Question 1

```
import numpy as np

def q1():
    mat = np.array([[1,2],
                    [3,4]])
    a = np.array([2,3])
    a = a/np.linalg.norm(a)
    for i in range(100):
        a = mat@a
        a = a/np.linalg.norm(a)
    print("Multiplied Vector:",a)
q1()
```

```
Multiplied Vector: [0.41597356 0.90937671]
```

Question 2

```
def q2():
    mat = np.array([[1,2],
                    [3,4]])
    print("Eigen Vector:",np.linalg.eig(mat)[1][:,1])
q2()
```

```
Eigen Vector: [-0.41597356 -0.90937671]
```

We see that the eigenvector is here has the same value as the normalized version of the result we obtain in Q1(direction is opposite but along the same line)

Question 3

```
def q3():
    a = np.matrix([
        [0,0,0,0.5,0.5,1,1,1],
        [0.5,0,0,0,0,0,0,0],
        [0.5,0,0,0,0,0,0,0],
        [0,0.5,0,0,0,0,0,0],
        [0,0.5,0,0,0,0,0,0],
        [0,0,0.5,0,0,0,0,0],
        [0,0,0.5,0,0,0,0,0],
        [0,0,0,0.5,0.5,0,0,0]
    ])

    for i in range(100):
        a = a@a
    print("Converged Vector:",a[:,0])
q3()
```

```
Converged Vector: [[0.30769231]
 [0.15384615]
 [0.15384615]
 [0.07692308]
```

```
 [0.07692308]
 [0.07692308]
 [0.07692308]
 [0.07692308]]
```

```
a = np.matrix([
        [0,0,0,0.5,0.5,1,1,1],
        [0.5,0,0,0,0,0,0,0],
        [0.5,0,0,0,0,0,0,0],
        [0,0.5,0,0,0,0,0,0],
        [0,0.5,0,0,0,0,0,0],
        [0,0,0.5,0,0,0,0,0],
        [0,0,0.5,0,0,0,0,0],
        [0,0,0,0.5,0.5,0,0,0]
    ])
print("Eigen Vector:",(np.linalg.eig(a)[1][:,0]/2.5).real)
```

```
Eigen Vector: [[0.29711254]
 [0.14855627]
 [0.14855627]
 [0.07427814]
 [0.07427814]
 [0.07427814]
 [0.07427814]
 [0.07427814]]
```

We see that on multiplying the matrix with itself multiple times it converges to (a scaled version of) principal eigenvector.

Question 4 - Random Walk

```
import networkx
import random
import matplotlib.pyplot as plt
```

```
d = {'A':['B','C'],
     'B':['D','E'],
     'C':['F','G'],
     'D':['A','H'],
     'E':['A','H'],
     'F':['A'],
     'G':['A'],
     'H':['A']}

node = random.choice(list(d.keys()))
final_ls = {k:0 for k in d.keys()}

for j in range(10000):
     node = random.choice(d[node])
     final_ls[node]+=1

print(final_ls)
```

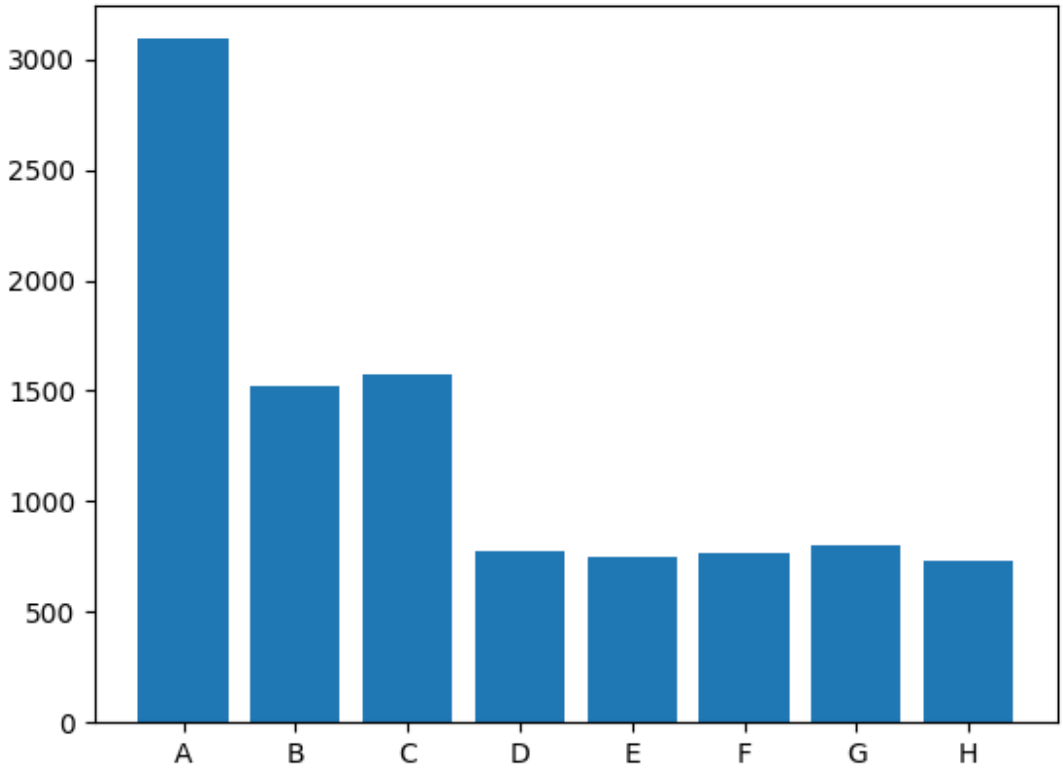
```
print("Visit distribution
Ratio:",np.array(list(final_ls.values()))/sum(final_ls.values()))
```

```
{'A': 3081, 'B': 1521, 'C': 1560, 'D': 772, 'E': 749, 'F': 772, 'G':
788, 'H': 757}
Visit distribution Ratio: [0.3081 0.1521 0.156  0.0772 0.0749 0.0772
0.0788 0.0757]
```

Distribution of visits is the same as principal eigenvector obtained using the convergence method used in the previous question.

```
plt.bar(final_ls.keys(), final_ls.values())
```

```
<BarContainer object of 8 artists>
```




```
plt.bar(final_ls.keys(), final_ls.values())
```

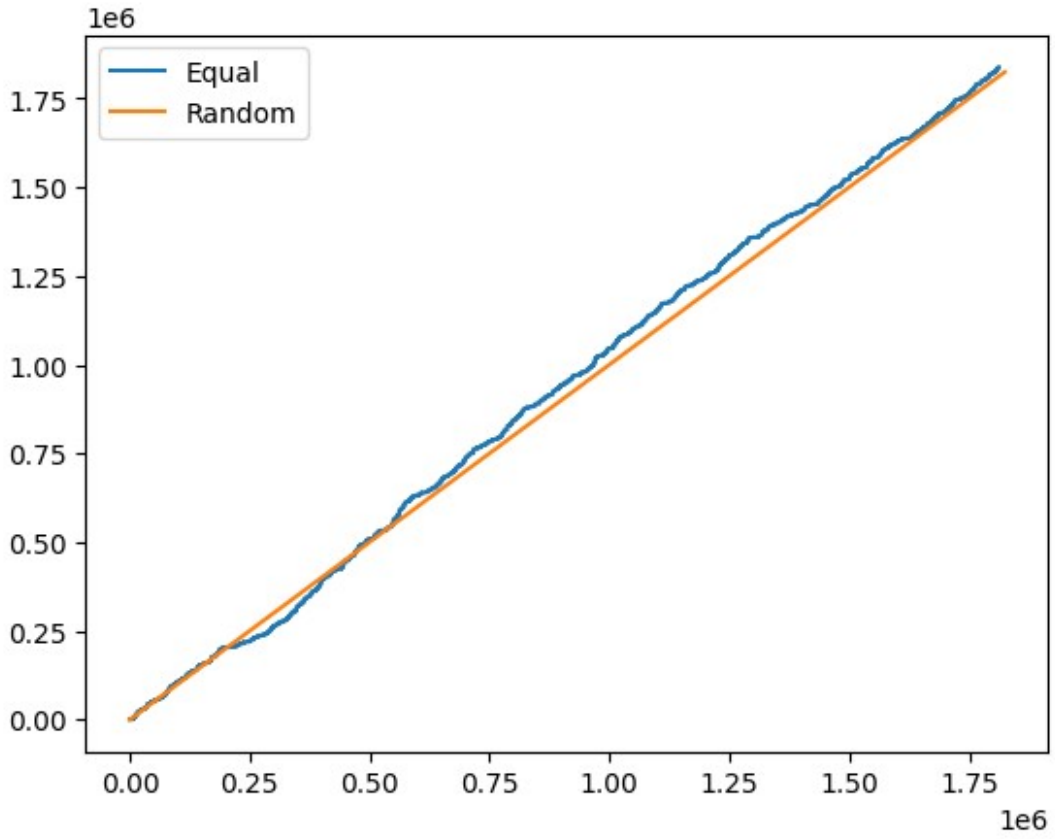
Question 5

```
a = []
x = []
y = []
for i in range(3650):
    a.append(random.choice(['h','t']))
    x.append(a.count('h')*1000)
    y.append(a.count('t')*1000)
print("Head count:", a.count('h'))
print("Tail count:", a.count('t'))
```

```
Head count: 1812
Tail count: 1838
```

```
plt.plot(x,y, label='Equal')
plt.plot([500*x for x in range(3650)],
         [500*x for x in range(3650)],label='Random')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x24541b3d8d0>
```



These two statements essesntially represent the idea of 'Equal Distribution' and 'Random Walk' which is the same for large values. In the long run the number of heads and number of tails for an unbiased coin is likely to remain the same provided the coin is unbiased as shown in the code above. Hence, by the end of the year both daughters are likely to receive, more or less the same amount of money, irrespective of the method we use.

Question 6

```
adj = np.matrix([
        [0,0,0,1,1,1,1,1],
        [1,0,0,0,0,0,0,0],
        [1,0,0,0,0,0,0,0],
        [0,1,0,0,0,0,0,0],
        [0,1,0,0,0,0,0,0],
        [0,0,1,0,0,0,0,0],
```

```
        [0,0,1,0,0,0,0,0],
        [0,0,0,1,1,0,0,0]
    ])                                                  # Adjacency
Matrix

adj = adj.T
np.linalg.eig(adj)
```

```
(array([ 1.72775350e+00+0.j        , -6.20853041e-01+1.41285018j,
        -6.20853041e-01-1.41285018j, -4.86047418e-01+0.j        ,
         9.02999445e-09+0.j        , -9.02999445e-09+0.j        ,
        -2.49044648e-17+0.j        ,  3.23415489e-17+0.j        ]),
 matrix([[ 6.63834069e-01+0.j        , -5.93276700e-01+0.j        ,
           -5.93276700e-01-0.j        ,  5.09966481e-02+0.j        ,
           -1.25204240e-16+0.j        , -1.33097250e-16+0.j        ,
           -9.39208632e-17+0.j        , -1.27427145e-16+0.j        ],
          [ 3.84218043e-01+0.j        ,  1.54659475e-01+0.35195232j,
            1.54659475e-01-0.35195232j, -1.04921138e-01+0.j        ,
            1.38845105e-16+0.j        ,  2.74725694e-16+0.j        ,
            1.00101035e-16+0.j        ,  1.89093010e-16+0.j        ],
          [ 3.84218043e-01+0.j        ,  1.54659475e-01+0.35195232j,
            1.54659475e-01-0.35195232j, -1.04921138e-01+0.j        ,
            3.68647997e-09+0.j        , -3.68647951e-09+0.j        ,
            6.00606208e-17+0.j        ,  4.72732524e-17+0.j        ],
          [ 2.22380127e-01+0.j        ,  1.68472621e-01-0.18349873j,
            1.68472621e-01+0.18349873j,  2.15866053e-01+0.j        ,
           -6.29321457e-09+0.j        ,  6.29321472e-09+0.j        ,
           -2.48556325e-01+0.j        , -4.66314405e-01+0.j        ],
          [ 2.22380127e-01+0.j        ,  1.68472621e-01-0.18349873j,
            1.68472621e-01+0.18349873j,  2.15866053e-01+0.j        ,
           -1.07974474e-09+0.j        ,  1.07974509e-09+0.j        ,
            2.48556325e-01+0.j        ,  4.66314405e-01+0.j        ],
          [ 2.22380127e-01+0.j        ,  1.68472621e-01-0.18349873j,
            1.68472621e-01+0.18349873j,  2.15866053e-01+0.j        ,
            4.08248293e-01+0.j        ,  4.08248288e-01+0.j        ,
           -2.32093482e-01+0.j        ,  4.45986147e-01+0.j        ],
          [ 2.22380127e-01+0.j        ,  1.68472621e-01-0.18349873j,
            1.68472621e-01+0.18349873j,  2.15866053e-01+0.j        ,
            4.08248293e-01+0.j        ,  4.08248288e-01+0.j        ,
            7.46775855e-01+0.j        ,  1.42209806e-01+0.j        ],
          [ 2.57421127e-01+0.j        , -3.05552842e-01-0.10421617j,
           -3.05552842e-01+0.10421617j, -8.88251001e-01+0.j        ,
           -8.16496578e-01+0.j        , -8.16496583e-01+0.j        ,
           -5.14682373e-01+0.j        , -5.88195953e-01+0.j        ]]))
```

Question 7

For a network of vertices in the order of thousands, it is extremely inefficient to calculate the eigenvectors through matrix multiplication convergence as it is of the order $O\left(n^2\right)$.

Random walk with teleportation is much more effecient since any random walk for n-steps takes only $O(n)$ time.

Question 8

```
def calculate_intersection(s1,y1,s2,y2):
```