Sortieren mit Computer (Teil 6)

Wie werden Sortierroutinen schneller? Wie kann die Garbage Collection verhindert werden? Wir beschreiben einige Verfahren dazu und bringen Quicksort in Maschinensprache.

Im letzten Teil unseres Sortierkurses sollen einmal Techniken erläutert werden, die die bisher besprochenen Sortieralgorithmen noch effektiver werden lassen. Außerdem werden wir auf die wichtigsten der vielen Leserreaktionen eingehen und etwaige Mißverständnisse und Fehler aus dem Weg räumen.

Die wichtigsten Sortieralgorithmen wurden in unserem Kurs ausführlich besprochen, wobei wir bisher jedoch recht wenig auf Programmiertechniken eingegangen sind, die unseren Programmen zu noch größeren Geschwindigkeiten verhelfen. Zwei Methoden seien an dieser Stelle schon einmal erwähnt:
1) das Umschreiben der Sortieralgorithmen in Maschinensprache;
2) das Verhindern der Garbage Collection durch Sortieren der String-Deskriptoren, wobei kein »Stringmüll« entsteht.

Leserreaktionen

Bevor wir uns jedoch auf die gestellten Probleme stürzen, möchte ich mich bei all jenen Lesern bedanken, die mir zu diesem Thema geschrieben haben. Wir machen ohnehin kein Geheimnis daraus, daß die Zeitschrift 64'er zu einem erheblichen Teil von der Mitarbeit aktiver Leser geprägt wird, und wir hoffen, daß das auch in Zukunft so bleibt.

Nun aber zu einigen wichtigen Informationen.

Ziemlich viel Rummel hat offensichtlich die Ankündigung von Sortieralgorithmen hervorgerufen, die schneller sein wollen als Quicksort.

Es kamen prompt Zuschriften von Lesern, die diese Behauptung mit den eingesendeten Quicksort-Algorithmen widerlegen konnten. Der Trick bei der Sache war ausschließlich auf ein System zurückzuführen, das ein Feld von Quicksort nur teilsortieren läßt und die Arbeit bei einer Teillistenlänge von beispielsweise 10 an ein »kleines« Sortierprogramm (Bubblesort 2 oder Straight Insertion) übergibt. Diese Methode ist natürlich korrekt! Es wurde in dem Kurs jedoch absichtlich vom Abdruck eines solchen Sortieralgorithmus abgesehen, weil es bei den Beispielprogrammen einzig und allein um die Struktur der großen Sortieralgorithmen ging. Auf die oben beschriebene Methode zum »Schnellermachen« von Quicksort (und auch Heapsort) hatte ich aber an anderer Stelle schon hingewiesen.

Quicksort ist doch am schnellsten

Ein sehr wichtiger Brief kam jedoch von unserem Leser Kurt Sörensen aus Hamburg. Er zeigte nämlich einen Fehler auf, der in der abgedruckten Beispielroutine von Quicksort steckt (Achtung Fehlerteufelchen...). Herr Sörensen analysierte das Quicksortprogramm und kam zu folgendem Ergebnis:

»... Beim Übergang von der linken zur rechten Hälfte wird die rechte Grenze der linken Hälfte, die nach der Theorie schon sortiert ist, zur linken Grenze der rechten Hälfte gemacht, die nach der Theorie noch nicht sortiert ist. Dadurch werden praktisch alle Elemente außer dem kleinsten und dem größten doppelt sortiert...«

Wie Sie aus unserem Kurs wissen, teilt Quicksort während des Sortierens das Variablenfeld in immer kleiner werdende Hälften (Teillisten) auf. Berücksichtigt man also diesen Fehler und erstellt das Quicksortprogramm neu (nach wie vor ohne anschließenden anderen Algorithmus), so kann sich Quicksort wieder unbeschadet an die Spitze unserer Stringsortierroutinen stellen. Es ist und bleibt der schnellste (und dabei der vielseitigste) Sortieralgorithmus (»Sonderanfertigungen« für spezielle Probleme laufen natürlich außer Konkurrenz!)

In Listing 1 sehen Sie die korrigierte Version von Quicksort abgedruckt.

Nun zu einem Problem, das offensichtlich in Zusammenhang mit unserem Hauptprogramm für die Sortieralgorithmen aufgetreten ist. Wie Sie wissen, haben wir zu den entsprechenden Sortierprogrammen auch ein Hauptprogramm abgedruckt, das einen Test der einzelnen Routinen ermöglichen sollte. Der Sinn dieses Programms ist aber von mir offensichtlich nicht genug verdeutlicht worden.

Bei dem beigefügten Rahmenprogramm handelt es sich um ein provisorisches Gerüst, das es dem Leser erlauben soll, die eingegebenen Sortierprogramme auf Funktionsfähigkeit und Geschwindigkeit zu testen. Daß das Rahmenprogramm also kein »Z« als Zufallswert verarbeitet und außerdem nur Elementzahlen in Zehnersprüngen zuläßt, dürfte bei der Arbeit kaum ein Hindernis darstellen, zumal das ja der Sinn der Sortierroutinen ist, nach Fertigstellung in andere Programme eingefügt zu werden. Im eigentlichen Sinne wichtig für den Anwender sind also jeweils die Programmteile in den Zeilen 10000 bis 20000: Der Rest entfällt.

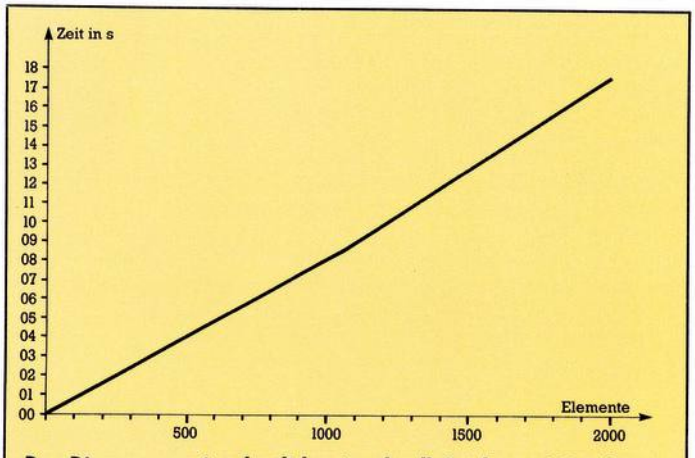
Nun aber zu den schon erwähnten Programmiertechniken, die der Beschleunigung eines Sortiervorgangs dienen.

Es geht noch schneller

Wir wollen uns dazu zuerst mit einem Artikel in der 64'er, Ausgabe 1/1985 beschäftigen. Es handelt sich hierbei um die erste Ausgabe in der Reihe »Effektives Programmieren«, die damals von dem Stringspezialisten Boris Schneider geschrieben wurde. Vorherrschend ging es um ein Problem bei der Stringverwaltung, nämlich um die »Müllabfuhr im Computer«, die Garbage Collection. Damals wurde sehr ausführlich auf den Aufbau von Strings im Speicher des Computers eingegangen, weshalb an dieser Stelle nur eine sehr knappe Wiederholung folgen soll.

Generell legt der C 64 seine Variablen direkt im Anschluß an das Basic-Programm im Speicher ab. Auch die Stringvariablen stehen dort. Der Textinhalt dieser Variablen wird jedoch an anderer Stelle im Speicher,

Fortsetzung auf Seite 153



Das Diagramm zeigt deutlich wie schnell Quicksort OBJ arbeitet

```
10000 REM        QUICKSORT                    <245>
10010 :                                       <080>
10020 T=TI:LG(1)=1:RG(1)=A:Z=0:GOSUB 10040    <173>
10030 RETURN                                  <182>
10040 Z=Z+1:IF LG(Z)>=RG(Z)THEN 10170         <146>
10050 X=LG(Z):Y=RG(Z):IF Y<=X+1 GOTO 10170    <099>
10060 B=(X+Y)/2:B=INT(B):VG$=A$(B)            <107>
10070 IF X>Y THEN 10150                       <066>
10080 IF A$(X)<VG$THEN X=X+1:GOTO 10080       <130>
10090 IF A$(Y)>VG$THEN Y=Y-1:GOTO 10090       <235>
10100 IF X>Y THEN 10150                       <096>
10110 S$=A$(X):A$(X)=A$(Y):A$(Y)=S$           <228>
10120 X=X+1:Y=Y-1:GOTO 10070                  <208>
10130 IF A$(X)<=A$(Y)GOTO 10170               <019>
10140 S$=A$(X):A$(X)=A$(Y):A$(Y)=S$:GOTO 1
      0170                                    <204>
10150 RG(Z+1)=Y:LG(Z+1)=LG(Z):GOSUB 10040     <223>
10160 LG(Z+1)=RG(Z+1)+1:RG(Z+1)=RG(Z):GOSU
      B 10040                                 <002>
10170 Z=Z-1:RETURN                            <023>
```

© 64'er

Listing 1. Die korrigierte Version von Quicksort

Fortsetzung von Seite 150

nämlich von oben anfangend, rückwärts nach unten gespeichert. Anstelle des Textes hinter dem Variablennamen wird dort ein Zeiger (Deskriptor) abgelegt, der auf die jeweilige Position des Textes zeigt.

Wird nun eine Stringvariable neu angelegt, nachdem sie zuvor einen anderen Inhalt aufwies, wird der neue Text an die Stringkette im oberen Speicherbereich angehängt und der Deskriptor der Variablen auf diesen neuen Text eingestellt. Der »alte« Variablentext bleibt im Speicher stehen und bildet »Stringmüll«. Dieser Müll steigt mit vielen Neudefinitionen von Strings rapide an, so daß es nur eine Frage der Zeit ist, wann die von oben kommenden Strings mit den von unten kommenden Variablen zusammenstoßen und einen »?OUT OF MEMORY ERROR« verursachen.

Um diesem Fehler vorzubeugen, gibt es die Garbage Collection. Der Interpreter überwacht laufend den Zustand des Speichers. Wird es zu eng, dann tritt die Garbage Collection in Kraft und räumt den gesamten Stringmüll weg. Dieser Vorgang erfordert enorme Speichersuch- und verschiebevorgänge und kann ungünstigenfalls sogar mehrere Stunden benötigen: Der Computer scheint »abgestürzt«.

Bei unseren Sortiervorgängen wird ziemlich häufig der sogenannte Dreiecktausch durchgeführt. Es handelt sich hierbei um das Vertauschen der Inhalte von zwei Stringvariablen, wobei eine dritte Variable als Zwischenspeicher dient: zum Beispiel:
x$ = a$(1):a$(1) = a$(2):a$(2) = x$

Bei diesem Tauschverfahren werden gleich drei Müllstrings erzeugt, nämlich der Inhalt von x$, der alte Inhalt von a$(1) und der alte Inhalt von a$(2).

Nun gibt es Basic-Interpreter, die bieten zu diesem Zweck den Befehl SWAP an. Mit diesem Befehl können die Inhalte zweier Strings direkt vertauscht werden. Zum Beispiel:
SWAP a$(1), a$(2)

Hiermit sparen wir Zeit und Speicherplatz. Zeit sparen wir durch die Ausführung eines einzigen Befehls anstatt der drei Variablenzuordnungen. Speicherplatz sparen wir durch das Wegfallen der Hilfsvariable x$, so daß nur zwei Müllstrings entstehen.

SWAP — ein Programm mit Pfiff

Aber pingelig, wie wir Computermenschen nun einmal sind, stellt uns auch diese Methode nicht zufrieden. Hatten wir vorhin nicht etwas von Deskriptoren, also von Zeigern auf den jeweiligen String gehört? Genau! Das ist unser neuer Ansatzpunkt!

Bei dem Vertauschen von zwei Variablen ändert sich nämlich eigentlich gar nichts im Speicher. Es bleiben sowohl die beiden Strings als auch die beiden Variablennamen erhalten. Warum reicht es also nicht aus, einfach die beiden Stringdeskriptoren zu vertauschen? Diese Frage ist überflüssig! Es reicht nämlich in der Tat aus, wenn wir den Deskriptor von Variable 1 auf den String von Variable 2 und umgekehrt setzen.

Und genau das macht das Programm SWAP, das Boris Schneider schon in der 64'er, Ausgabe 1/1985, Seite 123 vorgestellt hat.

Mit Hilfe dieser kleinen Maschinenspracheroutine sparen wir also Zeit und Stringmüll, da kein einziger überflüssiger String entsteht (Listing 2). Wenn Sie das »Prögrämmchen« eingetippt haben, dann starten Sie es mit RUN. Sie werden anschließend nach der Startadresse des Maschinenprogramms gefragt. Diese sollten Sie vorzugsweise in den $C-Bereich (49152 bis 53247) legen, wobei darauf zu achten ist, daß als Startadresse maximal 53199 gelten darf (Das Programm benötigt 48 Byte).

Muß die Routine aus irgendeinem Grund woanders untergebracht werden, so wäre noch der Kassettenpuffer (828 bis 1023) zu empfehlen, um Basic-Speicherplatz zu sparen. Andernfalls müssen Sie eben die maximale Speicheradresse entsprechend heruntersetzen, um die Swap-Routine vor dem Überschreiben mit Strings zu schützen. Der Einbau des SWAP in die Sortierprogramme ist vollkommen unproblematisch. Sie suchen sich einfach jeweils die Stelle mit dem Dreiecktausch heraus. Sie wird in den abgedruckten Sortierprogrammen durch die Variable S$ als Hilfsvariable gekennzeichnet. Bei Straight Select ist das beispielsweise die Zeile 10080:
10080 S$ = A$(X):A$(X) = A$(Z):A$(Z) = S$

Diese Zeile wird nun wie folgt geändert:
10080 SYS startadresse(A$(X), A$(Z))

»startadresse« gibt hierbei die Zahl an, die Sie beim Start des SWAP-Programms angegeben hatten. Die abgeänderte Version von Straight Select zeigt Listing 3. Sinnvoll wäre es, die SWAP-Routine direkt vor unser Sortier-Hauptprogramm zu setzen, so daß sie immer direkt vor dem Arbeiten automatisch installiert wird.

Von der Speicherplatzersparnis einmal ganz abgesehen, arbeitet auch diese SWAP-Routine schon erheblich schneller als der Dreiecktausch, so daß zum Beispiel beim Sortieren von 100 Elementen mit Bubblesort2 aus einer Sortierzeit von 1 Minute 46 Sekunden »nur« 1 Minute und 33 Sekunden wurden. Besonders bei sehr großen Elementzahlen zeigt sich aber dann die Effizienz dieses Programms, da sich die Garbage Collection kaum mehr blicken läßt. Unser Quicksort erfährt dadurch wiederum neue Bestzeiten im Sortieren von zufallsbesetzten Feldern.

Es geht doch nichts über Maschinensprache

Wem aber auch dieser Trick noch nicht reicht: Wer immer noch auf der Suche nach dem »HYPRA« ist, dem bleibt nichts anderes übrig, als auf der Maschinenspracheebene sein Glück zu versuchen.

Mit diesem Thema hat sich auch unser Leser Frank Probst aus Zweibrücken beschäftigt. Was dabei herausgekommen ist, wollen wir Ihnen jetzt vorstellen: Quicksort in Maschinensprache!

Es war natürlich von vornherein klar, daß wirklich gute Zeiten beim Sortieren von Feldern nur in Maschinensprache zu erreichen sind. Wenn man jedoch das Prinzip eines Sortieralgo-

Fortsetzung auf Seite 156

```
10 DATA 32,250,174,32,158,173,32,143          <194>
20 DATA 173,165,100,133,247,165,101,133       <144>
30 DATA 248,32,253,174,32,158,173,32          <054>
40 DATA 143,173,160,0,177,247,133,249         <228>
50 DATA 177,100,145,247,165,249,145,100       <158>
60 DATA 200,192,3,208,239,32,247,174          <105>
70 DATA 96,0,0,0,0,0,0,0                      <194>
100 INPUT"STARTADRESSE";SA                    <095>
110 FOR I=SA TO SA +48                        <152>
120 READ X:POKE I,X:CS=CS+X                   <220>
130 NEXT I                                    <214>
140 IF CS<>7314 THEN PRINT"FEHLER!!"          <125>
150 END                                       <152>
```

© 64'er

Listing 2. Die SWAP-Routine

```
10040 TI$="000000":G=A-1:FOR X=A-1 TO 1 ST
      EP-1                                    <154>
10050 F=0:FOR Y=1 TO G                        <116>
10060 IF A$(Y)<=A$(Y+1)THEN 10080             <003>
10070 F=Y:SYS 49152(A$(Y),A$(Y+1))            <052>
10080 NEXT Y                                  <130>
10090 G=F:IF F=0 THEN 50000                   <160>
10100 NEXT X                                  <142>
```

© 64'er

Listing 3. Sortieren ohne »Stringmüll«

```
2 IF PEEK(2051)<>3 THEN POKE 2051,3:LOAD"Q
  UICKSORT.OBJ",8,1                           <213>
5 REM BITTE ZEILE 2 GENAUSO EINGEBEN          <094>
10 INPUT "ANZAHL={2SPACE}100{5LEFT}";A        <025>
20 DIM A$(A)                                  <099>
23 REM                                        <085>
25 REM  JEDES ELEMENT MIT 3 ZEICHEN           <148>
28 REM                                        <090>
30 FOR I = 1 TO A                             <136>
35 : FOR J = 0 TO 2                           <161>
40 :   A$(I) = A$(I)+CHR$(RND(1)*26+65)       <091>
50 : NEXT J                                   <210>
55 NEXT I                                     <139>
56 REM                                        <118>
57 REM        ELEMEMTE AUSGEBEN               <077>
58 REM                                        <120>
60 FOR I = 1 TO A                             <166>
63 : PRINT A$(I);" ";                         <036>
65 NEXT I                                     <149>
66 PRINT                                      <168>
67 REM                                        <129>
68 REM        ELEMENTE SORTIEREN              <025>
69 REM                                        <131>
70 TI$ = "000000"                             <049>
74 SYS 52000                                  <186>
76 T = TI                                     <160>
77 REM                                        <139>
78 REM        ELEMENTE AUSGEBEN               <099>
79 REM                                        <141>
80 FOR I= 1 TO A                              <186>
85 : PRINT A$(I);" ";                         <058>
90 NEXT I                                     <174>
95 PRINT T/60                                 <019>
```

© 64'er

Listing 6. »Quicksort« demonstriert die Sortiergeschwindigkeit

Fortsetzung von Seite 153

rithmus verstehen will, ist es in der Regel besser, sein Glück erst einmal in Basic zu versuchen und das erworbene Wissen dann in Maschinensprache umzusetzen.

Quicksort in Maschinensprache!

Da Quicksort ja nun schon in Basic am schnellsten ist und erstaunliche Sortierzeiten hervorbringt, darf man auf die Maschinenpracheversion gespannt sein. Ich kann schon vorwegnehmen, daß dieses Programm (natürlich) alles bisher Dagewesene voll in den Schatten stellt. Bevor wir uns jedoch diesem Programm zuwenden, soll an dieser Stelle erwähnt sein, daß durchaus nicht alle Tricks, die der Geschwindigkeit von Nutzen sein könnten, angewendet wurden. So teilt dieses Quicksort alle Teillisten bis auf die Länge 1 herunter, anstatt bei zum Beispiel 10 aufzuhören und dann Straight Select ans Werk zu lassen. Die Methode des Deskriptortausches wurde jedoch auch hier angewendet, was zur Folge hat, daß die Garbage Collection so gut wie keine Arbeit bekommt (selbst wenn große Mengen an Daten sortiert werden müssen).

Die Bedienung von Quicksort-M (so möchte ich es für den weiteren Verlauf nennen, um es von der Basic-Version zu unterscheiden) ist denkbar einfach. Nachdem Sie das Listing 4 mit dem MSE eingetippt und auf Diskette gespeichert haben, steht Quicksort-M sofort zur Verfügung. Es belegt im Speicher den Bereich von $CB20 bis $CFFF und verträgt sich so mit einem eventuell eingeschalteten Turbo-Tape. Es wird mit SYS52000 aufgerufen, wobei jedoch einige Regeln zu beachten sind:

Das zu sortierende Feld muß ein Stringarray sein und als allererstes Feld in einem Programm dimensioniert werden. Andernfalls kann ein »Aussteigen« des Computers die Folge sein. Bei der Arbeit benötigt Quicksort-M die Speicherstellen von $00B2-$00B8 und $00FB-$00FE. Die Werte aus $00B2 bis $00B8 werden gerettet und nach der Sortierung wieder zurückgeschrieben.

```
 programm : quicksort.obj  cb20 cde7
 ------------------------------------


cb20 : 20 e0 cc a2 00 b5 b2 9d  eb
cb28 : bc 02 e8 e0 06 d0 f6 20  3e
cb30 : 3f cb a2 00 bd bc 02 95  f2
cb38 : b2 e8 e0 06 d0 f6 60 20  de
cb40 : 03 cc 20 b6 cd 20 ee cb  b9
cb48 : c9 00 f0 58 c9 02 f0 54  72
cb50 : 20 a8 cd 20 30 cd 20 d3  d5
cb58 : cb c9 02 f0 3b 20 1a cd  5f
cb60 : 20 a7 cb c9 02 f0 0a c9  e3
cb68 : 00 f0 06 20 27 cc 4c 5d  2b
cb70 : cb 20 20 cd 20 a7 cb c9  0f
cb78 : 01 f0 0a c9 00 f0 06 20  8d
cb80 : 39 cc 4c 71 cb 20 d3 cb  05
cb88 : c9 02 f0 0c 20 a8 cc 20  cb
cb90 : 27 cc 20 39 cc 4c 56 cb  6d
cb98 : 20 8b cd 20 3f cb 20 6e  a5
cba0 : cd 20 3f cb 4c 15 cc a0  a8
cba8 : ff c8 c4 b2 d0 05 a9 01  71
cbb0 : 4c ca cb c4 b5 d0 05 a9  36
cbb8 : 02 4c ca cb b1 b3 d1 b6  7a
cbc0 : f0 e7 90 03 a9 02 2c a9  d7
cbc8 : 01 60 a6 b2 e4 b5 d0 02  3c
cbd0 : a9 00 60 ad ea cd cd ec  75
cbd8 : cd d0 08 ad e9 cd cd eb  e1
cbe0 : cd f0 05 b0 06 a9 01 2c  87
cbe8 : a9 00 2c a9 02 60 ad f0  8d
cbf0 : cd cd f2 cd d0 08 ad ef  fe

cbf8 : cd cd f1 cd f0 ea b0 eb  e3
cc00 : a9 01 60 18 ad e7 cd 69  69
cc08 : 04 8d e7 cd ad e8 cd 69  b2
cc10 : 00 8d e8 cd 60 38 ad e7  19
cc18 : cd e9 04 8d e7 cd ad e8  02
cc20 : cd e9 00 8d e8 cd 60 18  42
cc28 : ad e9 cd 69 01 8d e9 cd  2a
cc30 : ad ea cd 69 00 8d ea cd  a7
cc38 : 60 38 ad eb cd e9 01 8d  e8
cc40 : eb cd ad ec cd e9 00 8d  62
cc48 : ec cd 60 ad e9 cd 0a aa  73
cc50 : ad ea cd 20 8d cc 6d e9  b2
cc58 : cd aa 98 6d ea cd 4c 92  c1
cc60 : cc ad eb cd 0a aa ad ec  3e
cc68 : cd 20 8d cc 6d eb cd aa  05
cc70 : 98 6d ec cd 4c 92 cc ad  9b
cc78 : ed cd 0a aa ad ee cd 20  ed
cc80 : 8d cc 6d ed cd aa 98 6d  fc
cc88 : ee cd 4c 92 cc 2a a8 8a  98
cc90 : 18 60 a8 18 8a 69 07 aa  6b
cc98 : 98 69 00 a8 18 8a 65 2f  c4
cca0 : 85 fb 98 65 30 85 fc 60  d9
cca8 : 20 4b cc a5 fb 85 fd a5  85
ccb0 : fc 85 fe 20 61 cc a0 00  31
ccb8 : b1 fb aa b1 fd 91 fb 8a  b9
ccc0 : 91 fd c8 c0 03 d0 f1 60  d9
ccc8 : 18 ad e9 cd 6d eb cd 8d  73
ccd0 : ed cd ad ea cd 6d ec cd  04
ccd8 : 4a 8d ee cd 6e ed cd 60  ac
cce0 : a9 00 8d e9 cd 8d ea cd  ba
cce8 : a9 07 8d e7 cd a9 ce 8d  f6

ccf0 : e8 cd 20 4b cc ee e9 cd  b8
ccf8 : 38 a5 fb e9 02 85 fb a5  c6
cd00 : fc e9 00 85 fc a0 01 38  eb
cd08 : b1 fb e9 01 8d eb cd 88  d2
cd10 : b1 fb e9 00 8d ec cd 4c  49
cd18 : d7 cd 20 4b cc 4c 23 cd  9f
cd20 : 20 61 cc a0 00 b1 fb 99  e9
cd28 : b2 00 c8 c0 03 d0 f6 60  78
cd30 : 20 c8 cc 20 77 cc a0 00  4c
cd38 : b1 fb 99 b5 00 c8 c0 03  53
cd40 : d0 f6 a5 b5 f0 1c c9 15  ed
cd48 : 90 04 a9 14 85 b5 a0 00  50
cd50 : b1 b6 99 f3 cd c8 c4 b5  e3
cd58 : d0 f6 a9 f3 85 b6 a9 cd  dc
cd60 : 85 b7 60 ad e7 cd 85 fd  8e
cd68 : ad e8 cd 85 fe 60 20 b6  8e
cd70 : cd 20 c6 cd a0 00 b9 e9  7d
cd78 : cd 91 fd c8 c0 02 d0 f6  f3
cd80 : b9 ef cd 91 fd c8 c0 04  07
cd88 : d0 f6 60 20 b6 cd 20 c6  d7
cd90 : cd a0 00 b9 ef cd 91 fd  94
cd98 : c8 c0 02 d0 f6 b9 e9 cd  db
cda0 : 91 fd c8 c0 04 d0 f6 60  dd
cda8 : a0 00 b9 ef cd 99 e9 cd  a1
cdb0 : c8 c0 04 d0 f5 60 20 63  9d
cdb8 : cd a0 00 b1 fd 99 ef cd  14
cdc0 : c8 c0 04 d0 f6 60 20 63  bd
cdc8 : cd 18 a5 fd 69 04 85 fd  93
cdd0 : a5 fe 69 00 85 fe 60 20  61
cdd8 : c6 cd a0 00 b9 e9 cd 91  f2
cde0 : fd c8 c0 04 d0 f6 60 00  38
```

Listing 4. Der Quicksort in Maschinensprache. Bitte beachten Sie die Eingabehinweise auf Seite 54

```
10 SYS8*4096:.OPT P,OO:*= 52000
20 LAENGE1 = $B2
30 LAENGE2 = $B5
40 STR1    = $B3
50 STR2    = $B6
55 UMULT1  = $28
56 UMULT2  = $71
57 UMULT   = $B357
58 AARRAY  = $2F
59 VEKTOR1 = $FB
60 VEKTOR2 = $FD
100             JSR REGSET
101             LDX #0
102 MARKE1      LDA LAENGE1,X
103             STA 700,X
104             INX
105             CPX #6
106             BNE MARKE1
110             JSR HAUPTSCHL
111             LDX #0
112 MARKE2      LDA 700,X
113             STA LAENGE1,X
114             INX
115             CPX #6
116             BNE MARKE2
117             RTS
130 ;
140 HAUPTSCHL   JSR HOCHZ
145             JSR HOLLR
150             JSR LRVERGL
160             CMP #0
170             BEQ Z350
180             CMP #2
190             BEQ Z350
195             JSR HOLXY
210             JSR EVINDI
220 Z270        JSR XYVERGL
230             CMP #2
240             BEQ Z330
250 Z280        JSR EXINDI
260             JSR EINSPR
270             CMP #2
280             BEQ Z290
285             CMP #0
286             BEQ Z290-
290             JSR HOCHX
300             JMP Z280
310 Z290        JSR EYINDI
320             JSR EINSPR
330             CMP #1
340             BEQ Z300
345             CMP #0
346             BEQ Z300
350             JSR RUNTERY
360             JMP Z290
370 Z300        JSR XYVERGL
380             CMP #2
390             BEQ Z330
400             JSR SWAP
410             JSR HOCHX
420             JSR RUNTERY
430             JMP Z270
435 ;
440 Z330        JSR PUSHLY
460             JSR HAUPTSCHL
470 ;
480             JSR PUSHXR
500             JSR HAUPTSCHL
510 ;
940 Z350        JMP RUNTERZ
1005 ;
1010 ;    VERGLEICH STR1 MIT VERGL$
1011 ;1) STR1<VERGL  2) STR1>VERGL
1015 ;
1020 EINSPR     LDY #$FF
1030 SCHL1      INY
1040            CPY LAENGE1
1050            BNE WEITER1
1060            LDA #1
1070            JMP RAUS
1080 WEITER1    CPY LAENGE2
1090            BNE WEITER2
1100            LDA #2
1110            JMP RAUS
1120 WEITER2    LDA (STR1),Y
1130            CMP (STR2),Y
1140 BEQ SCHL1
1150            BCC WEITER3+1
1160            LDA #2
1170 WEITER3    BIT $1A9 ;MASKIERUNG FUER
                         LDA #1
1180 ;LDA #1
1200            RTS
1210 RAUS       LDX LAENGE1
1220            CPX LAENGE2
1230            BNE FERTIG
1240            LDA #0
1250 FERTIG     RTS
1260 ;
1270 ;    VERGLEICHEN VON X UND Y
1275 ;X>Y LDA #2  X<Y LDA #1  X=Y LDA #0
1280 ;
1290 XYVERGL    LDA XREG+1
1300            CMP YREG+1
1310            BNE WEITER4
1320            LDA XREG
1330            CMP YREG
1340            BEQ GLEICH+1
1350 WEITER4    BCS GROESSER+1
1360            LDA #1
1370 GLEICH     .BYT $2C,$A9,0 ;BIT $00A9
1380 GROESSER   BIT $2A9
1390            RTS
1400 ;
1410 ;    VERGLEICHEN VON L UND R
1415 ;L>R LDA #2  L<R LDA #1  L=R LDA #0
1420 ;
1430 LRVERGL    LDA LREG+1
1440            CMP RREG+1
1450            BNE WEITER5
1460            LDA LREG
1470            CMP RREG
1480            BEQ GLEICH+1
1490 WEITER5    BCS GROESSER+1
1500            LDA #1
1510            RTS
1985 ;
1990 ; REGISTER HOCH- UNG RUNTERZAEHLEN
1995 ;
2000 HOCHZ      CLC
2020            LDA ZREG
2030 ADC #4
2040            STA ZREG
2050            LDA ZREG+1
2060            ADC #0
2070            STA ZREG+1
```

Listing 5. Listing 4 als Quelltext

Das eigentliche Quicksort-M-Programm belegt nur die Speicherstellen $CB20 bis $CDE7. Es benötigt jedoch den anschließenden Bereich als Speicher für den jeweiligen Vergleichsstring und für einen Software-Stack, der bei Quicksort ja generell notwendig ist.

Wer sich mit Quicksort-M weiter beschäftigen will, der findet in Listing 5 ein Source-Listing.

Nun aber zu den Daten von Quicksort-M. Hier erübrigt sich jeder weitere Kommentar, wenn Sie sich Bild 1 betrachten. Diese Zeiten wurden mit dem Basic-Programm Quicktester (Listing 6) ermittelt und lassen einen das Schwärmen anfangen. Quicksort-M benötigt beispielsweise für 1000 zufällig ausgewählte Elemente nur noch acht Sekunden (!). Diese Zeit dürfte sich dabei auf ein Maß beschränken, das auch den pingeligsten Anwender des C 64 zufriedenstellen dürfte. Immerhin schlägt Quicksort-M seine Basic-Konkurrenten alle um einige 1000 Prozent: Sogar das »normale« Quicksort wird hier haushoch geschlagen.

Ursprünglich hatte ich vor, an dieser Stelle auch noch einen Bubblesort-Algorithmus in Maschinensprache vorzustellen. Doch was brauchen wir jetzt noch Bubblesort? Quicksort-M dürfte, was die Geschwindigkeit angeht, wohl allen Anwendungen gewachsen sein. Wer sich dennoch mit Bubblesort auseinandersetzen will, der findet in einer anderen Zeitschrift aus dem Markt & Technik-Verlag einen Beitrag zu diesem Thema: Er steht unter dem Titel »Schneller als Quicksort« und erschien in der Ausgabe 14/1985 des Magazins Computer persönlich. Weitere Sortierprogramme finden Sie in Computer persönlich, Ausgabe 14/1984 (Top-Sort) und im 64'er, Ausgabe 11/84 (Exsort), ebenfalls als Maschinenprogramm.

Ich möchte mich an dieser Stelle von Ihnen verabschieden und hoffe, daß Ihnen die letzten Kurse ein wenig Spaß gemacht haben. Vielleicht haben Sie jetzt das Werkzeug, um sich das eine oder andere Projekt, das Sie sich schon länger vorgenommen hatten, zu verwirklichen.

(K. Schramm/F. Probst/gk)

```
2090            RTS
2100 RUNTERZ    SEC
2120            LDA ZREG
2130            SBC #4
2140            STA ZREG
2150            LDA ZREG+1
2160            SBC #0
2170            STA ZREG+1
2190            RTS
2200 HOCHX      CLC
2220            LDA XREG
2230            ADC #1
2240            STA XREG
2250            LDA XREG+1
2260            ADC #0
2270            STA XREG+1
2290            RTS
2300 RUNTERY    SEC
2320            LDA YREG
2330            SBC #1
2340            STA YREG
2350            LDA YREG+1
2360            SBC #0
2370            STA YREG+1
2390            RTS
2985 ;
2990 ; DIE MIT X/Y INDIZIERTE VARIABLE
2991 ; WIRD GESUCHT Z.B. ( A$(X) )
2995 ;
3000 XSUCH      LDA XREG
3010            ASL
3015            TAX
3020            LDA XREG+1
3030            JSR PRG1
3040            ADC XREG
3050            TAX
3060            TYA
3070            ADC XREG+1
3080            JMP PRG2
3100 YSUCH      LDA YREG
3110            ASL
3115            TAX
3120            LDA YREG+1
3130            JSR PRG1
3131            ADC YREG
3132            TAX
3133            TYA
3134            ADC YREG+1
3135            JMP PRG2
3140 VSUCH      LDA VERGL
3150            ASL
3155            TAX
3160            LDA VERGL+1
3170            JSR PRG1
3171            ADC VERGL
3172            TAX
3173            TYA
3174            ADC VERGL+1
3175            JMP PRG2
3200 PRG1       ROL
3210            TAY
3220            TXA
3230            CLC
3240            RTS
3250 PRG2       TAY
3260            CLC
3270            TXA
3280            ADC #7
3281            TAX
3282            TYA
3283            ADC #0
3284            TAY
3285            CLC
3286            TXA
3290            ADC AARRAY
3300            STA VEKTOR1
3310            TYA
3320            ADC AARRAY+1
3330            STA VEKTOR1+1
3340            RTS
3985 ;
3990 ;SWAP - VERTAUSCHEN ZWEIER STRINGS
3995 ;
4000 SWAP       JSR XSUCH
4010            LDA VEKTOR1
4020            STA VEKTOR2
4030            LDA VEKTOR1+1
4040            STA VEKTOR2+1
4050            JSR YSUCH
4060            LDY #0
4070 SCHL2      LDA (VEKTOR1),Y
4080            TAX
4090            LDA (VEKTOR2),Y
4100            STA (VEKTOR1),Y
4110            TXA
4120            STA (VEKTOR2),Y
4130            INY
4140            CPY #3
4150            BNE SCHL2
4160            RTS
4985 ;
4990 ;     VERGL = (XREG+YREG)/2
4995 ;
5000 RECHNUNG   CLC
5010            LDA XREG
5020            ADC YREG
5030            STA VERGL
5040            LDA XREG+1
5050            ADC YREG+1
5060            LSR
5070            STA VERGL+1
5080            ROR VERGL
5090            RTS
5100 ;
5110 ;REGISTER AUF AUSGANGSWERTE SETZEN
5120 ;
5200 REGSET     LDA #0
5210            STA XREG
5215            STA XREG+1
5220            LDA #<STACK
5225            STA ZREG
5230            LDA #>STACK
5235            STA ZREG+1
5240            JSR XSUCH
5245            INC XREG
5250            SEC
5260            LDA VEKTOR1
5270            SBC #2
5280            STA VEKTOR1
5290            LDA VEKTOR1+1
5300            SBC #0
5310            STA VEKTOR1+1
5320            LDY #1
5325            SEC
5330            LDA (VEKTOR1),Y
5335            SBC #1
5340            STA YREG
5350            DEY
5360            LDA (VEKTOR1),Y
5365            SBC #0
5370            STA YREG+1
5380            JMP PUSHXY
5985 ;
5990 ;DISCRIPTOREN IN DER ZP EINRICHTEN
5995 ;
6000 EXINDI     JSR XSUCH
6010            JMP DISCRIP1
6020 ;
6030 EYINDI     JSR YSUCH
6040 ;
6050 DISCRIP1   LDY #0
6060 SCHL3      LDA (VEKTOR1),Y
6070            STA LAENGE1,Y
6080            INY
6090            CPY #3
6100            BNE SCHL3
6110            RTS
6120 ;
6130 EVINDI     JSR RECHNUNG
6135            JSR VSUCH
6140            LDY #0
6150 SCHL4      LDA (VEKTOR1),Y
6160            STA LAENGE2,Y
6170            INY
6180            CPY #3
6190            BNE SCHL4
6200            LDA LAENGE2
6205            BEQ KZEICHEN
6210            CMP #21
6220            BCC KLEINER
6230            LDA #20
6240            STA LAENGE2
6250 KLEINER    LDY #0
6260 NZEICHEN   LDA (STR2),Y
6270            STA VSTR,Y
6280            INY
6290            CPY LAENGE2
6300            BNE NZEICHEN
6310            LDA #<VSTR
6320            STA STR2
6330            LDA #>VSTR
6340            STA STR2+1
6350 KZEICHEN   RTS
7000 STCKVEK    LDA ZREG
7010            STA VEKTOR2
7020            LDA ZREG+1
7030            STA VEKTOR2+1
7040            RTS
7045 ;
7100 PUSHXR     JSR HOLLR
7105            JSR VEKTOR4
7110            LDY #0
7120 SCHL5      LDA XREG,Y
7130            STA (VEKTOR2),Y
7150            INY
7160            CPY #2
7170            BNE SCHL5
7172 SCHL6      LDA RREG-2,Y
7173            STA (VEKTOR2),Y
7174            INY
7175            CPY #4
7176            BNE SCHL6
7177            RTS
7178 ;
7180 PUSHLY     JSR HOLLR
7185            JSR VEKTOR4
7190            LDY #0
7200 SCHL7      LDA LREG,Y
7210            STA (VEKTOR2),Y
7230            INY
7240            CPY #2
7250            BNE SCHL7
7261 SCHL8      LDA YREG-2,Y
7262            STA (VEKTOR2),Y
7263            INY
7264            CPY #4
7265            BNE SCHL8
7266            RTS
7270 ;
7280 HOLXY      LDY #0
7310 SCHL9      LDA LREG,Y
7320            STA XREG,Y
7330            INY
7340            CPY #4
7350            BNE SCHL9
7360            RTS
7370 ;
7380 HOLLR      JSR STCKVEK
7400            LDY #0
7410 SCHL10     LDA (VEKTOR2),Y
7420            STA LREG,Y
7430            INY
7440            CPY #4
7450            BNE SCHL10
7460            RTS
7465 ;
7645 ;
7650 VEKTOR4    JSR STCKVEK
7660            CLC
7670            LDA VEKTOR2
7680            ADC #4
7690            STA VEKTOR2
7700            LDA VEKTOR2+1
7710            ADC #0
7720            STA VEKTOR2+1
7730            RTS
7735 ;
7740 PUSHXY     JSR VEKTOR4
7750            LDY #0
7760 SCHL11     LDA XREG,Y
7770            STA (VEKTOR2),Y
7780            INY
7790            CPY #4
7800            BNE SCHL11
7810            RTS
9985 ;
9990 ; REGISTER & EIN SIMULIERTER STACK
9995 ;
10000 ZREG      .BYT 0,0
10010 XREG      .BYT 0,0
10020 YREG      .BYT 0,0
10050 VERGL     .BYT 0,0
10060 LREG      .BYT 0,0
10070 RREG      .BYT 0,0
10080 VSTR      .BYT 0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0  ; 20 * 0
10100 STACK     .BYT 0
20000 .END
READY.
```

Listing 5.
Listing 4 als Quelltext (Schluß)