In [102]:
```python
# importing necessary packages
import numpy as np
import matplotlib.pyplot as plt
```

In [105]:
```python
# Newton's Method

#first function
def f(x):
    return x**2 - 1

#derivative of first function
def fprime(x):
    return 2*x

# defining our newton's method
def Newtons_Method(maxIter, tol, f, fprime, x_0):

    # counting iterations
    iteration = 1
    # set conditions for loop
    while (iteration < maxIter and abs(f(x_0)) > tol):
        # calculating xn
        x_1 = x_0 - f(x_0)/fprime(x_0)
        # add to counter
        iteration += 1
        if abs(x_1-x_0) < tol:
            break
        else:
            x_0 = x_1
        print(iteration, x_1, abs(x_1-1))

    return x_1

# printing out first iteration
x_0 = 2.000
print('1', x_0, abs(x_0-1))

# running function
Newtons_Method(6, 10**-8, f, fprime, x_0)
```

```
1 2.0 1.0
2 1.25 0.25
3 1.025 0.02499999999999991
4 1.0003048780487804 0.00030487804878043256
5 1.0000000464611474 4.6461147373833e-08
6 1.000000000000001 1.1102230246251565e-15
```

Out[105]:
```
1.000000000000001
```

In [87]:
```python
# LaGrange Interpolation

# created class to deal with x and y values
class Data:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def lagrange_basis(f, target):
    n = len(f)
    # creating counter for sum
    starting = 0
    # conditions for product
    for i in range(n):
        poly = f[i].y
        for j in range(n):
            if j!= i:
                # calculating the lagrange polynomial
                poly = poly*(target-f[j].x)/(f[i].x-f[j].x)
        # adding polynomial to y
        starting += poly
    return starting
```

In [88]:
```python
# equal spaced between -1 and 1
x_pt = np.linspace(-1, 1, 3)
# 500 equal spaced between -1 and 1
targets1 = np.linspace(-1, 1, 500)

# empty array
y_pt = []
# plugging into function
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
# points of x and y
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_1 = []
# plugging into our function
# assigning to graph number
for k in targets1:
    gr_1.append(lagrange_basis(func, k))
```

In [89]:
```
# same process as above for all other n values
x_pt = np.linspace(-1, 1, 5)
targets2 = np.linspace(-1, 1, 500)

y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_2 = []
for k in targets2:
    gr_2.append(lagrange_basis(func, k))
```

In [90]:
```
x_pt = np.linspace(-1, 1, 9)
targets3 = np.linspace(-1, 1, 500)

y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_3 = []
for k in targets3:
    gr_3.append(lagrange_basis(func, k))
```

In [91]:
```
x_pt = np.linspace(-1, 1, 17)
targets4 = np.linspace(-1, 1, 500)

y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_4 = []
for k in targets4:
    gr_4.append(lagrange_basis(func, k))
```

In [ ]:

In [92]:
```
n = 3
i = np.arange(1, n+1)
cheby1 = []
# i values plugged into chebyshev function
for j in i:
    cheby1.append(np.cos(np.pi*((2*j-1)/(2*n))))

# making x_pt have chebyshev values
x_pt = cheby1
targets5 = np.linspace(-1, 1, 500)

# plug chebyshev values into our function
y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

# using class to plug into function
func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

# putting data points through function
# saving to graphing number
gr_5 = []
for k in targets5:
    gr_5.append(lagrange_basis(func, k))
```

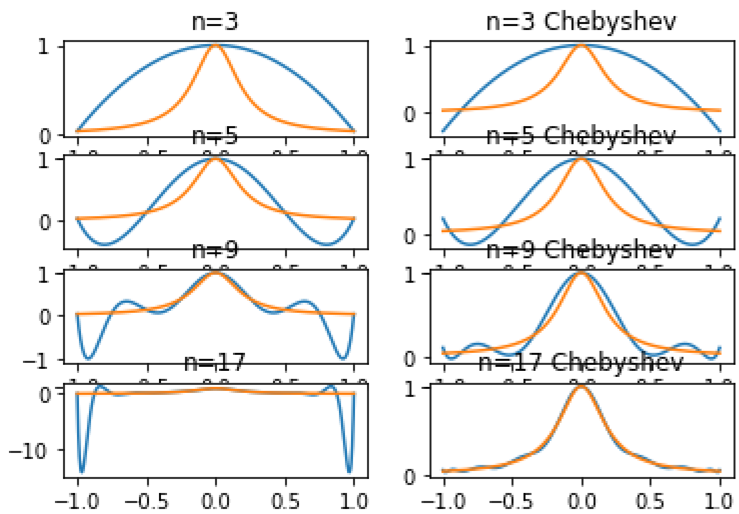
In [93]:
```
# same process as above repeated for all n values
n = 5
i = np.arange(1, n+1)
cheby2 = []
for j in i:
    cheby2.append(np.cos(np.pi*((2*j-1)/(2*n))))

x_pt = cheby2
targets6 = np.linspace(-1, 1, 500)

y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_6 = []
for k in targets6:
    gr_6.append(lagrange_basis(func, k))
```

In [94]:
```
n = 9
i = np.arange(1, n+1)
cheby3 = []
for j in i:
    cheby3.append(np.cos(np.pi*((2*j-1)/(2*n))))

x_pt = cheby3
targets7 = np.linspace(-1, 1, 500)

y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_7 = []
for k in targets7:
    gr_7.append(lagrange_basis(func, k))
```

In [95]:
```
n = 17
i = np.arange(1, n+1)
cheby4 = []
for j in i:
    cheby4.append(np.cos(np.pi*((2*j-1)/(2*n))))

x_pt = cheby4
targets8 = np.linspace(-1, 1, 500)

y_pt = []
for i in x_pt:
    y_pt.append(1/(1+(25*(i**2))))

func = []
for j in range(len(x_pt)):
    func.append(Data(x_pt[j],y_pt[j]))

gr_8 = []
for k in targets8:
    gr_8.append(lagrange_basis(func, k))
```

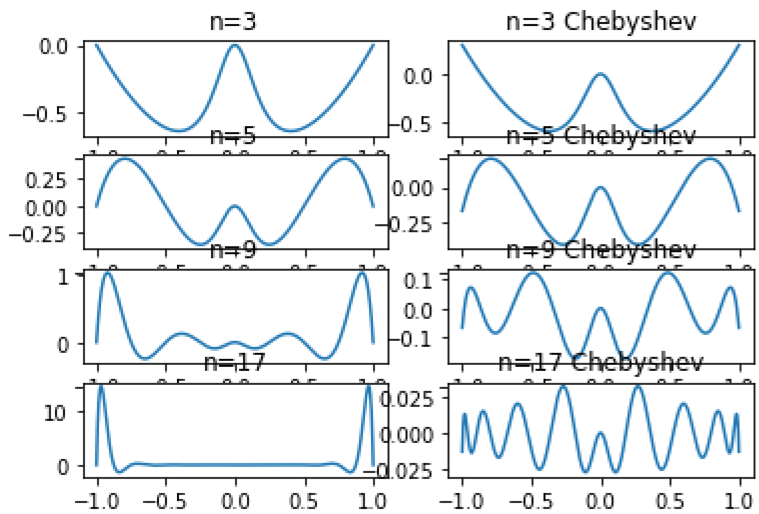
In [96]:
```
# creating the 500 points on real function to compare
x = np.linspace(-1, 1, 500)
y = []
for i in x:
    y.append(1/(1+(25*(i**2))))

# creating subplot and setting values/ titles
fig, axs = plt.subplots(4,2)
axs[0,0].plot(targets1, gr_1)
axs[0,0].plot(x, y)
axs[0, 0].set_title('n=3')

axs[1,0].plot(targets2, gr_2)
axs[1,0].plot(x, y)
axs[1, 0].set_title('n=5')

axs[2,0].plot(targets3, gr_3)
axs[2,0].plot(x, y)
axs[2, 0].set_title('n=9')

axs[3,0].plot(targets4, gr_4)
axs[3,0].plot(x, y)
axs[3, 0].set_title('n=17')

axs[0,1].plot(targets5, gr_5)
axs[0,1].plot(x, y)
axs[0, 1].set_title('n=3 Chebyshev')

axs[1,1].plot(targets6, gr_6)
axs[1,1].plot(x, y)
axs[1, 1].set_title('n=5 Chebyshev')

axs[2,1].plot(targets7, gr_7)
axs[2,1].plot(x, y)
axs[2, 1].set_title('n=9 Chebyshev')

axs[3,1].plot(targets8, gr_8)
axs[3,1].plot(x, y)
axs[3, 1].set_title('n=17 Chebyshev')
```

Out[96]: `Text(0.5, 1.0, 'n=17 Chebyshev')`



In [107]:
```
# setting up graphs to illustrate difference between the
# true function and our estimated function
x = np.linspace(-1, 1, 500)
y = []
for i in x:
    y.append(1/(1+(25*(i**2))))

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_1[i])
fig, axs = plt.subplots(4,2)
axs[0,0].plot(x, newy)
axs[0, 0].set_title('n=3')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_2[i])
axs[1,0].plot(x, newy)
axs[1, 0].set_title('n=5')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_3[i])
axs[2,0].plot(x, newy)
axs[2, 0].set_title('n=9')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_4[i])
axs[3,0].plot(x, newy)
axs[3, 0].set_title('n=17')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_5[i])
axs[0,1].plot(x, newy)
axs[0, 1].set_title('n=3 Chebyshev')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_6[i])
axs[1,1].plot(x, newy)
axs[1, 1].set_title('n=5 Chebyshev')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_7[i])
axs[2,1].plot(x, newy)
axs[2, 1].set_title('n=9 Chebyshev')

newy = []
for i in range(len(y)):
    newy.append(y[i]-gr_8[i])
axs[3,1].plot(x, newy)
axs[3, 1].set_title('n=17 Chebyshev')
```

Out[107]: Text(0.5, 1.0, 'n=17 Chebyshev')

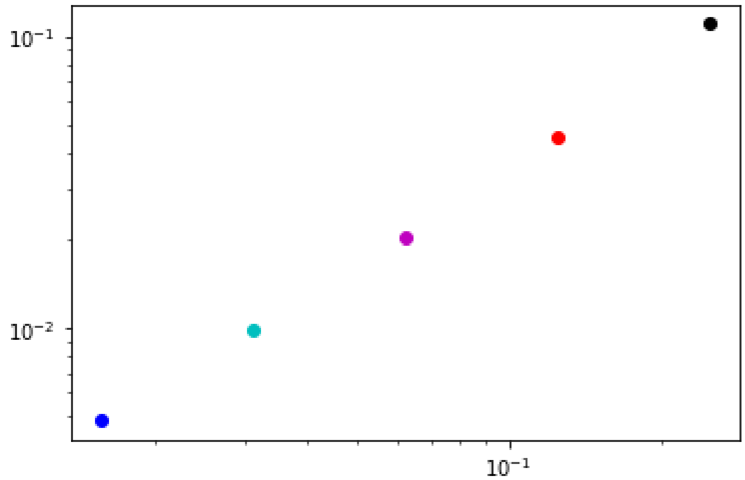


In [ ]:

In [111]:
```
def odeEuler(f,y0,initTime,finalT,tStep):
    # creating the t array
    t = np.linspace(initTime,finalT,tStep)
    # initializing y
    y = np.zeros(len(t))
    # adding first y value
    y[0] = y0
    #looping through values to find next y value
    for n in range(0,len(t)-1):
        y[n+1] = y[n] + f(y[n],t[n])*(t[n+1] - t[n])
    return (t, y)
```

In [148]:
```
# defining function
f = lambda y,t: -t*np.exp((-t**2)/2)
tStep1 = 1/4
# plugging into euler function
t, y = odeEuler(f,1,0, 1, int(1/(tStep1)))
# y values for true function
y_true = np.exp((-t**2)/2)
difference1 = abs(y[3]-y_true[3])
# plotting on loglog scale
plt.loglog(tStep1,difference1,'ko-')

#repeated for all timesteps
tStep2 = 1/8
t, y = odeEuler(f,1,0, 1, int(1/(tStep2)))
y_true = np.exp((-t**2)/2)
difference2 = abs(y[7]-y_true[7])
plt.loglog(tStep2,difference2,'ro-')

tStep3 = 1/16
t, y = odeEuler(f,1,0, 1, int(1/(tStep3)))
y_true = np.exp((-t**2)/2)
difference3 = abs(y[15]-y_true[15])
plt.loglog(tStep3,difference3,'mo-')

tStep4 = 1/32
t, y = odeEuler(f,1,0, 1, int(1/(tStep4)))
y_true = np.exp((-t**2)/2)
difference4 = abs(y[31]-y_true[31])
plt.loglog(tStep4,difference4,'co-')

tStep5 = 1/64
t, y = odeEuler(f,1,0, 1, int(1/(tStep5)))
y_true = np.exp((-t**2)/2)
difference5 = abs(y[63]-y_true[63])
plt.loglog(tStep5,difference5,'bo-')

rate = (difference5 - difference4)/(tStep5 - tStep4)
print("rate =", rate)

plt.show()
```

```
rate = 0.32222350857096416
```



In [137]:

```python
def Backward_Euler(y0, t0, finalTime, timeStep, F, Fdy):
    # initializing t using inverse timestep
    t = np.linspace(t0, finalTime, int(1/timeStep))
    # initializing y
    Y = np.zeros(len(t))
    #setting initial y
    Y[0] = y0
    # plugging into backward euler step function
    for i in range(1,len(t)):
        Y[i] = Backward_Euler_Step(Y[i-1],t[i], timeStep, F, Fdy)

    return t, Y

def Backward_Euler_Step(Yn, tNext, dt, F, Fdy):
    # setting conditions for looping
    MaxIter = 1000
    tol = 1e-6

    # functions defined
    G = lambda y: y-Yn-dt*F(y,tNext)
    Gdy = lambda y: 1-dt*Fdy(y, tNext)

    # finding next y value by plugging into newton's method
    YNext = Newtons_Method(MaxIter, tol, G, Gdy, Yn)
    return YNext
```

In [147]:
```
# defining functions again
f = lambda y,t: -t*np.exp((-t**2)/2)
fdy = lambda y,t: -t*np.exp((-t**2)/2)

# defining time step
timeStep1 = 1/4
# creating two return values and plugging vals into backward euler
t, y = Backward_Euler(1, 0, 1, timeStep1, f, fdy)
# y values of true function
y_true = np.exp((-t**2)/2)
diff1 = abs(y[3]-y_true[3])
# graphing difference between the plots on loglog scale
plt.loglog(timeStep1,diff1,'ko-')

#repeated for all timesteps
timeStep2 = 1/8
t, y = Backward_Euler(1, 0, 1, timeStep2, f, fdy)
y_true = np.exp((-t**2)/2)
diff2 = abs(y[7]-y_true[7])
plt.loglog(timeStep2,diff2,'ro-')

timeStep3 = 1/16
t, y = Backward_Euler(1, 0, 1, timeStep3, f, fdy)
y_true = np.exp((-t**2)/2)
diff3 = abs(y[15]-y_true[15])
plt.loglog(timeStep3,diff3,'mo-')

timeStep4 = 1/32
t, y = Backward_Euler(1, 0, 1, timeStep4, f, fdy)
y_true = np.exp((-t**2)/2)
diff4 = abs(y[31]-y_true[31])
plt.loglog(timeStep4,diff4,'co-')

timeStep5 = 1/64
t, y = Backward_Euler(1, 0, 1, timeStep5, f, fdy)
y_true = np.exp((-t**2)/2)
diff5 = abs(y[63]-y_true[63])
plt.loglog(timeStep5,diff5,'bo-')

rateConverge = (diff5-diff4)/(timeStep5-timeStep4)
print("rate =",rateConverge)

plt.show()
```

```
2 0.9269301382273949 0.0730698617726051
3 0.9215909335279273 0.07840906647207269
4 0.9212007985785615 0.07879920142143848
5 0.9211722914717387 0.07882770852826126
6 0.9211702084613836 0.07882979153861636
2 0.8034274762433488 0.19657252375665124
3 0.7895641252531809 0.21043587474681913
4 0.7879318164299008 0.21206818357009916
5 0.7877396239292243 0.21226037607077575
6 0.7877169946590828 0.21228300534091715
7 0.7877143302269882 0.21228566977301178
2 0.6560467685591211 0.34395323144087886
3 0.6387104217635596 0.36128957823644037
4 0.6364277872527595 0.3635722127472405
5 0.6361272383325435 0.3638727616674565
6 0.6360876657890567 0.36391233421094327
7 0.6360824553687469 0.36391754463125314
2 0.9826311555210006 0.017368844478999446
3 0.9823294787624649 0.017670521237535097
4 0.9823242389857629 0.017675761014237068
2 0.9491748860537745 0.050825113946225464
3 0.948076006453965 0.05192399354603505
4 0.9480395793062811 0.05196042069371887
5 0.9480383717699062 0.05196162823009376
2 0.9014446862046982 0.0985553137953018
3 0.8992737146701487 0.10072628532985128
4 0.8991725611050969 0.10082743889490309
5 0.8991678479876931 0.10083215201230689
2 0.8419690539745087 0.15803094602549128
3 0.838697351937946 0.16130264806205397
4 0.8385102145270842 0.16148978547291581
5 0.8384995104928681 0.16150048950713192
2 0.7737940450963738 0.22620595490362616
3 0.7696072478441969 0.23039275215580313
4 0.7693363391794741 0.23066366082052592
5 0.7693188099082432 0.2306811900917568
6 0.7693176756685901 0.2306823243314099
2 0.7002396736310926 0.29976032636890737
3 0.6954679032656002 0.30453209673439985
4 0.6951382789025702 0.3048617210974298
5 0.6951155091101492 0.3048844908898508
6 0.6951139362183819 0.3048860637816181
2 0.6246406312719629 0.3753593687280371
3 0.619674144561892 0.38032585543810804
4 0.6193241398294608 0.3806758601705392
5 0.6192994738392195 0.38070052616078054
6 0.6192977355453674 0.3807022644546326
2 0.9958597948741648 0.004140205125835239
3 0.9958426535756808 0.004157346424319197
2 0.9876507279977496 0.012349272002250444
3 0.9875836203530751 0.01241637964692488
2 0.9754794431669817 0.024520556833018348
3 0.9753329320616293 0.0246670679383707
4 0.9753311586652504 0.02466884133474956
2 0.9595012886797825 0.04049871132021754
3 0.9592507038960256 0.040749296103974375
4 0.9592467371714785 0.04075326282852154
```

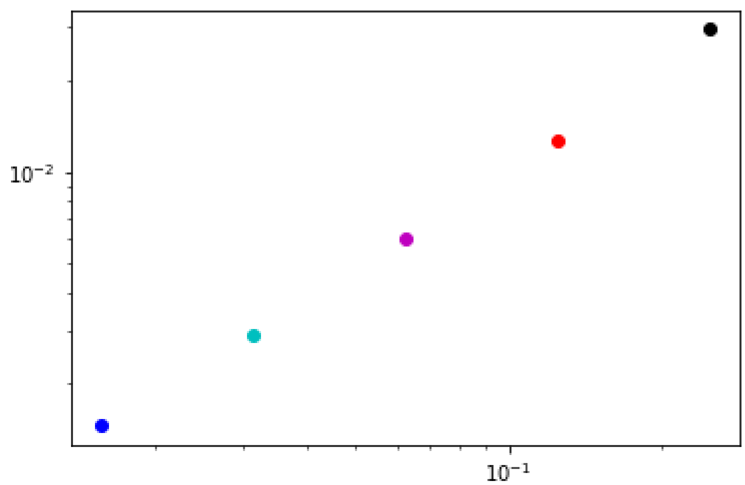
```
2 0.9399201271883256 0.06007987281167437
3 0.9395466093348848 0.060453390665115236
4 0.9395393905010095 0.06046060949899046
2 0.9169820579026542 0.08301794209734581
3 0.9164732246487014 0.0835267753512986
4 0.9164617467277548 0.08353825327224518
2 0.8909709791772105 0.10902902082278954
3 0.8903211999468946 0.10967880005310537
4 0.8903046365755755 0.10969536342442454
2 0.8622028958057862 0.13779710419421376
3 0.8614131879714938 0.1385868120285062
4 0.8613909958066507 0.13860900419334932
2 0.8310196726844209 0.1689803273155791
3 0.830097255416226 0.16990274458377397
4 0.8300692403833202 0.1699307596166798
2 0.7977824018950971 0.20221759810490292
3 0.7967399619555325 0.20326003804446746
4 0.7967063048655701 0.20329369513442985
5 0.7967052181845425 0.20329478181545746
2 0.7628634729618297 0.23713652703817034
3 0.7617182092421106 0.23828179075788936
4 0.7616794515190951 0.2383205484809049
5 0.7616781398901074 0.23832186010989265
2 0.7266427343367265 0.27335726566327345
3 0.725415254694436 0.2745847453055634
4 0.725372249447360 5 0.27462775055263955
5 0.7253707427410883 0.27462925725891174
2 0.68949798540214 0.31050201459786
3 0.688211130683041 0.31178886931695904
4 0.6881649676559722 0.31183503234402776
5 0.6881633116609042 0.31183668833909584
2 0.6517994095531109 0.3482005904468891
3 0.6504770761766058 0.3495229238233942
4 0.6504289909751486 0.3495710090248514
5 0.65042724240959 0.34957275759040995
2 0.6139036197372113 0.3860963802627887
3 0.612569644724097 0.38743035527590297
4 0.6125209231240636 0.3874790768759364
5 0.612519143634728 0.38748085636527196
2 0.9989934739501354 0.00100652604986462
3 0.9989924608554464 0.0010075391445536486
2 0.9969845631228655 0.0030154368771344586
3 0.996980531469561 0.0030194685304389957
2 0.9939794992767037 0.006020500723296296
3 0.9939704930824812 0.006029506917518801
2 0.9899875898257195 0.01001241017428045
3 0.9899717263073667 0.01002827369263326
2 0.985021170031748 0.01497882996825195
3 0.98499666202431 0.015003337975689979
2 0.9790955397479815 0.02090446025201853
3 0.9790607165038614 0.020939283496138605
2 0.9722288851678823 0.027771114832117694
3 0.972182211248479 0.027817788751520967
2 0.9644421861401539 0.03555781385984613
3 0.9643822781514764 0.03561772184852363
2 0.9557591097119279 0.0442408902880721
3 0.9556847506779911 0.04431524932200892
```

```
2 0.9462058906574433 0.053794109342556684
3 0.9461160418701542 0.05388395812984581
2 0.9358111998374387 0.06418880016256134
3 0.9357050100681195 0.06429498993188054
4 0.9357039157993212 0.06429608420067878
2 0.9246049070456929 0.07539509295430713
3 0.9244817190503797 0.07551828094962032
4 0.9244803517857414 0.0755196482142586
2 0.9126209376875226 0.08737906231247738
3 0.9124802919847695 0.08751970801523046
4 0.9124786240091395 0.08752137599086052
2 0.8998943459294066 0.10010565407059335
3 0.8997359818746186 0.10026401812538144
4 0.8997339889773153 0.10026601102268473
2 0.8864619967942596 0.11353800320574037
3 0.8862858510177525 0.11371414898224752
4 0.8862835132123836 0.11371648678761637
2 0.8723623895616263 0.12763761043837374
3 0.8721685918779266 0.1278314081220734
4 0.8721658939964085 0.12783410600359146
2 0.8576354753069549 0.14236452469304506
3 0.8574243422396641 0.14257565776033587
4 0.8574212743877972 0.14257872561220275
2 0.8423224697770255 0.15767753022297448
3 0.8420944958763512 0.1579055041236488
4 0.8420910537429686 0.15790894625703145
2 0.826465662815602 0.17353433718439804
3 0.8262215099739689 0.17377849002603107
4 0.8262176949903725 0.17378230500962755
2 0.8101082255603339 0.1898917744396661
3 0.8098487105550165 0.1901512894449835
4 0.8098445299059718 0.19015547009402822
2 0.7932940166288674 0.20670598337113255
3 0.7930200971391318 0.2069799028608682
4 0.79301556363098 0.20698443636901998
2 0.7760673884992234 0.2239326115007766
3 0.7757801478589268 0.2242198521410732
4 0.7757752796542501 0.22422472034574992
2 0.7584729952669766 0.24152700473302335
3 0.7581736262219585 0.2418263737780415
4 0.7581684464536049 0.24183155354639507
2 0.7405556029299912 0.25944439707000877
3 0.740245390673004 0.259754609326996
4 0.7402399269530625 0.2597600730469375
2 0.7223599033108663 0.2776400966891337
3 0.7220402080654208 0.2779597919345792
4 0.7220344919068739 0.27796550809312615
2 0.7039303326783273 0.2960696673216727
3 0.7036025720969546 0.29639742790304535
4 0.7035966382672006 0.29640336173279935
2 0.6853108960721862 0.3146891039278138
3 0.6849765277045637 0.31502347229543626
4 0.6849704135307952 0.31502958646920476
2 0.6665449982728301 0.3334550017271699
3 0.6662055023454017 0.33379449765459834
4 0.6661992469919604 0.3338007530080396
2 0.647675282286226 0.35232471771377405
```

```
3 0.6473321450178067 0.35266785498219333
4 0.6473257887551572 0.3526742112448428
2 0.6287434761399275 0.37125652386007246
3 0.6283981737977975 0.3716018262022025
4 0.6283917572817292 0.37160824271827075
2 0.6097902487053495 0.3902097512946505
3 0.6094442325840324 0.39055576741596765
4 0.6094377961621842 0.3905622038378158
2 0.9997520768502731 0.0002479231497268808
2 0.9992565406721305 0.0007434593278694557
2 0.998513888109173 0.0014861118908270354
2 0.9975248014882089 0.0024751985117911213
2 0.9962901477495625 0.0037098522504375175
3 0.9962886233797081 0.003711376620291862
2 0.9948094527795175 0.005190547220482511
3 0.9948072648338531 0.005192735166146933
2 0.9930848094213427 0.006915190578657304
3 0.9930818425686946 0.006918157431305416
2 0.9911175144417796 0.008882485558220354
3 0.9911136558567895 0.008886344143210545
2 0.9889090451997302 0.011090954800269781
3 0.988904184891581 0.01109581510841895
2 0.9864610577987577 0.013538942201242299
3 0.9864550889287661 0.0135449110712339
2 0.9837753850189827 0.01622461498101735
3 0.9837682042059385 0.016231795794061532
2 0.980854034031991 0.019145965968008993
3 0.9808455416441882 0.019154458355811754
2 0.9776991839027482 0.022300816097251763
3 0.9776892843357111 0.022310715664288883
2 0.974313182882872 0.025686817117127947
3 0.9743017848218523 0.025698215178147743
2 0.9706985454999767 0.029301454500023305
3 0.9706855621663659 0.029314437833634077
2 0.966857949448141 0.03314205055185904
3 0.9668432988290203 0.033156701170979686
2 0.9627942322848883 0.03720576771511175
3 0.9627778373450094 0.037222162654990565
2 0.9585103879403817 0.04148961205961832
3 0.9584921768159606 0.04150782318403945
2 0.9540095630448518 0.04599043695514815
3 0.9539894692186309 0.046010530781369074
2 0.9492950530805632 0.05070494691943683
3 0.9492730155376857 0.05072698446231427
2 0.9443702983649055 0.05562970163509451
3 0.9443462617292292 0.0556537382707708
2 0.9392388798714677 0.0607611201285323
3 0.9392127945220268 0.06078720547797323
2 0.9339045148961929 0.06609548510380714
3 0.9338763370636068 0.06612366293639316
2 0.9283710525759551 0.07162894742404491
3 0.9283407444186651 0.0716592555813349
2 0.9226424692671151 0.07735753073288487
3 0.9226099989274124 0.07739000107258764
2 0.9167228637918121 0.08327713620818789
3 0.9166882054317073 0.08331179456829274
2 0.9106164525599364 0.0893835474400636
```

```
3 0.9105795863770005 0.08942041362299946
2 0.9043275645748966 0.0956724354251034
3 0.9042884767982826 0.09571152320171739
2 0.8978606363314428 0.10213936366855725
3 0.897819319198375 0.10218068080162446
2 0.891220206613943 0.10877979338605626
3 0.8911766583270417 0.10882334167295826
2 0.8844109112036314 0.11558908879636864
3 0.8843651358694934 0.11563486413050661
2 0.87743747750342 0.12256252249657995
3 0.877389485052983 0.12261051494701702
2 0.8703047190889938 0.12969528091100624
3 0.8702545251802292 0.12974547481977083
2 0.8630175301949109 0.1369824698050891
3 0.8629651560984934 0.13703484390150655
2 0.8555808801445245 0.1444191198554755
3 0.8555263526131601 0.14447364738683988
2 0.8479998077325365 0.15200019226746353
3 0.8479431588546965 0.1520568411453035
2 0.8402794155690211 0.15972058443097892
3 0.8402206826078724 0.15977931739212758
2 0.8324248643937329 0.16757513560626713
3 0.832364089612105 0.16763591038789505
2 0.8244413673694968 0.1755586326305032
3 0.8243785978417633 0.17562140215823674
2 0.8163341843634285 0.18366581563657147
3 0.8162694717752181 0.18373052822478186
2 0.8081086162246738 0.19189138377532622
3 0.8080420166613569 0.1919579833386431
2 0.7997699990672773 0.2002300009327227
3 0.7997015727922006 0.20029842720779945
2 0.791323698566699 0.20867630143330096
3 0.7912535097901606 0.20874649020983937
2 0.7827751042783826 0.2172248957216174
3 0.7827032209183604 0.21729677908163958
2 0.7741296239866594 0.2258703760133406
3 0.7740561174223122 0.22594388257768783
2 0.7653926780921261 0.23460732190787392
3 0.7653176229110983 0.23468237708890172
2 0.7565696940454825 0.24343030595451753
3 0.7564931677860446 0.24350683221395542
2 0.747666100835645 0.252333899164355
3 0.7475881837246982 0.25241181627530185
2 0.7386873235397682 0.2613126764602318
3 0.7386080982277365 0.26139190177226346
2 0.7296387779426108 0.2703612220573892
3 0.7295583292362336 0.27044167076376635
2 0.7205258652324751 0.2794741347675249
3 0.720444279826496 0.27955572017350405
2 0.7113539667807275 0.2886460332192725
3 0.7112713329894574 0.28872866701054256
2 0.7021284390116761 0.2978715609883239
3 0.7020448465013871 0.2979551534986129
2 0.6928546083693407 0.30714539163065935
3 0.692770147892417 0.307229852107583
2 0.6835377663873975 0.3164622336126025
3 0.6834525295191433 0.31654747048085674
```

```
2 0.6741831648683194 0.3258168351316806
3 0.6740972437472894 0.3259027562527106
2 0.6647960111774649 0.3352039888225351
3 0.664709498250147 0.335290501749853
2 0.6553814636575894 0.3446185363424106
3 0.6552944514282295 0.3447055485717705
2 0.6459446271689683 0.3540553728310317
3 0.6458572079552892 0.3541427920447108
2 0.6364905487600323 0.3635094512399677
3 0.6364028144555521 0.36359718554444787
2 0.6270242134731167 0.37297578652688335
3 0.6269362553167289 0.3730637446832711
2 0.6175505402896257 0.38244945971037425
3 0.6174624486430558 0.3825375513569442
2 0.6080743782186079 0.3919256217813921
3 0.6079862423523135 0.39201375764768653
rate = 0.0928994438317261
```



In [ ]: