

Complexity Example *

Jens Lechtenbörger

Winter Term 2023/2024

Sample Algorithms

- Determine **Big O complexity** for following algorithms in Python!
- Background
 - This presentation embeds **klipse**, to enable live code execution.
 - * Thus, click into code on next slides, edit it, and have results immediately displayed.
 - If code does not execute, maybe reload without cache (Ctrl+F5 in Firefox)
 - Based on in-browser implementation of Python (**skulpt**), not complete.

Instructions

1. Figure out what the algorithms on the next slides do.
 - If you are not sure, maybe copy&paste into **Python Tutor**, which enables step-by-step execution with visualizations of values.
2. Determine the algorithms' complexities in terms of numbers of necessary plus operations.
 - If you are puzzled about the focus on plus operations, note that they occur at the inner-most level of nesting in **while** loops. For each iteration of a loop, a fixed number of other operations is executed, and those are covered by a constant factor in the definition of **Big O complexity** (M at Wikipedia.)

Subsequent quizzes lead to solutions. Please try yourself first.

*This PDF document is an inferior version of an **OER HTML page**; **free/libre Org** mode source repository.

Naive Multiplication

```
def naive_mult(op1, op2):
    if op2 == 0: return 0
    result = op1
    while op2 > 1:
        result += op1
    op2 -= 1
    return result

print(naive_mult(2, 3))
```

A solution

Naive Exponentiation

```
def naive_mult(op1, op2):
    if op2 == 0: return 0
    result = op1
    while op2 > 1:
        result += op1
    op2 -= 1
    return result

def naive_exp(op1, op2):
    if op2 == 0: return 1
    result = op1
    while op2 > 1:
        result = naive_mult(result, op1)
    op2 -= 1
    return result

print(naive_exp(2, 3))
```

- Some notes
 - Code on left is meant for non-negative integers
 - * Better code would test this
 - Python basics
 - * `def naive_mult(op1, op2)` declares function `naive_mult` with two operands
 - * `==` tests for equality, `=` is assignment to variable on left
 - * `result += op1` is short for `result = result + op1`
 - thus, `op1` is added to `result`
 - `-=` similarly
 - * `return` exits the function, delivers result

- Some notes
 - `naive_mult` is copied from [previous slide](#)
 - `naive_exp` shares same basic structure
 - * But with invocation of `naive_mult` instead of plus operation

A solution

A “Small” Change

- What happens if the order of arguments to `naive_mult` on the previous slide was reversed, i.e., if `naive_mult(op1, result)` instead of `naive_mult(result, op1)` was executed?
 - Clearly, as multiplication is commutative, the result does not change.
 - What about the resulting complexity?

A surprise?

License Information

Source files are available on [GitLab](#) (check out [embedded submodules](#)) under free licenses. Icons of custom controls are by [@fontawesome](#), released under [CC BY 4.0](#).

Except where otherwise noted, the work “Complexity Example”, © 2019-2022 [Jens Lechtenbörger](#), is published under the [Creative Commons license CC BY-SA 4.0](#).

No warranties are given. The license may not give you all of the permissions necessary for your intended use.

In particular, trademark rights are *not* licensed under this license. Thus, rights concerning third party logos (e.g., on the title slide) and other (trade-) marks (e.g., “Creative Commons” itself) remain with their respective holders.