Datenstrukturen und Algorithmen

Übung 3

FS 2020

Programm von heute

1. Feedback letzte Übung
2. Wiederholung Theorie

Eier werfen

- Strategie für beliebig viele Eier?

Eier werfen

- Strategie für beliebig viele Eier?
    - Binäre Suche, höchstens $\log_2 n$ Versuche.

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?

Eier werfen

- Strategie für beliebig viele Eier?
  - Binäre Suche, höchstens $\log_2 n$ Versuche.
- Strategie mit nur einem Ei?
  - Von unten anfangen. n Versuche.

Eier werfen

Strategie mit zwei Eiern

- 1. Ansatz. Intervalle gleicher Länge: Unterteile n in k Intervalle. Maximale Anzahl Versuche:

Eier werfen

Strategie mit zwei Eiern

- 1. Ansatz. Intervalle gleicher Länge: Unterteile n in k Intervalle.
  Maximale Anzahl Versuche: $f(k) = k + n/k - 1$
  Minimiere maximale Anzahl Versuche:

Eier werfen

Strategie mit zwei Eiern

- 1. Ansatz. Intervalle gleicher Länge: Unterteile n in k Intervalle.
  Maximale Anzahl Versuche: $f(k) = k + n/k - 1$
  Minimiere maximale Anzahl Versuche:
  $f'(k) = 1 - n/k^2 = 0 \ \Rightarrow \ k = \sqrt{n}$.
  $n = 100 \Rightarrow 19$ Versuche. $\Theta(\sqrt{n})$
- Zweiter Ansatz: Beziehe ersten Wurfversuch in die Berechnung ein mit kleiner werdenden Intervallen. Wähle kleinstes s mit
  $s + s - 1 + s - 2 + ... + 1 = s(s+1)/2 \geq 100 \Rightarrow s = 14$.
  Maximale Anzahl Versuche: $s \in \Theta(\sqrt{n})$

Asymptotisch sind beide Methoden gleich gut. Praktisch ist der zweite Ansatz vorzuziehen.

Selection-Algorithmus

- Was passiert bei vielen gleichen Elementen?
- $99, 99, \ldots, 99$, Pivot 99, kleiner Partition leer, grösser Parition hat $n-1$ mal 99.
- Kann Laufzeit auf n^2 verschlechtern
- Lösung?

Selection-Algorithmus

- Bei Gleichheit mit Pivot, wechsle Partition ab.

Selection-Algorithmus

- Bei Gleichheit mit Pivot, wechsle Partition ab.
- Erweitere Algorithmus um explizit Anzahl gleicher Elemente zu behandeln.

2. Wiederholung Theorie

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**

Master Method

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

1. $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0 \implies T(n) = \Theta(n^{\log_b a})$
2. $f(n) = \Theta(n^{\log_b a}) \implies T(n) = \Theta(n^{\log_b a} \log n)$
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n \implies T(n) = \Theta(f(n))$

Beispiele

Maximum Subarray / Mergesort

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

Beispiele

Maximum Subarray / Mergesort

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

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

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]

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

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

[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]

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

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

[2] Wird später im Kurs betrachtet

Beispiele

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

Beispiele

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

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

Beispiele

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

Beispiele

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

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

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

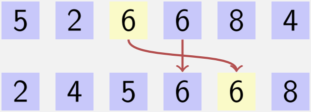
- Stabile Sortieralgorithmen ändern die relative Position von zwei gleichen Elementen nicht.

5	2	6	6	8	4
2	4	5	6	6	8

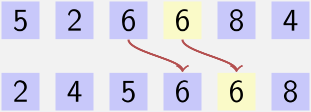
nicht stabil

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.

5	2	6	6	8	4	
						nicht stabil
2	4	5	6	6	8	

5	2	6	6	8	4	
						stabil
2	4	5	6	6	8	

- 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?