


**Zahlen
steigen wie
Seifenblasen
bis zum Ende**

[illegible]

"Bubble-Up"
(größtes ans Ende bewegen)

Vergleiche Zählen



$O(n)$

"genau so lange, wie einmal durch die Liste zu gehen"

BubbleSort

5 7+ 30 42 78 2 95

77 10 42 2 ~~11~~ 78 95

77 30 41 28 95

2 5 77 30 42 78 95

Komplexität:

"BubbleSort"

**Länge
der
Liste: n**

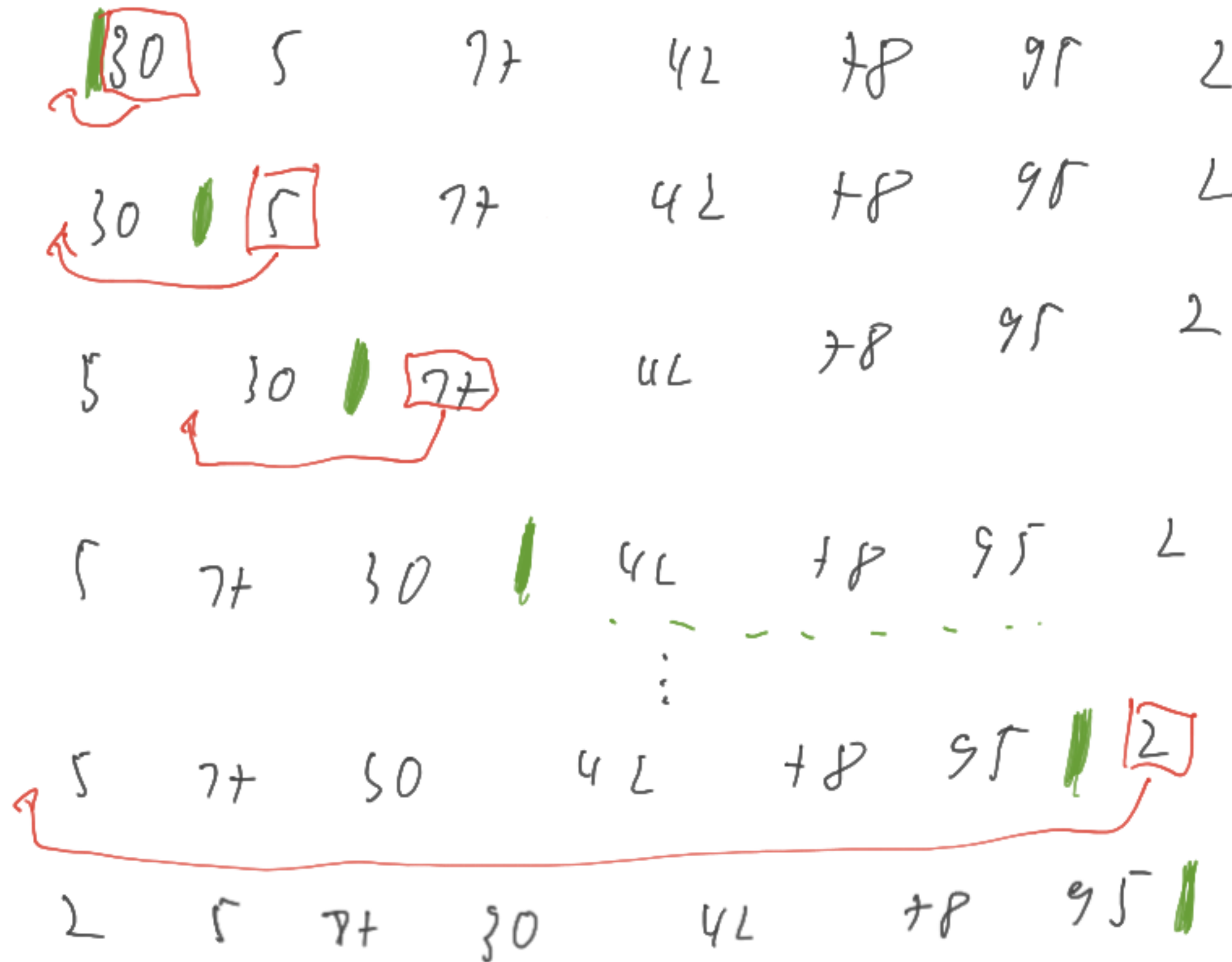
Vergleiche Zählen

Verfahren: n mal BubbleUp

 $O(n^2)$

InsertionSort

Komplexität:



Länge
der
Liste: n

Vergleiche
Zählen

n Mal das jeweils nächste
Element links einsortieren
"bubbleDown()"

jedes Einsortieren kostet n
Schritte.

$O(n^2)$

SelectionSort

130 5 77 78 42 95 2
2 130 5 77 78 42 95
2 5 130 77 78 42 95
⋮

Komplexität:

Länge
der
Liste: n

Vergleiche
Zählen

n Mal das jeweils kleinste
Element aus dem
unsortierten Teil links
anhängen.

jedes Suchen kostet n
Schritte.

$O(n^2)$

Je nach Art
der Liste
kostet auch
das Anhängen
lineare Zeit.

SelectionSort

Variante 2

30 5 77 78 42 95 2
2 5 77 78 42 95 30
2 5 77 78 42 95 30
2 5 77 78 42 95 30
2 5 77 30 42 95 78
⋮

Komplexität:

Länge
der
Liste: n

Vergleiche
Zählen

n Mal das jeweils kleinste
Element aus dem
unsortierten Teil mit dem
Anfang des unsortierten
Teils vertauschen.

jedes Suchen kostet n
Schritte.

$O(n^2)$

BubbleSort

**Bringe das
größte
Element nach
ganz rechts.**

Einmal jedes
Element mit dem
rechten Nachbarn
vertauschen, falls
diese falsch sortiert
sind.

**n Mal
wiederholen.**

InsertionSort

**Annahme:
Sortierter
Teil links.**

**Lasse das
nächste
Element nach
links
einsinken.**

Das Element so
lange mit dem
linken Nachbarn
vertauschen, bis es
größer ist als der
linke Nachbar.

**n Mal
wiederholen.**

SelectionSort

**Annahme:
Sortierter
Teil links.**

**Suche das kleinste
Element im
unsortierten Teil
und tausche es nach
vorne.**

**n Mal
wiederholen.**