Informatik II

Übung 2

Andreas Bärtschi, Andreea Ciuprina, Felix Friedrich, Patrick Gruntz,
Hermann Lehner, Max Rossmannek, Chris Wendler

FS 2018

Program Today

1. Repetition Theory
   - Induction
   - Analysis of programs
   - Divide & Conquer
2. Programming Task
   - Quick Select
   - Collections in Java

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 ($n \rightarrow n+1$):
  - From the validity of the statement for n (induction hypothesis) it follows the one for $n+1$.
  - e.g.: $\sum_{i=1}^{n+1} i = n+1+\sum_{i=1}^{n} i = n+1+\frac{n(n+1)}{2} = \frac{(n+2)(n+1)}{2}$.

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()`

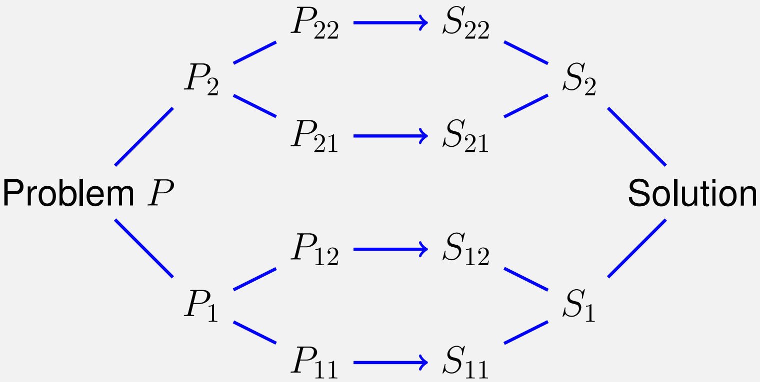
The second inner loop contains $\lfloor \log_2(n) \rfloor + 1$ calls to `f()`. Summing up yields $\Theta(n \log(n))$ calls.

divide et impera

Divide and Conquer

Divide the problem into subproblems that contribute to the simplified computation of the overal problem.

divide et impera



Binary Search Algorithm BSearch$(A[l..r], b)$

Input : Sorted array A of n keys. Key b. Bounds $1 \leq l \leq r \leq n$ or $l > r$ arbitrarily.
Output : Index of the found element. 0, if not found.

$m \leftarrow \lfloor (l+r)/2 \rfloor$
if $l > r$ **then** // Unsuccessful search
  return *NotFound*
else if $b = A[m]$ **then**// found
  return m
else if $b < A[m]$ **then**// element to the left
  return BSearch$(A[l..m-1], b)$
else // $b > A[m]$: element to the right
  return BSearch$(A[m+1..r], b)$

2. Programming Task

The Problem of Selection

Input

- unsorted array $A = (A_1, \ldots, A_n)$ 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

p									

Use a pivot

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

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

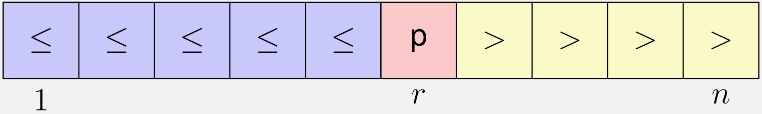
Use a pivot

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

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

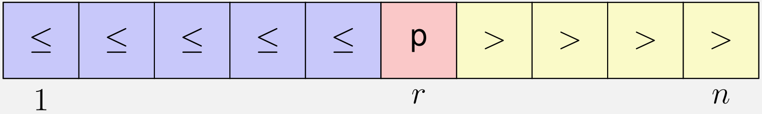
Use a pivot

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



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.



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

Input : Array A, that contains the pivot 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 ← RandomPivot(A[l..r])
m ← Partition(A[l..r], x)
if k < m then
    return QuickSelect(A[l..m − 1], k)
else if k > m then
    return QuickSelect(A[m + 1..r], k)
else
    return A[k]
```

Organizing Data

- Data Structures that we know
  - Arrays – Fixed-size sequences
  - Strings – Sequences of characters
  - Linked Lists (up to now: self-made for a fixed element type)
- General Collection Concept in Java
  - ArrayList on generic element types
  - LinkedList, HashMaps, Sets, Maps, ...

Generic List in Java: `java.util.List`

```
import java.util.ArrayList;
import java.util.List;
...

// Liste von Strings
List<String> list = new ArrayList<String>();

list.add("abc");
list.add("xyz");
list.add(1,"123"); // Fuege 123 an Position 1 ein
System.out.println(list.get(0)); // abc

for (String s: list) // For auf Iterator der Liste
       System.out.println(s); // abc 123 xyz
```

Lists of Integers

- Java generics (e.g. collections) can only operate on objects
- Fundamental types `int`, `float` (etc.) are no objects
- java offers wrapper classes for fundemental types, e.g. type `Integer`
- java provides *autoboxing* and automatically wraps a fundamental type into a wrapper class, where necessary.

Lists of Integers

```
import java.util.ArrayList;
import java.util.List;
...

// Lists of integers
List<Integer> list = new ArrayList<Integer>();

list.add(3);
list.add(4);
list.add(1,5); // Fuege 5 an Position 1 ein
System.out.println(list.get(0)); // 3

for (int i: list){ // For auf Iterator der Liste
      System.out.println(i); // 3 5 4
}
```

Lists of Integers

```
import java.util.ArrayList;
import java.util.List;
...

// Lists of integers
List<Integer> list = new ArrayList<Integer>();

list.add(3);
list.add(4);
list.add(1,5); // Fuege 5 an Position 1 ein
System.out.println(list.get(0)); // 3

for (int i: list){ // For auf Iterator der Liste
        System.out.println(i); // 3 5 4
}
```

Questions / Suggestions?