IN-PLACE MERGE SORT

Von Kompromissen und der Eleganzlüge

Florian Sihler

26. Juni 2022
SP, Universität Ulm

IN-PLACE MERGE SORT

Von Kompromissen und der Eleganzlüge

Florian Sihler

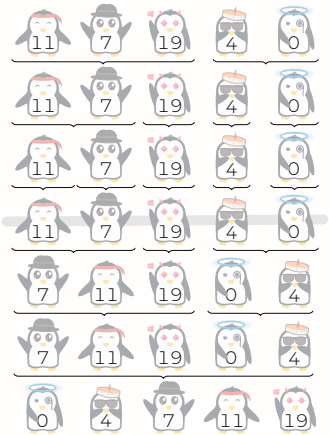
Officially supported by the Ping-Ping-Uh-Uh-Foundation for Emotional Peeps. Peep n' peace and wa… oh look, there's cake!

26. Juni 2022
SP, Universität Ulm

2 Merge Sort an sich

```
def sort(list: List) → List:
    if len list ≤ 1 : ret list;
    left, right ← split(list);
    ret merge(sort(left), sort(right));

def merge(a: List, b: List) → List:
    merged ← empty List;
    while a & b aren't empty :
        x ← a[0] ≤ b[0] ? a : b;
        merged.add(x.remove(0));
    merged.add(a not empty ? a : b);
    ret merged;
```



Wir werden nur den *aufsteigenden* Fall betrachten. Ein Beispiel für die Hordeeeee...

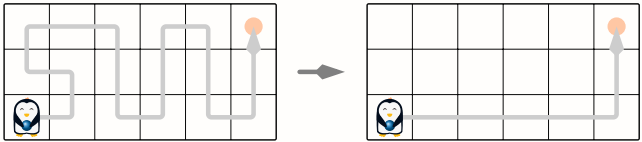
[4]: *LaTeX-Package, tikzpingus* Sihler, 2021

3 In-Place

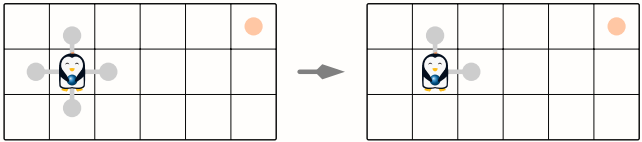
- Bedeutung variiert:
  - Konstanter zusätzlicher Speicher: $\mathcal{O}(1)$.
    Eventuell auch durch Compiler-Optimierungen
  - Konstante zusätzliche Zeiger-/Indexzugriffe
  - Konstanter Speicher für die Eingabemanipulation
  - …
- Wenn nicht In-Place: „Out-Of-Place“

4 Nichts ist umsonst

- Die Laufzeit verbessern wir durch:
  - Entfernen redundanter Operationen

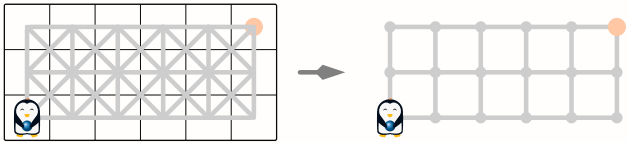


  - Ausnutzen der Einschränkungen (z.B. nur Zahlen)



Nichts ist umsonst

- Die Laufzeit verbessern wir durch:
  - Geschicktes abstrahieren (z.B. Reduktion)



  - Zusätzlichen Speicher (z.B. als Cache)

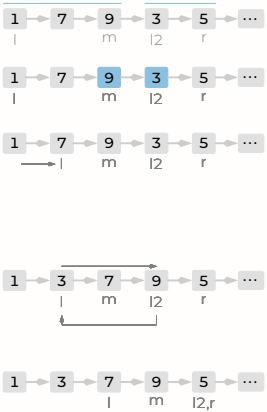
Nichts ist umsonst

- Die Laufzeit verbessern wir durch:
  - Entfernen redundanter Operationen
  - Ausnutzen der Einschränkungen
  - Geschicktes abstrahieren
  - Zusätzlichen Speicher
- Gängigster Kompromiss: Speicher vs. Laufzeit

7 Zeiger-Vergleich

Jetzt hat sich die Zeitkomplexität verschlechtert.
Von $\mathcal{O}(n \log n)$ auf $\mathcal{O}(n^2)$.

```
void merge(int[] arr, int l, int m, int r) {
    int l2 = m + 1;
    if(arr[m] <= arr[l2]) return;
    while(l <= m && l2 <= r) {
        if(arr[l] <= arr[l2]) { l++; }
        else { // shift elements from l, 1 to right
            int shift = arr[l2], idx = l2;
            while(l < idx) {
                arr[idx] = arr[idx - 1]; idx--;
            }
            arr[l] = shift;
            l++; m++; l2++;
}   }   }
```



InplaceMerge.java

Iteratives Split

Und was passiert hier?
Eine wilde Lüge…

```
void sort(int[] arr) {
   int n = arr.length - 1;
   for (int len = 1; len <= n; len = 2*len) {
      for (int left = 0; left < n; left += 2*len) {
         int mid = Math.min(left + len - 1, n);
         int right = Math.min(left + 2*len - 1, n);
         merge(arr, left, mid, right);
      }
   }
}
```

1. Beim Merge steigt die Listenlänge getreu 2^l.
2. Einstieg ist direkt bei ein-elementigen Listen möglich.

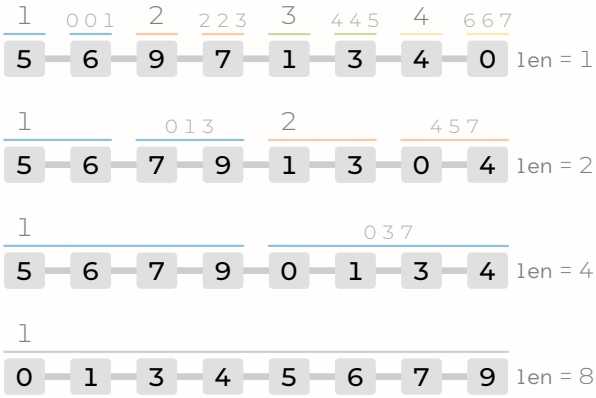
IterativeSplit.java

Iteratives Split

Falls das Array zu Beginn nur einen oder keinen Merge brauchen würde.

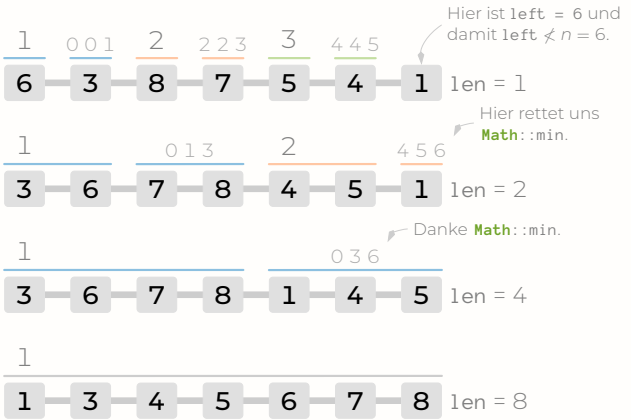
Die drei kleinen, hellgrauen Zahlen markieren die Indizes von `left`, `mid` und `right`.

```
void sort(int[] arr) {
   int n = arr.length - 1;
   for (int len = 1; len <= n; len = 2*len) {
      for (int left = 0; left < n; left += 2*len) {
         int mid = Math.min(left + len - 1, n);
         int right = Math.min(left + 2*len - 1, n);
         merge(arr, left, mid, right);
}  }  }
```



Iteratives Split

Was passiert, wenn die Länge keine Zweierpotenz ist?




```
void sort(int[] arr) {
   int n = arr.length - 1;
   for (int len = 1; len <= n; len = 2*len) {
      for (int left = 0; left < n; left += 2*len) {
         int mid = Math.min(left + len - 1, n);
         int right = Math.min(left + 2*len - 1, n);
         merge(arr, left, mid, right);
}  }  }
```

Es bleibt festzuhalten…

- Der zusätzliche Speicher ist nun gänzlich in $\mathcal{O}(1)$
- Aber die Laufzeit ist nun in $\mathcal{O}(n^2)$
  - Dazu trägt der iterative Sort „nur“ $\mathcal{O}(n \log n)$ bei
  - Die Komplexität entspringt dem Merge…

12 Abschließendes

- (Vermutlich) geht nicht unter $\mathcal{O}(n \log^2(n))$
- Dies erreicht man durch eine Shell-Sort-Inspiration
- Es gibt viele weitere Derivate, wie den „Natural Mergesort“…

[2]: *Efficiently merging two sorted arrays with O(1) extra space* , 2021

[5]: *The algorithm design manual* Skiena, 2020

[3]: *Merge path-parallel merging made simple* Odeh, 2012

[1]: *Sublinear merging and natural mergesort* Carlsson, 1993

[1] Svante Carlsson, Christos Levcopoulos und Ola Petersson. „Sublinear merging and natural mergesort“. 1993

[2] *Efficiently merging two sorted arrays with O(1) extra space*. 2021

[3] Saher Odeh u. a. „Merge path-parallel merging made simple“. 2012

[4] Florian Sihler. *LaTeX-Package, tikzpingus*. 2021

[5] Steven S. Skiena. *The algorithm design manual*. 2020

Florian Sihler
Ulm, 26. Juni 2022

Slides and LaTeX-sources on GitHub!