# Lecture 3 - January 14

## Recursion: Part 1

***splitArray: Implementation and Tracing***

## Announcements/Reminders

- Assignment 1 release
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- Contact Information of TAs on common eClass site

# Problem on Recursion

https://codingbat.com/prob/p185204

Given an array of ints, is it possible to divide the ints into two groups, so that the sums of the two groups are the same. Every int must be in one group or the other. Write a recursive helper method that takes whatever arguments you like, and make the initial call to your recursive helper from splitArray(). (No loops needed.)

splitArray([2, 5, 3]) → t

splitArray([2, 2]) → true

splitArray([2, 3]) → false

splitArray([5, 2, 3]) → true

Insight: The recursive & base cases are defined s.t. the tree of recursive calls represents an exhaustive search of all possible input scenarios.

worst case

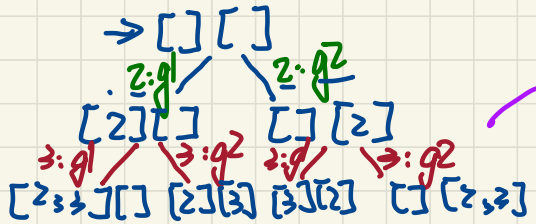
n elements

$m \times m \times \cdots \times m$

$m^n$

Intuition

How many ways to put n elements into m groups?




→ false.

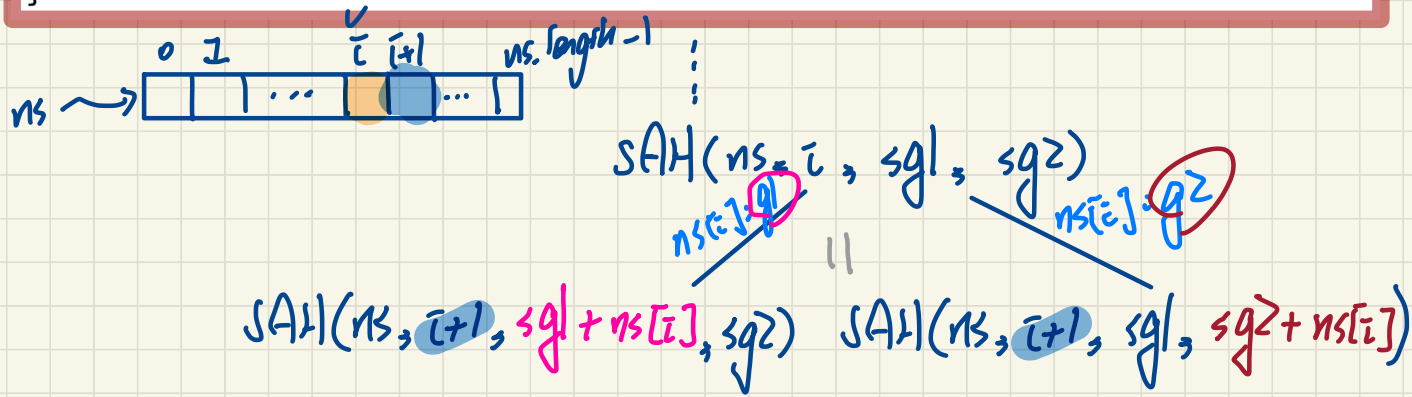
# splitArray: Java Implementation

```java
public boolean splitArray(int[] ns) {
    return splitArrayHelper(ns, 0, 0, 0);
}
```

empty groups

starting index to inspect : left to right

```java
private boolean splitArrayHelper(int[] ns, int i, int sumOfGroup1, int sumOfGroup2) {
    if(i == ns.length) {
        return sumOfGroup1 == sumOfGroup2;
    }
    else {
        return
            splitArrayHelper(ns, i + 1, sumOfGroup1 + ns[i], sumOfGroup2)
            ||
            splitArrayHelper(ns, i + 1, sumOfGroup1, sumOfGroup2 + ns[i]);
    }
}
```



## Math

Commutativity: $p \wedge q \equiv q \wedge p$

$p \vee q \equiv q \vee p$

(2.2) p eval. to F → also eval. q

p eval. to T → skip q → overall T

## Java

Evaluation order matters: (2.1) $p \;\&\&\; q \neq q \;\&\&\; p$

① Evaluation: left to right

(2.1) p eval. to T → eval q

p eval to F → skip q → overall F

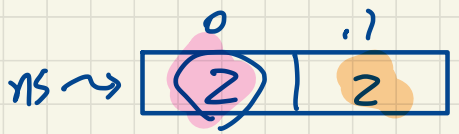
(2.2) ✓ $p \;||\; q \neq q \;||\; p$

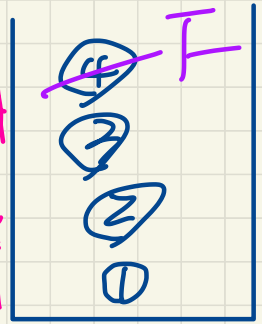
# splitArray: Tracing (1)

```
public boolean splitArray(int[] ns) {
    return splitArrayHelper(ns, 0, 0, 0);
}
```

```
@Test
public void testSplitArray_01() {
    RecursiveMethods rm = new RecursiveMethods();
    int[] input = {2, 2};
    assertEquals(true, rm.splitArray(input));
}
```

```
private boolean splitArrayHelper(int[] ns, int i, int sumOfGroup1, int sumOfGroup2) {
    if(i == ns.length) {
        return sumOfGroup1 == sumOfGroup2;
    }
    else {
        return
            splitArrayHelper(ns, i + 1, sumOfGroup1 + ns[i], sumOfGroup2)
            ||
            splitArrayHelper(ns, i + 1, sumOfGroup1, sumOfGroup2 + ns[i]);
    }
}
```



# splitArray: Tracing (2)

```java
public boolean splitArray(int[] ns) {
    return splitArrayHelper(ns, 0, 0, 0);
}
```

```java
private boolean splitArrayHelper(int[] ns, int i, int sumOfGroup1, int sumOfGroup2) {
    if(i == ns.length) {
        return sumOfGroup1 == sumOfGroup2;
    }
    else {
        return
            splitArrayHelper(ns, i + 1, sumOfGroup1 + ns[i], sumOfGroup2)
            ||
            splitArrayHelper(ns, i + 1, sumOfGroup1, sumOfGroup2 + ns[i]);
    }
}
```

```java
@Test
public void testSplitArray_04() {
    RecursiveMethods rm = new RecursiveMethods();
    int[] input = {5, 2, 2};
    assertEquals(false, rm.splitArray(input));
}
```

Exercises

(1) Trace on paper

(2) Trace on Eclipse.

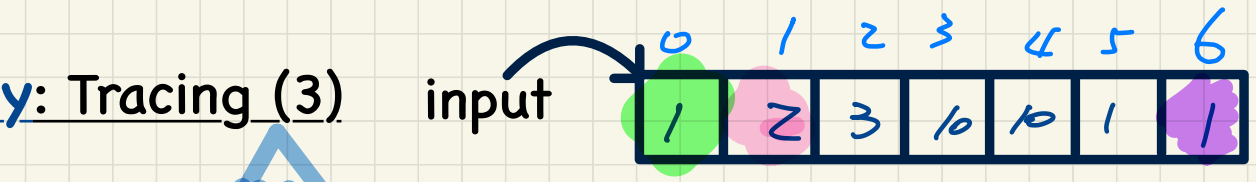
# splitArray: Tracing (3)

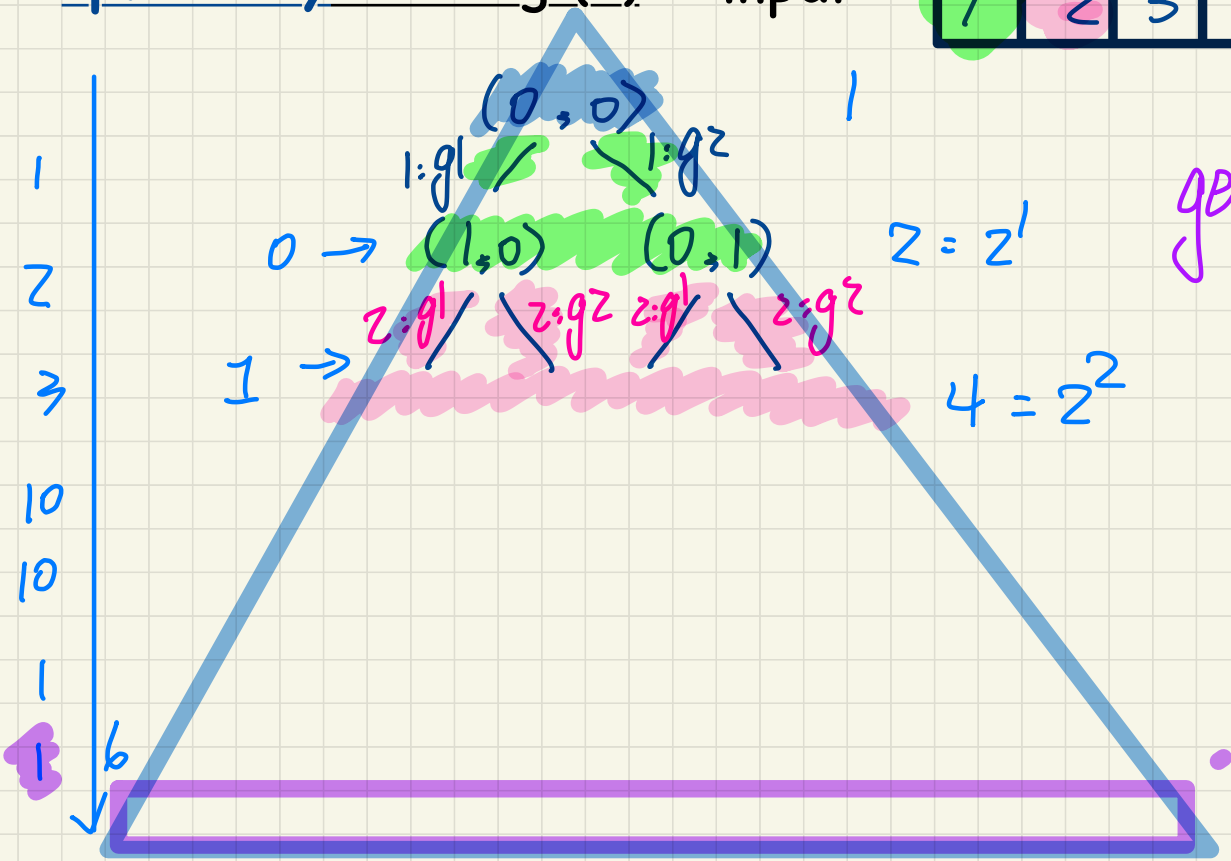
```
@Test
public void testSplitArray_13() {
    RecursiveMethods rm = new RecursiveMethods();
    int[] input = {1, 2, 3, 10, 10, 1, 1};
    assertEquals(true, rm.splitArray(input));
}
```

```
public boolean splitArray(int[] ns) {
    return splitArrayHelper(ns, 0, 0, 0);
}
```

```
private boolean splitArrayHelper(int[] ns, int i, int sumOfGroup1, int sumOfGroup2) {
    if(i == ns.length) {
        return sumOfGroup1 == sumOfGroup2;
    }
    else {
        boolean possibility1 = splitArrayHelper(ns, i + 1, sumOfGroup1 + ns[i], sumOfGroup2);
        boolean possibility2 = splitArrayHelper(ns, i + 1, sumOfGroup1, sumOfGroup2 + ns[i]);
        return possibility1 || possibility2;
    }
}
```

# splitArray: Tracing (3)




1

$2 = 2^1$

geometrical seq.

$4 = 2^2$

$2^7 = 128$