Prüfunsaufaben Informatik II

Dies ist ein Zusammenschnitt von alten Prüfungsaufgaben, welche wir während dem PVK lösen werden.

Hier ist eine Liste von allen Aufgaben zusammen mit der Anzahl Notenpunkte, welche diese Aufgabe an Ihrer Prüfung gab:

Name	**Prüfung**	**Notenpunkte**
Objektorientierung	HS17	1.17
Laufzeit- und Speicherkomplexität	HS17	0.67
Bäume	FS16	0.67
Bytecode	HS17	0.34
Syntaxdiagramme und Syntaxchecker	FS16	0.35
Parallelität	FS18	0.75
Programmverifikation	HS17	0.58
Bäume	FS16	0.67

Aufgabe 3: Objektorientierung *(14 Punkte)*

Da Sie Ihre Freizeit gerne mit Outdooraktivitäten verbringen, möchten Sie eine Software in Java schreiben, die diese Aktivitäten speichert und verwaltet, damit Sie und Ihre Freunde darauf zurückgreifen können. Eine Aktivität (`Activity`) hat eine gewisse Dauer in Stunden (`duration`) und eine Bewertung von 0 bis 5 (`rating`), je höher desto beliebter. Eine Akivität kann vielerlei Gestalt annehmen, z.B. die einer Wanderung (`Hike`). Eine Wanderung hat ausserdem einen Schwierigkeitsgrad (`difficulty`) zwischen 1 und 5 (beide einschliesslich), je höher desto schwieriger.

3 a) (3 Punkte) Implementieren Sie die Klasse `Activity` mitsamt Konstruktor und Getter-Methoden. Prüfen Sie, ob gültige Werte für die Dauer (grösser als 0) und die Bewertung (0 bis 5) übergeben werden und geben Sie sonst eine `IllegalArgumentException` mit einer entsprechenden Nachricht zurück. Setter-Methoden werden nicht benötigt.

```
1  public class Activity {
2      private String name;
3      private double duration;
4      private double rating;
5
6      ..........................................
7
8      ..........................................
9
10     ..........................................
11
12     ..........................................
13
14     ..........................................
15
16     ..........................................
17
18     ..........................................
19
20     ..........................................
21
22     ..........................................
23
24     ..........................................
25
26     ..........................................
27
28     ..........................................
29
30     ..........................................
31
32     ..........................................
33
34     ..........................................
35
36     ..........................................
37 }
```

3 b) (2 Punkte) Implementieren Sie die Klasse `Hike` mitsamt Konstruktor und Getter-Methode. Prüfen Sie hier ebenso auf gültige Wertübergabe und geben Sie im Fall ungültiger Werte ebenfalls eine `IllegalArgumentException` mit einer entsprechenden Nachricht zurück.

```
1
2  public class Hike ..................... {
3
4      private double difficulty;
5
6      ..........................................
7
8      ..........................................
9
10     ..........................................
11
12     ..........................................
13
14     ..........................................
15
16     ..........................................
17
18     ..........................................
19
20     ..........................................
21
22     ..........................................
23
24     ..........................................
25 }
```

3 c) (2 Punkte) Der `ActivityManager` speichert und verwaltet die Aktivitäten. Fügen Sie das Attribut `activities` hinzu, welches eine `ArrayList` ist, die Objekte vom Typ `Activity` hält. Implementieren Sie den Konstruktor der Klasse `ActivityManager`, der das Attribut `activities` als leere `ArrayList` initialisiert, und die Methode `addActivity`, die der `ArrayList` eine Instanz der Klasse `Activity` hinzufügt.

```
1  import java.util.ArrayList;
2
3  public class ActivityManager {
4
5      ..........................................
6
7      ..........................................
8
9      ..........................................
10
11     ..........................................
12
13     ..........................................
14
15     public void addActivity(Activity activity) {
16
17         ......................................
18     }
19
20     public Hike findBestHike(double maxDuration, double maxDifficulty) {
21
22         ......................................
23
24         ......................................
25
26         ......................................
27
28         ......................................
29
30         ......................................
31
32         ......................................
33
34         ......................................
35
36         ......................................
37
38         ......................................
39
40         ......................................
41     }
42 }
```

3 d) (4 Punkte) Implementieren Sie die Methode `findBestHike`, die für eine maximale Dauer und einen maximalen Schwierigkeitsgrad aus der Liste von Aktivitäten die Wanderung mit der höchsten Bewertung zurückgibt. Gehen Sie davon aus, dass dieser Methode nur gültige Werte übergeben werden. Falls es keine Wanderung gibt, die die passenden Kriterien erfüllt, geben Sie `null` zurück.

3 e) (1 Punkt) Welche Laufzeitkomplexität hat Ihre Implementierung der Methode `findBestHike` in Abhängigkeit der Anzahl n der gespeicherten Aktivitäten? Begründen Sie Ihre Antwort kurz.

.....

.....

```
1
2  public class Main {
3
4      public static void main(String[] args) {
5          ActivityManager manager = new ActivityManager();
6
7          // Adding activities
8          // Assume the parameter order for the Activity constructor is
9          // [name, duration, rating] and for the Hike constructor
10         // [name, duration, rating, difficulty]
11         manager.addActivity(new Activity("Spaziergang am See", 1.2, 2.5));
12         manager.addActivity(new Hike("Brunni - Kl. Mythen", 2, 4.5, 4.5));
13         manager.addActivity(new Activity("Velofahren Zuerich", 0.5, 1));
14         manager.addActivity(new Hike("Brunni - Gr. Mythen", 3, 3.5, 3.5));
15         manager.addActivity(new Activity("Schwimmen am Letten", 1, 4.5));
16         manager.addActivity(new Activity("Nichtstun", 0, 2.5));
17         manager.addActivity(new Hike("Weggis - Rigi Kulm", 5, 4, 3.5));
18         manager.addActivity(new Hike("Alpnach - Pilatus Kulm", 5, 5, 2));
19
20         double maxDuration = 4;
21         double maxDifficulty = 3.5;
22         Hike bestHike = manager.findBestHike(maxDuration, maxDifficulty);
23
24         System.out.println("Die beste Wanderung: " + bestHike.getName());
25     }
26
27 }
```

3 f) (2 Punkte) Was ist die Ausgabe des Programmstücks der obigen *main*-Methode?

.....

.....

Aufgabe 6: Laufzeit- und Speicherkomplexität, Parallelität *(17 Punkte)*

Analog zur Laufzeitkomplexität kann man auch die Speicherkomplexität definieren, die den Speicheraufwand in Relation zur Eingabegrösse n der Probleminstanz setzt. Hier sei die Speicherkomplexität der zusätzlich zur Eingabegrösse notwendige Speicheraufwand.

Das "Maximale Teilsummen"-Problem lautet wie folgt: Gegeben eine endliche Folge von Zahlen; was ist die maximale Summe, die aus einer Teilfolge, bestehend aus benachbarten Elementen der Folge, gebildet werden kann? Sind alle Zahlen positiv, dann ist die Lösung trivialerweise die Summe aller Elemente der Folge, enthält die Folge hingegen negative Zahlen, ist sie nicht-trivial.

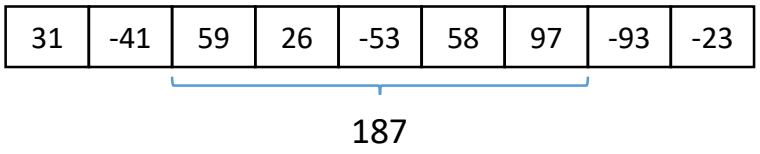


Abbildung 1: Beispiel für eine Folge mit maximaler Teilsumme 187.

6 a) (4 Punkte) Ein erster naiver Ansatz iteriert über das Eingabearray und bildet eine linke Grenze, iteriert für eine solche linke Grenze dann über alle möglichen rechten Grenzen rechts der linken Grenze und bildet die Summe der begrenzten Folgen. Der zugehörige Code ist im Folgenden abgebildet.

```
1  public static int algorithm1(int[] input) {
2      assert  input != null && input.length > 0;
3      int maxSum = input[0];
4      for (int i = 0; i < input.length; i++) {
5          for (int j = i; j < input.length; j++) {
6              int sum = 0;
7              for (int k = i; k <= j; k++) {
8                  sum += input[k];
9              }
10             if (sum > maxSum) {
11                 maxSum = sum;
12             }
13         }
14     }
15     return maxSum;
16 }
```

Geben Sie jeweils in O-Notation an, welche Laufzeit- und Speicherkomplexität der erste Algorithmus hat. Begründen Sie Ihre Antwort.
Laufzeitkomplexität:
.....
.....
Speicherkomplexität:
.....
.....

6 b) (4 Punkte) Ein zweiter Algorithmus erstellt ein Array `p` derselben Länge wie die Eingabe und berechnet im ersten Schritt die Teilsummen von Stelle 0, summiert bei der Iteration also immer ein Element hinzu. Nun kann man die Teilsummen zwischen zwei beliebigen Grenzen `left` und `right` ganz einfach berechnen über `p[right] - p[left]`, wie im folgenden Beispiel dargestellt.

31	-10	49	75	22	80	177	84	61

Abbildung 2: Die zugehörigen Partialsummen zur obigen Folge.

Im zweiten Schritt muss man dann nur noch mit Hilfe des Arrays `p` die maximale Teilsumme finden. Der gesamte Code ist unten abgebildet.

```
1  public static int algorithm2(int[] input) {
2      assert  input != null && input.length > 0;
3      // calculate partial sums
4      int[] p = new int[input.length];
5      for (int i = 0; i < input.length; i++) {
6          if (i == 0) {
7              p[i] = input[i];
8          } else {
9              p[i] = p[i - 1] + input[i];
10         }
11     }
12
13     // search for the maximum partial sum
14     int maxSum = input[0];
15     for (int i = 0; i < input.length; i++) {
16         for (int j = i; j < input.length; j++) {
17             int sum = (i == j) ? p[j] : p[j] - p[i];
18             if (sum > maxSum) {
19                 maxSum = sum;
20             }
21         }
22     }
23     return maxSum;
24 }
```

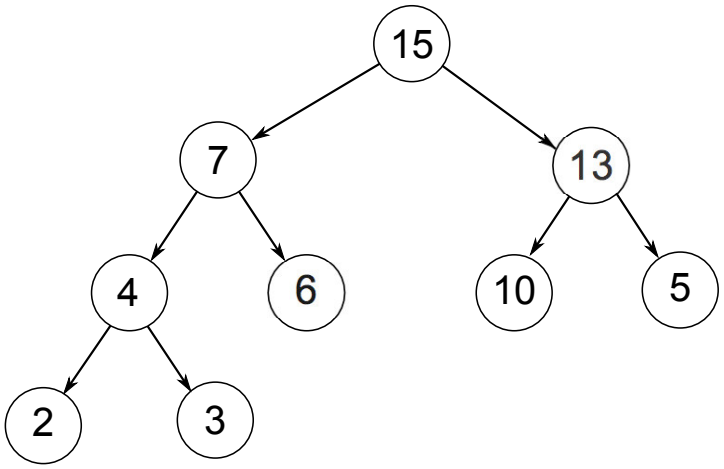
Geben Sie auch hierzu jeweils in O-Notation an, welche Laufzeit- und Speicherkomplexität der erste Algorithmus hat. Begründen Sie Ihre Antwort.
Laufzeitkomplexität:
.....
.....
Speicherkomplexität:
.....
.....

2. Aufgabe: Bäume *(8 Punkte)*

2 a **(3 Punkte)** Gegeben sei der untenstehende Baum. Bewerten Sie die folgenden Aussagen.

Achtung: Falsche Antworten geben einen halben Punkt Abzug!

	wahr	*falsch*
"Der untenstehende Baum ist ein Binärbaum."	☐	☐
"Der untenstehende Baum ist ein binärer Suchbaum."	☐	☐
"Der untenstehende Baum ist ein Heap."	☐	☐
"Manche nichtleere Heaps sind Suchbäume."	☐	☐
"Jeder Suchbaum mit $n > 0$ Knoten hat $n - 1$ Kanten."	☐	☐
"Die Inorder-Traversierung eines binären Suchbaums ergibt eine sortierte Folge."	☐	☐



2 b **(1 Punkt)** Geben Sie das Ergebnis eines Durchlaufs durch den Baum in *Inorder*-Traversierung an:

.....

2 c **(1 Punkt)** Was ist die *minimale* Knotenzahl eines *binären Suchbaums* der Höhe h in Abhängigkeit von h?

.....

2 d **(1 Punkt)** Was ist die *maximale* Zahl der Knoten eines binären *Min-Heaps* der Höhe h in Abhängigkeit von h?

.....

2 e **(2 Punkte)** Zeichnen Sie einen binären Suchbaum, in den die folgenden Zahlen in dieser Reihenfolge eingefügt werden (der Suchbaum sei anfangs leer): **4 – 5 – 8 – 1 – 2 – 3 – 7 – 0**

Aufgabe 5: Bytecode *(4 Punkte)*

Im Folgenden sehen Sie zwei Methoden `expr1()` und `expr2()`, die sich lediglich in der Nutzung der Operatoren "++expr" beziehungsweise "expr++" unterscheiden.

```
1  public class IncrementingInteger {
2
3      public static void main(String[] args) {
4          System.out.println(expr1(1));
5          System.out.println(expr2(1));
6      }
7
8      public static int expr1(int value) {
9          int x = (value++) * 2;
10         return x;
11     }
12
13     public static int expr2(int value) {
14         int x = (++value) * 2;
15         return x;
16     }
17 }
```

5 a) (1 Punkt) Was ist die Ausgabe wenn man das Programm laufen lässt?

.....

5 b) (3 Punkte)

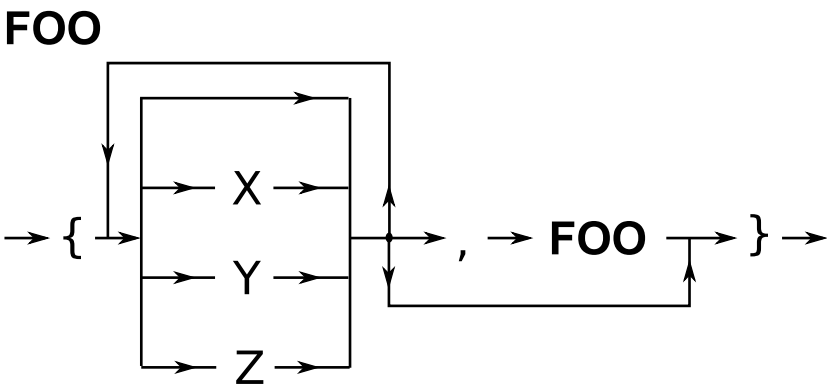
```
Code A:                    Code B:
   0: iload_0                 0: iinc    0, 1
   1: iinc      0, 1          3: iload_0
   4: iconst_2                4: iconst_2
   5: imul                    5: imul
   6: istore_1                6: istore_1
   7: iload_1                 7: iload_1
   8: ireturn                 8: ireturn
```

Sie sehen nun das Bytecode-Fragment für jede dieser beiden Methoden. Welches dieser Fragmente gehört zu `expr1()`, welches zu `expr2()`? Begründen Sie Ihre Antwort, indem Sie auf den Unterschied der Operatoren "++expr" und "expr++" eingehen. (iinc *varnum n*: Erhöht den Wert der Variable mit Referenz *varnum* um *n*)

.....
.....
.....
.....

5. Aufgabe: Syntaxdiagramme und Syntaxchecker *(4 Punkte)*

Gegeben sei das folgende (rekursive) Syntaxdiagramm, welches Strings vom Typ "FOO" erzeugt.



Welche der folgenden Zeichenfolgen sind vom Typ "FOO" und können daher durch das Syntaxdiagramm erzeugt werden? Achtung: Falsche Antworten geben einen halben Punkt Abzug!

	`true`	`false`
$\{XYX\}$	☐	☐
$\{X\{Y\{Z\}\}\}$	☐	☐
$\{X,Y,\{Z\}\}$	☐	☐
$XYZ\}$	☐	☐
$\{XYZ$	☐	☐
$\{$	☐	☐
$\{\}$	☐	☐
$\{XY,\{\}\}$	☐	☐

Aufgabe 4: Parallelität *(9 Punkte)*

Gegeben sei folgendes Programm, in dem fünf Threads erzeugt werden, die eine gemeinsame Zählvariable inkrementieren. Die Inkrement-Operation selbst ist mit einem Sperrobjekt geschützt.

```
1  public class ParIncr extends Thread {
2
3      int i; // individuelle Variable
4      static int j = 0; // gemeinsame Var.
5      static Object Sperre = new Object();
6
7      public void run() {
8          for (i = 0; i < 400000000; i++)
9              synchronized(Sperre) { j++;}
10         System.out.println("i: " + i + " j: " + j);
11     }
12
13     public static void main(String [] args) {
14         ParIncr[] workers = new ParIncr[5];
15         for (int k = 0; k < 5; k++) {
16             workers[k] = new ParIncr();
17             workers[k].start();
18         }
19         for (int k = 0; k < 5; k++) {
20             try {
21                 workers[k].join();
22             } catch (InterruptedException e) {
23                 e.printStackTrace();
24             }
25         }
26     }
27 }
```

4 a) (1 Punkt) Wie lautet das Ergebnis des Programms, d.h. was ist die höchste Ausgabe für `j`?

.....

4 b) (2 Punkte) Wie würde das Ergebnis höchstwahrscheinlich ausfallen, wenn man keine Sperre verwenden würde? Begründen Sie Ihre Antwort.

.....
.....
.....

4 c) (2 Punkte) Was ist zu erwarten, wenn man `synchronized(Sperre)` in Zeile 9 mit `synchronized(this)` ersetzen würde? Wieso würde sich das Ergebnis mit hoher Wahrscheinlichkeit ändern?

.....
.....
.....

4 d) (2 Punkte) Was ist zu erwarten, wenn man das `static` bei `static Object Sperre = new Object();` weglassen würde? Würde sich das Ergebnis mit hoher Wahrscheinlichkeit ändern? Begründen Sie Ihre Antwort.

.....
.....
.....

4 e) (2 Punkte) Angenommen, Sie testen das Programm auf einem Computer mit vier CPU-Kernen und messen die Zeit. Dann schreiben Sie das Programm um, sodass nur noch ein einziger Thread erzeugt wird, der dafür aber bis zu einer fünf mal höheren Zahl inkrementiert, also insgesamt die gleiche Arbeit verrichtet. Dabei bemerken Sie, dass die Variante mit nur einem einzigen Thread deutlich schneller ist als mit fünf, obwohl vier Kerne zur Verfügung stehen. Wieso kann dieser Fall auftreten?

.....
.....
.....
.....

Aufgabe 2: Programmverifikation *(7 Punkte)*

In der Vorlesung haben Sie das Verfahren der altägyptischen Multiplikation kennengelernt und auch einen Beweis mittels einer Schleifeninvariante. Ähnlich zur Multiplikation kann man auch die Potenzierung durchführen. In dieser Aufgabe sollen Sie das zugehörige Verfahren implementieren und den Beweis der Korrektheit führen.

Die altägyptische Multiplikation ist wie folgt definiert (auf $\mathbb{N}^+$):

$$f_m(a,b) = \begin{cases} a, \text{falls } b = 1 \\ f_m(2a, b/2), \text{falls } b \text{ gerade} \\ a + f_m(2a, (b-1)/2), \text{sonst} \end{cases}$$

2 a) (3 Punkte) Unten sehen Sie den Java-Code für die altägyptische Multiplikation, wie in der Vorlesung angegeben. Schreiben Sie den Code, damit a mit b potenziert wird, anstatt dass a mit b multipliziert wird.

```
1  public static int mult(int i, int j) {
2      int a = i;
3      int b = j;
4      int z = 0;
5      while (b > 0) {
6          if (b % 2 != 0) {
7              z = z + a;
8              b = b - 1;
9          }
10         b = b / 2;
11         a = 2 * a;
12     }
13     return z;
14 }
```

```
1  public static int power(int i, int j) {
2
3  ..................................................
4
5  ..................................................
6
7  ..................................................
8
9  ..................................................
10
11 ..................................................
12
13 ..................................................
14
15 ..................................................
16
17 ..................................................
18
19 ..................................................
20
21 ..................................................
22
23 ..................................................
24 }
```

2 b) (2 Punkte) Geben Sie eine Schleifeninvariante für die Schleife zur Potenzierung an und begründen Sie, warum dies eine Invariante ist.

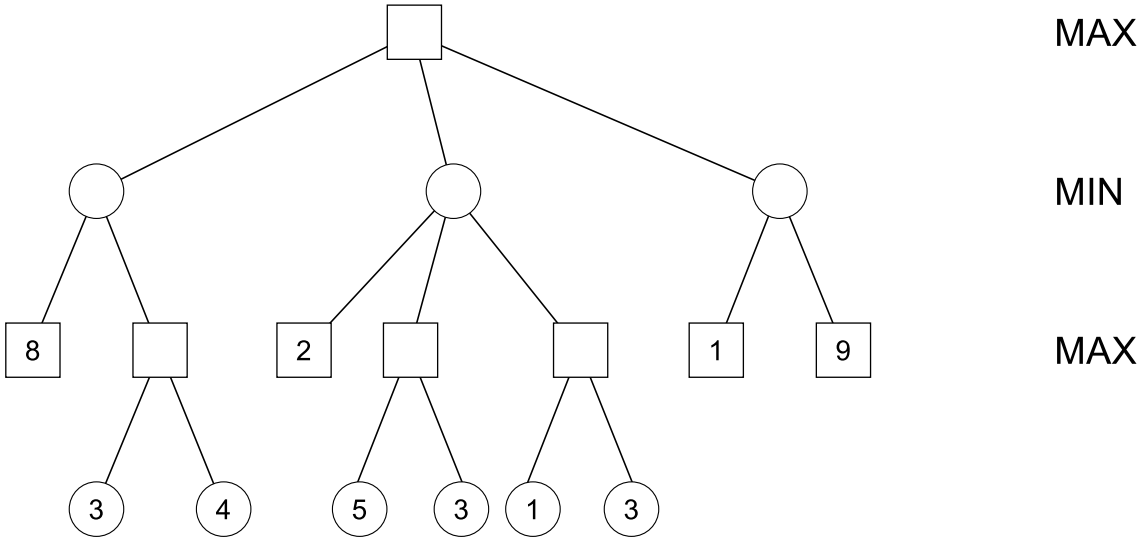
.....
.....
.....
.....

2 c) (2 Punkte) Beweisen Sie mit Hilfe der Schleifeninvariante die Korrektheit des Algorithmus.

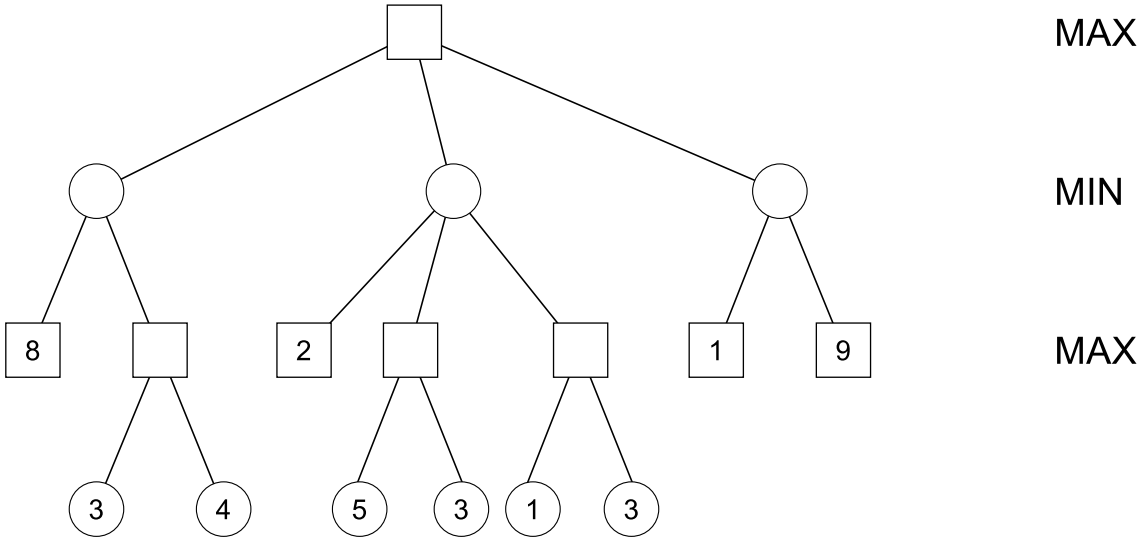
.....
.....
.....
.....

1. Aufgabe: Bäume *(13 Punkte)*

1 a **(1 Punkte)** Gegeben sei der untenstehende Spielbaum. Führen Sie eine Bewertung aller Knoten nach dem *Minimax*-Algorithmus durch.



1 b **(3 Punkte)** Gegeben sei der untenstehende Spielbaum (es ist derselbe Spielbaum wie in Aufgabe 1a). Führen Sie eine Bewertung aller Knoten nach dem α-β-Algorithmus durch. Zeichnen Sie ausserdem **sämtliche Schnitte** ein.



1 c **(4 Punkte)**

Geben Sie an, ob die folgenden Aussagen wahr oder falsch sind.

Achtung: Falsche Antworten geben einen halben Punkt Abzug.

	wahr	falsch
Der Alpha-Beta-Algorithmus und der Minimax-Algorithmus liefern immer den selben Gewinnwert der Wurzel.	☐	☐
Der Parameter α des Alpha-Beta-Algorithmus ist eine obere Schranke für den Gewinnwert des MAX-Spielers.	☐	☐
Der Parameter β des Alpha-Beta-Algorithmus kann zum Wegfall von tiefer liegenden Zügen des MAX-Spielers führen.	☐	☐
Die Evaluationsreihenfolge kann einen wesentlichen Einfluss auf die Geschwindigkeit des Alpha-Beta-Algorithmus haben.	☐	☐