Script generated by TTT

Title: Täubig: GAD (05.06.2012)

Date: Tue Jun 05 14:32:47 CEST 2012

Duration: 87:57 min

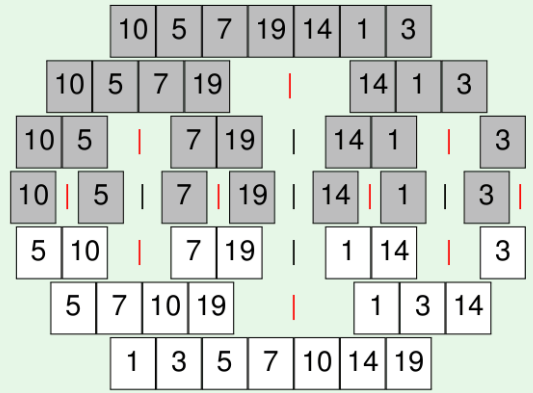
Pages: 26

MergeSort

Sortieren durch Verschmelzen

Zerlege die Eingabesequenz rekursiv in Teile, die separat sortiert und dann zur Ausgabesequenz verschmolzen werden

Beispiel



MergeSort

Sortieren durch Verschmelzen

```
mergeSort(Element[] a, int l, int r) {
  if (l == r) return;              // nur ein Element
  m = ⌊(r + l)/2⌋;                 // Mitte
  mergeSort(a, l, m);              // linken Teil sortieren
  mergeSort(a, m + 1, r);          // rechten Teil sortieren
  j = l;   k = m + 1;              // verschmelzen
  for i = 0 to r − l do
    if (j > m) { b[i] = a[k];  k++; }          // linker Teil leer
    else
      if (k > r) { b[i] = a[j];  j++; }        // rechter Teil leer
      else
        if (a[j] ≤ a[k]) { b[i] = a[j];  j++; }
        else { b[i] = a[k];  k++; }
  for i = 0 to r − l do  a[l + i] = b[i];      // zurückkopieren
}
```


Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- nichtrekursive Unterprogrammaufrufe sind einfach zu analysieren (separate Analyse des Funktionsaufrufs und Einsetzen)
- rekursive Aufrufstrukturen liefern Rekursionsgleichungen

⇒ Funktionswert wird in Abhängigkeit von Funktionswerten für kleinere Argumente bestimmt
gesucht: nichtrekursive / geschlossene Form

Anwendung: Divide-and-Conquer-Algorithmen

- gegeben: Problem der Größe $n = b^k$ $(k \in \mathbb{N}_0)$
- falls $k \geq 1$:
  - zerlege das Problem in d Teilprobleme der Größe n/b
  - löse die Teilprobleme (d rekursive Aufrufe)
  - setze aus den Teillösungen die Lösung zusammen
- falls $k = 0$ bzw. $n = 1$: investiere Aufwand a zur Lösung

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- Betrachte den Aufwand für jede Rekursionstiefe
- Anfang: Problemgröße n
- Level für Rekursionstiefe i: d^i Teilprobleme der Größe n/b^i

$\Rightarrow$ Gesamtaufwand auf Rekursionslevel i:

$$d^i c\frac{n}{b^i} = cn\left(\frac{d}{b}\right)^i \qquad \text{(geometrische Reihe)}$$

- $d < b$ Aufwand sinkt mit wachsender Rekursionstiefe; *erstes* Level entspricht konstantem Anteil des Gesamtaufwands
- $d = b$ Gesamtaufwand für jedes Level gleich groß; maximale Rekursionstiefe logarithmisch, Gesamtaufwand $\Theta(n \log n)$
- $d > b$ Aufwand steigt mit wachsender Rekursionstiefe; *letztes* Level entspricht konstantem Anteil des Gesamtaufwands

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

Geometrische Folge: $(a_i)_{i\in\mathbb{N}}$

Verhältnis benachbarter Folgenglieder konstant: $q = a_{i+1}/a_i$

n-te Partialsumme der geometrischen Reihe:

$$s_n = \sum_{i=0}^{n} a_i = a_0 + \ldots + a_n = a_0 + a_0 q + a_0 q^2 + \ldots a_0 q^n$$

Wert:

$$s_n = a_0 \frac{q^{n+1} - 1}{q - 1} = a_0 \frac{q^{n+1} - 1}{q - 1} \qquad \text{für } q \neq 1$$

bzw.

$$s_n = a_0(n + 1) \qquad \text{für } q = 1$$

Sortieren MergeSort

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- Betrachte den Aufwand für jede Rekursionstiefe
- Anfang: Problemgröße n
- Level für Rekursionstiefe i: d^i Teilprobleme der Größe n/b^i

$\Rightarrow$ Gesamtaufwand auf Rekursionslevel i:

$$d^i c\frac{n}{b^i} = cn\left(\frac{d}{b}\right)^i \qquad \text{(geometrische Reihe)}$$

- $d < b$ Aufwand sinkt mit wachsender Rekursionstiefe; *erstes* Level entspricht konstantem Anteil des Gesamtaufwands
- $d = b$ Gesamtaufwand für jedes Level gleich groß; maximale Rekursionstiefe logarithmisch, Gesamtaufwand $\Theta(n \log n)$
- $d > b$ Aufwand steigt mit wachsender Rekursionstiefe; *letztes* Level entspricht konstantem Anteil des Gesamtaufwands

H. Täubig (TUM) GAD SS'12 235 / 637

Sortieren MergeSort

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

Geometrische Folge: $(a_i)_{i\in\mathbb{N}}$

Verhältnis benachbarter Folgenglieder konstant: $q = a_{i+1}/a_i$

n-te Partialsumme der geometrischen Reihe:

$$s_n = \sum_{i=0}^{n} a_i = a_0 + \ldots + a_n = a_0 + a_0 q + a_0 q^2 + \ldots a_0 q^n$$

Wert:

$$s_n = a_0 \frac{q^{n+1} - 1}{q - 1} = a_0 \frac{q^{n+1} - 1}{q - 1} \qquad \text{für } q \neq 1$$

bzw.

$$s_n = a_0(n + 1) \qquad \text{für } q = 1$$

H. Täubig (TUM) GAD SS'12 236 / 637

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

- Betrachte den Aufwand für jede Rekursionstiefe
- Anfang: Problemgröße n
- Level für Rekursionstiefe i: d^i Teilprobleme der Größe n/b^i

$\Rightarrow$ Gesamtaufwand auf Rekursionslevel i:

$$d^i c \frac{n}{b^i} = cn\left(\frac{d}{b}\right)^i \qquad \text{(geometrische Reihe)}$$

- $d < b$ Aufwand sinkt mit wachsender Rekursionstiefe; *erstes* Level entspricht konstantem Anteil des Gesamtaufwands
- $d = b$ Gesamtaufwand für jedes Level gleich groß; maximale Rekursionstiefe logarithmisch, Gesamtaufwand $\Theta(n \log n)$
- $d > b$ Aufwand steigt mit wachsender Rekursionstiefe; *letztes* Level entspricht konstantem Anteil des Gesamtaufwands

Analyse rekursiver Funktionen

Divide-and-Conquer-Algorithmen

Geometrische Folge: $(a_i)_{i \in \mathbb{N}}$

Verhältnis benachbarter Folgenglieder konstant: $q = a_{i+1}/a_i$

n-te Partialsumme der geometrischen Reihe:

$$s_n = \sum_{i=0}^{n} a_i = a_0 + \ldots + a_n = a_0 + a_0 q + a_0 q^2 + \ldots a_0 q^n$$

Wert:

$$s_n = a_0 \frac{q^{n+1} - 1}{q - 1} = a_0 \frac{q^{n+1} - 1}{q - 1} \qquad \text{für } q \neq 1$$

bzw.

$$s_n = a_0(n+1) \qquad \text{für } q = 1$$

Übersicht

6 Sortieren
- Einfache Verfahren
- MergeSort
- Untere Schranke
- QuickSort
- Selektieren
- Schnelleres Sortieren
- Externes Sortieren

Master-Theorem

Lösung von Rekursionsgleichungen

Satz (vereinfachtes Master-Theorem)

Seien a, b, c, d positive Konstanten und $n = b^k$ mit $k \in \mathbb{N}$.

Betrachte folgende Rekursionsgleichung:

$$r(n) = \begin{cases} a & \text{falls } n = 1, \\ cn + d \cdot r(n/b) & \text{falls } n > 1. \end{cases}$$

Dann gilt:

$$r(n) = \begin{cases} \Theta(n) & \text{falls } d < b, \\ \Theta(n \log n) & \text{falls } d = b, \\ \Theta(n^{\log_b d}) & \text{falls } d > b. \end{cases}$$

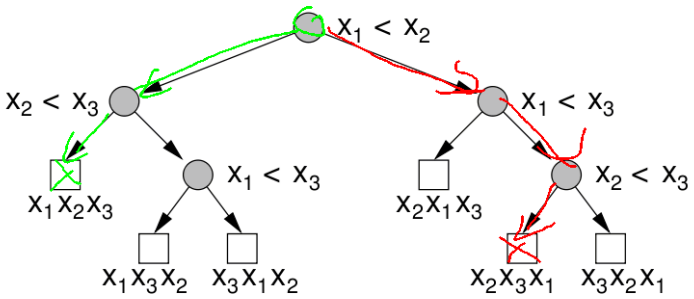
Übersicht

6 Sortieren
- Einfache Verfahren
- MergeSort
- Untere Schranke
- QuickSort
- Selektieren
- Schnelleres Sortieren
- Externes Sortieren

Untere Schranke

Vergleichsbasiertes Sortieren

Entscheidungsbaum mit Entscheidungen an den Knoten:



Untere Schranke

Vergleichsbasiertes Sortieren

muss insbesondere auch funktionieren, wenn alle n Schlüssel verschieden sind

$\Rightarrow$ Annahme: alle verschieden

Wieviele verschiedene Ergebnisse gibt es?
$\Rightarrow$ alle Permutationen:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+O\left(\frac{1}{n}\right)\right) \geq \frac{n^n}{e^n}\sqrt{2\pi n}$$

Binärbaum der Höhe h hat höchstens 2^h Blätter bzw.
Binärbaum mit b Blättern hat mindestens Höhe $\log_2 b$

$\Rightarrow h \geq \log_2(n!) \geq n\log n - n\log e + \frac{1}{2}\log(2\pi n)$

log n

Untere Schranke

Vergleichsbasiertes Sortieren

Average-case complexity

- gleiche Schranke gilt auch für randomisierte Sortieralgorithmen und durchschnittliche Laufzeit
(hier ohne Beweis)

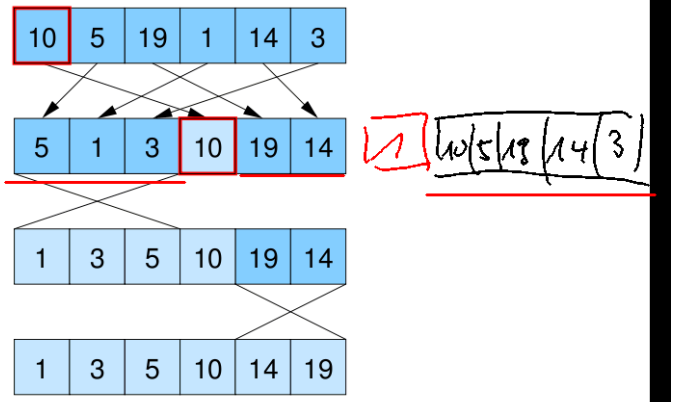
- ebenso für das scheinbar einfachere Problem festzustellen, ob ein Element in einer Sequenz doppelt vorkommt

Übersicht

6 Sortieren
- Einfache Verfahren
- MergeSort
- Untere Schranke
- QuickSort
- Selektieren
- Schnelleres Sortieren
- Externes Sortieren

QuickSort

Idee:
Aufspaltung in zwei Teilmengen, aber nicht in der Mitte der Sequenz wie bei MergeSort, sondern getrennt durch ein Pivotelement

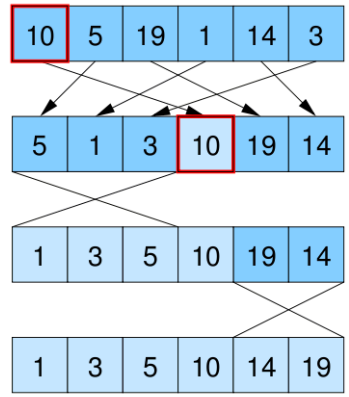


QuickSort

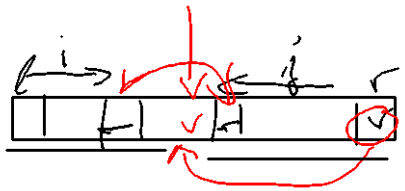
```
quickSort(Element[] a, int l, int r) {
  // a[l...r]: zu sortierendes Feld
  if (r > l) {
    v = a[r];
    int i = l − 1;   int j = r;
    do {   // spalte Elemente in a[l,...,r − 1] nach Pivot v
      do { i++ } while (a[i] < v);
      do { j−− } while (j ≥ l ∧ a[j] > v);
      if (i < j) swap(a[i], a[j]);
    } while (i < j);
    swap (a[i], a[r]);   // Pivot an richtige Stelle
    quickSort(a, l, i − 1);
    quickSort(a, i + 1, r);
}
```


QuickSort

Idee:
Aufspaltung in zwei Teilmengen, aber nicht in der Mitte der Sequenz wie bei MergeSort, sondern getrennt durch ein Pivotelement



QuickSort



```
quickSort(Element[] a, int l, int r) {
  // a[l...r]: zu sortierendes Feld
  if (r > l) {
    v = a[r];
    int i = l − 1;   int j = r;
    do {   // spalte Elemente in a[l,...,r − 1] nach Pivot v
      do { i++ } while (a[i] < v);
      do { j−− } while (j ≥ l ∧ a[j] > v);
      if (i < j) swap(a[i], a[j]);
    } while (i < j);
    swap (a[i], a[r]);   // Pivot an richtige Stelle
    quickSort(a, l, i − 1);
    quickSort(a, i + 1, r);
}
```


QuickSort

Problem:

- im Gegensatz zu MergeSort kann die Aufteilung in Teilprobleme unbalanciert sein (also nur im Optimalfall eine Halbierung)
- im worst case quadratische Laufzeit
  (z.B. wenn Pivotelement immer kleinstes oder größtes aller Elemente ist)

Lösungen:

- wähle zufälliges Pivotelement:
  Laufzeit $O(n \log n)$ mit hoher Wahrscheinlichkeit
- berechne Median (mittleres Element):
  mit Selektionsalgorithmus, spätere Vorlesung

QuickSort

Laufzeit bei zufälligem Pivot-Element

- Zähle Anzahl Vergleiche (Rest macht nur konstanten Faktor aus)
- $\bar{C}(n)$: erwartete Anzahl Vergleiche bei n Elementen

Satz

Die erwartete Anzahl von Vergleichen für QuickSort ist

$$\bar{C}(n) \leq 2n \ln n \leq 1.39 n \log n$$

QuickSort

Beweis.

- Betrachte für die Analyse die sortierte Sequenz $s' = \langle e'_1, \ldots, e'_n \rangle$
- nur Vergleiche mit Pivotelement
- Pivotelement ist nicht in den rekursiven Aufrufen enthalten

$\Rightarrow$ e'_i und e'_j werden höchstens einmal verglichen und zwar dann, wenn e'_i oder e'_j Pivotelement ist

QuickSort

Beweis.

- Zufallsvariable $X_{ij} \in \{0,1\}$
- $X_{ij} = 1 \quad \Leftrightarrow \quad e'_i$ und e'_j werden verglichen

$$\begin{aligned}\bar{C}(n) &= \mathbb{E}\left[\sum_{i<j} X_{ij}\right] = \sum_{i<j} \mathbb{E}\left[X_{ij}\right] \\ &= \sum_{i<j} 0 \cdot \Pr\left[X_{ij}=0\right] + 1 \cdot \Pr\left[X_{ij}=1\right] \\ &= \sum_{i<j} \Pr\left[X_{ij}=1\right]\end{aligned}$$

QuickSort

Lemma

$$\Pr[X_{ij} = 1] = 2/(j-i+1)$$

Beweis.

- Sei $M = \{e'_i, \ldots, e'_j\}$
- Irgendwann wird ein Element aus M als Pivot ausgewählt.
- Bis dahin bleibt M immer zusammen.
- e'_i und e'_j werden genau dann *direkt* verglichen, wenn eines der beiden als Pivot ausgewählt wird
- Wahrscheinlichkeit:

$$\Pr[e'_i \text{ oder } e'_j \text{ aus } M \text{ ausgewählt}] = \frac{2}{|M|} = \frac{2}{j-i+1}$$

□

QuickSort

Beweis.

$$\begin{aligned}\bar{C} &= \sum_{i<j} \Pr\left[X_{ij}=1\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1} \frac{2}{k} = 2\sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1} \frac{1}{k} \\ &\le 2\sum_{i=1}^{n-1}\sum_{k=2}^{n} \frac{1}{k} = 2(n-1)\sum_{k=2}^{n} \frac{1}{k} = 2(n-1)(H_n - 1) \\ &\le 2(n-1)(1+\ln n - 1) \le 2n\ln n\end{aligned}$$

□