**More Recursion**

1. What is wrong with these methods?
   1. public static double bad (double a, double b) {

a = a / 2;

b = b\*2;

return bad(a, b);

}

**//The recursive function never terminates; it lacks a base case**

* 1. public static int badToo (int n) {

if (n < 1)

return 0;

else if (n==1)

return 5;

else

return 2 \* badToo(n+1) + 3;

}

//This recursive function will never reach the base case, it is **increasing** at a linear rate if the input is an integer > 1

1. In the Towers of Hanoi problem, the number of moves grows rapidly as the number of disks increases. By following the algorithm given in class, write a recursive method countMoves that takes in the size of the tower and return the number of moves.

| /\*  \* Program name: TowersOfHanoi.java  \*  \* By: Lucas Chow (Last edited: 2022-10-30)  \*  \* ICS4U1: More Recursion  \*  \* This class features countCases method, which calculates the minimum number of moves for the  \* towers of hanoi problem  \*  \*  \*/  import java.util.Scanner;  public class TowersOfHanoi{  /\*  \* Relating to the towers of hanoi problem, given a tower of n disc high,  \* calculate the minimum number of moves needed to move the disc to the third pole,  \* given that disc of bigger radius cannot stack on disc of smaller radius  \*  \* int countCases(int towerSize)  \*  \* int towerSize -> the inputed number of disc on the base tower  \*  \* returns int - the minimum number of moves required  \*  \*/  public static int countCases(int towerSize)  {  if (towerSize == 1)  {  return 1;  }  else  {  return (int) (Math.pow(2,towerSize-1)+countCases(towerSize-1));  }  }  public static void main(String[] args)  {  //prompting user for tower size  int inputTowerSize;  Scanner sc = new Scanner(System.in);  System.out.print("Enter the number of disc: ");  inputTowerSize = sc.nextInt();  System.out.println("A tower of hanoi of "+inputTowerSize+" disc would take a minimum of "+countCases(inputTowerSize)+ " moves");  sc.close();  }  } |
| --- |

* 1. Give a recursive description of the process of printing a row containing n asterisks.

| public static void printAsterick(int inputN)  {  System.out.print("\*");  if (inputN != 1)  {  printAsterick(inputN-1);  }    } |
| --- |

* 1. Complete the definition of a recursive method printRow whose header is shown below. The method should print a line containing a row of n asterisks. If n is less than one, the method should print nothing.

public static void printRow (int n)

| public static void printRow (int n)  {  if (n > 0)  {    if (n != 1)  {  System.out.print("\*");  printRow(n-1);  }  else if (n == 1)  {  System.out.print("\*");  }  }  } |
| --- |

1. Suppose that the following pattern is called a *5-triangle*

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

* 1. Give a recursive description of the process of printing an *n-triangle*

| /\*  \* This program takes a different intuitive approach, knowing that the nTriangle method  \* won't print out the "last" line of 5 "\*" (\*\*\*\*\*), until it prints n-1, n-2... until n > 1  \*  \* int n -> the input  \*  \*/  public static void nTriangle (int n)  {  if (n > 0)  {  if (n > 1)  {  nTriangle(n-1);  }  System.out.println("\*".repeat(n));  }  } |
| --- |

* 1. Write a recursive method printTriangle with a single int parameter n. The method should print a triangle of asterisks like the one shown here but containing n rows. If n is less than one, the method should print nothing.

| /\*  \* This program takes a different intuitive approach, knowing that the nTriangle method  \* won't print out the "last" line of 5 "\*" (\*\*\*\*\*), until it prints n-1, n-2... until n > 1  \*  \* int n -> the input  \*  \*/  public static void printTriangle(int n)  {  if (n > 0)  {  if (n > 1)  {  printTriangle(n-1);  }  System.out.println("\*".repeat(n));  }  } |
| --- |

1. 1. Write a recursive function with the following header to determine the largest value of a list of values. The parameter list is the array represent the list of integers, start represents the index of the start of the list and end represents the index of the end of the list.

largest(int[] list, int start, int end)

| public static int largest(int[] list, int start, int end)  {  //1 element array  if (end == 1)  {  return list[0];  }    return Math.max(list[end-1], largest(list, start, end-1));  } |
| --- |

* 1. Write a wrapper method with the following header for the recursive method above.

largest(int[] list)

| int largest(int[] list)  {  return largest(list,list.length-1)  }  largest(int[] list, int end)  {  if (end == 1)  {  return list[0];  }    return Math.max(list[end-1], largest(list, start, end-1));  } |
| --- |