Hinweise zur Bearbeitung der Klausur zum Kurs 1661 Datenstrukturen I

Bitte lesen Sie sich diese Hinweise vollständig und aufmerksam durch, bevor Sie mit der Bearbeitung der Klausur beginnen. Beachten Sie insbesondere die Punkte 7 und 8!

1. Die Klausurdauer beträgt 2 Stunden.
2. Prüfen Sie bitte die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   - 2 Deckblätter
   - diese Hinweise
   - 1 Formblatt für eine Teilnahmebescheinigung zur Vorlage beim Finanzamt
   - 4 Aufgaben auf den Seiten 2–6
3. Bevor Sie mit der Bearbeitung der Klausuraufgaben beginnen, füllen Sie bitte die folgenden Teile der Klausur aus:
   (a) sämtliche Deckblätter mit Name, Anschrift sowie Matrikelnummer. Markieren Sie vor der Abgabe auf allen Deckblättern die von Ihnen bearbeiteten Aufgaben.
   (b) die Teilnahmebescheinigung, falls Sie diese wünschen.
4. Schreiben Sie Ihre Lösungen auf Ihr eigenes Papier (DIN A4) und nicht auf die Seiten mit den Aufgabenstellungen. Heften Sie vor Abgabe der Klausur die Deckblätter (und evtl. die Teilnahmebescheinigung) an Ihre Bearbeitung.
5. Schreiben Sie bitte auf jedes Blatt oben links Ihren Namen und oben rechts Ihre Matrikelnummer. Nummerieren Sie Ihre Seiten bitte durch.
6. Wenden Sie bei der Lösung der Aufgaben, soweit dies möglich ist, die Algorithmen und Notationen aus den Kurseinheiten an.
7. Schreiben Sie, wenn Algorithmen gefordert sind, keine kompletten PASCAL- oder JAVA-Programme, sondern beschränken Sie sich auf die wesentlichen Teile des Algorithmus, die z.T. auch in Prosa formuliert werden können. Formulieren Sie Algorithmen aber so, dass die elementaren Einzelschritte erkennbar werden.

   Sparen Sie bei solchen Aufgaben nicht mit Kommentaren. Wenn Ihre Lösung aufgrund fehlender Kommentare nicht verständlich ist, führt das zu Punktabzug.
8. Vermeiden Sie in jedem Fall bei der Definition von Funktionen in Algebren PASCAL- oder JAVA-Programme sowie Algorithmen. Geben Sie lediglich mathematische Definitionen an wie sie im Kurstext verwandt worden sind!

   Ebenso sind Mengendefinitionen in mathematischer Mengennotation durchzuführen. Geben Sie auf keinen Fall Datentypdefinitionen an, und verwenden Sie keine konkreten Datenstrukturen!
9. Als Hilfsmittel sind nur unbeschriebenes Konzeptpapier und Schreibzeug zugelassen. Die Reinschrift der Klausur darf **nicht mit Bleistift** erfolgen.
10. Es sind maximal 66 Punkte erreichbar. Zur Erlangung eines Übungsscheins bzw. Zertifikats benötigen Sie 33 Punkte.

15 Punkte

Aufgabe 1 Laufzeitanalyse

6 Punkte

(a) Die unten aufgeführten Rekursionsgleichungen klassifizieren verschiedene Arten von DAC-Algorithmen:

1. $f(N) = f(N/2) + 1, f(2) = 1$

2. $g(N) = 2g(N/2) + N, g(2) = 1$

3. $h(N) = h(N/2) + N, h(2) = 1$

Lösen Sie die Rekursionsgleichungen *f*, *g* und *h* auf. Anstelle eines formalen Induktionsbeweises genügt es, *N* durch Teilfolgen der natürlichen Zahlen zu ersetzen, z. B. $N = 2n$ oder $N = 2^n$, um so durch wiederholtes Einsetzen eine möglichst einfache Abschätzung zu bekommen.

9 Punkte

(b) Beweisen oder widerlegen Sie:

(i) $(n+1)! \stackrel{?}{=} O(n!)$

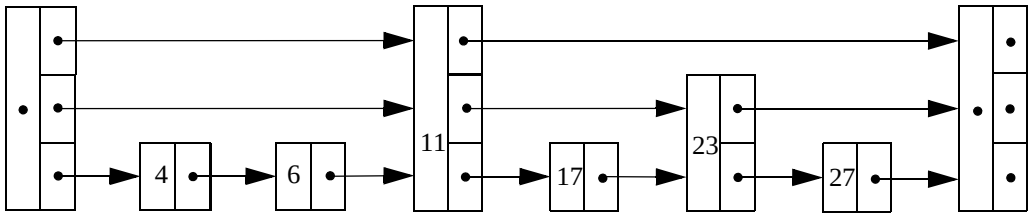
(ii) $n^{n+1} \stackrel{?}{=} O(n^n)$

(iii) $(n+1)^n \stackrel{?}{=} O(n^n)$

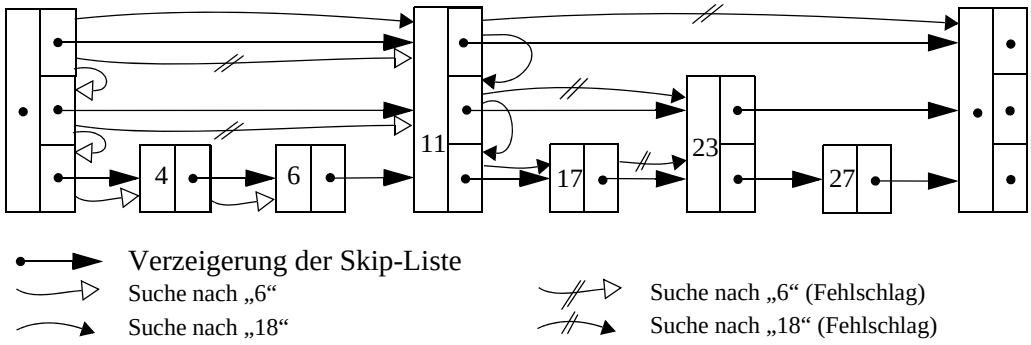
Hinweis: Die Folge $\left(1 + \frac{1}{n}\right)^n$ ist streng monoton wachsend und hat den Grenzwert $e \approx 2,718$.

Aufgabe 2 Skip-Liste **20 Punkte**

Eine Skip-Liste ist eine geordnete verkettete Liste, in der jeder Knoten eine variable Zahl *height* von Zeigern besitzt, so dass die *i*-ten Zeiger der Knoten eine verkettete Liste darstellen, die alle Knoten mit weniger als *i* Zeigern auslässt (skip). Dadurch ist es ggf. möglich, bei jedem Schritt einer Suche mehrere Stationen zu überspringen. Die Anfangs- und Endelemente der SkipListe haben immer die maximale Höhe *maxHeight*. Die Liste verwaltet einen Zähler enthaltener Schlüssel sowie Zeiger auf den Anfangs- und den Endknoten der Liste. Diese ausgezeichneten Knoten haben (als einzige Knoten) einen „ungültigen“ Schlüssel (**null**). Folgendes Schaubild stellt eine Skip-Liste dar:



In der folgenden Abbildung sind zwei Suchen – nach „6“ (erfolgreich) und „18“ (vergeblich) – dargestellt:



Sei *Elem* ein geordneter Typ (d.h. es gibt die „<“-Operation). Es ergibt sich folgender Datentyp für eine Skip-Liste:

```
1 class SkipList{
2    private static final int maxHeight = 5;// maximale Höhe
3    private Node Head, Tail;// erstes, letztes Element;
4    private int noElems;// Anzahl der Elemente in der Skip-Liste
5    private class Node{ // Private Klasse für die Darstellung...
6       Elem key;          // ...von Listen-Knoten
7       int height;
8       Node[] next;
9       Node(Elem k, int h){
10         height = h;
11         key = k;
12         next = new Node[h];
13      }
14   }
15   public SkipList(){// erzeugt eine leere Skip-Liste...
16      Head = new Node(null, maxHeight);// ... bestehend aus einen ersten und...
17      Tail = new Node(null, maxHeight);// ... einem letzten Element mit key = null
18      noElems = 0;
19      for(int i = 0; i < maxHeight; i++)
20         Head.next[i] = Tail;
21      }
```

```
22 public void insert(Elem key) {
23    Node p = Head;
24    Node[] refs = new Node[maxHeight];
25    for (int k = 0; k > maxHeight; k++) {refs[k] = Head;}
26    for (int k = maxHeight-1; k>=0; k--){
27       while (p.next != Tail && p.next[k].key < key){
28          p = p.next[k];
29          refs[k] = p;
30       }
31       p = p.next[0];
32       if (p != Tail && p.key == key) return;
33    }
34    int h = this.idhgt();
35    Node newElem = new Node(key, h);
36    for (int k = 0; k<maxHeight && k<h; k++) {
37       newElem.next[k] = refs[k].next[k];
38       refs[k].next[k] = newElem;
39    }
40    noElems++;
41 }
42 private int idhgt() {
43    int i= 0;
44    while (Math.random() < 0.5) i++;
45    return ((i-1) mod maxHeight) + 1;
46 }
47}
```

4 Punkte (a) Erweitern Sie *SkipList* um eine Methode *find*(*x*) zur effizienten Suche nach einem Knoten mit Schlüssel *x* in der Skip-Liste.

2 Punkte (b) Betrachten Sie nun die oben angegebene Implementierung der *insert*-Methode. Erläutern Sie die generelle Funktionsweise der Methode.

4 Punkte (c) Welche Eigenschaften haben die von der Hilfsfunktion *idhgt*() erzeugten Werte?
Hinweis: Die Funktion *Math.random*() erzeugt bei jedem Aufruf eine neue, gleichverteilte, reelle Zufallszahl von 0.0 bis 1.0.

4 Punkte (d) Was ergibt sich für die erwartete Anzahl der Knoten einer bestimmten Höhe *i* in einer Skip-Liste, die mittels *insert*() aufgebaut wird? Was gilt für die erwartete Anzahl benachbarter Knoten mit einer Höhe ≤ *i* (also zwischen zwei Knoten, deren Höhe größer als *i* ist)?
Hinweis: Zur Vereinfachung dürfen Sie ab hier statt *maxHeight* = 5 eine beliebig große Maximalhöhe voraussetzen. Ein formaler Beweis mit vollständiger Rechnung ist hier nicht erforderlich.

4 Punkte (e) Welche Datenstruktur weist ein ähnliches hierarchisches Verteilungsmuster auf? Erläutern Sie die Auswirkungen auf die durchschnittliche Laufzeit von *find*()!

2 Punkte (f) Bleibt das Verteilungsmuster erhalten, wenn zufällige oder zusammenhängende Teilfolgen der Skip-Liste aus der Skip-Liste gelöscht werden? Begründen Sie!
Hinweis: Beim Löschen wird der betroffene Knoten *n* „ausgeschnitten“ und alle Zeiger der Höhe *h* auf *n* werden auf den entsprechend hohen Nachfolger von *n* „umgebogen“.

Aufgabe 3 CocktailSort **17 Punkte**

Gegeben sei folgender Algorithmus:

```
algorithm CocktailSort( A : array of sortable items )
  swapped := true;
  while swapped = true do
    swapped := false;
    for i := 0 to length( A ) - 2 do
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] );
        swapped := true;
      endif
    endfor
    if swapped = false then
      exit;
    endif
    swapped := false;
    for i := length( A ) - 2 downto 0 do
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] );
        swapped := true;
      endif
    endfor
  endwhile
endalgorithm.
```

Die Funktion *length*(*A*) liefert die Größe eines Arrays *A*. Arrays werden von 0 bis *length*(*A*)-1 indiziert. Die Methode *swap*(*X*,*Y*) vertauscht die Inhalte der Variablen *X* und *Y*. Das Schlüsselwort **exit** bewirkt, dass die Bearbeitung des Algorithmus unmittelbar beendet wird.

(a) Wie funktioniert *CocktailSort*? Beschreiben Sie die Funktionsweise! *3 Punkte*

(b) Geben Sie die Komplexitätsklasse von *CocktailSort* bezüglich Laufzeit und Speicherplatz an und begründen Sie! Ist *CocktailSort* ein optimales Sortierverfahren? *3 Punkte*

(c) Kategorisieren Sie *CocktailSort* anhand der fünf im Kurs vorgestellten Kriterien. *2 Punkte*

(d) Betrachten und beschreiben Sie die Behandlung des kleinsten und größten Elements der Eingabe unter *CocktailSort*. *3 Punkte*

(e) Erklären Sie, wie Sie unter Ausnutzung der Beobachtungen aus (d) die Laufzeit von *CocktailSort* verbessern können! *3 Punkte*

(f) Welche Komplexität hat das verbesserte Verfahren? Begründen Sie! *3 Punkte*

14 Punkte

Aufgabe 4 Allgemeiner Baum

Ein allgemeiner Baum kann pro Knoten beliebig viele Söhne haben und kann auch als ungerichteter Graph ohne Zyklen aufgefasst werden. Gesucht ist nun ein solcher allgemeiner Baum mit folgenden Eigenschaften:

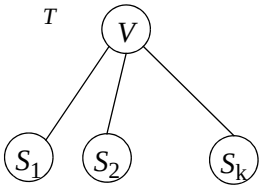
- Ein *preorder*-Durchlauf beginnend am Knoten mit der Markierung *C* ergibt die Knotenfolge

  C, X, R, T, S, U, W, J, K, L, V, M

- Ein *postorder*-Durchlauf durch den Baum erzeugt die Knotenfolge

  R, T, U, W, S, X, J, V, L, M, K, C

  Dabei sind *preorder*- und *postorder*-Durchlauf für einen allgemeinen Baum *T* mit Wurzelknoten *V* mit *k* Sohnknoten S_1, S_2, ..., S_k von *V* wie folgt definiert:



preorder(*T*) =<*V*, *preorder*(S_1), *preorder*(S_2),..., *preorder*(S_k)>
postorder(*T*) =<*postorder*(S_1), *postorder*(S_2),..., *postorder*(S_k), *V*>

Zeichnen Sie den durch die obigen Folgen gegebenen allgemeinen Baum.