Problem 4 (15 pts) Write the method

```
void nQueens_Print_k_solns_Rec(int i, int [] col, int n, int k)
```

This is a variation of the *n*Queens method, where an additional integer parameter *k* is passed in, and this method prints only the first *k* solutions. (If there are fewer than *k* solutions, then all the solutions are printed.) ***Once the kth solution is found and printed, the recursive method must not look for any further solutions***.

This method is called initially with $i = 0$, and an array *col* indexed from 1 to *n* to store the columns of the queens in rows 1 through *n*.

For this problem, you ***MAY use*** `col[0]` to keep a count of the number of solutions reached so far. You may NOT USE `System.exit` to halt your program once *k* solutions have been printed.

Problem 5 (15 pts) To receive ANY credit for this problem, you may ***not use global variables***, and you may ***NOT access*** `col[0]`. Write the method

```
int nQueens_numSolns_Rec(int i, int [] col, int n)
```

The return value of `nQueens_numSolns` is the ***number of solutions to the*** *n*Queens problem for the value of *n* passed in. This recursive method does not print out any of the solutions, it just returns the number of solutions.

This method is called by initially with $i = 0$, and an array *col* indexed from 1 to *n* to store the columns of the queens in rows 1 through *n*.

Problem 6 (15 pts) Write the method

```
int greedyColoring(int n, int [][] W, int [] vcolor)
```

which implements the greedy graph coloring algorithm that we discussed in class and that you traced through in the first three problems. (Recall that this is the same idea as the backtracking algorithm, however there is no backtracking—no number of colors *m* is passed in—you "color" each vertex in succession with the lowest number possible. Note: Arrays *vcolor* and *W* are indexed from 1 to *n*.

(Problems 7 and 8 on next page…)

Problem 7 (10 pts) To receive credit for this problem, you may not ***not access*** `vcolor[0]`, and you may ***not access any cells in row 0 or column 0*** of array *W*.

Write the method

```
boolean mColorableRec(int i, int n, int [][] W, int m, int [] vcolor)
```

The method is called by initially with $i = 0$, and an array *vcolor* to store the colors (integers from 1 to *m*) of the *n* vertices in graph *W* (a 0-1, 2D array with rows and columns indexed from 1 to *n*).

This method does not print any solutions. If and when this method finds a solution, it ***stops looking for further solutions***, and returns **true**, with a valid *m*-coloring stored in array *vcolor*. If there is no *m*-coloring, the method returns **false**.

Problem 8 (5 pts). Write the methods in the queue class *Intqueue*. Test your *Intqueue* methods, and turn in the output. To receive any credit, you ***may NOT import anything*** (additional—I use the Scanner for testing in *main*).

Don't forget to change the filename, classname, constructor, and within *main* to have your name instead of LASTNAME_FIRSTNAME.

Problem 9 (15 pts) To receive credit for this problem, you may not ***not access*** `vcolor[0]`, and you may not access any cells in row 0 or column 0 of array *W*.

Write the method

```
boolean fastTwoColor(int n, int [][] W, int [] vcolor)
```

which implements the fast two-coloring algorithm. The method returns **true** with a 2-coloring in array *vcolor* if the graph is 2-colorable. Otherwise, it returns **false.**

This method will use the *IntQueue* class you wrote in the previous problem.
