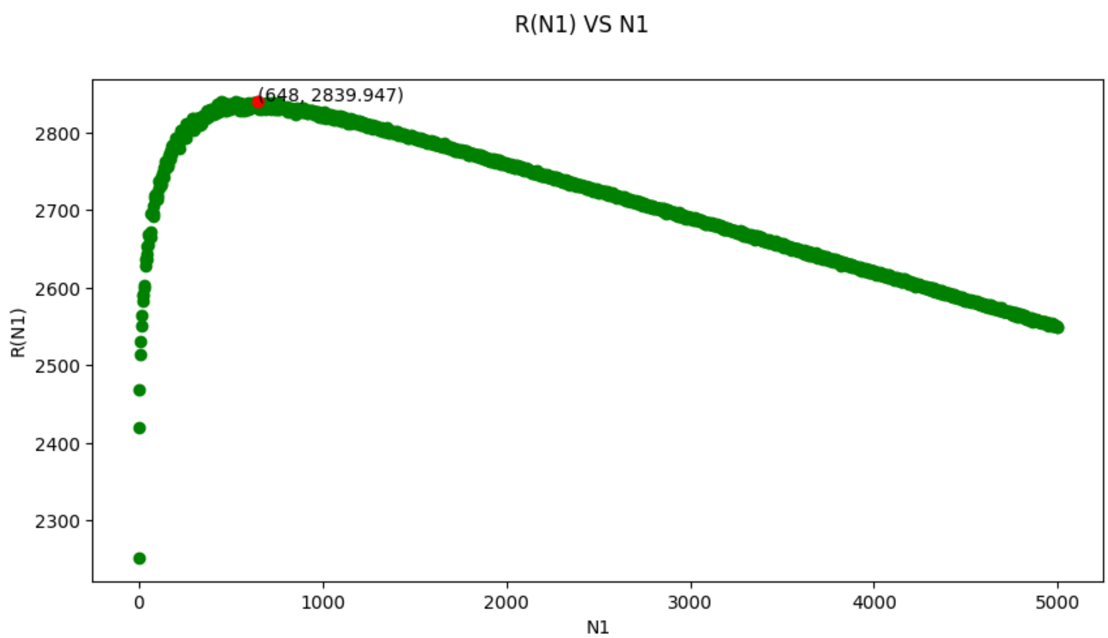
N1 = 100



N1 = 1000



N1= 5000




```
def nCr(n,r):
  mul1=1
  mul2=1
  for i in range(n-r+1,n+1):
      mul1=mul1*i
  for i in range(1,r+1):
      mul2=mul2*i
  return mul1/mul2;
```

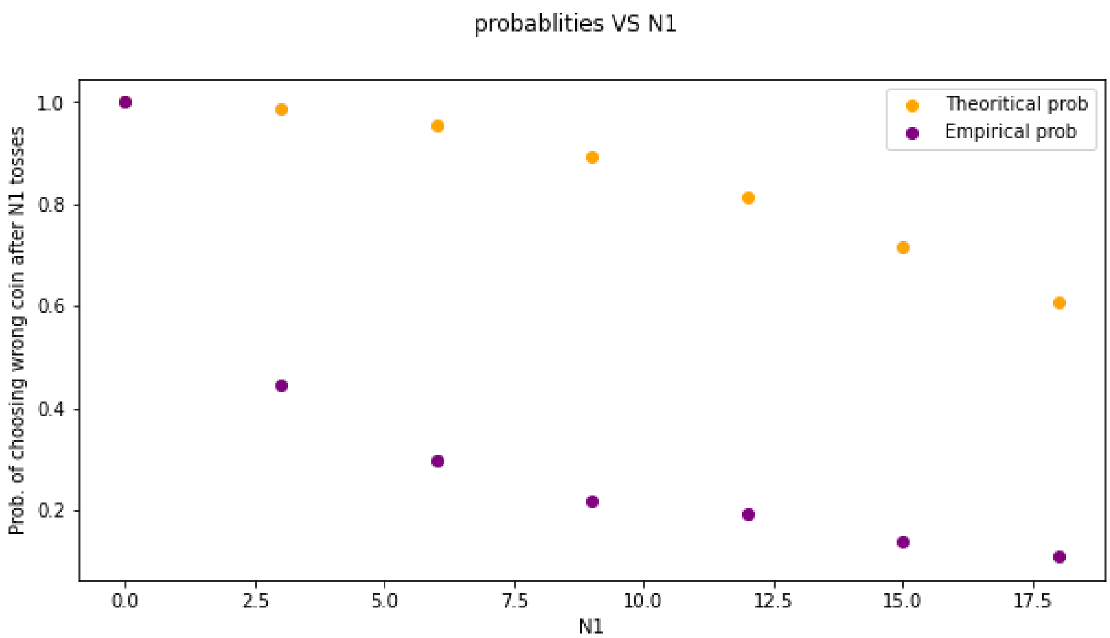
```
#Ga = nCr(N,na)*pow(Pa,na)*pow(1-Pa,N-na)
#Gb = nCr(N,nb)*pow(Pa,nb)*pow(1-Pa,N-nb)
#Gc = nCr(N,nc)*pow(Pa,nc)*pow(1-Pa,N-nc)

for i in range(0,g+1):
 G=0.0
 for nc in range(0,i+1):
   Gc = nCr(N,nc)*pow(Pc,nc)*pow(1-Pc,N-nc)
   for nb in range(0,nc):
     Gb = nCr(N,nb)*pow(Pb,nb)*pow(1-Pb,N-nb)
     for na in range(0,nc):
       Ga = nCr(N,na)*pow(Pa,na)*pow(1-Pa,N-na)
       G=G+(Ga*Gb*Gc)

  G1=0.0
 for nc in range(0,i+1):
  Gc = nCr(N,nc)*pow(Pc,nc)*pow(1-Pc,N-nc)
  for nb in range(0,i+1):
     Gb = nCr(N,nb)*pow(Pb,nb)*pow(1-Pb,N-nb)
     for na in range(0,i+1):
       Ga = nCr(N,na)*pow(Pa,na)*pow(1-Pa,N-na)
       G1=G1+(Ga*Gb*Gc)
 therotical_prob = G/G1.  #this is therotical probablity
   for i in range(0,g+1):
     plt.scatter(i*3,1-therotical_prob,color='orange')
     plt.scatter(i*3,1-(ntcc[i]/1000),color='purple')

plt.xlabel('N1')
plt.ylabel('Prob. of choosing wrong coin after N1 tosses')
plt.suptitle('probablities VS N1')
plt.legend(['Theoritical prob','Empirical prob'])
```

When N1 was 1000 and 5000 then at that time denominator for calculating the conditional probablity that wrong coin is picket was approching to zero and we were not able to store that data coz in every data type that python offers values were not fitting and approximation of that values were 0. and as our denominator was comming zero we were not able to calculate our conditional probablity

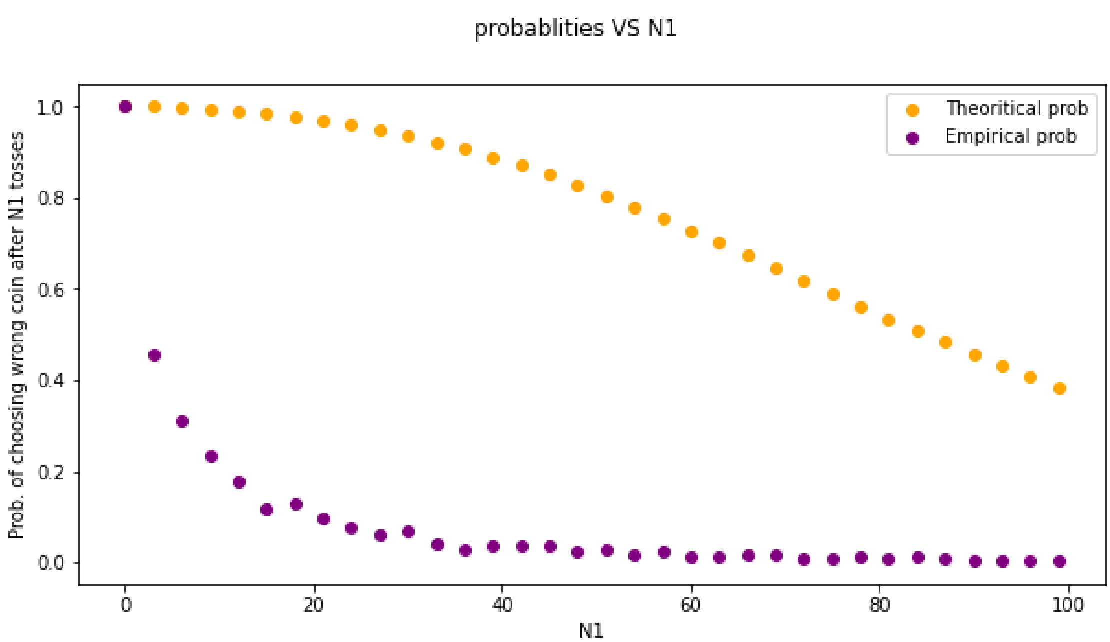
Therefore for case of N1=1000 and N1=5000 we have only plotted Empirical probablity graph as a function of N1

FOR Pa=0.2, Pb=0.4, Pc=0.7

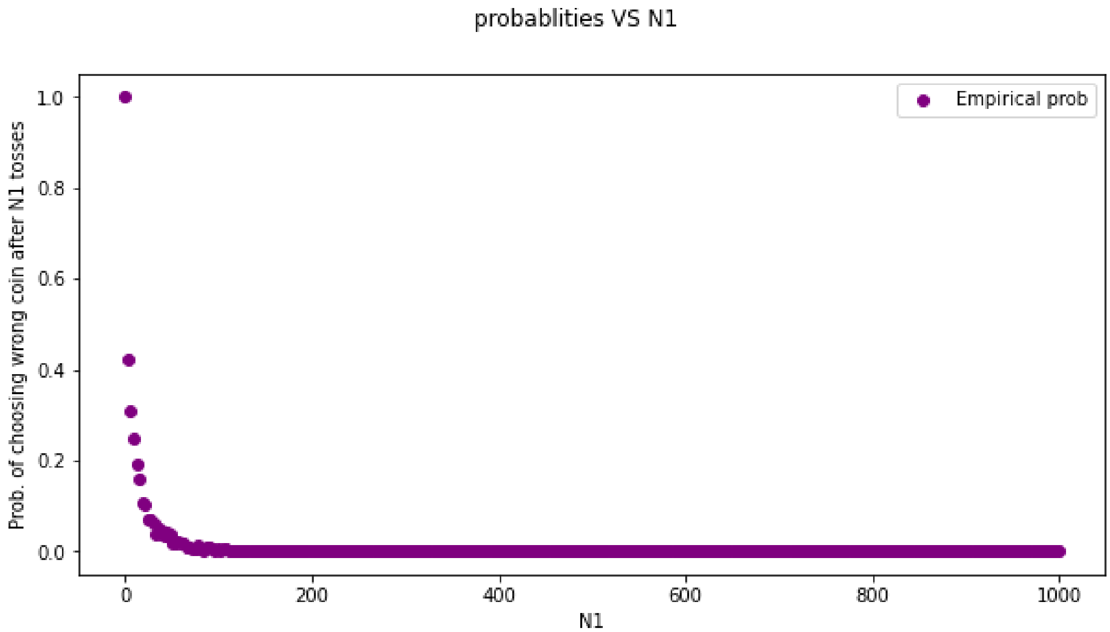
N1=20



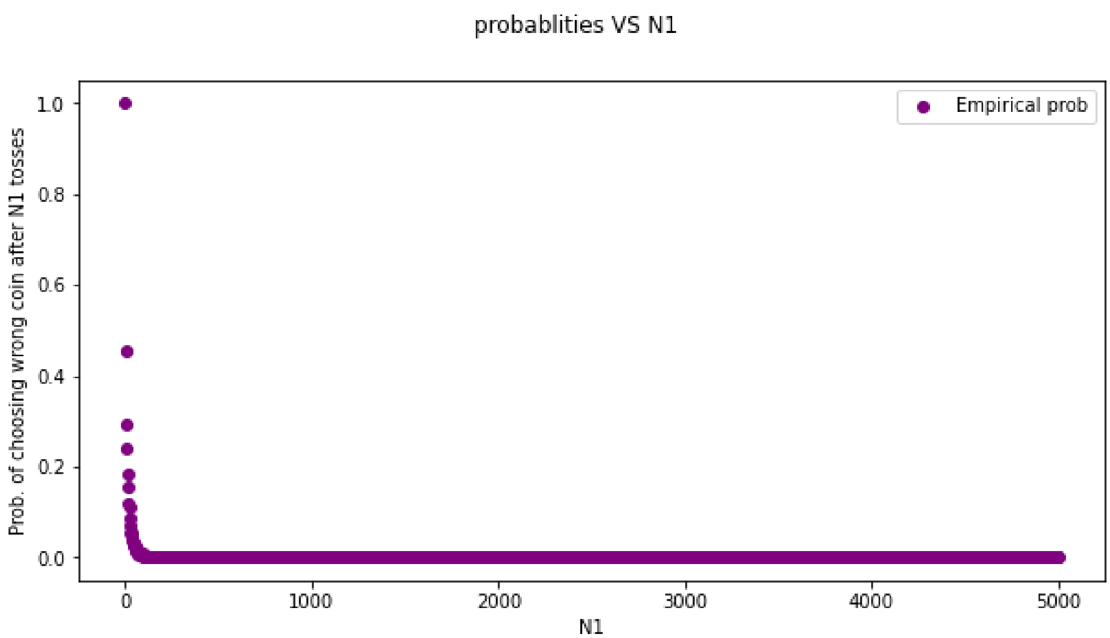
N1=100



N1=1000

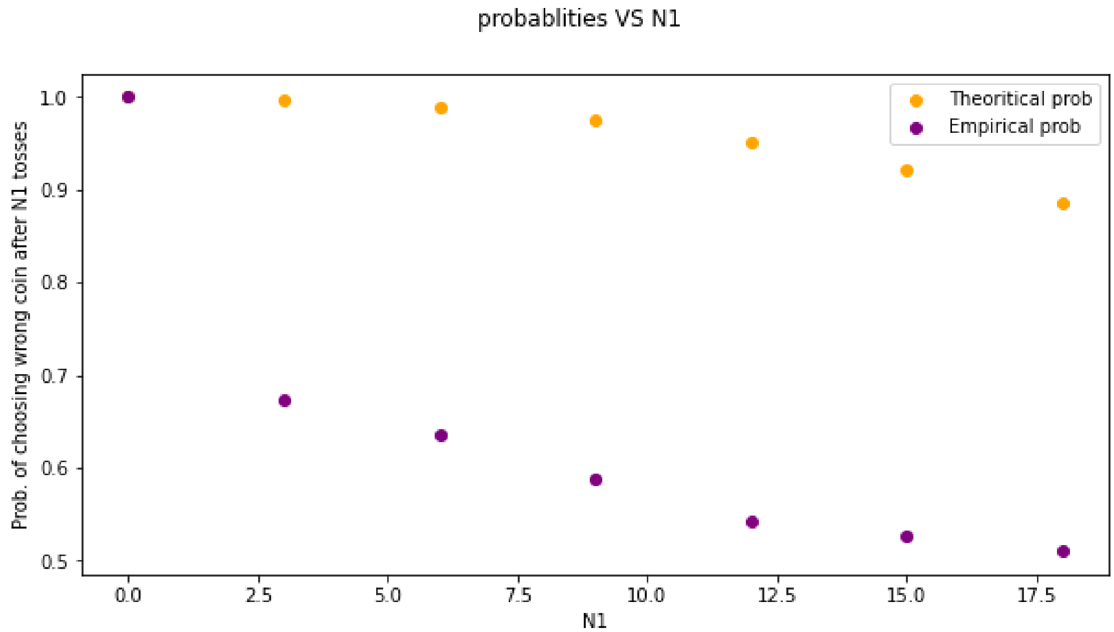


N1=5000

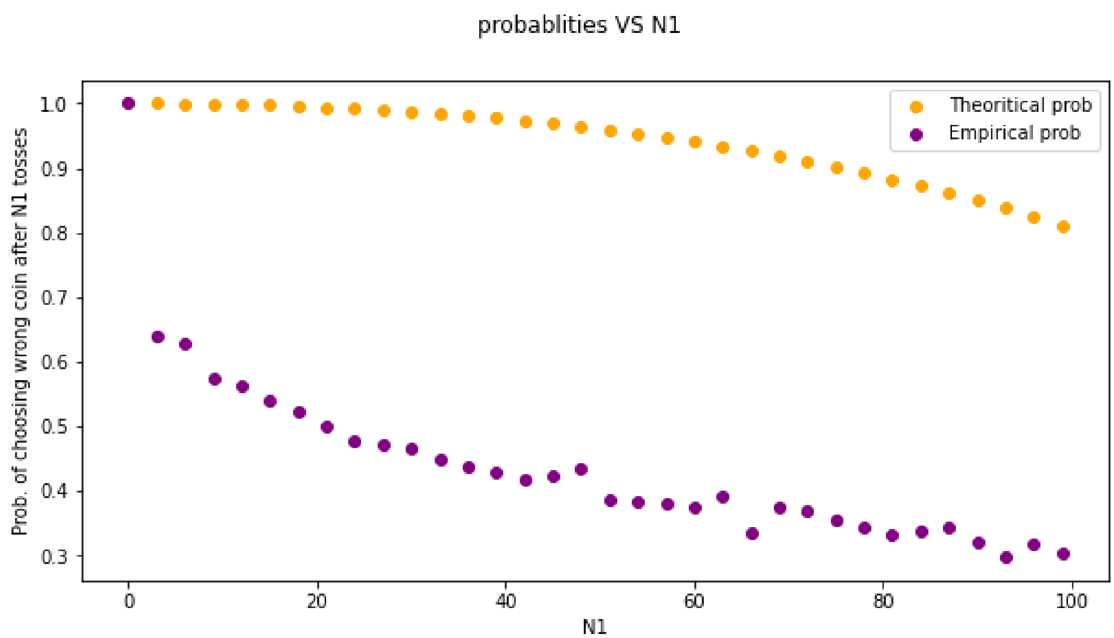


FOR Pa=0.45, Pb=0.5, Pc=0.58

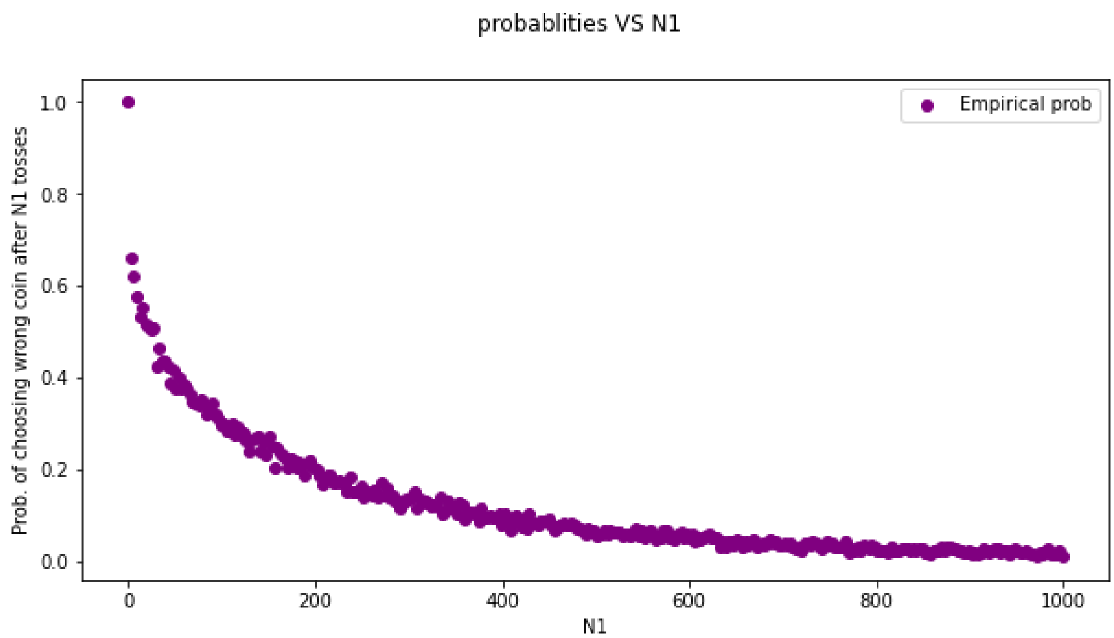
N1 = 20



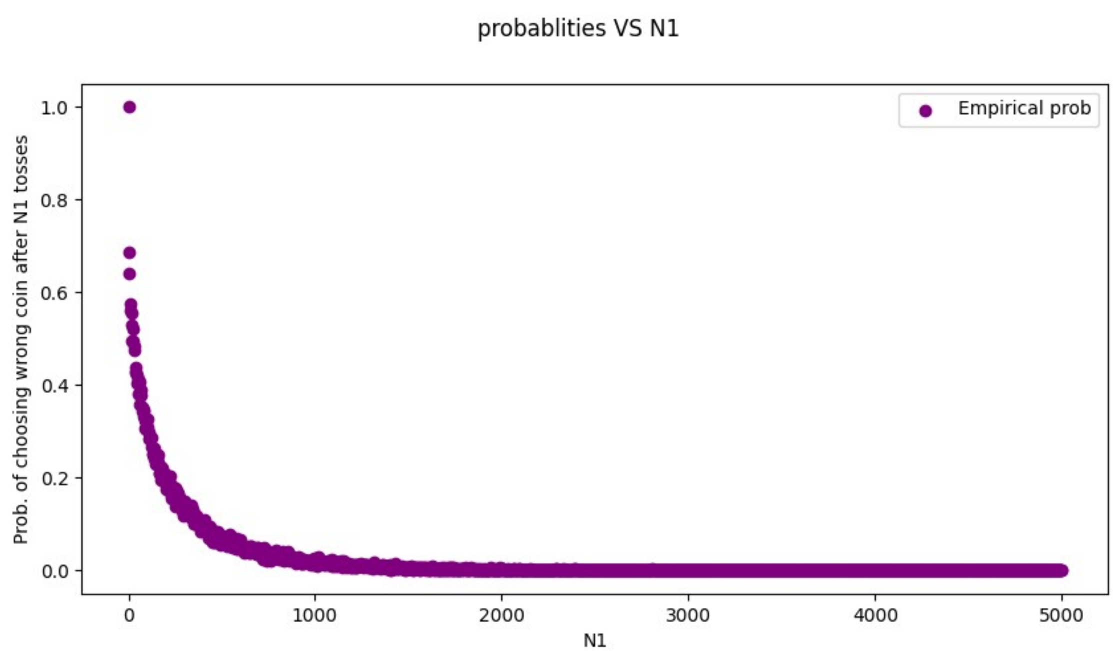
N1=100



N1=1000



N1=5000

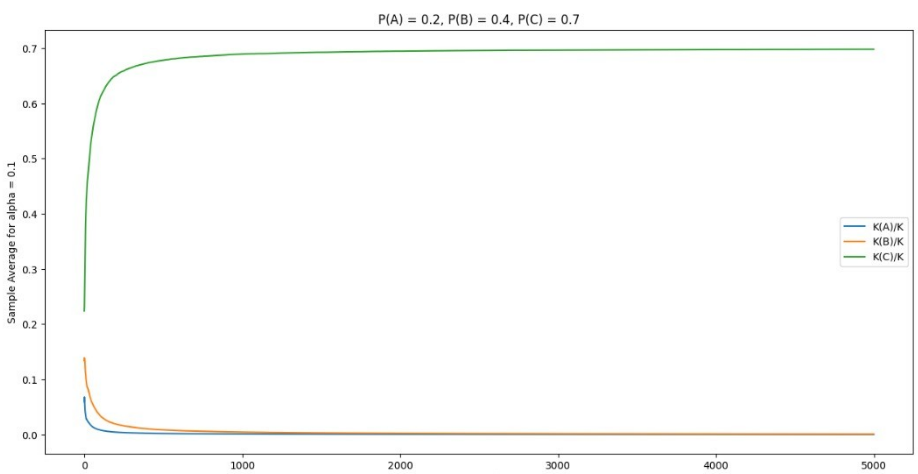


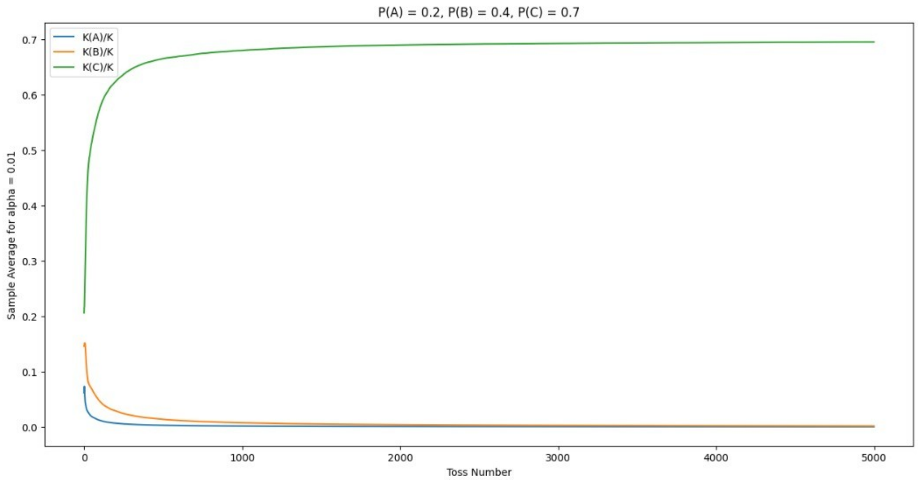
Algorithm B:

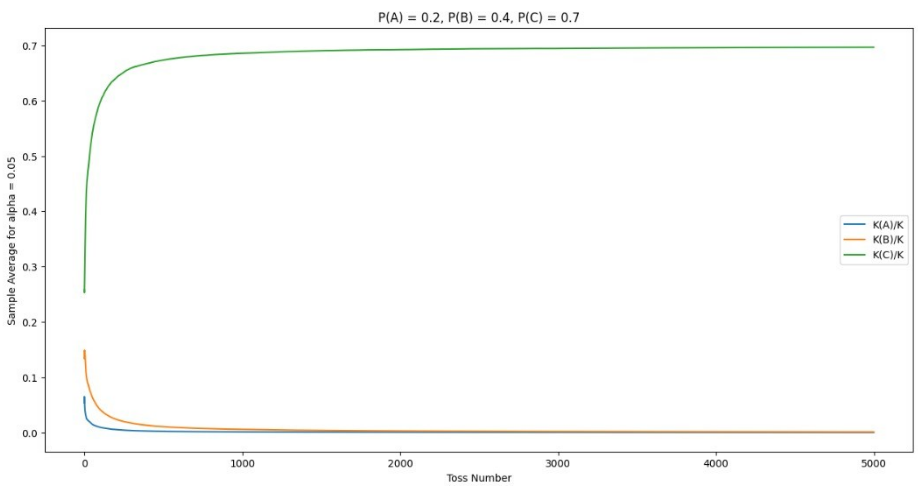
```
p = [0.2, 0.4 , 0.7]
alpha_values = [0.1 , 0.05 , 0.01 ]
no_of_trials = 500

def toss(p):
    if p>=random.random():
```

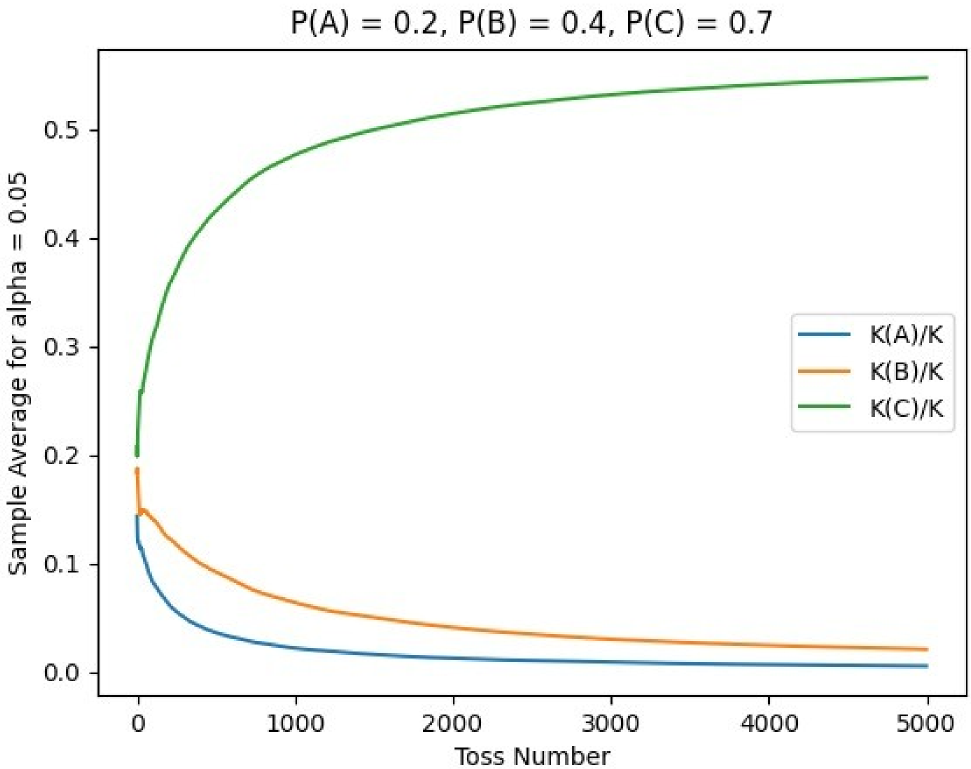
```
        return 1
    else:
        return 0

def calc_ucb(n, k, alpha):
    x = 2*math.log(10)-math.log(alpha*100)
    return k/n + pow((x/(2*n)),0.5)

def simulation(N, alpha):
    n = [0,0,0]
    k = [0,0,0]
    ucb = [1.0,1.0,1.0]
    ka = [0 for x in range(5000)]
    kb = [0 for x in range(5000)]
    kc = [0 for x in range(5000)]

    for i in range(N):
        u = max(ucb[0], ucb[1], ucb[2])
        x, z= 0, 0
        y = []
        for j in range(3):
            if ucb[j]==u:
                x = x+1
                y.append(j)
        z = y[random.randint(0,x-1)]

        n[z]=n[z]+1
        k[z]=k[z]+toss(p[z])
        ucb[z] = calc_ucb(n[z],k[z],alpha)
        ka[i] = k[0]/(i+1)
        kb[i] = k[1]/(i+1)
        kc[i] = k[2]/(i+1)

    results = {"n": n , "k": k, "ka": ka, "kb" :kb, "kc" :kc}
    return results


def table_data():
    for alpha in alpha_values:
        print(" For alpha = ", alpha)
        for N in [20, 100, 1000, 5000]:
            heads_net = 0
            expected_reward = p[2]*N
        #ka_net = [0 for x in range(5000)]
        #kb_net = [0 for x in range(5000)]
        #kc_net = [0 for x in range(5000)]
            for i in range(no_of_trials):
                results = simulation(N, alpha)
                heads_net = heads_net + results["k"][0]+ results["k"]
[1]+ results["k"][2]
```

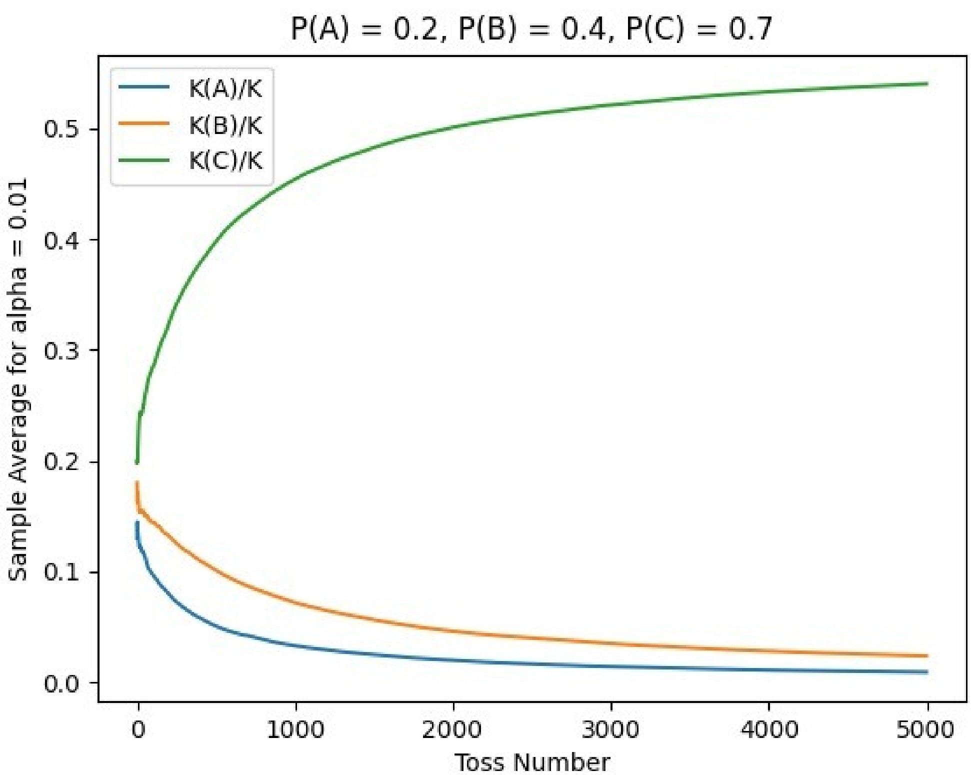
```
            #if N==5000:
                #for j in range(5000):
                    #ka_net[j] = ka_net[j] + results["ka"][j]
                    #kb_net[j] = kb_net[j] + results["kb"][j]
                    #kc_net[j] = kc_net[j] + results["kc"][j]

            avg_reward = heads_net/no_of_trials
        #ka_avg = [ x /no_of_trials for x in ka_net]
        #kb_avg = [ x /no_of_trials for x in kb_net]
        #kc_avg = [ x /no_of_trials for x in kc_net]

        #x_axis = [x for x in range(5000)]
        #plt.plot(x_axis,ka_avg,label = "K(A)/K")
        #plt.plot(x_axis,kb_avg,label = "K(B)/K")
        #plt.plot(x_axis,kc_avg,label = "K(C)/K")
        #plt.xlabel("Toss Number")
        #plt.ylabel("Sample Average for alpha = " + str(alpha))
        #plt.title("P(A) = 0.2, P(B) = 0.4, P(C) = 0.7")
        #plt.legend()
        #plt.show()

            print("  N = ",N, " -> Sample avg of Total Reward =
",avg_reward , ", Expected Reward = ", expected_reward)

print("For P(A,B,C) = ", p)
table_data()
p = [0.45,  0.5 , 0.58]
print("For P(A,B,C) = ", p)
table_data()
```

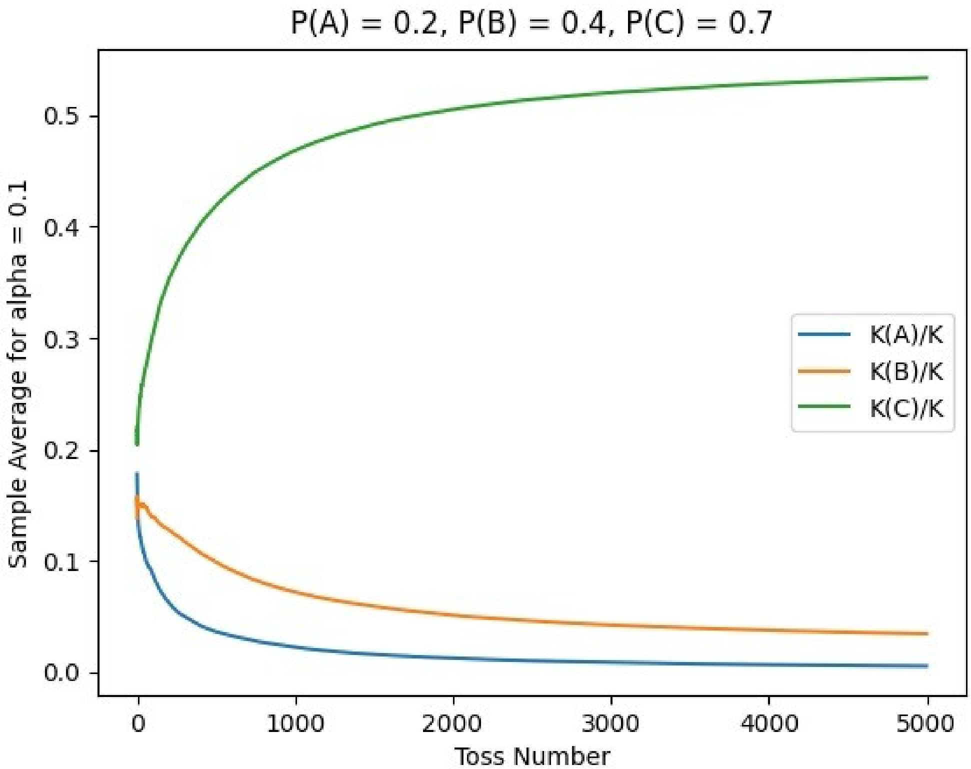



P(A) = 0.2, P(B) = 0.4, P(C) = 0.7
K(A)/K
K(B)/K
K(C)/K
Sample Average for alpha = 0.01
Toss Number
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1000
2000
3000
4000
5000


P(A) = 0.2, P(B) = 0.4, P(C) = 0.7
K(A)/K
K(B)/K
K(C)/K
Sample Average for alpha = 0.05
Toss Number
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1000
2000
3000
4000
5000


P(A) = 0.2, P(B) = 0.4, P(C) = 0.7
Sample Average for alpha = 0.05
Toss Number
K(A)/K
K(B)/K
K(C)/K
0.0
0.1
0.2
0.3
0.4
0.5
0
1000
2000
3000
4000
5000


P(A) = 0.2, P(B) = 0.4, P(C) = 0.7
Sample Average for alpha = 0.01
Toss Number
K(A)/K
K(B)/K
K(C)/K
0.0
0.1
0.2
0.3
0.4
0.5
0
1000
2000
3000
4000
5000




```
For P(A,B,C) =  [0.2, 0.4, 0.7]
 For alpha =  0.1
  N =  20  -> Sample avg of Total Reward =  11.476 , Expected Reward =  14.0
  N =  100  -> Sample avg of Total Reward =  65.498 , Expected Reward =  70.0
  N =  1000  -> Sample avg of Total Reward =  693.848 , Expected Reward =  700.0
  N =  5000  -> Sample avg of Total Reward =  3493.28 , Expected Reward =  3500.0
 For alpha =  0.05
  N =  20  -> Sample avg of Total Reward =  11.292 , Expected Reward =  14.0
  N =  100  -> Sample avg of Total Reward =  64.978 , Expected Reward =  70.0
  N =  1000  -> Sample avg of Total Reward =  692.494 , Expected Reward =  700.0
  N =  5000  -> Sample avg of Total Reward =  3492.99 , Expected Reward =  3500.0
 For alpha =  0.01
  N =  20  -> Sample avg of Total Reward =  11.212 , Expected Reward =  14.0
  N =  100  -> Sample avg of Total Reward =  63.288 , Expected Reward =  70.0
  N =  1000  -> Sample avg of Total Reward =  689.406 , Expected Reward =  700.0
  N =  5000  -> Sample avg of Total Reward =  3488.002 , Expected Reward =  3500.0
For P(A,B,C) =  [0.45, 0.5, 0.58]
 For alpha =  0.1
  N =  20  -> Sample avg of Total Reward =  10.448 , Expected Reward =  11.6
  N =  100  -> Sample avg of Total Reward =  53.222 , Expected Reward =  57.99999999999999
  N =  1000  -> Sample avg of Total Reward =  566.142 , Expected Reward =  580.0
  N =  5000  -> Sample avg of Total Reward =  2865.066 , Expected Reward =  2900.0
 For alpha =  0.05
  N =  20  -> Sample avg of Total Reward =  10.426 , Expected Reward =  11.6
  N =  100  -> Sample avg of Total Reward =  52.844 , Expected Reward =  57.99999999999999
  N =  1000  -> Sample avg of Total Reward =  561.19 , Expected Reward =  580.0
  N =  5000  -> Sample avg of Total Reward =  2871.654 , Expected Reward =  2900.0
 For alpha =  0.01
  N =  20  -> Sample avg of Total Reward =  10.25 , Expected Reward =  11.6
  N =  100  -> Sample avg of Total Reward =  52.922 , Expected Reward =  57.99999999999999
  N =  1000  -> Sample avg of Total Reward =  559.596 , Expected Reward =  580.0
  N =  5000  -> Sample avg of Total Reward =  2865.848 , Expected Reward =  2900.0
```

C)

Effect of N : As N increases, both expected reward and best reward increase because the number of tosses has increased so there will be more number of heads.

Effect of p: Increasing p increases the expected reward and best reward as UCB for expected reward increases. Best reward is simply N*p(max) so best reward also increases.

Effect of alpha: Alpha has no effect on best reward. But as alpha decreases, expected reward decreases because UCB will also increase for all 3 coins. Thus it will take longer for UCB to reach close to it's actual p.