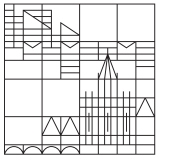
Universität Konstanz



Konzepte der Informatik

Algorithmik
Iterative Verfahren

Barbara Pampel

Universität Konstanz, WiSe 2023/2024

Inhalt

Inhalt

Sortieren (formal)

- Gegeben
  - Menge von Objekten U
  - Schlüsselmenge S mit linearer Ordnung $\leq$
  - Schlüsselfunktion $s : U \to S$
- Gesucht
  - Permutation $i_1, i_2, \ldots, i_n$ der Objekte aus U, so dass
  - $\forall 1 \leq j \leq n-1 : s(u_{i_j}) \leq s(u_{i_{j+1}})$
- Für Zahlen ist die Schlüsselfunktion die Identitätsfunktion
- Alternativ binäre Vergleichsfunktion
  - $c : U \times U \to \mathbb{Z}$
  - es soll gelten $\forall 1 \leq j \leq n-1 : c(u_{i_j}, u_{i_{j+1}}) \leq 0$
  - siehe `Comparable`-Interface in Java

Stabile Sortierverfahren I

- Verschiedene Objekte können gleichen Schlüsselwert haben
  - Studenten, bei denen nur der Vorname verglichen wird
- Auf einer Menge können mehrere Schlüsselfunktion s_i definiert sein
  - Vergleich nur nach Nachname, nur nach Vorname, nach Matrikelnummer, ...
- Stabile Sortierverfahren stellen sicher, dass
  - Objekte, die den gleichen Schlüsselwert für s_1 haben, aber unterschiedliche Werte für s_2,
  - nach der Sortierung basierend auf s_1 immer noch in der gleichen Reihenfolge sind

Stabile Sortierverfahren II

- Beispiel
  - Studenten vorher sortiert nach Matrikelnummer (s_2)
  - jetzt Sortieren nach Nachname (s_1)
  - innerhalb des gleichen Nachnamens bleibt die Sortierung nach Matrikelnummer erhalten

123456	Schmidt
234567	Huber
345678	Huber
456789	Degen

456789	Degen
234567	Huber
345678	Huber
123456	Schmidt

Stabil

456789	Degen
345678	Huber
234567	Huber
123456	Schmidt

Instabil

Stabile Sortierverfahren III

- Formal
  - $s_1 : U \to S_1$ und $s_2 : U \to S_2$
  - angenommen, eine Eingabe ist nach s_2 vorsortiert
    (d.h. gegeben eine Eingabereihenfolge)
  - ein Verfahren ist stabil, wenn nach einer Sortierung nach s_1 gilt:
  - bei gleichem Wert für s_1 bleiben die Elemente nach s_2 sortiert.

Inhalt

Iteration

- Charakteristisch ist das Wiederholen eines Prozesses/Anweisungsblocks
- dadurch schrittweise Annäherung an eine Lösung
- meist durch For- oder While-Schleife

Multiplikation $s = a * b$

```
s = 0
for 1 ≤ i ≤ a
    s = s + b
```

Divisionsrest $r = a \mod b$

```
r = a
while r ≥ b
    r = r − b
```

Sortieren durch Auswahl

512 87 503 61 908 170 897

Sortieren durch Auswahl

512 87 503 61 908 170 897

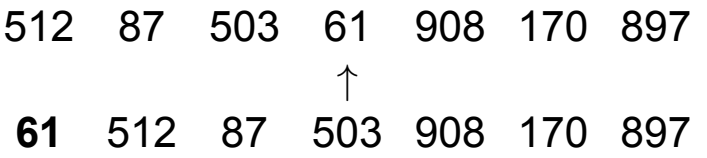
- Die wohl intuitivste Idee:
  - Extremum suchen $\Rightarrow$ an ein Anfang oder Ende der Sequenz legen/stellen
  - Extremum im Restbereich suchen $\Rightarrow$ daneben legen/stellen
  - usw.
- Algorithmisches Vorgehen (wiederholte **Minimums**suche)
  - Aufteilen in zwei Bereiche
    - R, den unsortierten Restbereich der Liste
    - L, den bereits sortierten Bereich der Liste
  - Beginn mit $L = \varnothing$
  - Ende wenn $|R| = 1$
  - Minimum in R suchen und ans Ende von L setzen

Beispiel

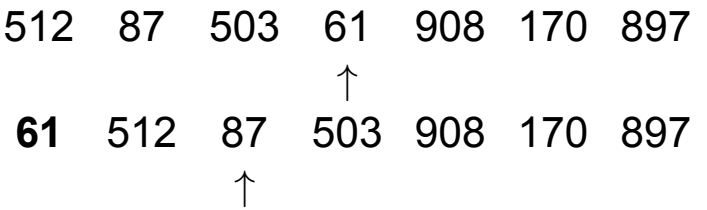
512 87 503 61 908 170 897

Beispiel

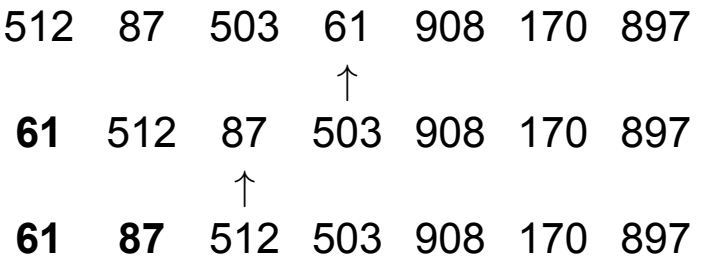
Beispiel



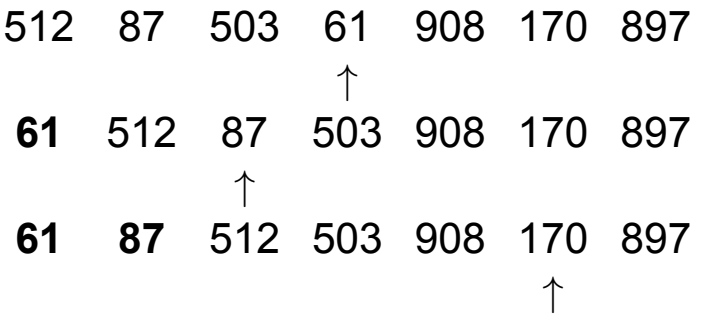
Beispiel



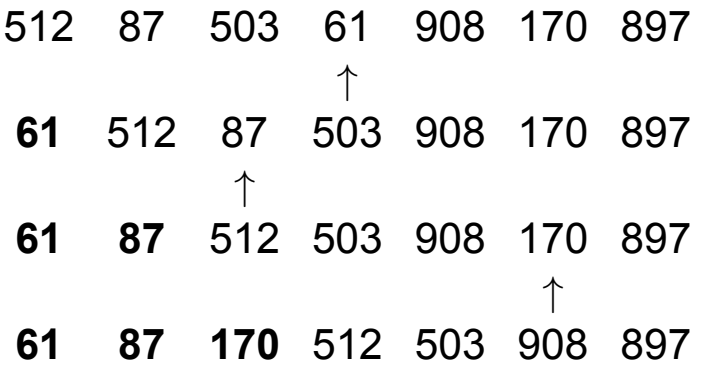
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897

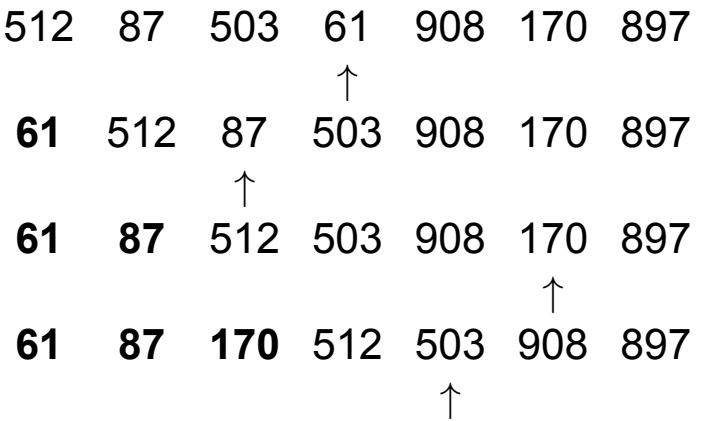
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897

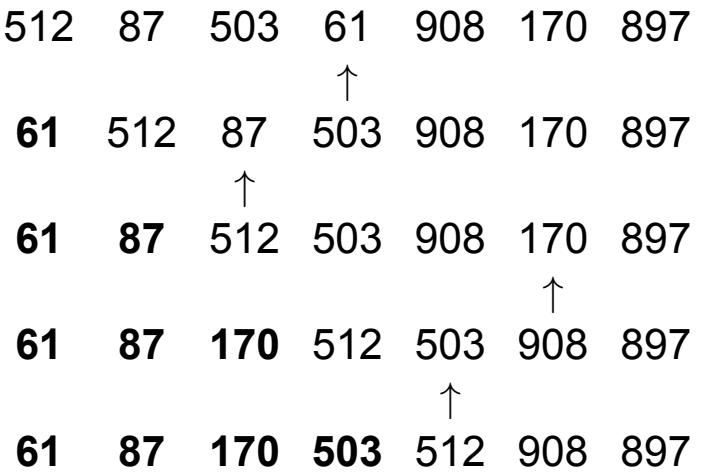
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897

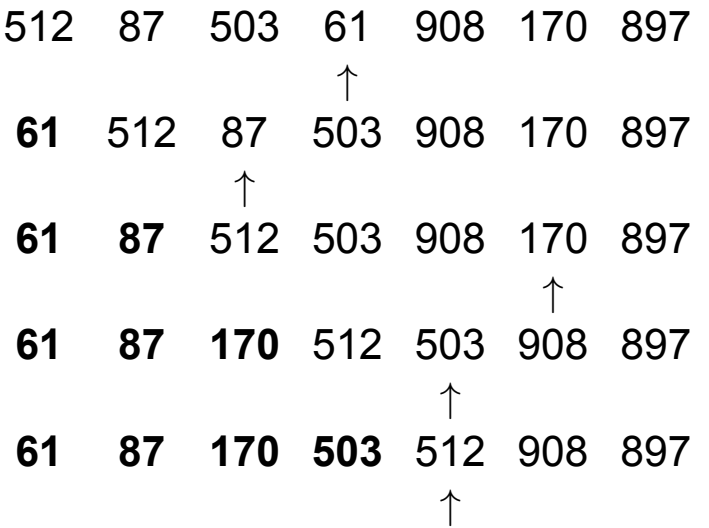
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897

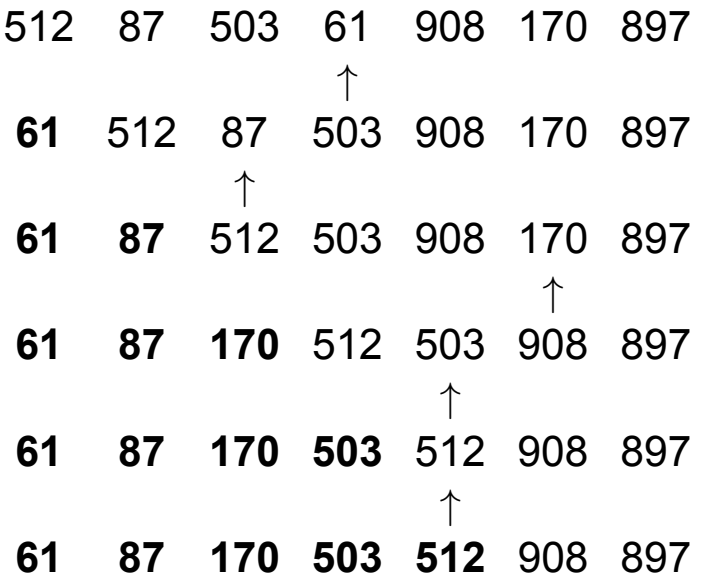
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897
61 87 170 503 512 908 897

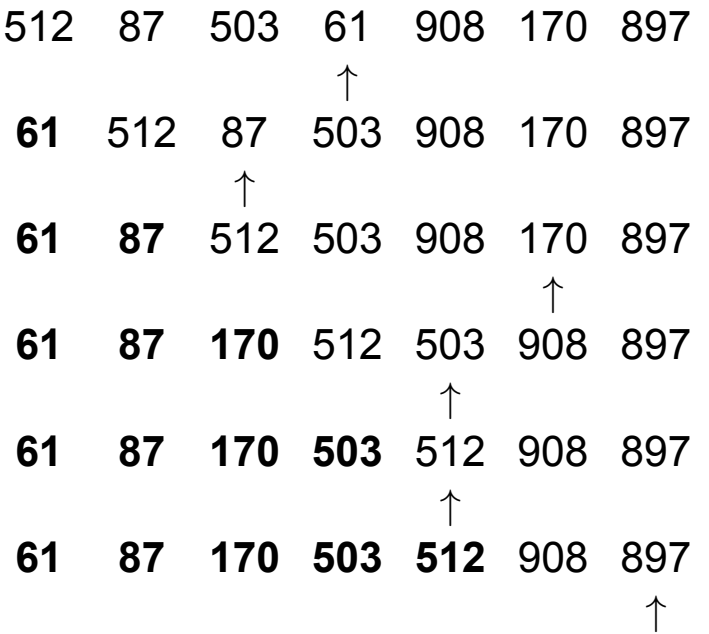
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897
61 87 170 503 512 908 897

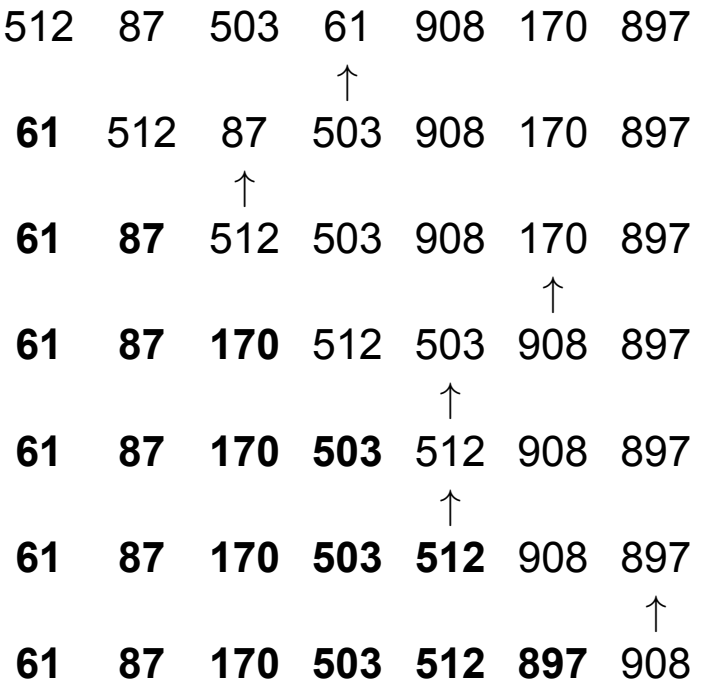
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897
61 87 170 503 512 908 897
61 87 170 503 512 908 897

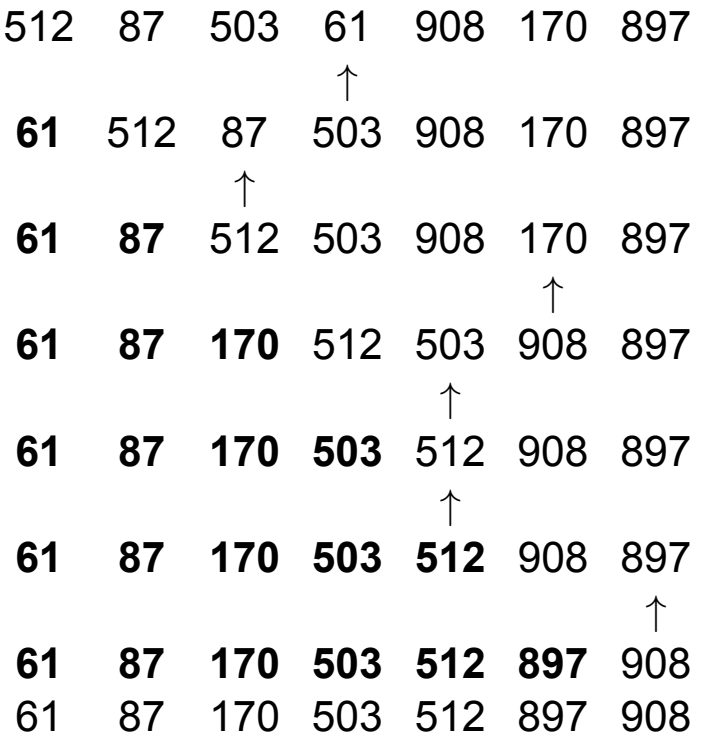
Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897
61 87 170 503 512 908 897
61 87 170 503 512 908 897

Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897
61 87 170 503 512 908 897
61 87 170 503 512 908 897
61 87 170 503 512 897 908

Beispiel


512 87 503 61 908 170 897
61 512 87 503 908 170 897
61 87 512 503 908 170 897
61 87 170 512 503 908 897
61 87 170 503 512 908 897
61 87 170 503 512 908 897
61 87 170 503 512 897 908
61 87 170 503 512 897 908

Sortieren durch Minimumauswahl

Algorithm 1: Sortieren durch Auswahl

Input: Array $A[1, n]$

begin

for $i = 1, \ldots, n-1$ **do**

$m \leftarrow i$

for $j = i+1, \ldots, n$ **do**

if $A[j] < A[m]$ **then** $m \leftarrow j$

füge $A[m]$ am Beginn des Restbereichs ein und verschiebe alle folgenden Elemente in die entstehende Lücke

Selection Sort

- Verschieben der Elemente ist in der Praxis zu aufwändig
  $\Rightarrow$ erstes Element mit Minimum vertauschen

Algorithm 2: SelectionSort

Input: Array $A[1, n]$

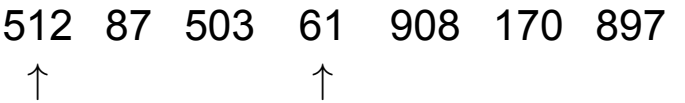
begin

```
for i = 1, ..., n − 1 do
    m ← i
    for j = i + 1, ..., n do
        if A[j] < A[m] then m ← j
    vertausche A[i] und A[m]
```

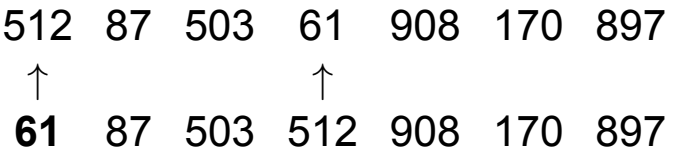
Beispiel Selection Sort

512 87 503 61 908 170 897

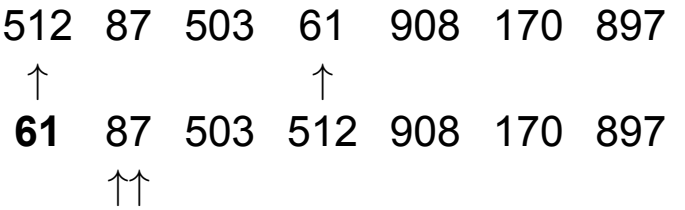
Beispiel Selection Sort



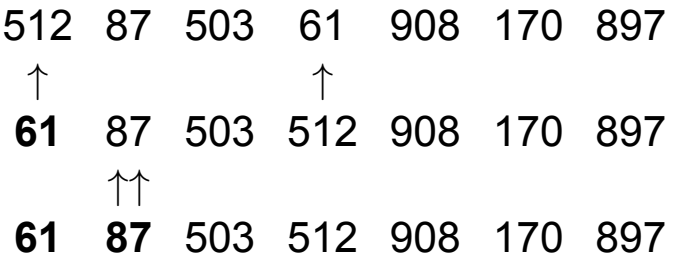
Beispiel Selection Sort



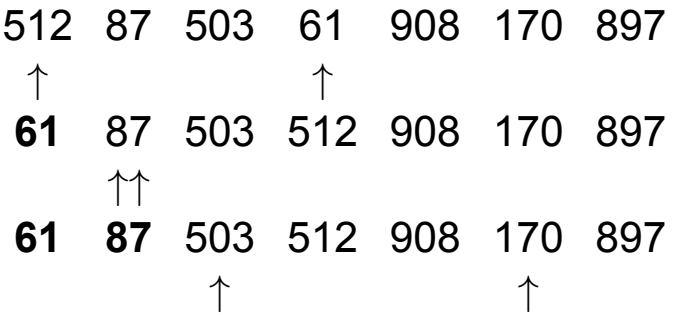
Beispiel Selection Sort



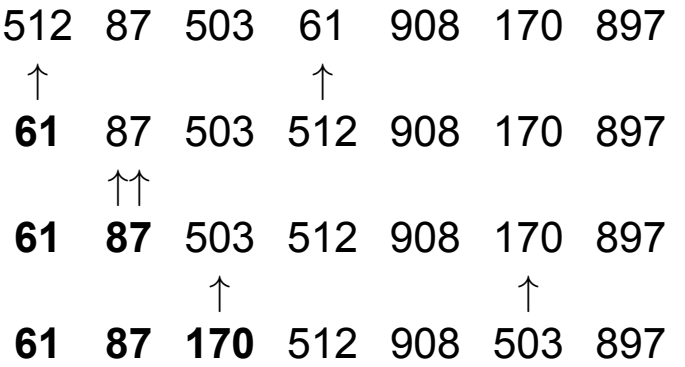
Beispiel Selection Sort



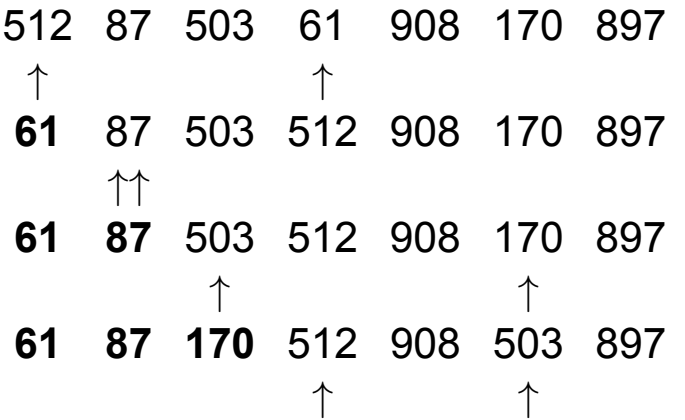
Beispiel Selection Sort



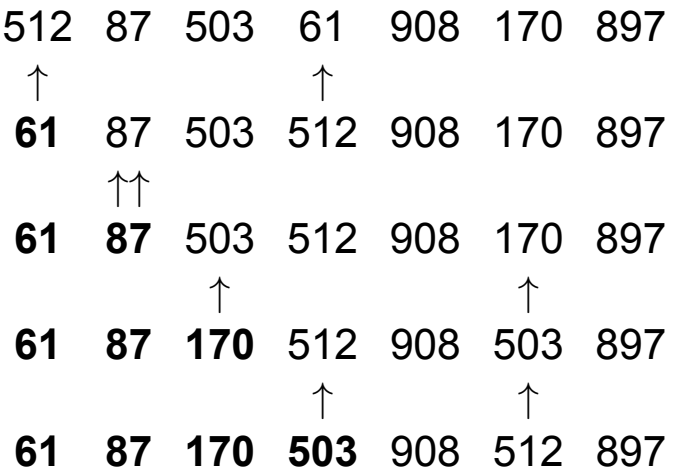
Beispiel Selection Sort



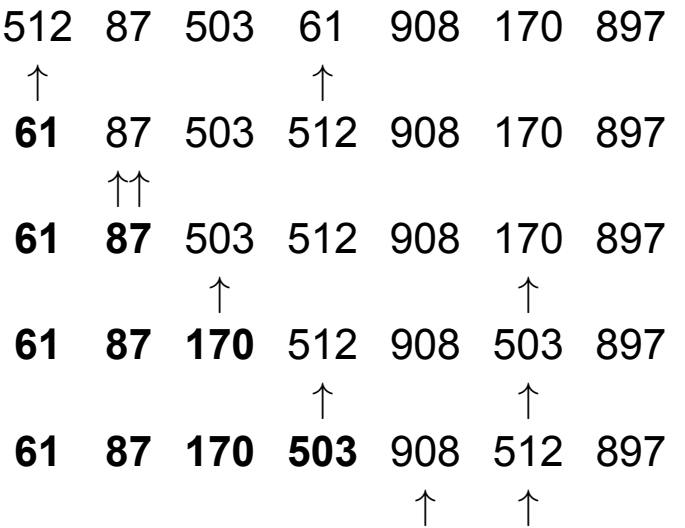
Beispiel Selection Sort



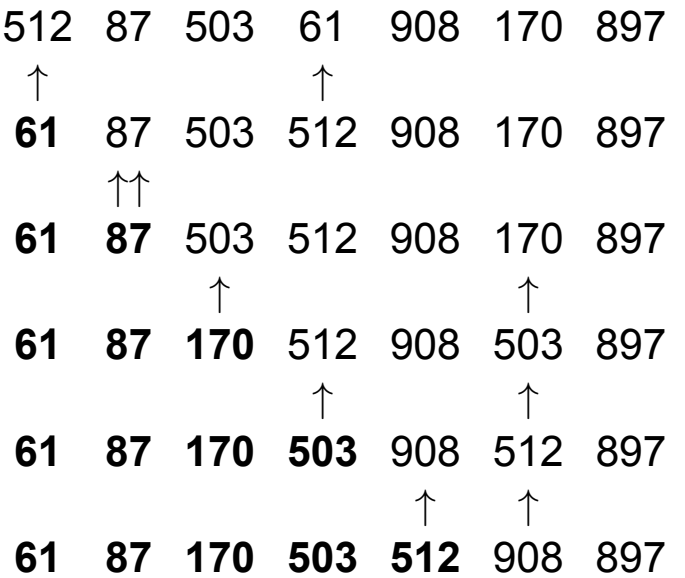
Beispiel Selection Sort



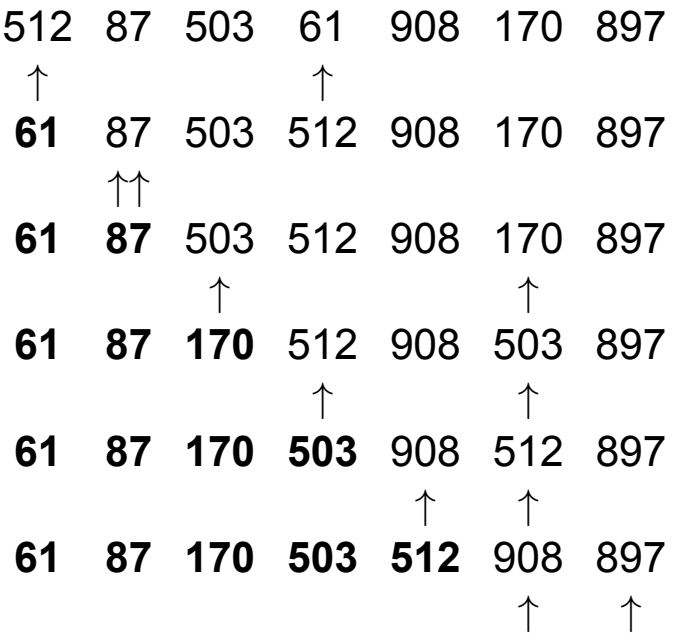
Beispiel Selection Sort



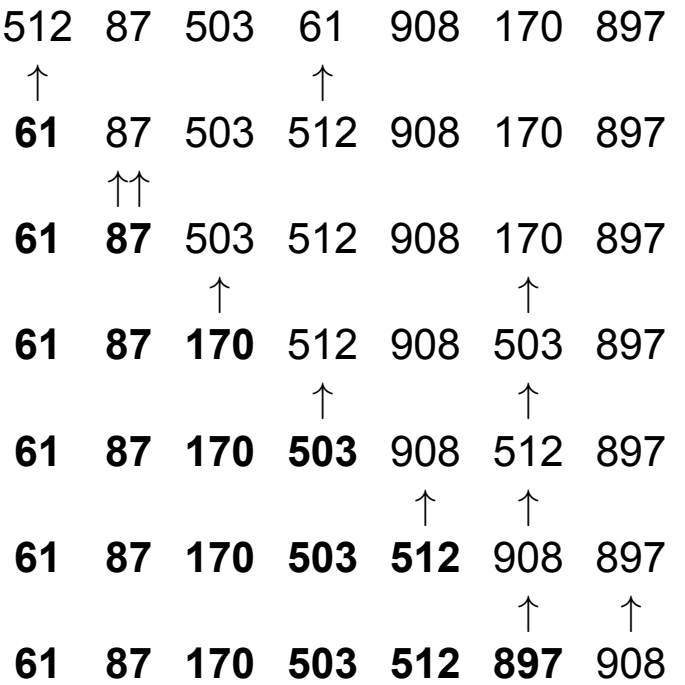
Beispiel Selection Sort



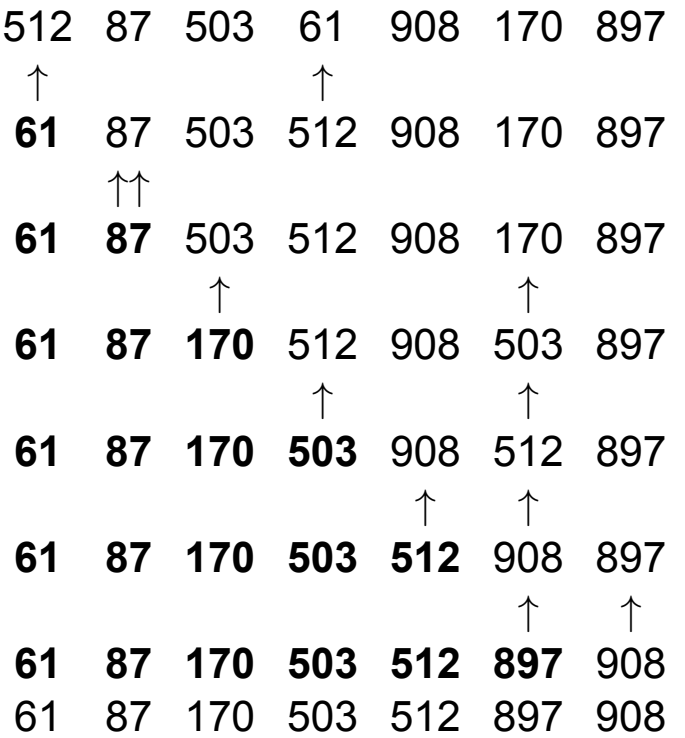
Beispiel Selection Sort



Beispiel Selection Sort



Beispiel Selection Sort



Aufwandabschätzung

Aufwandabschätzung

- Auf jeden Fall n-mal Suche nach dem Minimum
- Dazu muss der Restbereich komplett durchsucht werden, um das Minimum zu finden
  - vor erster Vertauschung: $n-1$ Vergleiche
  - vor zweiter Vertauschung: $n-2$ Vergleiche
  - vor dritter Vertauschung: $n-3$ Vergleiche

  $\Rightarrow \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{1}{2}n^2$

Aufwandabschätzung

- Auf jeden Fall n-mal Suche nach dem Minimum
- Dazu muss der Restbereich komplett durchsucht werden, um das Minimum zu finden
  - vor erster Vertauschung: $n-1$ Vergleiche
  - vor zweiter Vertauschung: $n-2$ Vergleiche
  - vor dritter Vertauschung: $n-3$ Vergleiche

  $\Rightarrow \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{1}{2}n^2$
- Sortieren einer doppelt so großen Reihung dauert viermal so lang
  - $\frac{1}{2}(2n)^2 = 4 \cdot \frac{1}{2}n^2$
- Sortieren von 10-mal so vielen Elementen dauert 100-mal so lang!

Überlegungen zur Korrektheit

Überlegungen zur Korrektheit

- Partielle Korrektheit
  - Zu Beginn ist das komplette Array der Restbereich.
  - Zu jedem Zeitpunk wird ein Minimum des Restbereichs an dessen Anfang gestellt.
  - Dieses Minimum ist grösser oder gleich aller Elemente, die weiter nach vorn gestellt wurden.
  - Die Reihenfolge der weiter vorn stehenden wird nicht mehr verändert
  - Damit ist jedes Element des Restbereichs grösser oder gleich aller, im sortierten Bereich.
    $\Rightarrow$ sortiert
- Terminiert der Algorithmus?
  - Ja, denn der Restbereich verkleinert sich in jedem Schritt, ist irgendwann leer und alles ist im sortierten Bereich.

Stabilität

Stabilität

Sortieren durch Auswahl

- Algorithmus ist stabil
  - existieren Elemente e und e' mit $s(e) = s(e')$, bleibt das Minimum das vorderste und wir auch in der sortierten Teilliste am weitesten vorn bleiben, dann neue Minima hinter den alten eingefügt werden
  - die geschieht auch, falls noch ein weiteres Element mit gleichem Wert existiert

Selection Sort

- Algorithmus ist nicht stabil!
  - durch das Vertauschen wird in der Restliste evtl. ein Element hinter ein anderes mit gleichem Wert gelegt

Einsortieren

- Beispiel Spielkarten sortieren

Einsortieren

- Beispiel Spielkarten sortieren
  - Karte aufnehmen und von links nach rechts mit den Karten auf der Hand vergleichen
  - neue Karte kommt links neben die erste Karte auf der Hand, die grösser als die neue Karte ist

Einsortieren

- Beispiel Spielkarten sortieren
  - Karte aufnehmen und von links nach rechts mit den Karten auf der Hand vergleichen
  - neue Karte kommt links neben die erste Karte auf der Hand, die grösser als die neue Karte ist
- Algorithmisches Vorgehen
  - Aufteilen in zwei Bereiche
    - R, den unsortierten Restbereich der Liste
    - L, den bereits sortierten Bereich der Liste
  - Beginn mit $L = \varnothing$ (auf der Hand)
  - Ende wenn $|R| = 0$ (auf dem Stapel)
  - erstes Element e aus R mit den Elementen in L vergleichen (von links nach rechts)
  - vor dem ersten Element einfügen, welches grösser e ist

Beispiel

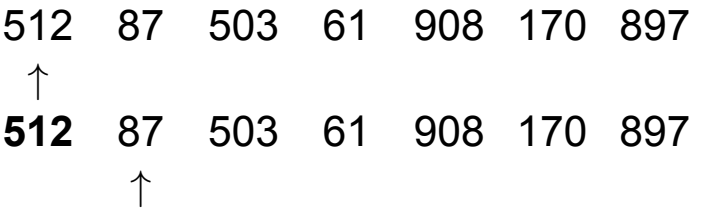
512 87 503 61 908 170 897

Beispiel

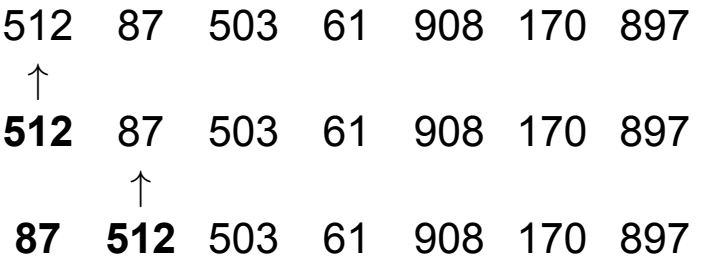
Beispiel

512	87	503	61	908	170	897
↑						
512	87	503	61	908	170	897

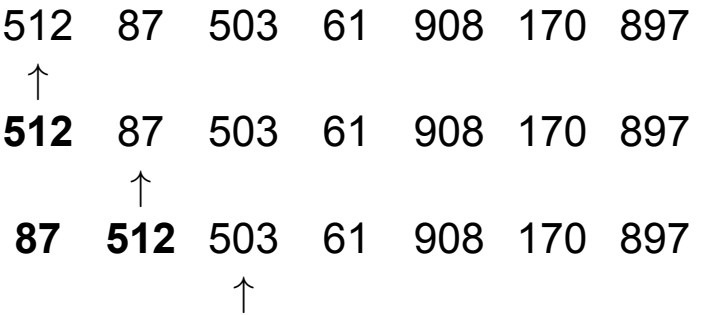
Beispiel



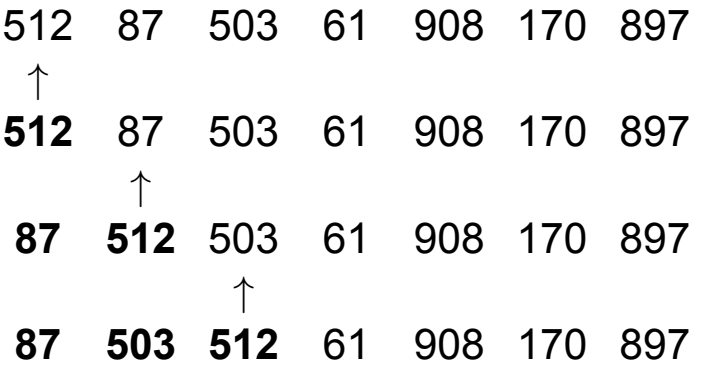
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897

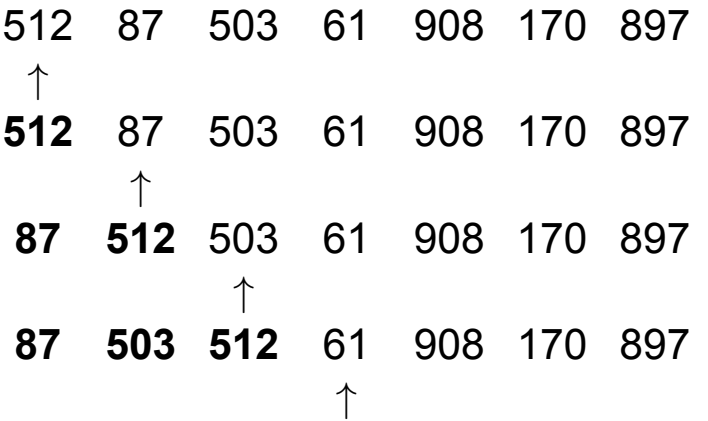
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897

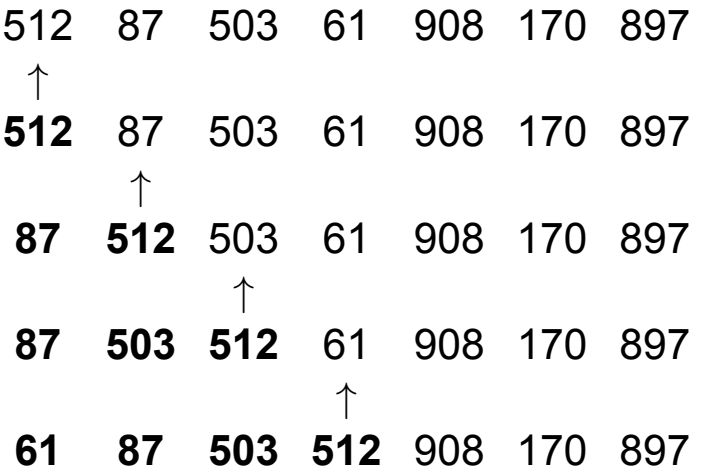
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897

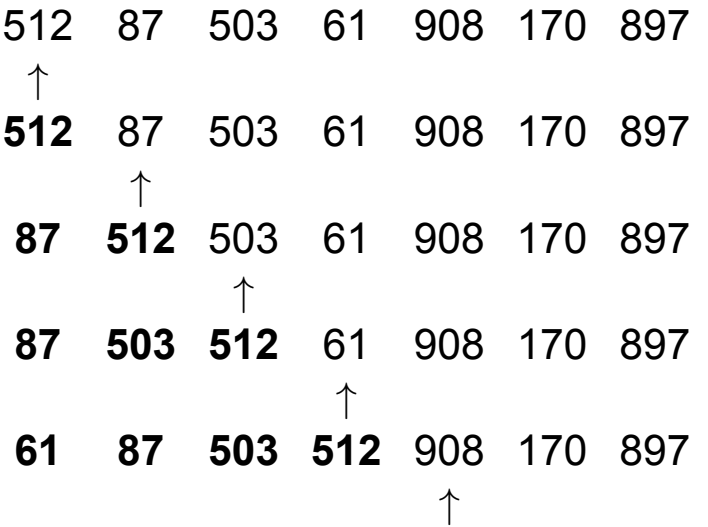
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897

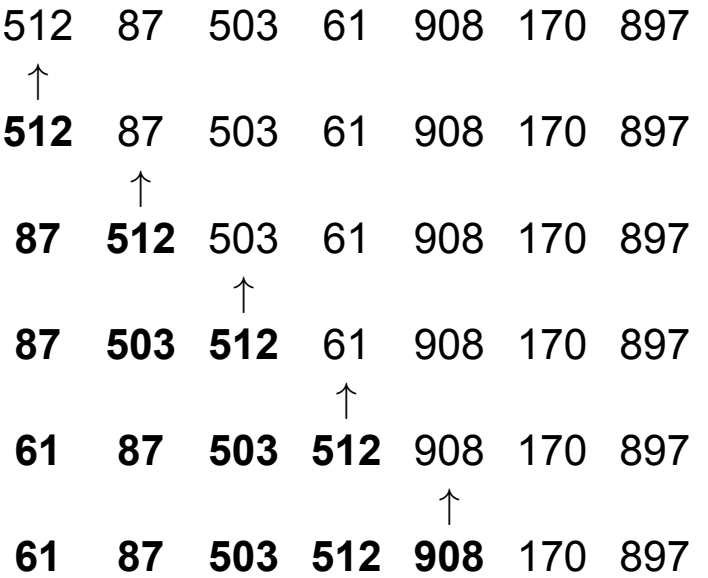
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897

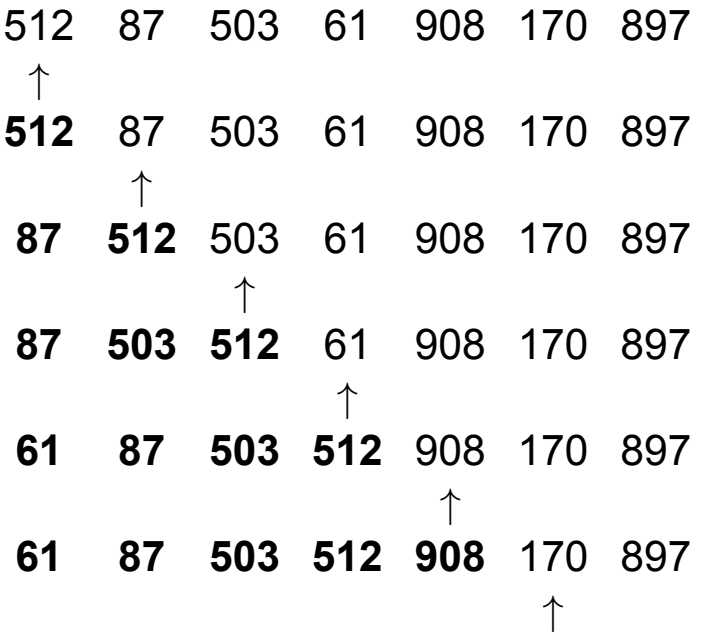
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897

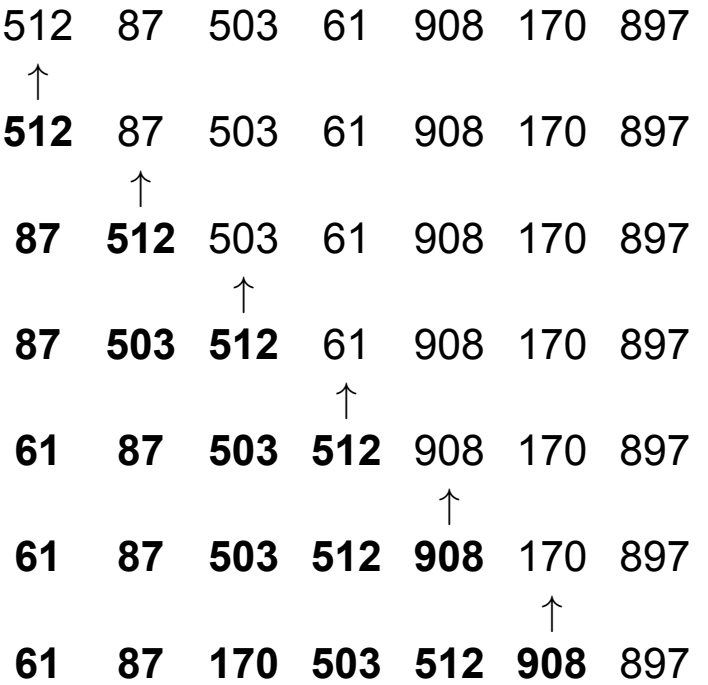
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897
61 87 503 512 908 170 897

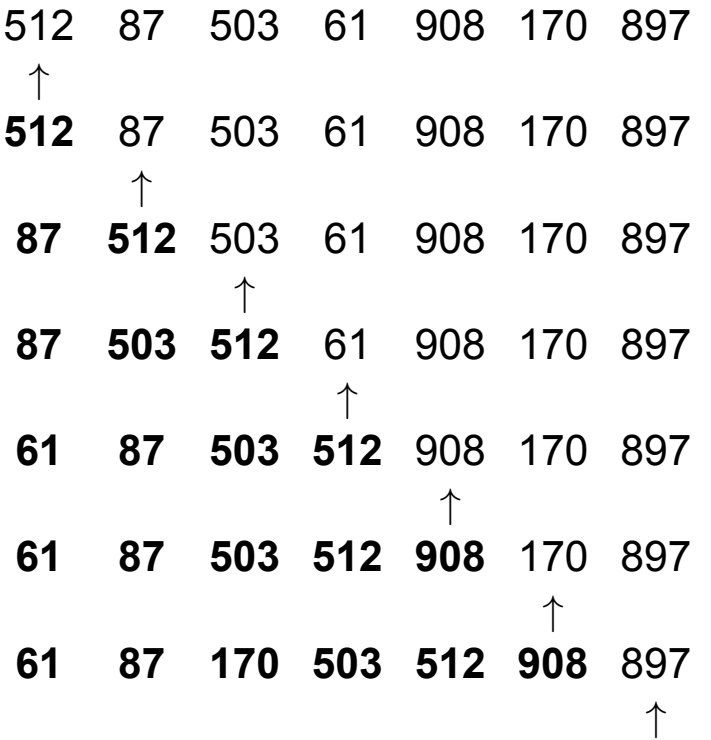
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897
61 87 503 512 908 170 897

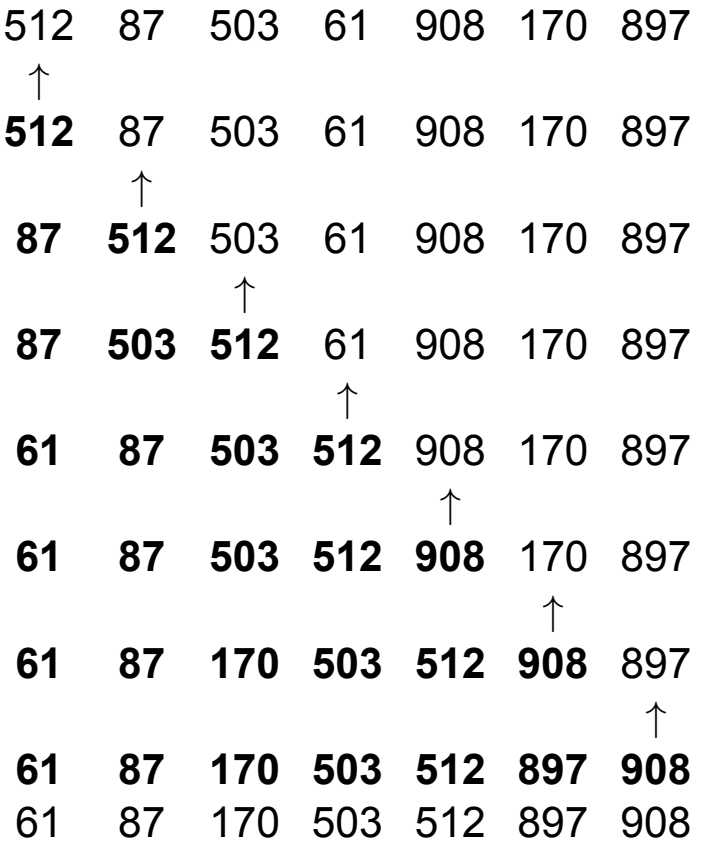
Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897
61 87 503 512 908 170 897
61 87 170 503 512 908 897

Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897
61 87 503 512 908 170 897
61 87 170 503 512 908 897

Beispiel


512 87 503 61 908 170 897
512 87 503 61 908 170 897
87 512 503 61 908 170 897
87 503 512 61 908 170 897
61 87 503 512 908 170 897
61 87 503 512 908 170 897
61 87 170 503 512 908 897
61 87 170 503 512 897 908
61 87 170 503 512 897 908

InsertionSort

- Restliste ist ganzes Liste
- Restliste ist nicht leer
  - Entnehme e als erstes Element der Restliste
  - Setze u_k auf das erste Element der sortierten Liste
  - Solange $u_k \neq null$ und $s(e) \geq s(u_k)$
    - $u_k = u_k.next$
  - Füge e vor u_k in die sortierten Liste ein

Hinweis: Einfügeoperation muss so implementiert werden, dass ein Einfügen ”vor”dem Null-Element ein Einfügen am Ende ist bzw. bzw. ggf. ein Einfügen in eine leere Liste.

Aufwandabschätzung

- Auf jeden Fall n-mal Suche nach der richtigen Stelle in der sortierten Liste
- Dazu muss man evtl. durch die komplette sortierte Liste
  - beim ersten Einsortieren: höchstens 0 Vergleich
  - beim zweiten Einsortieren: höchstens 1 Vergleich
  - beim dritten Einsortieren: höchstens 2 Vergleiche

  $$\Rightarrow \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \approx \frac{1}{2}n^2$$

Überlegungen zur Korrektheit

- Partielle Korrektheit
  - Vor dem Einfügen des ersten Elements, ist die sortierte Liste leer und damit soritiert
  - Jedes Element wird so eingefügt, dass die Liste hinterher auch noch sortiert ist.
- Terminiert der Algorithmus?
  - Der Restbereich wird bei jedem Schritt kleiner und ist irgendwann leer.

Stabilität

- Algorithmus ist stabil
- ein Element e' in der Ausgangsliste rechts eines e mit $s(e) = s(e')$
  wird beim Einügen in die sortierte Liste auch am vorher eingefügten e vorbeigeführt
- die geschieht auch, falls noch ein weiteres Element mit gleichem Wert existiert

Literatur

T. Ottmann und P. Widmayer.
Algorithmen und Datenstrukturen — Kapitel 2.
Spektrum Akademischer Verlag, 4. Ausgabe, 2002, ISBN 978-3-8274-1029-0.

Robert Sedgewick.
Algorithms in Java – Parts 1-4 – Kapitel 6–9.
Addison-Wesley Longman, Amsterdam, 3. Auflage, 2003, ISBN 0-210-36120-5.

H. P. Gumm und M. Sommer.
Einführung in die Informatik — Kapitel 4.2, 4.3.
Oldenburg Verlag, 7. Ausgabe, 2006, ISBN 978-3-486-58115-7.