Repetitionsfrage I

1. Mit Hilfe der Ackermann-Funktion kann man ein Problem illustrieren, das beim Einsatz von Rekursion beachtet werden sollte. Worauf beruht dieses Problem?
   Jeder Rekursionsaufruf benötigt Stack-Speicher (Memory), welcher zu einem Memory Overflow führen kann.

Kontrollfragen A

1. Illustrieren Sie das binäre Suchen mit einer Zeichnung.
   Die binäre Suche ist ein Algorithmus, der auf einem Feld sehr effizient ein gesuchtes Element findet bzw. eine zuverlässige Aussage über das Fehlen dieses Elementes liefert. Voraussetzung ist, dass die Elemente in dem Feld entsprechend einer totalen Ordnungsrelation angeordnet („sortiert“) sind. Der Algorithmus basiert auf einer einfachen Form des Schemas Teile und Herrsche. (Quelle: http://de.wikipedia.org/wiki/Bin%C3%A4re_Suche)

   Wir wollen die Zahl 13 suchen. Wir starten in der Mitte der 32 Zahlen. Bei 32 Elementen gibt es leider keine genaue Mitte, die Mitte würde zwischen den Elementen 16 und 17 liegen. Wir beginnen daher links von der Mitte mit der Zahl 16. Dann vergleichen wir die Suchzahl mit der mittleren Zahl des Suchbereichs, also der 16. Da die 13 kleiner als die 16 ist, setzen wir unsere Suche in dem Bereich links von der 16 fort.

   (Quelle: http://www.u-helmich.de/inf/BlueJ/kurs121/seite12/seite12.html)

2. Mit Hilfe von virtuellem Arbeitsspeicher lassen sich praktisch beliebig grosse Datenmengen intern sortieren. Was heisst virtueller Arbeitsspeicher?
   Um den physischen Arbeitsspeicher zu erweitern, können moderne Betriebssysteme zusätzlichen virtuellen Arbeitsspeicher auf Massenspeichern allozieren (platzieren, zuteilen).

3. Welche Rolle kommt dem Sortierschlüssel zu?
   Sie sind das Kriterium nachdem wir sortieren.

4. Illustrieren Sie an einem einfachen Beispiel instabiles Sortieren.

	8	*3_1*	*3_2*	*1*
Folge: 2	*3_2*	*1*	*8*	*3_1*
Folge: 1	*1*	*3_2*	*3_1*	*8*

5. Welches sind die massgebenden Operationen beim internen Sortieren?
   - *Vergleiche: <, <=, ==, >=, >, !=*
   - *Zuweisungen: =*

6. Was für einen Aufwand besitzen einfache/direkte Sortier-Algorithmen? Wie sieht es bei höheren Sortieralgorithmen aus? Machen Sie einen rudimentären Vergleich für n = 1'000.
   n = 1000 => 10^3
   $O(n^2)$ => $O(10^{3*2}) = O(10^6)$

   $\log_2(1000) \approx 10$
   $O(n * \log(n))$ => $1000 * \log_2(1000) => \approx O(10^4)$

Kontrollfragen B

1. Welche direkten Sortieralgorithmen arbeiten stabil?
   Insertion Sort / direktes Einfügen
   Bubble Sort / direktes Austauschen

2. Alle direkten Sortierverfahren sind grundsätzlich von der Ordnung $O(n^2)$. Von welchem Verfahren erwarten Sie im Vergleich aber die beste Performance? Wieso?
   Insertionsort, weil das zu sortierende Element direkt eingefügt wird (und nicht „Sesselrücken“).

3. Bei welchem Sortieralgorithmus ist der Sortieraufwand unabhängig von der Ausgangslage (vgl. bereits sortiert, zufällig, ...)?
   selectionsort

4. Wie lässt sich folgende Summe einfach berechnen: 1 + 2 + 3 + 4 + ... + (n-1) + n = ? (Entsprechende Formel ist hilfreich, wenn man direkte Sortierverfahren analysiert.)
   Gausssche Summenformel:
   $$1+2+3+4+\ldots+n=\sum_{k=1}^{n}k=\frac{n(n+1)}{2}.$$

5. Wie lässt sich das direkte Einfügen allenfalls optimieren?
   Zuerst suchen wo es eingefügt werden muss (z.B. mittels Binarysearch)

Kontrollfragen C

1. Arbeitet Shellsort stabil oder instabil?
   Instabil !

2. Wieso muss eine Folge immer mit 1 enden?
   Sonst werden benachbarte Elemente nie ausgetauscht.

3. Ist die Folge im Beispiel gut gewählt?
   Nein

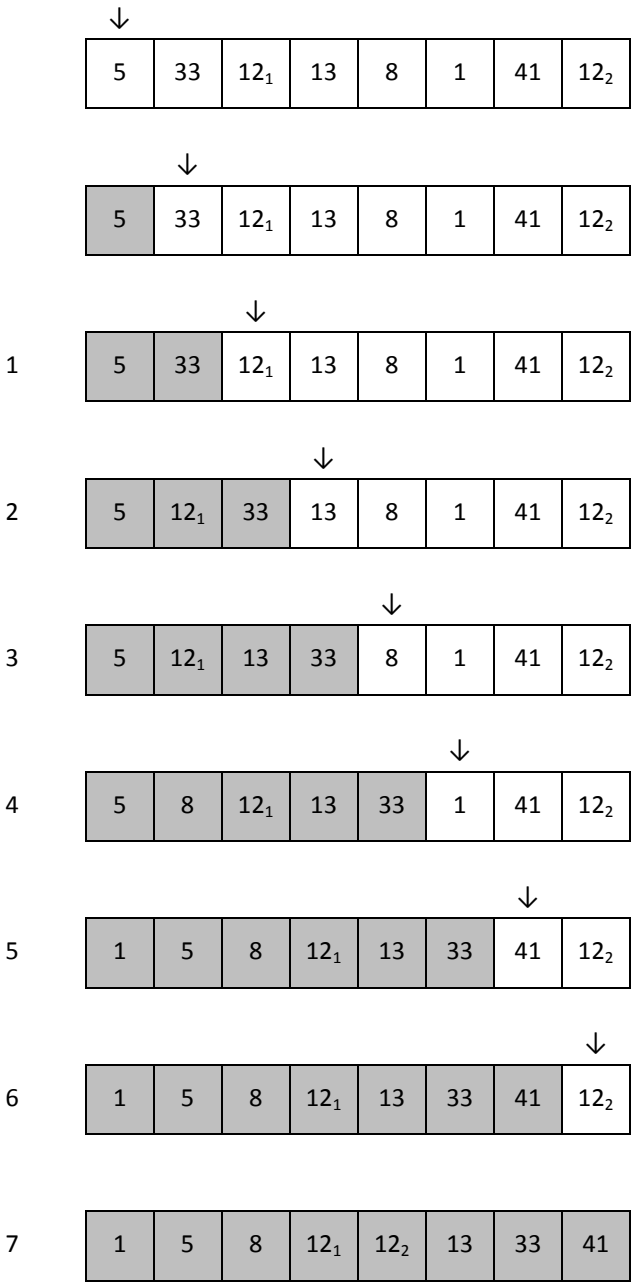
Direktes Einfügen / Insertion Sort

Insertionsort (englisch insertion ‚Einfügen' und englisch sort ‚sortieren') ist ein einfaches stabiles Sortierverfahren. Es ist einfach zu implementieren, effizient bei (ziemlich) kleinen Eingabemengen, effizient bei Eingabemengen, die schon vorsortiert sind, stabil (d. h. die Reihenfolge von Elementen mit gleichem Schlüsselwert bleibt unverändert) und minimal im Speicherverbrauch, da der Algorithmus in-place arbeitet.

Insertionsort entnimmt der unsortierten Eingabefolge ein beliebiges Element und fügt es an richtiger Stelle in die (anfangs leere) Ausgabefolge ein. Geht man hierbei in der Reihenfolge der ursprünglichen Folge vor, so ist das Verfahren stabil. Wird auf einem Array gearbeitet, so müssen die Elemente hinter dem neu eingefügten Element verschoben werden. Dies ist die eigentlich aufwändige Operation von Insertionsort, da das Finden der richtigen Einfügeposition über eine binäre Suche vergleichsweise effizient erfolgen kann.

Grau: sortiert

Weiss: unsortiert

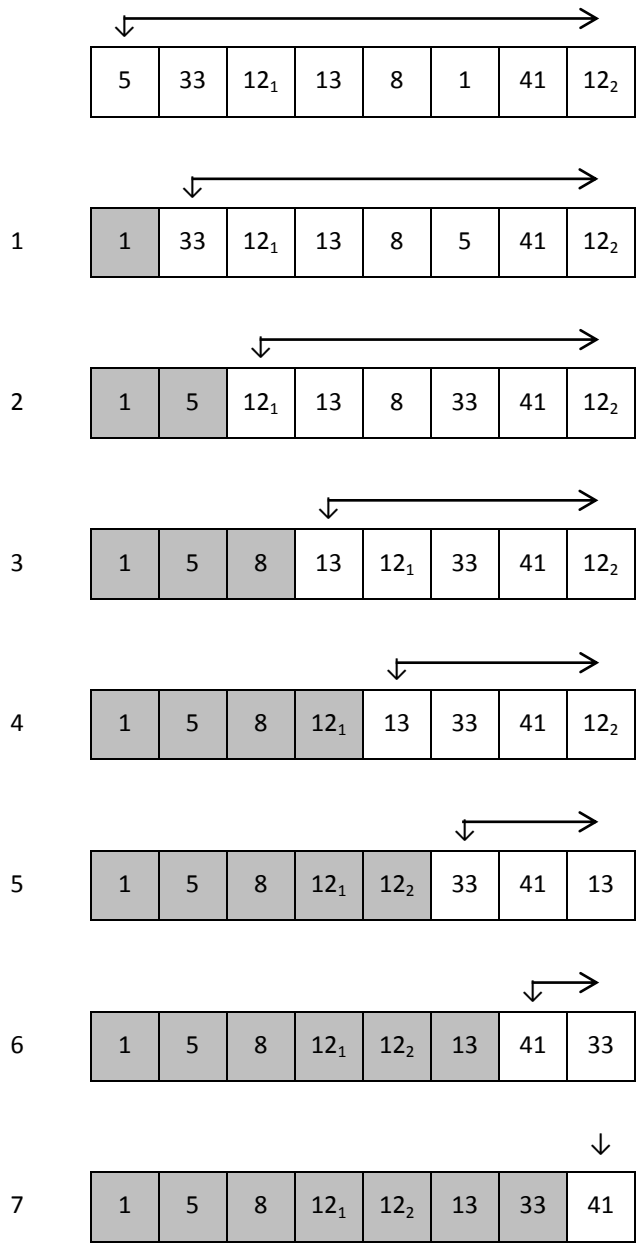


Direktes Auswählen / Selection Sort

Finde zuerst das kleinste Element im Datenarray und tausche dieses gegen das an der ersten Stelle befindliche Element aus. Anschließend finde das zweitkleinste Element, tausche es gegen das zweite aus...

Grau: sortiert

Weiss: unsortiert



Direktes Austauschen / Bubble Sort

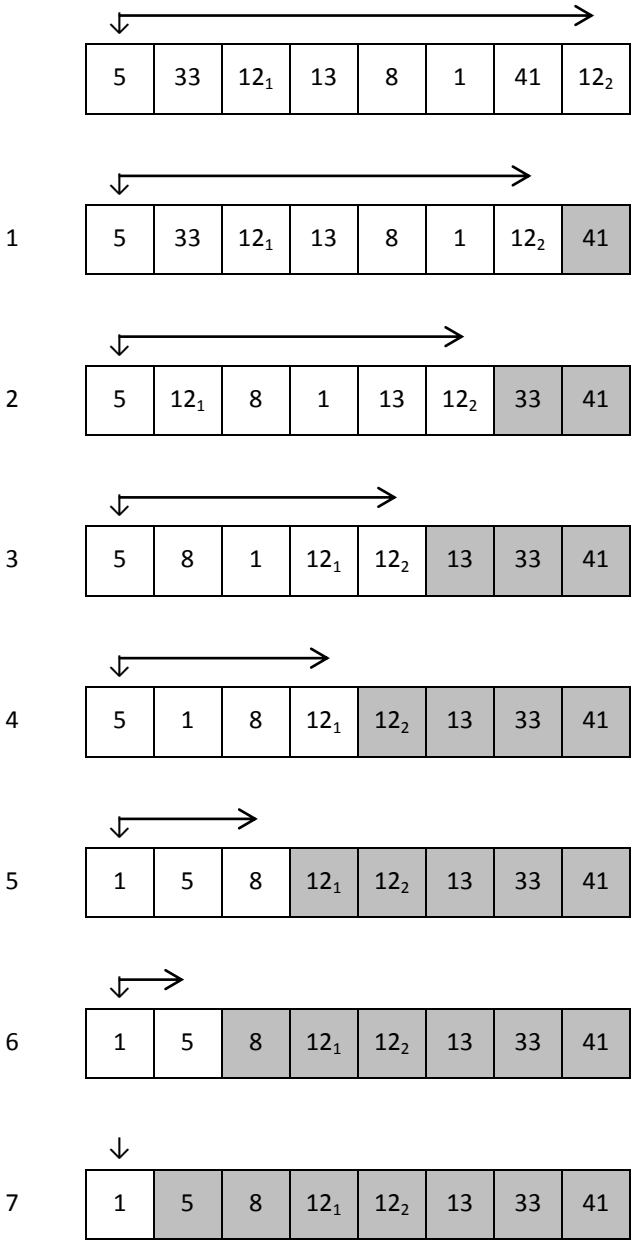
Das Sortieren durch Aufsteigen (englisch Bubble sort, "Blasensortierung") bezeichnet einen einfachen, stabilen Sortieralgorithmus, der eine Reihe zufällig angeordneter Elemente (etwa Zahlen) der Größe nach ordnet.

Bubblesort gehört zur Klasse der In-place-Verfahren, was bedeutet, dass der Algorithmus zum Sortieren keinen zusätzlichen Arbeitsspeicher außer den lokalen Laufvariablen der Prozedur benötigt. Dadurch, dass grundsätzlich nur aneinandergrenzende Elemente miteinander vertauscht werden, eignet sich dieses Verfahren auch zum Sortieren von Listen mit unterschiedlich großen Elementen.

Wegen der einerseits mangelhaften Effizienz und anderseits breiten Verfügbarkeit deutlich besserer Alternativen wird Bubblesort meistens nur als Beispiel für einen schlechten oder naiven Sortieralgorithmus verwendet.

Grau: sortiert

Weiss: unsortiert



Shellsort

Shellsort ist ein von Donald L. Shell im Jahre 1959 entwickeltes Sortierverfahren, das auf dem Sortierverfahren des direkten Einfügens (Insertionsort) basiert.

Shellsort bedient sich prinzipiell des Insertionsorts. Es versucht den Nachteil auszugleichen, dass hier Elemente in der Sequenz oft über weite Strecken verschoben werden müssen. Dies macht Insertionsort ineffizient. Shellsort verfolgt den Ansatz, dass die Sequenz z.B. erst 4-sortiert wird, dann 2-sortiert, und zuletzt mit normalem Insertionsort sozusagen 1-sortiert.

Grau: sortiert

Weiss: unsortiert