Damit haben wir die Voraussetzungen erfüllt, den WAIT-Befehl zu verstehen.

WAIT: Ein Aschenputtel in Basic

Wie es schon der Name sagt (wait = warten), hält WAIT ein Programm so lange an, bis in einer spezifizierten Speicherstelle ein bestimmtes Bit-Muster aufgetreten ist. Dieses Bit-Muster gibt man durch zwei Masken vor, von denen die erste (obligatorische) eine AND-, die zweite eine EOR-Maske ist. So sieht die Syntax des WAIT-Befehls aus:
WAIT Speicherstelle, AND-Maske, EOR-Maske

Der effektive Einsatz von WAIT erfordert eine recht gute Kenntnis der Speicherbelegung unseres Commodore 64, was vermutlich einer der Gründe für die seltene Benutzung dieses Befehls ist.

Wie eben schon kurz gesagt, kann WAIT mit einem oder mit zwei Argumenten betrieben werden. Sehen wir uns zunächst mal an, was mit nur einem Argument möglich ist.

Hier wird der in der adressierten Speicherstelle enthaltene Wert mit der AND-Maske verknüpft. Ergibt sich dabei ein Wert, der ungleich Null ist, dann fährt das Programm fort. Im anderen Fall wartet es, bis der Speicherinhalt dieser Anforderung entspricht (also bis irgendwann einmal die AND-Verknüpfung des Speicherinhaltes mit der AND-Maske keine Null mehr ergibt) oder aber bis man, des Wartens müde, das Programm durch RUN/STOP-RESTORE unterbricht. Sehen wir uns dazu ein praktisches Beispiel an. Vielfach werden Programme bis zu einem Tastendruck mittels:
100 GET A$:IFA$=""THEN 100
angehalten. Das hat den Nachteil, daß man dafür eine Basic-Zeile vergeben muß, was beispielsweise beim VC 20 in der Grundversion eine unverzeihliche Sünde sein kann. Statt dessen bedienen wir uns einer Speicherstelle in der Zeropage, die die Anzahl der gültigen Zeichen im Tastaturpuffer angibt: 198. Schreiben wir:
POKE 198,0:WAIT198,1
dann wartet das Programm, bis durch einen Tastendruck ein Zeichen im Tastaturpuffer aufgetreten ist. Diese Befehls-Sequenz kann nun auch mitten zwischen anderen Befehlen stehen. Was passiert dabei? Folgendes:

Inhalt von 198 nach einem Tastendruck:	0000 0001
AND-Maske:	0000 0001
Ergebnis der AND-Operation:	0000 0001

Das Ergebnis ist ungleich Null, das Programm läuft weiter.

Wenn wir WAIT198,2 eingeben, wartet das Programm auf zwei Tastendrucke. Was geschieht bei WAIT 198,3? Probieren Sie es aus: Schon nach dem ersten Tastendruck läuft das Programm weiter. Sehen wir uns mal an, weshalb. 3 sieht im Binärformat so aus: 0000 0011. Wenn nun nur ein Tastendruck in 198 registriert ist, ergibt die AND-Verknüpfung auch schon ein Ergebnis, das ungleich Null ist:

0000	0001	Eine Taste
0000	0011	Maske AND
0000	0001	

Man kann also nur auf 1, 2, 4, 8 etc. Tastendrucke warten.

Ein anderes nützliches Beispiel ist die Speicherstelle 653, die die Kombinationstasten (SHIFT, Commodore und CTRL) überwacht. WAIT653,1 wartet auf die SHIFT-Taste, WAIT653,2 auf die Commodore-Taste und WAIT653,4 auf die CTRL-Taste.

Damit wollen wir für diesmal aufhören zu »logeln«. In der nächsten Folge geht es dann um den WAIT-Befehl mit zwei Argumenten. Außerdem werden wir feststellen, ob ein Computer auch Schlußfolgerungen ziehen kann.

(Heimo Ponnath/gk)

Sortieren mit dem Computer (Teil 4)

Quicksort verdient seinen Namen zu Recht. Aber er kann auch zu einem ausgesprochenen Langweiler werden. Sogar Bubblesort kann schneller sein als alle anderen Sortierroutinen. Woran das liegt, erfahren Sie im folgenden Artikel.

Der Quicksort-Algorithmus wurde bereits 1962 von R. Hoare entwickelt und ist trotzdem bis heute die schnellste Methode zum Sortieren großer, zufallsbesetzter Felder geblieben. Der Deutlichkeit halber betone ich noch einmal zufallsbesetzt, denn nur bei dieser Art der Verteilung der Feldelemente kann Quicksort wirklich effektiv arbeiten. Haben wir ein Array, das zum Beispiel absteigend sortiert ist (9, 8, 7, ..., 1) und sollen hier nur wenige Elemente neu einsortiert werden, so wird Quicksort im wahrsten Sinne des Wortes zum »Slowsort« und benötigt eine ewige Zeit zur Bearbeitung des Feldes.

Andererseits lohnt sich der Einsatz von Quicksort auch dann nicht, wenn es darum geht, beispielsweise eine Kartei zu führen, bei der laufend neue Elemente in ein schon vorsortiertes Feld eingefügt werden sollen (zum Beispiel der Buchstabe D in das Feld A, B, C, F, H, M, N, O, ...). Hier eignet sich unser (normalerweise langsamster) Bubblesort-2-Algorithmus sehr gut; in diesem Fall findet nämlich nur ein einziger Durchlauf statt, bevor der Algorithmus beendet wird (wie Sie wissen, verwendet Bubblesort 2 eine zusätzliche Variable, die anzeigt, ob noch weitere Sortierdurchläufe notwendig sind).

Bei der Effektivität unserer Sortierprogramme sollen uns diese Spezialanwendungen jedoch nicht weiter interessieren. Hier geht es nur um das Problem, ein völlig »durcheinandergeratenes« Feld wieder in Ordnung zu bringen, und das bitte möglichst schnell.

Wie funktioniert Quicksort?

So wollen wir uns nun einmal den Quicksort-Algorithmus etwas genauer betrachten. Er ist nicht im mindesten mit den bisherigen Methoden zu vergleichen, und wunderbarerweise zählt er trotz seiner Leistungsfähigkeit zu den etwas leichter verständlichen Algorithmen.

Das Prinzip von Quicksort ist folgendes:

Zuallererst wird aus dem gesamten Variablenfeld ein (beliebiges) Element herausgegriffen und »beiseite gelegt«. Dieses Element dient nun als Vergleichswert für das gesamte übrige Array. Jetzt werden alle Elemente, die kleiner als das Vergleichselement sind, links (also auf niedrigere Positionen) und alle Elemente, die größer sind, rechts (also auf höhere Positionen) vom Vergleichselement abgespeichert. Wir erhalten so ein Variablenfeld, bei dem alle kleineren Elemente oberhalb und alle größeren Elemente unterhalb des Vergleichselements stehen. Bild 1 zeigt noch einmal genau, was gemeint ist.

Nachdem diese »Gesamtsortierung« vollzogen wurde, haben wir ein schon sehr grob sortiertes Feld vorliegen. Nun teilen wir die beiden neuen Teilfelder (oberhalb und unterhalb des Vergleichselements) wiederum durch jeweils ein neues Vergleichselement auf, wobei diese beiden Teilfelder auf die oben beschriebene Art erneut sortiert werden. Als Ergebnis haben wir dann vier teilsortierte Feldabschnitte vorliegen, die wiederum, jedes für sich, geteilt und sortiert werden.

Setzen wir diese Teilungen immer weiter fort, so werden wir irgendwann einmal die Teilfeldlänge 1 erreichen, womit unser Gesamtfeld fertig sortiert wäre.

Wie Sie sicherlich bemerken, ist die Effektivität dieses Algorithmus natürlich stark von der Wahl des jeweiligen Vergleichselements abhängig. Optimal wäre jedesmal ein Vergleichselement, das etwa das Mittel der zugehörigen Teilliste ausmacht und diese so in zwei gleich große Abschnitte trennt.

Die Suche nach einem solchen optimalen Vergleichselement würde sich jedoch so aufwendig gestalten, daß die Geschwindigkeit von Quicksort stark darunter leiden müßte. Aus diesem Grund geht man bei der Wahl des betreffenden Elements einen anderen Weg. Es wird einfach jedesmal das Element gewählt, das genau in der Mitte der Teilliste steht. Auf diese Weise erhält man bei zufallsbesetzten Feldern zwar auch sehr ungünstige Werte; dieser Mangel wird jedoch durch eine ebensogroße Zahl von extrem günstigen Werten ausgeglichen.

03	05	23	01	17	88	47	33	21	14
03	05	14	01	17	88	47	33	21	23
03	05	01	14	17	47	33	21	23	88
01	03	05	14	17	21	23	33	47	88
01	03	05	14	17	21	23	33	47	88

Bild 1. Sortierbeispiel von Quicksort

Im Geschwindigkeitstest zeigt dieser Quicksort-Algorithmus dann auch seine verblüffenden Eigenschaften.

Bevor wir jedoch auf einen Vergleich sämtlicher bisher besprochener Sortiermethoden eingehen, noch etwas zur Programmierung von Quicksort.

Was ist rekursives Programmieren?

Während des Programmablaufs kommt ein Unterprogramm immer wieder in der gleichen Form vor und zwar das Sortieren eines Teilfeldes anhand des Vergleichselements. Dieses Unterprogramm stellt nun auch die neuen Teillisten fest und müßte bei einem sofortigen Rücksprung mit »RETURN« sämtliche neuen Parameter zwischenspeichern, um dann erneut aufgerufen zu werden ...

Diese umständliche Programmiertechnik, die für einen linearen Programmablauf sorgt, wurde bei Quicksort nicht verwendet. Hier werden vielmehr die neu festgestellten Teillisten sofort wieder bearbeitet, um danach ebenfalls wieder die nächsten Teillisten herzustellen.

Wie funktioniert nun eine solche Programmiertechnik?

Man geht davon aus, daß der eigentliche Sortier- und Teilalgorithmus in einer Unterroutine untergebracht ist, wobei diese mit GOSUB aufgerufen wird. Wurde nun die erste Teilung des Variablenfeldes durchgeführt, so werden sofort die Werte für die nächste Halbierung dieser beiden Teilfelder bereitgestellt. Danach ruft sich die Unterroutine sofort wieder **selbst** auf, um auch die neue Sortierung und Teilung ablaufen zu lassen. Dabei entstehen wieder zwei neue Teilfelder, die wiederum sofort bearbeitet werden, etc. ...

Bild 2 zeigt anschaulich, was gemeint ist.

Eine solche Programmiertechnik, bei der sich Programmteile selbst aufrufen, nennt man rekursiv!

Nun, könnten Sie jetzt einwenden, dieses Programm verschachtelt sich doch immer weiter, ohne daß ein Ende abzusehen ist. Wie findet der Computer denn aus diesem »Irrgarten« wieder heraus?

Die Sache ist relativ einfach. Irgendwann wird der Computer bei der Teillistenlänge 1 angekommen sein. Hat er das festgestellt, so erfolgt kein weiterer Selbstaufruf mehr, sondern es wird ein »RETURN« durchgeführt. Jetzt ist aber unser Unterprogramm so ausgelegt, daß es jeweils **alle** Teillisten der gleichen Länge nacheinander bearbeitet. Erfolgt also das erste RETURN, so bearbeitet der Computer die zweite Teilliste der Länge 2, bis er auch hier wieder teilt und bei 1 ankommt ...

Dieses Bearbeiten der zwei letzten Teillisten setzt sich so lange fort, bis der Computer bei der letzten Teilliste dieser Ebene angekommen ist. Hier erfolgt wiederum ein RETURN, so daß nun auf die drittletzte Ebene mit der Größe 4 zurückgesprungen wird. Auch dort werden sämtliche Elemente bearbeitet, bis wir irgendwann wieder bei einer Teilliste der Länge A (gesamtes Feld) angekommen sind, worauf der Quicksortalgorithmus verlassen wird.

Dieses Prinzip ist nicht so ohne weiteres verständlich. Am besten sehen Sie sich noch einmal Bild 2 an; dort ist der ganze rekursive Algorithmus grafisch dargestellt.

An der Geschwindigkeit von Quicksort wird deutlich, wie effektiv rekursive Programmiertechnik sein kann.

Probleme der Rekursion in Basic

Bei unserem Quicksort in Basic stellt sich jedoch noch ein zusätzliches Problem: Basic ist keine strukturierte Sprache!

Was das bedeutet, sei am Beispiel von Pascal kurz erläutert.

Bei einer strukturierten Sprache werden Unterprogramme als eigene Einheiten mit eigenen Variablen betrachtet. Es kann also vorkommen, daß eine Unterroutine die gleichen Variablen wie das übergeordnete Programm enthält, wobei diesen jedoch andere Werte zugeordnet sind.

Hat also beispielsweise das Hauptprogramm in Pascal eine Variable X mit dem Wert 3 belegt und springt nun ein Unterprogramm an, in dem diese Variable ebenfalls verwendet wird, so wird der Inhalt von X auf einen Software-Stack gerettet und erst anschließend das Unterprogramm aufgerufen. Bei der Rückkehr aus dieser Unterroutine holt der Computer den Wert von X wieder vom Stack und übergibt ihn dem Hauptprogramm.

Bei einer rekursiven Programmiertechnik bleiben also alle Parameter, die von einer Routine erarbeitet wurden, erhalten und können später, gemäß der Reihenfolge, weiter verwendet werden.

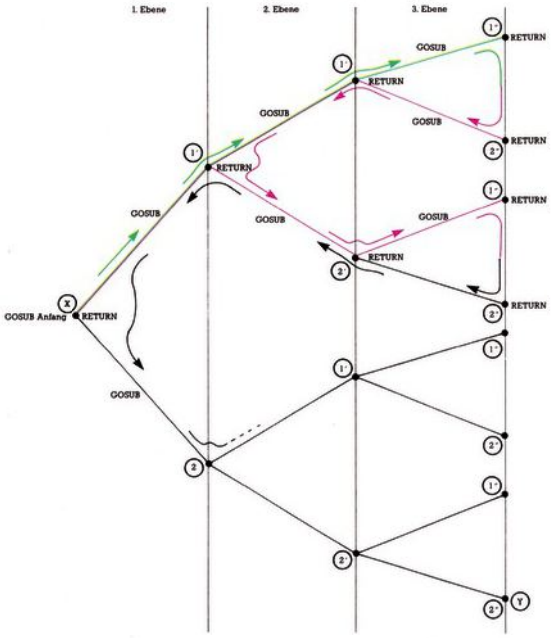
Nicht so in Basic!

Hier gibt es nur jeweils **eine** Variable gleichen Namens, deren Wert sowohl vom Hauptprogramm als auch von Unterroutinen gleichermaßen beeinflußt werden kann. Haben Sie also die Variable X mit dem Wert 3, so erfolgt bei einem Unterprogramm, in dem es heißt: X=5, ein Verlust der 3, der auch bei der Rückkehr ins Hauptprogramm nicht aufgehoben wird.

Aus diesem Grund können Sie im Basic-Quicksort noch ein weiteres Variablenfeld erkennen, in dem die wichtigen Parameter der Teilfelder abgespeichert werden, damit sie bei der Rückkehr von einer Ebene auf eine höhere wieder zur Verfügung stehen.

Wenn Sie das Prinzip der rekursiven Programmiertechnik noch nicht ganz verstanden haben, sollten Sie nicht die Mühe scheuen, noch einmal von vorne

Bild 2. Grafische Darstellung der rekursiven Programmierung bei Quicksort. Der besseren Übersicht halber sind nur drei Verschachtelungsebenen dargestellt.



Erklärung zu Bild 2:

Das rekursive Unterprogramm arbeitet immer innerhalb **eines** Aufrufes **zwei** Teilfelder ab, indem es zwischen beiden die Elemente vertauscht (Sortierung anhand des Vergleichselements).

Die rekursive Technik funktioniert nun wie ein Baum, bei dem nacheinander alle Äste abgefahren werden.

Zuerst wird das Gesamtfeld in zwei Teilfelder (entsprechend des Vergleichselements) aufsortiert (1. Ebene). Von diesen zwei Teilfeldern wird nun zuerst das erste Teilfeld wiederum aufgeteilt (grüner Pfeil X → 1). Da noch nicht Länge 1 erreicht wurde, wird das neue Teilfeld wiederum aufgeteilt (grüner Pfeil 1' → 1''). Jetzt haben die Teilfelder die Länge 1 und sind vollständig sortiert, weshalb nun auf eine höhere Ebene (RETURN bei 1'' und 2'') zurückgekehrt wird (roter Pfeil 2'' → 1' → 1).

Jetzt wird (wiederum mit GOSUB) das zweite Teilfeld der 2. Ebene (2') ebenfalls vollständig sortiert.....

Diese Vorgänge wiederholen sich, bis beim Punkt Y mit drei RETURNs wieder ganz zurückgesprungen wird, da das Unterprogramm auf 1. Ebene vollständig abgearbeitet wurde.

mit dem Lesen dieses Artikels zu beginnen. Diese Programmiermethode eignet sich nämlich zu allen algorithmischen Lösungen von Problemen mit ähnlicher Struktur, und Sie werden anhand von Quicksort erkennen können, wie effektiv und leistungsstark ein solches Programm wird.

Wie leistungsstark unsere sämtlichen Sortierprogramme sind, das soll im folgenden dargestellt werden, wobei Sie auch für Quicksort wieder einen Programmablaufplan vorfinden, der das Umschreiben auf andere Programmiersprachen erleichtern soll (Bild 3).

Nun aber zu unserem Vergleichstest.

Für diesen Test habe ich sämtliche Sortierprogramme soweit optimiert, indem ich alle REMs und alle zusätzlichen Leerzeichen entfernt habe. Außerdem wurde die dauernde Zwischenausgabe der Daten auf Bildschirm oder Drucker weggelassen, so daß sich die Zeitmessung jetzt nur auf den reinen Algorithmus bezieht.

Sortieralgorithmen im Vergleich

Getestet wurde bei allen Kandidaten ein zufallsbesetztes Feld von jeweils 100, 250 und 500 Elementen. Die Ergebnisse dieses Tests sind in Bild 4 dargestellt. Wie Sie sehen, schneidet Quicksort mit Abstand am besten ab; dicht gefolgt von Heapsort und etwas weiter von Shellsort. Diese drei Sortiermethoden eignen sich also für den praktischen Einsatz bei zufallsbesetzten Feldern, wobei im Vergleich zu den »niederen« Algorithmen sehr große Zeitvorteile zu verzeichnen sind. Bei 500 Elementen ist beispielsweise Quicksort über 16mal so schnell, wie unser (verbesserter) Bubblesort-2-Algorithmus.

Wie Sie sicherlich ahnen, hat diese schlechte Zeit, die sogar unser »normales« Bubblesort unterbietet, etwas mit der zusätzlichen Abfrage auf Vertauschungen zu tun.

Bei zufallsbesetzten Feldern ist Bubblesort 2 nicht akzeptabel!

Haben wir jedoch ein schon aufsteigend sortiertes Feld vorliegen, bei dem nur einige neue Elemente eingefügt werden sollen, so wird Bubblesort 2 fast unschlagbar, da hier die zusätzliche Abfrage für ein schnelles Ende, ohne zusätzliche, unnötige Arbeit, sorgt.

Bei unseren kleinen Variablenfeldern können also mit den höheren Sortierprogrammen ohne weiteres gute Zeiten erreicht werden.

Kritisch wird die ganze Angelegenheit jedoch, wenn Sie mit großen Feldern arbeiten, wobei unter Umständen noch ein riesiges Programm im Speicher steht.

Hier bekommt man es dann sehr schnell und auf höchst unangenehme Weise mit der »Müllabfuhr im Computer«, der Garbage Collection, zu tun (64'er, Ausgabe 1/1985, Seite 122).

Diese Tatsache ist jedoch nicht weiter verwunderlich, wenn man bedenkt, daß wir bei zufallsbesetzten Feldern beinahe das gesamte Array neu definieren und dabei entsprechend viel »Müll« erzeugen.

Sortieren ohne Müll

Haben Sie vielleicht schon einmal an eine andere Methode der Sortierung von Arrays gedacht? Es gibt nämlich noch eine weitere Möglichkeit, bei der man ohne viele Stringverschiebungen auskommt. Diese Möglichkeit der Sortierung und das Problem der Garbage Collection soll uns das nächstemal interessieren, wo wir uns dann einmal mit dem Sortieren der Indizes der Feldvariablen befassen. Außerdem können Sie sich schon einmal Gedanken zum Thema »Sortieren mehrdimensionaler Felder« machen. Das ist nämlich nicht annähernd so leicht, wie es vielleicht aussehen mag und erfordert eine ganze Menge Schweiß.

Für heute wollen wir es aber nun genug sein lassen. Experimentieren Sie doch ein wenig mit Quicksort, und lernen Sie diesen Algorithmus genauer kennen. Ich bin davon überzeugt, daß sich Ihnen früher oder später ein Problem stellt, bei dem Sie auf einen guten und schnellen Sortieralgorithmus angewiesen sein werden und den Sie nun Dank R. Hoare auch besitzen.

(Karsten Schramm/gk)

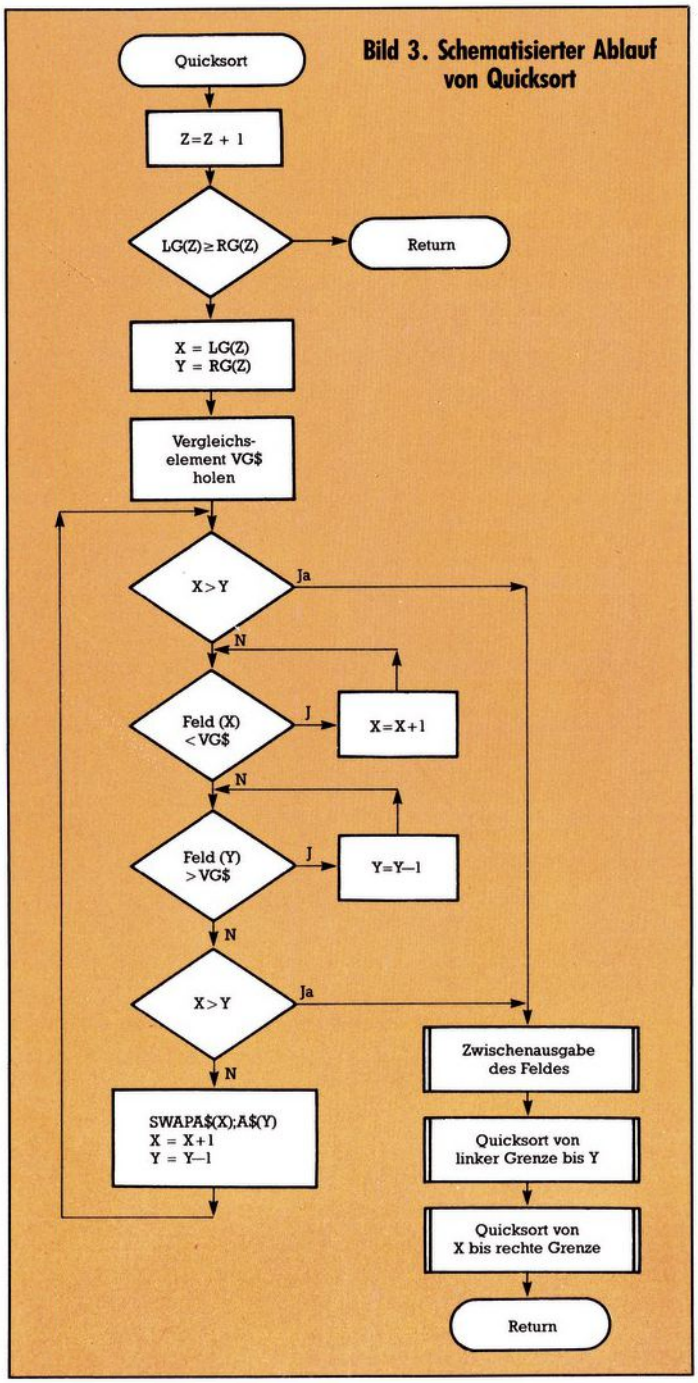


Bild 3. Schematisierter Ablauf von Quicksort

```
10000 REM SORTIEREN DURCH ZERLEGEN          <144>
10010 REM                                   <166>
10020 REM          QUICKSORT                <009>
10030 REM                                   <186>
10040 REM LG = LINKE GRENZE                 <253>
10050 REM RG = RECHTE GRENZE                <025>
10060 REM VG$ = VERGLEICHSELEMENT           <070>
10070 REM                                   <226>
10080 REM EINGANG DES HAUPTMODULS           <132>
10090 REM                                   <246>
10100 DIM LG(100),RG(100):Z=0:LG(1)=1:RG(1
      )=A                                   <128>
10110 GOSUB 10200: REM QUICKSORT            <251>
10120 GOTO 50000:REM ENDE                   <158>
10200 REM EINGANG DER REKURSIVSCHLEIFE      <172>
10210 Z=Z+1:IF LG(Z)>=RG(Z)THEN 10350       <058>
10220 X=LG(Z):Y=RG(Z)                       <023>
10225 REM VERGLEICHSELEMENT HOLEN           <158>
10230 VG$=A$(INT((X+Y)/2))                  <007>
10240 IF X>Y THEN 10320                     <230>
10250 IF A$(X)<VG$THEN X=X+1:GOTO 10250     <044>
10260 IF A$(Y)>VG$THEN Y=Y-1:GOTO 10260     <150>
10270 IF X>Y THEN 10320                     <004>
10280 S$=A$(X):A$(X)=A$(Y):A$(Y)=S$         <144>
10290 X=X+1:Y=Y-1                           <078>
10300 GOTO 10240                            <169>
10310 REM                                   <212>
10320 GOSUB 3000                            <132>
10330 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 10200   <212>
10340 LG(Z+1)=X:RG(Z+1)=RG(Z):GOSUB 10200   <132>
10350 Z=Z-1:RETURN                          <205>
10360 REM                                   <006>
```

Listing 1. Der Quicksort

Sortierprogramm	Anzahl der Elemente 100	250	500
1) Straight Insertion:	45 s	290 s	1225 s
2) Bubblesort:	99 s	639 s	2771 s
3) Bubblesort 2:	99 s	661 s	3030 s
4) Straight Select:	100 s	623 s	2989 s
5) Shellsort:	37 s	108 s	295 s
6) Heapsort:	31 s	95 s	214 s
7) Quicksort:	30 s	82 s	186 s

Bild 4. Direktvergleich aller Sortierprogramme (s = Sekunden)

Die unter der waagerechten Achse stehenden Ziffern bedeuten:

1 = Straight Insertion
2 = Bubblesort
3 = Bubblesort 2
4 = Straight Select
5 = Shellsort
6 = Heapsort
7 = Quicksort

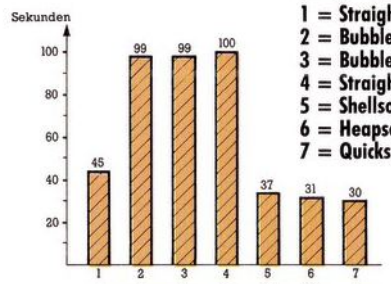


Bild 4a. Zeitvergleich bei 100 Elementen

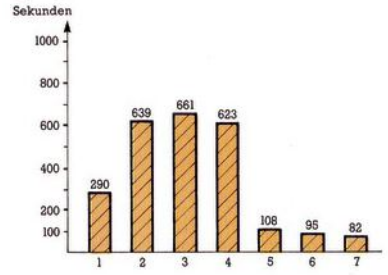


Bild 4b. Zeitvergleich bei 250 Elementen

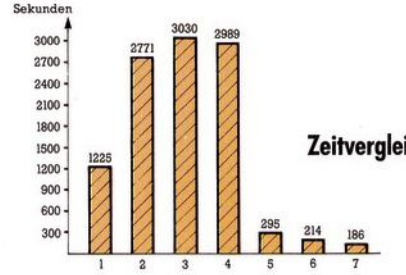


Bild 4c. Zeitvergleich bei 500 Elementen

Listing 2 bis 8. Hier sind noch einmal alle Sortieralgorithmen in optimierter Form zusammengefaßt. Beachten Sie dabei, daß das Erstellen des Sortierfeldes und die Ausgabe der Ergebnisse in dem Rahmenprogramm geschieht, das bereits in der 64'er-Ausgabe 4/85 abgedruckt wurde (Sortieren — Teil 1), aber auch auf der Leserservicediskette gespeichert ist. Beachten Sie die Eingabehinweise auf Seite 53.

```
10040 FOR X=2 TO A                                  <136>
10050 IF A$(X)>=A$(X-1)THEN 10120                   <038>
10070 X$=A$(X):FOR Y=X-1 TO 1 STEP-1                <117>
10080 A$(Y+1)=A$(Y)                                 <110>
10090 IF X$<=A$(Y-1)THEN 10110                      <114>
10100 A$(Y)=X$:GOTO 10120                           <186>
10110 NEXT Y                                        <160>
10120 NEXT X                                        <162>
```

Listing 2. Straight Insertion

```
10040 FOR X=A-1 TO 1 STEP-1                         <063>
10050 FOR Y=1 TO X                                  <006>
10060 IF A$(Y)<=A$(Y+1)THEN 10080                   <003>
10070 S$=A$(Y):A$(Y)=A$(Y+1):A$(Y+1)=S$             <094>
10080 NEXT Y                                        <130>
10090 NEXT X                                        <132>
```

Listing 3. Bubblesort

```
10040 FOR X=A TO 2 STEP-1:X$=""                     <096>
10050 FOR Y=1 TO X                                  <006>
10060 IF A$(Y)>X$THEN X$=A$(Y):Z=Y                  <238>
10070 NEXT Y                                        <120>
10080 S$=A$(X):A$(X)=A$(Z):A$(Z)=S$                 <218>
10090 NEXT X                                        <132>
```

Listing 4. Straight Select

```
10040 G=A-1:FOR X=A-1 TO 1 STEP-1                   <148>
10050 F=0:FOR Y=1 TO G                              <116>
10060 IF A$(Y)<=A$(Y+1)THEN 10080                   <003>
10070 F=Y:S$=A$(Y):A$(Y)=A$(Y+1):A$(Y+1)=S
      $                                             <143>
10080 NEXT Y                                        <130>
10090 G=F:IF F=0 THEN 50000                         <160>
10100 NEXT X                                        <142>
```

Listing 5. Bubblesort 2

```
10035 DIM AA$(A)                                    <016>
10040 S=INT(A/2)                                    <114>
10050 FOR X=1 TO S                                  <123>
10060 FOR Y=1 TO INT(A/S)                           <074>
10070 AA$(Y)=A$((Y-1)*S+X)                          <230>
10080 NEXT Y                                        <130>
10090 AA=Y-1:GOSUB 20000                            <082>
10100 FOR Y=1 TO INT(A/S)                           <114>
10110 A$((Y-1)*S+X)=AA$(Y)                          <025>
10120 NEXT Y                                        <170>
10130 NEXT X                                        <172>
10140 S=INT(S/2)                                    <090>
10160 IF S GOTO 10050                               <061>
10180 GOTO 50000                                    <014>
20000 FOR XX=2 TO AA                                <239>
20010 IF AA$(XX)>=AA$(XX-1) THEN 20080              <243>
20030 XX$=AA$(XX):FOR YY=XX-1 TO 1 STEP-1           <071>
20040 AA$(YY+1)=AA$(YY)                             <137>
20050 IF XX$<=AA$(YY-1)THEN 20070                   <184>
20060 AA$(YY)=XX$:GOTO 20080                        <246>
20070 NEXT YY                                       <107>
20080 NEXT XX                                       <093>
20090 RETURN                                        <080>
```

Listing 6. Shellsort

```
10040 LG=INT(A/2)+1:RG=A                            <075>
10050 IF RG<=1 THEN 50000                           <217>
10060 IF LG<=1 THEN 10110                           <195>
10080 LG=LG-1                                       <164>
10090 I=LG:GOTO 10140                               <105>
10110 S$=A$(1):A$(1)=A$(RG):A$(RG)=S$               <140>
10120 RG=RG-1                                       <165>
10130 I=1                                           <173>
10140 X$=A$(I)                                      <119>
10150 P=0                                           <205>
10160 IF 2*I<=RG AND P=0 THEN 10210                 <160>
10170 A$(I)=X$                                      <028>
10180 GOTO 10050                                    <047>
10210 J=2*I                                         <248>
10220 IF J<RG THEN IF A$(J)<A$(J+1)THEN J=
      J+1                                           <199>
10230 IF X$>=A$(J)THEN 10260                        <057>
10240 A$(I)=A$(J)                                   <209>
10250 I=J:GOTO 10160                                <228>
10260 P=1:GOTO 10160                                <121>
```

Listing 7. Heapsort

```
10100 DIM LG(100),RG(100):Z=0:LG(1)=1:RG(1
      )=A                                           <128>
10110 GOSUB 10210                                   <056>
10120 GOTO 50000                                    <210>
10210 Z=Z+1:IF LG(Z)>=RG(Z)THEN 10350               <058>
10220 X=LG(Z):Y=RG(Z)                               <023>
10230 VG$=A$(INT((X+Y)/2))                          <007>
10240 IF X>Y THEN 10330                             <234>
10250 IF A$(X)<VG$THEN X=X+1:GOTO 10250             <044>
10260 IF A$(Y)>VG$THEN Y=Y-1:GOTO 10260             <150>
10270 IF X>Y THEN 10330                             <008>
10280 S$=A$(X):A$(X)=A$(Y):A$(Y)=S$                 <144>
10290 X=X+1:Y=Y-1:GOTO 10240                        <116>
10330 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 10210           <020>
10340 LG(Z+1)=X:RG(Z+1)=RG(Z):GOSUB 10210           <196>
10350 Z=Z-1:RETURN                                  <205>
```

Listing 8. Quicksort