Datenstrukturen und Algorithmen

Exercise 2

FS 2020

Program of today

1. Feedback of last exercise

2. Repetition theory
   - Induction
   - Analysis of programs
   - Solving Simple Recurrence Equations

Landau Notation

- Give a correct definition of the set $\Theta(f)$ as compact as possible analogously to the definitions for sets $\mathcal{O}(f)$ and $\Omega(f)$.

Landau Notation

- Give a correct definition of the set $\Theta(f)$ as compact as possible analogously to the definitions for sets $\mathcal{O}(f)$ and $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\}$

Landau Notation

- Give a correct definition of the set $\Theta(f)$ as compact as possible analogously to the definitions for sets $\mathcal{O}(f)$ and $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\}$
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0,\ n_0 \in \mathbb{N} : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\ \forall n \geq n_0\}$

Landau Notation

Prove or disprove the following statements, where $f, g : \mathbb{N} \to \mathbb{R}^+$.

(a) $f \in \mathcal{O}(g)$ if and only if $g \in \Omega(f)$.

(e) $\log_a(n) \in \Theta(\log_b(n))$ for all constants $a, b \in \mathbb{N} \setminus \{1\}$

(g) If $f_1, f_2 \in \mathcal{O}(g)$ and $f(n) := f_1(n) \cdot f_2(n)$, then $f \in \mathcal{O}(g)$.

Landau Notation

Sorting functions: if function f is left to function g, then $f \in \mathcal{O}(g)$.

$2^{16},\ \log(n^4),\ \log^8(n),\ \sqrt{n},\ n\log n,\ \binom{n}{3},\ n^5+n,\ \frac{2^n}{n^2},\ n!,\ n^n.$

Sum of elements in two-dimensional array

Problems / Questions?

2. Repetition theory

Induction: what is required?

- Prove statements, for example $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

Induction: what is required?

- Prove statements, for example $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Base clause:
  - The given (in)equality holds for one or more base cases.
  - e.g. $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.

Induction: what is required?

- Prove statements, for example $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Base clause:
  - The given (in)equality holds for one or more base cases.
  - e.g. $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Induction hypothesis: we assume that the statement holds for some n

Induction: what is required?

- Prove statements, for example $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- Base clause:
  - The given (in)equality holds for one or more base cases.
  - e.g. $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$.
- Induction hypothesis: we assume that the statement holds for some n
- Induction step $(n \to n+1)$:
  - From the validity of the statement for n (induction hypothesis) it follows the one for $n+1$.
  - e.g.: $\sum_{i=1}^{n+1} i = n+1+\sum_{i=1}^{n} = n+1+\frac{n(n+1)}{2} = \frac{(n+2)(n+1)}{2}$.

Induction: Example

- Show $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.

Induction: Example

- Show $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.
- Base clause:
  $n = 0$: $\sum_{i=0}^{0} r^i = 1 = \frac{1-r^1}{1-r}$.

Induction: Example

- Show $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$.
- Base clause:
  $n = 0$: $\sum_{i=0}^{0} r^i = 1 = \frac{1-r^1}{1-r}$.
- Induction step $(n \to n+1)$:

$$\sum_{i=0}^{n+1} r^i = r^{n+1} + \sum_{i=0}^{n} r^i$$

$$= r^{n+1} + \frac{1 - r^{n+1}}{1 - r} = \frac{r^{n+1} - r^{n+2} + 1 + r^{n+1}}{1 - r} = \frac{1 - r^{n+2}}{1 - r}.$$

[Besides..]

It can be shown easily in a direct manner

$$\frac{r^{n+1}-1}{r-1} \overset{!}{=} \sum_{i=0}^{n} r^i$$

$$\begin{aligned}
(r-1)\cdot\sum_{i=0}^{n} r^i &= \sum_{i=0}^{n} r^{i+1} - \sum_{i=0}^{n} r^i \\
&= \sum_{i=1}^{n+1} r^i - \sum_{i=0}^{n} r^i = \sum_{i=0}^{n+1} r^i - 1 - \sum_{i=0}^{n} r^i \\
&= r^{n+1} - 1
\end{aligned}$$

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

The code fragment implies $\Theta(n^2)$ calls to `f()`: the outer loop is executed $n/9$ times and the inner loop contains i calls to `f()`

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`
The second inner loop contains $\lfloor \log_2(n) \rfloor + 1$ calls to `f()`. Summing up yields $\Theta(n \log(n))$ calls.

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$$T(0) = 1$$

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$
$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$
$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

n	0	1	2	3	4
$T(n)$	1	2	4	8	16

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$
$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

n	0	1	2	3	4
$T(n)$	1	2	4	8	16

Hypothesis: $T(n) = 2^n$.

Induction

Hypothesis: $T(n) = 2^n$.
Induction step:

$$\begin{aligned} T(n) &= 1 + \sum_{i=0}^{n-1} 2^i \\ &= 1 + 2^n - 1 = 2^n \end{aligned}$$

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

You can also see it directly:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

$$\Rightarrow T(n-1) = 1 + \sum_{i=0}^{n-2} T(i)$$

$$\Rightarrow T(n) = T(n-1) + T(n-1) = 2T(n-1)$$

Recurrence Equation

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \frac{n}{2} + 1, & n > 1 \\ 3 & n = 1 \end{cases}$$

Specify a closed (non-recursive), simple formula for $T(n)$ and prove it using mathematical induction. Assume that n is a power of 2.

Recurrence Equation

$$
\begin{aligned}
T(2^k) &= 2T(2^{k-1}) + 2^k/2 + 1 \\
&= 2(2(T(2^{k-2}) + 2^{k-1}/2 + 1) + 2^k/2 + 1 = \ldots \\
&= 2^k T(2^{k-k}) + \underbrace{2^k/2 + \ldots + 2^k/2}_{k} + 1 + 2 + \ldots + 2^{k-1} \\
&= 3n + \frac{n}{2}\log_2 n + n - 1
\end{aligned}
$$

$\Rightarrow$ Assumption $T(n) = 4n + \frac{n}{2}\log_2 n - 1$

Induction

1. Hypothesis $T(n) = f(n) := 4n + \frac{n}{2}\log_2 n - 1$
2. Base Case $T(1) = 3 = f(1) = 4 - 1.$
3. Step $T(n) = f(n) \longrightarrow T(2 \cdot n) = f(2n)$ ($n = 2^k$ for some $k \in \mathbb{N}$):

$$\begin{aligned} T(2n) &= 2T(n) + n + 1 \\ &\overset{i.h.}{=} 2(4n + \frac{n}{2}\log_2 n - 1) + n + 1 \\ &= 8n + n\log_2 n - 2 + n + 1 \\ &= 8n + n\log_2 n + n\log_2 2 - 1 \\ &= 8n + n\log_2 2n - 1 \\ &= f(2n). \end{aligned}$$

Questions or Suggestions?