

## Algorithmen und Datenstrukturen

### Aufgabe 1      Algorithmenanalyse

- a) Gegeben sind folgende Algorithmen in Pseudocode. Geben Sie an, welche Probleme die Algorithmen lösen.

(i)  $f(a)$  :

```
if (a == 0)
    return true
if (a == 1)
    return false
if (a < 0)
    return f(-a)
return f(a - 2)
```

(ii)  $g(a, b)$  :

```
if (b == 1)
    return a
return a * g(a, b - 1)
```

- b) Schauen sie sich die folgende Implementierung der Methode `removeVal` einer verketteten Liste an. Was ist anders im Vergleich zu der Implementierung in der Vorlesung? Auf welchen Eingaben arbeitet die Methode korrekt? Auf welchen nicht?

```
void removeVal(int v, ListNode* &ptr) {
    if (ptr == NULL) return;
    if (ptr->data == v) remove(ptr);
    removeVal(v, ptr->next);
}
```

### Aufgabe 2 (P) Binomialkoeffizienten - Memoization

Im vorherigen Aufgabenblatt haben Sie bereits die folgende Rekursionsvorschrift für Binomialkoeffizienten kennengelernt:

$$\binom{n}{0} = \binom{n}{n} = 1 \quad \text{für } n \geq 0$$
$$\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \quad \text{für } n > 0 \text{ und } 0 \leq k < n$$

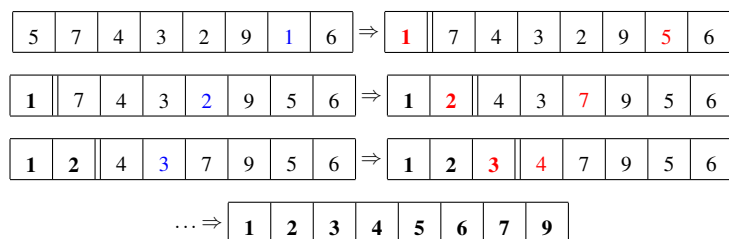
Schreiben Sie ein rekursives C/C++ Programm, das den Binomialkoeffizienten für gegebene  $n$  und  $k$  berechnet ( $n, k \leq 1000$ ). Benutzen Sie die *Memoizationstechnik*, wie sie in der Vorlesung für die Fibonacci-Zahlen gezeigt wurde. Nutzen Sie gerne das Testing Framework.

### Aufgabe 3 (P) 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 `selection_sort(vector<int>& vec)` um die gegebene Liste mit Hilfe des *Selectionsort* Algorithmus in-place (ohne eine Kopie des vectors anzulegen) zu sortieren.

Sie benötigen dafür eine rekursive Hilfsfunktion. Entscheiden Sie, ob Sie das Problem mit Iteratoren oder mit Indices lösen möchten und überlegen Sie welche Parameter Ihre Hilfsfunktion benötigt. Sie können für diese Aufgabe den Typen des Iterators `vector_iterator` verwenden.

### Aufgabe 4 (P) Binäre Suche

Aus der Vorlesung kennen wir bereits die binäre Suche in einem sortierten Array. In dieser Aufgabe wollen wir die binäre Suche in einem *zyklisch verschobenen sortierten Array* durchführen. Ein solches entsteht aus einem sortierten Array, wenn für einen gegebenen Shift-Wert  $0 \leq c < n$  jeder Wert von Index  $i$  auf Index  $(i + c) \bmod n$  verschoben wird, für alle  $i$ .

Beispiel:  $[1, 4, 8, 12, 15, 18, 23, 25, 36] \xrightarrow{\text{shift 3 rechts}} [23, 25, 36, 1, 4, 8, 12, 15, 18]$

Wir nehmen an, dass das Array  $n$  verschiedene Elemente, also *keine Duplikate* enthält. Nutzen Sie gerne das Testing Framework.

- a) Implementieren Sie die C/C++ Methode `search(vector<int> vec, int c, int x)`. Die Methode soll durch binäre Suche in einem sortierten, um  $c$  Stellen zyklisch verschobenen Array `vec` ein gegebenes Element  $x$  finden und den entsprechenden Index zurückgeben. Die Laufzeit Ihrer Implementierung sollte in  $O(\log n)$  sein.

- b) Implementieren Sie die C/C++ Methode `getShift (vector<int> vec)`. Die Methode soll den Shift  $c$  ausgeben, d.h. die Anzahl der Stellen um welche das Array  $vec$  zyklisch (nach rechts) verschoben ist.  
Die Laufzeit Ihrer Implementierung sollte in  $O(\log n)$  sein.