Tutorium 6

Partition

- $O(n)$, inplace, instabil
- zwei Algorithmen
- beide Instabil
- Stabilität kann mit zusätzlichem Platz oder Laufzeit erkauft werden

Partition – gegenläufige Indexe

```
fn partition(array: Array<T>, pivot: T): Index
    i1 := 0
    i2 := array.size()
    while true:
        i1 := forward_search(A, (x) -> (x > pivot))
        i2 := backward_search(A, (x) -> (x < pivot))
        if i1 >= i2:
            break
        swap(arr[i1], arr[i2])
    return i1
```

- Schematisch, unterschlägt Pivot

Partition – gleichläufige Indexe

```
fn partition(array: Array<T>,  pivotIndex: Index): Index
    pivot := array[pivotIndex]
    swap(array[pivotIndex], array.back())
    i1 := 0
    i2 := 0
    while i1 < array.size():
        if array[i1] <= pivot:
            swap(array[i1], array[i2])
            ++i2
        ++i1
    swap(array[i2], array.back())
    return i2
```

Quickselect

- Erwartet in $O(n)$, worstcase in $O(n^2)$
- Klausurrelevant!

Algorithmus

- Gesucht: Element an Index I im Arrays A, wäre es sortiert
- Wähle Pivot und partitioniere
- Falls der Index des Pivot ($=: P$) kleiner als I ist:
  - $I := I - P$
  - Quickselect auf rechter Hälfte
- Sonst: Falls $P > I$:
  - Quickselect auf linker Hälfte
- Sonst:
  - Das gesuchte Element steht and Index P

Vergleichsbasiertes Sortieren

- Für sehr viele Dinge anwendbar (nicht nur zaheln)
- Bester average-case: $O(n)$
- Beweisskizze:
  - Binärbaum aller $n!$ möglichen Permutatione hat Höhe $O(\log_2(n!))$
  - Man kann zeigen™, dass $\Theta(\log_2(n!)) = \Theta(n \log_2 n)$

Bucketsort

- sortiert Elemente die als Index in ein Array benutzt werden können
- Lege ein Array von Buckets aller möglichen Werte an und füge alle Elemente in die richtigen Buckets ein
- nicht vergleichsbasiert, stabil, O(n)

Radixsort

- Teilt die Elemente in k absteigend wichtige Unterelemente
  - $123 \rightarrow [1, 2, 3]$, $k = 3$
- Sortiert der Reihe nach über die Unterelemente mit Bucketsort
  - funktioniert, da Bucketsort stabil ist
- $O(n \log k)$
- Primär für Ganzzahlen, prinzipiell aber auch für Fließkommazahlen und Arrays von beschränkter Größe

Kreativaufgabe

- Heute keine vom Lehrstuhl
- Wer letztes mal nicht dazu kam: Langsames und vertauschungsarmes Sortieren
- Sortieren in $O(n)$:
  - Gegeben:
    - Ein Stapel Pfannkuchen
    - Möglichkeit diesen von oben zu betrachten
    - Möglichkeit beliebigen oberen Teil zu wenden
  - Gesucht:
    - Algorithmus der den Stapel in $O(n)$ der Größe nach sortiert
    - Erklärung warum das in $O(n)$ funktioniert