Datenstrukturen und Algorithmen

Exercise 2

FS 2019

Program of today

1. Feedback of last exercise
2. Repetition theory
   - Induction
   - Analysis of programs
3. Programming Task

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

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

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()`

Analysis

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();
}
```

Analysis

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()`

Analysis

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.

Analysis

How many calls to `f()`?

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

Analysis

How many calls to `f()`?

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

$T(0) = 1$

Analysis

How many calls to `f()`?

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

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

Analysis

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

Analysis

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

Analysis

How many calls to `f()`?

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

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

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

3. Programming Task

The Problem of Selection

Input

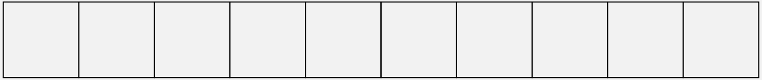
- unsorted array $A = (A_1, \dots, A_n)$ with pairwise different values
- Number $1 \leq k \leq n$.

Output $A[i]$ with $|\{j : A[j] < A[i]\}| = k - 1$

Special cases

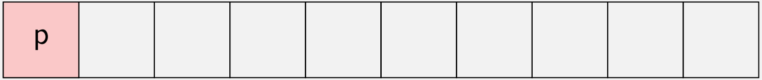
$k = 1$: Minimum: Algorithm with n comparison operations trivial.
$k = n$: Maximum: Algorithm with n comparison operations trivial.
$k = \lfloor n/2 \rfloor$: Median.

Use a pivot



Use a pivot

1. Choose a *pivot* p



Use a pivot

1. Choose a *pivot* p
2. Partition A in two parts, thereby determining the rank of p.

p	$>$	$\leq$	$\leq$	$>$	$>$	$\leq$	$\leq$	$>$	$\leq$

Use a pivot

1. Choose a *pivot* p
2. Partition A in two parts, thereby determining the rank of p.

p	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$>$	$>$	$>$	$>$

Use a pivot

1. Choose a *pivot* p
2. Partition A in two parts, thereby determining the rank of p.

$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	p	$>$	$>$	$>$	$>$
1					r				n

Use a pivot

1. Choose a *pivot* p
2. Partition A in two parts, thereby determining the rank of p.
3. Recursion on the relevant part. If $k = r$ then found.

$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	p	$>$	$>$	$>$	$>$
1					r				n

Algorithmus Partition($A[l..r], p$)

Input: Array A, that contains the sentinel p in the interval $[l, r]$ at least once.
Output: Array A partitioned around p. Returns position of p.

while $l \leq r$ **do**
    while $A[l] < p$ **do**
        $l \leftarrow l + 1$
    while $A[r] > p$ **do**
        $r \leftarrow r - 1$
    swap($A[l]$, $A[r]$)
    if $A[l] = A[r]$ **then**
        $l \leftarrow l + 1$
return $l - 1$

Algorithm Quickselect ($A[l..r], k$)

Input: Array A with length n. Indices $1 \leq l \leq k \leq r \leq n$, such that for all $x \in A[l..r] : |\{j|A[j] \leq x\}| \geq l$ and $|\{j|A[j] \leq x\}| \leq r$.
Output: Value $x \in A[l..r]$ with $|\{j|A[j] \leq x\}| \geq k$ and $|\{j|x \leq A[j]\}| \geq n - k + 1$

if l=r **then**
    return $A[l]$;
$x \leftarrow$ RandomPivot$(A[l..r])$
$m \leftarrow$ Partition$(A[l..r], x)$
if $i < m$ **then**
    return QuickSelect$(A[l..m-1], k)$
else if $i > m$ **then**
    return QuickSelect$(A[m+1..r], k)$
else
    return $A[l]$

Algorithm RandomPivot ($A[l..r]$)

Input: Array A with length n. Indices $1 \leq l \leq i \leq r \leq n$
Output: Random "good" pivot $x \in A[l..r]$
repeat
  choose a random pivot $x \in A[l..r]$
  $p \leftarrow l$
  for $j = l$ **to** r **do**
    if $A[j] \leq x$ **then** $p \leftarrow p + 1$
until $\left\lfloor \frac{3l+r}{4} \right\rfloor \leq p \leq \left\lceil \frac{l+3r}{4} \right\rceil$
return x

This algorithm is only of theoretical interest and delivers a good pivot in 2 expected iterations. Practically, in algorithm QuickSelect a uniformly chosen random pivot can be chosen.

Questions or Suggestions?