

Laufzeitanalyse und Sortieralgorithmen

Laufzeitanalyse

```
public static void laufzeit() {  
    int x = Convert.ToInt32(Console.ReadLine());  
    for (int i = 1; i < x; i++) {  
        Console.WriteLine(x);  
    }  
}  
}
```

Komplexität – Was ist das?

Wie komplex ist ein Problem?

- ▶ So komplex wie der bestmögliche das Problem lösende Algorithmus.
- ▶ **Dadurch:** Abstraktion von programmiertechnischen Eigenheiten.

Komplexität – Was ist das?

Wie komplex ist ein Problem?

- ▶ So komplex wie der **bestmögliche** das Problem lösende Algorithmus.
- ▶ **Dadurch:** Abstraktion von programmiertechnischen Eigenheiten.

Wie wird Komplexität gemessen?

- ▶ **Zeit:** Zählen möglichst **elementarer Operationen**.
- ▶ **Platz:** Zählen der **benötigten Speicherplätze**.
- ▶ **Dadurch:** Abstraktion von **physikalischen Gegebenheiten**.

Laufzeiten analysieren

```
[4] bool CS$4$0000)
IL_0000: nop
IL_0001: ldstr      "Hello World!"
IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
IL_000b: nop
IL_000c: ldc.i4.7
IL_000d: stloc.0
IL_000e: ldc.i4.8
IL_000f: stloc.1
IL_0010: ldloc.0
IL_0011: ldloc.1
IL_0012: add.ovf
IL_0013: stloc.2
IL_0014: ldc.i4.0
IL_0015: stloc.3
IL_0016: br.s       IL_0022
IL_0018: nop
IL_0019: ldloc.2
IL_001a: ldloc.3
IL_001b: add.ovf
IL_001c: stloc.2
IL_001d: nop
IL_001e: ldloc.3
IL_001f: ldc.i4.1
IL_0020: add.ovf
IL_0021: stloc.3
IL_0022: ldloc.3
IL_0023: ldc.i4.s   10
IL_0025: clt
IL_0027: stloc.s   CS$4$0000
IL_0029: ldloc.s   CS$4$0000
IL_002b: brtrue.s  IL_0018
IL_002d: ldc.i4.1
```

```
using System;

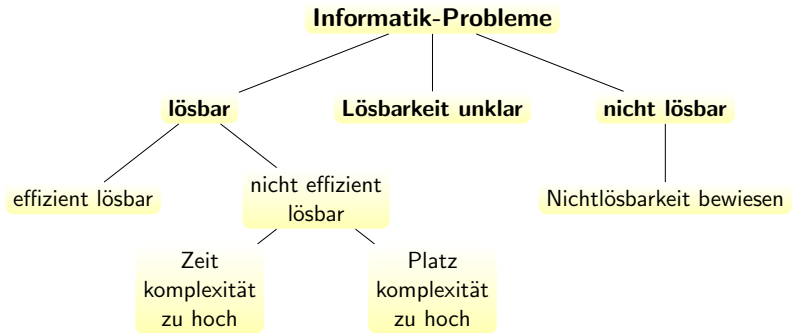
namespace Test
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");

            int x=7;
            int y=8;
            int erg = x+y;

            for(int i=0;i<10;i++)
            {
                erg=erg+i;
            }

            Console.ReadKey(true);
        }
    }
}
```

Probleme der Informatik

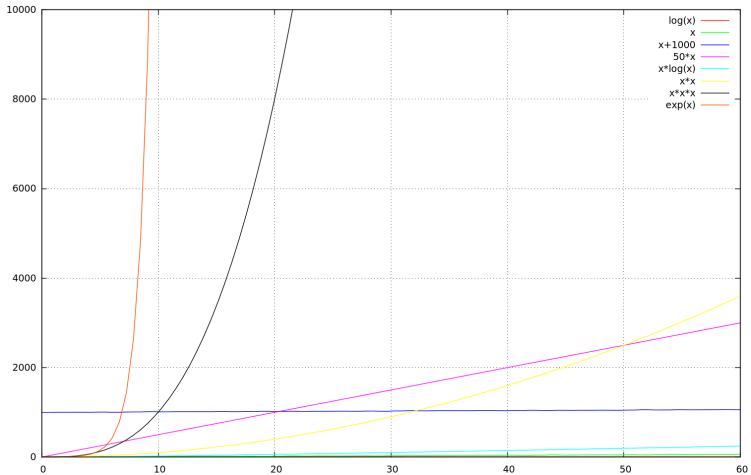


Laufzeiten

gut programmierbar, gut einsetzbar in E2 und Q1

- ▶ $O(n)$ - Suche kleinstes Element
- ▶ $O(n^2)$ - Quadrat, Dreieck zeichnen
- ▶ $O(n^3)$ - 100 Cent sollen mit 1,2 und 5 Cent Stücken bezahlt werden → wieviel Möglichkeiten gibt es?
- ▶ $O(2^n)$ - rekursive Variante von Fibonacci

Übersicht Laufzeiten



Kerncurriculum

E.3 Grundlagen der Programmierung

Konsolenprogramme allein werden den Anforderungen an einen allgemeinbildenden Informatikunterricht nicht gerecht. Die Programmiersprache wird als Mittel zum Zweck verstanden und steht nicht im Vordergrund des Unterrichts. Neue Sprachelemente werden nur dann eingeführt, wenn sie zur Implementierung erforderlich sind. Es wird eine objektorientierte Programmiersprache eingesetzt, aber die objektorientierte Modellierung in der Einführungsphase nicht thematisiert.

- grafische Benutzeroberflächen und ereignisgesteuerte Programmierung:
Fenster, Label, Textfeld und Button
- einfache Datentypen:
- Variablen, Operationen, logische Ausdrücke, Typkonvertierungen
- Modellierung und Implementierung einfacher Algorithmen bezogen auf die genannten Kontexte:
Anweisung, Kontrollstrukturen, Struktogramme
- strukturierte Datentypen mit Operationen und Relationen:
Zeichenkette (String), Feld (Array)
- Modularisierung:
Funktionen, Prozeduren, Parameter

Kerncurriculum

Q1.2 Rekursion

grundlegendes Niveau (Grundkurs und Leistungskurs)

- Rekursion:
rekursive Grafiken, mathematische Funktionen, „teile und herrsche“-Prinzip, Grundstrukturen für die Implementierung: Terminationsbedingung, Parameterübergabe, einfache und mehrfache Rekursion, Visualisierung
- Rekursion versus Iteration:
Vor- und Nachteile rekursiver Algorithmen gegenüber iterativen Algorithmen

erhöhtes Niveau (Leistungskurs)

- Backtrackingverfahren

Sortieren

- Eingabe ist ein Feld mit Elementen. Das Feld soll sortiert zurückgegeben werden.

Kerncurriculum

Q1.1 Such- und Sortieralgorithmen

grundlegendes Niveau (Grundkurs und Leistungskurs)

- grundlegende Algorithmen:
lineare und binäre Suche, einfache Sortieralgorithmen mit quadratischer Laufzeit, Analyse und Bewertung von Sortieralgorithmen unter dem Aspekt Laufzeit

erhöhtes Niveau (Leistungskurs)

- effiziente Algorithmen:
ein effizienter Sortieralgorithmus

Laufzeitbetrachtung

- ▶ Worst-Case:
ungünstigster Fall – Maximum aller möglichen Laufzeiten bei einer Eingabe fester Länge n
- ▶ Average-Case:
mittlerer Fall, Durchschnitt – arithmetisches Mittel aller möglichen Laufzeiten bei einer Eingabe fester Länge n
- ▶ Best-Case:
günstigster Fall – Minimum aller möglichen Laufzeiten bei einer Eingabe fester Länge n

Warum Sortieren im Unterricht?

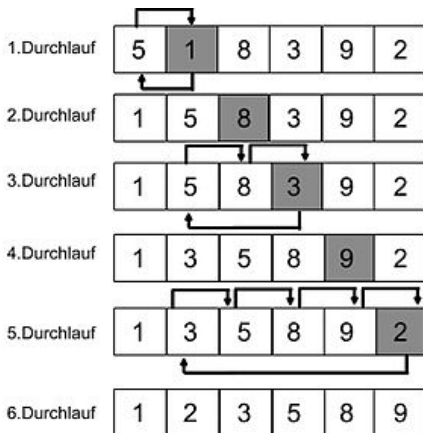
- ▶ Worst-Case, Best-Case Szenarien können wunderbar thematisiert werden.
- ▶ Das Problem des Sortierens hat eine bewiesene minimale Komplexität von $n \cdot \log n$.
- ▶ Es existieren viele verschiedene Algorithmen mit unterschiedlichen Anwendungsbereichen und unterschiedlichen Grundideen.
- ▶ Arrays und Schleifen können auf einem höheren Niveau geübt werden.

Einfache Sortiervverfahren – Insertion Sort

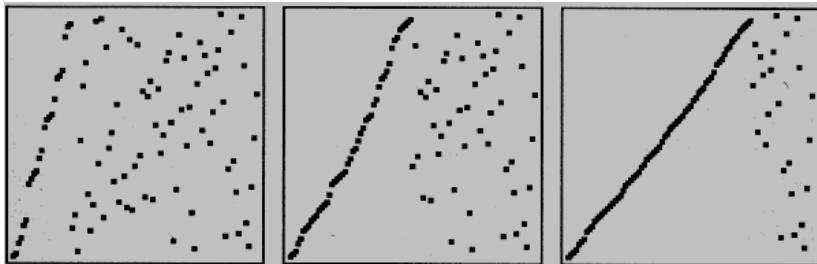
Sortieren durch Einfügen (Insertion Sort)

- ▶ Betrachte die Elemente eines nach dem anderen und füge jedes an seinen richtigen Platz zwischen den bereits betrachteten ein.
- ▶ Das gerade betrachtete Element wird eingefügt, indem die größeren Elemente um eins nach rechts bewegt werden und das Element dann an den freigewordenen Platz eingefügt wird.

Insertion Sort



Insertion Sort



Einfache Sortierverfahren – Insertion Sort

```
public static int[] insertionSort(int[] sortieren) {  
    int temp;  
    for (int i = 1; i < sortieren.length; i++) {  
        temp = sortieren[i];  
        int j = i;  
        while (j > 0 && sortieren[j - 1] > temp) {  
            sortieren[j] = sortieren[j - 1];  
            j--;  
        }  
        sortieren[j] = temp;  
    }  
    return sortieren;  
}
```

Einfache Sortiervverfahren – Selection Sort

Sortieren durch Auswählen (Selection Sort)

- ▶ Finde zuerst das kleinste Element und tausche es gegen das an erster Stelle befindliche Element.
- ▶ Finde danach das zweitkleinste Element und tausche es gegen das an zweiter Stelle befindliche Element.
- ▶ Fahre in dieser Weise fort bis das komplette Feld sortiert ist.

Selection Sort

(a)

7	2	8	1	4
---	---	---	---	---

(b)

1	2	8	7	4
---	---	---	---	---

(c)

1	2	8	7	4
---	---	---	---	---

(d)

1	2	4	7	8
---	---	---	---	---

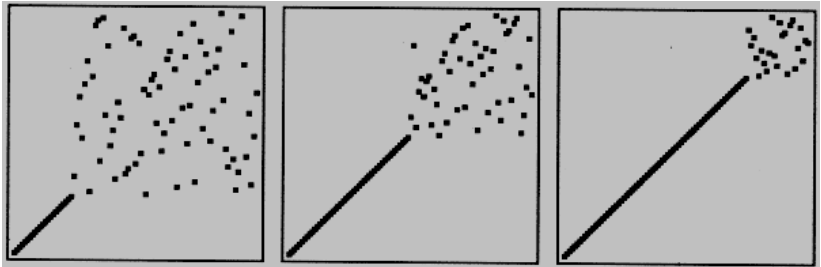
(e)

1	2	4	7	8
---	---	---	---	---

(f)

1	2	4	7	8
---	---	---	---	---

Selection Sort



Einfache Sortierverfahren – Selection Sort

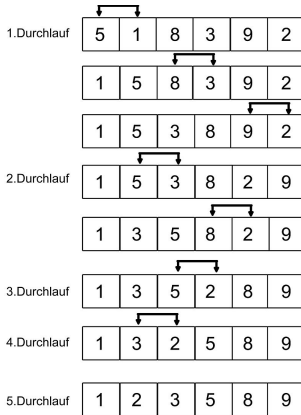
```
public static int[] selectionsort(int[] sortieren) {  
    int k;  
    for (int i = 0; i < sortieren.length - 1; i++) {  
        k=i;  
        for (int j = i + 1; j < sortieren.length; j++) {  
            if (sortieren[k] > sortieren[j]) {  
                k=j  
            }  
        }  
        int temp = sortieren[i];  
        sortieren[i] = sortieren[k];  
        sortieren[k] = temp;  
    }  
  
    return sortieren;  
}
```

Einfache Sortiervverfahren – Bubble Sort

Sortieren durch Tauschen (Bubble Sort)

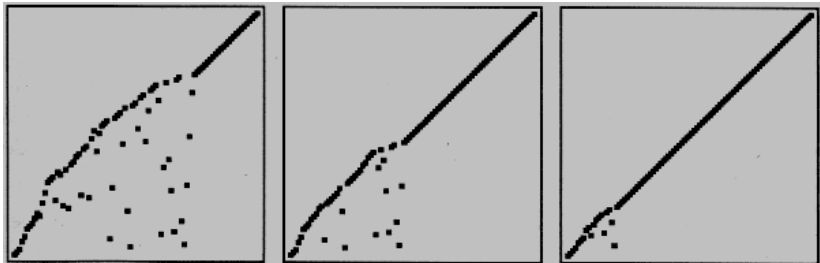
- ▶ Durchlaufe das Array immer wieder und vertausche jedesmal, wenn es notwendig ist, benachbarte Elemente.
- ▶ Wenn bei einem Durchlauf kein Austausch mehr erforderlich ist, ist das Feld sortiert.

Bubble Sort



Abbruch, da keine Vertauschungen
mehr auftreten.

Bubble Sort



Einfache Sortiervverfahren – Bubble Sort

```
public static int[] bubblesort(int[] sortieren) {  
    int temp;  
    for(int i=1; i<sortieren.length; i++) {  
        for(int j=0; j<sortieren.length-i; j++) {  
            if(sortieren[j]>sortieren[j+1]) {  
                temp=sortieren[j];  
                sortieren[j]=sortieren[j+1];  
                sortieren[j+1]=temp;  
            }  
        }  
    }  
    return sortieren;  
}
```

Übersicht Sortierverfahren

Sortierverfahren

- ▶ Einfache Verfahren (n^2)
 - ▶ Insertion Sort
 - ▶ Selection Sort
 - ▶ Bubble Sort
- ▶ Erweiterte Verfahren ($n \cdot \log n$)
 - ▶ Quick Sort
 - ▶ Merge Sort
 - ▶ Heap Sort

Worst-Case und Best-Case Szenarien

Worst-Case für alle drei Verfahren

76	51	49	37	34	21	18	16	4	1
----	----	----	----	----	----	----	----	---	---

Best-Case für Insertion Sort (mindestens 1 Vertauschung)

4	1	16	18	21	34	37	49	51	76
---	---	----	----	----	----	----	----	----	----

Worst-Case und Best-Case Szenarien

Worst-Case für alle drei Verfahren

76	51	49	37	34	21	18	16	4	1
----	----	----	----	----	----	----	----	---	---

Best-Case für Selection Sort (mindestens 1 Vertauschung)

49	4	16	18	21	34	37	1	51	76
----	---	----	----	----	----	----	---	----	----

Worst-Case und Best-Case Szenarien

Worst-Case für alle drei Verfahren

76	51	49	37	34	21	18	16	4	1
----	----	----	----	----	----	----	----	---	---

Best-Case für Bubble Sort (mindestens 1 Vertauschung)

49	1	4	16	18	21	34	37	51	76
----	---	---	----	----	----	----	----	----	----

Worst-Case und Best-Case Szenarien

Worst-Case für alle drei Verfahren

76	51	49	37	34	21	18	16	4	1
----	----	----	----	----	----	----	----	---	---

Best-Case für Bubble Sort (mindestens 1 Vertauschung)

49	1	4	16	18	21	34	37	51	76
----	---	---	----	----	----	----	----	----	----

Szenario: Neuer Telefonbucheintrag bei der Telekom

37	1	4	5	18	21	40	42	47	51
----	---	---	---	----	----	----	----	----	----

Sortieren im Unterricht

Was muss man mit SuS üben, bevor man Sortieralgorithmen behandelt?

Sortieren im Unterricht

Was muss man mit SuS üben, bevor man Sortieralgorithmen behandelt?

- ▶ Vergleichen im Array
- ▶ Vertauschen von Elementen im Array
- ▶ Verschieben von Elementen im Array