Technische Universität München
Fakultät für Informatik, I-14
Prof. Harald Räcke
Ruslan Zabrodin

SoSe 2021
Übungsblatt 4
07. Mai 2021

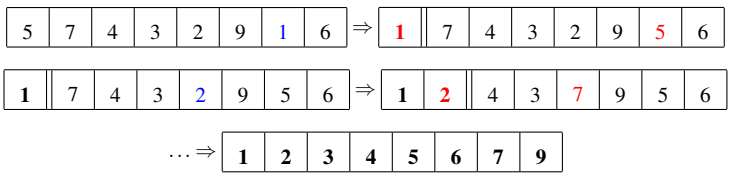
Algorithmen und Datenstrukturen

Aufgabe 1 Selectionsort

Als Alternative zu dem in der Vorlesung vorgestellten Algorithmus *Insertionsort* entwickeln wir hier einen weiteren Sortieralgorithmus – *Selectionsort*. Das Array soll Schritt für Schritt sortiert werden, indem, angefangen mit dem kleinsten Element, das jeweils nächstgrößere an den Anfang gestellt wird.

Um ein Array der Länge n zu sortieren, gehen wir in $n-1$ Schritten vor. Im ersten Schritt wird die Zelle im Array mit dem kleinsten Wert gesucht. Dieser Wert wird mit dem aus der ersten Zelle vertauscht, so dass der kleinste Wert nun ganz am Anfang des Arrays steht. Vor dem i-ten Schritt enthalten die ersten $i-1$ Zellen bereits die $i-1$ kleinsten Werte des Arrays in aufsteigender Reihenfolge. Im i-ten Schritt sucht man den kleinsten Wert der Zellen i bis n des Arrays . Dieser wird wieder mit dem Wert aus der i-ten Zelle vertauscht. Dies führt dazu, dass am Ende des i-ten Schritts die ersten i Zellen die i kleinsten Werte des Arrays in aufsteigender Reihenfolge enthalten.

Beispiel:



Schreiben Sie eine rekursive C/C++ Funktion

void selection_sort(**int** *a, **int** n);

Die Funktion bekommt ein Array a und dessen Länge n als Parameter. Die Funktion soll das Array a mit Hilfe des *Selectionsort* Algorithmus in-place sortieren.

Aufgabe 2 Komplexitätsschätzung

Schätzen Sie die Komplexitäten der folgenden Algorithmen asymptotisch ab!

a) **rot13**(*string*):

```
encrypted = string;
for i = 0 to size(string)−1 {
    if (string[i] ≥ 'a' ∧ string[i] ≤ 'z') {
        abstand = string[i] − 'a';
        rotierter_abstand = (abstand + 13) mod 26;
        encrypted[i] = 'a' + rotierter_abstand;
    }
}

return encrypted;
```

b) **MatrixMultiplikation**(A, B):

```
if (cols(A)≠rows(B)) {
    return −∞;
```

```
}
n = cols(A);
C = matrix(rows(A), cols(B));
for i = 0 to rows(A)−1 {
    for j = 0 to cols(B)−1 {
        C[i][j] = 0;
        for k = 0 to n − 1 {
            C[i][j] = C[i][j] + A[i][k] · B[k][j];
        }
    }
}
return C;
```

Aufgabe 3 Komplexität der Polynom-Auswertung

In allgemeiner Form ist ein *Polynom* gegeben durch

$$p(x) := \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0 \ ,$$

wobei $a_i \in \mathbb{R}$ der Koeffizient für die i-te Potenz x^i ist. Die höchste Potenz n ist dabei der *Grad des Polynoms*. Einen einzigen Summanden $a_i x^i$ bezeichnet man als *Monom*.

Wir untersuchen nun drei verschiedene Methoden, ein Polynom auszuwerten. Gegeben sei dazu ein Array a der Länge $n+1$, wobei $a[i]$ den Koeffizienten der Potenz x^i enthält. Die allgemeine Signatur der Funktion in Pseudo-Code ist also wie folgt:

Input: Koeffizienten-Array a der Länge $n+1$, Auswertungspunkt x
Output: Ergebnis $p = a[n] \cdot x^n + ... + a[1] \cdot x + a[0]$
EvaluatePolynomial(a,x)**:**

```
p = ...
return p;
```

a) In einer ersten Implementation gehen wir vor, wie dies auf dem Papier erfolgen würde. Wir beginnen also bei der höchsten Potenz, berechnen jeweils die Monome und addieren diese dann auf:

```
p = 0;
for i = n down to 0 {
    // Berechne i-te Potenz
    m = 1;
    for j = 1 to i {
        m = m · x;
    }

    // Berechne i-tes Monom durch Multiplikation der Potenz mit dem Koeffizienten
    m = a[i] · m;

    // Aktualisiere das Polynom durch Addition des Monoms
    p = p + m;
}
```

Bestimmen Sie die Laufzeitfunktion dieser Implementierung! In welcher Effizienzklasse in O-Notation ist dieser Code?

b) Offensichtlich ist in diesem Code die Berechnung der Potenzen redundant. Wir stellen den Code nun so um, dass die Potenzen inkrementell auf Basis vorheriger Iterationen errechnet werden:

```
p = 0;
h = 1;
for i = 0 to n {
    // Berechne i-tes Monom durch Multiplikation der (vorberechneten) Potenz mit
    // dem Koeffizienten
    m = a[i] · h;

    // Aktualisiere die Potenz für den nächsten Schritt i + 1
    h = h · x;

    // Aktualisiere das Polynom durch Addition des Monoms
    p = p + m;
}
```

Bestimmen Sie auch die Laufzeitfunktion dieser neuen Implementierung, sowohl exakt wie auch in O-Notation!

c) Zuletzt untersuchen wir das sogenannte *Horner-Schema*. Die Idee ist, das Polynom umzuformen:

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0 = (((...) \cdot x + a_2) \cdot x + a_1) \cdot x + a_0$$

Dies kommt dadurch zustande, dass man immer wieder jeweils x aus den höheren Potenzen ausklammert. In Pseudo-Code lautet eine entsprechende Implementierung dann wie folgt:

```
p = a[n];
for i = n − 1 down to 0 {
    // Multipliziere ausgeklammertes x an den bereits ausgewerteten Teil
    p = p · x;

    // Addiere den nächsten Koeffizienten
    p = p + a[i]
}
```

Bestimmen Sie zuletzt auch die Laufzeitfunktion dieser Implementierung, sowohl genau wie auch in O-Notation!

d) Basierend auf Ihren vorherigen Ergebnissen, welche Methode halten Sie für die beste?

Aufgabe 4 Binäre Suche

Aus der Vorlesung kennen wir bereits die binäre Suche in einem sortierten Array. In dieser Aufgabe wollen wir in einem zyklisch verschobenen sortierten Array ohne Duplikate die binäre Suche durchführen. Es soll also angenommen werden, dass im Array keine Zahl mehrfach enthalten ist.

Zum Beispiel wird aus dem sortierten Array $[1,4,8,12,15,18,23,25,36]$ durch zyklisches Verschieben (nach rechts) um drei Stellen das folgende zyklisch verschobene sortierte Array $[23,25,36,1,4,8,12,15,18]$

a) Implementieren Sie die Methode *search(a, c, x)* in Pseudocode. Die Methode soll durch binäre Suche in einem sortierten, um c Stellen zyklisch verschobenen Array a ein gegebenes Element x findet und den entsprechenden Index zurückgeben. Die Laufzeit Ihrer Implementierung sollte in $O(\log n)$ sein, wobei n die Anzahl der Elemente im Array ist.

b) Implementieren Sie die Methode *getShift(a)* in Pseudocode. Die Methode soll den Shift c ausgeben, d.h. die Anzahl der Stellen um welche das Array a zyklisch (nach rechts) verschobenen ist. Die Laufzeit Ihrer Implementierung sollte in $O(\log n)$ sein, wobei n die Anzahl der Elemente im Array ist.