1(a)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(24787)
a = []
a = np.random.randint(0,8,(3,4,4))
print(a)
row = []
column = []
locations = []
locations = np.argwhere(a == 4)
l0 = locations[0, 1:]
l1 = locations[1,1:]
l2 = locations[2,1:]
l3 = locations[3,1:]
l4 = locations[4,1:]
l5 = locations[5,1:]
l6 = locations[6,1:]

row.append(l0[0])
row.append(l1[0])
row.append(l2[0])
row.append(l3[0])
row.append(l4[0])
row.append(l5[0])
row.append(l6[0])
print("Row:",row)

column.append(l0[1])
column.append(l1[1])
column.append(l2[1])
column.append(l3[1])
column.append(l4[1])
column.append(l5[1])
column.append(l6[1])
print("Column:",column)
```

```
[[[2 6 4 1]
  [0 4 4 3]
  [6 6 1 2]
  [7 0 6 5]]

 [[1 3 3 7]
  [4 7 2 5]
  [0 4 6 7]
  [5 5 7 1]]

 [[7 2 4 5]
  [6 7 7 0]
  [6 2 0 4]
  [2 0 7 6]]]
Row: [0, 1, 1, 1, 2, 0, 2]
Column: [2, 1, 2, 0, 1, 2, 3]
```

1(b)

In [2]:
```
b =  np.tile(a,(2,2))
print(b)
```

```
[[[2 6 4 1 2 6 4 1]
  [0 4 4 3 0 4 4 3]
  [6 6 1 2 6 6 1 2]
  [7 0 6 5 7 0 6 5]
  [2 6 4 1 2 6 4 1]
  [0 4 4 3 0 4 4 3]
  [6 6 1 2 6 6 1 2]
  [7 0 6 5 7 0 6 5]]

 [[1 3 3 7 1 3 3 7]
  [4 7 2 5 4 7 2 5]
  [0 4 6 7 0 4 6 7]
  [5 5 7 1 5 5 7 1]
  [1 3 3 7 1 3 3 7]
  [4 7 2 5 4 7 2 5]
  [0 4 6 7 0 4 6 7]
  [5 5 7 1 5 5 7 1]]

 [[7 2 4 5 7 2 4 5]
  [6 7 7 0 6 7 7 0]
  [6 2 0 4 6 2 0 4]
  [2 0 7 6 2 0 7 6]]]
```

1(c)

In [3]:
```
c = np.sum(b,0)
print(c)
print(c.shape)
```

```
[[10 11 11 13 10 11 11 13]
 [10 18 13  8 10 18 13  8]
 [12 12  7 13 12 12  7 13]
 [14  5 20 12 14  5 20 12]
 [10 11 11 13 10 11 11 13]
 [10 18 13  8 10 18 13  8]
 [12 12  7 13 12 12  7 13]
 [14  5 20 12 14  5 20 12]]
(8, 8)
```

1(d)

In [4]:
```
np.random.seed(24787)
A = np.random.random((1000,1000))
B = np.random.random((1000,1000))

i = 0
j = 0
l1 = np.zeros((1000,1000))
l2 = np.zeros((1000,1000))
l_ans = np.zeros((1000,1000))
def matmul(l1,l2):
    for i in range(0,len(l1)):
        for j in range(0,len(l2)):
            l_ans[i,j] = np.dot(l1[i,:],l2[:,j])
    return l_ans
```

In [5]:
```
%%time
ans = matmul(A,B)
print(ans)
```

```
[[262.02681889 250.35010895 255.20209698 ... 255.11959541 248.13659427
  246.60845719]
 [267.28382531 248.10413271 253.0417709  ... 255.93196541 253.71831423
  246.48668303]
 [268.78477885 252.60592752 268.07751687 ... 262.21855106 259.70257958
  254.42145539]
 ...
 [256.28928623 241.29542559 253.63761207 ... 255.58972957 248.44848129
  245.45765948]
 [261.245718   248.87670911 259.54001149 ... 257.58727381 252.51065398
  250.76435101]
 [256.55021563 242.88337935 249.70633094 ... 251.16608605 245.52951864
  237.41175273]]
CPU times: user 3.37 s, sys: 5.91 ms, total: 3.38 s
Wall time: 3.38 s
```

In [6]:
```
ans1 = A@B
print(ans-ans1)
```

```
[[ 5.68434189e-14  5.68434189e-14  5.68434189e-14 ... -5.68434189e-14
  -2.84217094e-14  5.68434189e-14]
 [ 5.68434189e-14  5.68434189e-14  5.68434189e-14 ...  2.84217094e-14
  -1.13686838e-13  0.00000000e+00]
 [ 5.68434189e-14  5.68434189e-14 -1.13686838e-13 ...  0.00000000e+00
   5.68434189e-14  0.00000000e+00]
 ...
 [-5.68434189e-14  0.00000000e+00 -2.84217094e-14 ... -5.68434189e-14
   2.84217094e-14  8.52651283e-14]
 [-5.68434189e-14 -2.84217094e-14  5.68434189e-14 ... -1.13686838e-13
   2.84217094e-14  0.00000000e+00]
 [ 1.70530257e-13  0.00000000e+00 -8.52651283e-14 ...  0.00000000e+00
   2.84217094e-14 -2.84217094e-14]]
```

Since the difference between ans and ans1 is essentially a 1000 by 1000 zero matrix, the implementation is correct.

The @ operator is faster than the function I wrote because it does not have loops. Loops makes code implementation slower

In [ ]:
```
```