ITI 1120
Lab #7 - Recursion

Contributors:
Diana Inkpen, Daniel Amyot, Romelia Plesa, Alan Williams

Lab 8 Agenda

- Recursion
  - 2 simple examples (to refresh your memory)
  - 3 algorithm and programming exercises

Recursion Example #1

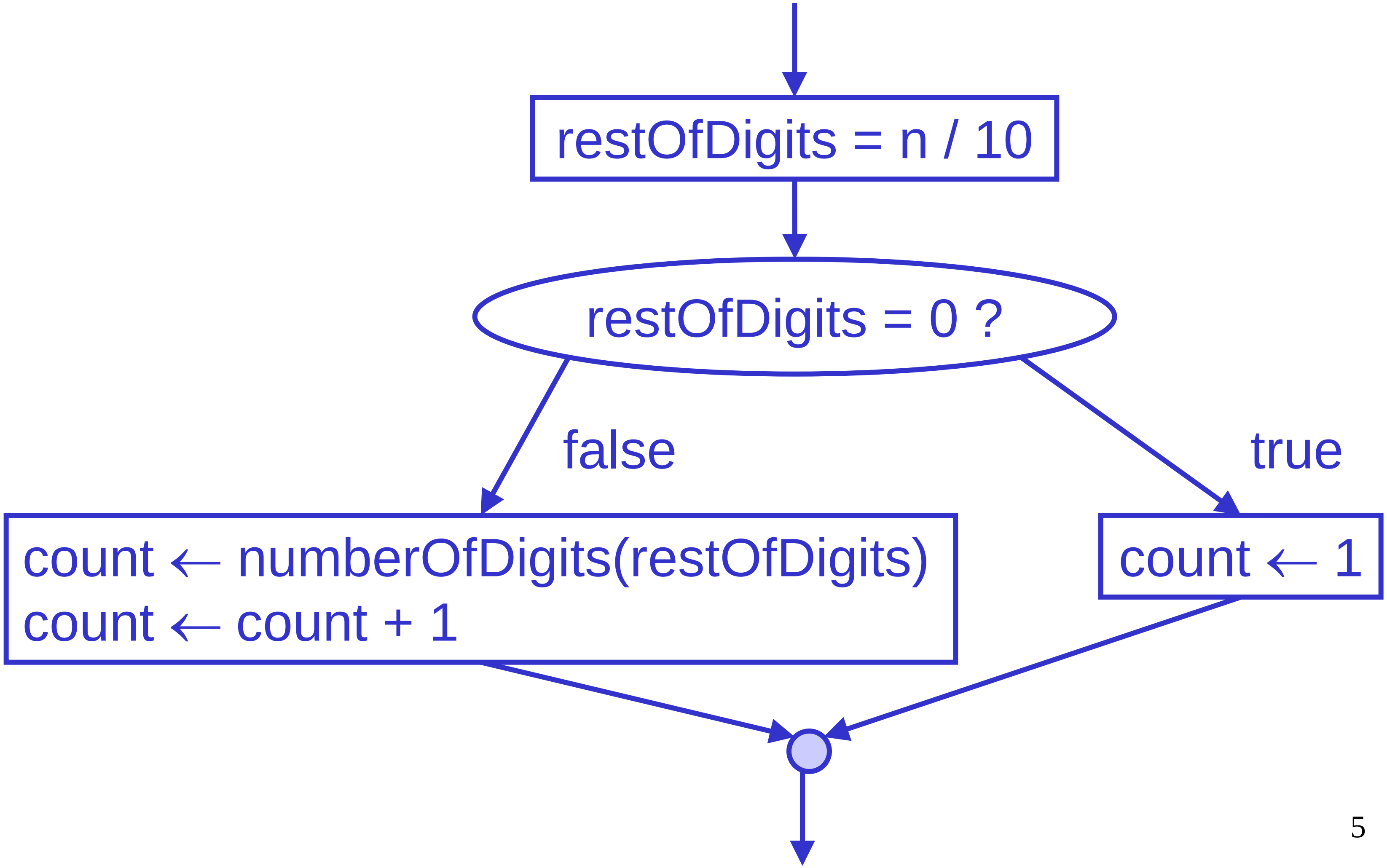
- Write a recursive algorithm for counting the number of digits in a non-negative integer, n
  - Example: if n = 34567, then the result is 5
  - If n = 1234567890, then the result is 10

Example #1 - solution

GIVENS:	n
INTERMEDIATE:	restOfDigits
RESULT:	count (the number of digits in n)
HEADER:	count $\leftarrow$ numberOfDigits(n)

Example #1 - solution - cont'd

BODY:



Trace for n = 254 (page1)

Line	n	restOfDigits	count
Initial values	254	?	?
(1) restOfDigits = n / 10		25	
(2) restOfDigits = 0 ? false			
(3) CALL count ← numberOfDigits(restOfDigits)			
(4) count ← count + 1			

Trace, page 2

count ← numberOfDigits(restOfDigits)

25

count ← numberOfDigits(n)

Line	n	restOfDigits	count
Initial values	25	?	?
(1) restOfDigits = n / 10		2	
(2) restOfDigits = 0 ? false			
(3) *CALL* count ← rumberOfDigits(restOfDigits)			
(4) count ← count + 1			

Trace , page 3

count ← numberOfDigits(restOfDigits)

1 2

count ← numberOfDigits(n)

Line	n	restOfDigits	count
Initial values	2	?	?
(1) restOfDigits = n / 10		0	
(2) restOfDigits = 0 ? true			
(5) count ← 1			1

Trace , page 2 (cont)

count ← numberOfDigits(restOfDigits)

2 25

count ← numberOfDigits(n)

Line	n	restOfDigits	count
Initial values	25	?	?
(1) restOfDigits = n / 10		2	
(2) restOfDigits = 0 ? false			
(3) CALL count ← numberOfDigits(restOfDigits)			
(4) count ← count + 1			2

Trace, page 1 (cont)

Line	n	restOfDigits	count
Initial values	254	?	?
(1) restOfDigits = n / 10		25	
(2) restOfDigits = 0 ? false			
(3) *CALL* count ← numberOfDigits(restOfDigits)			
(4) count ← count + 1			3

Let's see how the supplied code runs

- See NumberOfDigits.java
- The supplied recursive method contains `System.out.println()` statements at the following locations (print the actual value of `n`) to allow you to trace its execution.
  - Immediately after local variable declarations
  - Just before recursive method call:
  - Just after recursive method call:
  - Just before return statement
  - In the base case

Example #2

- Write a recursive algorithm to test if a given array is in sorted order.
  - Note: *sorted* is different than *strickly sorted* (where 2 elements cannot be equal)
- The size of a is greater or equal to 2.
- Examples:
  - a = {3, 6, 8, 5, 9}: False
  - a = {4, 5, 6, 6, 9, 14}: True

Example #2 - solution

GIVENS:

a	(an array of integers)
n	(the size of the array)

RESULT:

sorted	(Boolean: true if the array is sorted)

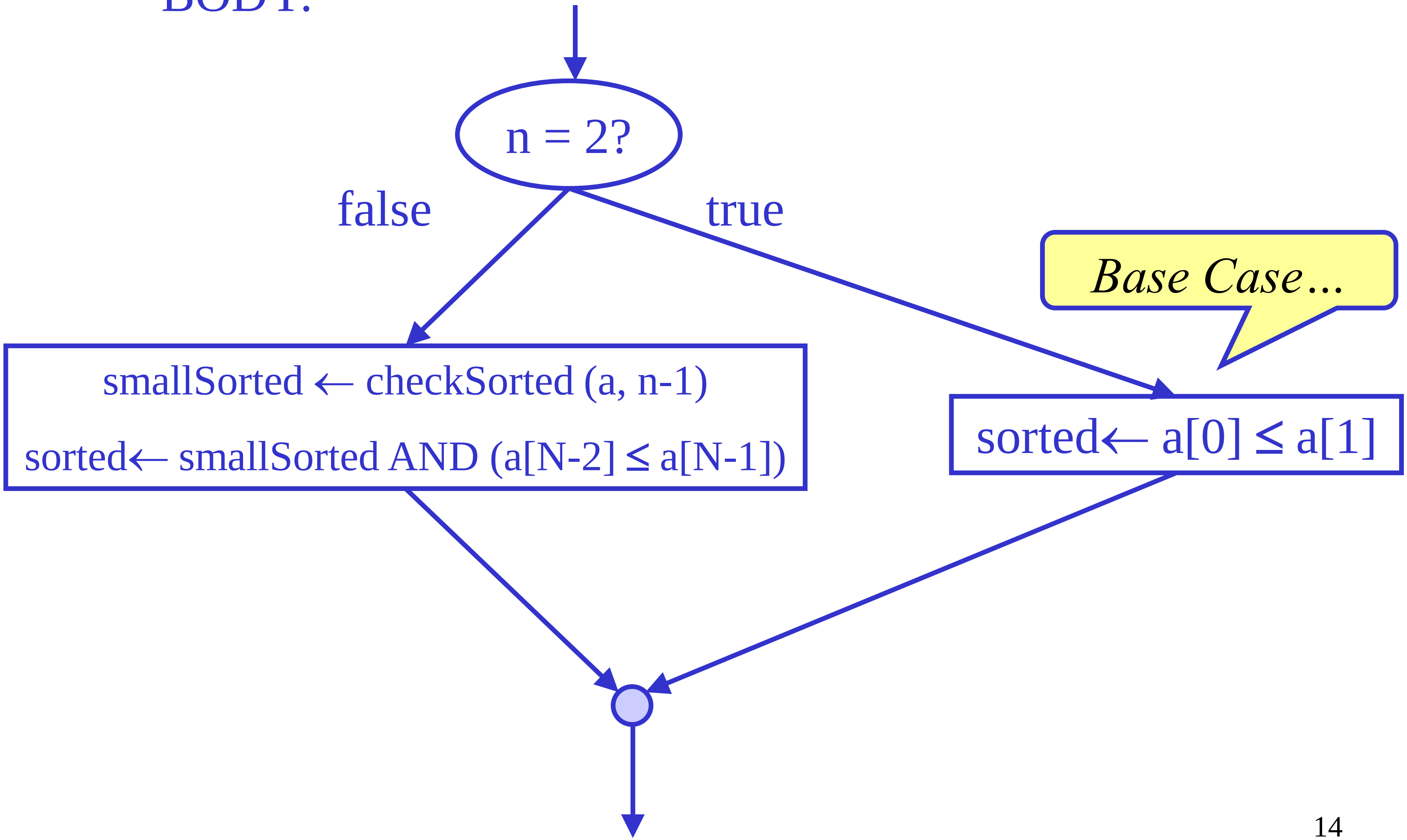
ASSUPTION: $n \geq 2$

HEADER:

sorted $\leftarrow$ checkSorted(a,n)

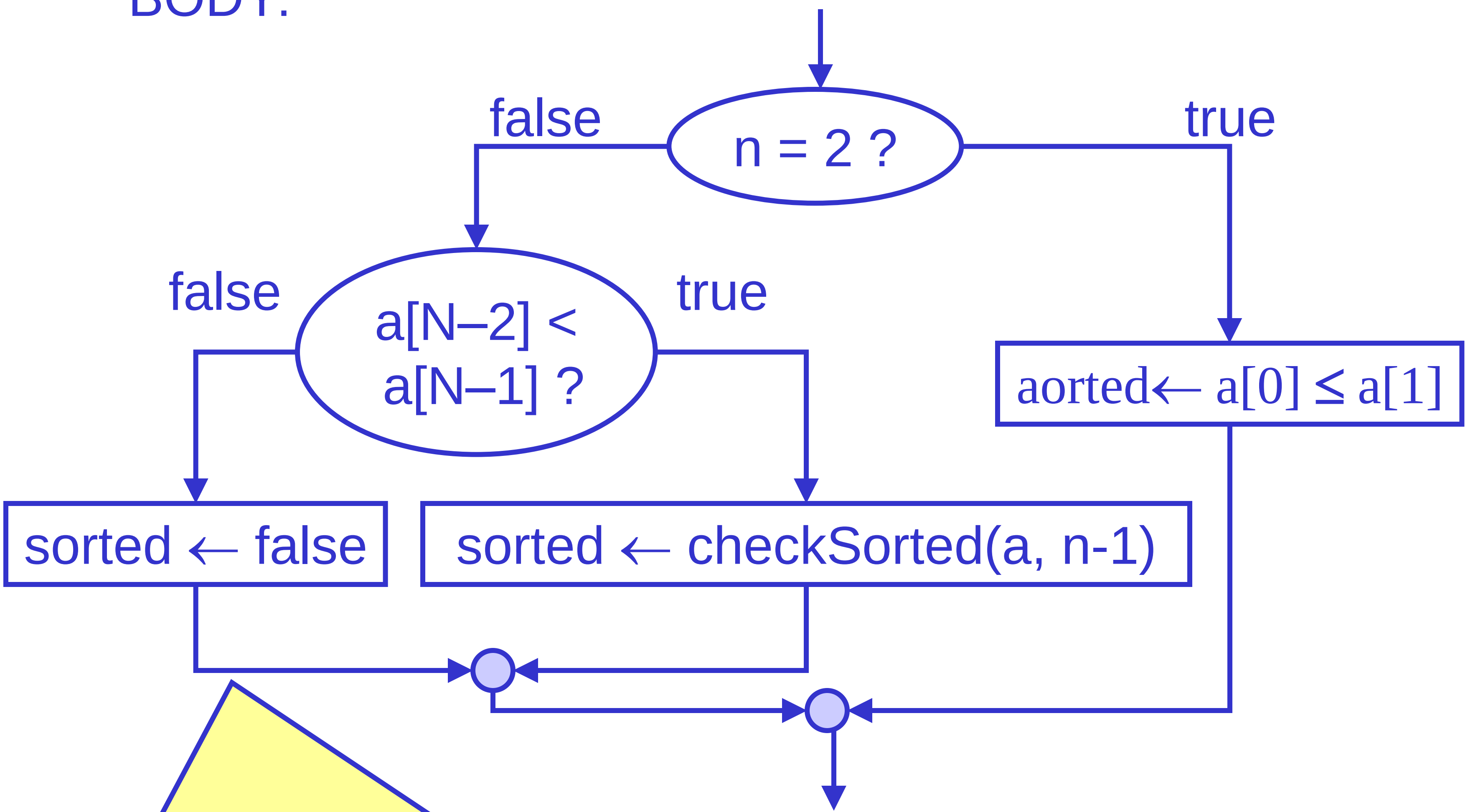
2e exemple - simple solution

BODY:



Example #2 - solution - efficient version

BODY:



Not necessary to go to base case if determine that array is not sorted. No need for recursive call.

Example 2 – Java Program

- See ArraySorted.java

- Examine the translation of the algorithm to Java, execute and test.
  - Insert System.out.println calls to trace the execution of the recursive method.

Exercise #1

- Write a recursive algorithm to test if all the characters in positions 0...n-1 of an array, a, of characters are digits.
- Start with the Word document Lab7Ex1.doc to develop your algorithm
  - Note that the main algorithm is provided.
- Translate the algorithm to Java
  - A partial program is provided – CheckDigits.java.

Exercise #2

- Write a recursive algorithm to create an array containing the values 1 to n (note array is indexed from 0 to n-1)
- Start with the Word document Lab7Ex2.doc to develop your algorithm
  - Note that the main algorithm is provided.
- Translate the algorithm to Java
  - A partial program is provided – CreateArray.java
- Hint:
  - Sometimes you need 2 algorithms:
    - The first is a “starter” algorithm that does some setup actions, and then starts off the recursion by calling the second algorithm
    - The second is a recursive algorithm that does most of the work.

Exercice #3: Euclid's algorithm

- The *greatest common divisor* (GCD) of two positive integers is the largest integer that divides both values with remainders of 0.
- Euclid's algorithm for finding the *greatest common divisor* is as follows:

  gcd(a, b) is …

  - b if $a \geq b$ and a mod b is 0
  - gcd(b, a) if $a < b$
  - gcd(b, a mod b) otherwise
- If we ensure that a ≥ b, the algorithm can be reduced to

  gcd(a, b) is …

  - b a mod b is 0
  - gcd(b, a mod b) otherwise :

Exercise #3 – Euclid's Algorithm

- Our recursive algorithm for our software can be:
  - m ← max(a, b) (need to develop the Max algorithm)
  - n ← a + b – m (i.e., the minimum of x and y)
- Base Case:
  - m MOD n = 0 → result is n
- Recursive Case:
  - Reduction: m ← m MOD n
  - Recursion: resPartiel ← gcd(n, m)
  - Result: resPartiel
- Question: will this algorithm always reach the base case?
  - Note that m MOD n is at most n – 1.

Exercise #3: Euclid's algorithm

- Develop recursive algorithm that takes two integers a and b and returns their greatest common divisor. You may assume that a and b are integers greater than or equal to 1.
  - Start with the Word file Lab7Ex3.doc.
  - Note that you must develop 2 algorithms, one for max and one for findGCD
  - You do NOT need to develop a main algorithm
- Translate the algorithms to Java
  - Create the Class Euclid and translate both algorithms to Java methods
  - Test the method using the Dr Java Interaction Pane to call the method findGCD method.

```
Euclid.findGCD(1234, 4321)
Euclid.findGCD(8192, 192)
```