Tutorium 5

Algorithmen I SS 14

Institut für Theoretische Informatik

Sortieren (immer noch)

Bucket Sort

- Array aus anfänglich leeren Buckets, denen jeweils ein Schlüssel zugewiesen ist
- Basierend auf den Schlüsseln werden die Elemente in die Buckets sortiert

Eigenschaften

- stabil
- nicht inplace
- Laufzeit $\mathcal{O}(n+k)$

$\Rightarrow$ Sinnvoll bei kleiner Schlüsselmenge

Radix Sort

Nutze Stabilität von Bucket Sort: Sortieren nacheinander nach einzelnen Ziffern

Mehrere Varianten

- Beginnend beim Most Significant Digit (MSD)
- Beginnend beim Least Significant Digit (LSD)

Eigenschaften

- stabil
- nicht inplace
- Laufzeit $\mathcal{O}(d * (n + k))$ für d = Anzahl digits

Beispiel

$\langle 978, 557, 963, 587, 718, 863, 497 \rangle$

Vergleichsbasiert vs. Nicht Vergleichsbasiert

Pro Nicht Vergleichsbasiert:

- asymptotisch schneller

Pro Vergleichsbasiert:

- weniger Voraussetzungen an die zu sortierenden Elemente
- Cache-Effizienz weniger schwierig
- bei langen Schlüsseln oft schneller
- robust gegen beliebige Eingabeverteilungen

Partitionierung bei Quicksort

Partitionierung mit Zeigern von beiden Seiten

Karlsruher Institut für Technologie

- Liste wird durch zwei Zeiger (i, j mit $i \leq j$) in drei Teile unterteilt:
  - am Anfang i = Anfang der Folge, j = Ende der Folge
  - bis i: Elemente $< p$
  - bis $j - 1$: unbetrachtete Elemente
  - bis r: Elemente $\geq p$
- Zeiger laufen aufeinander zu, solange die Zuteilung stimmt
- Sobald beide Zeiger bei falsch positionierten Elementen angekommen sind wird vertauscht
- Abbruch bei $i > j$

Partitionierung mit beiden Zeigern von links

- Pivotelement (p) steht an der letzten Stelle der Folge (r)
- Folge wird durch zwei Zeiger (i, j mit $i \leq j$) in drei Teile unterteilt:
  - am Anfang i = j = Anfang der Folge
  - bis $i - 1$: Elemente $\leq$ p
  - bis $j - 1$: Elemente $>$ p
  - bis $r - 1$: unbetrachtete Elemente
- Wenn Element an der j-ten Stelle $\leq p$, dann tausche die Elemente an Stelle i und j und inkrementiere i
- inkrementiere j
- Wenn j bei $r - 1$ ankommt, vertausche Element bei r mit Element bei i

Quicksort: Worst Case

Worst case bei Quicksort $\Leftrightarrow$ Pivot ist immer Max/Min

Gedankenspiel:

Array bestehenend aus nur gleichen Elementen: $\langle 2, 2, 2, 2 \rangle$

$\Rightarrow$ Standard-Quicksort schlecht bei vielen gleichen Elementen

Stattdessen: 3-way-partition

- 3 Pointer: i, j ,k
  - bis i - 1: Elemente $< p$
  - bis j - 1: Elemente $> p$
  - bis k: unbetrachtete Elemente
  - bis r: Elemente $= p$
- Wie 2-way von links
- Außer: Wenn Element an j-ter Stelle $= p$, tausche mit k-tem Element und inkrementiere j *nicht*
- Am Ende den $(= p)$ Teil zwischen $(< p)$ und $(> p)$ schieben.

Beispiel Partition

Partitioniere $\langle 16, 52, 50, 17, 80, 27, 29, 21, 23, 29, 17, 33, 50, 83 \rangle$ (29 als Pivot) mit

1. von beiden Seiten
2. von links
3. 3-way von links

Quickselect

Quickselect

- **Rang** eines Elements: Position des Elements in der sortierten Folge
- Nicht eindeutig bei mehreren gleichen Elementen!
- Gesucht: *select(s, k)* soll das Element mit Rang k in der (unsortierten) Folge s liefern
- Lösung: (einseitiges) Quicksort
- Erwartet $\mathcal{O}(n)$, Worst-Case $\mathcal{O}(n^2)$

Quickselect

Ziel: Element mit Rang k

1. Wähle ein Pivot-Element
2. Partitioniere Folge wie bei Quicksort in a ($<$), b ($=$), c ($>$)
3. Vergleiche Größe von Teillisten mit Rang:
   - Element in a: return *select(a, k)*
   - Element in b: return Pivot
   - Element in c: return *select*$(c, k - |a| - |b|)$

Beispiel Quickselect

$s = \langle 45, 31, 93, 30, 5, 67, 0, 39, 19, 41, 45 \rangle$
Finde das Element mit Rang 7 mit Quickselect (erstes Element als Pivot)

Seat Selection