Problem 1 Rewrite the method *mColoring* (call it *mColorable*) which instead of printing out all valid *m*-colorings of graph *W*, returns true if it is *m*-colorable and false if not. The method should not search for additional solutions once it has found one solution. (You may not use "System.exit").

```
boolean mColor(int n, int [][] W, int m) {…}
```

Problem 2 Using the modified boolean method *mColoring* above as an auxiliary function, write a method that finds the minimum number of colors needed to color graph *G*.

```
int minNumColorsNecessary(int n, int [][] W) {…}
```

Problem 3 Write a two-coloring algorithm that runs in time $O(n^2)$. You will need to use a Queue.

```
boolean twoColorable(int n, int [][] W) {…}
```

Problem 4 Without using recursion, write the boolean algorithm *getNextSequence*. This method takes a positive integer *n* and an integer array *A* of length *n* (indexed from 1 to *n*). All values in *A* are between 1 and *n*, inclusive. The array *A* is altered to contain the next sequence. For example, if $n = 4$ and $A = \{3, 1, 4, 4\}$, the array *A* would be changed to {3, 2, 1, 1} and it would return true. If $n = 4$ and $A = \{4, 4, 4, 4\}$, the method returns the value false.

```
boolean getNextSequence(int n, int [] A)   {
```

Problem 5 You are given a graph *W* on *n* vertices and an array *vcolor*. Return true if no two adjacent vertices are colored the same. No recursion.

```
boolean validColoring(int n, int [][] W, int [] vcolor)   {
```

Problem 6 Write an algorithm that takes a positive integer *n*, an integer array *A* indexed from 1 to *n*. Assume that each of the values in *A* is between 1 and *n*, inclusive. Your algorithm must return the number of distinct values in *A*. You may NOT call auxiliary methods. You may NOT alter the array *A* in your algorithm. No recursion allowed.

```
int howManyDistinctValues(int n, int [] A)    {
```

Problem 7 Write a brute force algorithm with NO RECURSION which takes a graph *W* on *n* vertices and returns the minimium number of colors needed to make a valid coloring of *W*. You may call any of the methods in Problems 8, 9, and 10.

```
int minNumColorsNeeded (int n, boolean [][] W)   {
```