CS206

Various Review

Recursion

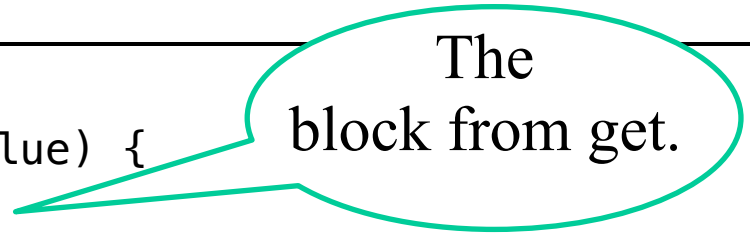
Hashtable Get

```
public V get(K key) {
        int hashVal = stringHasher(key.toString());
        int qnum = 0;
        int nVal = hashVal;
        while (backingArray[nVal] != null && !backingArray[nVal].key.equals(key)) {
            qnum++;
            nVal = (hashVal + qnum * qnum) % backingArray.length;
        }
        if (backingArray[nVal]==null)
            return null;
        return backingArray[nVal].value;
    }
```

quadratic prober

a
really useful block of code.
So make it a method

Hashtable Put




```
public void put(K key, V value) {
    int nVal = getLoc(key);
    backingArray[nVal] = new Pair<K,V>(key, value);
    if (backingArray[nVal] == null) {
        itemCount++;
        if (itemCount > (MAX_OCCUPANCY * backingArray.length)) {
            rehash((int)(backingArray.length * GROWTH_RATE));
        }
        return;
    }
}
```

Special Merge v1

```
public ArrayList<String> doMerge1(String[] a1, String[] a2) {
    ArrayList<String> ret = new ArrayList<>();

    for (int i = 0; i < 1; i++) {
        if (i < a1.length)
            ret.add(a1[i]);
    }
    for (int i = 0; i < 1; i++) {
        if (i < a2.length)
            ret.add(a2[i]);
    }

    for (int i = 1; i < 3; i++) {
        if (i < a1.length)
            ret.add(a1[i]);
    }
    for (int i = 1; i < 3; i++) {
        if (i < a2.length)
            ret.add(a2[i]);
    }
```

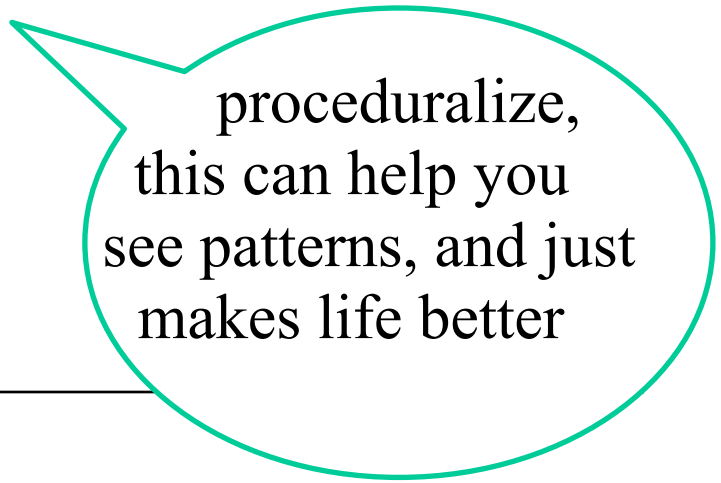
Start by writing what you know how to do. Ignore efficiency or completeness

etc

Special Merge v2

```
private void mergeHelper(int strt, int count, String[] arr,
ArrayList<String> m) {
        for (int i = strt; i < strt + count; i++) {
            if (i < arr.length)
                m.add(arr[i]);
        }
    }

    public ArrayList<String> doMerge2(String[] a1, String[] a2) {
        ArrayList<String> ret = new ArrayList<>();
        mergeHelper(0, 1, a1, ret);
        mergeHelper(0, 1, a2, ret);
        mergeHelper(1, 2, a1, ret);
        mergeHelper(1, 2, a2, ret);
        mergeHelper(3, 3, a1, ret);
        mergeHelper(3, 3, a2, ret);
        return ret;
    }
```



Special Merge v3

```
public ArrayList<String> doMerge3(String[] a1, String[] a2) {
     ArrayList<String> ret = new ArrayList<>();
     int ii = 0;
     int mx = a1.length;
     if (a2.length > mx)
         mx = a2.length;
     for (int j = 1; ii < mx; j++) {
         mergeHelper(ii, j, a1, ret);
         mergeHelper(ii, j, a2, ret);
         ii += j;
     }
     return ret;
 }
```

Recursion

Any method that calls itself, either directly or indirectly

Idea, take a problem,
break that problem down into a slightly simpler problem,
ask yourself to solve that slightly simpler problem,
repeat

STOPPING Recursion

Importantly, need a way to stop

```
public void badRecurse(int c) {
  System.out.println("A" + c);
  badRecurse(c-1);
}

public void okRecurse(int c){
  System.out.println("OK" + c);
  if (c==0) return;
  okRecurse(c-1);
}

public void goodRecurse(int c) {
  System.out.println("B" + c);
  if (c<=0) return;
  goodRecurse(c-1);
}
```

Class Recurser

The Factorial

- Recursive definition: $f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot f(n-1) & else \end{cases}$

- Java method

```
public int factorial(int n) {
  if (n<=0)
    return 1;
  else
    return n*factorial(n-1)
}
```

Recursive Method

- Base case(s):
  - no recursive calls are performed
  - every chain of recursive calls must reach a base case eventually
- Recursive calls:
  - Calls to the same method in a way that progress is made towards a base case

Compiled Code

```
1.   void main() {
2.     int A = 10;
3.     int B = factorial(5);
4.     System.out.println(B);
5.   }
```

```
1.   int factorial(int n) {
2.     if (n == 1) {
3.       return 1;
4.     } else {
5.       int F = n *
     factorial(n-1);
6.       return F;
7.     }
8.   }
```

Executing Function

Call Stack

Compiled Code

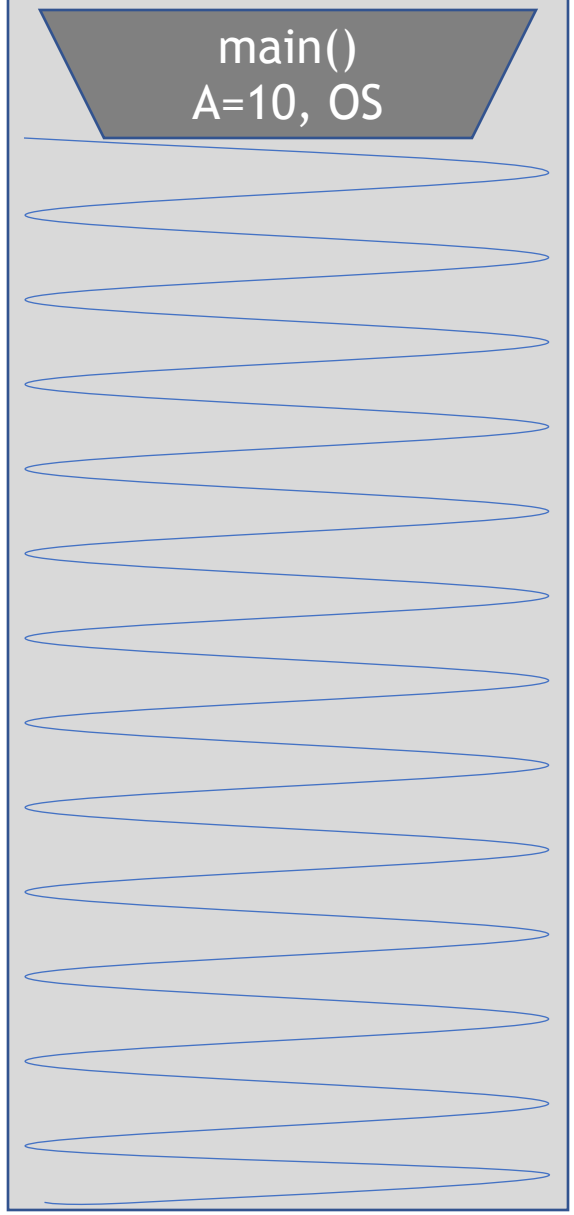
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

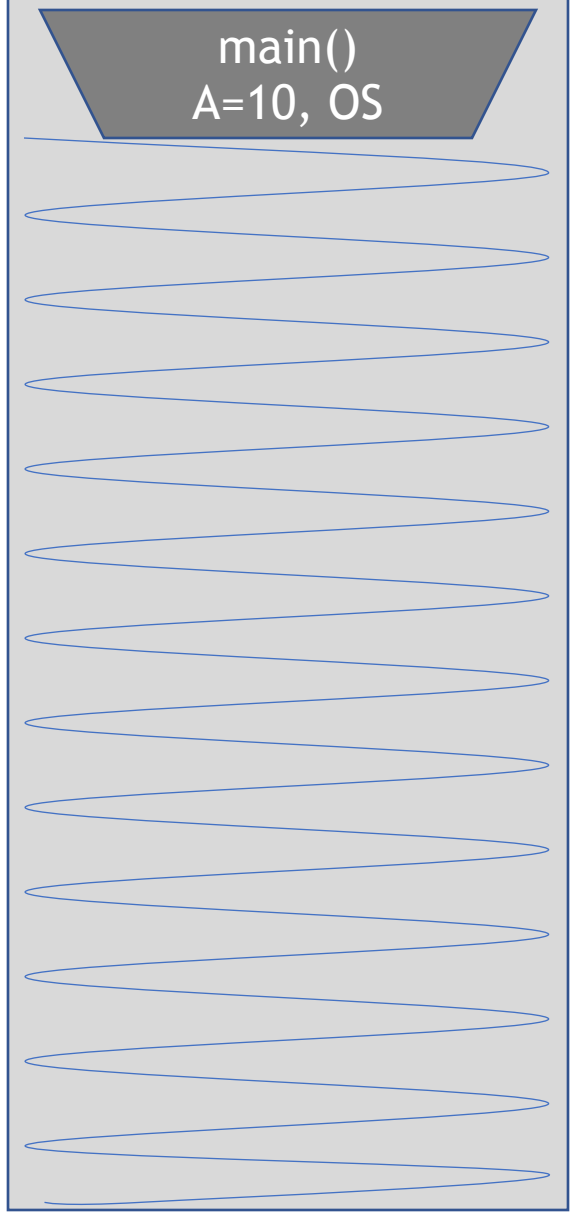
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

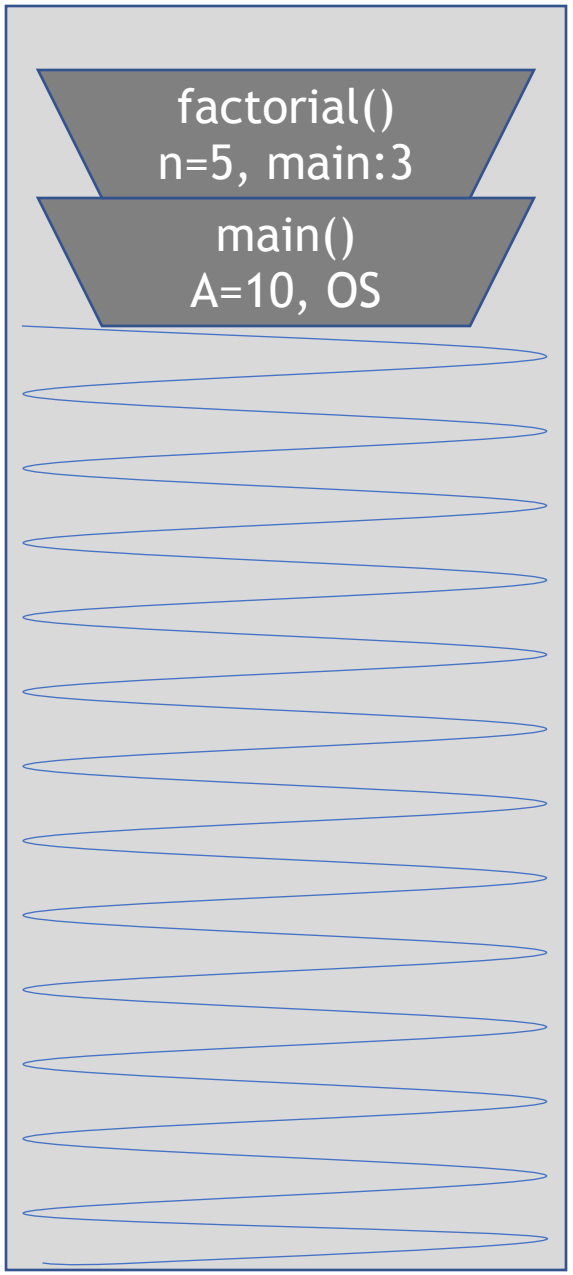
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

Call Stack



Compiled Code

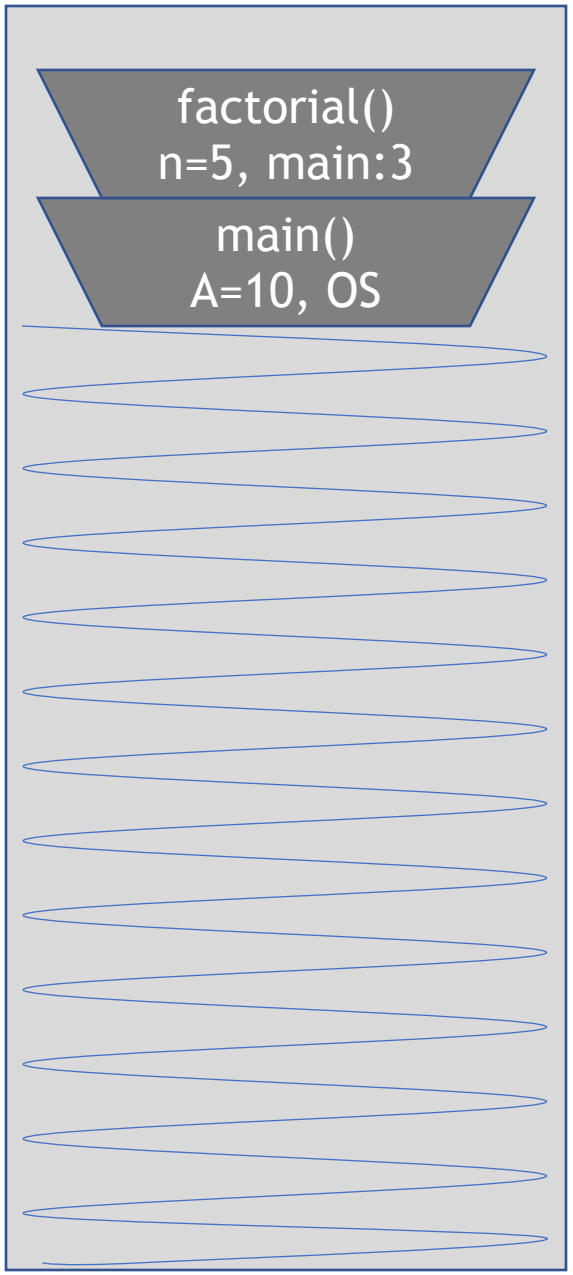
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

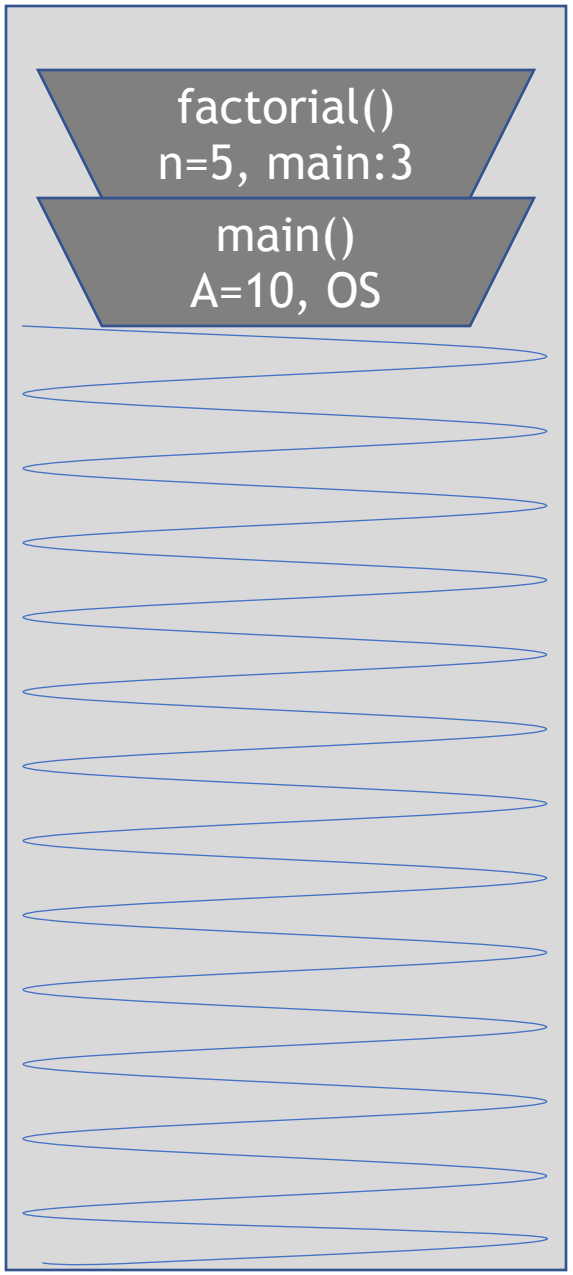
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

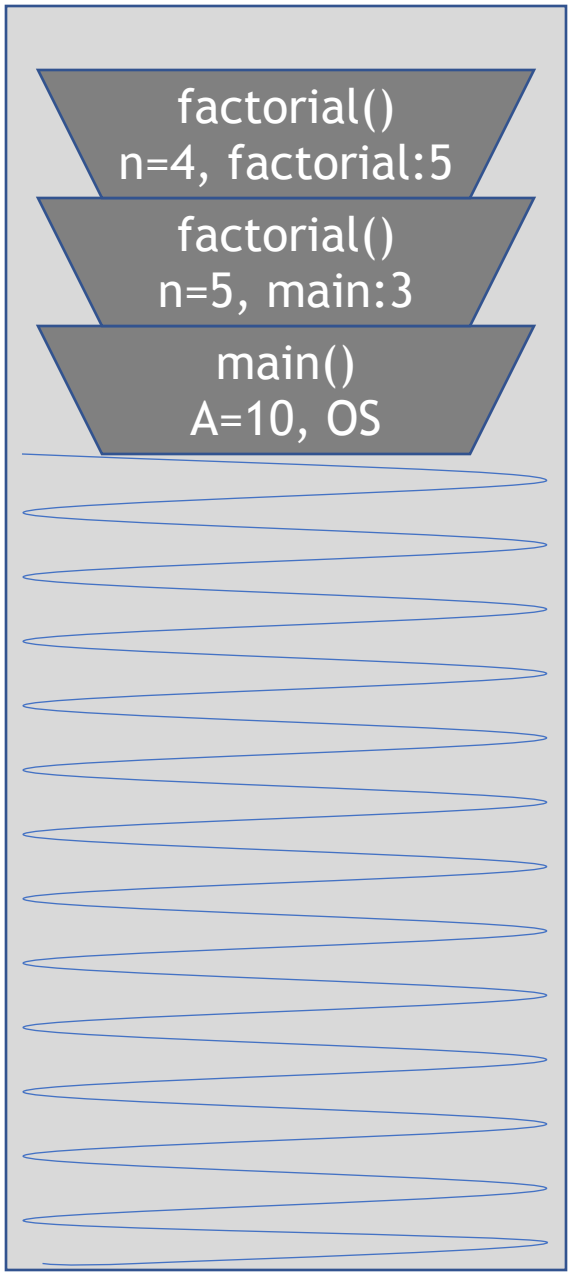
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

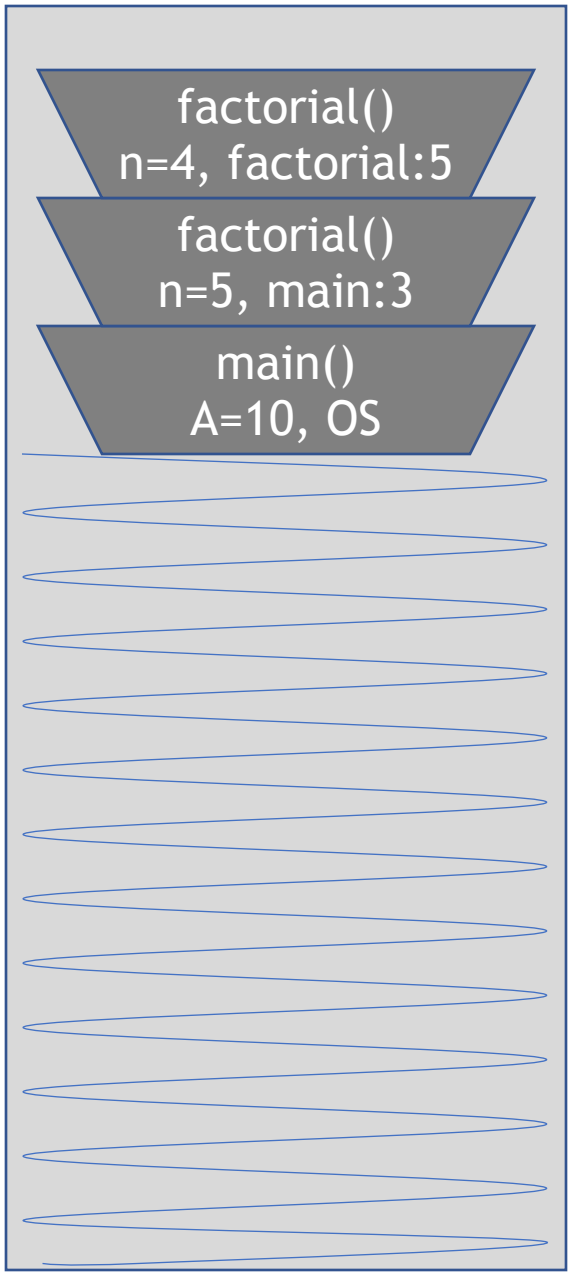
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

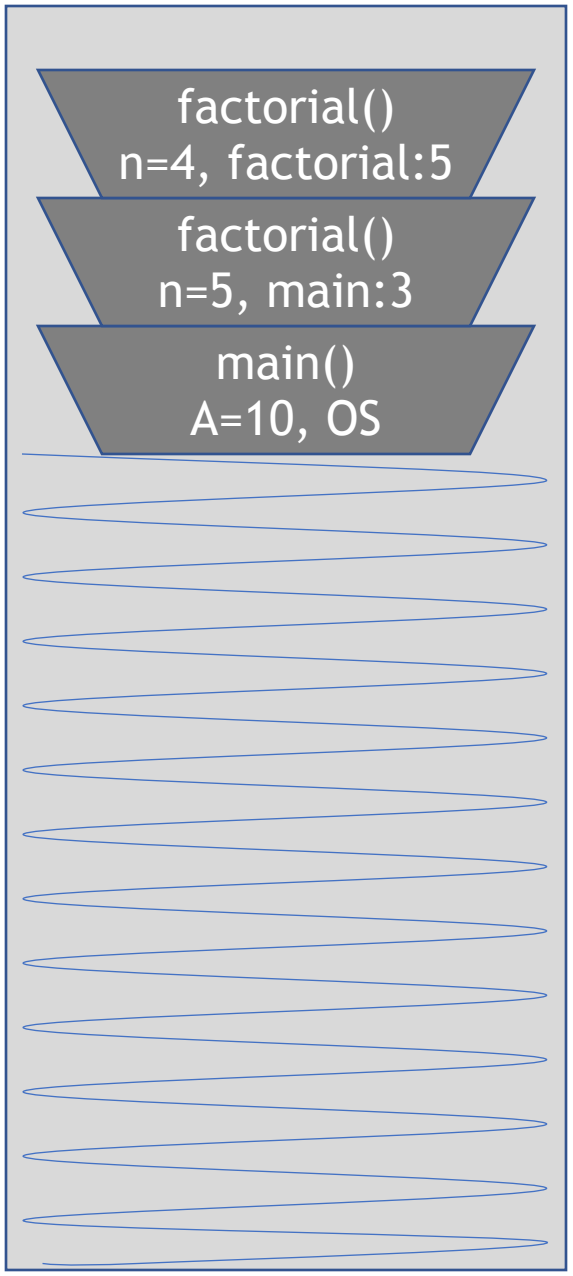
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

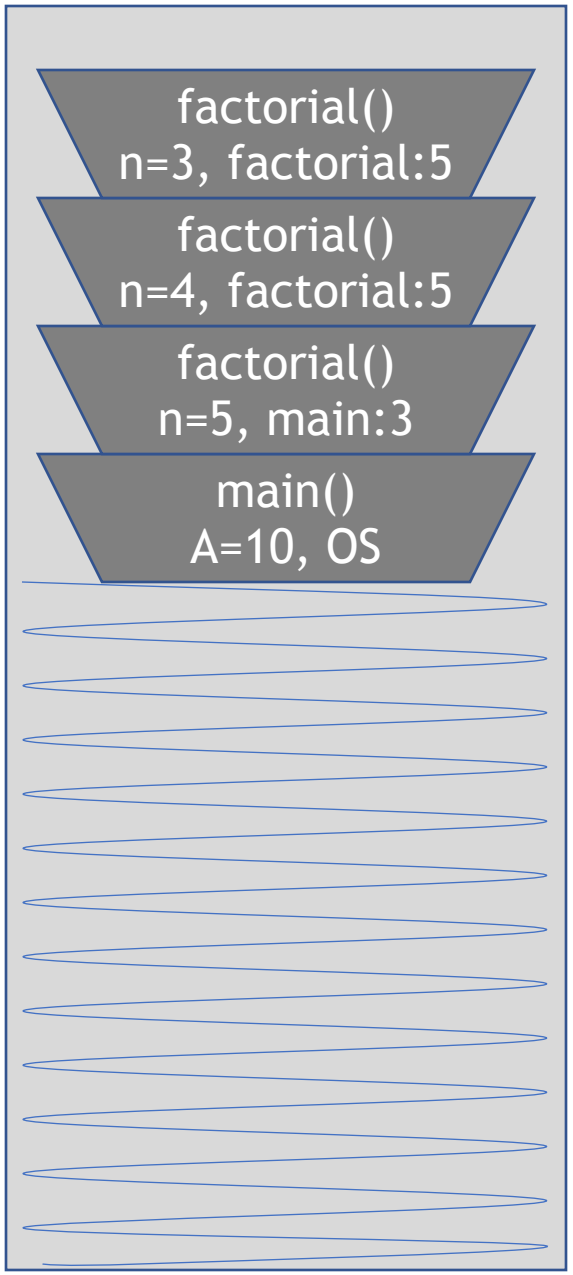
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

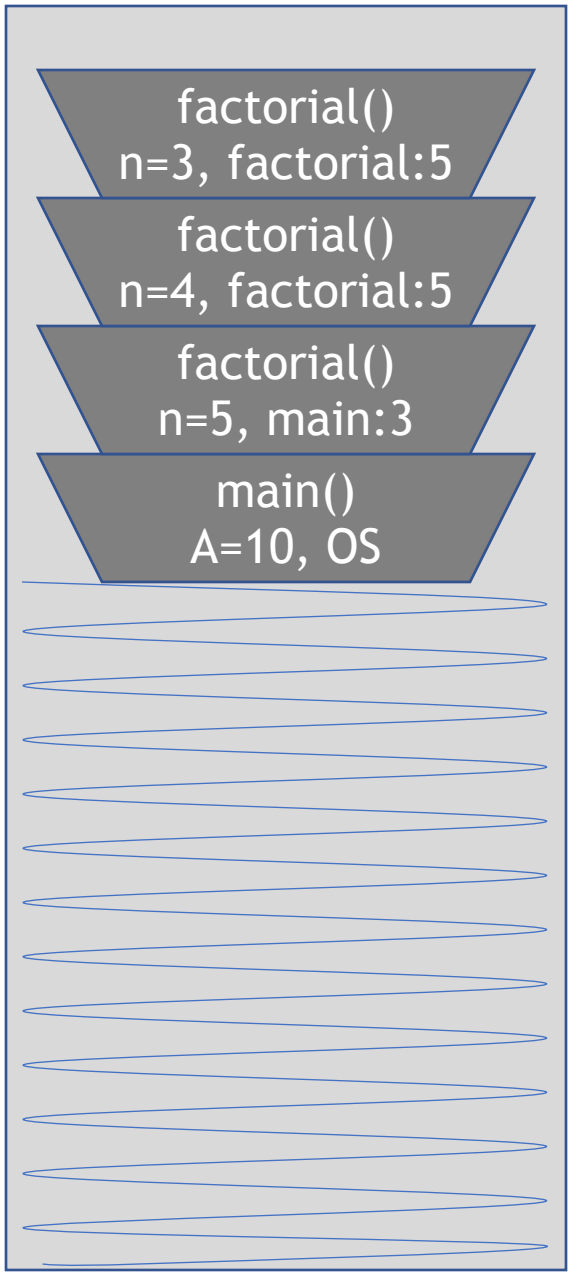
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
   int factorial(int n=3) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int F = n *
    factorial(n-1);
6.     return F;
7.   }
8.  }
```

Call Stack



Compiled Code

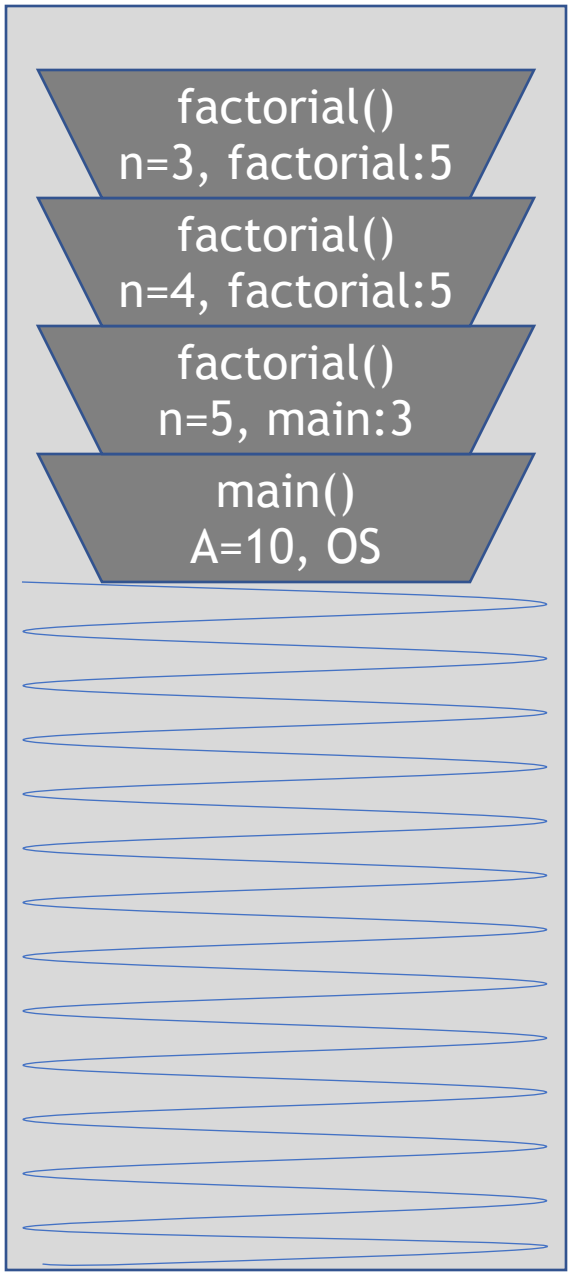
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

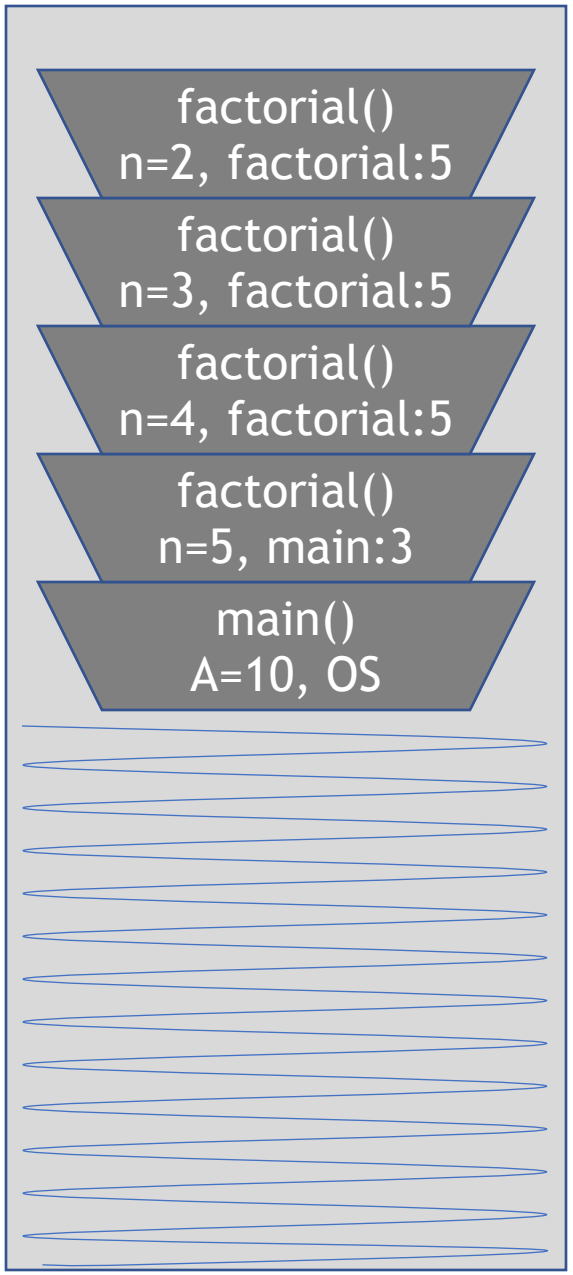
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

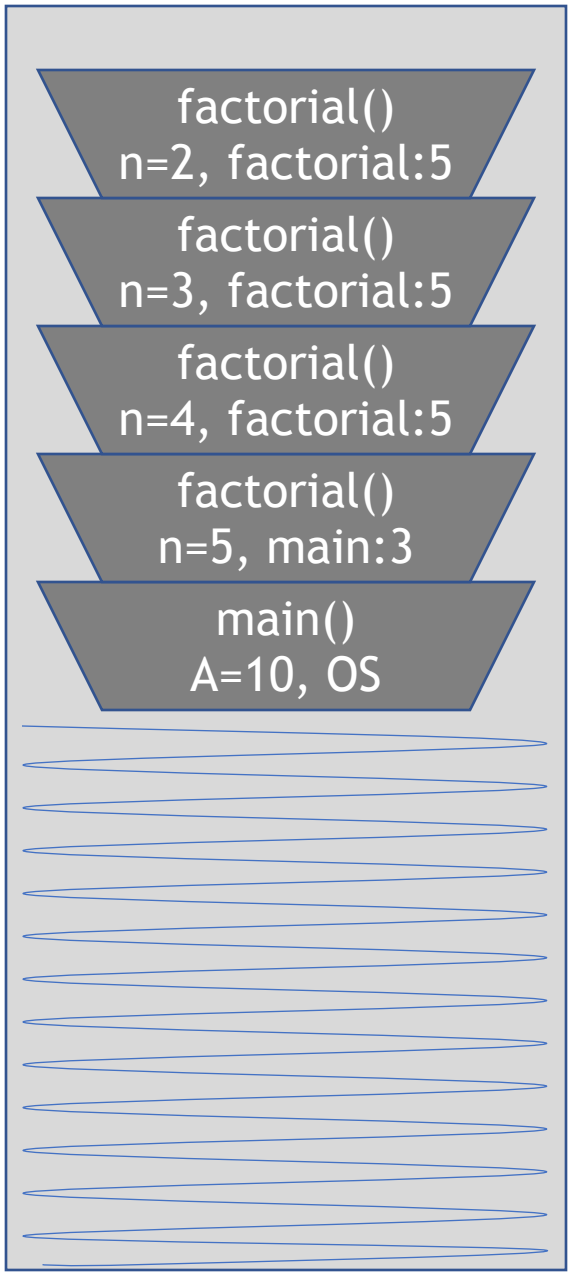
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

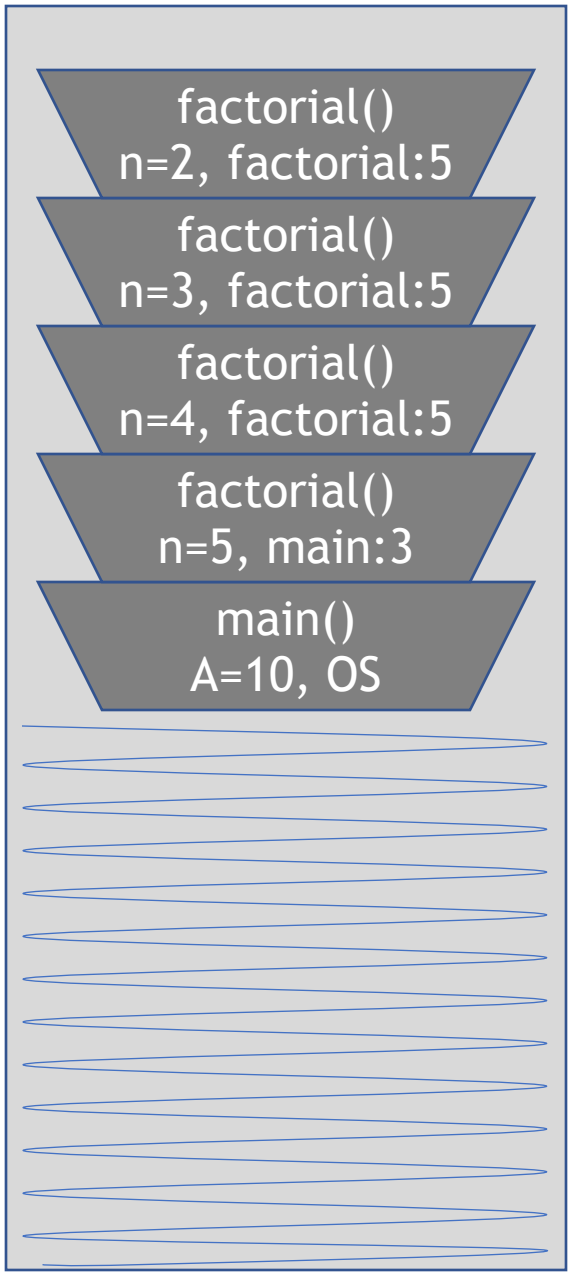
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



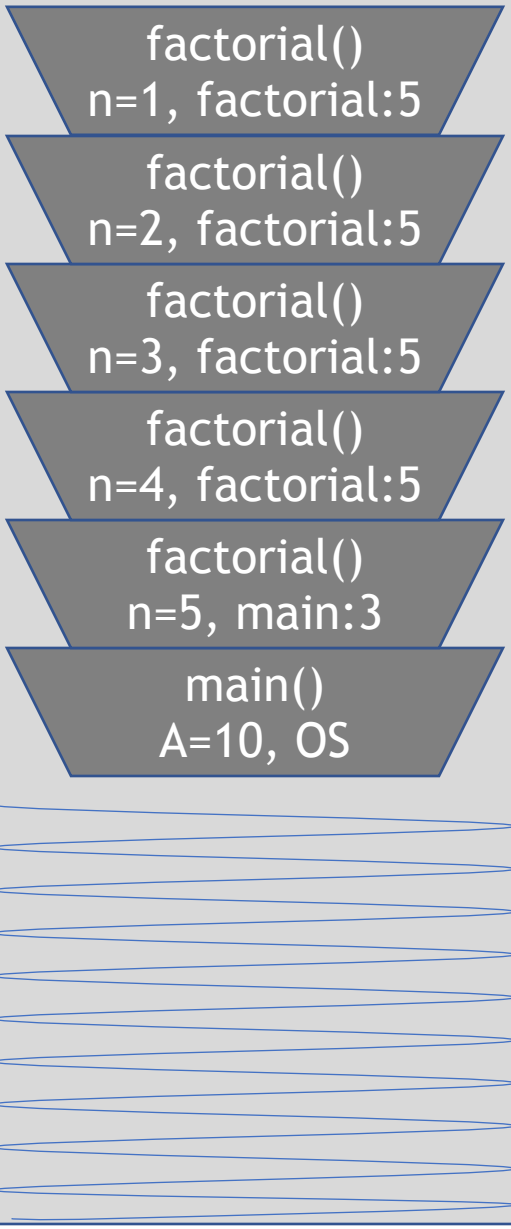
Compiled Code

```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```



Compiled Code

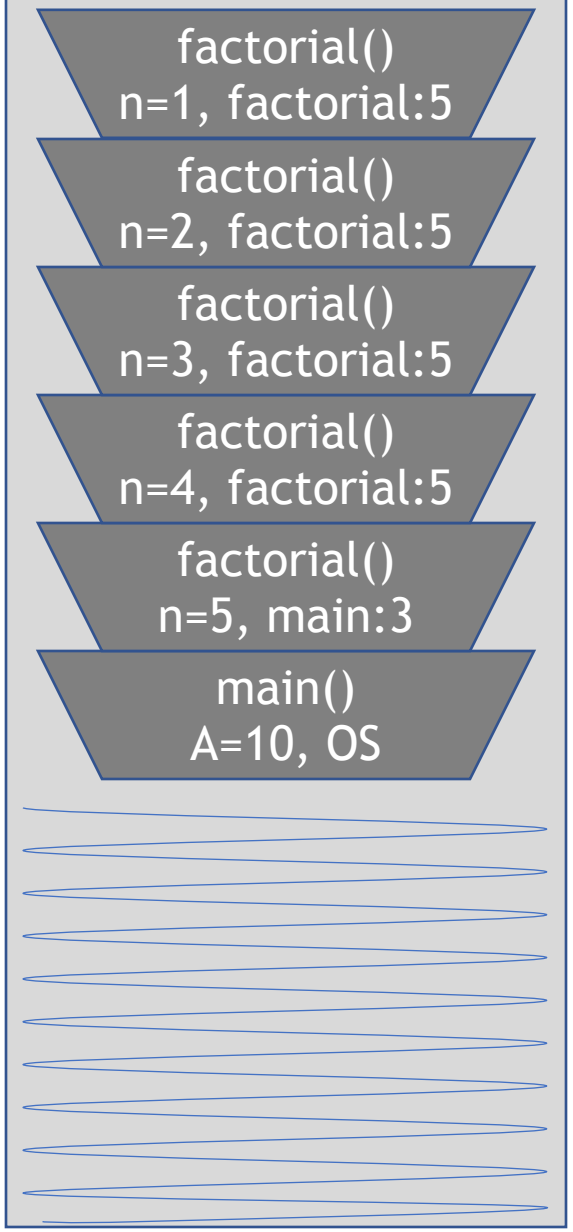
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=1) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

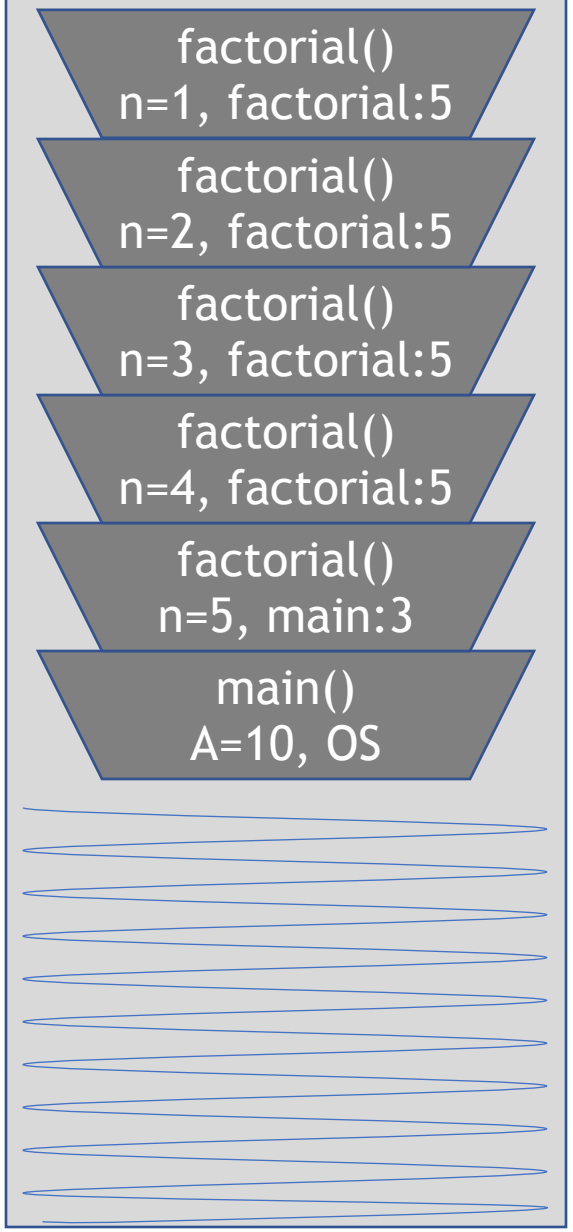
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=1) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

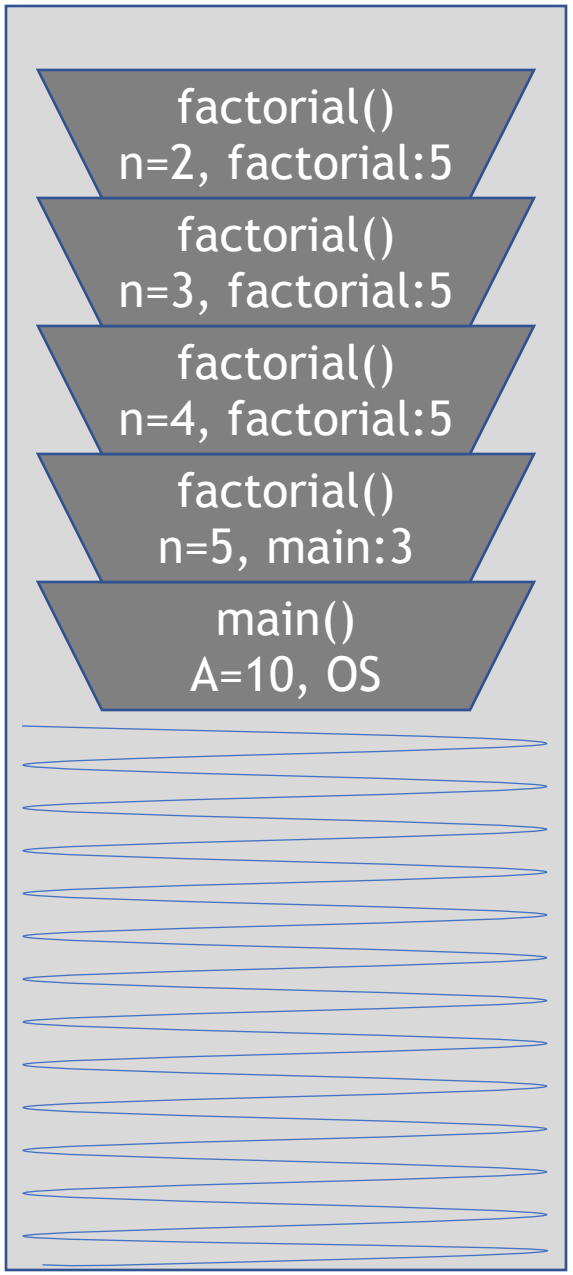
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=2) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 1;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

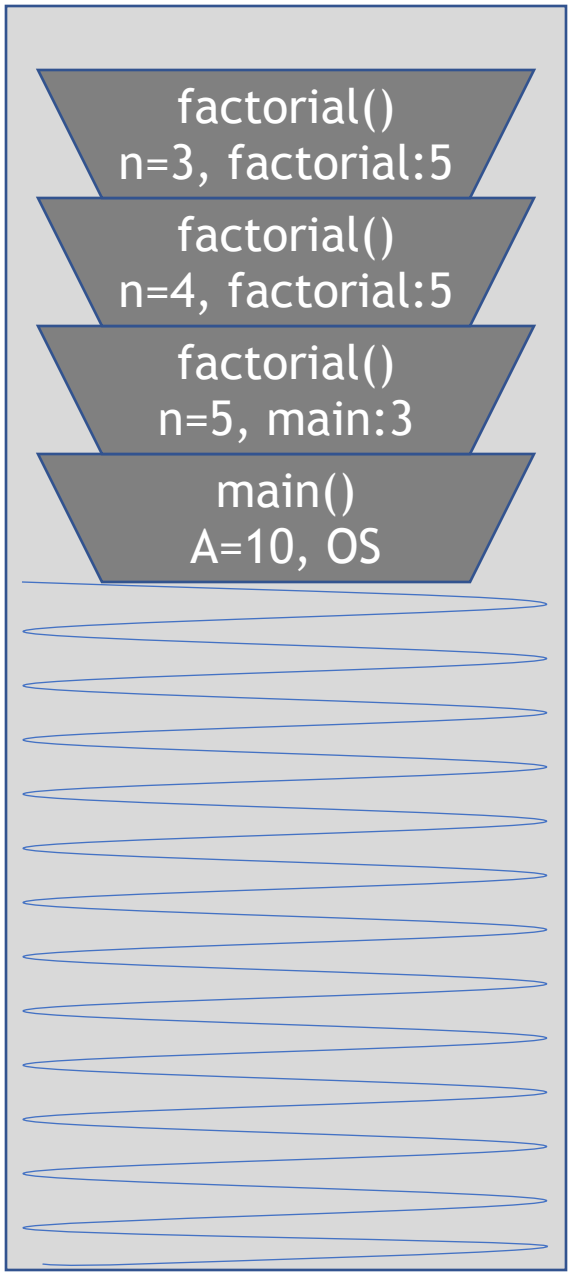
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=3) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 2;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

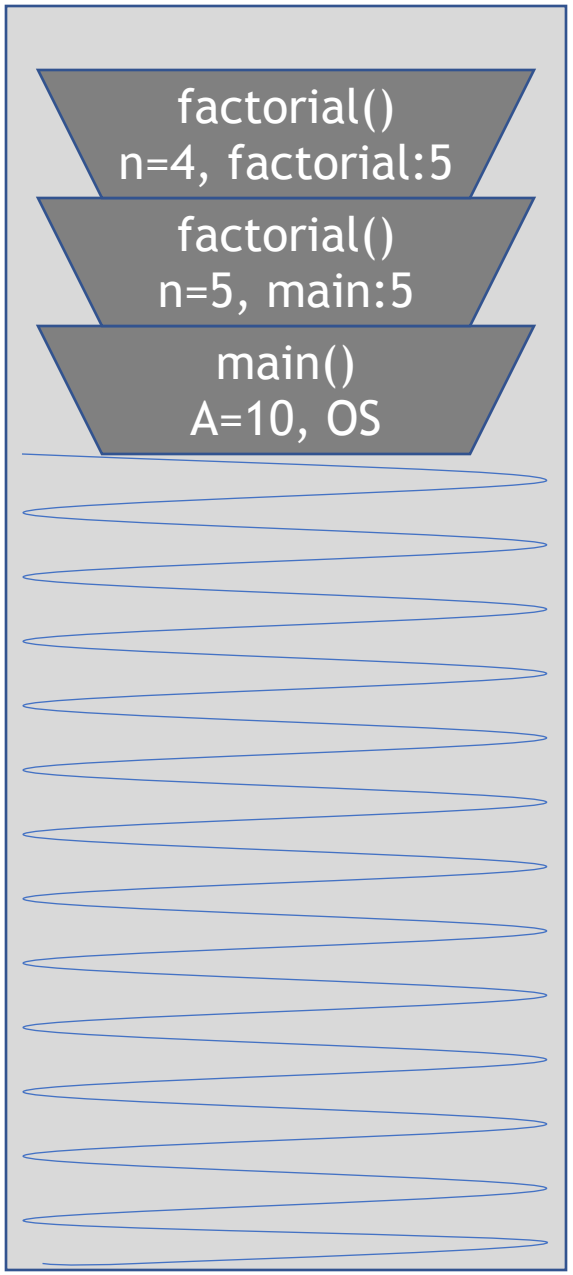
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=4) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 6;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

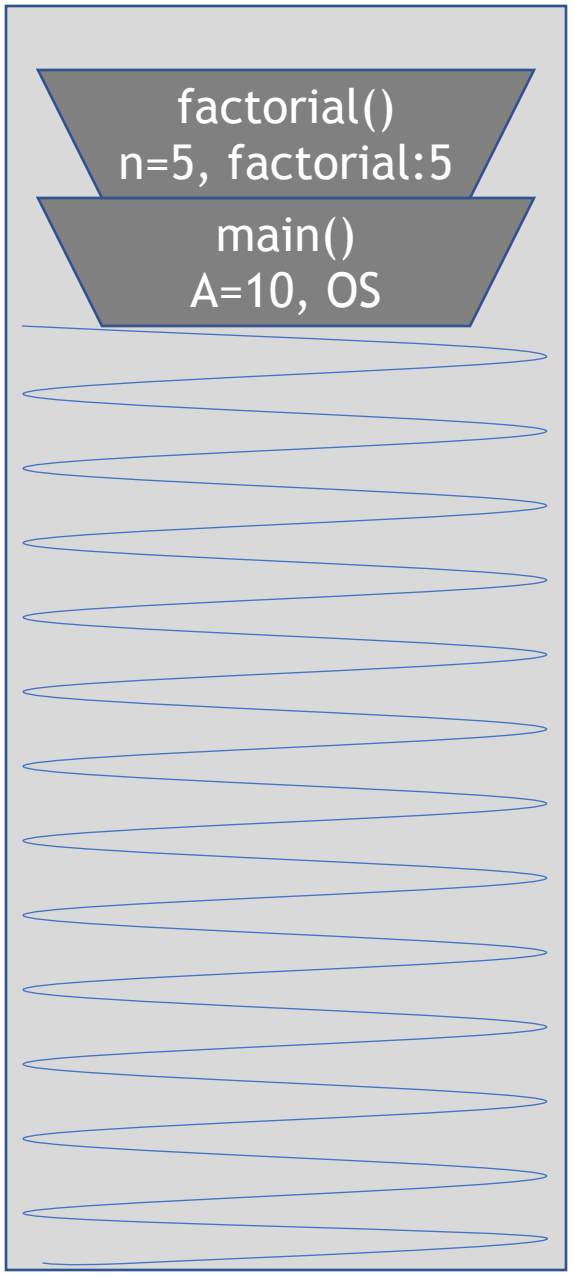
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  int factorial(int n=5) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n * 24;
6.      return F;
7.    }
8.  }
```

Call Stack



Compiled Code

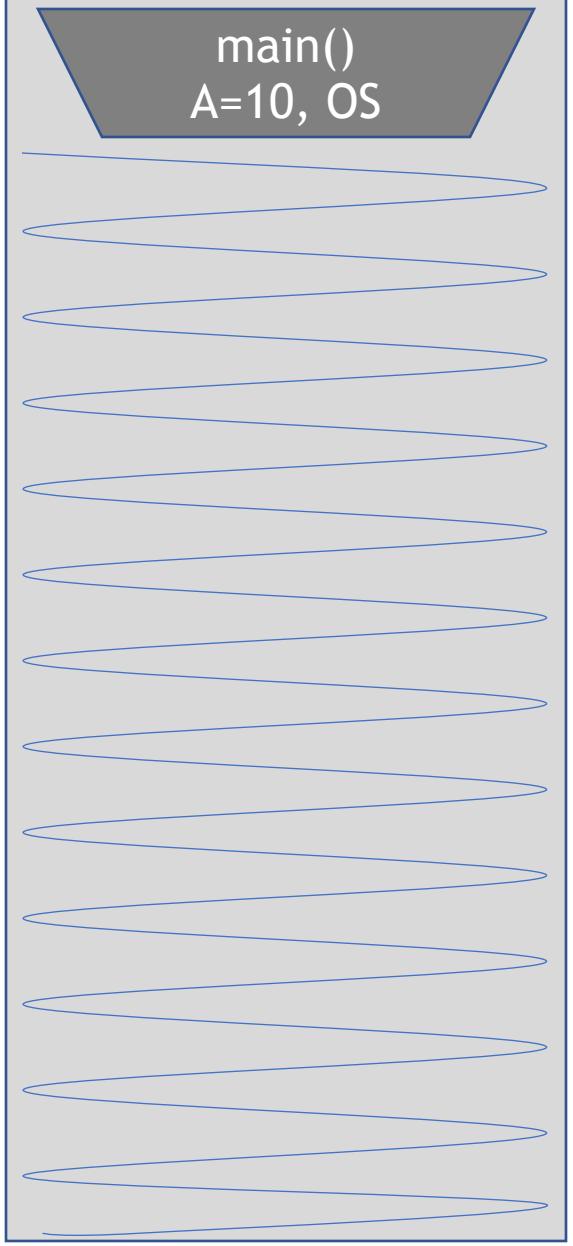
```
1.  void main() {
2.    int A = 10;
3.    int B = factorial(5);
4.    System.out.println(B);
5.  }
```

```
1.  int factorial(int n) {
2.    if (n == 1) {
3.      return 1;
4.    } else {
5.      int F = n *
    factorial(n-1);
6.      return F;
7.    }
8.  }
```

Executing Function

```
1.  void main() {
2.    int A = 10;
3.    int B = 120;
4.    System.out.println(B);
5.  }
```

Call Stack



Recursion — return values

```
public int rAdder(int num1, int num2) {
     if (num2<=0)
          return num1;
     return rAdder(num1+1, num2-1);
 }

 public int rAdderB(int num1, int num2) {
     if (num2<=0)
          return 0;
     return 1+rAdderB(num1, num2-1);
 }
```

Recursion — returning values & private recursive functions

```
public BigInteger fibonacci(int n) {
        if (n<=0) return BigInteger.valueOf(0);
         if (n<3) return BigInteger.valueOf(1);
         return iFibonacci(BigInteger.valueOf(1), BigInteger.valueOf(1)
 n-2);
     }
 private BigInteger iFibonacci(BigInteger fibNumA, BigInteger
 fibNumB, int counter)
     {
         if (counter==1)
             return fibNumA.add(fibNumB);
         return iFibonacci(fibNumB, fibNumA.add(fibNumB),
 counter-1);
     }
```

more returning values

```
public ArrayList<Integer> rAccmulate(int count)
{
    if (count <= 0)
        return new ArrayList<Integer>();
    ArrayList<Integer> alAcc = rAccmulate(count-1);
    alAcc.add(count);
    return alAcc;
}
```

recursion practice

```
/**
 * Compute the base 2 log of a number.
 * The integer part only.  So base2log(7)==2
*/
public int base2log(int num)
```

```
/**
 * Compute the sum of the components of the array
 */
public int addArray(int[] array);
```

```
/**
  * Count the number of times char occurs in str
  * hint: str.substring(1) curs the first letter off of a string
  **/
public int numOccur1(char ch, String str) {
```

Counter the number of occurrences of a letter in a string

```
public int numOccur1(char ch, String str) {
     if (str == null || str.equals("")) {
         return 0;
     }
     int count = 0;
     if (str.charAt(0) == ch) {
         count++;
     }
     numOccur1(ch, str.substring(1));
     return count;
 }
```

What does this return on “a” , “abc”
Why?

Occurrence count v2

```
int acount = 0;

 public int numOccur2(char ch, String str) {
     if (str == null || str.equals("")) {
         return 0;
     }
     if (str.charAt(0) == ch) {
         acount++;
     }
     numOccur2(ch, str.substring(1));
     return acount;
 }
```

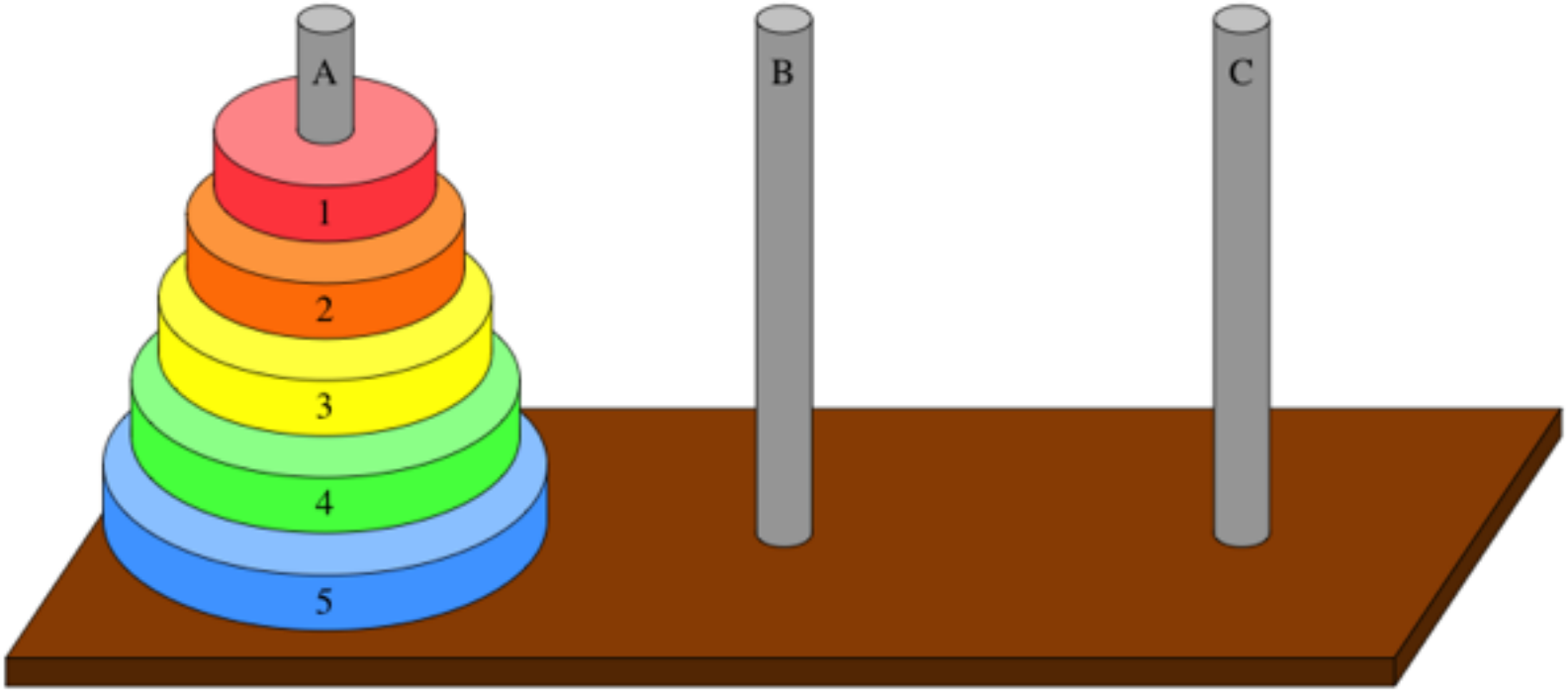
Correct, but a BAD solution

Occurrence count v3 and v4

```
public int numOccur3(char ch, String str) {
    if (str == null || str.equals("")) { return 0; }
    int count = 0;
    if (str.charAt(0) == ch) { count = 1; }
    return count + numOccur3(ch, str.substring(1));
}


public int numOccur4(char ch, String str) {
    return numOccur4Util(ch, str, 0);
}
private int numOccur4Util(char ch, String str, int count) {
    if (str == null || str.equals("")) { return count; }
    if (str.charAt(0) == ch) { count++; }
    return numOccur4Util(ch, str.substring(1), count);
}
```

Towers of Hanoi



Lab

Write a recursive function to compute the sum of the numbers 1 to n
For example: given 7 this should return 28

Here is the prototype for the function

```
int summation(int number);
```