Übungsblatt 1

Maurice Donner Jan Hubrich Adrian Müller

May 14, 2022

Aufgabe 1

$$O(f(n)) = \{ g(n) : \exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : g(n) \leq c \cdot f(n) \}$$

a) 1 $\{n^3\} \subseteq O(n^3 - 2n^2)$

zz. $\exists c > 0, \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : n^3 \leq c(n^3 - 2n^2)$

Def 1: $n_0 = 3$, $c = 3$

zz $n^3 \leq 3(n^3 - 2n^2)$

$n \in \mathbb{N}^+$ $\Leftrightarrow$ $n \leq 3(n-2)$

$\Leftrightarrow$ $3 \leq n$

$\Rightarrow n^3 \subseteq O(n^3 - 2n^2)$

2 $\{n^{\frac{3}{2}}\} \subseteq O(n^2)$

zz: $\exists c > 0 ; \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : n^{\frac{3}{2}} \leq cn^2$

$\Leftrightarrow 1 \leq c\sqrt{n}$

Let $c = 1 \Rightarrow 1 \leq \sqrt{n}$ true for all $n \geq n_0 = 1$

since $f(x) = \sqrt{x}$ is strictly increasing

3 $\{n^3\} \not\in O(10^6 n^2)$

zz. $\forall c>0 : \exists n \in \mathbb{N}^+ : \; g(n) > c \cdot f(n)$

let $g(n) = n^3$ and $f(n) = 10^6 n^2$

zz $\exists n \in \mathbb{N}^+ : \; n^3 > c \cdot 10^6 n^2$

$n > c \cdot 10^6$ true because $n \in \mathbb{N}^+$

b) 1 $\{2^{n+1}\} \subseteq O(2^n)$

zz. $\exists c>0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : \; 2^{n+1} \leq c \cdot 2^n$

$\Leftrightarrow \; 2 \cdot 2^n \leq c \cdot 2^n$

$\Leftrightarrow \; 2 \leq c$ true for all $n_0 \in \mathbb{N}^+$ $c \geq 2$

We will imply this from now on...

2 $\{5\cos(n)+n\} \not\in O(1)$

zz: $\forall c>0 \;\; \exists n \in \mathbb{N}^+ : 5\cos(n)+n > c$

Since $\max(5\cos(n)) = 5$ -> Term will vanish for large n

=> zz: $n > c$, true because $n \in \mathbb{N}^+$

3 $\{(n+1)!\} \notin O(n!)$

zz: $\forall c > 0 : \exists n \in \mathbb{N}^+ : (n+1)! > c \cdot n!$

$\Leftrightarrow (n+1) \cdot n! > c \cdot n!$

$\Leftrightarrow n+1 > c$, true because $n \in \mathbb{N}^+$

c) 1 $\{\log_2 n\} \subseteq O(\log_{10} n) \wedge \{\log_2 n\} \subseteq \Omega(\log_{10} n)$

$\log_2(x) = \frac{\log(x)}{\log(2)}$

zz $\log_2(n) = c \cdot \log_{10}(n)$

$\Leftrightarrow \log_2(n) = c \cdot \frac{\log_2(n)}{\log_2(10)}$

Let $c = \log_2(10) \Rightarrow \log_2(n) = \log_2(n)$, true for all $n \in \mathbb{N}^+$

Since $g(n) \leq cf(n)$ and $g(n) \geq cf(n) \Rightarrow \{\log_2(n)\} \subseteq \Theta(\log_{10} n)$

2 -> Next Page

c) 2. $\{2^{\log_a(n)}\} \notin \Theta(2^{\log_b(n)})$ for $a \neq b$ (*)

~~case 1: a > b~~

if $g(n) \notin O(f(n)) \Rightarrow g(n) \notin \Theta(f(n))$ (*1)

if $g(n) \notin \Omega(f(n)) \Rightarrow g(n) \notin \Theta(f(n))$ (*2)

To show (*) it suffice to show either (*1) or (*2)

case 1: $a > b$

To show: $\{2^{\log_a(n)}\} \notin O(2^{\log_b(n)})$ (**)

$\log_2 2^{\log_a(n)} > c \cdot 2^{\log_b(n)}$

$\Leftrightarrow n^{\log_a(2)} > c \cdot n^{\log_b(n)} \quad | : n^{\log_b(n)}$

$\Leftrightarrow n^{\left(\frac{\log_a(2)}{\log_b(2)}\right)} > c$ (*3)

Because $\frac{\log_a(2)}{\log_b(2)} > 0$ for $a > b$ (prop. of log's) and n^d with $d > 0$ is strictly increasing, there is always an "n" for which (*3) for an arbitrary $c > 0$. Therefore (**) is true.

case 2: $b > a$

For symmetry reasons if $b > a$:

$\{2^{\log_a(n)}\} \notin \Omega(2^{\log_b(n)})$ (**2)

In either cases for ~~all~~ $a > b$ or $b > a$ either (**) and (**2) are true and therefore also (*).

d) 1 $\{f(n)+g(n)\} \notin O(\min(f(n), g(n)))$

$$\min(f(n),g(n)) = \begin{cases} f(n) : f(n) < g(n) \\ g(n) : f(n) \geq g(n) \end{cases}$$

$\exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : f(n)+g(n) \leq c \cdot \min(f(n), g(n))$

1st case: $f(n) < g(n)$

$\Rightarrow f(n)+g(n) \leq c \cdot f(n)$

$\Leftrightarrow g(n) \leq f(n)(c-1)$

Let $f(n) = n$, $g(n) = n^2$ $n \in \mathbb{N}^+$

$\Rightarrow n^2 \leq n \cdot (c-1)$

$\Leftrightarrow n \leq c-1$ false, because $n \in \mathbb{N}^+$

Proof through contradiction

2 $\{f(n)+g(n)\} \subseteq O(\max[f(n), g(n)])$

$$\max(f(n),g(n)) = \begin{cases} f(n) : f(n) > g(n) \\ g(n) : f(n) \leq g(n) \end{cases}$$

zz $\exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : f(n)+g(n) \leq c \cdot \max(f(n), g(n))$

1st case: Let $f(n) > g(n)$

$\Rightarrow f(n)+g(n) \leq c \cdot f(n)$

$\Leftrightarrow g(n) \leq f(n)(c-1)$, true for $c > 1$

2nd case: Let $f(n) \leq g(n)$

$\Rightarrow f(n)+g(n) \leq c \cdot g(n)$

$\Leftrightarrow f(n) \leq g(n)(c-1)$, true for $c \geq 1$

3 $\{f(n)g(n)\} \subseteq O(f(n)g(n))$

zz $\exists c > 0 : \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : f(n)g(n) \leq c \cdot f(n)g(n)$

Let $c = 1$: $\Rightarrow f(n)g(n) \leq f(n)g(n)$ true for all $n \in \mathbb{N}^+$

e) $\{f(n)\} \subseteq O(n) \not\Rightarrow \{2^{f(n)}\} \subseteq O(2^n)$

Let $f(n) = 10n$

$\{10n\} \subseteq O(n)$ ✓ but $\{2^{10n}\} \not\subseteq O(2^n)$, because

$2^{10n} = (2^n)^{10} \not\subseteq O(2^n)$

zz: $\forall c > 0 : \exists n \in \mathbb{N}^+ : 2^{10n} > c \cdot 2^n$

$\Leftrightarrow 10n > \log_2(c) + n$, true because $n \in \mathbb{N}^+$

2 $\{f(n)\} \subseteq O(n) \Rightarrow \{f(n)^2\} \subseteq O(n^2)$

$\exists c > 0; \exists n_0 \in \mathbb{N}^+ : \forall n \geq n_0 : f(n)^2 \leq c \cdot n^2$

$\Leftrightarrow f(n) \leq \sqrt{c}\, n$, true, because $\forall \sqrt{c} : \exists c' = \sqrt{c}$, and $f(n) \leq c' \cdot n$ (assumption)

f)

Z.z.: $\Theta(f) \overset{(*)}{=} \Theta(g) \Leftrightarrow f \in \Theta(g)$

suppose $h(n) \in \Theta(f(n)) \overset{(*)}{\Leftrightarrow} h(n) \in \Theta(g(n))$

$\Leftrightarrow \exists c_1, c_2, c_1', c_2' > 0 : \exists n_0 \in \mathbb{N}^+ : n_0 < n : h(n) \leq c_1 f(n)$

$\wedge\ h(n) \geq c_2 f(n) \wedge h(n) \leq c_1' g(n) \wedge h(n) \geq c_2' g(n)$

1. case

$\Leftrightarrow c_1 f(n) \geq h(n) \geq c_2' g(n)$

$\Leftrightarrow f(n) \geq \frac{c_2'}{c_1} g(n)$ let $\frac{c_2'}{c_1} = d_1 > 0$

$\Leftrightarrow f(n) \geq d_1 g(n)$

$\Leftrightarrow f(n) \in \Omega(g(n))$ $(*_1)$

2. case

$\Leftrightarrow c_2 f(n) \leq h(n) \leq c_1' g(n)$

$\Leftrightarrow f(n) \leq \frac{c_1'}{c_2} g(n)$ let $\frac{c_1'}{c_2} = d_2 > 0$

$\Leftrightarrow f(n) \leq d_2 g(n)$

$\Leftrightarrow f(n) \in O(g(n))$ $(*_2)$

with $(*_1 \wedge *_2) \Leftrightarrow f(n) \in \Theta(g)$ $=$

Aufgabe 2

Operationen: +, -, <, >, int x

1. Der Algorithmus muss ein exponentielles Wachstum aufweisen:

```
int main() {
    int n = 5;
    for (int i=0; i<=n; i++) {
        i+=i;
        for (int j=0; i<=n; i++) {
            j+=j;
        }
    }
}
```

Hier benötigt jeder for-loop allein

$3 \cdot \log n + 1$

$(i \leq n,\ i{+}{+},\ i{+}{=}i)$ $(\text{int } i{=}0)$

Operationen, und ist damit $O(\log n)$

Werden nun zwei dieser Loops verschachtelt, so erhalten wir

$$O(\log n) \cdot O(\log n) = O((\log n)^2)$$

2.

```
int main(){
    int n = 10;

    for (int i=0; i<n; i++){            // linear to n
        for (int j=0; j<n; j++){        // linear to n
            for (int k=0; k<n; k++){    // log(n)
                k+=k;
            }
        }
    }
    // in total n * n * log(n) = n^2*log(n)
}
```

3.

```
int substraction(int n) {
    if (n == 0) return 0;
    else for (n; n>=1; n--) substraction(n-1);
    return 0;
}
```

Ruft die Rekursion n mal auf.
Das führt zu $2^n - 1$ Rechenschritten

Aufgabe 3

The code can be found in the file `Aufgabe_3.cpp`. The code is commented, so every step can be followed. The program `Aufgabe_3` can be run and takes a string of zeros and ones as console input, followed by the variable bitlength `k`.
The algorithm is of the order $\mathcal{O}(n)$, because it's based on the recursive divide & conquer algorithm, where $d < b$.

Aufgabe 4

$C_1 = \overbrace{1231}^{a} \cdot \overbrace{1622}^{b}$ $(k=2)$

rechnet

$n = 4 = 2^2 = 2k$

$a = \overbrace{12}^{a_1} \cdot 100 + \overbrace{31}^{a_0}$

$b = \overbrace{16}^{b_1} \cdot 100 + \overbrace{22}^{b_0}$

$c_{11} = \overbrace{12}^{a_1} \cdot \overbrace{16}^{b_1}$ $(k=1)$

$a' = \overbrace{1}^{a_1'} \cdot 10 + \overbrace{2}^{a_0'}$

$b' = \overbrace{1}^{b_1'} \cdot 10 + \overbrace{6}^{b_0'}$

$c_{11}' = 1 \cdot 1 = 1 \quad (k=0)$

$c_{00}' = 2 \cdot 6 = 12 \quad (k=0)$

$= 1 \cdot 100 + [(1+2) \cdot (1+6) - 1 - 12] \cdot 10 + 12$

$3 \cdot 7 = 21 \quad k=0$

$= 1 \cdot 100 + (21 - 1 - 12) \cdot 10 + 12$

$= 100 + 80 + 12$

$= 192$

$c_{00} = \overbrace{31}^{a_0} \cdot \overbrace{22}^{b_0}$ $(k=1)$

$a' = 3 \cdot 10 + 1$

$b' = 2 \cdot 10 + 2$

$c_{11}' = 3 \cdot 2 = 6 \quad (k=0)$

$c_{00}' = 1 \cdot 2 = 2 \quad (k=0)$

$= 6 \cdot 100 + [(3+1) \cdot (2+2) - 6 - 2] \cdot 10 + 2$

$4 \cdot 4 = 16 \quad (k=0)$

$= 6 \cdot 100 + (16 - 6 - 2) \cdot 10 + 2$

$= 600 + 80 + 2$

$= 682$

$C = 192 \cdot 10000 + [(12+31) \cdot (16+22) - 192 - 682] \cdot 100 + 682$

$43 \cdot 38 \quad (k=1)$

$a = 4 \cdot 10 + 3$

$b = 3 \cdot 10 + 8$

$c_{11} = 4 \cdot 3 = 12 \quad (k=0)$

$c_{00} = 3 \cdot 8 = 24 \quad (k=0)$

$a \cdot b = 12 \cdot 100 + [(4+3) \cdot (3+8) - 12 - 24] \cdot 10 + 24$

$\downarrow$

$07 \cdot 11 \quad (k=1)$

$a' = 0 \cdot 10 + 7$

$b' = 1 \cdot 10 + 1$

$c_{11}' = 0 \cdot 1 = 0$

$c_{00}' = 7 \cdot 1 = 7$

$a' \cdot b' = 0 \cdot 100 + [(0+7) \cdot (1+1) - 0 - 7] \cdot 10 + 7$

$\downarrow$

$7 \cdot 2 = 14 \quad (k=0)$

$= 0 + 70 + 7$

$= 77$

$= 12 \cdot 100 + [77 - 12 - 24] \cdot 10 + 24$

$= 1200 + 410 + 24$

$= 1634$

$C = 192 \cdot 10000 + [1634 - 192 - 682] \cdot 100 + 682$

$= 1920000 + 76000 + 682$

$= 1996682$

Additional contents

The code for exercise 3 can be found below

```
#include <iostream>
#include <vector>
#include <string>
#include <bitset>
#include <cmath>

using namespace std;

// converts string of '0' and '1' into an integer
int strtoint(string s){
    int output = 0;

    int count = 0;
    for (int i=s.length()-1; i>=0; --i){
        if (s.at(i) == '1') output += pow(2, count);
        count++;
    }
    return output;
}


int search(string s, int k){// s is the input string containing only '0' and '1's
                            // k is the bit length of the numbers in s

    int numOfNums = s.length() / k; // calculates the amount of numbers in "s"
    if (numOfNums == 1){    // break condition if divide and conquer has divided and conquered
                            // only one number in bit form is left in string

        if (s.at(k-1) == '1'){ // looks a smallest bit. If it's 1 -> number is uneven, else number is even
            return strtoint(s) - 1; // -1, because missing number is one less compared to the remaining one
        }
        else{
            return strtoint(s) + 1; // +1, because missing number is one more compared to the remaining one
        }
    }

    int middle_bit = (numOfNums / 2 + 1) * k - 1;   // calculates the index of the smallest bit of the number
                                                    // in the middle of the array. There is always a "middle number"
                                                    // because the array has 2^k - 1 numbers in it

    if (s.at(middle_bit) == '1'){   // if the number in the middle is uneven
                                    // the missing number has to be to the right,
                                    // thus the search interval can be split in
                                    // half (divided). The first half of the
                                    // string can be discarded

        s.erase(0, (numOfNums/2+1)*k);              // divide step
        return search(s, k);                        // recursion step
    }
    if (s.at(middle_bit) == '0'){   // if the number in the middle is even, the
                                    // missing number has to be to the left.
                                    // Thus, the last half of the string can
                                    // be discarded.

        s.erase(middle_bit-k+1, (numOfNums/2+1)*k); // divide step
        return search(s, k);                        // recursion step
    }
    // after one iteration the string has 2^(k-1) numbers in it
    // after k-1 iterations, the array is conquered and the break condition
    // is true
}


int main(){
    string s;
    int k = 2;
    cout << "Please input the string" << endl;
    cin >> s;
    cout << "Please input the bitlength k" << endl;
    cin >> k;

    int lostNum = search(s, k);

    cout << "The lost Number is: "  << lostNum << endl;

    return 0;
}
```