Caleb Song

Documentation: EI from Dr. Hadfield on 2/9 and 2/12.

1.a)

/\* 43A  
Suppose we have nine identical-looking coins numbered 1 through  
9 and only one of the coins is heavier than the others. Suppose  
further that you have one balance scale and are allowed only two  
weighings. Develop a method for finding the heavier counterfeit  
coin given these constraints.  
 \*/  
public static int sort1(int[] arrayCoins){  
 //Split the coins into 3 groups and add their weights  
 int sum1 = arrayCoins[0] + arrayCoins[1] + arrayCoins[2];  
 int sum2 = arrayCoins[3] + arrayCoins[4] + arrayCoins[5];  
 int sum3 = arrayCoins[6] + arrayCoins[7] + arrayCoins[8];  
 //compare group 1 and group 2, if its greater then the heavier is in group 1  
 if (sum1 > sum2){  
 //compare 1st coin and 2nd coin in group, if its heavier return 1 as the fake one is the first coin  
 if (arrayCoins[0] > arrayCoins[1]){  
 return 1;  
 }  
 //if the first coin is lighter than the second coin then return 2 as the fake one is the second coin  
 else if (arrayCoins[0] < arrayCoins[1]){  
 return 2;  
 }  
 //else they are equal which means the last coin is the heavier one so return 3  
 else{  
 return 3;  
 }  
 }  
 //compare group 1 and group 2, if its less then the heavier is in group 2  
 else if (sum1 < sum2){  
 //compare 1st coin and 2nd coin in group, if its heavier return 4 as the fake one is the first coin  
 if (arrayCoins[3] > arrayCoins[4]){  
 return 4;  
 }  
 //if the first coin is lighter than the second coin then return 5 as the fake one is the second coin  
 else if (arrayCoins[3] < arrayCoins[4]){  
 return 5;  
 }  
 //else they are equal which means the last coin is the heavier one so return 6  
 else{  
 return 6;  
 }  
 }  
 //else last group has the heavier coin  
 else{  
 //compare 1st coin and 2nd coin in group, if its heavier return 7 as the fake one is the first coin  
 if (arrayCoins[6] > arrayCoins[7]){  
 return 7;  
 }  
 //if the first coin is lighter than the second coin then return 8 as the fake one is the second coin  
 else if (arrayCoins[6] < arrayCoins[7]){  
 return 8;  
 }  
 //else they are equal which means the last coin is the heavier one so return 9  
 else{  
 return 9;  
 }  
 }  
  
}

b)

/\*  
Suppose we now have an integer n (that represents n coins) and  
only one of the coins is heavier than the others. Suppose further  
that n is a power of 3 and you are allowed log3 n weighings to  
determine the heavier coin. Write an algorithm that solves this  
problem. Determine the time complexity of your algorithm.  
 \*/  
public static int sort2(int[] arrayCoins, int start, int end){  
 //get the number of coins  
 //split coins into 3 groups  
 int groupSize = (end+1-start)/3;  
 //base case if groupSize is 1 compare 3 coins in group  
 if (groupSize == 1){  
 if (arrayCoins[start] > arrayCoins[start+1]){  
 //return first coin in group plus 1 for 1-base indexing  
 return start+1;  
 }  
 else if (arrayCoins[start] < arrayCoins[start+1]){  
 //return second coin in group plus 1 for 1-base indexing  
 return start+2;  
 }  
 else{  
 //return third coin in group plus 1 for 1-base indexing  
 return start+3;  
 }  
  
 }  
 //recursive case, first get weights of all three groups  
 else{  
 int groupOne = *getWeight*(arrayCoins, start, start+groupSize-1);  
 int groupTwo = *getWeight*(arrayCoins, start+groupSize, start+(2\*groupSize)-1);  
 //if groupOne has a higher weight then call sort again with that smaller group  
 if (groupOne > groupTwo){  
 return *sort2*(arrayCoins, start, start+groupSize-1);  
 }  
 else if (groupOne < groupTwo){  
 return *sort2*(arrayCoins, start+groupSize, start+(2\*groupSize)-1);  
 }  
 else{  
 return *sort2*(arrayCoins, start+(2\*groupSize), end);  
 }  
  
  
 }  
  
}  
public static int getWeight(int[] coinArray, int start, int end){  
 int weight = 0;  
 //loop through to find weights  
 for (int i = start; i<=end;i++){  
 weight += coinArray[i];  
  
 }  
 return weight;  
}

2.

public static boolean searchValue(int[][] array, int value, int startRow, int startCol, int endRow, int endCol) {  
 //base case - nothing to search therefore value is not in array  
 if (startRow > endRow || startCol > endCol) {  
 return false;  
 }  
 //find middle row and column  
 int middleRow = (startRow + endRow) / 2;  
 int middleCol = (startCol + endCol) / 2;  
 //set value we are checking at the middle of the array  
 int target = array[middleRow][middleCol];  
 //base case - found value  
 if (target == value) {  
 return true;  
 }  
 else {  
 if (value > target) {  
 //search bottom left, bottom right, top right  
 return *searchValue*(array, value, middleRow+1, startCol, endRow, middleCol) ||  
 *searchValue*(array, value, middleRow, middleCol+1, endRow, endCol) ||  
 *searchValue*(array, value, startRow, middleCol+1, middleRow-1, endCol);  
 } else {  
 //search top left, bottom left, top right  
 return *searchValue*(array, value, startRow, startCol, middleRow-1, middleCol-1) ||  
 *searchValue*(array, value, middleRow, startCol, endRow, middleCol-1) ||  
 *searchValue*(array, value, startRow, middleCol, middleRow-1, endCol);  
 }  
 }  
}

3.

a)

**containsForbidden pseudocode:**

loop through startRow and startCol to endRow and endCol to see if trominoBoard array has it

containsForbidden (startR, startC, endR, endC):

loop while startR <= endR:

loop while startC <= endC:

see if forbidden is there

**doTiling pseudocode**:

find middle row and column

base case: if size ==1, fill startR and startC as black

recursive cases:

do the same thing for top right, bottom right, and bottom left. For top left

if it has the forbidden recursively call doTiling

else: set temporary forbidden, do Tiling and draw the tromino

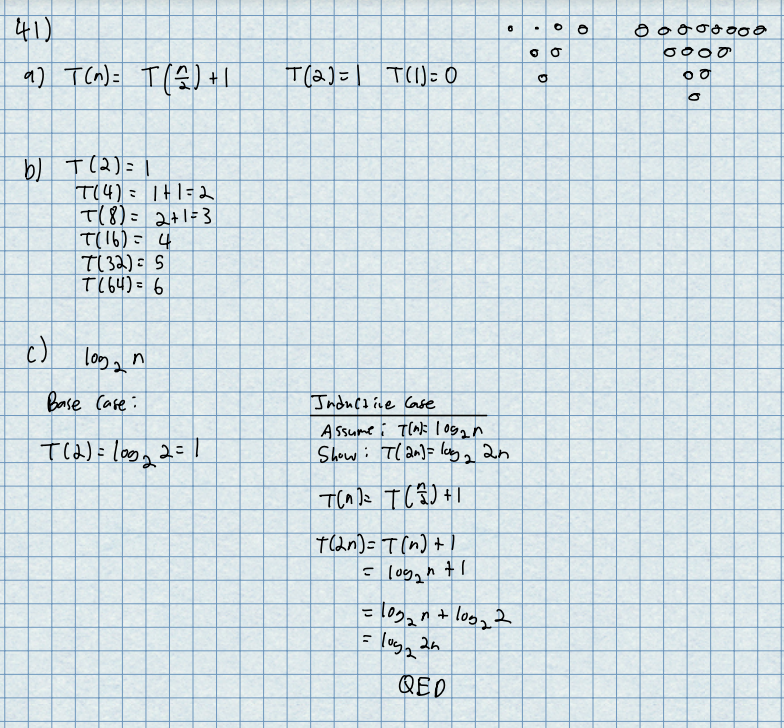
b) In provided java folder

boolean containsForbidden( int startRow, int endRow,  
 int startCol, int endCol ) {  
  
 boolean hasForbidden = false; // assume no forbidden  
  
 // YOUR CODE GOES HERE  
 for (int i = startRow; i <= endRow; i++){  
 for (int j = startCol; j <= endCol; j++){  
 if (trominoBoard[i][j] == true) {  
 hasForbidden = true;  
 return hasForbidden;  
 }  
 }  
 }  
  
 return hasForbidden; // return results  
  
} // end containsForbidden()  
  
*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* doTiling() - main recursive method that implements the divide-and-  
 \* conquer algorithm for doing the tiling.  
 \*/*void doTiling( int startRow, int endRow, int startCol, int endCol ) {  
  
 // set a random color for this tromino  
  
 Color trominoColor = new Color(rand.nextInt(255),  
 rand.nextInt(255),rand.nextInt(255));  
  
 int size = (endRow+1)-startRow; // determine sub-grid size  
  
 // YOUR CODE CODES HERE  
  
 int middleRow = (startRow+endRow)/2;  
 int middleCol = (startCol+endCol)/2;  
  
 if (size == 1){  
 g.setColor(Color.*BLACK*);  
 g.fillRect(startCol\*cellSize, startRow\*cellSize, cellSize, cellSize);  
 }  
 else{  
 //top right  
 if (containsForbidden(startRow,middleRow, middleCol+1,endCol)){  
 doTiling(startRow,middleRow,middleCol+1,endCol);  
 }  
 else{  
 setForbiddenCell(middleRow,middleCol+1);  
 doTiling(startRow,middleRow,middleCol+1,endCol);  
 g.setColor(trominoColor);  
 g.fillRect((middleCol+1)\*cellSize, middleRow\*cellSize, cellSize, cellSize);  
 panel.copyGraphicsToScreen();  
 }  
 //top left  
 if (containsForbidden(startRow,middleRow, startCol,middleCol)){  
 doTiling(startRow,middleRow, startCol,middleCol);  
 }  
 else{  
 setForbiddenCell(middleRow,middleCol);  
 doTiling(startRow,middleRow, startCol,middleCol);  
 g.setColor(trominoColor);  
 g.fillRect(middleCol\*cellSize, middleRow\*cellSize, cellSize, cellSize);  
 panel.copyGraphicsToScreen();  
 }  
 //bottom left  
 if (containsForbidden(middleRow+1,endRow, startCol,middleCol)){  
 doTiling(middleRow+1,endRow, startCol,middleCol);  
 }  
 else{  
 setForbiddenCell(middleRow+1,middleCol);  
 doTiling(middleRow+1,endRow, startCol,middleCol);  
 g.setColor(trominoColor);  
 g.fillRect((middleCol)\*cellSize, (middleRow+1)\*cellSize, cellSize, cellSize);  
 panel.copyGraphicsToScreen();  
 }  
 //bottom right  
 if (containsForbidden(middleRow+1,endRow, middleCol+1,endCol)){  
 doTiling(middleRow+1,endRow, middleCol+1,endCol);  
 }  
 else{  
 setForbiddenCell(middleRow+1,middleCol+1);  
 doTiling(middleRow+1,endRow, middleCol+1,endCol);  
 g.setColor(trominoColor);  
 g.fillRect((middleCol+1)\*cellSize, (middleRow+1)\*cellSize, cellSize, cellSize);  
 panel.copyGraphicsToScreen();  
 }  
 }  
  
} // end doTiling()

c)

Complexity

4)



5)

