1. Strategy:
This approximation algorithm uses a randomized, greedy approach. The algorithm takes all the vertices of the graph and shuffles the order of them randomly to avoid possible worst-case performance where this greedy algorithm may perform poorly. The greedy approach is dependent on the order of the vertices, some orderings give optimal or near optimal results, while other orderings cause the algorithm to use more colors than it needs to. This randomization helps reduce the chance of these bad orderings. After the random shuffling the algorithm goes through each vertex and makes a greedy choice, assigning the vertex the smallest color that is not used by any of its neighbors. This runs very quickly and provides a reasonable approximation for most graphs, however there are some scenarios where this algorithm does not yield the optimal result.

2. Analytical Runtime:
When the approximation algorithm runs, it goes through each vertex. For each vertex, it checks all of that vertex's neighbors and assigns the vertex the minimum color that isn't used by its neighbors. Looping through each vertex is O(V), checking a vertex's neighbors and assigning the minimum color relative to its neighbors each take degree of V steps. We know that the sum of degree of V for all vertices is 2E, so the total analytical runtime is O(V + E) which runs in polynomial time.

3. Gradescope performance:
The approximation algorithm performs well on Gradescope. It runs efficiently on all tests and is correct for all tests.

4. Test cases:
My run_test_cases.sh file tests 7 test cases. It tests two small graphs (10-20 vertices), one that is lightly connected and one that is highly connected. It then tests two medium sized graphs (120-150 vertices), one that is lightly connected and one that is highly connected. It then tests two large graphs (about 1200 vertices), one that is lightly connected and one that is highly connected. Finally it tests a graph with around 10 vertices that usually doesn't yield the optimal result. Since the approximation algorithm uses randomness, sometimes when this test is run it does yield the optimal result, however most of the time for this test, it does not. This test is also inside run_nonopt_cases.sh.