EE325 Assignment 2

Anoushka Dey 210010010
Savaliya Abhishek 21D070065
Ritesh Bahl 21D070057

October 3, 2022

Algorithm A

Here first we tossed all coins N1/3 times and found out which coin gave max number of heads, then tossed that coin for remaining N - N1/3 times. Theorical probability of choosing wrong coin is given by

$$P_{th} = P(N_A > N_C or N_B > N_C)$$

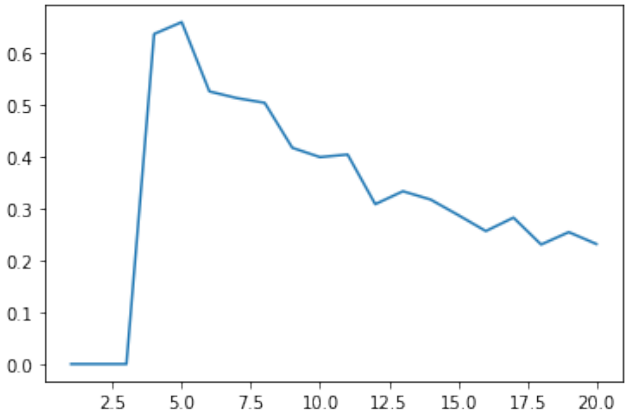
This is the code:

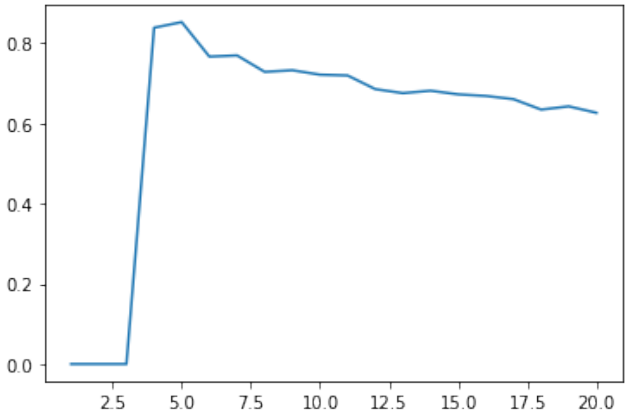
```
import matplotlib.pyplot as plt
import numpy as np
import math
import random

pa=0.2
pb=0.4
pc=0.7
N=[20,100,1000,5000]
A1=[1,1,0,0,0,0,0,0,0,0]
B1=[1,1,1,1,0,0,0,0,0,0]
C1=[1,1,1,1,1,1,1,0,0,0]
x=0
N1=np.arange(1,N[x]+1)
sample_avg_RN1=np.zeros(N[x])
empirical_prob=np.zeros(N[x])
theorical_prob=np.zeros(N[x])

for i in range(3,N[x]):
  RN1=np.zeros(10)
  w=0
  for j in range(10):
      na=0
      nb=0
```

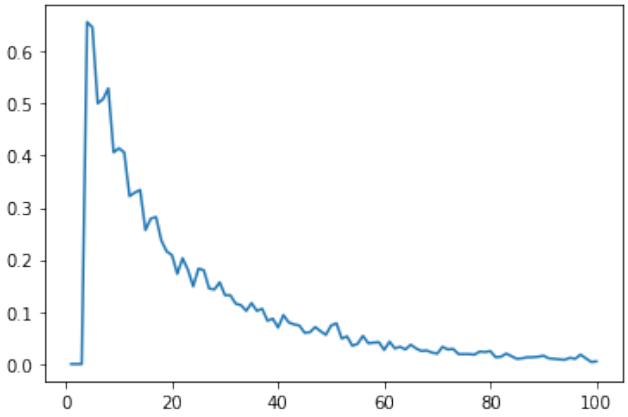
```
        nc=0
        for k in range(int(N1[i]/3)):
          na += random.choice(A1)
          nb += random.choice(B1)
          nc += random.choice(C1)
        RN1[j]=na+nb+nc        #no of heads till N1 tosses
        D=np.zeros(10)
        if max(na,nb,nc) == na :
          D=A1
        elif max(na,nb,nc) == nb :
          D=B1
        elif na==nb & na>nc :
          D=B1
        else :
          D=C1
        if D == A1 or D == B1:
          w += 1
        for k in range(N[x]-int(N1[i])) :
          RN1[j] += random.choice(D)
    sample_avg_RN1[i]= RN1.mean()
    empirical_prob[i]= w/1000
print(sample_avg_RN1)
plt.plot(N1,sample_avg_RN1)

def annot_max(x,y, ax=None):
    xmax = x[np.argmax(y)]
    ymax = y.max()
    text= "x={:.3f},␣y={:.3f}".format(xmax, ymax)
    if not ax:
        ax=plt.gca()
    bbox_props = dict(boxstyle="square,pad=0.3", fc="w",
        ec="k", lw=0.72)
    arrowprops=dict(arrowstyle="->",connectionstyle="
        angle,angleA=0,angleB=60")
    kw = dict(xycoords='data',textcoords="axes␣fraction",
              arrowprops=arrowprops, bbox=bbox_props, ha=
                  "right", va="top")
    ax.annotate(text, xy=(xmax, ymax), xytext=(0.94,0.96)
        , **kw)

annot_max(N1,sample_avg_RN1)

plt.show()
print(empirical_prob)
plt.plot(N1,empirical_prob)
plt.show()
```

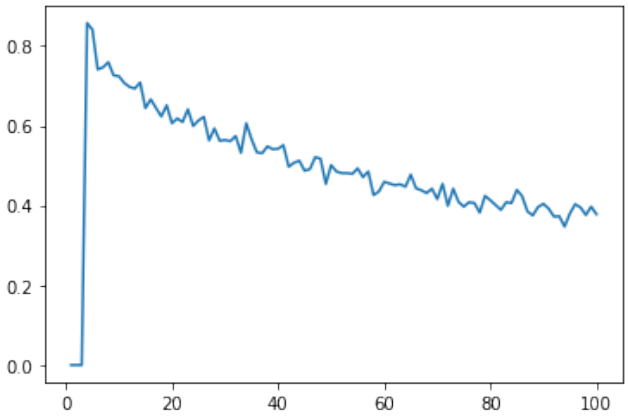
```
def nCr(n,r):
    f = math.factorial
    return f(n) // f(r) // f(n-r)

def pmfc(i,N1):
   ncr=nCr(int(N1/3),i)
   return ncr*(math.pow(pc,i))*(math.pow(1-pc,int(N1/3)-i
       ))
def pmfb(i,N1):
   ncr=nCr(int(N1/3),i)
   return ncr*(math.pow(pb,i))*(math.pow(1-pb,int(N1/3)-i
       ))
def pmfa(i,N1):
   ncr=nCr(int(N1/3),i)
   return ncr*(math.pow(pa,i))*(math.pow(1-pa,int(N1/3)-i
       ))
for j in range(N[x]):
  terma=0
  termb=0
  for i in range(int(N1[j]/3)):
    localsum1=0
    localsum2=0
    for k in range(1,int(N1[j]/3)-i):
       localsum1 +=pmfa(i+k,N1[j])
       localsum2 +=pmfb(i+k,N1[j])
    terma += localsum1*pmfc(i,N1[j])
    termb += localsum2*pmfc(i,N1[j])
  theorical_prob[j]= terma + termb -terma*termb
print(theorical_prob)
plt.plot(N1,theorical_prob)
```

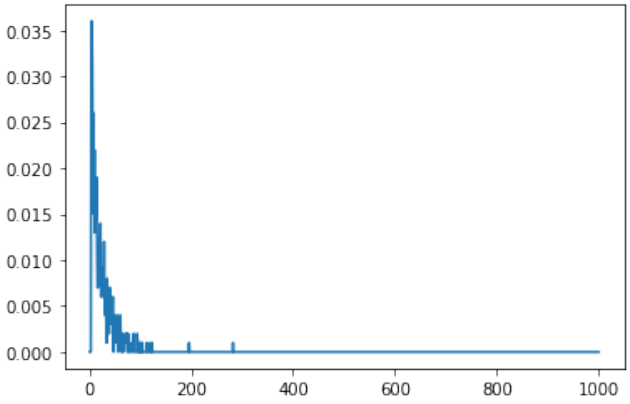
By changing value of x, pa, pb, pc in above code we can get graphs for different cases.
Emperical Probability Graphs:

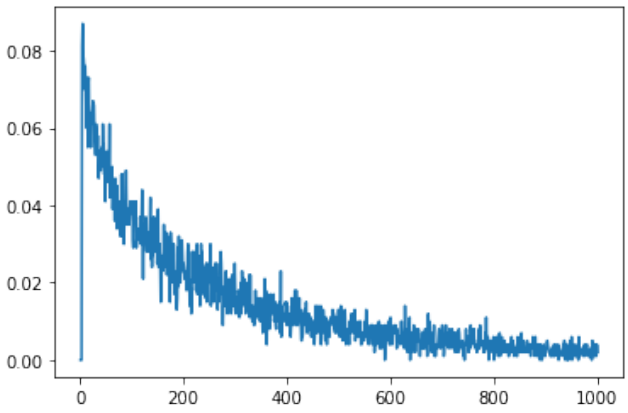

0.6
0.5
0.4
0.3
0.2
0.1
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

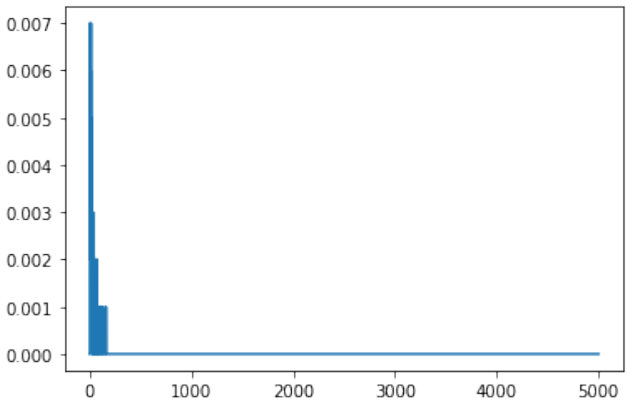

0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

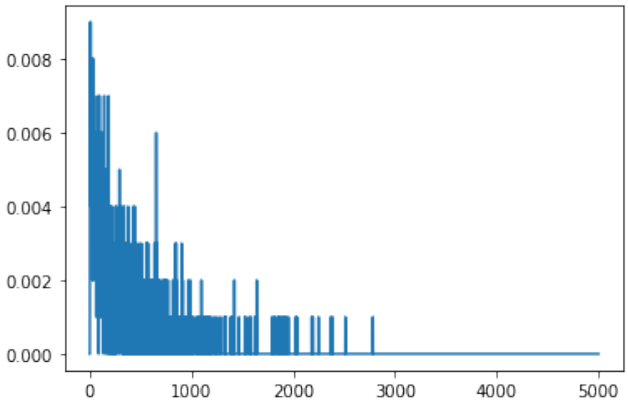

0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

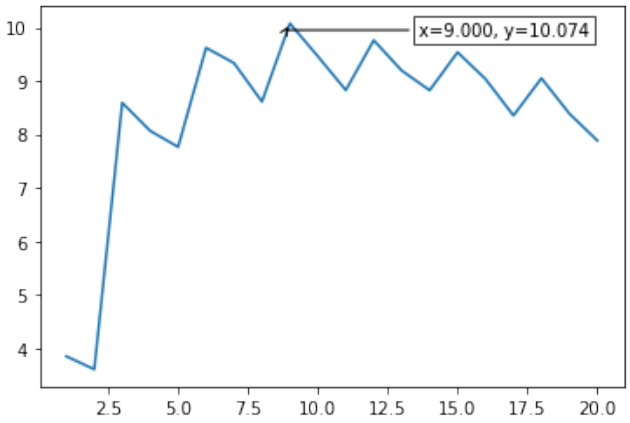

0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

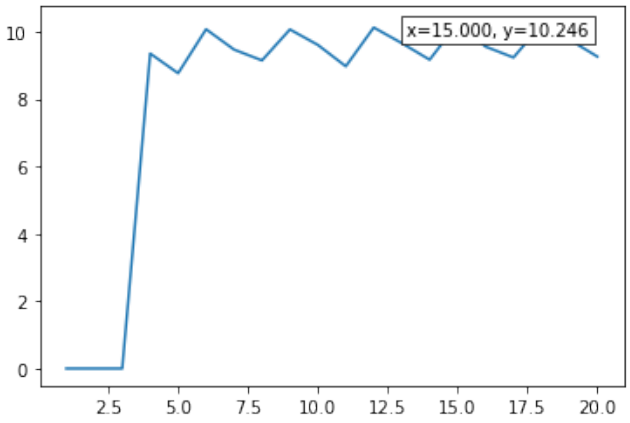

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
200
400
600
800
1000


0.08
0.06
0.04
0.02
0.00
0
200
400
600
800
1000

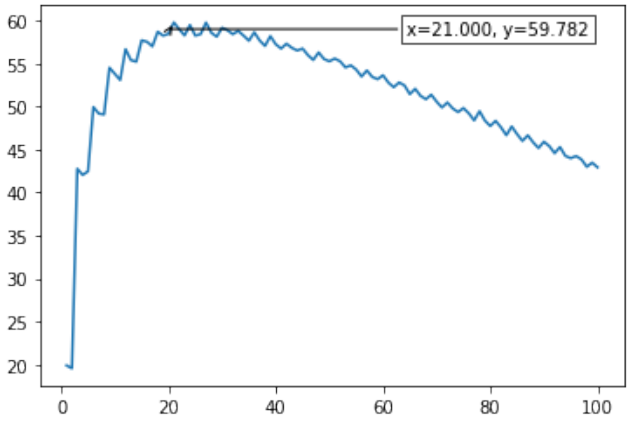

0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
1000
2000
3000
4000
5000

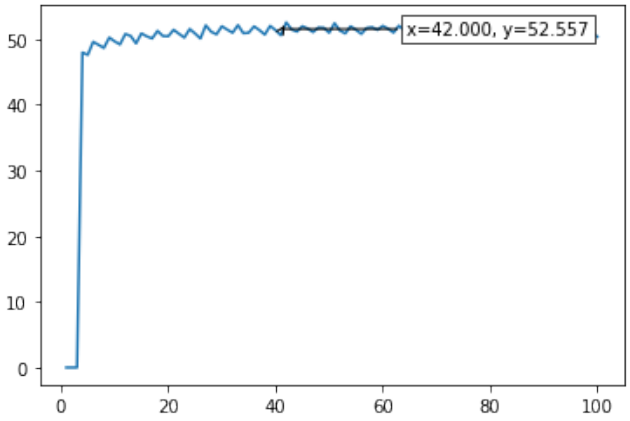

0.008
0.006
0.004
0.002
0.000
0
1000
2000
3000
4000
5000

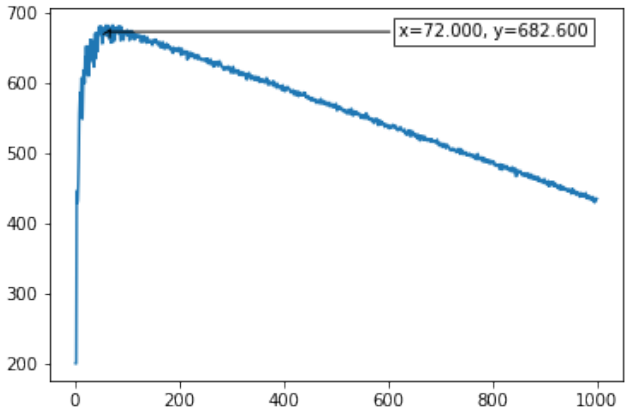


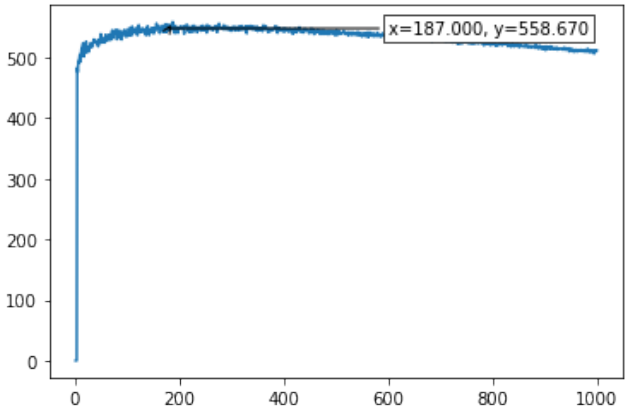


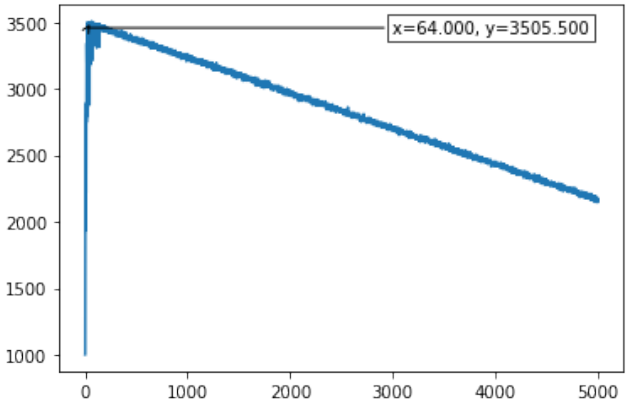
The R(N1) vs N1 graphs:

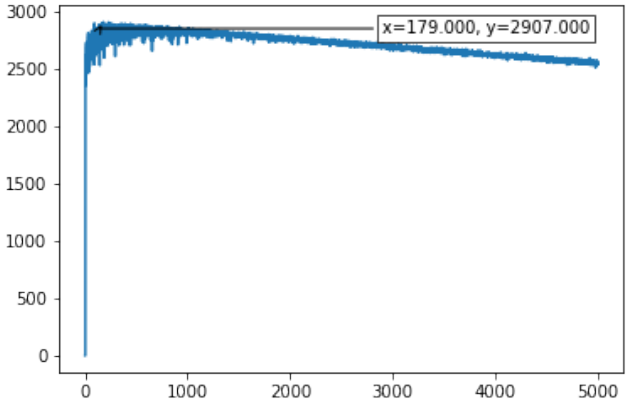

x=21.000, y=59.782
60
55
50
45
40
35
30
25
20
0
20
40
60
80
100

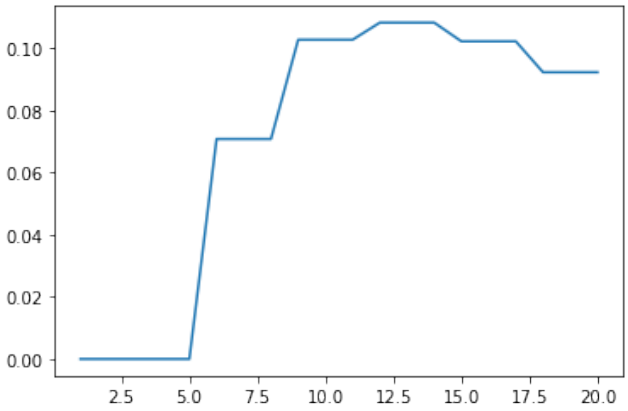

x=42.000, y=52.557
50
40
30
20
10
0
0
20
40
60
80
100

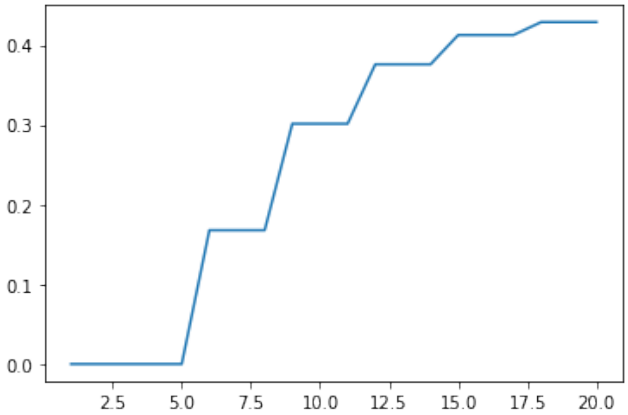

x=72.000, y=682.600
700
600
500
400
300
200
0
200
400
600
800
1000


x=187.000, y=558.670
500
400
300
200
100
0
0
200
400
600
800
1000

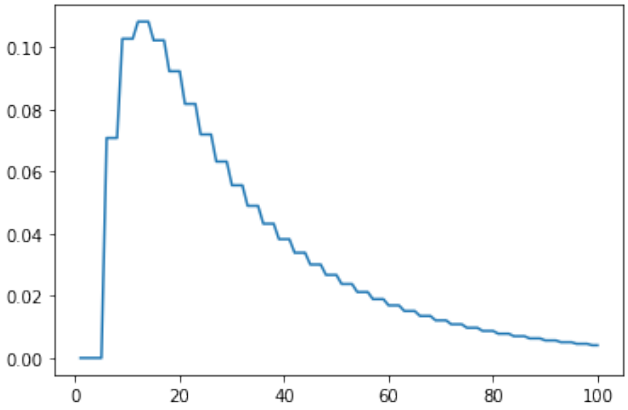

x=64.000, y=3505.500
3500
3000
2500
2000
1500
1000
0
1000
2000
3000
4000
5000

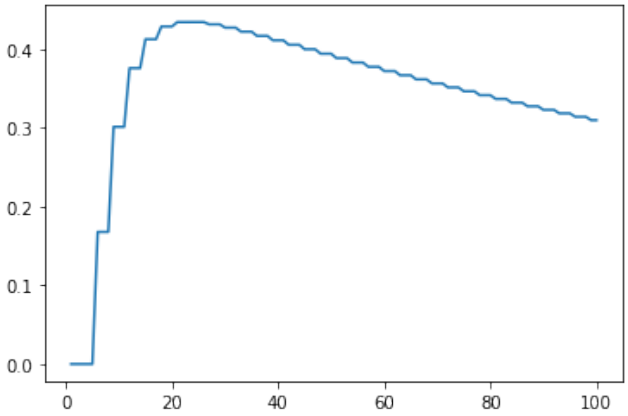

x=179.000, y=2907.000
3000
2500
2000
1500
1000
500
0
0
1000
2000
3000
4000
5000

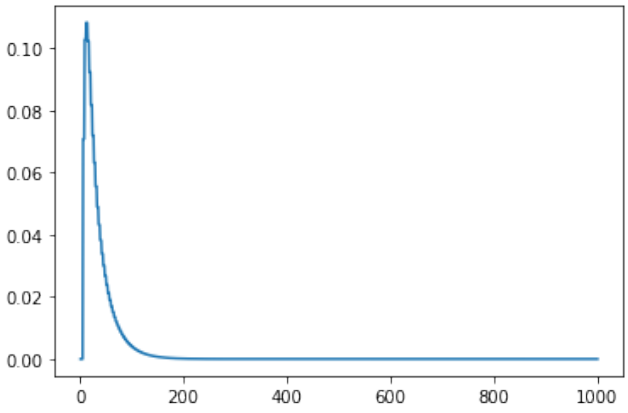


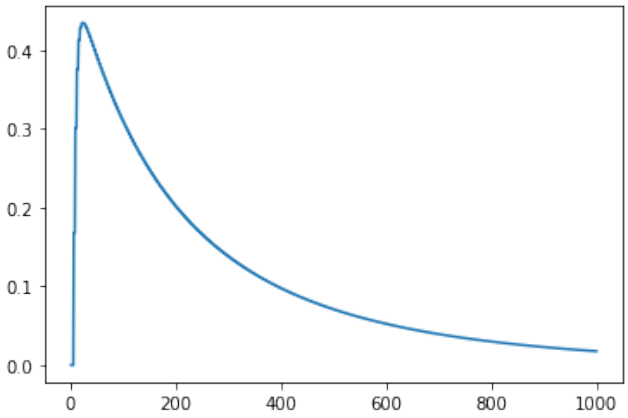


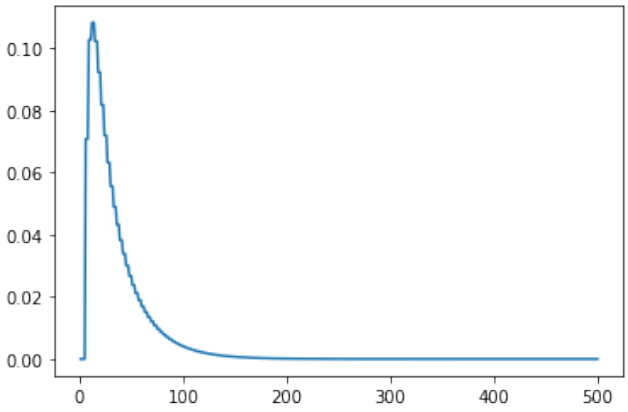
Theoretical Probability Graphs:

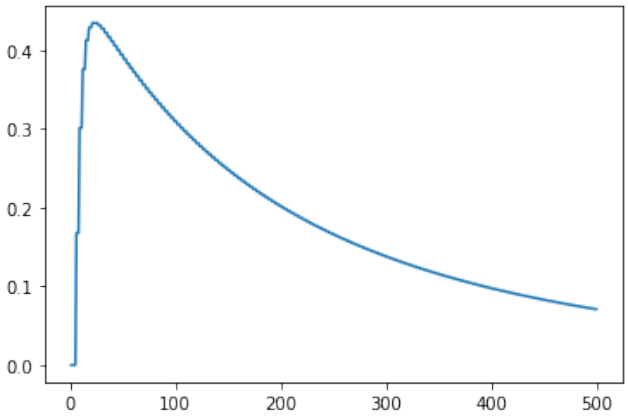

0.10
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80
100


0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100


0.10
0.08
0.06
0.04
0.02
0.00
0
200
400
600
800
1000


0.4
0.3
0.2
0.1
0.0
0
200
400
600
800
1000


0.10
0.08
0.06
0.04
0.02
0.00
0
100
200
300
400
500


0.4
0.3
0.2
0.1
0.0
0
100
200
300
400
500

Algorithm B

Using Hoeffding's Inequality, we arrive at the relation:

$$UCB_A(k) = \frac{k_a(k)}{n_a(k)} + \sqrt{\frac{-ln(\alpha)}{2n_a(k)}}$$

Functions were defined to generate heads when a random number generated was less than the value of probability passed to the function, to find the UCB and the maximum UCB of the coins being used. Then a function was defined that carried out the algorithm of tossing the coins, finding the maximum UCB, choosing the best coin for the $(k+1)^{th}$ and counting $k_a(k)$, $k_b(k)$ and $k_c(k)$ along with $n_a(k)$, $n_b(k)$ and $n_c(k)$. The algorithm here has been repeated 500 times to find the sample averages as mentioned in part (i) of the question. The corresponding values have also been plotted.
This is the code:

```
import numpy as np
import matplotlib.pyplot as plt
import random
import math
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (15,10)
def heads(p):
  return 1*(random.random()<p)
def find_ucb(k,n,a):
   return k/n + (-math.log(a)/(2*n))**0.5
def find_max_ucb(ucb):
  umax=max(ucb)
  pos=[]
  c=0
  for u in ucb:
    if u==umax:
      c+=1
      pos.append(ucb.index(u))
  if c==1:
    return pos[0]
  elif c==2:
    return (random.random()<=0.5)*pos[0]+ (random.random
        ()>0.5)*pos[1]
  else:
    return (random.random()<=1/3)*pos[0]+ (random.random
        ()<=2/3 and random.random()>1/3)*pos[1]+ (random.
        random()>2/31/3)*pos[2]
N=5000
p=[[0.2,0.4,0.7],[0.45,0.5,0.58]]
```

```
a=[0.1,0.05,0.01]
def bandit(p,a):
  headarr=np.zeros((5000,3))
  for j in range(0,500):
      n=[0,0,0]
      k=[0,0,0]
      ucb=[10,10,10]
      for i in range(0,N):
          id=find_max_ucb(ucb)
          n[id]+=1
          k[id]+=heads(p[id])
          ucb[id]=find_ucb(k[id],n[id],a)
          headarr[i]+=np.array(k)/(i+1)
  return np.array(headarr)/500
x=np.linspace(1,5000,5000)
def pl(p,a,coinNo):
  h=bandit(p,a)
  plt.plot(x,h[:,(coinNo-1)])
  plt.xlabel("No. of tosses")
  plt.ylabel("Sample Average")
plt.figure(1)
pl(p[0],0.1,1)
pl(p[0],0.1,2)
pl(p[0],0.1,3)
plt.legend(["p=0.2","p=0.4","p=0.7"])
plt.title("Alpha=0.1")
plt.figure(2)
pl(p[0],0.05,1)
pl(p[0],0.05,2)
pl(p[0],0.05,3)
plt.legend(["p=0.2","p=0.4","p=0.7"])
plt.title("Alpha=0.05")
plt.figure(3)
pl(p[0],0.01,1)
pl(p[0],0.01,2)
pl(p[0],0.01,3)
plt.legend(["p=0.2","p=0.4","p=0.7"])
plt.title("Alpha=0.01")
plt.figure(4)
pl(p[1],0.1,1)
pl(p[1],0.1,2)
pl(p[1],0.1,3)
plt.legend(["p=0.45","p=0.5","p=0.58"])
plt.title("Alpha=0.1")
plt.figure(5)
pl(p[1],0.05,1)
```

```
pl(p[1],0.05,2)
pl(p[1],0.05,3)
plt.legend(["p=0.45","p=0.5","p=0.58"])
plt.title("Alpha=0.05")
plt.figure(6)
pl(p[1],0.01,1)
pl(p[1],0.01,2)
pl(p[1],0.01,3)
plt.legend(["p=0.45","p=0.5","p=0.58"])
plt.title("Alpha=0.01")
def tot_reward(N,p,a):
  tot=0
  for j in range(0,500):
      n=[0,0,0]
      k=[0,0,0]
      ucb=[10,10,10]
      for i in range(0,N):
          id=find_max_ucb(ucb)
          n[id]+=1
          k[id]+=heads(p[id])
          ucb[id]=find_ucb(k[id],n[id],a)
      tot+=sum(np.array(k))
  return tot/500
print(tot_reward(5000,p[0],0.1))
print(tot_reward(5000,p[0],0.05))
print(tot_reward(5000,p[0],0.01))
print(5000*0.7)
print(tot_reward(5000,p[1],0.1))
print(tot_reward(5000,p[1],0.05))
print(tot_reward(5000,p[1],0.01))
print(5000*0.58)
print(tot_reward(1000,p[0],0.1))
print(tot_reward(1000,p[0],0.05))
print(tot_reward(1000,p[0],0.01))
print(1000*0.7)
print(tot_reward(1000,p[1],0.1))
print(tot_reward(1000,p[1],0.05))
print(tot_reward(1000,p[1],0.01))
print(1000*0.58)
print(tot_reward(100,p[0],0.1))
print(tot_reward(100,p[0],0.05))
print(tot_reward(100,p[0],0.01))
print(100*0.7)
print(tot_reward(100,p[1],0.1))
print(tot_reward(100,p[1],0.05))
print(tot_reward(100,p[1],0.01))
```

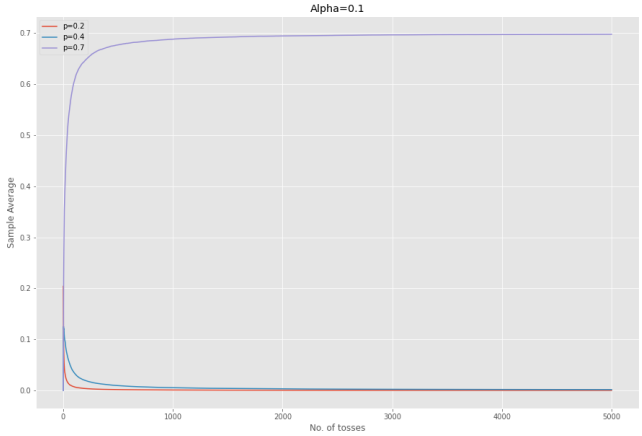
```
119 print(100*0.58)
120 print(tot_reward(20,p[0],0.1))
121 print(tot_reward(20,p[0],0.05))
122 print(tot_reward(20,p[0],0.01))
123 print(20*0.7)
124 print(tot_reward(20,p[1],0.1))
125 print(tot_reward(20,p[1],0.05))
126 print(tot_reward(20,p[1],0.01))
127 print(20*0.58)
```

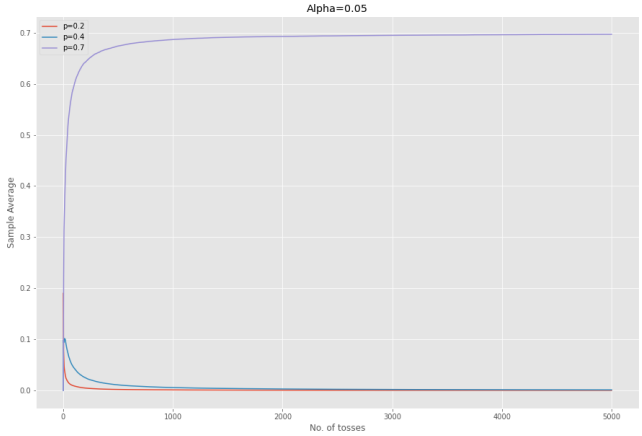
For part(ii) of the question the tabulated form is:

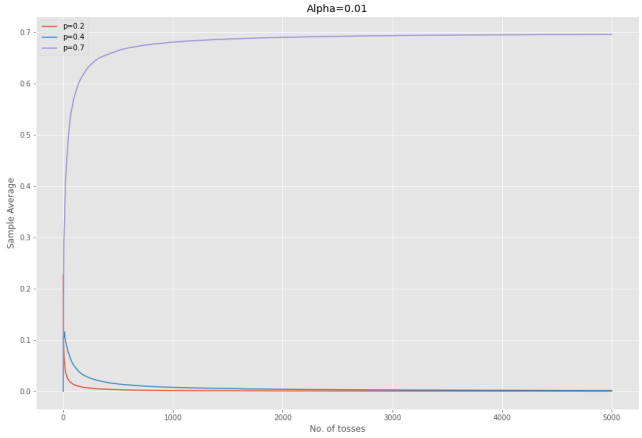
N	p_a	p_b	p_c	α	Sample Avg	Best Expected Reward
5000	0.2	0.4	0.7	0.1	3494.43	3500.0
5000	0.2	0.4	0.7	0.05	3492.956	3500.0
5000	0.2	0.4	0.7	0.01	3490.926	3500.0
5000	0.2	0.4	0.7	0.01	3490.926	3500.0
5000	0.45	0.5	0.58	0.1	2872.184	2900.0
5000	0.45	0.5	0.58	0.05	2872.686	2900.0
5000	0.45	0.5	0.58	0.01	2872.366	2900.0
1000	0.2	0.4	0.7	0.1	693.044	700.0
1000	0.2	0.4	0.7	0.05	693.022	700.0
1000	0.2	0.4	0.7	0.01	690.702	700.0
1000	0.45	0.5	0.58	0.1	562.786	580.0
1000	0.45	0.5	0.58	0.05	562.026	580.0
1000	0.45	0.5	0.58	0.01	558.88	580.0
100	0.2	0.4	0.7	0.1	65.044	70.0
100	0.2	0.4	0.7	0.05	64.74	70.0
100	0.2	0.4	0.7	0.01	63.29	70.0
100	0.45	0.5	0.58	0.1	53.172	58.0
100	0.45	0.5	0.58	0.05	53.164	58.0
100	0.45	0.5	0.58	0.01	52.73	58.0
20	0.2	0.4	0.7	0.1	11.082	14.0
20	0.2	0.4	0.7	0.05	11.004	14.0
20	0.2	0.4	0.7	0.01	10.762	14.0
20	0.45	0.5	0.58	0.1	10.488	11.6
20	0.45	0.5	0.58	0.05	10.232	11.6
20	0.45	0.5	0.58	0.01	10.224	11.6

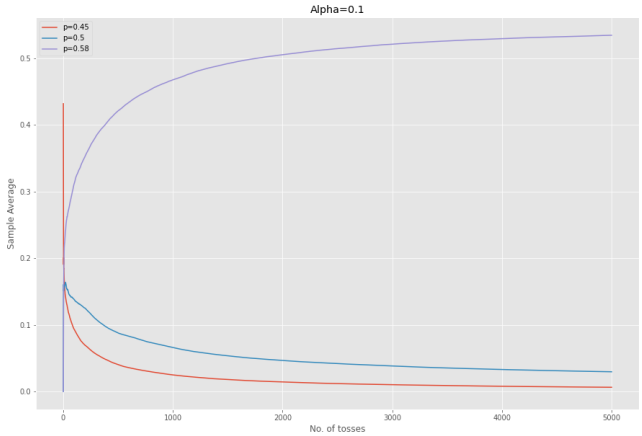
In part c), we can conclude that as N increases, the sample average of the expected reward increases as we are tossing all the coins more number of times, especially the best coin after we establish the coin with the highest UCB after every k tosses. Here k ranges from 1 to N. Also the best expected reward increases as it is simply Np*, where p* is the coin whose probability of yielding heads is the highest. As p increases, we can say that it will eventually result in having a higher UCB and hence a higher chance of being selected and being tossed. Also, it also yields heads more number of times. Hence, the reward increases. In the case of α, the sample average of the total reward increases.

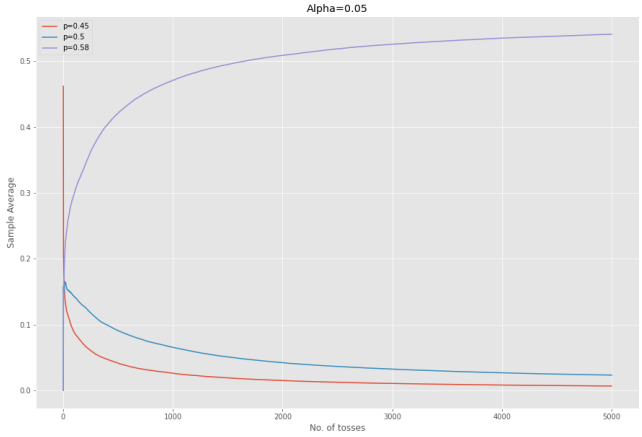
These are the graphs:

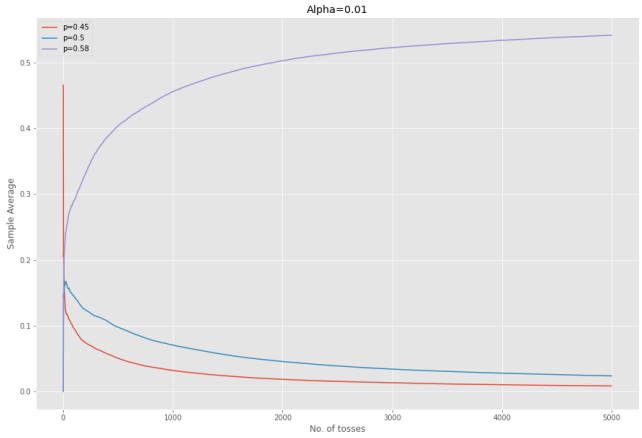





Alpha=0.01
p=0.2
p=0.4
p=0.7
Sample Average
No. of tosses


Alpha=0.1
p=0.45
p=0.5
p=0.58
Sample Average
No. of tosses


Alpha=0.05
p=0.45
p=0.5
p=0.58
Sample Average
No. of tosses


Alpha=0.01
p=0.45
p=0.5
p=0.58
Sample Average
No. of tosses
0.0
0.1
0.2
0.3
0.4
0.5
0
1000
2000
3000
4000
5000