Here we collects data and make plots:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
size=np.array([4,8,16, 32, 64, 128, 256, 512, 1024, 2048])
```

The data is collected from executions

In [3]:
```
time1=np.array([0.000001,0.000001,0.000004,0.000023,0.000241,0.002454,

rate1=np.array([1.07374e+09,1.07374e+09,2.02116e+09,2.83379e+09,2.1772
                6.14573e+08,3.6274e+08,1.67158e+08])
```

In [4]:
```
time2=np.array([0.000001,0.000001,0.000003,0.000017,0.000268,0.008706,

rate2=np.array([1.07374e+09,1.07374e+09,2.64306e+09,3.87152e+09,1.9564
                1.76696e+08,1.3748e+08,9.86601e+07])
```

In [5]:
```
time3=np.array([0.000001,0.000001,0.000002,0.000019,0.000174,0.002822,

rate3=np.array([1.07374e+09,1.07374e+09,4.29497e+09,3.43597e+09,3.0123
                6.36375e+08,3.65845e+08,1.36422e+08])
```

In [6]:
```
time4=np.array([0.000001,0.000001,0.000002,0.000012,0.000089,0.000671,

rate4=np.array([1.07374e+09,1.07374e+09,4.29497e+09,5.49756e+09,5.8955
                6.79374e+09,7.9305e+09,6.62886e+09])
```

In [7]:
```
time5=np.array([0.000001,0.000001,0.000003,0.000022,0.000315,0.007912,

rate5=np.array([1.07374e+09,1.07374e+09,2.64306e+09,2.9878e+09,1.66467
                1.65814e+08,1.43553e+08,9.472e+07])
```

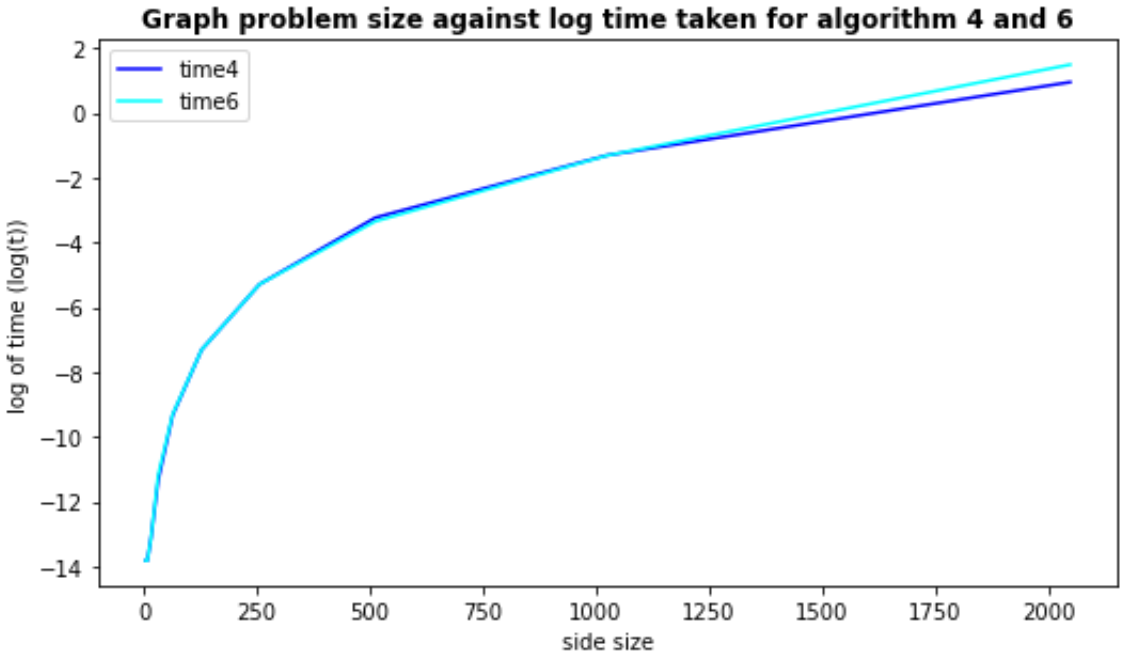
In [8]:
```
time6=np.array([0.000001,0.000001,0.000002,0.000014,0.000092,0.000672,

rate6=np.array([1.07374e+09,1.07374e+09,4.29497e+09,4.65895e+09,5.6969
                7.60419e+09,7.98544e+09,3.86161e+09])
```

Question 1:

From observation of the executions, we find number 4 and 6 are the two fastest algorithm.

In [9]:
```
plt.rcParams["figure.figsize"] = [7.50, 4.5]
plt.rcParams["figure.autolayout"] = True
plt.plot(size,np.log(time4),label='time4',color='blue')
plt.plot(size,np.log(time6),label='time6',color='cyan')
plt.legend(loc='best',prop ={'size': 10})
plt.xlabel('side size')
plt.ylabel('log of time (log(t))')
plt.title('Graph problem size against log time taken for algorithm 4 a
plt.show()
```



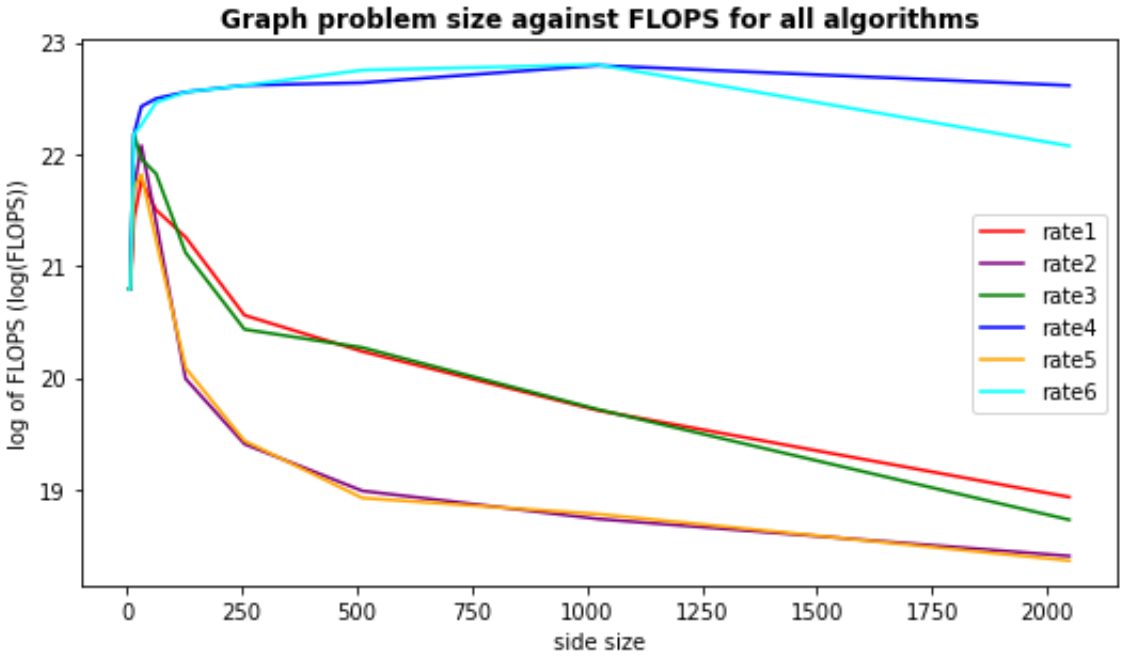
Question 2:

Below is the plots of Graph problem size against log of FLOPS for all algorithms

In [10]:
```
plt.rcParams["figure.figsize"] = [7.50, 4.5]
plt.rcParams["figure.autolayout"] = True


plt.plot(size,np.log(rate1),label='rate1',color='red')
plt.plot(size,np.log(rate2),label='rate2',color='purple')
plt.plot(size,np.log(rate3),label='rate3',color='green')
plt.plot(size,np.log(rate4),label='rate4',color='blue')
plt.plot(size,np.log(rate5),label='rate5',color='orange')
plt.plot(size,np.log(rate6),label='rate6',color='cyan')

plt.legend(loc='best',prop ={'size': 10})
plt.xlabel('side size')
plt.ylabel('log of FLOPS (log(FLOPS))')
plt.title('Graph problem size against FLOPS for all algorithms',fontwe
plt.show()
```



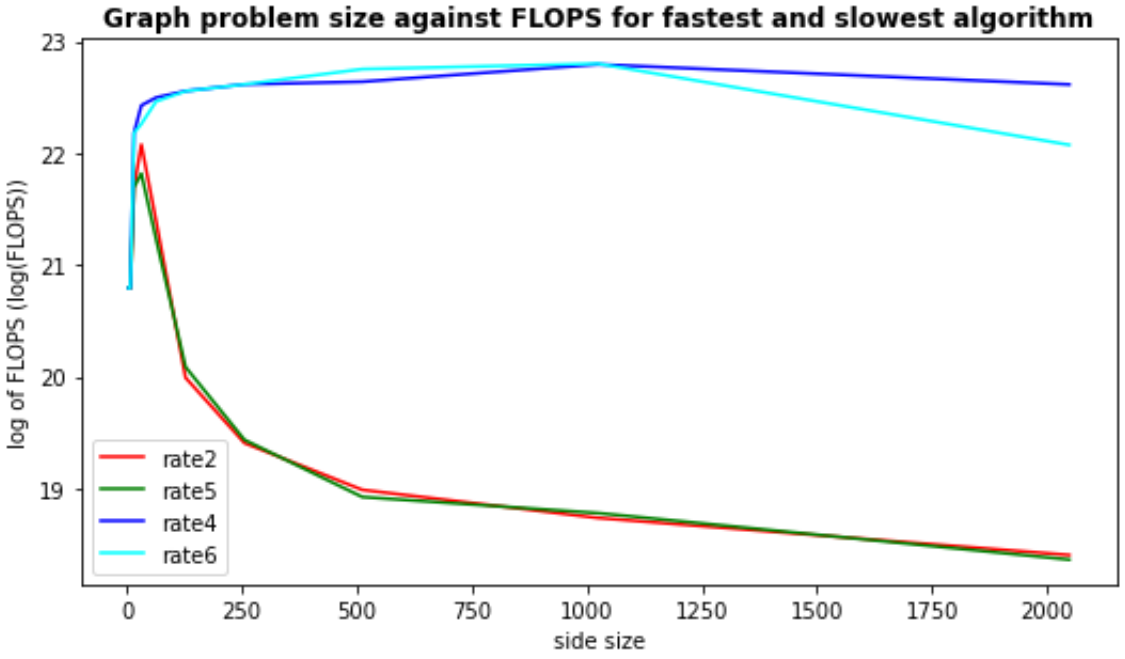
Question 3:

From observation above, we can find fastest algorithm are number 4 and 6.
The slowest are number 1 and 3.
The little difference between 4,6 and 1,3 may be caused when running the program.
Then,The plots are shown below.

In [11]:
```
plt.rcParams["figure.figsize"] = [7.50, 4.5]
plt.rcParams["figure.autolayout"] = True

plt.plot(size,np.log(rate2),label='rate2',color='red')
plt.plot(size,np.log(rate5),label='rate5',color='green')
plt.plot(size,np.log(rate4),label='rate4',color='blue')
plt.plot(size,np.log(rate6),label='rate6',color='cyan')


plt.legend(loc='best',prop ={'size': 10})
plt.xlabel('side size')
plt.ylabel('log of FLOPS (log(FLOPS))')
plt.title('Graph problem size against FLOPS for fastest and slowest al
plt.show()
```



Conclusion

According to the defination of the structure,matrix_t, it stores stores a matrix with all its elements in a single one-dimensional vector in memory, in column major order, that is, with all the elements in a column lined up next to each other in order. Since it stores data in Column major order, to access those data in order, we nesting loops with loop of rows inside and loop of columns outside. For algorithm 4(jki), data in all three matrix, A,B,C are accessed in order. So, algorithm is the fastest. For algorithm 6(kji), data in C and A are accessed in order,only data in B are not accessed in order. Since the parameters for matrix B are k. and j, which are in the outer loop, when the side size of matrix is small, it almost has no difference from algorithm 4. However, as the side size becomes bigger, from the graph above we can see that the FLOPS of 6 is decreasing faster than 4. Since the largest side size is 2048, which is not super large, the performence of algorithm 6 in this observation does not make much difference from 4, so we still consider it to be one of the fastest.The situation for algorithm 2, and 5 are opposite. None of the matrices in 2 are accessed in order, so it is one of the worst. And, only one matrix is accessed in order but its parameters are at outer loop, and the testing examples is not super large,so it still performed as badly as algorith 2 with no obvious difference, and is considered as one of the worst.

Code and other necessary files:

Here is are github link for our code and dependent files:
https://github.com/ZeyuanLu/140lab1/tree/main
(https://github.com/ZeyuanLu/140lab1/tree/main)