Prof. Dr. Titus Dose
Fakultät für Informatik und Mathematik
OTH Regensburg

Theoretische Informatik
Wintersemester 2024/2025
30.10.2024

Übungsblatt 4

(Besprechung am 07.11.2024)

1. **O-Notation** 

   Order the following sets of functions with regard to inclusion $\subseteq$ and determine which of the sets are equal! Here log stands for the logarithm to the base 2 (rounded off) with $\log(0) =_{\text{def}} 0$. A reasoning of the inclusions and equalities is not required here.

   1. $O((\log(n^2)) + 17)$
   2. $O(2^n)$
   3. $O((\log n)^2)$
   4. $O((n+1)(n-1))$
   5. $O(2^{(2^n)})$
   6. $O((\log(n^2))^2 + 1)$
   7. $O(n \cdot 2^n)$
   8. $O(2n+1)$
   9. $O(3n^3+1)$
   10. $O(\log n)$
   11. $O(n^3)$
   12. $O(1)$
   13. $O(n \log n)$
   14. $O(n)$
   15. $O(2)$
   16. $O(n+1)$
   17. $O(n^2 \cdot 2^{\log n})$
   18. $O(n^2)$
   19. $O(n^n)$
   20. $O(2^{2n})$

2. **While program for prodZ with runtime $O(n^3)$** 

   We are looking for a commented while program (test with syntax checker!), that computes the function $\text{prodZ} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ with $\text{prodZ}(x, y) = x \cdot y$ in the time $O(n^3)$. Give a comprehensible estimate of the runtime of your program as a function of the input length $n = |x| + |y|$, where you can use the O notation.

Hints

Exercise 1:
For a total function $t\colon \mathbb{N} \to \mathbb{N}$, $O(t)$ is a set of functions (see page 186). For these sets, inclusions and equalities should be specified. For example, it holds $O(3) \subsetneq O(2n) = O(3n)$.

Exercise 2:

- Use the school method for multiplication (how does this method work for numbers represented in binary representation?).
- In the context of While programs for the modulo-2 function, we have discussed programs that are able to read an arbitrary bit of an input number in runtime $O(n^2)$ (and $O(n)$ respectivelyl).
- Check that the runtime of the resulting method is $O(n^3)$.
- Implement your method as a While program. Test your program for `prodZ(9**20,8**20)`, among other things and compare the result with the value computed by the Python interpreter.

Extra tasks

1. **While programs for multiplication and division in linear runtime** 
   We are looking for commented while programs (test with syntax checker!) with runtime $O(n)$ for the following functions.

   (a) $\text{prodZ}\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ mit $\text{prodZ}(x, y) = x \cdot y$

   (b) $\text{divZ}\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ with $\text{divZ}(x, y) = \begin{cases} \lfloor x/y \rfloor, & \text{if } y \neq 0 \\ \text{n.d.} & \text{otherwise} \end{cases}$

Solutions

Solution for exercise 1:

- $O(1) = O(2)$
- $O(\log n) = O((\log(n^2)) + 17)$
- $O((\log n)^2) = O((\log(n^2))^2 + 1)$
- $O(n) = O(n+1) = O(2n+1)$
- $O(n \log n)$
- $O(n^2) = O((n+1)(n-1))$
- $O(n^3) = O(3n^3 + 1) = O(n^2 \cdot 2^{\log n})$
- $O(2^n)$
- $O(n \cdot 2^n)$
- $O(2^{2n})$
- $O(n^n)$
- $O(2^{(2^n)})$

Hinweis: Da wir mit dem abgerundeten Logarithmus arbeiten, gelten die Logarithmengesetze nur eingeschränkt. Z.B. gilt $O(2^{n \log n}) \subsetneq O(n^n)$, obwohl man mit Blick auf die Logarithmengesetze eine Gleichheit erwarten würde. Das Runden beim Logarithmus kann jedoch den Exponenten um ca. n absenken, was beim Gesamtwert einer Division durch ca. 2^n entspricht.

Beweis für $2^{2^n} \notin O(n^n)$:
Wegen $n \leq 2^{\log n+1}$ gilt $n^n \leq (2^{\log n+1})^n = 2^{n \cdot (\log n+1)}$.
Wäre $2^{(2^n)} \in O(n^n)$, so gäbe es $n_0, c \in \mathbb{N}^+$, sodass für alle $n \geq n_0$ gälte: $2^{(2^n)} \leq c \cdot n^n \leq c \cdot 2^{n \cdot (\log n+1)}$. Dann wäre $2^n \leq c \cdot n \cdot (\log n + 1) \leq cn^2$ für alle $n \geq n_0$, insbesondere für $n \geq \max(c, n_0)$ also $2^n \leq n^3$, was jedoch für alle $n \geq 10$ falsch ist. Ein Widerspruch. Somit $2^{(2^n)} \notin O(n^n)$

Solution for exercise 2:

```
def prodFast(x,y):
  if (x < 0):               # transfer netative sign from x to y
    x = (0 - x)
    y = (0 - y)
  p = 1                     # p = 2**e
  e = 0
  while (p <= x):           # search smalles e with 2**e > x
    p = (p + p)             # i.e. |bin(x)|=e, if x!=0
    e = (e + 1)
  z = 0                     # future result
  for i in range(0,e):      # |bin(x)| iterations
    x = (x + x)             # left shift for x
    z = (z + z)             # left shift for z
    if (x >= p):            # if bit (e+1) in x from the left is one
      x = (x - p)           # remove bit
      z = (z + y)           # add y to the result
  return z
```

Regarding the runtime: Everything apart from the loops requires $O(1)$. Both loops are executed precisely the same number of times. So it sufficies to consider the first loop (there are only $O(1)$ computation steps in each of the loop iterations).

After iteration i of the first loop, $p = 2^i$. The loop is terminated once $p = 2^i \geq x$, i.e., the number of loop iterations is at most the smallest i greater than $\log x$. As $x \leq 2^{|x|+1}$, the overall runtime on input (x, y) is $\in O(\log(2^{|x|+1})) = O(|x|)$.

This shows that the runtime of the algorithm is in $O(n)$.