3.4 Quicksort (C. A. R. HOARE)

Quicksort (von engl. quick – schnell, to sort – sortieren) ist ein **schneller**, **rekursiver** Sortieralgorithmus, der nach dem Prinzip **Teile und herrsche** (lat. Divide et impera!, engl. **Divide and conquer**) arbeitet.

Er wurde ca. 1960 von **C. Antony R. Hoare** entwickelt. Der Algorithmus hat den Vorteil, dass er über eine sehr kurze innere Schleife verfügt (was die Ausführungsgeschwindigkeit stark erhöht) und ohne zusätzlichen Speicherplatz auskommt (abgesehen von dem für die Rekursion zusätzlichen benötigten Platz auf dem Aufruf-Stack).

Verfahren:

(a) Es wird ein beliebiges Listen-/Vektorelement als sogenanntes **Pivotelement** ausgewählt (das erste Element oder das letzte Element oder zufällig oder drei zufällig, davon das mittlere Elemen, o.a.)

(b) Es werden **zwei Teillisten/Teilvektoren** erzeugt:

Die **erste Teilliste/Teilvektor** erhält alle Elemente, die **kleiner oder gleich** als das **Pivotelement** sind (außer dem Pivotelement selbst).

Die **zweite Teilliste** erhält alle anderen Elemente, d.h. diejenigen, die **größer als** das **Pivoelement** sind (außer dem Pivotelement selbst).

Innerhalb der Bereiche sind die Elemente noch nicht sortiert.

(c) Jeder der **beiden Teillisten** wird wiederum mit dem **gleichen Verfahren** sortiert. Leere oder 1-elementige Listen gelten als sortiert (und brauchen nicht weiter behandelt zu werden). Die **Abbruchbedingung** ist erreicht, wenn die Dimension der **Sequenz 1** ist.

(d) Das **Gesamtergebnis** ergibt sich, indem man an die **sortierte erste Liste** das **Pivotelement** anschließt und daran dann die **sortierte zweite Liste** anhängt.

Mittlere Anzahl von Vergleichen ~ **n*ln(n)**, wobei **n** die Anzahl der Listenelemente ist

Die Schritte (a) und (b) kann man zusammen fassen:

Bei einem Durchlauf der Liste mit Hilfe von zwei gegenläufigen Zeigern (einer vom ersten Element und einer vom letzten Element beginnend) erzeugt man innerhalb des Speicherbereiches der Liste die beiden Teillisten und wählt das dazwischen stehende Element als Pivotelement.

Beispiel: Datei **QSort.in** (Textdatei, mit Editor les- und bearbeitbar):

```
a[0] a[1] a[2] a[3] a[4] a[5] a[6]  n = 7
 345   67   13  789   90  76  -45
```

Datei **QSort.out** (Textdatei, als Ergebnis des Sortierens mit QuickSort):

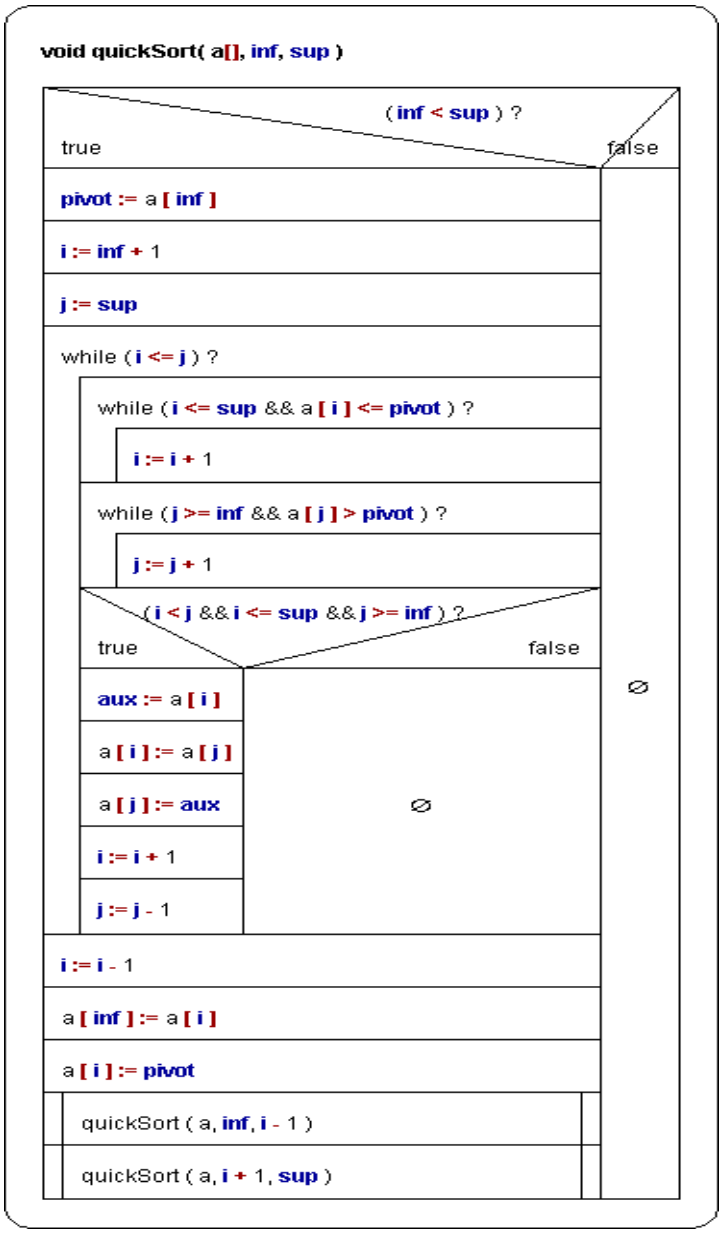
```
a[0] a[1] a[2] a[3] a[4] a[5] a[6]
 -45   13   67   76   90  345  789
```

Aufgabe: Notieren Sie auf dem Papier die Schritte für das Beispiel anhand des **Pseudocodes** auf der nächsten Seite bzw. anhand des **C++ - Codes** auf der übernächsten Seite.

Quicksort - Algorithmus als Pseudocode

```
funktion quicksort(daten, links, rechts)
     falls links < rechts dann
         teiler := teile(daten, links, rechts)
         quicksort(daten, links, teiler-1)
         quicksort(daten, teiler+1, rechts)
     ende falls
 ende

funktion teile(daten, links, rechts)
     // Starte mit i rechts vom PivotElement (PE)
     i := links + 1

     j := rechts
     pivot := daten[links]

     wiederhole solange i <= j  // solange i an j nicht
                                // vorbeigelaufen ist

      //Suche von links Element, welches größer als PE ist
         wiederhole solange i <= rechts und daten[i] <= pivot
             i := i + 1
         ende wiederhole

      //Suche von rechts Element, welches kleinergleich
      // als PE ist
         wiederhole solange j >= links und daten[j] > pivot
             j := j - 1
         ende wiederhole

         falls i < j und i<=rechts und j>=links dann
             begin
                   tausche daten[i] mit daten[j]; i++; j--;
             ende
     ende wiederhole
     i--;

  // Tausche PE mit neuer endgültiger Position (daten[i])

     tausche daten[i] mit daten[links]

  // gib die Position des Pivotelements zurück

     return i
 ende
```

Struktogramm Algorithmus QuickSort fuer C++ - Code



```
#include <fstream>              // Header-File zur Dateiarbeit
#include <vector>               // Header-File fuer vector - Template

using namespace std;

void readData(vector<int> &a){  // Funktion zum Lesen nach int-Vektor a
  int aux;
  ifstream f("QSort.in");       // Eingabestrom aus Datei QSort.in
  while(f && !f.eof() && f>>aux)// Eingabestrom exist. und nicht f.EOF
                                // und fehlerfrei Lesen von f nach aux ?
    a.push_back(aux);           // fuegt Wert von aux an Ende des Vektors an
}

void quickSort(vector<int> &a, int inf, int sup){ // Parameter a, inf, sup
  if(inf<sup){                // Falls linker Index inf kleiner als rechter Index sup
    int pivot = a[inf], aux;       // pivot := linkes Element a[inf]
    int i = inf + 1, j = sup;      // Beginne mit i: = inf+1 und j: = sup
    while(i<=j){            // Wiederhole, solange linker Index i <= rechter Index j
      while(i<=sup && a[i] <= pivot) i++; // suche von links El., welches
                                          // groesser als Pivotelement ist
      while(j>=inf && a[j]  > pivot) j--; // suche von rechts El., welches
                                          // kleinergleich als Pivotel. ist
      if(i<j && i<=sup && j>=inf){        // Falls Index i des linken El. kleiner
                                          // als Index j des rechten Elementes
                                          // und Indexe i und j zulaessig, dann
        aux = a[i];                       // vertausche a[i] mit a[j]
        a[i] = a[j];
        a[j] = aux;
        i++; j--;                         // Erhoehe i um 1, verringere j um 1
      }
    }                         // Ende while(i<=j), nach while(i<=j){...} gilt i > j
    i--;                  // i:=i-1, damit gilt i == j, a[i] wird neues Trennelement

    a[inf] = a[i]; a[i] = pivot;   // vertausche a[inf] == pivot mit a[i]

    quickSort(a, inf, i-1); // rekursiver Aufruf fuer den linken Bereich
                            // a[inf] ... a[i-1] von a
                            // a[i] ist Trennelement
    quickSort(a, i+1, sup); // rekursiver Aufruf fuer den rechten Bereich
                            // a[i+1] ... a[sup] von a
  }                         // Ende von if(inf<sup), d.h. Ende wegen
}                           // inf >= sup (einelementiger Bereich von a)

void write(vector<int> a){              // Funktion zum Schreiben nach int-Vektor a
  ofstream f("QSort.out");              // Ausgabestrom nach Datei QSort.out
  for(int i=0; i<(int)a.size(); i++)    // alle a[i] durchlaufen und in
        f << a[i] << " ";               // Datei schreiben, die mit f verbunden
}

int main(){
  vector<int> a;                        // int-Vektor a vereinbaren
  readData(a);                          // Aufruf readData(a) zum Lesen in Vektor a

  quickSort(a, 0, (int)a.size()-1);     // Aufruf Quicksort fuer Vektor a von
                                        // a[0] bis a[a.size()-1]
  write(a);                             // Aufruf von write(a) zum Schreiben
  return 0;                             //  vektor a in File
}
```

Aufgabe:
Modifizieren Sie das obige **C++ - Programm** bzw. den **Pseudocode** so, daß das **Pivot**-Element **zufällig** aus dem Intervall genommen wird.