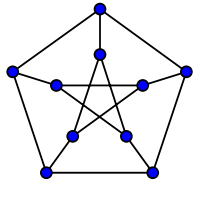
**CS 301 PROJECT REPORT – 3COL**

**3-COL**

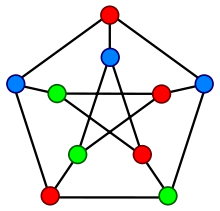
The 3-COL problem is a problem, that works with an input that is an n-node undirected graph G(V,E); with node set V and edge set E, is checking each node of a graph G(V,E) can be assigned exactly one of the colors which are Red, Blue, Green in such a way that no two nodes which are joined by an edge are assigned the same color.

Example:

Question: Can Petersen Graph be colored with 3 colors?

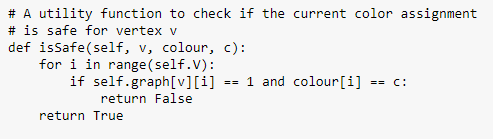


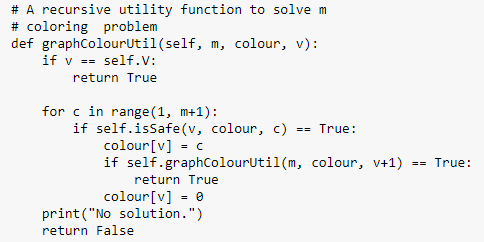
Answer: Yes, each node of the Petersen Graph can be assigned exactly one of the color that are Red, Blue, Green in such a way that no two adjacent vertices have same color. The result graph is in below.

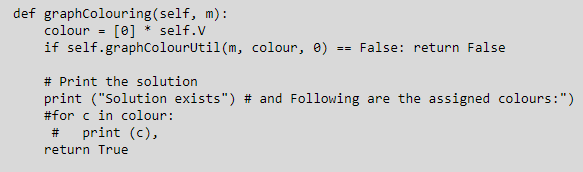


**Algorithm Description: Backtracking**

The aim of the algorithm is to check whether a graph can be 3 colorable or not. The idea of backtracking algorithm is assign colors one by one to vertices by checking the safety of assignment of color. The algorithm working mechanism is that select a vertex as a start vertex, and color it with one color. Then check the adjacent vertex whether it can be colorable or not with a function named as is safe through considering vertices that are already assigned a color. If the vertex is safe the function returns true and assign a color to the safe vertex, if no it returns false and backtrack the previous vertex and do the same thing for all adjacent vertices of it. If all vertices can be colored, the result will be yes, the graph can be 3 colorable, but if there exist not colorable vertices, the algorithm will return the graph can not be 3 colorable as a result.

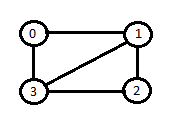




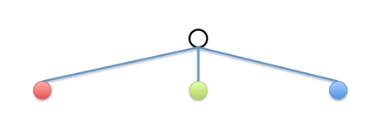
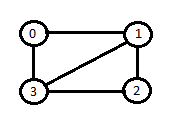


Working mechanism of the algorithm on an example:

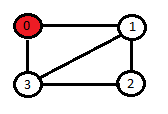
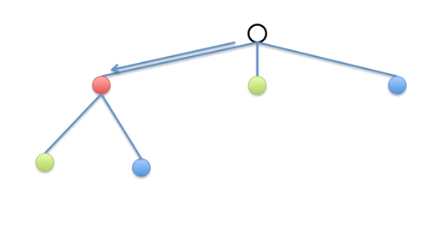
Let the input graph will be like below:



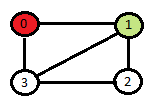
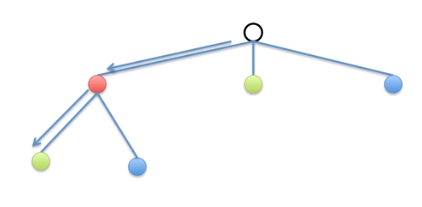
**Step 1:**

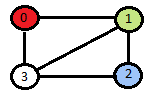
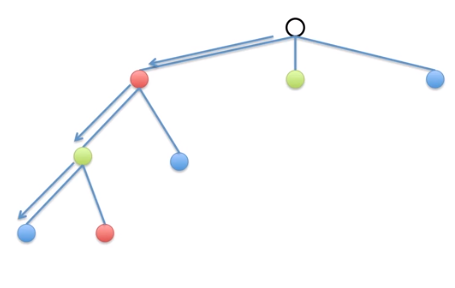
**Step 2:**



**Step 3:**

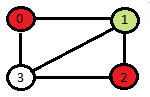
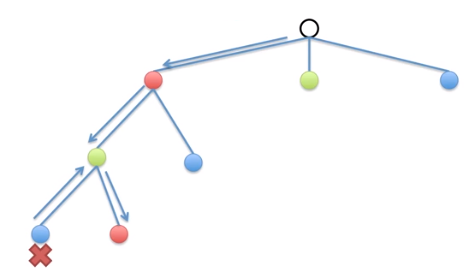


**Step 4:**

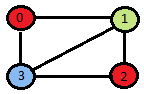
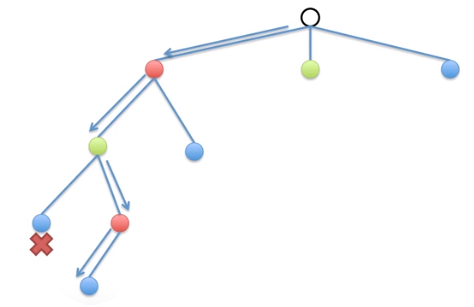


**Step 5:**

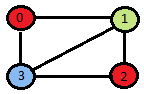
Vertex 3 can not be red, green or blue, therefore backtrack.



**Last Step:**



**Result Graph:**

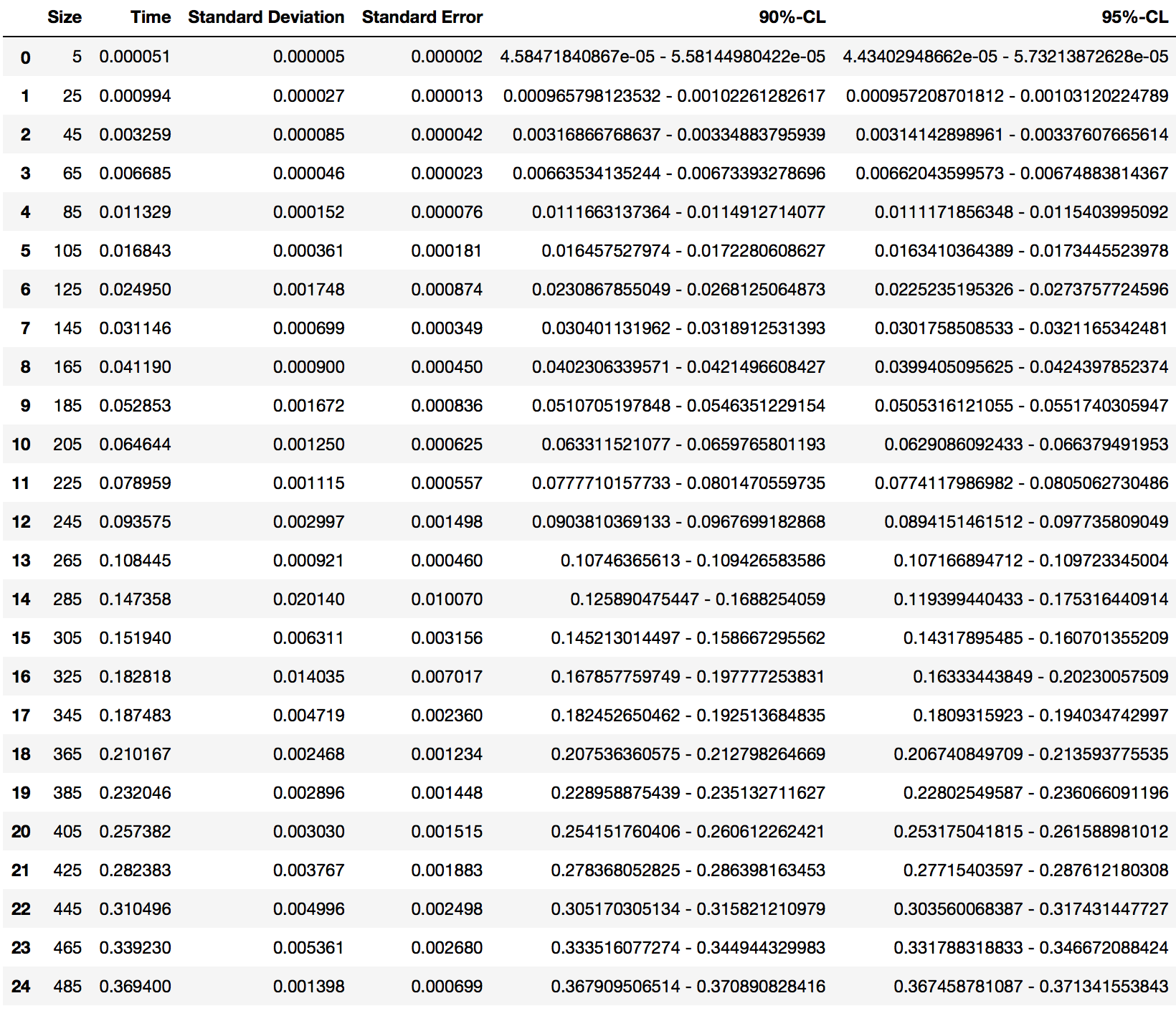


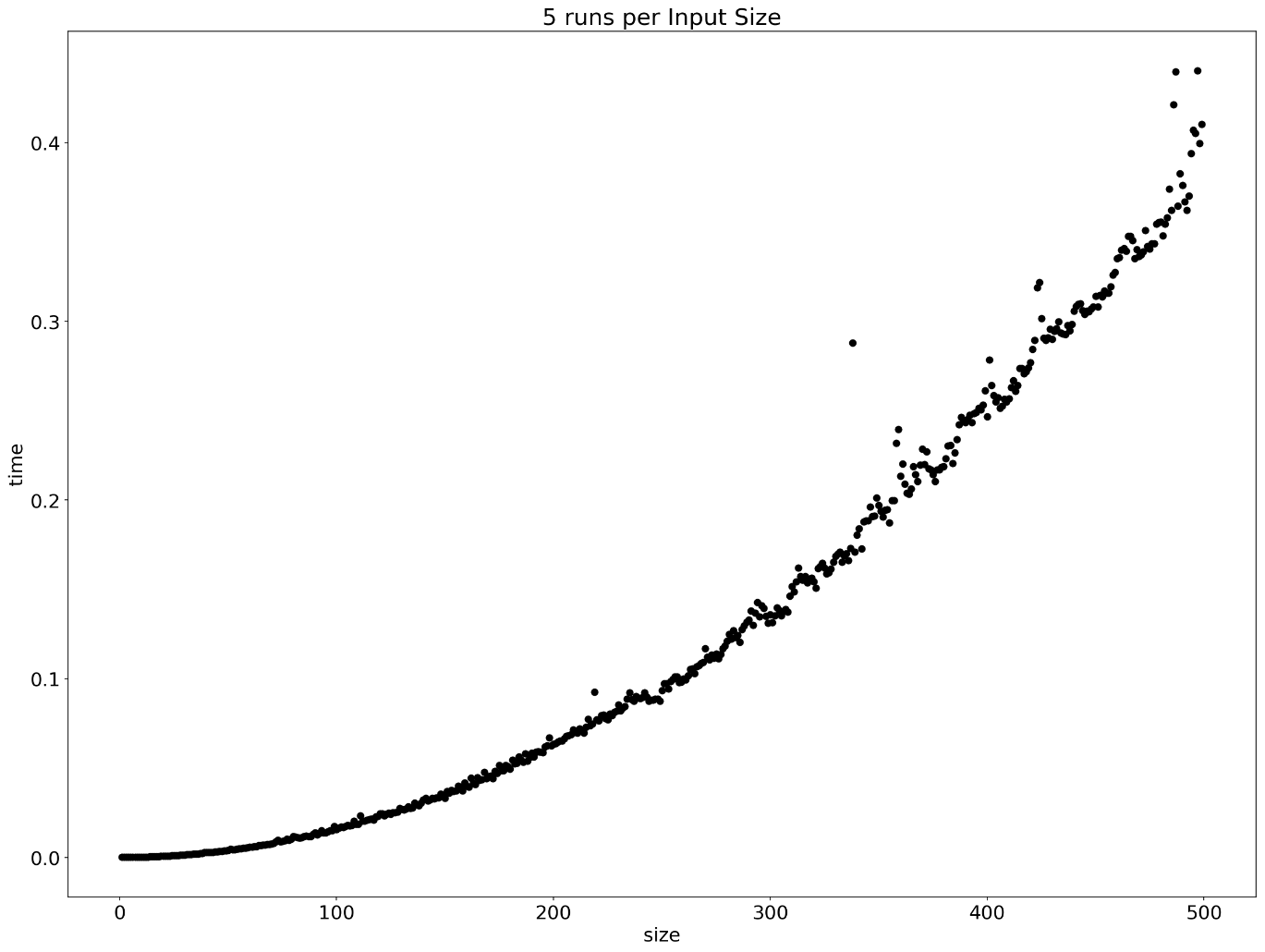
**Algorithm Analysis**

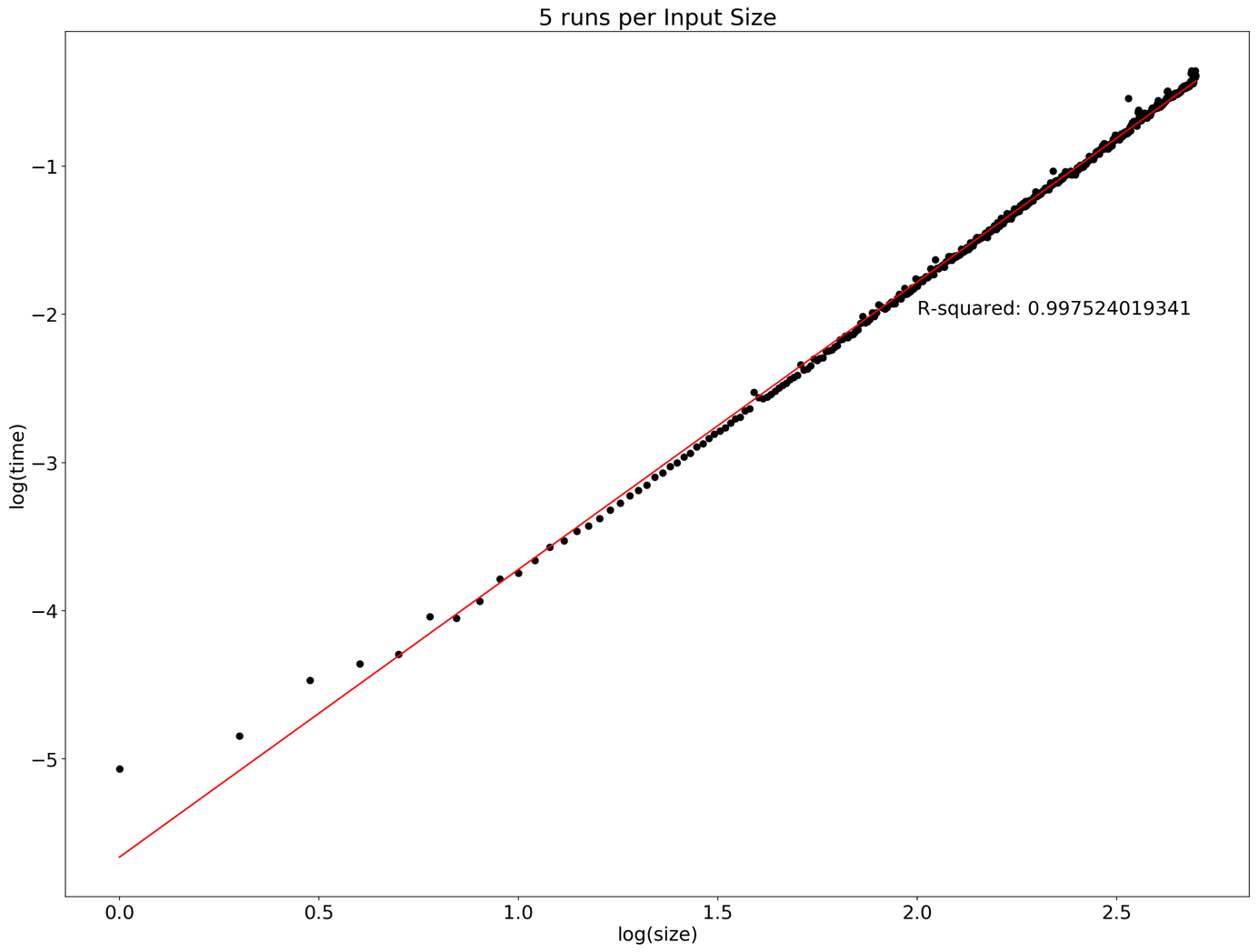
**TODO**

**Experimental Analysis**

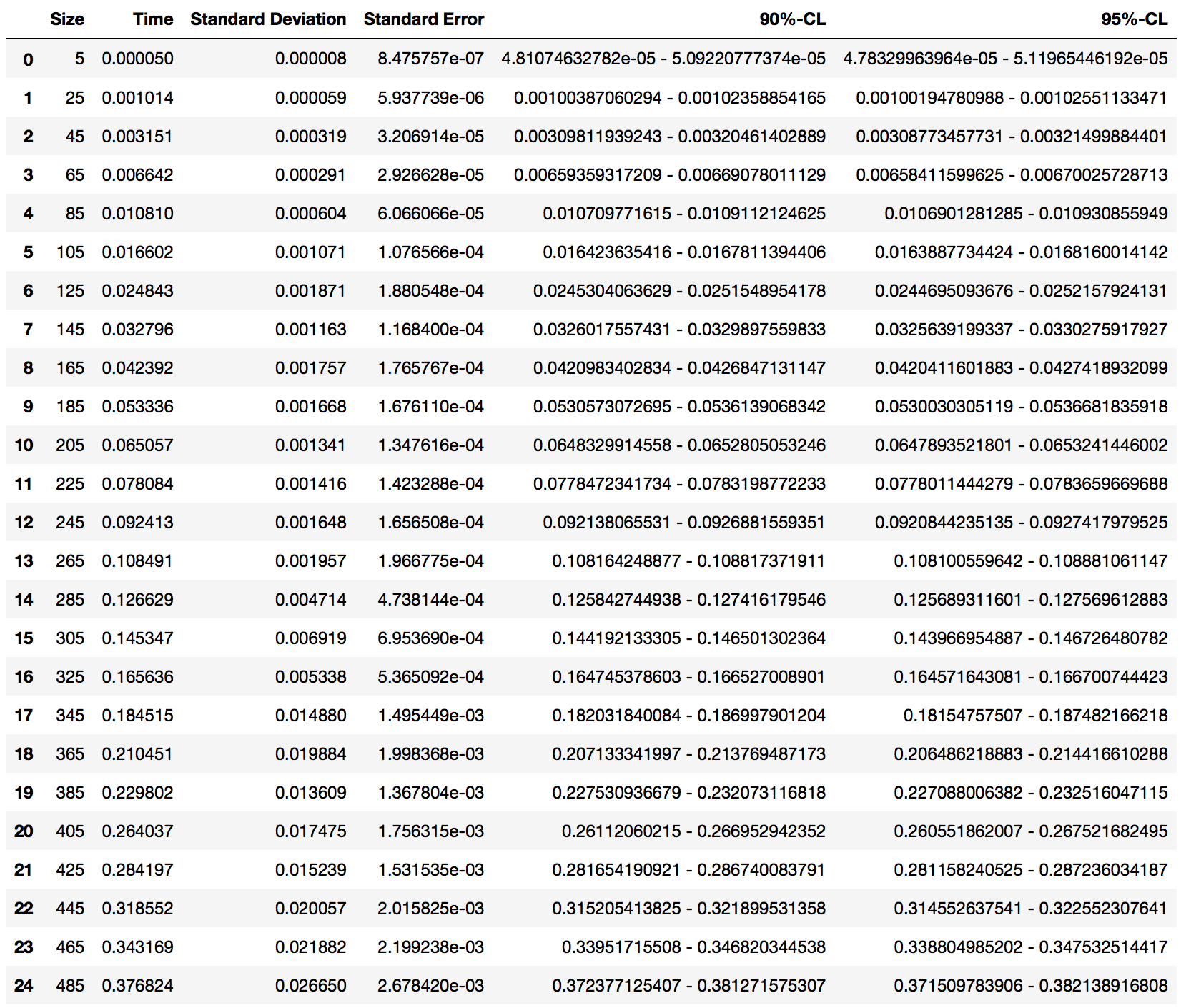
**5 Runs per Input Size**

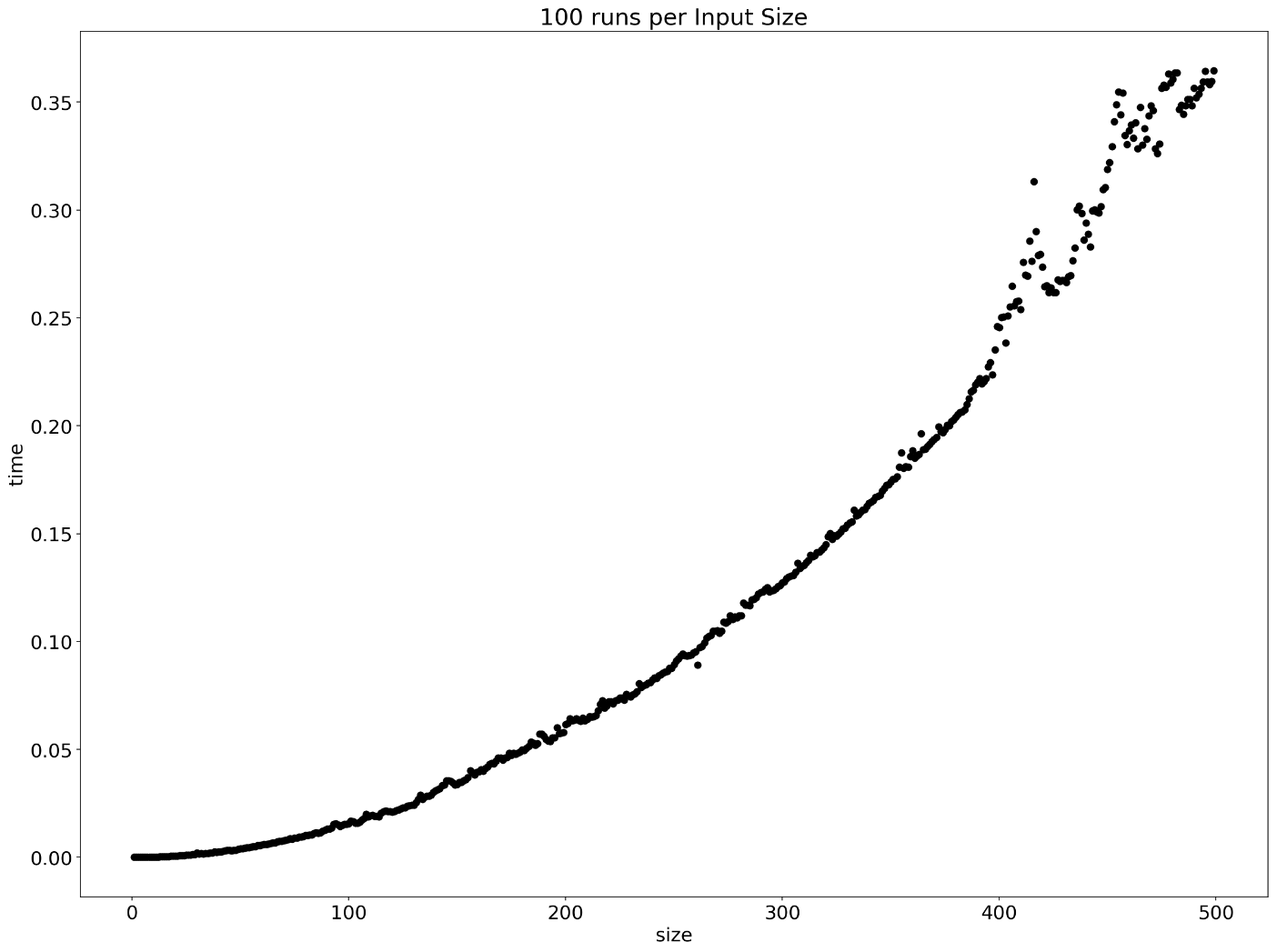
****

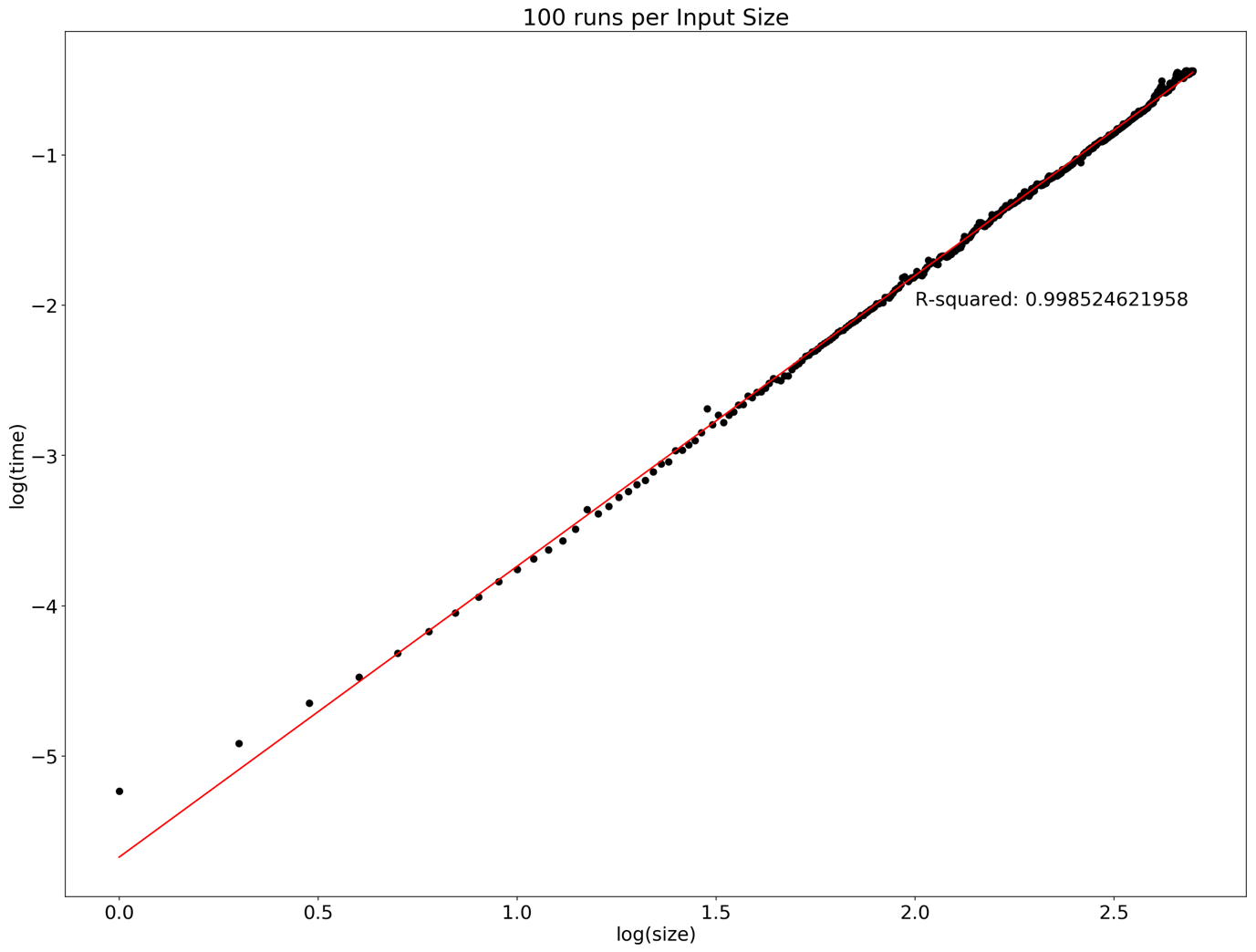
****

****

**100 Runs per Input Size**

****

****

****

**Testing**

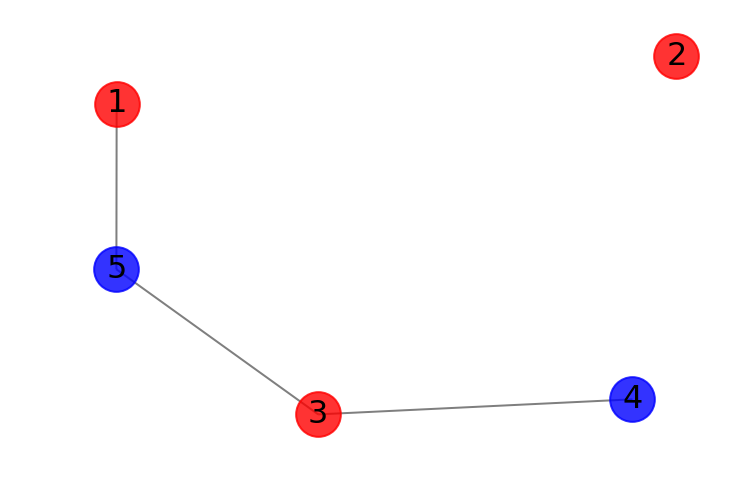
We decided to use black box testing. We create random graphs with giving number of vertices and how many graphs that we want to create with our code.

For **v=5**, we want to create 5 random graph and check the results. We use adjacency matrix representation for represent graphs. Numbers in the output represent the colors that are used.

**1.** [[0, 0, 0, 0, 1], [0, 0, 0, 0, 0], [0, 0, 0, 1, 1], [0, 0, 1, 0, 0], [1, 0, 1, 0, 0]]

Solution exists

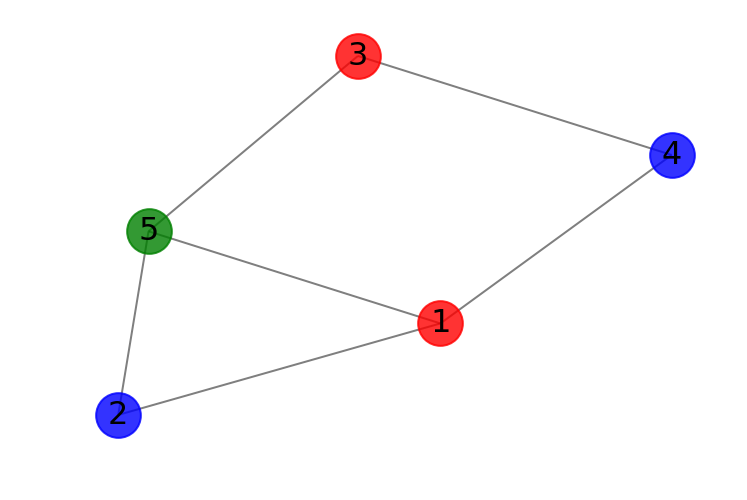
1,1,1,2,2



**2.** [[0, 1, 0, 1, 0], [1, 0, 0, 0, 1], [0, 0, 0, 1, 1], [1, 0, 1, 0, 0], [0, 1, 1, 0, 0]]

Solution exists

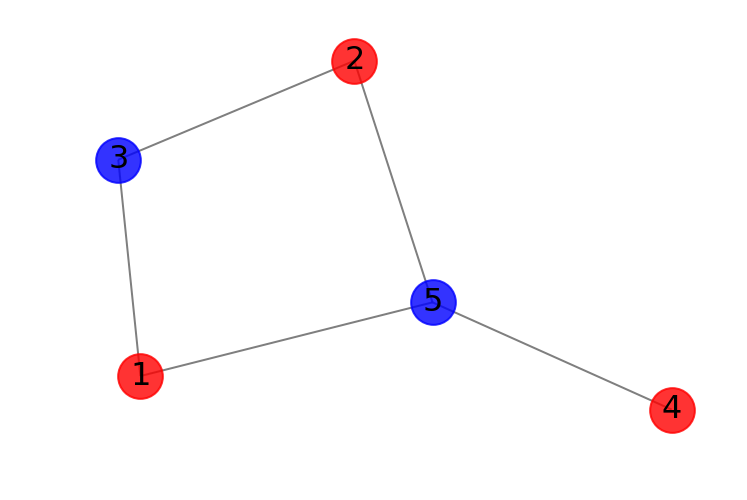
1,2,1,2,3



**3.** [[0, 0, 1, 0, 1], [0, 0, 1, 0, 1], [1, 1, 0, 0, 0], [0, 0, 0, 0, 1], [1, 1, 0, 1, 0]]

Solution exists

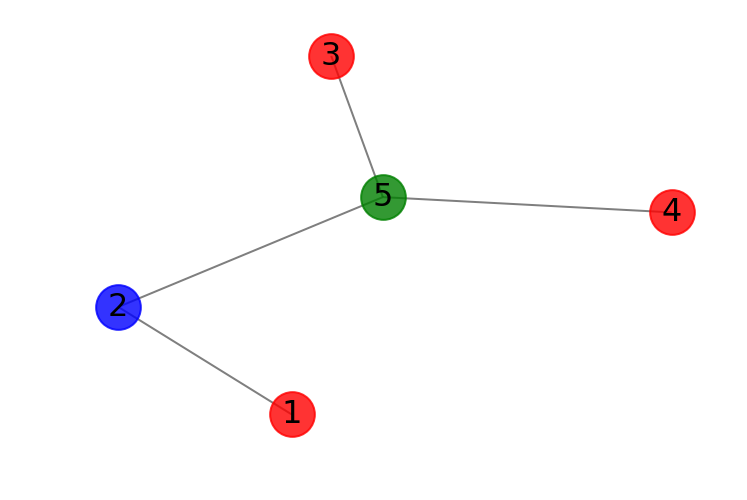
1,1,2,1,2



**4.** [[0, 1, 0, 0, 0], [1, 0, 0, 0, 1], [0, 0, 0, 0, 1], [0, 0, 0, 0, 1], [0, 1, 1, 1, 0]]

Solution exists

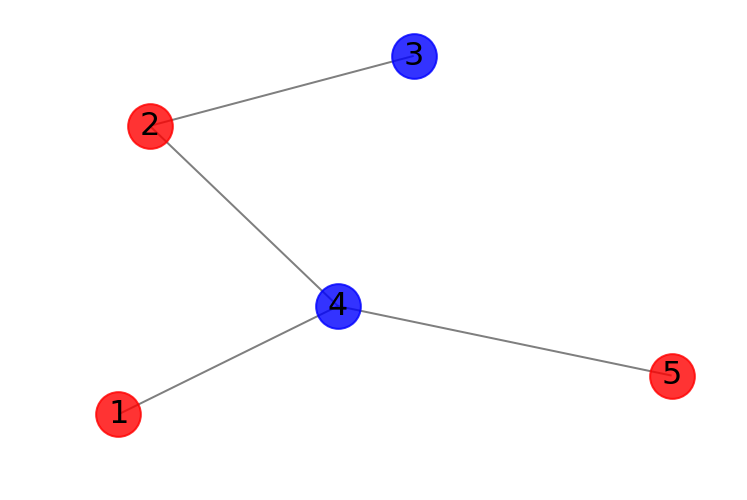
1,2,1,1,3



**5.** [[0, 0, 0, 1, 0], [0, 0, 1, 1, 0], [0, 1, 0, 0, 0], [1, 1, 0, 0, 1], [0, 0, 0, 1, 0]]

Solution exists

1,1,2,2,1



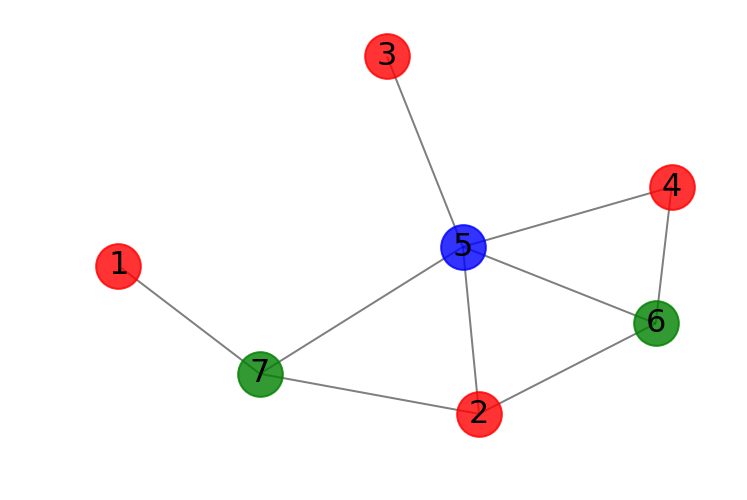
Another test for **v=7.**

Again, we create 5 graphs.

**1.** [[0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 1, 1, 1], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 1, 0], [0, 1, 1, 1, 0, 1, 1], [0, 1, 0, 1, 1, 0, 0], [1, 1, 0, 0, 1, 0, 0]]

Solution exists

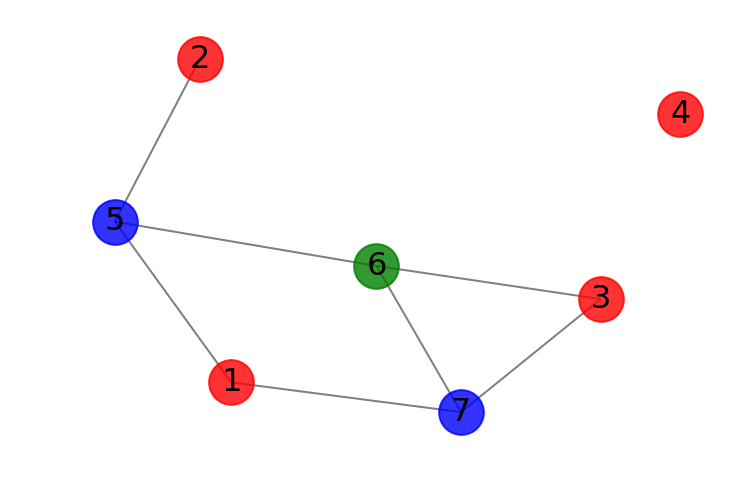
1,1,1,1,2,3,3



**2.** [[0, 0, 0, 0, 1, 0, 1], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0, 0], [1, 1, 0, 0, 0, 1, 0], [0, 0, 1, 0, 1, 0, 1], [1, 0, 1, 0, 0, 1, 0]]

Solution exists

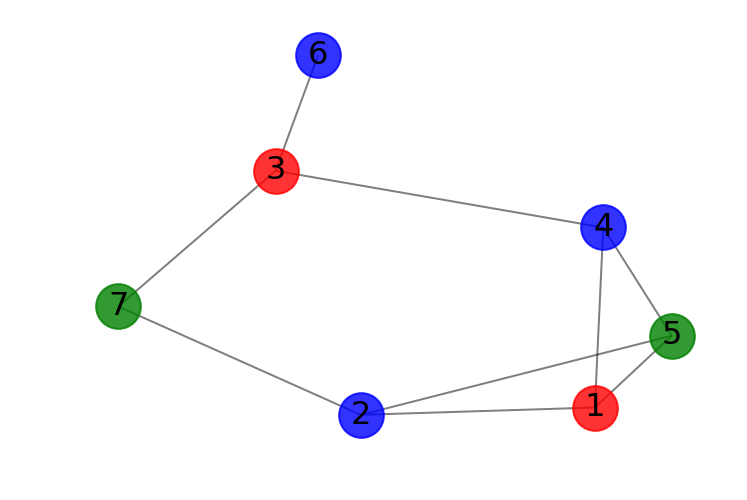
1,1,1,1,2,3,2



**3.** [[0, 1, 0, 1, 1, 0, 0], [1, 0, 0, 0, 1, 0, 1], [0, 0, 0, 1, 0, 1, 1], [1, 0, 1, 0, 1, 0, 0], [1, 1, 0, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 1, 1, 0, 0, 0, 0]]

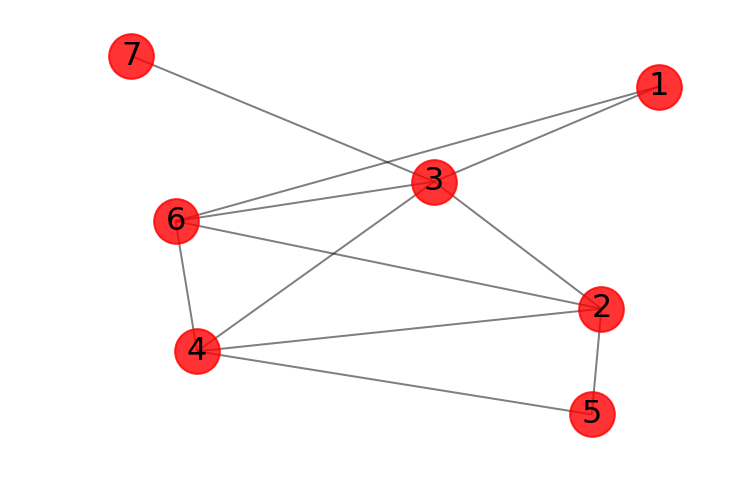
Solution exists

1,2,1,2,3,2,3



**4.** [[0, 0, 1, 0, 0, 1, 0], [0, 0, 1, 1, 1, 1, 0], [1, 1, 0, 1, 0, 1, 1], [0, 1, 1, 0, 1, 1, 0], [0, 1, 0, 1, 0, 0, 0], [1, 1, 1, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]]

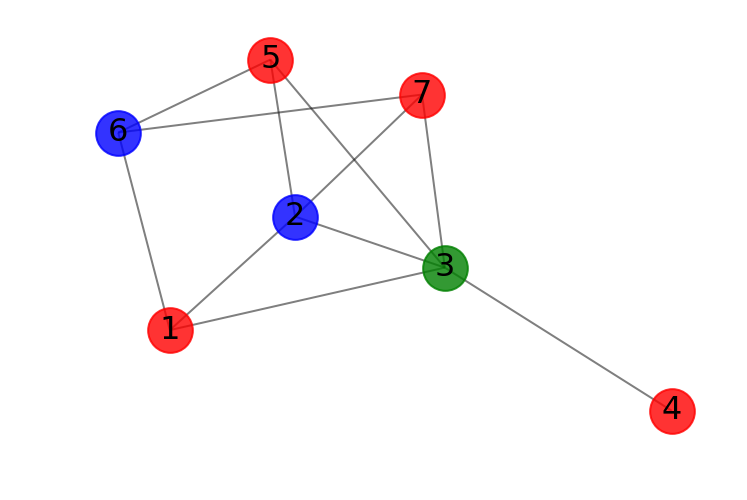
No solution.



**5.** [[0, 1, 1, 0, 0, 1, 0], [1, 0, 1, 0, 1, 0, 1], [1, 1, 0, 1, 1, 0, 1], [0, 0, 1, 0, 0, 0, 0], [0, 1, 1, 0, 0, 1, 0], [1, 0, 0, 0, 1, 0, 1], [0, 1, 1, 0, 0, 1, 0]]

Solution exists

1,2,3,1,1,2,1

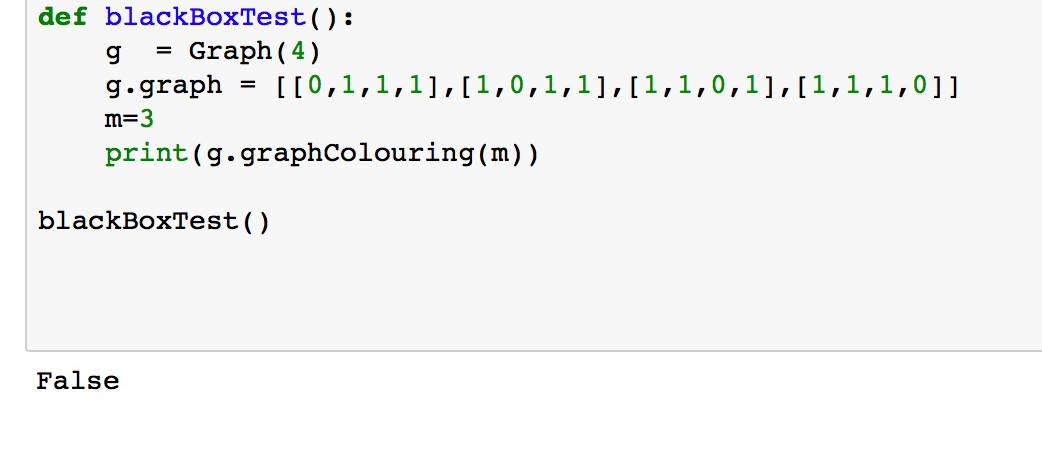


We observe that all solutions are correct, algorithm works correctly.

Another test Peterson Graph that we gave an example in description, and algorithm finds a solution, and it is correct.



Another test is with the graph that we know there should not exists a result. And the result is False which is correct.



**Conclusion**

TODO