2.3 Sortieren

Selection-Sort

- Idee:
  - Suche kleinstes / größtes Element
  - Kopiere es an die erste Stelle
  - Wende den gleichen Algorithmus auf die restlichen (n−1) Elemente an
- Vorteil: Jedes Objekt wird nur dreimal kopiert (lohnt sich bei großen Objekten)

Selection-Sort: Algorithmus

```
• SelectionSort( A[1..n] )
    i ← 1
    while i < n do
      min ← i
      j ← i+1
      while j ⩽ n do
        if A[ j ] < A[min] then
          min ← j
        j ← j+1
      swap( A[i], A[min] )
      i ← i+1
```


Selection-Sort: Beispiel

Array mit 512 Einträgen. Je Wert eine Spalte, Höhe des Wertes entspriche Höhe in Spalte

Selection-Sort: Aufwand

- Aufwandsabschätzung
  - Swap(a,b) : { c ← a; a ← b; b ← c } = 3 x Kopieren
  - n x Swap = 3 × n × C_{ass} = O(n)
  - Im i-ten Schleifendurchlauf werden (n−i) Vergleiche durchgeführt
  - (n−1)+(n−2)+...+1 = n×(n−1)/2 = O(n^2)
- Best = Worst = Average Case

Egal, wie das Array aussieht, die Komplexität bleibt gleich. Dies hat den Grund, dass in jedem Durchlauf das
gesamte Array durchlaufen wird, um das jeweils kleinste Element in der Restliste zu ermitteln.

Bubble-Sort

- Idee
  - Nutze die Vergleiche in Selection-Sort, um den unsortierten Teil A[i] ... A[n] der Objekte vorzusortieren.
  - Innere Schleife läuft in umgekehrter Richtung.
  - Swap-Operation in der inneren Schleife

Betrachte Restliste und vertausche immer zwei hintereinander folgende Element, wenn das hintere Element kleiner ist als das vordere.
Dies wird vom letzten Element beginnend gemacht (innere Schleife läuft in umgekehrter Reihenfolge).

Bubble-Sort: Algorithmus

- BubbleSort(A[1..n])

```
i ← 1
while i < n do
    j ← n
    while j > i do
        if A[ j ] < A[ j−1] then
            swap(A[ j ], A[ j−1])
        j ← j−1
    i ← i+1
```


Bubble-Sort: Beispiel

Der Algorithmus hier ist eine etwas veränderte Variante des vorgestellen. Hier wird von vorne ausgehend immer das größere Element nach hinten vertauscht, so dass das Array von hinten nach vorne sortiert wird.

Bubble-Sort: Vorsortierung

- Vorteil: Pro innerem Schleifendurchlauf wird mehr Ordnung geschaffen.
- Nachteil: Vertauschung nur zwischen direkten Nachbarn. Objekte, die weit wandern werden oft kopiert.
- Vorsortierung kann nicht ausgenutzt werden.

Bubble-Sort: Aufwand

- Aufwandsabschätzung
  - # Vergleiche unabhängig von der Verteilung
    $n \times (n-1)/2 = O(n^2)$
  - # Swaps
    - Best Case: 0 (Elemente schon sortiert)
    - Average Case: $n \times (n-1)/4 = O(n^2)$
    - Worst Case: $n \times (n-1)/2 = O(n^2)$
      (Inverse Vorsortierung)

Bei Rückwärtssortierung wird das kleinste Element am Ende sukkzessive nach ganz vorne vertauscht, usw.

Insertion-Sort

- Idee
  - Vorbild: manuelles Sortieren
  - In jedem Schritt:
    Nimm das nächste Element und füge es in der schon sortierten Teilfolge an der richtigen Stelle ein.

Idee: Sortieren, so wie wir es manuell tun würden. Beispiel: Karten sortieren. Man steckt gezogene Karte immer an die richtige Stelle auf der Hand.

Insertion-Sort: Algorithmus

- InsertionSort(A[1..n])

```
i ← 2
while i ⩽ n do
    h ← A[ i ]
    j ← i
    while j > 1 and A[ j−1] > h do
        A[ j ] ← A[ j−1]
        j ← j−1
    A[ j ] ← h
    i ← i+1
```


Eine Karte ist natürlich schon sortiert. Von daher starten wir ab Index 2.
Intuition: Nimm erstes nichtsortiertes Element und füge es an der richtigen Stelle im sortierten Teil der Liste ein. Wiederhole dies solange, bis kein nichtsortiertes Element mehr vorhanden ist.

Insertion-Sort: Beispiel

Insertion-Sort: Aufwand

- Aufwandsabschätzung
  - Vergleiche
    - Best Case: $n-1$ (vollständig vorsortiert)
    - Worst Case: $n \times (n-1)/2 = O(n^2)$ (invers vorsortiert)
    - Average Case: $n \times (n-1)/4 = O(n^2)$ (alle Listenplätze gleich wahrscheinlich)
- Kopieren:
  Faktor 3 besser, da einfaches Kopieren anstatt Swap(), aber noch immer $O(n^2)$

Der Best-Case ist mitunter das Beste, was man bei den vorgestellten einfachen Sortieralgorithmen erreichen kann.
Swap ist dreimal so teuer wie einfaches Kopieren, da man zum Vertauschen zweier Elemente drei Kopieroperationen
durchführen muss:

```
var a = 2;
var b = 3;

swap(a, b) :
  var t = a;
  a = b;
  b = t;
```

Insertion-Sort: Vorsortierung

- Größere Elemente treten mit höherer Wahrscheinlichkeit am Ende der Folge auf.
- Größere Elemente werden später in die sortierte Teilfolge eingeordnet.
- Dazu sind weniger Vergleiche notwendig, denn größere Elemente stehen auch in der sortierten Teilfolge weiter hinten. (Einsortieren von hinten)
- Nahezu linearer Aufwand für „fast sortierte“ Folgen.

Einfache Sortierverfahren

Vergleiche	Best	Average	Worst
Selection-Sort	$\sim n^2/2$	$\sim n^2/2$	$\sim n^2/2$
Bubble-Sort	$\sim n^2/2$	$\sim n^2/2$	$\sim n^2/2$
Insertion-Sort	$\sim n$	$\sim n^2/4$	$\sim n^2/2$

Kopieren	Best	Average	Worst
Selection-Sort	$\sim 3n$	$\sim 3n$	$\sim 3n$
Bubble-Sort	0	$\sim 3n^2/4$	$\sim 3n^2/2$
Insertion-Sort	$\sim 2n$	$\sim n^2/4$	$\sim n^2/2$