14. November 2017

Such- und Sortieralgorithmen

Dargestellt sind Lösungen für wichtige Such- und Sortierverfahren so, wie sie im Unterricht besprochen wurden. Die Implementierungen sind von Schülern erarbeitet.

1. Suchen in linearen Datensammlungen

1.1 Lineare Suche: Liegt eine unsortierte Sammlung von Datenobjekten vor, dann kann man nur sequenziell die Sammlung von vorne beginnend durchsuchen. Der Vergleich von jedem Element der Sammlung mit dem zu suchenden erfordert im Mittel n/2, (n - Anzahl der zu durchsuchenden Elemente) Vergleichsoperationen.

```
1 public static bool linSearch(int gesucht, int[] data) {
2   for (int i = 0; i < data.Length; i++) {
3     if (data[i] == gesucht) {
4       return true;
5     }
6   }
7   return false;
8 }
```

1.2 Binäre Suche: Ist die Sammlung allerdings schon sortiert, dann kann man unter Ausnutzung dieser Eigenschaft die binäre Suche verwenden. Der Aufwand verringert sich aufgrund des Prinzips *Teile und Herrsche* auf $\log_2 n$.

```
1 class Program {
2   static void Main(string[] args) {
3     Console.WriteLine("Geben␣Sie␣eine␣Zahl␣zwischen␣1␣und␣100'000
          ␣␣ein.");
4     int eingabe = Convert.ToInt32(Console.ReadLine());
5     Console.WriteLine();
6
7     int laenge = 100000;
8     int[] a = new int[laenge];
9     for (int z = 0; z < laenge; z++) {
10      a[z] = (z+ 1);
11    }
12    vorhanden(a, eingabe);
13    Console.ReadLine();
```

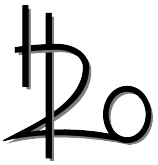
```
14    }
15
16    static bool vorhanden(int[] a, int suche) {
17      int noetigeversuche = 0;
18      int first = 0;
19      int last = a.Length - 1;
20      teste:
21      int mitte = (first + last) / 2;
22      if (first - last == 0 && a[mitte] != suche) {
23        Console.WriteLine("Die␣gesuchte␣Zahl␣befindet␣sich␣nicht␣in
            ␣dem␣Bereich!");
24        return false;
25      }
26      if (a[mitte] == suche) {
27        Console.WriteLine("Die␣gesuchte␣Zahl␣befindet␣sich␣an␣
            folgender␣Stelle:␣{0}", mitte + 1);
28        Console.WriteLine("Dafuer␣waren␣nur␣{0}␣Versuche␣noetig!",
            noetigeversuche);
29        return true;
30      }
31      if (a[mitte] > suche) {
32        last = mitte - 1;
33        Console.WriteLine("Erste␣Zahl␣des␣Bereiches,␣der␣durchsucht
            ␣wird:␣{0}", a[first]);
34        Console.WriteLine("Letzte␣Zahl␣des␣Bereiches,␣der␣
            durchsucht␣wird:␣{0}", a[last]);
35        noetigeversuche += 1;
36        Console.ReadLine();
37        goto teste;
38      }
39      if (a[mitte] < suche) {
40        first = mitte + 1;
41        Console.WriteLine("Erste␣Zahl␣des␣Bereiches,␣der␣durchsucht
            ␣wird:␣{0}", a[first]);
42        Console.WriteLine("Letzte␣Zahl␣des␣Bereiches,␣der␣
            durchsucht␣wird:␣{0}", a[last]);
43        noetigeversuche += 1;
44        Console.ReadLine();
45        goto teste;
46      }
47      return false;
48    }
49  }
```

2. Sortierverfahren

2.1 Einfache Sortierverfahren: Sollen eine Menge von Datenobjekten mit einer ordinalen Eigenschaft sortiert werden, so kann der *Bubblesort, Insertionsort* oder der *Selectionsort* verwendet werden. Alle Verfahren kennzeichnet ein grosser Aufwand, da jedes Element mit jedem anderen der Sammlung verglichen werden muss, um so die richtige Reihenfolge zu realisieren.

2.1.1 Selectionsort

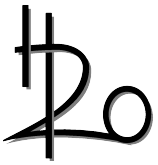
```
1  using System;
2  using System.Collections;
3  using System.Diagnostics;
4
5  namespace ConsoleApplication1 {
6    class Program {
7      static void Main(string[] args) {
8        Console.WriteLine("Wie viele Elemente soll es geben?");
9        int anzahl = Convert.ToInt32(Console.ReadLine());
10
11       ArrayList text = new ArrayList();
12       zuordnen(text,anzahl);
13
14       Stopwatch watch = new Stopwatch();
15       Console.WriteLine("Das Zuweisen von zufaelligen Zahlen
             ist beendet. Enter fuer Sortieren!");
16       Console.ReadLine();
17
18       watch.Start();
19       text = selectionsort(text);
20       watch.Stop();
21
22       Console.WriteLine();
23       Console.WriteLine("Das Sortieren dauerte 'nur': {0}",
             watch.Elapsed);
24       Console.WriteLine("Enter druecken fuer die Ausgabe.");
25       Console.ReadLine();
26
27       ausgabe(text);
28       Console.ReadLine();
29     }
30
31     static ArrayList zuordnen(ArrayList a,int anzahl) {
```



```
32        Random rn = new Random();
33        for (int z = 0; z < anzahl; z++) {
34          a.Add(rn.Next(10000));
35        }
36        return a;
37      }
38
39      static void ausgabe(ArrayList a) {
40        for (int z = 0; z < a.Count; z++) {
41          Console.WriteLine("Das␣{0}.te␣Element␣ist:␣{1}", z +
                1, a[z]);
42        }
43      }
44
45      static ArrayList selectionsort(ArrayList a) {
46        for (int laenge = a.Count; laenge > 0; laenge--) {
47          int element = 0;
48          int stelle = 0;
49          for (int z = 0; z < laenge; z++) {
50            if (Convert.ToInt32(a[z]) > element) {
51              element = Convert.ToInt32(a[z]);
52              stelle = z;
53            }
54          }
55          a[stelle] = a[laenge - 1];
56          a[laenge - 1] = element;
57        }
58        return a;
59      }
60    }
61  }
```

2.1.2 Insertionsort

```
1  // Insertionsort mit C#
2
3  using System;
4
5  public class Insertionsort {
6  //Sortieren mit insertionsort
7
8    static int listenlaenge = 10;
9    static int[] liste = {11,4,33,6,34,55,23,87,22,34};
10   static int[] neueliste = new int[listenlaenge];
11   static int zaehler = 0;
```

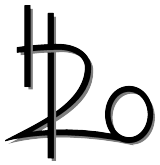


```
12
13    public static void sortieren() {
14      for (int j = 0; j < liste.Length; j++) {
15        insert (liste[j]);
16        ausgeben(neueliste);
17      }
18    }
19
20  //insert
21
22    public static void insert (int zahl) {
23      int i = 0;
24      while (i != zaehler) {
25        if (neueliste[i] <= zahl) {
26          i++;
27        }
28        else {
29          int hilf = neueliste[i];
30          neueliste[i] = zahl;
31          zahl = hilf;
32          i++;
33        }
34      }
35      neueliste[zaehler] = zahl;
36      zaehler++;
37    }
38
39    public static void ausgeben(int[] l) {
40    //Ausgabe der Liste
41      Console.Write("Deine␣Liste␣lautet:␣");
42      for (int i = 0; i < l.Length; i++) {
43        Console.Write("{0:d}␣",l[i]);
44      }
45      Console.WriteLine();
46    }
47
48    public static void Main(String[] args) {
49      //Eingabe der Liste
50      /*for (int i = 0; i < listenlaenge; i++) {
51          Console.Write("Bitte zahlen eingeben: ",i + 1);
52          liste[i] = int.Parse(Console.ReadLine());
53          }
54        */
```

```
55      Console.Write ("Folgende␣Liste␣wird␣sortiert:␣");
56      ausgeben(liste);
57      Console.WriteLine ();
58      Console.WriteLine ("Nun␣wird␣schrittweise␣sortiert.");
59      sortieren();
60      Console.WriteLine ();
61      Console.Write ("Die␣sortierte␣Liste␣lautet:␣");
62      ausgeben(neueliste);
63    }
64  }
```

2.1.3 Bubblesort

```
1  using System;
2
3  namespace BubbleSort {
4    class Program {
5      static void Main(string[] args) {
6        BubbleSort();
7        Console.ReadKey();
8      }
9
10     static void BubbleSort() {
11       Random randomInt = new Random();
12       Console.WriteLine("\nPlease␣insert␣the␣length␣of␣the␣
             array:␣");
13       int length = Convert.ToInt32(Console.ReadLine());
14       int help = 0;
15       int[] array = new int[length];
16
17       for (int i = 0; i < length; i++) {
18         array[i] = randomInt.Next(-100, 100);
19         Console.Write(array[i]);
20         if (i != length - 1) {
21           Console.Write(",␣");
22         }
23       }
24
25       for (int n = array.Length; n > 1; n = n - 1) {
26         for (int i = 0; i < n - 1; i = i + 1) {
27           if (array[i] > array[i + 1]) {
28             help = array[i];
29             array[i] = array[i + 1];
30             array[i + 1] = help;
31           }
```

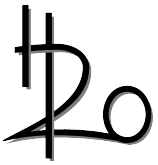


```
32          }
33        }
34
35        Console.WriteLine("\nSorted:␣");
36        for (int i = 0; i < length; i++) {
37          Console.Write(array[i]);
38          if (i != length - 1) {
39            Console.Write(",␣");
40          }
41        }
42      }
43    }
44  }
```

2.2 Höhere Sortierverfahren: Die höheren Sortierverfahren machen von dem Prinzip des *Teile und Herrsche* Gebrauch. Dadurch wird der Aufwand in den meisten Fällen stark reduziert. Der Aufwand steigt nicht mehr quadratisch mit der Menge der Eingabewerte, sondern nur noch logarithmisch. Der Quicksort birgt allerdings die Gefahr bei ungünstiger Wahl des Pivotelementes zu entarten und wieder mehr Aufwand zu benötigen.

2.2.1 Quicksort

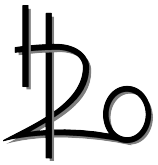
```
1  static ArrayList quicksort(ArrayList untere) {
2    ArrayList obere = new ArrayList();
3    ArrayList gleich = new ArrayList();
4    if (untere.Count <= 1) {
5      return untere;
6    }
7    if (untere.Count > 1) {
8      int vergleich = Convert.ToInt32(untere[0]);
9      for (int z = 0; z < untere.Count; z++) {
10       if (Convert.ToInt32(untere[z]) == vergleich) {// x.tes
             Element entspricht Vergleichselement
11         gleich.Add(untere[z]);
12         untere.RemoveAt(z);
13         z -= 1;
14       }
15     }
16     for (int z = 0; z < untere.Count; z++) {
17       if (Convert.ToInt32(untere[z]) > vergleich)  {// x.tes
             Element ist groesser als Vergleichselement
18         obere.Add(untere[z]);
19         untere.RemoveAt(z);
```



```
20          z -= 1;
21        }
22      }
23  //Nun die Rekursion einbauen
24      untere = quicksort(untere);
25      obere = quicksort(obere);
26
27  //die Einzelteile wieder zu einem Ganzen zusammengef{\"u}gt
28      for (int z = 0; z < gleich.Count; z++) {  // die Elemente
            aus "gleich" werden an "untere" angehangen
29        untere.Add(gleich[z]);
30      }
31      for (int z = 0; z < obere.Count; z++) { // das gleiche nun
            mit den Elementen aus dem jeweiligen Array "obere"
32        untere.Add(obere[z]);
33      }
34    }
35    return untere;
36  }
```

2.2.1 Mergesort

```
1  using System.Collections;
2
3  namespace Mergesort {
4    class Program {
5      static ArrayList MergeSort(ArrayList unsort) {
6        if (unsort.Count > 1) {
7          ArrayList ListeL = new ArrayList();
8          ArrayList ListeR = new ArrayList();
9          for (int i = 0; i <= (unsort.Count) / 2 - 1; i++)
10         { ListeL.Add(unsort[i]); }
11         for (int i = unsort.Count / 2; i < unsort.Count; i++)
12         { ListeR.Add(unsort[i]); }
13         return (Merge(MergeSort(ListeL), MergeSort(ListeR)));
14       }
15       else
16       { return unsort; }
17     }
18
19     static ArrayList Merge(ArrayList ListeL, ArrayList ListeR)
           {
20       ArrayList MischList = new ArrayList();
21       while (ListeL.Count != 0 && ListeR.Count != 0) {
```



```
22          if (Convert.ToInt32(ListeL[0]) <= Convert.ToInt32(
                ListeR[0]))
23          { MischList.Add(ListeL[0]); ListeL.RemoveAt(0); }
24          else
25          { MischList.Add(ListeR[0]); ListeR.RemoveAt(0); }
26        }
27        for (int i = 0; i < ListeL.Count; i++)
28        { MischList.Add(ListeL[i]); }
29        for (int i = 0; i < ListeR.Count; i++)
30        { MischList.Add(ListeR[i]); }
31
32        return MischList;
33      }
34      static void Main(string[] args) {
35        ArrayList Zahlen = new ArrayList();
36        Zahlen.Add(1);
37        Zahlen.Add(23);
38        Zahlen.Add(11);
39        Zahlen.Add(21);
40        Zahlen.Add(31);
41
42        Console.Write ("Das␣sortierte␣Array␣lautet:␣");
43        foreach (int zahl in MergeSort(Zahlen))
44          Console.Write("{0},␣", zahl);
45        Console.ReadKey ();
46      }
47    }
48  }
```