ETH zürich

Übung 3

Datenstrukturen und Algorithmen, D-MATH, ETH Zurich

Programm von heute

Feedback letzte Übung

Laufzeitanalyse von (rekursiven) Funktionen

Lösen einfacher Rekurrenzgleichungen

Sortieralgorithmen

1. Feedback letzte Übung

2. Laufzeitanalyse von (rekursiven) Funktionen

Analyse

Wie oft wird `f()` aufgerufen?

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

Analyse

Wie oft wird `f()` aufgerufen?

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

Das Code-Fragment ruft `f()` $\Theta(n^2)$ mal auf: die äußere Schleife wird $n/9$ mal durchlaufen, und die innere Schleife ruft `f()` i mal auf.

Wie oft wird `f()` aufgerufen?

```
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();
}
```

Wie oft wird `f()` aufgerufen?

```
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();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.

Wie oft wird `f()` aufgerufen?

```
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();
}
```

Wir können die erste innere Schleife ignorieren, weil sie `f()` nur konstant oft aufruft.
Die zweite innere Schleife ruft `f()` $\lfloor \log_2(n) \rfloor + 1$ mal auf, in Summe haben wir $\Theta(n \log(n))$ Aufrufe.

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  if (n>0){
    g(n-1);
    f();
  }
}
```

Wie oft wird `f()` aufgerufen?

```
void g(unsigned n) {
  if (n>0){
    g(n-1);
    f();
  }
}
```

$$M(n) = M(n-1) + 1 = M(n-2) + 2 = ... = M(0) + n = n \in \Theta(n)$$

Wie oft wird `f()` aufgerufen?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    f()
  }
}
```

Wie oft wird `f()` aufgerufen?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    f()
  }
}
```

$$M(n) = 1 + M(n/2) = 1 + 1 + M(n/4) = k + M(n/2^k) \in \Theta(\log n)$$

Wie oft wird `f()` aufgerufen?

```
// pre: n is a power of 2
void g(int n){
  if (n>0){
    f();
    g(n/2);
    f();
    g(n/2);
  }
}
```

Wie oft wird `f()` aufgerufen?

```
// pre: n is a power of 2
void g(int n){
  if (n>0){
    f();
    g(n/2);
    f();
    g(n/2);
  }
}
```

$$M(n) = 2M\left(\frac{n}{2}\right) + 2 = 4M\left(\frac{n}{4}\right) + 4 + 2 = 8M\left(\frac{n}{8}\right) + 8 + 4$$
$$= n + n/2 + \ldots + 2 \in \Theta(n)$$

Wie oft wird `f()` aufgerufen?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    g(n/2);
  }
  for (int i = 0; i < n; ++i){
    f();
  }
}
```

Wie oft wird `f()` aufgerufen?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    g(n/2);
  }
  for (int i = 0; i < n; ++i){
    f();
  }
}
```

$$M(n) = 2M(n/2) + n = 4M(n/4) + n + 2n/2 = ... = (k+1)n \in \Theta(n \log n)$$

Wie oft wird `f()` aufgerufen?

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

Wie oft wird `f()` aufgerufen?

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

$T(0) = 1$

Wie oft wird `f()` aufgerufen?

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

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

Wie oft wird `f()` aufgerufen?

```
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

Wie oft wird `f()` aufgerufen?

```
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

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

Induktion

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

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

Wie oft wird `f()` aufgerufen?

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

Man sieht es auch direkt:

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

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

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

3. Lösen einfacher Rekurrenzgleichungen

Rekursionsgleichung

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

Geben Sie eine geschlossene (nicht rekursive), einfache Formel für $T(n)$ an und beweisen Sie diese mittels vollständiger Induktion. Gehen Sie davon aus, dass n eine Potenz von 2 ist.

Rekursionsgleichung

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

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

Induktion

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 $k \in \mathbb{N}$):

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

Master Methode

$$T(n) = \begin{cases} aT(\frac{n}{b}) + f(n) & n > 1 \\ f(1) & n = 1 \end{cases} \qquad (a, b \in \mathbb{N}^+)$$

1. $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0 \Longrightarrow T(n) \in \Theta(n^{\log_b a})$
2. $f(n) = \Theta(n^{\log_b a}) \Longrightarrow T(n) \in \Theta(n^{\log_b a} \log n)$
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ für eine Konstante $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ für eine Konstante $c < 1$ und alle genügend grossen $n \Longrightarrow T(n) \in \Theta(f(n))$

Beispiele

Maximum Subarray / Mergesort

$$T(n) = 2T(n/2) + \Theta(n)$$

Beispiele

Maximum Subarray / Mergesort

$$T(n) = 2T(n/2) + \Theta(n)$$

$a = 2, b = 2,\ f(n) = cn = cn^1 = cn^{\log_2 2} \overset{[2]}{\Longrightarrow} T(n) = \Theta(n \log n)$

Beispiele

Naive Matrix Multiplication Divide & Conquer[1]

$$T(n) = 8T(n/2) + \Theta(n^2)$$

[1] Wird später im Kurs betrachtet

Beispiele

Naive Matrix Multiplication Divide & Conquer[1]

$$T(n) = 8T(n/2) + \Theta(n^2)$$

$a = 8, b = 2,\ f(n) = cn^2 \in \mathcal{O}(n^{\log_2 8 - 1}) \overset{[1]}{\Longrightarrow} T(n) \in \Theta(n^3)$

[1] Wird später im Kurs betrachtet

Beispiele

Strassens Matrix Multiplication Divide & Conquer[2]

$$T(n) = 7T(n/2) + \Theta(n^2)$$

[2] Wird später im Kurs betrachtet

Beispiele

Strassens Matrix Multiplication Divide & Conquer[2]

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$a = 7, b = 2,\ f(n) = cn^2 \in \mathcal{O}(n^{\log_2 7 - \epsilon}) \overset{[1]}{\Longrightarrow} T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$

[2] Wird später im Kurs betrachtet

Beispiele

$$T(n) = 2T(n/4) + \Theta(n)$$

Beispiele

$$T(n) = 2T(n/4) + \Theta(n)$$

$$a = 2, b = 4,\ f(n) = cn \in \Omega(n^{\log_4 2+0.5}),\ 2f(n/4) = c\tfrac{n}{2} \leq \tfrac{c}{2}n^1 \overset{[3]}{\Longrightarrow} T(n) \in \Theta(n)$$

Beispiele

$$T(n) = 2T(n/4) + \Theta(n^2)$$

Beispiele

$$T(n) = 2T(n/4) + \Theta(n^2)$$

$a = 2, b = 4$, $f(n) = cn^2 \in \Omega(n^{\log_4 2+1.5})$, $2f(n/4) = \frac{n^2}{8} \leq \frac{1}{8}n^2 \overset{[3]}{\Longrightarrow}$
$T(n) \in \Theta(n^2)$

4. Sortieralgorithmen

Quiz

Nachfolgend sehen Sie drei Folgen von Momentaufnahmen (Schritten) der Algorithmen (a) Sortieren durch Einfügen, (b) Sortieren durch Auswahl und (c) Bubblesort. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Nachfolgend sehen Sie drei Folgen von Momentaufnahmen (Schritten) der Algorithmen (a) Sortieren durch Einfügen, (b) Sortieren durch Auswahl und (c) Bubblesort. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Nachfolgend sehen Sie drei Folgen von Momentaufnahmen (Schritten) der Algorithmen (a) Sortieren durch Einfügen, (b) Sortieren durch Auswahl und (c) Bubblesort. Geben Sie unter den Folgen jeweils den Namen des zugehörigen Algorithmus an.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

Auswahl

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

Bubblesort

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Einfügen

Quiz

Führen Sie auf dem folgenden Array zwei weitere Iterationen des Algorithmus Quicksort aus. Als Pivot wird jeweils das erste Element des (Sub-)Arrays genommen.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	15	10	13

Quiz

Führen Sie auf dem folgenden Array zwei weitere Iterationen des Algorithmus Quicksort aus. Als Pivot wird jeweils das erste Element des (Sub-)Arrays genommen.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	15	10	13

Quiz

Führen Sie auf dem folgenden Array zwei weitere Iterationen des Algorithmus Quicksort aus. Als Pivot wird jeweils das erste Element des (Sub-)Arrays genommen.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	15	10	13
2	7	5	6	3	**8**	**9**	15	10	13
2	3	5	6	**7**	**8**	**9**	13	10	**15**

Algorithmus NaturalMergesort(A)

Input: Array A der Länge $n > 0$
Output: Array A sortiert

```
repeat
    r ← 0
    while r < n do
        l ← r + 1
        m ← l; while m < n and A[m + 1] ≥ A[m] do m ← m + 1
        if m < n then
            r ← m + 1; while r < n and A[r + 1] ≥ A[r] do r ← r + 1
            Merge(A, l, m, r);
        else
            r ← n
until l = 1
```

Quicksort mit logarithmischem Speicherplatz

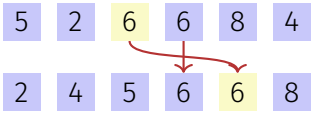
Input: Array A der Länge n. $1 \leq l \leq r \leq n$.
Output: Array A, sortiert zwischen l und r.

while $l < r$ **do**
    Wähle Pivot $p \in A[l, \ldots, r]$
    $k \leftarrow$ Partition$(A[l, \ldots, r], p)$
    if $k - l < r - k$ **then**
        Quicksort$(A[l, \ldots, k-1])$
        $l \leftarrow k + 1$
    else
        Quicksort$(A[k+1, \ldots, r])$
        $r \leftarrow k - 1$

Der im ursprünglichen Algorithmus verbleibende Aufruf an Quicksort$(A[l, \ldots, r])$ geschieht iterativ (Tail Recursion ausgenutzt!): die If-Anweisung wurde zur While Anweisung.

Stabile und in-situ-Sortieralgorithmen

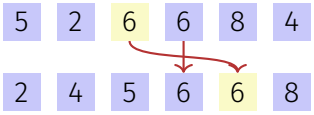
- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.



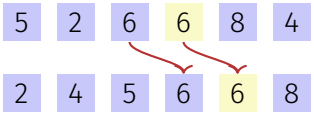
nicht stabil

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.



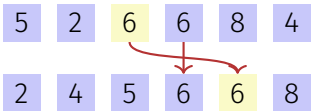
nicht stabil



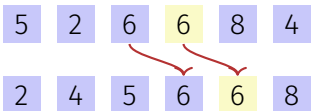
stabil

Stabile und in-situ-Sortieralgorithmen

- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.



nicht stabil



stabil

- In-situ-Algorithmen brauchen nur konstant viel zusätzlichen Speicher.

  Welche der Sortieralgorithmen sind stabil? Welche in-situ? (Wie) kann man sie stabil / in-situ machen?

Fragen?