8. Sortieren I

Einfache Sortierverfahren

8.1 Einfaches Sortieren

Sortieren durch Auswahl, Sortieren durch Einfügen, Bubblesort
[Ottman/Widmayer, Kap. 2.1, Cormen et al, Kap. 2.1, 2.2, Exercise 2.2-2, Problem 2-2

Problemstellung

Eingabe: Ein Array $A = (A[1], ..., A[n])$ der Länge n.
Ausgabe: Eine Permutation A' von A, die sortiert ist: $A'[i] \leq A'[j]$ für alle $1 \leq i \leq j \leq n$.

Algorithmus: `IsSorted`(A)

Input: Array $A = (A[1], ..., A[n])$ der Länge n.
Output: Boolesche Entscheidung "sortiert" oder "nicht sortiert"

```
for i ← 1 to n − 1 do
    if A[i] > A[i + 1] then
        return "nicht sortiert";
return "sortiert";
```

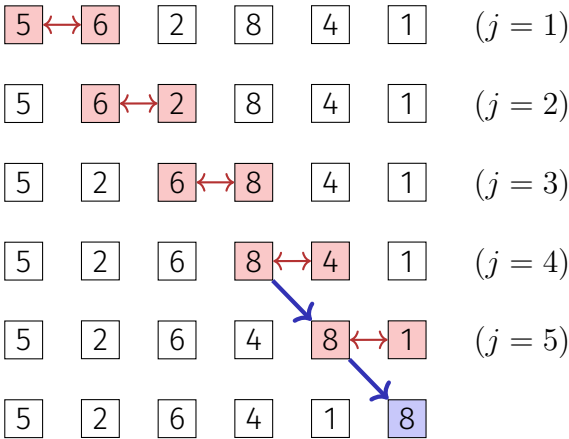
Beobachtung

IsSorted(A):“nicht sortiert”, wenn $A[i] > A[i+1]$ für ein i.

$\Rightarrow$ Idee:

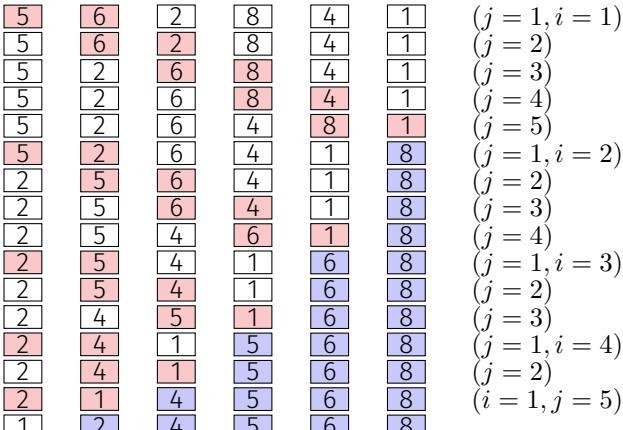
for $j \leftarrow 1$ **to** $n-1$ **do**
  if $A[j] > A[j+1]$ **then**
    swap($A[j], A[j+1]$);

Ausprobieren



- Nicht sortiert! ☹.
- Aber das grösste Element wandert ganz nach rechts.
  $\Rightarrow$ Neue Idee! ☺

Ausprobieren



- Wende das Verfahren iterativ an.
- Für $A[1, \ldots, n]$, dann $A[1, \ldots, n-1]$, dann $A[1, \ldots, n-2]$, etc.

Algorithmus: `Bubblesort`

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

```
for i ← 1 to n − 1 do
    for j ← 1 to n − i do
        if A[j] > A[j + 1] then
            swap(A[j], A[j + 1]);
```

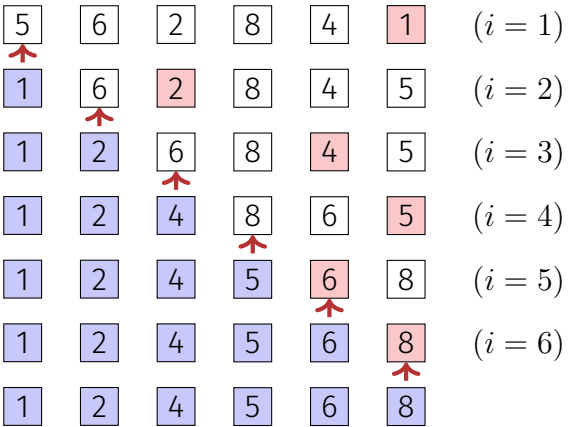
Analyse

Anzahl Schlüsselvergleiche $\sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = \Theta(n^2)$.
Anzahl Vertauschungen im schlechtesten Fall: $\Theta(n^2)$

Was ist der schlechteste Fall?

Wenn A absteigend sortiert ist.

Sortieren durch Auswahl



- Auswahl des kleinsten Elementes durch Suche im unsortierten Teil $A[i..n]$ des Arrays.
- Tausche kleinstes Element mit dem ersten Element des unsortierten Teiles.
- Unsortierter Teil wird ein Element kleiner ($i \to i+1$). Wiederhole bis alles sortiert. ($i = n$)

Algorithmus: Sortieren durch Auswahl

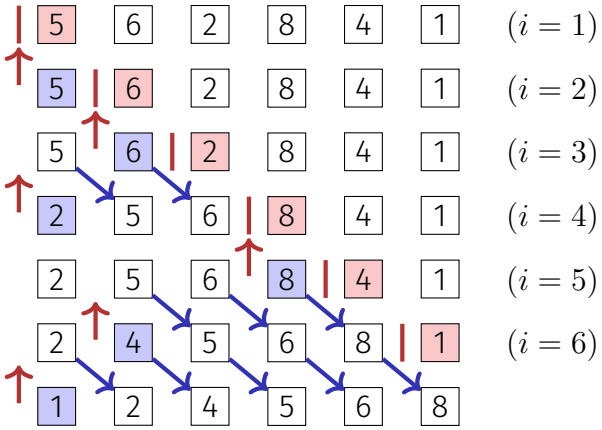
Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

```
for i ← 1 to n − 1 do
    p ← i
    for j ← i + 1 to n do
        if A[j] < A[p] then
            p ← j;
    swap(A[i], A[p])
```

Analyse

Anzahl Vergleiche im schlechtesten Fall: $\Theta(n^2)$.
Anzahl Vertauschungen im schlechtesten Fall: $n - 1 = \Theta(n)$

Sortieren durch Einfügen



- Iteratives Vorgehen: $i = 1...n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen

Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

Im schlechtesten Fall viele Elementverschiebungen.

Welchen Vorteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

Der Suchbereich (Einfügebereich) ist bereits sortiert. Konsequenz: binäre Suche möglich.

Algorithmus: Sortieren durch Einfügen

Input: Array $A = (A[1], \ldots, A[n])$, $n \geq 0$.
Output: Sortiertes Array A

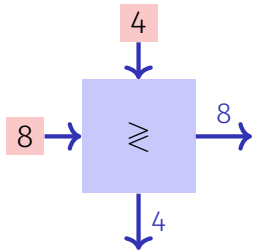
```
for i ← 2 to n do
    x ← A[i]
    p ← BinarySearch(A, 1, i − 1, x); // Kleinstes p ∈ [1, i] mit A[p] ≥ x
    for j ← i − 1 downto p do
        A[j + 1] ← A[j]
    A[p] ← x
```

Analyse

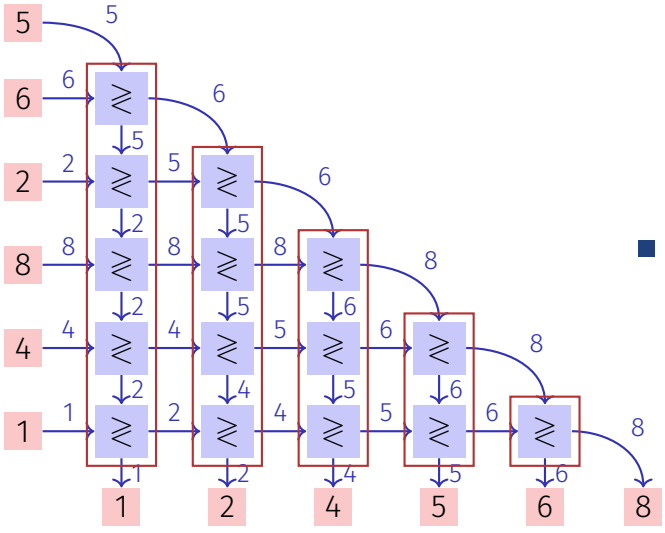
Anzahl Vergleiche im schlechtesten Fall:
$\sum_{k=1}^{n-1} a \cdot \log k = a \log((n-1)!) \in \Theta(n \log n)$.
Anzahl Vertauschungen im schlechtesten Fall: $\sum_{k=2}^{n}(k-1) \in \Theta(n^2)$

Anderer Blickwinkel

Sortierknoten:

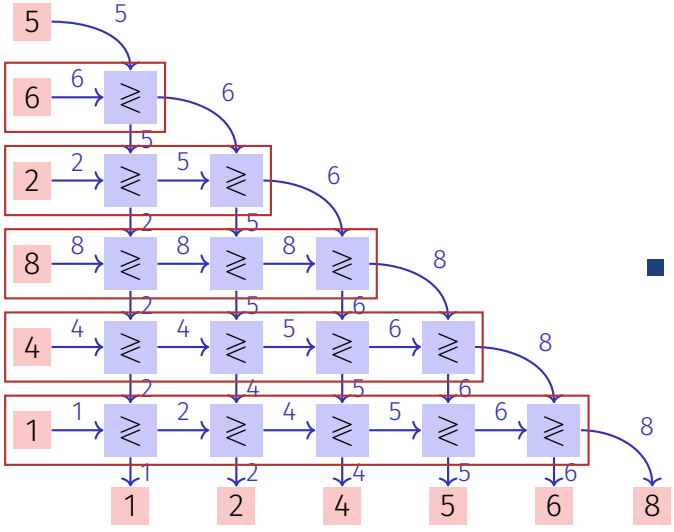


Anderer Blickwinkel



- Wie Selection Sort [und wie Bubble Sort]

Anderer Blickwinkel



- Wie Insertion Sort

Schlussfolgerung

Selection Sort, Bubble Sort und Insertion Sort sind in gewissem Sinne dieselben Sortieralgorithmen. Wird später präzisiert. [9]

[9] Im Teil über parallele Sortiernetzwerke. Für sequentiellen Code gelten natürlich weiterhin die zuvor gemachten Feststellungen.

Shellsort (Donald Shell 1959)

Intution: Verschieben weit entfernter Elemente dauert lange bei obigen naiven Verfahren

Insertion Sort auf Teilfolgen der Form $(A_{k \cdot i})$ $(i \in \mathbb{N})$ mit absteigenden Abständen k. Letzte Länge ist zwingend $k = 1$.

Worst-case Performance hängt kritisch von den gewählten Teilfolgen ab. Beispiele:

- Ursprünglich mit Folge $1, 2, 4, 8, ..., 2^k$ konzipiert. Laufzeit: $\mathcal{O}(n^2)$
- Folge $1, 3, 7, 15, ..., 2^k - 1$ (Hibbard 1963). $\mathcal{O}(n^{3/2})$
- Folge $1, 2, 3, 4, 6, 8, ..., 2^p 3^q$ (Pratt 1971). $\mathcal{O}(n \log^2 n)$

Shellsort

9	8	7	6	5	4	3	2	1	0	
2	8	7	6	5	4	3	9	1	0	insertion sort, $k = 7$
2	1	7	6	5	4	3	9	8	0	
2	1	0	6	5	4	3	9	8	7	
2	1	0	3	5	4	6	9	8	7	insertion sort, $k = 3$
2	1	0	3	5	4	6	9	8	7	
2	1	0	3	5	4	6	9	8	7	
0	1	2	3	4	5	6	7	8	9	insertion sort, $k = 1$