Informatik II
Tutorial 2
Mihai Bâce
mihai.bace@inf.ethz.ch
08.03.2017
Overview

- Debriefing Exercise 1
- Briefing Exercise 2
a) Is it possible to prove the correctness by induction over $a$?
   - It is **NOT** possible
   - The induction step already fails for $b > 1$
   - The size of $a$ is growing
   - No conclusion is possible on already proven cases and no induction hypothesis is formulated

b) Does the algorithm terminate?
   - Yes, if we can make the value of $b$ reach 1
   - Does it always happen?
     - Yes, at each iteration, the value of $b$ is halved
     - After $\lfloor \log_2 b \rfloor$ steps, $b$ will always be 1
c) How do we prove the correctness of the algorithm when $b=0$?

$$f(a, b) = \begin{cases} 
0 & \text{falls } b=0 \\
 f(2a, b/2) & \text{falls } b \text{ gerade} \\
 a + f\left(2a, \frac{b-1}{2}\right) & \text{sonst}
\end{cases}$$

- The induction hypothesis becomes:
  $$\forall a \in \mathbb{N}, \forall b \in \{0, \ldots, n\} : f(a, b) = a \cdot b$$

- In 1b) we have shown that the case of $b=1$ is always reached. Since the integer division of 0 (or 1) by 2 gives 0, then the case of $b = 0$ is also always reached. No change in the proof of this step is required.
### U1.A2a

#### Gerade

```java
static boolean gerade(int x) {
    if (x == 0) return true;
    return !gerade(x-1);
}
```

#### Verdopple

```java
static int verdopple(int x) {
    if (x == 0) return 0;
    return 2 + verdopple(x-1);
}
```

#### Halbiere

```java
static int halbiere(int x) {
    if (x == 0) return 0;
    if (x == 1) return 0;
    return 1 + halbiere(x-2);
}
```

How many recursive calls?

- **Gerade**: \( X \) or \( X + 1 \)
- **Verdopple**: \( X \) or \( X + 1 \)
- **Halbiere**: \( \lfloor x/2 \rfloor \) or \( \lfloor x/2 \rfloor + 1 \)
U1.A2b

- The total number of method calls in terms of \(a\) and \(b\), for a single call to \(f\):

\[
\text{static int } f(\text{int } a, \text{int } b) \{ \\
    \text{if (} b == 0 \text{)} \text{ return 0;} \\
    \text{if (gerade(b)) return } f(\text{verdopple(a)}, \text{halbiere(b)}); \\
    \text{return } a + f(\text{verdopple(a)}, \text{halbiere(b)}); \\
\}
\]

- \(\text{gerade(b)} \rightarrow b+1\)
- \(\text{verdopple(a)} \rightarrow a+1\)
- \(\text{halbiere(b)} \rightarrow \lfloor b/2 \rfloor +1\)

- Total:
  - \(b+1 + a+1 + \lfloor b/2 \rfloor +1 \approx a + 3b/2 + 3\)
The total number of method calls:

- **K recursive steps**

\[ k \ast (a + 3 \frac{b}{2} + 3) \]

This is not correct!

\[
N(a, b) = \left( a + \frac{3b}{2} + 3 \right) + N\left(2a, \frac{b}{2}\right)
\]

\[
= \left( a + \frac{3b}{2} + 3 \right) + \left( 2a + \frac{3b}{4} + 3 \right) + N\left(4a, \frac{b}{4}\right)
\]

\[
= (a + 2^1a + 2^2a + \ldots) + \left( \frac{3b}{2^1} + \frac{3b}{2^2} + \frac{3b}{2^3} + \ldots \right) + (3 + 3 + \ldots)
\]

\[
= \sum_{i=0}^{k-1} 2^i a + \sum_{i=1}^{k} \frac{3b}{2^i} + k \cdot 3
\]

\[
\approx 2ab - a + 3b
\]

\[ k = \lfloor \log_2 b \rfloor + 1 \]
Validating inputs

Throw `IllegalArgumentException`

Javadoc

```java
/**
 * This function implements the ancient Egyptian multiplication.
 *
 * @param a must be a positive integer
 * @param b must be a positive integer
 * @return the product of a and b
 * @throws IllegalArgumentException if a or b is not positive
 */
public static int mult(int a, int b) throws IllegalArgumentException {
    if (a < 1) throw new IllegalArgumentException("Parameter a must be a positive integer but is "+ a);
    if (b < 1) throw new IllegalArgumentException("Parameter b must be a positive integer but is "+ b);
    return f(a, b);
}
```
Try/Catch

```java
public static int mult(int a, int b)
{
    try {
        if (a <= 0)
            throw new IllegalArgumentException("A negative!");
    }
    catch (IllegalArgumentException e) {
        . . .
    }
}
```

try - catch (- finally) must be outside the function that throws the exception
Overview

- Debriefing Exercise 1
- Briefing Exercise 2
Exercise 2

1. Rooted trees (theory)
   a) Given a tree, represent using:
      i. Brackets
      ii. Indented

   b) Given a bracket representation:
      i. Draw tree
      ii. Indented

   c) Can the tree in 1b) be clearly reconstructed? Why/why not?

   d) For the trees in 1a) and 1b) give:
      i. Height
      ii. Longest paths (trees are directed)
      iii. Set of leaves
Exercise 2

2. Recursive sorting
   a. Constructor: Create array of given size and fill with random numbers
   b. Build method toString
   c. Create recursiveSort(int until) to sort numbers in descending order

3. Binary trees
   a. Functions: leftChild, rightChild, father
   b. toString function that returns the indented tree
   c. Check if a given array is a valid representation of a tree
U2, A1 and A3: Overview of trees

General tree

Binary search tree

Binary tree
U2.A2

- Recursive Sorting

- How to generate random numbers?

```java
import java.util.Random;

Random randomGenerator = new Random(); // Constructor to create a new random number generator
int randVal = randomGenerator.nextInt(100); // How to use it, [0, 100); 0 inclusive, 100 exclusive
```

- Method toString()

```java
String s = "";
for (int i=0; i<array.length; i++) {
    // Code to create String s
}

return s;
```
Recursion = try to split the large problem into smaller problems that can be solved easier

recursiveSort(int until)
- until is an index from an array
- E.g. recursiveSort(4) will sort the elements from index 0 to 3

Given a list with N elements
- recursiveSort(i) sorts the elements from 0 to i-1
- In position i we need to add the maximum element remaining in the list (index i to N-1)
recursiveSort(4)

recursiveSort(3)

recursiveSort(2)

recursiveSort(1)

Ist sortiert!

2 <- findLargest(0,3)
swap(0,2)

recursiveSort(0)

3 <- findLargest(2,3)
swap(2,3)

Swap is not necessary anymore...

→ List sorted in descending order!
Binary trees can be represented as an array

- Root is always at index 0
- Node (i)
  - Left child: position \((2i + 1)\) in the array
  - Right child: position \((2i + 2)\) in the array

\[2^{height-1} \leq length < 2^{height}\]
Verify if a list is a valid representation of a binary tree

- checkTree()
- Root at index 0
- Direct successors for $i$ are at position $2i + 1$ and $2i + 2$
- What about array length?

Check if this applies for the passed array

- Test: Every element has a parent node
- "The root is its own father."
- What about the empty nodes?
No tutorial with ME next week (15.03)!

- You can attend the tutorial offered by Vincent

- Wednesday
  - Same time, 13:00 – 14:00
  - Location: HG G 3

- 22.03 back to our regular tutorial!
Extra

- Eclipse commands
  - Auto format code: Control + Shift + F
  - Code completion: Control + Space
Have Fun!