The strategy we used was greedy. We had two different greedy approaches. One was a simple greedy approach which moves across each vertex assigning it the first valid coloring. The next greedy approach, known as the Welsh-Powell Algorithm, colors vertices from highest degree to lowest degree. The runtime complexity of our simple algorithm was O(V + E) while the runtime of our Welsh-Powell implementation was O(V log V + E). Both of these ran in polynomial time and were able to handle very large input sizes (1000 vertices and 500,000 edges).

```
Test cases:
        test    result  runtime
        test_large_connected    passed  .6069031s
        test_large_connected2   passed  1.0258840s
        test_nonoptimal passed  .0809894s
        test_petersen_graph     passed  .1010330s
        test_small_connected    passed  .0859973s
```

Our large connected 2 test for example contains 288,420 edges and our approximation code ran in only 1.02 seconds.

```
def color_highest_degree(G):
    Colors = {}
    pq = PriorityQueue()
    add all v in G to pq where key is degree

    while pq isn't empty:
        v = pq.get()
        color = assign_color(G, v, colors)
        colors[v] = color

    return len(set(values of colors)
```

```
def assign_color(G, v, colors):
    neighbor_colors = set()
    for u in G[v]:
        if u in colors:
            neighbor_colors.add(colors[u])

    color = 1
    while color in neighbor_colors:
        color += 1
    return color
```

Overall time complexity: O(V log V + E)

Above is the pseudo code for the Welsh-Powell algorithm. It runs a while loop over all the nodes which have been added to a priority queue. Each iteration it calls pq.get() which runs in O(log n) time. The loop combined with the get runs in O(V log V). The assign_color function in the worst case will take in a vertex whose degree is the same as the number of edges in the graph. This runs in O(E) time. Overall this approximation is polynomial.