CS112

Recursion (Part 2)
Chapter 18
Lecture 14

الفصل الدراسي الثاني 1443- Spring 2022

College of Computer Science and Engineering

Think Recursively

- Many of the problems presented in the early chapters can be solved using recursion if you *think recursively*. For example, the palindrome problem can be solved recursively as follows:

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

Recursive Helper Methods

- The preceding recursive isPalindrome method is not efficient, because it creates a new string for every recursive call. To avoid creating new strings, use a helper method:

```
public static boolean isPalindrome(String s) {
  return isPalindrome(s, 0, s.length() - 1);
}
public static boolean isPalindrome(String s, int low, int high) {
  if (high <= low) // Base case
    return true;
  else if (s.charAt(low) != s.charAt(high)) // Base case
    return false;
  else
    return isPalindrome(s, low + 1, high - 1);
}
```

Case Study - Recursive Selection Sort

1. Find the smallest number in the list and swaps it with the first number.
2. Ignore the first number and sort the remaining smaller list recursively.

- See RecursiveSelectionSort.java

Recursion vs. Iteration

- Recursion is an alternative form of program control. It is essentially repetition without a loop.
- Recursion bears substantial overhead. Each time the program calls a method, the system must assign space for all of the method’s local variables and parameters. This can consume considerable memory and requires extra time to manage the additional space.
- Advantages of Using Recursion:
  - Recursion is good for solving the problems that are inherently recursive.

Case Study – Computing GCD (Greatest Common Devisor)

- The **gcd(m, n)** can also be defined recursively as follows:
  - If **m % n** is **0**, **gcd(m, n)** is **n**.
  - Otherwise, **gcd(m, n)** is **gcd(n, m % n)**.

gcd(2, 3) = 1

gcd(2, 10) = 2

gcd(25, 35) = 5

gcd(205, 301) = 5

gcd(m, n)

Approach 1: Brute-force, start from min(n, m) down to 1, to check if a number is common divisor for both m and n, if so, it is the greatest common divisor.

Approach 2: <u>Euclid's algorithm</u>

Approach 3: Recursive method

Approach 2: `Euclid’s` algorithm

```
// Get absolute value of m and n;
t1 = Math.abs(m); t2 = Math.abs(n);
// r is the remainder of t1 divided by t2;
r = t1 % t2;
while (r != 0) {
  t1 = t2;
  t2 = r;
  r = t1 % t2;
}

// When r is 0, t2 is the greatest common
// divisor between t1 and t2
return t2;
```

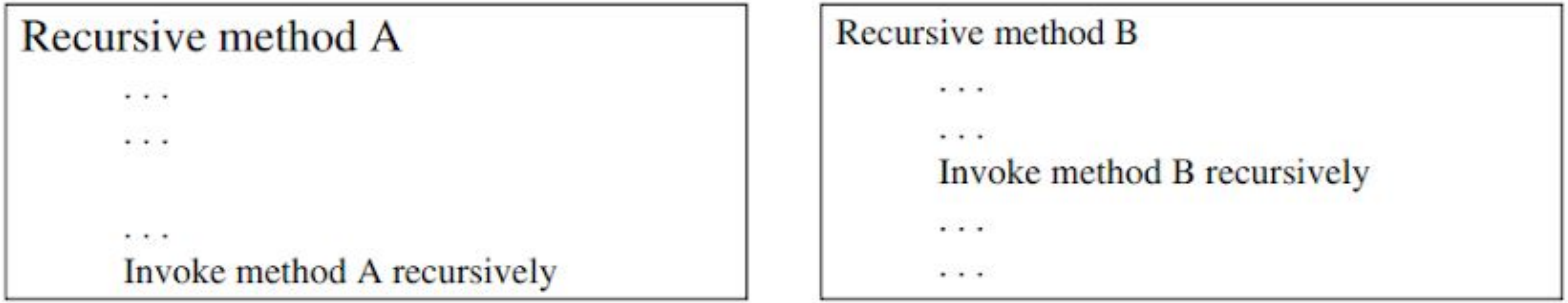
Approach 3: `Recursive Method`

gcd(m, n) = n if m % n = 0;
gcd(m, n) = gcd(n, m % n); otherwise;

- See GCD.java

Tail Recursion

- A recursive method is said to be *tail recursive* if there are no pending operations to be performed on return from a recursive call.
- Examples:
  - Non-tail recursive: ComputeFactorial.java
  - Tail Recursive: ComputeFactorialTailRecursion.java



(a) Tail recursion (b) Nontail recursion

Recommended Readings

- Recursive Binary Search: Page 716
- Finding Directory Size: Page 717
- Tower of Hanoi: Page719