# Informatik II D-BAUG Midterm, 27. April 2017 Lösung

Name, Vorname: .....

Legi-Nummer: .....

Die Note des Midterms geht dann und nur dann zu 30% in Ihre Informatik II Basisprüfungsnote ein, wenn es Ihnen zum Vorteil gereicht.

*The midterm grade contributes 30% to your Informatik II grade of the Basisprüfung, if and only if it provides a benefit for you.*

**Allgemeine Richtlinien:**

***General guidelines:***

1. Dauer der Prüfung: 75 Minuten.

   *Exam duration: 75 minutes.*

2. Erlaubte Unterlagen: Wörterbuch (für gesprochene Sprachen). 4 A4 Seiten handgeschrieben oder $\geq$ 11pt Schriftgrösse.

   *Permitted examination aids: dictionary (for spoken languages). 4 A4 pages hand written or $\geq$ 11pt font size*

3. Benützen Sie einen Kugelschreiber (blau oder schwarz) und keinen Bleistift. Bitte schreiben Sie leserlich. Nur lesbare Resultate werden bewertet.

   *Use a pen (black or blue), not a pencil. Please write legibly. We will only consider solutions that we can read.*

4. Lösungen sind direkt auf das Aufgabenblatt in die dafür vorgesehenen Boxen zu schreiben (und direkt darunter, falls mehr Platz benötigt wird). Ungültige Lösungen sind deutlich durchzustreichen! Korrekturen bei Multiple-Choice Aufgaben bitte unmissverständlich anbringen! Lösungen auf Notizblättern werden nicht berücksichtigt.

   *Solutions must be written directly onto the exam sheets in the provided boxes (and directly below, if more space is needed). Invalid solutions need to be crossed out clearly. Provide corrections to answers of multiple choice questions without any ambiguity! Solutions on extra sheets will not be considered.*

5. Es gibt keine Negativpunkte für falsche Antworten.

   *There are no negative points for false answers.*

6. Störungen durch irgendjemanden oder irgendetwas melden Sie bitte sofort der Aufsichtsperson.

   *If you feel disturbed by anyone or anything, let the supervisor of the exam know immediately.*

7. Wir sammeln die Prüfung zum Schluss ein. Wichtig: stellen Sie unbedingt selbst sicher, dass Ihre Prüfung von einem Assistenten eingezogen wird. Stecken Sie keine Prüfung ein und lassen Sie Ihre Prüfung nicht einfach am Platz liegen. Dasselbe gilt, wenn Sie früher abgeben wollen: bitte melden Sie sich lautlos, und wir holen die Prüfung ab. Vorzeitige Abgaben sind nur bis 15 Minuten vor Prüfungsende möglich.

   *We collect the exams at the end. Important: you must ensure that your exam has been collected by an assistant. Do not take any exam with you and do not leave your exam behind on your desk. The same applies when you want to finish early: please contact us silently and we will collect the exam. Handing in your exam ahead of time is only possible until 15 minutes before the exam ends.*

8. Wenn Sie zur Toilette müssen, melden Sie dies einer Aufsichtsperson durch Handzeichen.

   *If you need to go to the toilet, raise your hand and wait for a supervisor.*

9. Wir beantworten keine inhaltlichen Fragen während der Prüfung. Kommentare zur Aufgabe schreiben Sie bitte auf das Aufgabenblatt.

   *We will not answer any content-related questions during the exam. Please write comments referring to the tasks on the exam sheets.*

| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | $\sum$ |
|---|---|---|---|---|---|---|---|
| Punkte | | | | | | | |
| Maximum | 6 | 4 | 5 | 8 | 10 | 7 | 40 |

# 1 Code Lesen *(6 Punkte)*

Beschreiben Sie in den folgenden Boxen jeweils in Worten, was die darunter stehende Funktion macht. Beschreiben Sie nicht die einzelnen Schritte sondern die Bedeutung.

*Provide in the following boxes what the functions are doing. Do not describe the single steps but rather the meaning.*

(a) Gibt zurück, ob b in a enthalten ist. 2 P

```
// pre: non-null array a
static boolean P(int[] a, int b){
  int r = a.length;
  while (r > 0){
      if (a[--r] == b) {return true;}
  }
  return false;
}
```

(b) Teilersumme von $n$ 2 P

```
// pre: n >= 0
static int S(int n){
    int r = 0;
    for (int i = 1; i<=n; ++i){
        if (n % i == 0){ r += i; }
    }
    return r;
}
```

(c) Rückgabe von $a^b$ 2 P

```
// pre: a,b > 0
static int P(int a, int b){
  if (b == 0){
      return 1;
  }
  return a * P(a,b-1);
}
```

## 2 Code Schreiben *(4 Punkte)*

Vervollständigen Sie folgende Funktion so, dass sie die Elemente des Array Parameters a umdreht. Verwendungsbeispiel:

*Complement the following function such that it reverses the elements of the array parameter* a. *Application example:*

```
float[] a = {1,2,3,4};
R(a); // now a is {4,3,2,1}
```

```
static void R(float[] a){
    int l = 0;
    int u = a.length-1;
    while (u > l){
    float t=a[u];
    a[u] = a[l];
    a[l] = t;
    --u;
    ++l;
    }
}
```

## 3 Asymptotisches Verhalten *(5 Punkte)*

Sortieren Sie die folgenden Funktionen von links nach rechts so, dass wenn $f$ links von $g$ steht, dann gilt $f \in O(g)$. Folgende Funktionen stehen zum Beispiel in korrekter Ordnung: $n^3, n^5, n^n$.

*Sort the following functions from left to right such that: if function $f$ is left of function $g$ then $f \in O(g)$. The following functions, for example, are ordered correctly:* $n^3, n^5, n^n$.

$$n^n, n^2 + 5 \cdot n, n \log n, 3 \cdot n, (\log n)^2, \log(n^2), 2^n, \sqrt{n^5}, 10$$

$10, \log(n^2), (\log n)^2, 3 \cdot n, n \log n, n^2 + 5 \cdot n, \sqrt{n^5}, 2^n, n^n$

# 4 Listen *(8 Punkte)*

Sei für diese Aufgabe folgende Listknoten-Klasse gegeben:

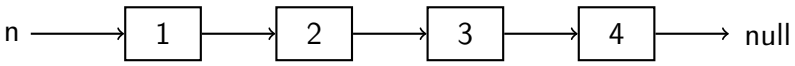
*Let for this task be the following list-node class be given:*

```
public class ListNode{
    ListNode next;
    int value;

    ListNode (int v, ListNode n){
        value = v; next = n;
    }
}
```

Es sei eine verkettete Liste von Knoten gegeben. Das Ende der Liste ist durch einen Null-Knoten gegeben. Beispiel:

*Assume a linked list of node elements. The end of the linked ist is provided by a null-node. Example:*




Die folgende rekursive Funktion gibt die Werte der Knoten aus, startend beim gegebenen Knoten $n$ bis zum Ende der verketteten Liste.

*The following recursive function outputs the node's values, starting from the given node element $n$ and traversing to the end of the linked list.*

```
void Output(ListNode n){
  if (n != null){
    System.out.println(n.value);
    Output(n.next);
  }
}
```

Ausgabe Beispiel
*Example Output:*
1
2
3
4

(a) Vervollständigen Sie den folgenden Code so, dass die Werte in umgekehrter Reihenfolge ausgegeben werden. Verwenden Sie Rekursion!

*Complement the following Code such that the values in the list are output in reversed order. Use recursion!* 4 P

```
void OutputR(ListNode n){
    
    if (n != null){
    OutputR(n.next);
    System.out.println(n.value);
    }
    
}
```

Ausgabe Beispiel
*Example Output:*
4
3
2
1

(b) Nachfolgend sehen Sie eine (korrekte) Implementation einer Listenklasse, mit Operationen out, empty und append. Jemand behauptet, dass die Implementation mancher Operationen einfacher wird, wenn die Liste am Anfang immer mit einem nicht-leeren (Wächter-) Element beginnt und schlägt daher die danach folgende SentinelListe vor. Vervollständigen Sie also die Methoden der SentinelListe so, dass die beiden Klassen List und SentinelList dieselbe Semantik aufweisen.

*In the following you see a (correct) implementation of a list class with operations out, empty and append. Someone claims that the implementation of some operations becomes simpler when the list always starts with a non-empty (sentinel) element. Therefore he proposes the following SentinelList. Complement the methods of the SentinelList such that both classes List and SentinelList provide the same semantics.*

4 P

Vervollständigen Sie die Methoden der SentinelListe auf der rechten Seite so, dass die beiden Klassen List und SentinelList dieselbe Semantik aufweisen.

*Complement the methods of the SentinelList on the right page such that both classes List and SentinelList provide the same semantics.*

```
class List{
  ListNode head;

  List(){ head = null; }

  // post: outputs the list content (first in, first out)
  public void out(){
    ListNode n = head;
    while (n != null){
        System.out.print(n.value + " ");
        n = n.next;
    }
  }

  // post: returns if the list is empty
  public boolean empty(){
    return head == null;
  }

  // post: appends a new node with value at the end of the list
  public void append(int value){
    if (head == null){
      head = new ListNode(value, null);
      return;
    }
    ListNode p = head;
    while (p.next != null){
      p = p.next;
    }
    p.next = new ListNode(value,null);
  }
}
```

```
class SentinelList{
  ListNode head;

  SentinelList(){
    head = new ListNode(0, null); // head = sentinel
  }

  // post: outputs the list content (first in, first out)
  public void out(){
     ListNode n = head.next;
     while (n != null){
         System.out.print(n.value + " ");
         n = n.next;
     }
  }

  // post: returns if the list is empty
  public boolean empty(){
     return head.next == null;
  }

  // post: appends a new node with value at the end of the list
  public void append(int value){


     ListNode p = head;
     while (p.next != null){
     p = p.next;
     }
     p.next = new ListNode(value,null);



  }
}
```

# 5 Algorithmen *(10 Punkte)*

(a) Erklären Sie mit Ihren eigenen Worten kurz, wie der Algorithmus Quickselect zur Suche des $k$-kleinsten Elementes in einem unsortierten Array funktioniert. Hinweis: Pivot Element.

*Explain with your own words shortly, how the algorithm QuickSelect for searching the $k$-smallest element in an unsorted array works. Hint: pivot element.*

4 P

Unterteilung des Arrays in zwei Teile: Elemente die grösser oder kleiner dem Pivot sind. Wenn der Pivot an $k$-ter Stelle steht, Element gefunden. Nach Einteilung des Arrays Rekursion: Suche auf dem "richtigen" Teil, also dem Teil, welcher das $k$-kleinste Element enthalten kann.

(b) Geben Sie asymptotisch scharfe Schranken an (Landau-Notation!) für die Anzahl Vergleichsoperationen für folgende Algorithmen. Die Eingabegrösse oder Datenlänge ist $n$.

*Provide asymptotic tight bounds (Landau-Notation) for the number of comparison operations for the following algorithms. The input size or data length is $n$.*

6 P

| Algorithmus<br>*Algorithm* | Bester Fall<br>*Best case* | Schlechtester Fall<br>*Worst case* |
|---|---|---|
| Lineare Suche im unsortierten Array<br>*Linear search in unsorted array* | $O(1)$ | $O(n)$ |
| Binäre Suche im sortierten Array<br>*Binary search in sorted array* | $O(1)$ | $O(\log n)$ |
| Element suchen in verketteter Liste<br>*Search element in a linked list* | $O(1)$ | $O(n)$ |
| Minimum eines Arrays bestimmen<br>*Determine the minimum of an array* | $O(n)$ | $O(n)$ |
| Median bestimmen mit Quickselect<br>*Determine the median with Quickselect* | $O(n)$ | $O(n^2)$ |
| Array sortieren mit Selection Sort<br>*Sort an array with Selection Sort* | $O(n^2)$ | $O(n^2)$ |
| Schlüssel suchen mit linearem Sondieren in Hashtabelle<br>*Search key using linear probing in a hash table* | $O(1)$ | $O(n)$ |

# 6 Der Elefant *(7 Punkte)*

Ein Elefant hat einen Lieblingstrampelpfad, auf dem er sich permanent aufhält. Jede Minute bewegt sich der Elefant bis zu zwei Schritte nach vorne (+1,+2) oder zurück (-1,-2) oder er bleibt stehen (0). Der folgende Vektor $P$ gibt die entsprechenden Wahrscheinlichkeiten an:

| $Schritte$ | $-2$ | $-1$ | $0$ | $1$ | $2$ |
|---|---|---|---|---|---|
| $Wahrscheinlichkeit$ | $0.1$ | $0.1$ | $0.4$ | $0.3$ | $0.1$ |

Vervollständigen Sie folgenden Code, so dass er das Verhalten des Elefanten simuliert und die Endposition des Elefanten nach einem Tag ausgibt.

Schreiben Sie Ihren Code so, dass er auch für ein $P$ mit anderer Grösse funktionieren würde. Zum Beispiel könnte $P$ alternativ 7 Einträge haben für Bewegungen von maximal 3 Schritten nach vorne oder zurück. Wenn Sie nicht wissen, wie das geht, schreiben Sie die statische Variante hin, das gibt auch Punkte (aber weniger).

*An elefant has a favourite trail that it stays on permanently. Every minute the elefant moves maximally two steps forwards (+1,+2) or backwards (-1, -2) or he stays still (0). The following vector $P$ provides the respective probabilities:*

| $Steps$ | $-2$ | $-1$ | $0$ | $1$ | $2$ |
|---|---|---|---|---|---|
| $Probability$ | $0.1$ | $0.1$ | $0.4$ | $0.3$ | $0.1$ |

*Complement the following code such that it simulates the behavior of the elefant and outputs the end position of the elefant after one day. Write the sample function such that it would also work for a $P$ with a different size. For example: $P$ could alternatively have 7 entries for moving forwards and backwards up to 3 steps. If you don't know how this works, try the static variant, which also yields some points (but less).*

```
public class Main {

    // sample from the probability vector p:
    // return int k with probability p[k]
    // ( 0 <= k < p.length )
    static int Sample(double[] p){
        double u = Math.random();
            int k=0;
            while (k < p.length && u>= 0){
            u -= p[k];
            k++;
            }
            return k-1;
    }

    public static void main(String[] args){
        double[] P = {0.1 ,0.1 ,0.4 ,0.3 ,0.1};
        int[] steps = {-2, -1, 0, 1, 2};
        int current = 0;
        for (int i = 0; i<24*60; ++i){
            current += steps[Sample(P)];
        }
        System.out.println("Elephant position: " + current);
    }
}
```