Recursive Function

A breif recap

- Recursive functions in Java call themselves and are used to solve problems that can be broken down into smaller subproblems.
- They have a base case that eventually stops the recursion.
- Examples of problems that can be solved with recursive functions include factorials, Fibonacci sequences, and binary search.

Critical Thinking

- What is the purpose of the following recursive function?

```java
public static int mystery(int a, int b) {
    if (b == 0) {
        return a;
    } else {
        return mystery(a, b -1) + 1;
    }
}
```

answer: This function returns the sum of a and b.

- I want this function to calculate the multiplication of positive integers a and b. What should be in third line?

```java
public static int multiplication(int a, int b) {
    if (a == 1) {
        // what should be here?
    } else {
        return multiplication(a, b -1) + b;
    }
}
```

answer: return b;

- what will happen if I call a()?

```java
public static void a() {
    b();
}
public static void b() {
    a();
}
```

answer: The program will crash because the function will call itself infinitely. (it crash because system run out memory)

- What will happen when we call a(n) with a non-negative n value?

```
static void a(int n) {
    if(n==0)
        System.out.println("ends in a()");
    else
        b(n-1);
    }
static void b(int n) {
    if(n==0)
        System.out.println("ends in b()");
    else
        a(n-1);
}
```

answer: It ends in a() if n is even, and ends in b() if n is odd.

Let's say n=5

a(5) –> b(4) –> a(3) –> b(2) –> a(1) –> b(0) prints "ends in b()"

- In previous question, what will happen when we call b(n) with a non-negative n value?

answer: infinite recursion

Practice

- Write a recursive function that takes two non-negative integers "n" and "m" and computes the power n^m. you are not allowed to use "for" or "while".

  example:

  - power(2, 3) –> 8

Main method:

```
public static void main(String[] args) {
    System.out.println(power(2, 3));
}
```

soloution1:

```
public static int pow(int a, int b) {
    if(b==1)
        return a;
    else
        return pow(a, b-1) * a;
}
```

soloution2:

```
public static int power(int n, int m) {
    if (m == 0) {
        return 1;
    } else {
        return n * power(n, m - 1);
    }
}
```

- Write a recursive function which takes two positive integer arguments n,m and returns n % m. Don't use %, *, / operators.

  example:

  - mod(3, 2) –> 1

Main method:

```
public static void main(String[] args) {
    System.out.println(mod(3, 2));
}
```

soloution:

```
public static int mod(int a, int b) {
    if(a<b)
        return a;
    else
        return mod(a-b, b);
}
```

- Write a recursive function which takes a string parameter and checks if all its characters appear only once.

  example:

  - isUnique("pickle") –> true
  - isUnique("moon") –> false
  - isUnique("trash") –> true

Main method:

```
public static void main(String[] args) {
    System.out.println(isUnique("pickle"));
    System.out.println(isUnique("moon"));
    System.out.println(isUnique("trash"));
}
```

soloution:

```
public static boolean isUnique(String s) {
    if(s.length() == 1)
        return true;
    else {
        for(int i=1; i<s.length(); i++)
```

```
            if(s.charAt(0) == s.charAt(i))
                return false;
        return isUnique(s.substring(1));
    }
}
```

- Write a method that takes three integer arguments and returns their maximum. (You can use Math.max() function)

Main method:

```
public static void main(String[] args) {
    System.out.println(maxThree(1, 2, 3));
    System.out.println(maxThree(3, 2, 1));
    System.out.println(maxThree(2, 3, 1));
}
```

soloution:

```
public static int maxThree(int a, int b, int c) {
    return Math.max(a, Math.max(b, c));
}
```

Project

1. Write a recursive function to calculate the factorial of a number.

main method:

```
public static void main(String[] args) {
    System.out.println(factorial(5));
}
```

soloution:

```
public static int factorial(int n) {
    if(n==1)
        return 1;
    else
        return factorial(n-1) * n;
}
```

2. Write a recursive function to find the nth number in the Fibonacci sequence.

main method:

```
public static void main(String[] args) {
    System.out.println(fibonacci(5));
}
```

soloution:

```
// the index of the first number in the sequence is 0
public static int fibonacci(int n) {
    if(n==1 || n==2)
        return 1;
```

```
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

3. Write a recursive function to calculate the sum of an array of integers.

main method:

```
public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5};
    System.out.println(sum(arr));
}
```

soloution:

```
public static int sum(int[] arr) {
    if(arr.length == 1)
        return arr[0];
    else {
        int[] newArr = new int[arr.length-1];
        for(int i=1; i<arr.length; i++)
            newArr[i-1] = arr[i];
        return arr[0] + sum(newArr);
    }
}
```

4. Write a recursive function to reverse a string.

main method:

```
public static void main(String[] args) {
    System.out.println(reverse("hello"));
}
```

soloution:

```
public static String reverse(String s) {
    if(s.length() < 2)
        return s;
    else
        return s.charAt(s.length()-1) + reverse(s.substring(0, s.length()-1));
}
```

5. Write a recursive function to find the maximum value in an array of integers.

main method:

```
public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5};
    System.out.println(max(arr));
}
```

soloution:

```
public static int max(int[] arr) {
    if(arr.length == 1)
```

```
        return arr[0];
    else {
        int[] newArr = new int[arr.length-1];
        for(int i=1; i<arr.length; i++)
            newArr[i-1] = arr[i];
        return Math.max(arr[0], max(newArr));
    }
}
```

6. Write a recursive function to check if a given string is a palindrome.

main method:

```
public static void main(String[] args) {
    System.out.println(isPalindrome("racecar"));
    System.out.println(isPalindrome("hello"));
}
```

soloution:

```
public static boolean isPalindrome(String s) {
    if(s.length() == 1 || s.length() == 0)
        return true;
    else if(s.charAt(0) == s.charAt(s.length()-1))
        return isPalindrome(s.substring(1, s.length()-1));
    else
        return false;
}
```

7. Write a recursive function to count the number of occurrences of a given character in a string.

main method:

```
public static void main(String[] args) {
    System.out.println(count("hello", 'l'));
}
```

soloution:

```
public static int count(String s, char c) {
    if(s.length() == 0)
        return 0;
    else if(s.charAt(0) == c)
        return 1 + count(s.substring(1), c);
    else
        return count(s.substring(1), c);
}
```

8. Write a recursive function to find the greatest common divisor (GCD) of two numbers.

main method:

```
public static void main(String[] args) {
    System.out.println(gcd(12, 18));
}
```

soloution:

```
public static int gcd(int a, int b) {
    if(a == b)
        return a;
    else if(a > b)
        return gcd(a-b, b);
    else
        return gcd(a, b-a);
}
```

9. Write a recursive function to merge two sorted arrays into a single sorted array.

main method:

```
public static void main(String[] args) {
    int[] arr1 = {1, 3, 5, 7, 9};
    int[] arr2 = {2, 4, 6, 8, 10};
    int[] arr = merge(arr1, arr2);
    for(int i=0; i<arr.length; i++)
        System.out.print(arr[i] + " ");
}
```

soloution:

```
public static int[] merge(int[] arr1, int[] arr2) {
    int[] arr = new int[arr1.length + arr2.length];
    int i=0, j=0, k=0;
    while(i<arr1.length && j<arr2.length) {
        if(arr1[i] < arr2[j])
            arr[k++] = arr1[i++];
        else
            arr[k++] = arr2[j++];
    }
    while(i<arr1.length)
        arr[k++] = arr1[i++];
    while(j<arr2.length)
        arr[k++] = arr2[j++];
    return arr;
}
```

10. write a recursive function to sort an array using merge sort

main method:

```
public static void main(String[] args) {
    int[] arr = {5, 4, 3, 2, 1};
    int[] sorted = mergeSort(arr);
    for(int i=0; i<sorted.length; i++)
        System.out.print(sorted[i] + " ");
}
```

soloution:

```
public static int[] mergeSort(int[] arr) {
    if(arr.length == 1)
        return arr;
    else {
        int[] arr1 = new int[arr.length/2];
        int[] arr2 = new int[arr.length - arr.length/2];
        for(int i=0; i<arr.length/2; i++)
            arr1[i] = arr[i];
        for(int i=arr.length/2; i<arr.length; i++)
            arr2[i-arr.length/2] = arr[i];
        return merge(mergeSort(arr1), mergeSort(arr2));
        // the merge function is the same as the one in the previous question
    }
}
```

Extra

- write a recursive function to find the the value of the nth row and kth column in pascals triangle

main method:

```
public static void main(String[] args) {
    System.out.println(pascalValue(4, 2));
}
```

soloution:

```
public static int pascalValue(int row, int column) {
    if (row == 0 || column == 0 || row == column) {
        return 1;
    } else {
        return pascalValue(row - 1, column - 1) + pascalValue(row - 1, column);
    }
}
```

In this implementation, the pascalValue function takes two parameters: row, the row of the value; and column, the column of the value.

The index of the first row is 0, and the index of the first column of each row is 0.

- write a java function to solve tower of hanoi problem

The Tower of Hanoi is a classic puzzle or mathematical problem that involves moving a set of disks from one pole or peg to another, with the constraint that only one disk can be moved at a time, and no larger disk can be placed on top of a smaller disk.

The puzzle consists of three poles or pegs and a set of disks of different sizes. The disks are initially stacked in decreasing size order, with the largest disk at the bottom and the smallest at the top, on one of the poles. The goal is to move the entire stack of disks to another pole, while following the rules mentioned above.

The Tower of Hanoi problem has a recursive solution, which involves breaking down the problem into smaller subproblems, each of which can be solved recursively. The recursive solution involves moving n-1 disks from the starting pole to the auxiliary pole, moving the remaining largest disk from the starting pole to the destination pole, and then moving the n-1 disks from the auxiliary pole to the destination pole. This process is repeated recursively until all the disks are moved to the destination pole

main method:

```
public static void main(String[] args) {
    towerOfHanoi(3, 'A', 'C', 'B');
}
```

soloution:

```
public static void towerOfHanoi(int n, char from, char to, char aux) {
    if(n == 1)
        System.out.println("Move disk 1 from rod " + from + " to rod " + to);
    else {
        towerOfHanoi(n-1, from, aux, to);
        System.out.println("Move disk" + n + " from rod " + from + " to rod " + to);
        towerOfHanoi(n-1, aux, to, from);
    }
}
```

In this implementation, the towerOfHanoi function takes three parameters: n, the number of disks to move; fromRod, the starting rod; toRod, the destination rod; and auxRod, the auxiliary rod.

The function uses recursion to move the disks. If n is 1, the function simply moves the top disk from the fromRod to the toRod. Otherwise, it recursively moves n-1 disks from the fromRod to the auxRod, then moves the nth disk from the fromRod to the toRod, and finally recursively moves the n-1 disks from the auxRod to the toRod.

- Prove that weird(n) returns 1 for all positive integers n.

```
public static int weird(int n) {
    if(n==1)
        return 1;
    else if(n%2 == 0)
        return weird(n/2);
    else
        return weird(n+1);
}
```

main method:

```
public static void main(String[] args) {
    System.out.println(weird(5));
}
```

- It is a famous conjecture in mathematics that the following function weirder(n) returns 1 for all positive integers n. No one has been able to prove it so far. Simple-looking recursive functions may exhibit complex behavior.

```
public static int weirder(int n) {
    if(n==1)
        return 1;
    else if(n%2 == 0)
        return weirder(n/2);
    else
        return weirder(3*n+1);
}
```

main method:

```
public static void main(String[] args) {
    System.out.println(weirder(5));
}
```