ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

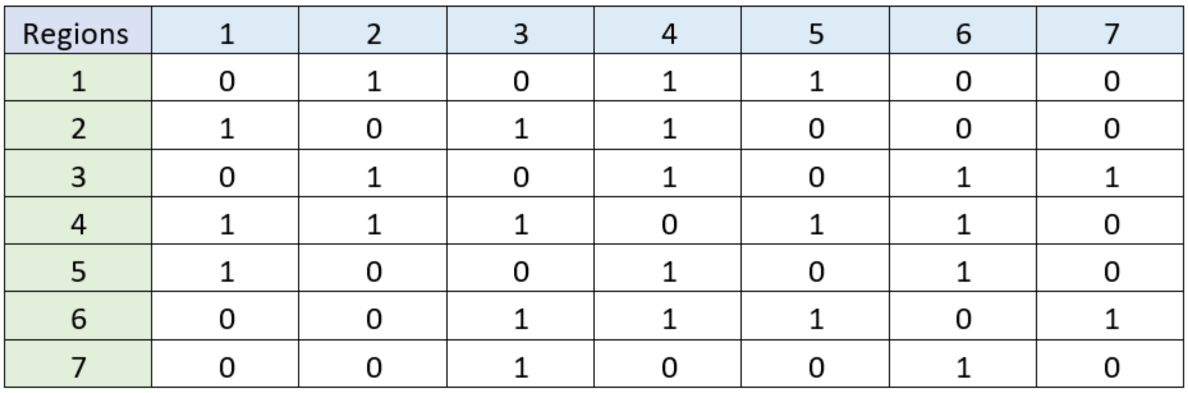
BLG 336E
ANALYSIS OF ALGORITHMS
ASSIGNMENT 2

070170450 : MUHAMMED ENES DENİZ

SPRING 2022

Contents

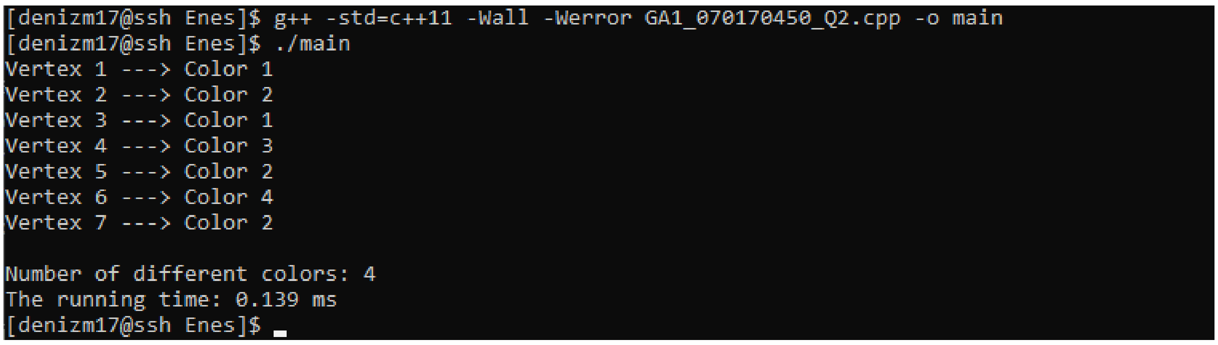
1 Question 1



Regions	1	2	3	4	5	6	7
1	0	1	0	1	1	0	0
2	1	0	1	1	0	0	0
3	0	1	0	1	0	1	1
4	1	1	1	0	1	1	0
5	1	0	0	1	0	1	0
6	0	0	1	1	1	0	1
7	0	0	1	0	0	1	0

Figure 1: Adjacency Matrix Representation for G2

2 Question 2

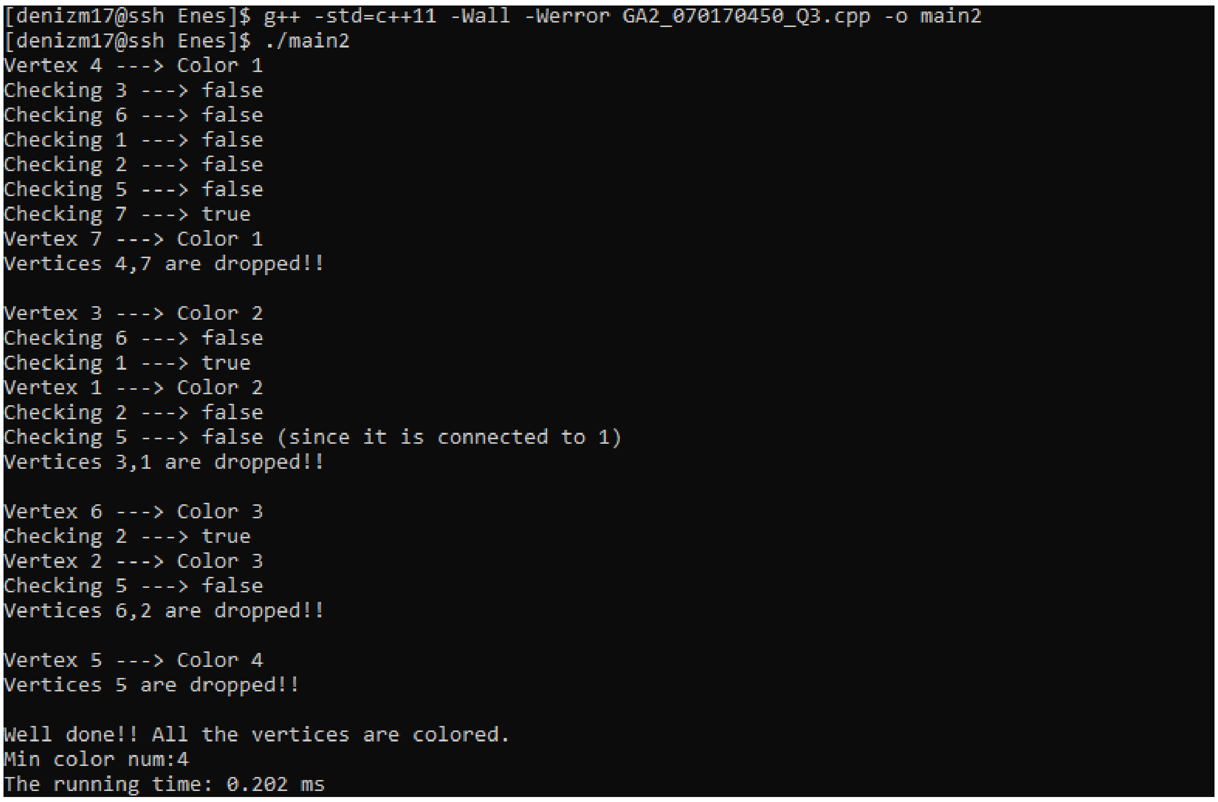


```
[denizm17@ssh Enes]$ g++ -std=c++11 -Wall -Werror GA1_070170450_Q2.cpp -o main
[denizm17@ssh Enes]$ ./main
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 1
Vertex 4 ---> Color 3
Vertex 5 ---> Color 2
Vertex 6 ---> Color 4
Vertex 7 ---> Color 2

Number of different colors: 4
The running time: 0.139 ms
[denizm17@ssh Enes]$ 
```

Figure 2: Greedy Algorithm 1 Graph Coloring Result

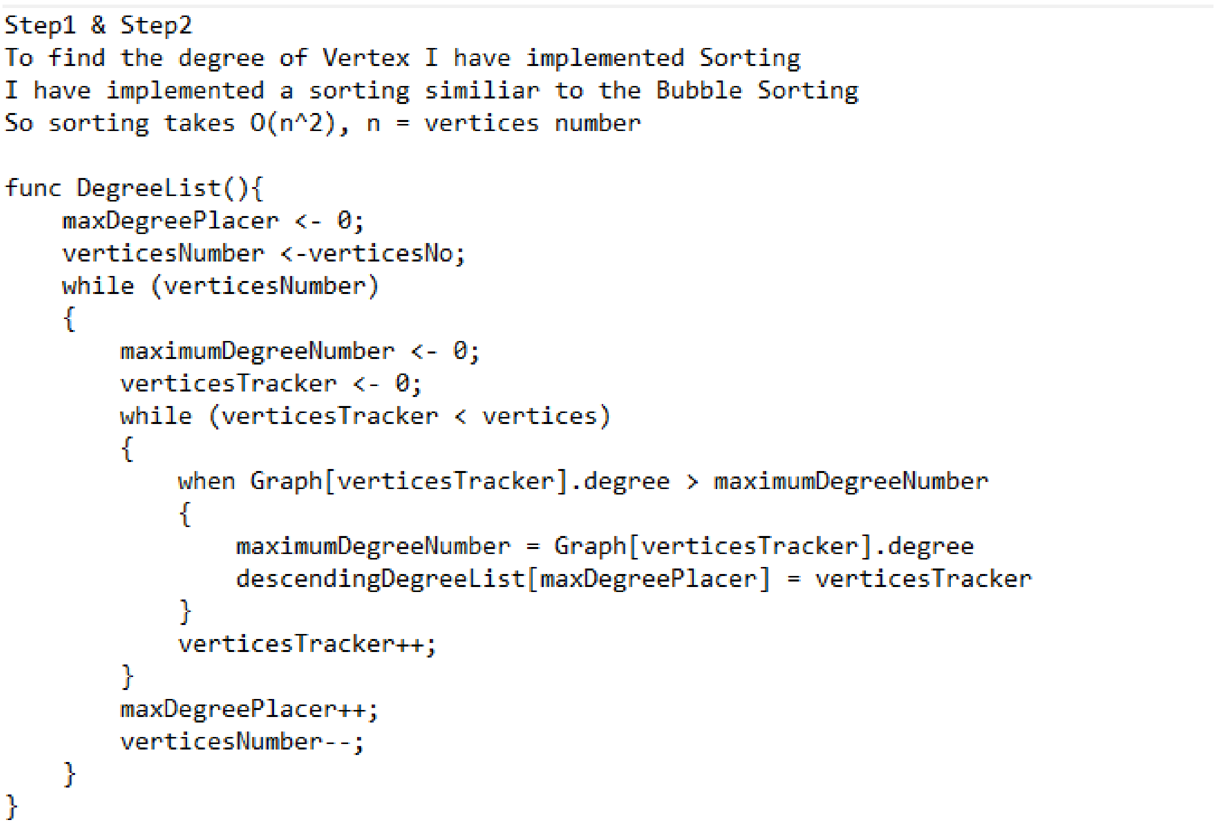
4 color used and it took about 0.139 milliseconds.

3 Question 3



```
[denizm17@ssh Enes]$ g++ -std=c++11 -Wall -Werror GA2_070170450_Q3.cpp -o main2
[denizm17@ssh Enes]$ ./main2
Vertex 4 ---> Color 1
Checking 3 ---> false
Checking 6 ---> false
Checking 1 ---> false
Checking 2 ---> false
Checking 5 ---> false
Checking 7 ---> true
Vertex 7 ---> Color 1
Vertices 4,7 are dropped!!

Vertex 3 ---> Color 2
Checking 6 ---> false
Checking 1 ---> true
Vertex 1 ---> Color 2
Checking 2 ---> false
Checking 5 ---> false (since it is connected to 1)
Vertices 3,1 are dropped!!

Vertex 6 ---> Color 3
Checking 2 ---> true
Vertex 2 ---> Color 3
Checking 5 ---> false
Vertices 6,2 are dropped!!

Vertex 5 ---> Color 4
Vertices 5 are dropped!!

Well done!! All the vertices are colored.
Min color num:4
The running time: 0.202 ms
```

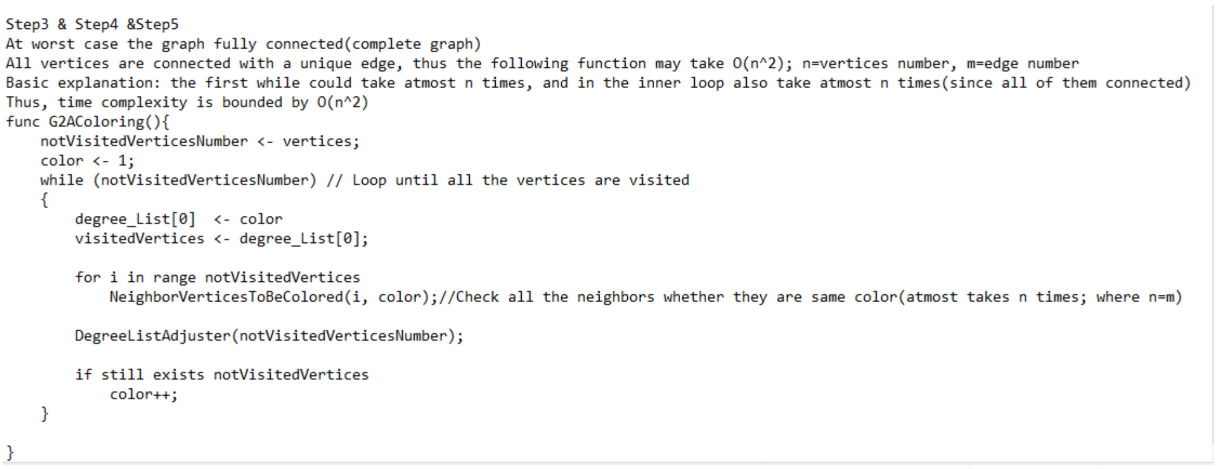
Figure 3: Greedy Algorithm 2 Graph Coloring Result



```
Step1 & Step2
To find the degree of Vertex I have implemented Sorting
I have implemented a sorting similiar to the Bubble Sorting
So sorting takes O(n^2), n = vertices number

func DegreeList(){
    maxDegreePlacer <- 0;
    verticesNumber <-verticesNo;
    while (verticesNumber)
    {
        maximumDegreeNumber <- 0;
        verticesTracker <- 0;
        while (verticesTracker < vertices)
        {
            when Graph[verticesTracker].degree > maximumDegreeNumber
            {
                maximumDegreeNumber = Graph[verticesTracker].degree
                descendingDegreeList[maxDegreePlacer] = verticesTracker
            }
            verticesTracker++;
        }
        maxDegreePlacer++;
        verticesNumber--;
    }
}
```

Figure 4: Greedy Algorithm PseudoCode I

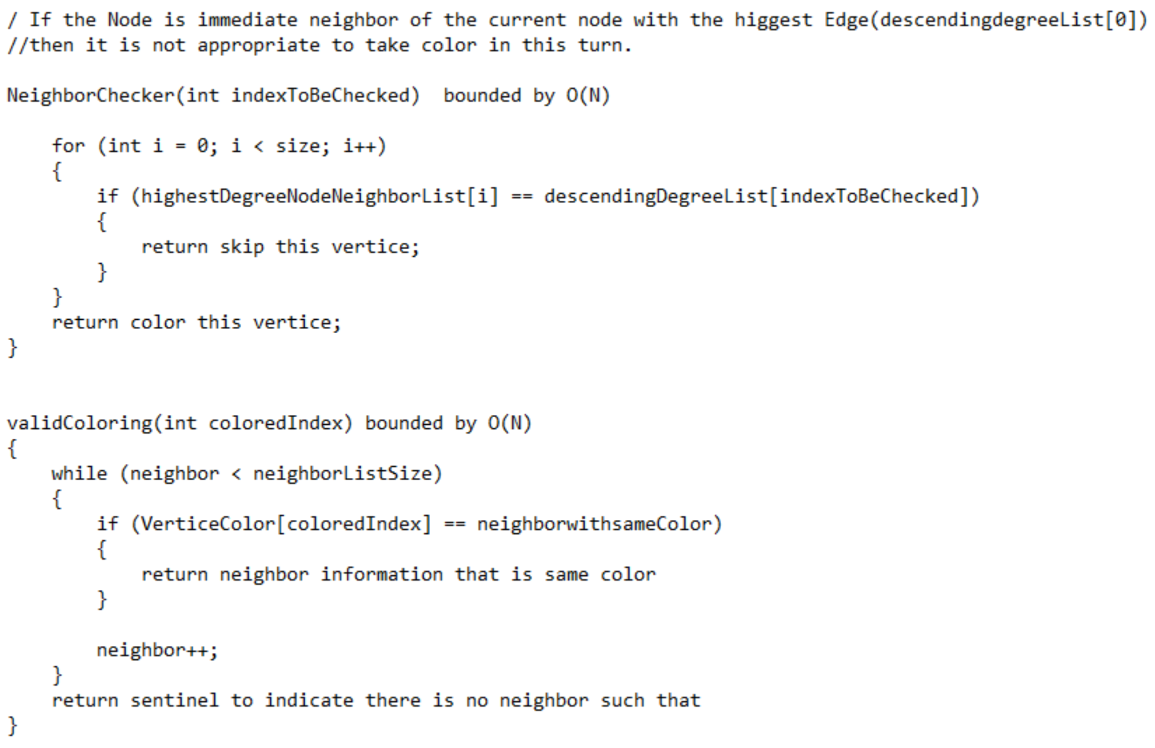
Finding the degrees of vertices and sorting them in descending order could be take O(n^2) as I implemented bubble sort kinda approach to the sorting.



```
Step3 & Step4 &Step5
At worst case the graph fully connected(complete graph)
All vertices are connected with a unique edge, thus the following function may take O(n^2); n=vertices number, m=edge number
Basic explanation: the first while could take atmost n times, and in the inner loop also take atmost n times(since all of them connected)
Thus, time complexity is bounded by O(n^2)
func G2AColoring(){
    notVisitedVerticesNumber <- vertices;
    color <- 1;
    while (notVisitedVerticesNumber) // Loop until all the vertices are visited
    {
        degree_List[0]  <- color
        visitedVertices <- degree_List[0];

        for i in range notVisitedVertices
            NeighborVerticesToBeColored(i, color);//Check all the neighbors whether they are same color(atmost takes n times; where n=m)

        DegreeListAdjuster(notVisitedVerticesNumber);

        if still exists notVisitedVertices
            color++;
    }

}
```

Figure 5: Greedy Algorithm PseudoCode II

As depicted in the figure, this greedy algorithm's part3 and rest could also take O(n^2) times when the graph is fully connected(complete graph). Since it needs to check every vertex to be color (n) the outer loop and also it needs to be validate the color by checking it's all neighbors when this case the neighbor size is equal to the (n) inner loop, thus time complexity of the greedy algorithm 2 is bounded by O(nˆ2) at the worst case.



```
/ If the Node is immediate neighbor of the current node with the higgest Edge(descendingdegreeList[0])
//then it is not appropriate to take color in this turn.

NeighborChecker(int indexToBeChecked)  bounded by O(N)

    for (int i = 0; i < size; i++)
    {
        if (highestDegreeNodeNeighborList[i] == descendingDegreeList[indexToBeChecked])
        {
            return skip this vertice;
        }
    }
    return color this vertice;
}


validColoring(int coloredIndex) bounded by O(N)
{
    while (neighbor < neighborListSize)
    {
        if (VerticeColor[coloredIndex] == neighborwithsameColor)
        {
            return neighbor information that is same color
        }

        neighbor++;
    }
    return sentinel to indicate there is no neighbor such that
}
```

Figure 6: Greedy Algorithm PseudoCode III

The mentioned validity check operations could be seen above.

4 Question 4

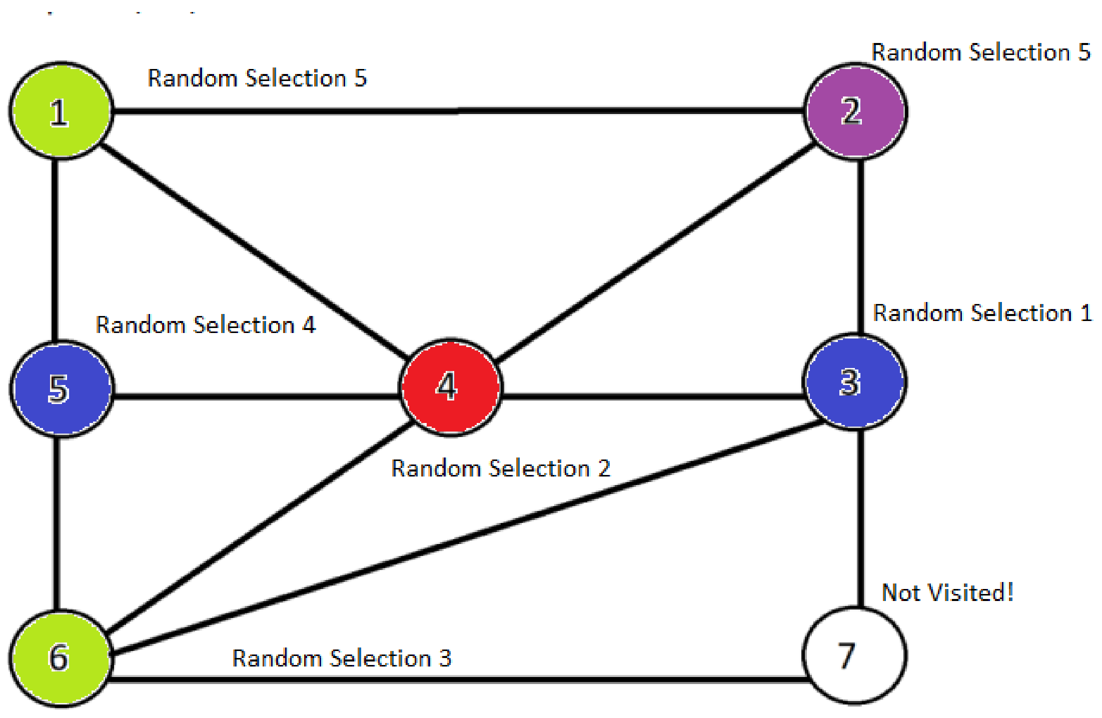


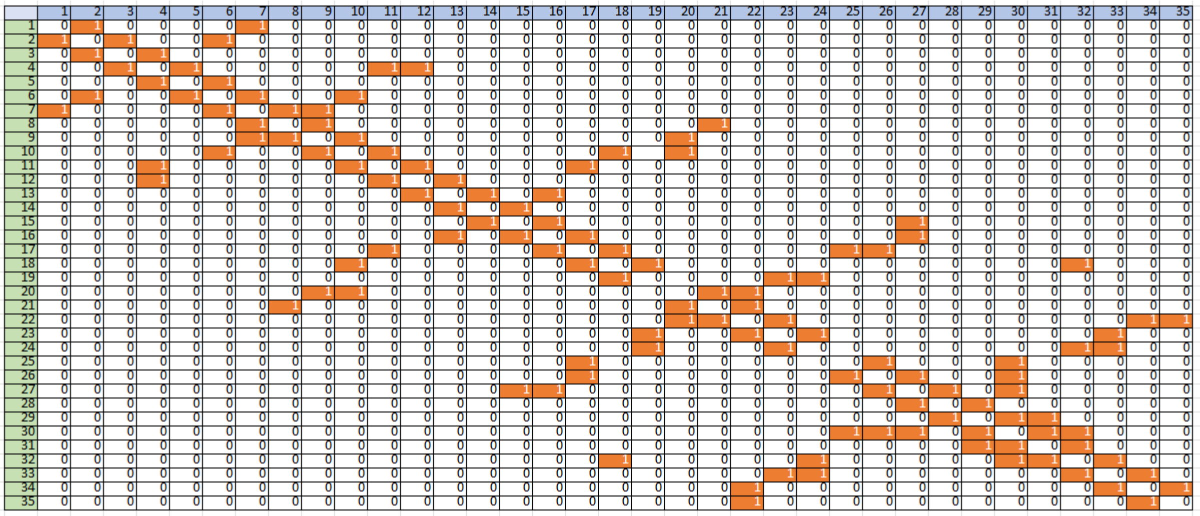
Figure 7: Counter Example

Let's assume blue < red < yellow < purple order. Blue is the lowest numbered color.

If the graph is not fully connected (not all vertices have an edge between them) the given algorithm will fail to paint all of the vertices. In the figure such scenario is given; Firstly, algorithm choose the vertice 3 to start and color it blue. Then it will select one of it's non-colored adjacent(neighbor vertice which have an edge to our current vertice) vertice the possible list is (2,4,6,7) but let's assume the algorithm choose the 4 and color it red, then again it will look the adjacent(neighbor) non colored vertices which are in this case (1,2,5,6) and let's assume that it chooses to follow 6 and colored it with yellow (since neighbor vertices could not have some color) for 6's adjacent non colored vertices list is (5,7) now let's assume that our randomization goes to 5 and color it with blue. Vertice 5's non colored adjacent vertice is only 1 so it goes there and color it with appropriate color ,yellow,. Vertice 1 non colored neighbor vertice is 2 so it goes there and color the vertice 2 with purple. Since all of the neighbor(adjacent) vertices of the vertice 2 is already colored our algorithm could not have any possible chance to color vertice 7. Thus vertice 7 is never visited or colored. Another possible sequence is 4 - 5 - 6 - 3 - 2 - 1 : 7 again will not be colored.

5 Question 5

5.1 A

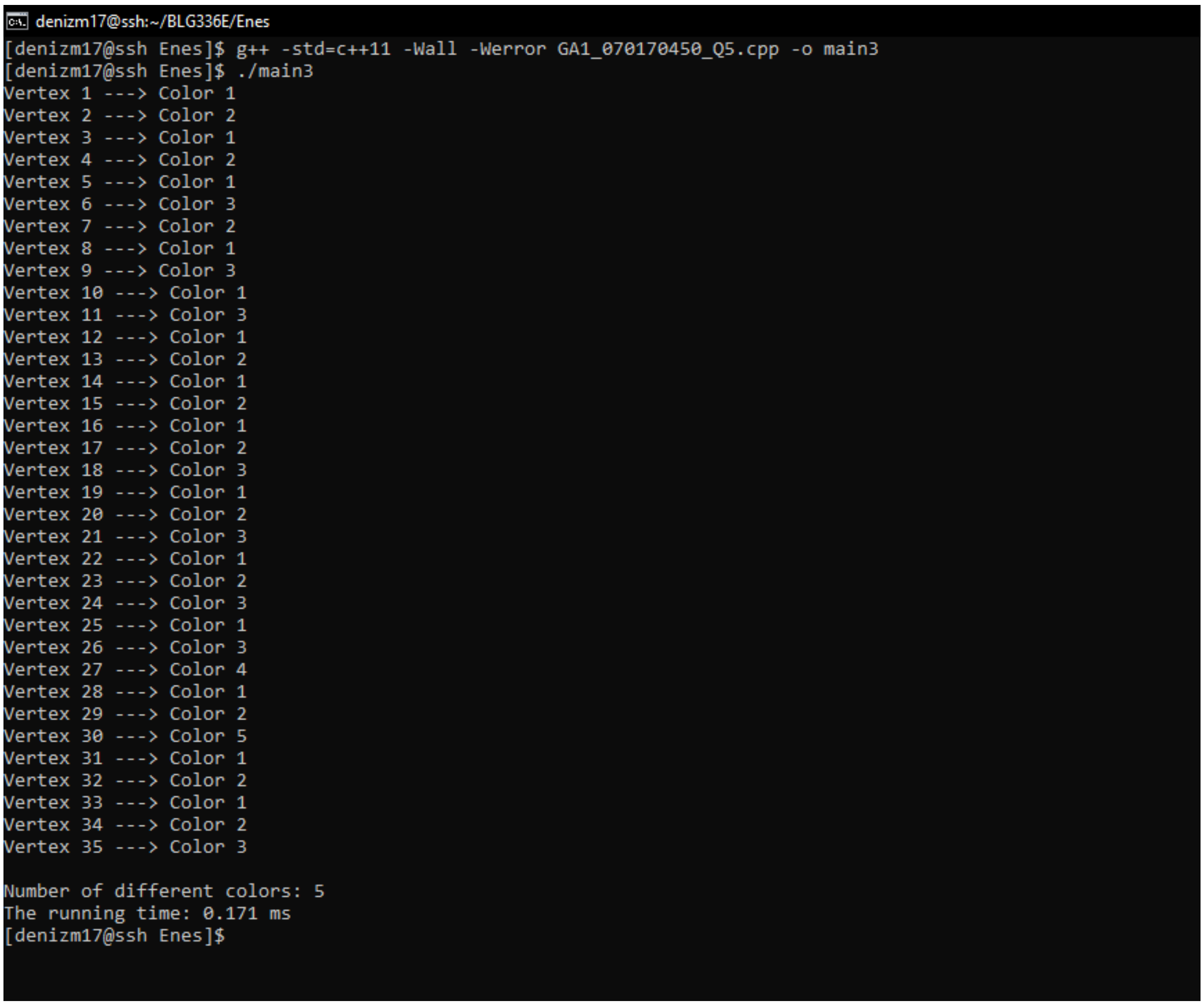


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	1	1	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 8: SNA Adjacency Matrix

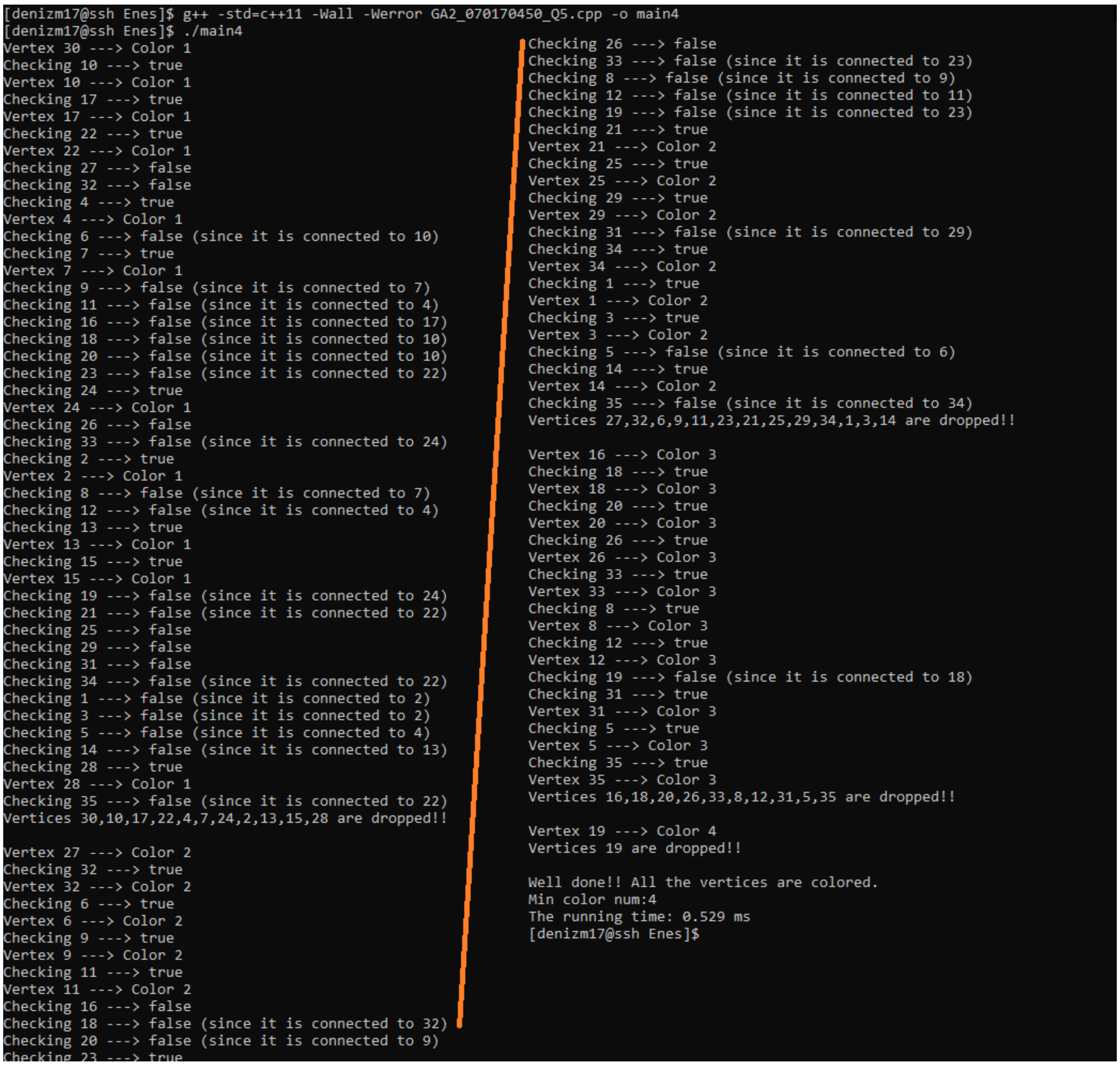
From my code all of the adjacency relations could be easily observable but also I have prepared an excel to visualize it much more.

5.2 B



```
denizm17@ssh:~/BLG336E/Enes
[denizm17@ssh Enes]$ g++ -std=c++11 -Wall -Werror GA1_070170450_Q5.cpp -o main3
[denizm17@ssh Enes]$ ./main3
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 1
Vertex 4 ---> Color 2
Vertex 5 ---> Color 1
Vertex 6 ---> Color 3
Vertex 7 ---> Color 2
Vertex 8 ---> Color 1
Vertex 9 ---> Color 3
Vertex 10 ---> Color 1
Vertex 11 ---> Color 3
Vertex 12 ---> Color 1
Vertex 13 ---> Color 2
Vertex 14 ---> Color 1
Vertex 15 ---> Color 2
Vertex 16 ---> Color 1
Vertex 17 ---> Color 2
Vertex 18 ---> Color 3
Vertex 19 ---> Color 1
Vertex 20 ---> Color 2
Vertex 21 ---> Color 3
Vertex 22 ---> Color 1
Vertex 23 ---> Color 2
Vertex 24 ---> Color 3
Vertex 25 ---> Color 1
Vertex 26 ---> Color 3
Vertex 27 ---> Color 4
Vertex 28 ---> Color 1
Vertex 29 ---> Color 2
Vertex 30 ---> Color 5
Vertex 31 ---> Color 1
Vertex 32 ---> Color 2
Vertex 33 ---> Color 1
Vertex 34 ---> Color 2
Vertex 35 ---> Color 3

Number of different colors: 5
The running time: 0.171 ms
[denizm17@ssh Enes]$
```

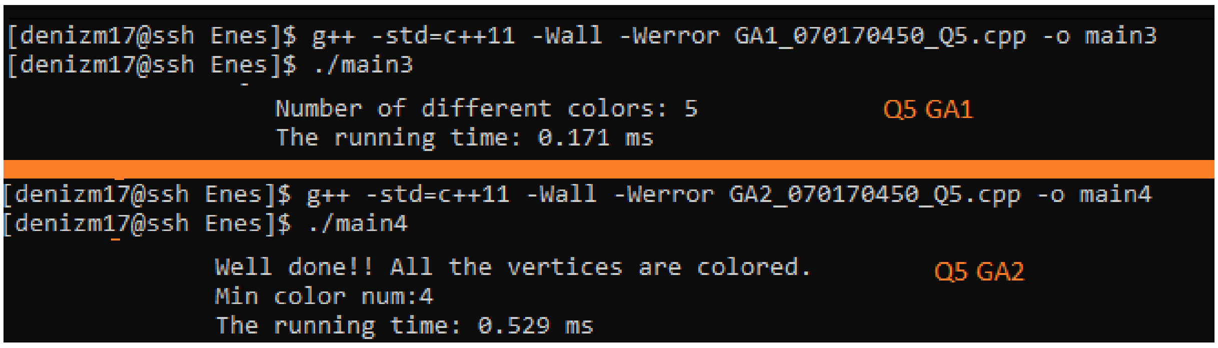
Figure 9: Question 5 Greedy Algorithm 1 Result



```
[denizm17@ssh Enes]$ g++ -std=c++11 -Wall -Werror GA2_070170450_Q5.cpp -o main4
[denizm17@ssh Enes]$ ./main4
Vertex 30 ---> Color 1
Checking 10 ---> true
Vertex 10 ---> Color 1
Checking 17 ---> true
Vertex 17 ---> Color 1
Checking 22 ---> true
Vertex 22 ---> Color 1
Checking 27 ---> false
Checking 32 ---> false
Checking 4 ---> true
Vertex 4 ---> Color 1
Checking 6 ---> false (since it is connected to 10)
Checking 7 ---> true
Vertex 7 ---> Color 1
Checking 9 ---> false (since it is connected to 7)
Checking 11 ---> false (since it is connected to 4)
Checking 16 ---> false (since it is connected to 17)
Checking 18 ---> false (since it is connected to 10)
Checking 20 ---> false (since it is connected to 10)
Checking 23 ---> false (since it is connected to 22)
Checking 24 ---> true
Vertex 24 ---> Color 1
Checking 26 ---> false
Checking 33 ---> false (since it is connected to 24)
Checking 2 ---> true
Vertex 2 ---> Color 1
Checking 8 ---> false (since it is connected to 7)
Checking 12 ---> false (since it is connected to 4)
Checking 13 ---> true
Vertex 13 ---> Color 1
Checking 15 ---> true
Vertex 15 ---> Color 1
Checking 19 ---> false (since it is connected to 24)
Checking 21 ---> false (since it is connected to 22)
Checking 25 ---> false
Checking 29 ---> false
Checking 31 ---> false
Checking 34 ---> false (since it is connected to 22)
Checking 1 ---> false (since it is connected to 2)
Checking 3 ---> false (since it is connected to 2)
Checking 5 ---> false (since it is connected to 4)
Checking 14 ---> false (since it is connected to 13)
Checking 28 ---> true
Vertex 28 ---> Color 1
Checking 35 ---> false (since it is connected to 22)
Vertices 30,10,17,22,4,7,24,2,13,15,28 are dropped!!

Vertex 27 ---> Color 2
Checking 32 ---> true
Vertex 32 ---> Color 2
Checking 6 ---> true
Vertex 6 ---> Color 2
Checking 9 ---> true
Vertex 9 ---> Color 2
Checking 11 ---> true
Vertex 11 ---> Color 2
Checking 16 ---> false
Checking 18 ---> false (since it is connected to 32)
Checking 20 ---> false (since it is connected to 9)
Checking 23 ---> true
```

```
Checking 26 ---> false
Checking 33 ---> false (since it is connected to 23)
Checking 8 ---> false (since it is connected to 9)
Checking 12 ---> false (since it is connected to 11)
Checking 19 ---> false (since it is connected to 23)
Checking 21 ---> true
Vertex 21 ---> Color 2
Checking 25 ---> true
Vertex 25 ---> Color 2
Checking 29 ---> true
Vertex 29 ---> Color 2
Checking 31 ---> false (since it is connected to 29)
Checking 34 ---> true
Vertex 34 ---> Color 2
Checking 1 ---> true
Vertex 1 ---> Color 2
Checking 3 ---> true
Vertex 3 ---> Color 2
Checking 5 ---> false (since it is connected to 6)
Checking 14 ---> true
Vertex 14 ---> Color 2
Checking 35 ---> false (since it is connected to 34)
Vertices 27,32,6,9,11,23,21,25,29,34,1,3,14 are dropped!!

Vertex 16 ---> Color 3
Checking 18 ---> true
Vertex 18 ---> Color 3
Checking 20 ---> true
Vertex 20 ---> Color 3
Checking 26 ---> true
Vertex 26 ---> Color 3
Checking 33 ---> true
Vertex 33 ---> Color 3
Checking 8 ---> true
Vertex 8 ---> Color 3
Checking 12 ---> true
Vertex 12 ---> Color 3
Checking 19 ---> false (since it is connected to 18)
Checking 31 ---> true
Vertex 31 ---> Color 3
Checking 5 ---> true
Vertex 5 ---> Color 3
Checking 35 ---> true
Vertex 35 ---> Color 3
Vertices 16,18,20,26,33,8,12,31,5,35 are dropped!!

Vertex 19 ---> Color 4
Vertices 19 are dropped!!

Well done!! All the vertices are colored.
Min color num:4
The running time: 0.529 ms
[denizm17@ssh Enes]$
```

Figure 10: Question 5 Greedy Algorithm 2 Result



```
[denizm17@ssh Enes]$ g++ -std=c++11 -Wall -Werror GA1_070170450_Q5.cpp -o main3
[denizm17@ssh Enes]$ ./main3
                 Number of different colors: 5
                 The running time: 0.171 ms
```

```
[denizm17@ssh Enes]$ g++ -std=c++11 -Wall -Werror GA2_070170450_Q5.cpp -o main4
[denizm17@ssh Enes]$ ./main4
              Well done!! All the vertices are colored.
              Min color num:4
              The running time: 0.529 ms
```

Figure 11: Color Numbers and Run Time Millisecond

5.3 C

As it could be observable from the previous section, Greedy Algorithm 1 only takes 0.171 millisecond to execute whereas Greedy Algorithm 2 takes 0.529 millisecond. Thus, Greedy Algorithm 1 showed better performance in terms of execution time. Greedy Algorithm 1 is kinda 3 times faster than the Greedy Algorithm 2.

5.4 D

As it could be observable from the figure, Greedy Algorithm 2 used 4 color for coloring whereas Greedy Algorithm 1 used 5 color for coloring the vertices. Thus, Greedy Algorithm 2 performed better performance in terms of finding the minimum number of colors. Greedy Algorithm 2 uses 1 less color compared to Greedy Algorithm 1.

6 Discussion

As visualized and tested in the Question 5, The Greedy Algorithm 1 did not always give the possible minimum number of color for any given graph. Greedy Algorithm 2 accomplished to use less color then the Greedy Algorithm 1 to color the SNA graph.

Better optimization cases could be applied over our Greedy Algorithm's specific to problem context. By this optimization we may obtain less different color number but this optimizations could not be generalized and depends on the situation. From my web searches and tries I have found that If we also order the ID's of the vertices descendant when the degree numbers are equal Greedy Algorithm could color the graph with 3 colors. By applying such optimization mechanism we could lower the different color numbers but yet Greedy Algorithm 2 will again use less color then the Greedy Algorithm 1.

If we look at the other side, Greedy Algorithm 1 steadily gives better run time compared to Greedy Algorithm 2, so there exists a trade off between them. To answer which is one is better also depends highly on the problem context. For example in the given Turkey Regions Graph, but algorithms accomplished to color with the same color number yet Greedy Algorithm 1 is faster. And in the case of a big graph like SNA Greedy Algorithm 2 uses less color yet it's slower. In case of sparse graphs, if we value the run time we could use Greedy Algorithm 1, and for the case of dense graphs and run time is not prioritized we could use Greedy Algorithm 2.

Greedy Algorithm 3, will work for the case of fully connected graph from my observations and researches. But case in the case of Turkey Region Graph, if not all the vertices are connected with the each other due to the random selection of the vertices there could be cases that a vertice could not be colored.