Junior-Prof. Dr. Robert Elsässer, Marco Muñiz, Phillip Heidegger WS 2009/2010

Algorithms Theory, Solution for Assignment 0

http://lak.informatik.uni-freiburg.de/lak_teaching/ws09_10/algo0910.php

Exercise 0.1 - Proof by induction

1. **base case** $n = 1$

$$\sum_{i=0}^{1} i^2 = 0^2 + 1^2 = 1 = \frac{6}{6} = \frac{1 \cdot (1+1) \cdot (2+1)}{6}$$

$n \to n+1$

$$\begin{aligned}\sum_{i=0}^{n+1} i^2 &= \sum_{i=0}^{n} i^2 + (n+1)^2 \\ &=^{\text{I.H.}} \frac{n \cdot (n+1) \cdot (2n+1)}{6} + \frac{6(n+1)^2}{6} \\ &= \frac{(n+1)(2n^2+7n+6)}{6} \\ &= \frac{(n+1)(n+2)(2n+3)}{6}\end{aligned}$$

2. The exercise states:

$$L(t) - I(t) = 1$$

base case $t = \square$. It holds:

$$L(t) - I(t) = 1 - 0 = 1$$

induction step $t = N(t_1, t_2)$. The hypothesis holds for t_1 and t_2. Then,

$$\begin{aligned}L(t) - I(t) &= L(t_1) + L(t_2) - (I(t_1) + I(t_2) + 1) \\ &= L(t_1) - I(t_1) + L(t_2) - I(t_2) - 1 \\ &=^{\text{I.H.}} 1 + 1 - 1 \\ &= 1\end{aligned}$$

Exercise 0.2 - Complexity

1. It holds $g(n) = \sqrt{n} = n^{\frac{1}{2}}$. Since $0.99998 > \frac{1}{2}$, $f(n)$ grows asymptotically faster than $g(n)$. Therefore, the following holds $f(n) = \Omega(g(n))$.

2. We have $f(n) = 2^{\log^2(n)} = (2^{\log(n)})^{\log(n)} = n^{\log(n)}$. In addition, it holds that $g(n) = \sum_{k=1}^{n^2} \frac{n}{2^k} = n \sum_{k=1}^{n^2} \frac{1}{2^k}$. Since this is a geometric progression with common ratio $\left|\frac{1}{2^k}\right| < 1$, then $\frac{1}{2} \leq \sum_{k=1}^{n^2} \frac{1}{2^k} - 1 \leq \sum_{k=0}^{\infty} \frac{1}{2^k} - 1 \leq 1$. Therefore, $f(n) = \Omega(g(n))$.

3. It holds $f(n) = n \log_2(n) = \Omega(n)$. Since $n = \Omega(n^{\frac{1}{3}})$ where $n^{\frac{1}{3}} = \sqrt[3]{n}$, it follows that $f(n) = \Omega(g(n))$.

4. We have $f(n) = \sqrt{n} = n^{\frac{1}{2}}$. Then $f(n)$ does not asymptotically grow faster than n and in particular not as fast as $1000n$. Thus, $f(n) = \mathcal{O}(g(n))$.

Exercise 0.3 - Complexity

We obtain the following table.

$n \leq \ldots$	A_1	A_2	A_3	A_4	A_5
max. 1 s.	10^6	118649	10^6	468	29
max. 1 h.	3.6×10^9	257685504	234892058	7113	41

Algorithm A_5 suffers from its high complexity and is useful for very small values of n. For $n = 1$, A_2 is the one with the lowest time complexity, $T_2 = 0$. A_2 is faster than A_1 for $\log_2(n) \leq 2$. For $n > 4$, A_1 will behave better than A_2. For $n < 1000000$, A_3 will have the best behavior. However, for a bigger $n > 1000000$, A_1 will have the best behavior, because then $\sqrt{n} > 1000$.

Exercise 0.4 - Hornor

A small reformulation make the decrement of i more explicit.

```
POLYNOMIAL(A,n,x) begin
  y := 0
  i := n + 1
  while (i >= 0) do begin
    i := i - 1
    y := y * x + A[i]
  end
  return y
end
```

The invariant of the for loop is:

$$y = \sum_{j=i}^{n} A[j] \cdot x^{j-i} \tag{1}$$

Before the loop is executed, it holds that:

$$y = 0 = \sum_{j=i}^{n} A[j] x^{j-i}$$

for $i = n + 1$.

At the beginning of the loop (1) holds. We have to show that (1) holds afterwards, too. Because in the loop i decreases by 1, we have to show, that

$$y_n = \sum_{j=i_n}^{n} A[j] \cdot x^{j-i_n} \tag{2}$$

while $y_n = y_o \cdot x + A[i_n]$, $i_n = i_o - 1$ and

$$y_o = \sum_{j=i_o}^{n} A[j] \cdot x^{j-i_o} \tag{3}$$

$$
\begin{aligned}
y_n &= y_o \cdot x + A[i_n] \\
&= {}^{(3)} \left(\sum_{j=i_o}^{n} A[j] \cdot x^{j-i_o} \right) \cdot x + A[i_n] \\
&= \sum_{j=i_o}^{n} A[j] \cdot x^{j-i_o+1} + A[i_n] \cdot x^0 \\
&= \sum_{j=i_n+1}^{n} A[j] \cdot x^{j-i_n} + A[i_n] \cdot x^{i_n-i_n} \\
&= \sum_{j=i_n}^{n} A[j] \cdot x^{j-i_n}
\end{aligned}
$$

After the loop, it holds that $i = 0$, and $y = \sum_{j=0}^{n} A[j] \cdot x^j$.