Sortieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

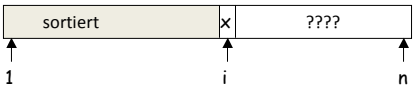
Eingabe: Feld A[1..n] von "Schlüsseln"

Ausgabe: A[1..n] so umgestellt, dass $A[1] \leq A[2] \leq \cdots \leq A[n]$

Insertionsort:

```
IS( A[] )
    n = length( A )
    for i = 2 to n do
        x = A[i]
        j = i
        while j>1 and A[j-1] > x do A[j] = A[j-1]
                                    j = j-1
        A[j] = x
```

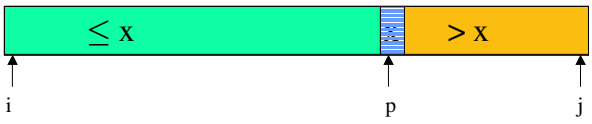
Invariante für for-Loop



Quicksort

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Idee: wähle „Pivotelement" x in Feld und stelle Feld so um:



sortiere Teilfeld der „kleinen" Elemente ($\leq$ x) rekursiv
sortiere Teilfeld der „großen" Elemente (> x) rekursiv

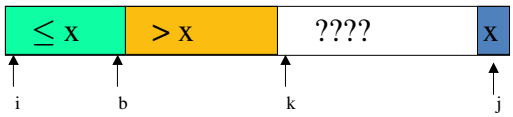
```
QS( A[] )
        n = length( A )
        Quicksort( A , 1 , n )

Quicksort( A , i , j )
        if  i < j then      p = Partition( A , i , j , A[j] )
                            Quicksort( A , i , p-1 )
                            Quicksort( A , p+1 , j )
```

Partitionieren

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Invariante:




```
Partition( A , i , j , x )
        b = i-1
        for k = i to j do  swap( A[k] , A[b+1] )
                           if  A[b+1] ≤ x then b++
        return b
```

Überlege Korrektheit des Rückgabewertes im letzten Schritt !

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Ziel: Algorithmen vergleichen, um vorherzusagen, welcher besser, schneller ist.

Möglichkeit 1, Empirisch:
Algorithmen implementieren, auf einem oder mehreren Rechnern auf diversen Eingaben laufen lassen und Laufzeiten vergleichen.

Problem: wenig Vorhersagekraft für andere Eingaben und andere Rechner.

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Ziel: Algorithmen vergleichen, um vorherzusagen, welcher besser, schneller ist.

Möglichkeit 2, Analytisch:
Algorithmen bzgl. eines ***Laufzeitmodells*** analysieren und die ***Laufzeitfunktionen*** vergleichen

Probleme: Welches ***Laufzeitmodell*** ?
Wie vergleicht man ***Laufzeitfunktionen*** ?

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Ziel: Algorithmen vergleichen, um vorherzusagen, welcher besser, schneller ist.

Möglichkeit 2, Analytisch:
Algorithmen bzgl. eines ***Laufzeitmodells*** analysieren und die ***Laufzeitfunktionen*** vergleichen

Probleme: Welches ***Laufzeitmodell*** ?
Wie vergleicht man ***Laufzeitfunktionen*** ?

Einfaches, abstraktes *Laufzeitmodell*:

Jede „primitive“ Rechenoperation (arithmetische Operation wie +,*,-,/,mod,*etc.*
logische Opertaion wie **and**, **or**, *etc.*,
Speicherzugriff,
(bedingter) Sprungbefehl,
etc.)
braucht eine Zeiteinheit.

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Ziel: Algorithmen vergleichen, um vorherzusagen, welcher besser, schneller ist.

Möglichkeit 2, Analytisch:
Algorithmen bzgl. eines ***Laufzeitmodells*** analysieren und die ***Laufzeitfunktionen*** vergleichen

Probleme: Welches ***Laufzeitmodell*** ?
Wie vergleicht man ***Laufzeitfunktionen*** ?

Laufzeitfunktion für primitives Laufzeitmodell

A Algorithmus E Eingabe

$T_A(E)$ = Anzahl der primitiven Operationen,
die Algorithmus A braucht bei Abarbeitung von Eingabe E

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Ziel: Algorithmen vergleichen, um vorherzusagen, welcher besser, schneller ist.

Möglichkeit 2, Analytisch:
Algorithmen bzgl. eines ***Laufzeitmodells*** analysieren und die ***Laufzeitfunktionen*** vergleichen

Probleme: Welches ***Laufzeitmodell*** ?
Wie vergleicht man ***Laufzeitfunktionen*** ?

Laufzeitfunktion für primitives Laufzeitmodell

A Algorithmus E Eingabe

$T_A(E)$ = Anzahl der primitiven Operationen,
die Algorithmus A braucht bei Abarbeitung von Eingabe E

Problem: Wie vergleicht man Funktionen $T_A()$ und $T_B()$ für verschiedene Algorithmen A und B ?

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Problem: Wie vergleicht man Funktionen $T_A()$ und $T_B()$ für verschiedene Algorithmen A und B ?

1. Menge der Eingaben in Klassen einteilen, z.B. nach Größe **n**
2. Innerhalb einer Klasse Funktionen zusammenfassen, z.B, nach
   schlechtesten Fall
   „durchschnittlichen" Fall
3. Die Zusammenfassungsfunktionen **asymptotisch** betrachten, also nach dem Wachstumsverhalten bei sehr großem Argument

WC-T_A(n) = max{ T_A(E) | E ist Eingabe der Größe n }

"Worst-case Laufzeit", also Laufzeit im schlechtesten Fall

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Problem: Wie vergleicht man Funktionen $T_A()$ und $T_B()$ für verschiedene Algorithmen A und B ?

1. Menge der Eingaben in Klassen einteilen, z.B. nach Größe **n**
2. Innerhalb einer Klasse Funktionen zusammenfassen, z.B, nach
   schlechtesten Fall
   „durchschnittlichen" Fall
3. Die Zusammenfassungsfunktionen **asymptotisch** betrachten, also nach dem Wachstumsverhalten bei sehr großem Argument

WC-T_A(n) = max{ T_A(E) | E ist Eingabe der Größe n }

"Worst-case Laufzeit", also Laufzeit im schlechtesten Fall

WC-T_A(n) = Average{ T_A(E) | E ist Eingabe der Größe n }

"Durchschnittliche Laufzeit", also Laufzeit im "typischen Fall"
Aber: bezügliche welcher Verteilung? Begründung?

Laufzeitanalyse von Algorithmen

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Problem: Wie vergleicht man Funktionen $T_A()$ und $T_B()$ für verschiedene Algorithmen A und B ?

1. Menge der Eingaben in Klassen einteilen, z.B. nach Größe **n**
2. Innerhalb einer Klasse Funktionen zusammenfassen, z.B, nach
   schlechtesten Fall
   „durchschnittlichen" Fall
3. Die Zusammenfassungsfunktionen **asymptotisch** betrachten, also nach dem Wachstumsverhalten bei sehr großem Argument

Für Algorithmen *A* und B sind *WC*-$T_A(n)$ und *WC*-$T_A(n)$ einfach Funktionen $\mathbb{N} \to \mathbb{R}$

Welche Funktion ist "kleiner" ?

Asymptotischer Vergleich: Welche Funktion wächst langsamer?
Welche Funktion ist kleiner für sehr großes **n**?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Def: $f,g : \mathbb{N} \to \mathbb{R}$

$f \leq_{fü} g$: für alle n, bis auf endlich viele Ausnahmen gilt $f(n) \leq g(n)$

$(\exists\, n_0 \in \mathbb{N}\ \forall\, n \geq n_0 : f(n) \leq g(n))$

Def: $g \in FÜP = \{ h : \mathbb{N} \to \mathbb{R} \mid h \geq_{fü} 0 \}$

$O(g) = \{ f \in FÜP \mid \exists c>0 : f \leq_{fü} c \cdot g \}$

$o(g) = \{ f \in FÜP \mid \forall c>0 : f \leq_{fü} c \cdot g \}$

$\Omega(g) = \{ f \in FÜP \mid \exists c>0 : f \geq_{fü} c \cdot g \}$

$\omega(g) = \{ f \in FÜP \mid \forall c>0 : f \geq_{fü} c \cdot g \}$

$\Theta(g) = O(g) \cap \Omega(g)$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Def: $g \in FÜP = \{ h : \mathbb{N} \to \mathbb{R} \mid h \geq_{fü} 0 \}$

$O(g) = \{ f \in FÜP \mid \exists c>0 : f \leq_{fü} c \cdot g \}$

$o(g) = \{ f \in FÜP \mid \forall c>0 : f \leq_{fü} c \cdot g \}$

$\Omega(g) = \{ f \in FÜP \mid \exists c>0 : f \geq_{fü} c \cdot g \}$

$\omega(g) = \{ f \in FÜP \mid \forall c>0 : f \geq_{fü} c \cdot g \}$

$\Theta(g) = O(g) \cap \Omega(g)$

$f \in O(g)$ entspricht f "$\leq$" g
$f \in o(g)$ entspricht f "$<$" g
$f \in \Omega(g)$ entspricht f "$\geq$" g
$f \in \omega(g)$ entspricht f "$>$" g
$f \in \Theta(g)$ entspricht f "$=$" g

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Wichtige Regeln

$f \in O(g) \Leftrightarrow g \in \Omega(f)$
$f \in o(g) \Leftrightarrow g \in \omega(f)$
$f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$

$f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$
$f \in o(g) \wedge g \in o(h) \Rightarrow f \in o(h)$

Wenn $f_1 \in O(g_1)$, $f_2 \in O(g_2)$

dann $f_1 + f_2 \in O(\max\{g_1,g_2\})$
$f_1 \cdot f_2 \in O(g_1 \cdot g_2)$

$0<a<b \quad 0<\alpha<\beta \quad 1<A<B$

$\log^\alpha n$ "<" $\log^\beta n$ "<" n^a "<" $n^a \log^\alpha n$ "<" n^b "<" A^n "<" $n^a A^n$ "<" B^n