Exkurs
Komplexitätstheorie

Komplexität

- **Ziele:** Charakterisierung von Algorithmen bzgl. *Speicherbedarf* und *Rechenzeit*
  - unabhängig von konkreter Hardware
  - abstraktes Rechnermodell, z.B.
    - Turingmaschine
    - sequentielle Registermaschine mit definierten Grundoperationen
- **Abstraktion:** Unterschiedliche Rechnermodelle, z.B.
  - Einadress- und Dreiadressregistermaschinen

  unterscheiden sich in der Laufzeit i.d.R. nur um einen konstanten Faktor, der i.Allg. unberücksichtigt bleibt.

- ***Informelle Definition der Komplexität eines Algorithmus***
  - *Zeitkomplexität* (*time complexity*) = **Anzahl** der Elementaroperationen z.B. +, –, *, /, mod, sqrt, shift, OR, NOT, :=, =, <, ≥...,
    als **Funktion der Größe der Eingabeparameter**, z.B.
    - Anzahl der zu sortierenden Elemente oder
    - der Bits in der Darstellung der Zahlen bei der Potenzberechnung
  - *Speicherkomplexität* (*space complexity*) = Anzahl der benötigten Elementarvariablen.

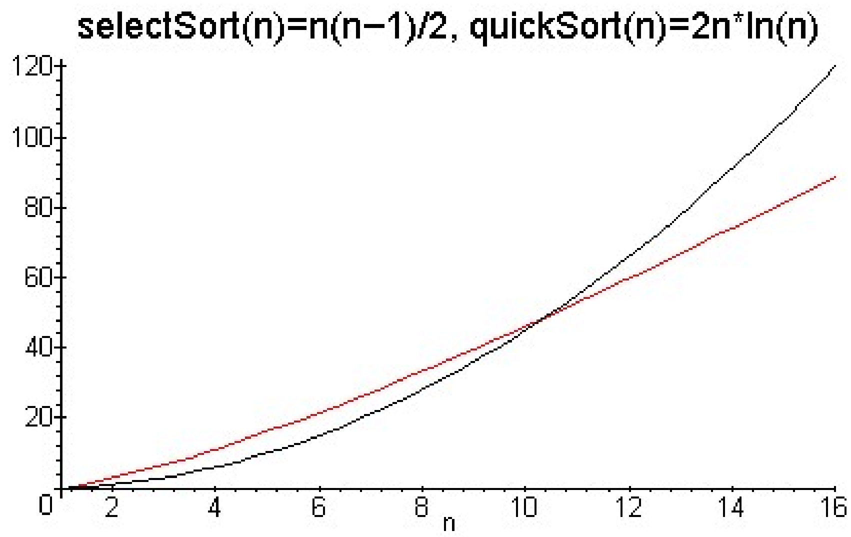
Beispiel: Komplexität des Sortierens durch direkte Auswahl

```
typedef float T;
void selectSort(/*inout*/ T a[], const int n /*length of a*/) {
   T x;
   for (int i=0; i<n-1; i++) {  // Invariant:       a = permutation of old a and
                                //                  a[0]...a[i-1] sorted and
                                //                  a[0]...a[i-1] <= a[i]...a[n-1]
     int k=i;                   // find the position k of the smallest element
     for (int j=i+1;j<n;j++)    // a[k] = min(a[i],...,a[j-1])
       if (a[j]<a[k])           //  a[j] = min(a[i],...,a[j])
         k=j;                   // a[k] = min(a[i],...,a[j])
     // swap a[i] and a[k]      // a[k] = min(a[i],...,a[j-1]) and j = n
     x=a[k]; a[k]=a[i]; a[i]=x; // a = permutation of old a and a[0]...a[i] sorted
                                // and a[0]...a[i] <= a[i+1]...a[n-1]
   }                            // a = permutation of old a and a[0]..a[n-1] sorted
} // selectSort
```

- **Speicherkomplexität** von `selectSort` := Anzahl der unstrukturierten Variablen = 1+ 3 + 1.
- **Zeitkomplexität** := Anzahl der Vergleiche von Daten (ohne Schleifenkontrolle) = n-1 + n-2 +...+ 1 = ½·n·(n-1). Anzahl der swap-Operationen: n-1

Komplexität von SelectSort (Forts.)

```
#include <iostream>
using namespace std;
int main(void) {
   const int n=10;
   T a[n];
   cout <<"Enter "<<n<<" numbers: ";
   for (int i=0; i<n; i++)
         cin >> a[i];
   selectSort(a,n);
   cout << "Sorted numbers:  ";
   for (int i=0; i<n; i++)
         cout<<' '<<a[i];
}
```



- **Abstraktion:** nur Größenordnung interessant
  konstante (maschinenabhängige) Faktoren unberücksichtigt, z.B.
  - `selectSort` benötigt ½ n^2 - ½ n Vergleiche
  - Quicksort benötigt im Mittel nur $2 \cdot n \cdot \log(n)$ Vergleiche
    (und einen Stack von der Größenordnung log(n))
  - n > 10: Quicksort schneller,
  - $n \leq 10$: Quicksort langsamer,
    Werte für große n entscheidend $\Rightarrow$ von den Faktoren abstrahieren:
    Selectsort braucht **$O(n^2)$**, Quicksort **$O(n \cdot \log(n))$** Vergleiche.

Landau-Symbole “Groß-O”, “-Omega” und “-Theta”

Definition:

- $c:\mathbb{N}\to\mathbb{R}^+$ (oft $c:\mathbb{N}\to\mathbb{N}$) monoton steigend, heißt *Komplexitätsfunktion*
- Vergleichsrelationen auf der Menge *C* der Komplexitätsfunktionen:

Groß-O, Groß-Omega
$c' \preccurlyeq c \quad :\Leftrightarrow c'(n)/c(n)$ ist nach oben beschränkt $\Leftrightarrow \exists\, a \in \mathbb{R}^+, N \in \mathbb{N} : c'(n) \le a\cdot c(n)\ \forall\, n > N$
$O(c) = \{c': \mathbb{N}\to\mathbb{R}^+ \mid c' \preccurlyeq c\}$, $\Omega(c) = \{c': \mathbb{N}\to\mathbb{R}^+ \mid c \preccurlyeq c'\}$
Schreibweise: $c' = O(c)$ oder $c'(n) = O(c(n))$ statt $c' \in O(c)$ oder $c'(n) \in O(c(n))$
Sprechweise: c’ höchstens die Komplexität von c

c obere Komplexitäts-schranke für die Funktionen in $O(c)$

c untere Komplexitäts-schranke für die Funktionen in $\Omega(c)$

klein-o, klein-Omega
$c' \prec c \quad :\Leftrightarrow \lim c'(n)/c(n) = 0$
$o(c) = \{c': \mathbb{N}\to \mathbb{R}^+ \mid c' \prec c\}$, $\omega(c) = \{c': \mathbb{N}\to \mathbb{R}^+ \mid c \prec c'\}$
Schreibweise: $c' = o(c)$ oder $c'(n) = o(c(n))$ statt $c' \in o(c)$ oder $c'(n) \in o(c(n))$
Sprechweise: c’ hat geringere Komplexität als c

Landau-Symbole (Forts.)

- **Äquivalenzklassen von Komplexitätsfunktionen:**
  - c und c' heißen *von gleicher Komplexität*, wenn sie sich "im Wesentlichen" nur um einen konstanten Faktor unterscheiden; genauer:

    $c' \sim c :\Leftrightarrow c' \preccurlyeq c$ und $c \preccurlyeq c'$

    $\Leftrightarrow \exists\, a,b \in \mathbb{R}^+, N \in \mathbb{N}: b \cdot c(n) \leq c'(n) \leq a \cdot c(n)\ \forall\, n > N$

    $\Leftrightarrow \exists\, a \in \mathbb{R}^+, N \in \mathbb{N}: 1/a \cdot c(n) \leq c'(n) \leq a \cdot c(n)\ \forall\, n > N.$
  - Die Äquivalenzklassen bezeichnet man mit

    $\Theta(c) = \{c':\mathbb{N} \to \mathbb{R}^+ \mid c' \preccurlyeq c \wedge c \preccurlyeq c'\}$
    $= \{c':\mathbb{N} \to \mathbb{R}^+ \mid \exists\, a \in \mathbb{R}^+, N \in \mathbb{N}: 1/a \cdot c(n) \leq c'(n) \leq a \cdot c(n)\ \forall\, n > N\}$

Landau-Symbole (Forts.)

- **Rechenregeln:**
  - $a \cdot c(n) \in O(c(n))\ \forall a \in \mathbb{R}^+$ konstante Faktoren ändern Komplexität nicht
  - $c' \preccurlyeq c \Rightarrow O(c' + c) = O(c)$ größere Komplexität dominiert
    insbesondere $c' \in O(x^n)$, $c \in O(x^m) \Rightarrow c' + c \in O(x^{\max(n,m)})$
  - $c' \in O(x^n)$, $c \in O(x^m) \Rightarrow c' \cdot c \in O(x^{n+m})$ für $c: \mathbb{N} \to \mathbb{N}$

- **Beispiele:**
  - $O(\ln(n)) = O(\log(n))$
    da sich Logarithmen nur um einen konstanten Faktor unterscheiden
  - $c(n) = 3 \cdot n^2 + n \cdot \log(n) \in O(n^2)$, da der größere Summand dominiert
  - $\log(n) \prec n$, denn $\lim \log(n)/n = \lim n^{-1}/1 = 0$ nach L'Hospital
  - $n^{27} \prec e^n$,
    denn $\lim n^{27}/e^n = \lim 27 \cdot n^{26}/e^n = \ldots = 27! \cdot \lim 1/e^n = 0$ nach L'Hospital
  - $2^n \prec e^n$, denn $\lim 2^n/e^n = \lim (2/e)^n = 0$.
  - $e^n \prec n!$

Komplexitätsklassen von *Algorithmen*

Problemgröße n $\in \mathbb{N}$

- **konstant: O(1)** unabhängig von n,
  z.B. Speicherkomplexität des Euklidischen Algorithmus,
- **linear: O(n)**,
  z.B. Zeitkomplexität des Hornerschemas, wenn n der Polynomgrad ist,
- **quadratisch: $O(n^2)$**,
  z.B. Zeitkomplexität von Bubble Sort, wenn n die Elementanzahl ist,
- **kubisch: $O(n^3)$**,
  z.B. Zeitkomplexität des Gauß-Jordan-Algorithmus für $n \times n$-Matrizen,
- **polynomial: $O(n^k)$, k** $\in \mathbb{N}$,
  z.B. quadratisch oder kubisch,
- **logarithmisch: O(log n)**,
  z.B. Bisektionssuche, Suche in balanzierten Binärbäumen
- **exponentiell: $O(k^{p(n)})$, k > 1**, p(n) Polynom vom Grad ≥ 1,
  z.B. Backtracking-Algorithmen
  (Auch O(n!) wird oft als exponentiell bezeichnet, obwohl $e^n \prec n!$.)