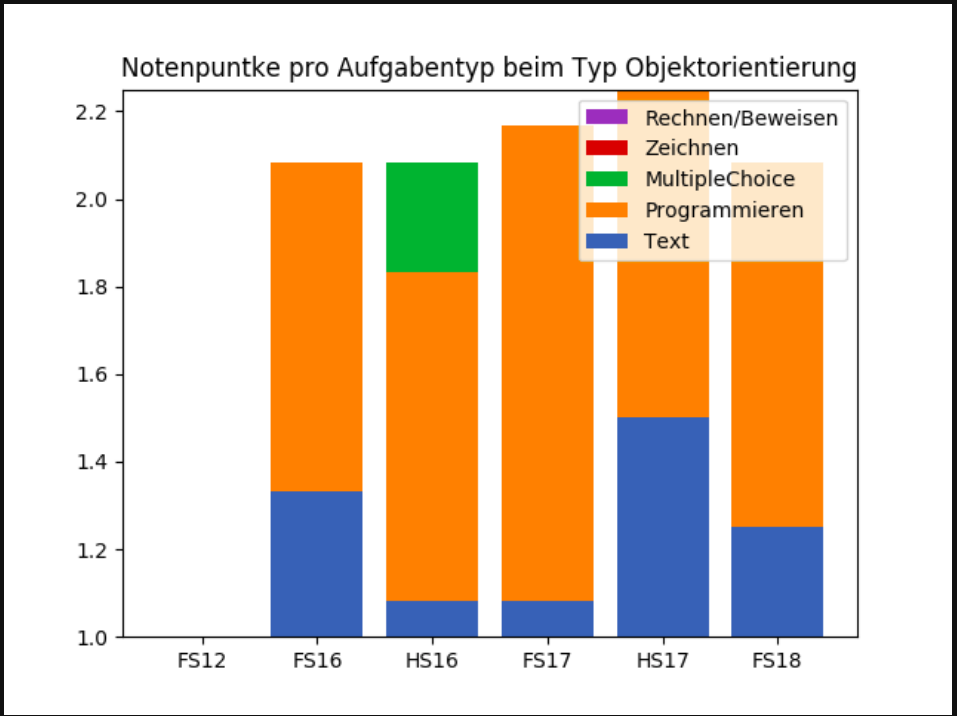
Prüfungsaufgabe

Objektorientierung



Herbst 2017, 1.17 Notenpunkte

Prüfungsaufgabe

Objektorientierung

Herbst 2017, 1.17 Notenpunkte

```java
public class Activity{
    private String name;
    private double duration;
    private double rating;

    public Activity(String name, double duration, double rating) {
        if(duration <= 0) {
            throw new IllegalArgumentException("The duration can not be negative!");
        }
        if(rating < 0 || rating > 5) {
            throw new IllegalArgumentException("The rating has to be between 0 and 5!");
        }
        this.name = name;
        this.duration = duration;
        this.rating = rating;
    }
    public String getName() {
        return name;
    }
    public double getDuration() {
        return duration;
    }
    public double getRating() {
        return rating;
    }
}
```

Prüfungsaufgabe

Objektorientierung

Herbst 2017, 1.17 Notenpunkte

```java
public class Hike extends Activity {
    private double difficulty;

    public Hike(String name, double duration, double rating,
            double difficulty) {
        super(name, duration, rating);
        if(difficulty < 1 || difficulty > 5) {
            throw new IllegalArgumentException("The difficulty has to "
                    + "be between 1 and 5!");
        }
        this.difficulty = difficulty;
    }

    public double getDifficulty() {
        return difficulty;
    }
}
```

Prüfungsaufgabe

Objektorientierung

Herbst 2017, 1.17 Notenpunkte

```java
import java.util.ArrayList;

public class ActivityManager{
    ArrayList<Activity> activities;

    public ActivityManager() {
        activities = new ArrayList<Activity>();
    }

    public void addActivitiy(Activity activity){
        activities.add(activity);
    }

    public Hike findBestHike(double maxDuration, double maxDifference){
        // [..]
    }
}
```

Prüfungsaufgabe

Objektorientierung

Herbst 2017, 1.17 Notenpunkte

```java
public Hike findBestHike(double maxDuration, double maxDifficulty){
    Hike bestHike = null;
    for (Activity activity : activities) {
      if (activity instanceof Hike) {
        Hike hike = (Hike) activity;
        if (hike.getDuration() <= maxDuration
            && hike.getDifficulty() <= maxDifficulty) {
          if (bestHike == null
              || hike.getRating() > bestHike.getRating()) {
            bestHike = hike;
          }
        }
      }
    }
    return bestHike;
}
```

Prüfungsaufgabe

Objektorientierung

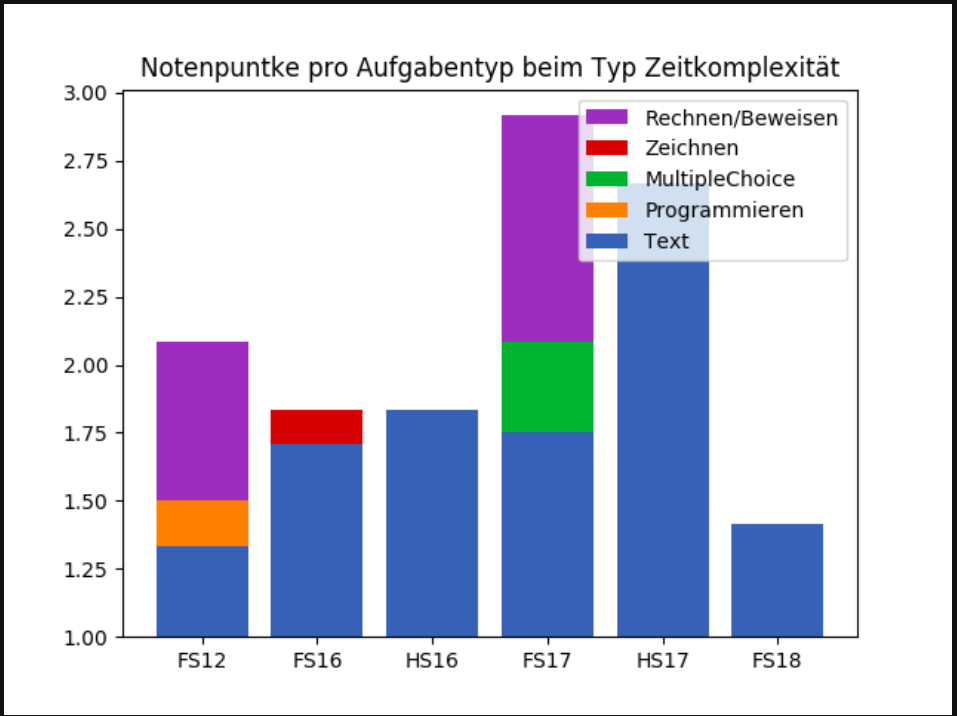
Herbst 2017, 1.17 Notenpunkte

```java
public class Main{
    public static void main(String[] args){
        ActivityManager = new ActivityManager();
        // Adding activities
        // Assume the parameter order for the Activity constructor is
        // [name, duration, rating] and for the Hike constructor
        // [name, duration, rating, difficulty]
        manager.addActivitiy(new Activity("Spaziergang am See", 1.2, 2.5));
        manager.addActivitiy(new Hike("Brunni - Kl. Mythen", 2, 4.5, 4.5));
        manager.addActivitiy(new Activity("Velofahren Zuerich", 0.5, 1));
        manager.addActivitiy(new Hike("Brunni - Gr. Mythen", 3, 3.5, 3.5));
        manager.addActivitiy(new Activity("Schwimmen am Letten", 1, 4.5));
        manager.addActivitiy(new Activity("Nichtstun", 0, 2.5));
        manager.addActivitiy(new Hike("Weggis - Rigi Kulm", 5, 4, 3.5));
        manager.addActivitiy(new Hike("Alpnach - Pilatus Kulm", 5, 5, 2));

        double maxDuration = 4;
        double maxDifficulty = 3.5;
        Hike bestHike = manager.findBestHike(maxDuration, maxDifference);

        System.out.println("Die beste Wanderung: " + bestHike.getName());
    }
}
```

```
Die beste Wanderung: Brunni - Gr. Mythen
```

Prüfungsaufgabe

Laufzeit und Speicherkomplexität



Herbst 2017, 0.67 Notenpunkte

Prüfungsaufgabe

Laufzeit und Speicherkomplexität

Herbst 2017, 0.67 Notenpunkte

```java
public static int algorithm1(int[] input){
    assert input != null && input.length > 0;
    int maxSum = input[0];
    for(int i = 0; i < input.length; i++){
        for(int j = 0; j < input.length; j++){
            int sum = 0;
            for(int k = i; k <= j; k++){
                sum += input[k];
            }
            if(sum > maxSum){
                maxSum = sum;
            }
        }
    }
    return maxSum;
}
```

Zeit:

$O(input.length^3)$

Zusätzlicher Speicher:

$O(1)$

Prüfungsaufgabe

Laufzeit und Speicherkomplexität

Herbst 2017, 0.67 Notenpunkte

```java
public static int algorithm2(int[] input){
    assert input != null && input.length > 0;
    // calculate partial sums
    int[] p = new int[input.length];
    for(int i = 0; i < input.length; i++){
        if(i == 0) p[i] = input[i];
        else p[i] = p[i-1] + input[i];
    }
    // search for the maximum partial sum
    int maxSum = input[0];
    for(int i = 0; i < input.length; i++){
        for(int j = i; j < input.length; j++){
            int sum = (i == j) ? p[j] : p[j] - p[i];
            if(sum > maxSum){
                maxSum = sum;
            }
        }
    }
    return maxSum;
}
```

Zeit:

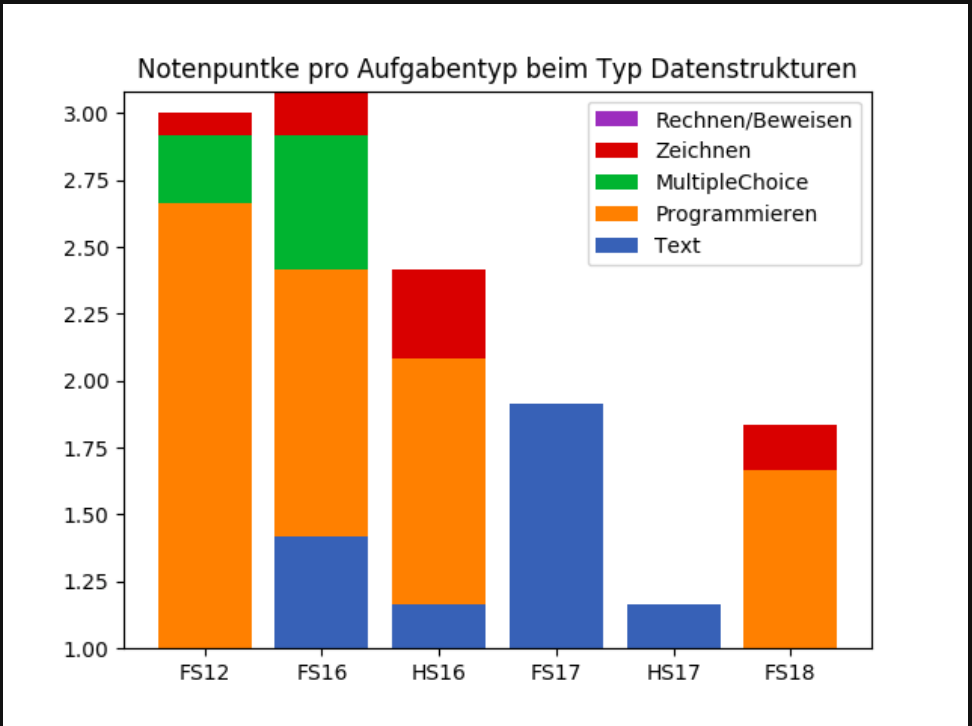
$O(input.length^2)$

Zusätzlicher Speicher:

$O(input.length)$

Prüfungsaufgabe

Bäume

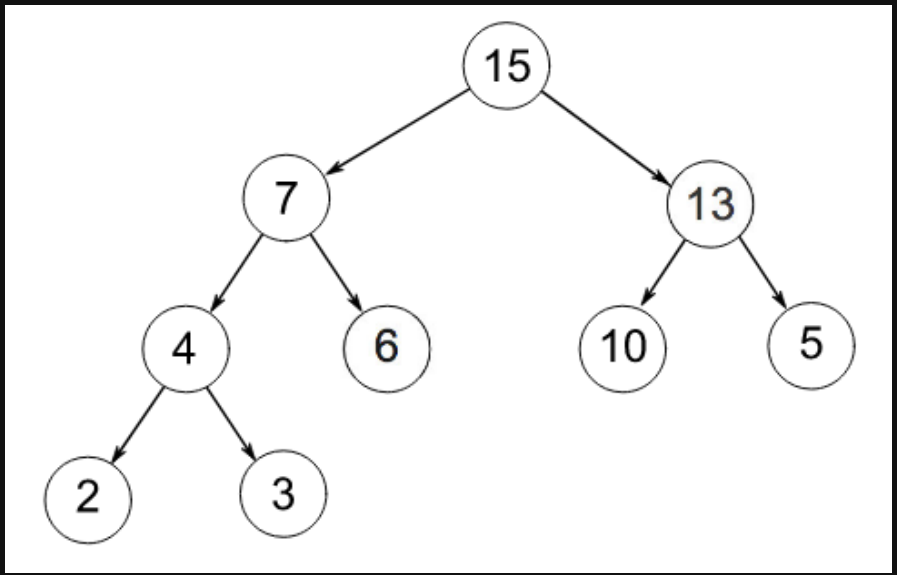


Frühling 2016, 0.67 Notenpunkte

Prüfungsaufgabe

Bäume

Frühling 2016, 0.67 Notenpunkte



Inorder Traversierung:

2 4 3 7 6 15 10 13 5

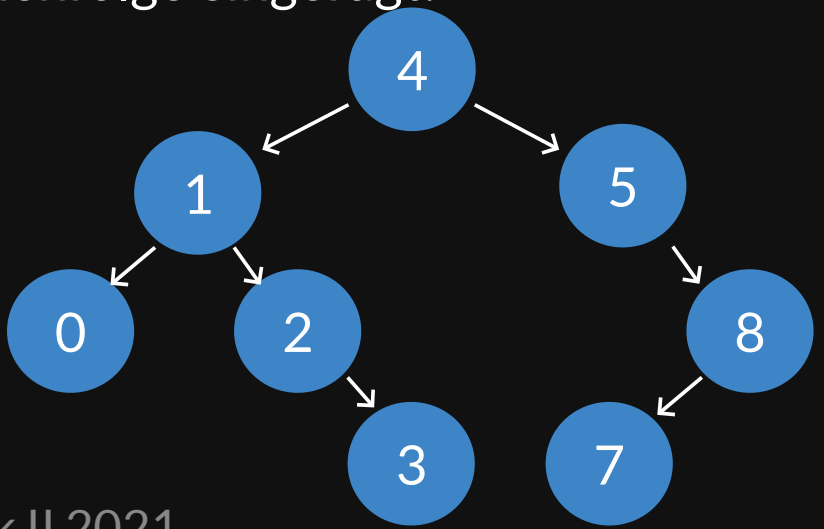
- Der Baum ist ein Binärbaum
- Er ist kein Binärer Suchbaum
- Er ist ein (Max-)Heap
- Es gibt nichtleere Heaps, welche Suchbäume sind
- Jeder (Such-)Baum mit n>0 Knoten hat n-1 Kanten
- Die Inorder Traversierung von einem Binären Suchbaum gibt immer eine sortierte Folge

Prüfungsaufgabe

Bäume

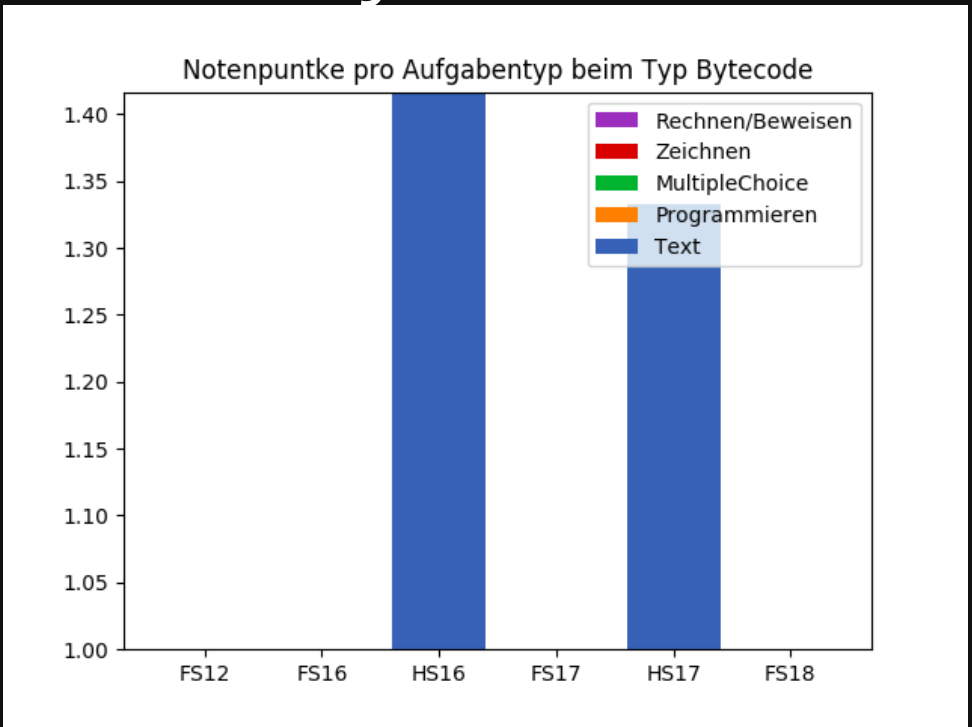
Frühling 2016, 0.67 Notenpunkte

- Minimale Knotenzahl eines binären Suchbaums der Höhe h
  - $h + 1$ (Baum mit 1 Knoten hat Höhe 0)
- Maximale Knotenzahl bei Max-Heap der Höhe h
  - $2^{h+1} - 1$
- Elemente in dieser Reihenfolge eingefügt: 4 – 5 – 8 – 1 – 2 – 3 – 7 – 0



Prüfungsaufgabe

Bytecode



Herbst 2017, 0.34 Notenpunkte

Prüfungsaufgabe

Bytecode

Herbst 2017, 0.34 Notenpunkte

```java
public class IncrementingInteger{
    public static void main(){
        System.out.println(expr1(1));
        System.out.println(expr2(1));
    }

    public static int expr1(int value){
        int x = (value++)*2;
        return x;
    }

    public static int expr2(int value){
        int x = (++value)*2;
        return x;
    }
}
```

```
2
4
```

```
1  Code A:
2       0: iload_0
3       1: iinc      0, 1
4       2: iconst_2
5       3: imul
6       4: istore_1
7       5: iload_1
8       6: ireturn
```

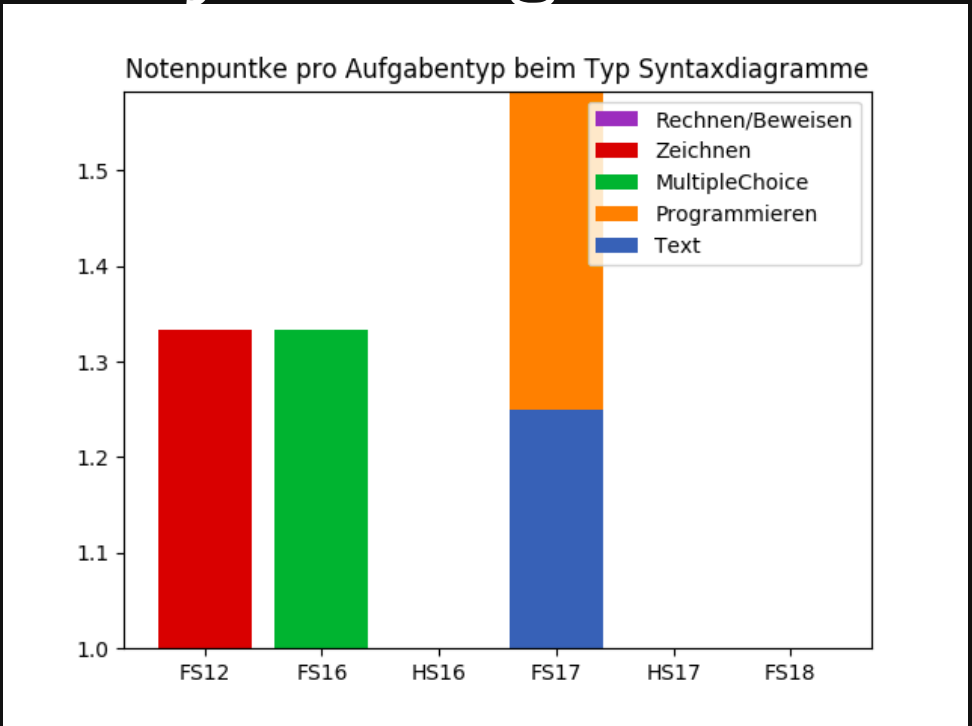
expr1

```
1  Code B:
2       0: iinc      0, 1
3       1: iload_0
4       2: iconst_2
5       3: imul
6       4: istore_1
7       5: iload_1
8       6: ireturn
```

expr2

Prüfungsaufgabe

Syntaxdiagramme

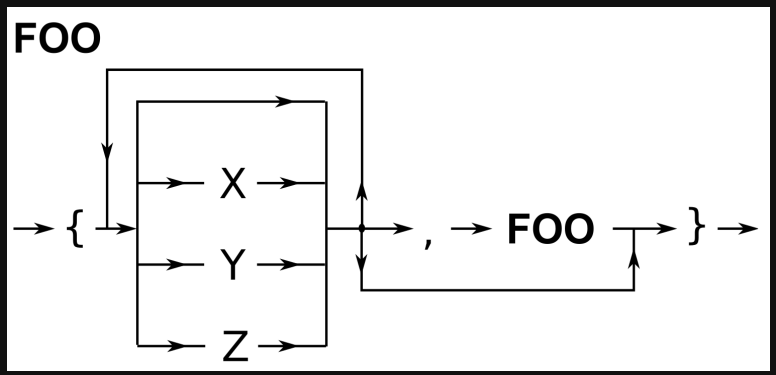


Frühling 2016, 0.35 Notenpunkte

Prüfungsaufgabe

Syntaxdiagramme

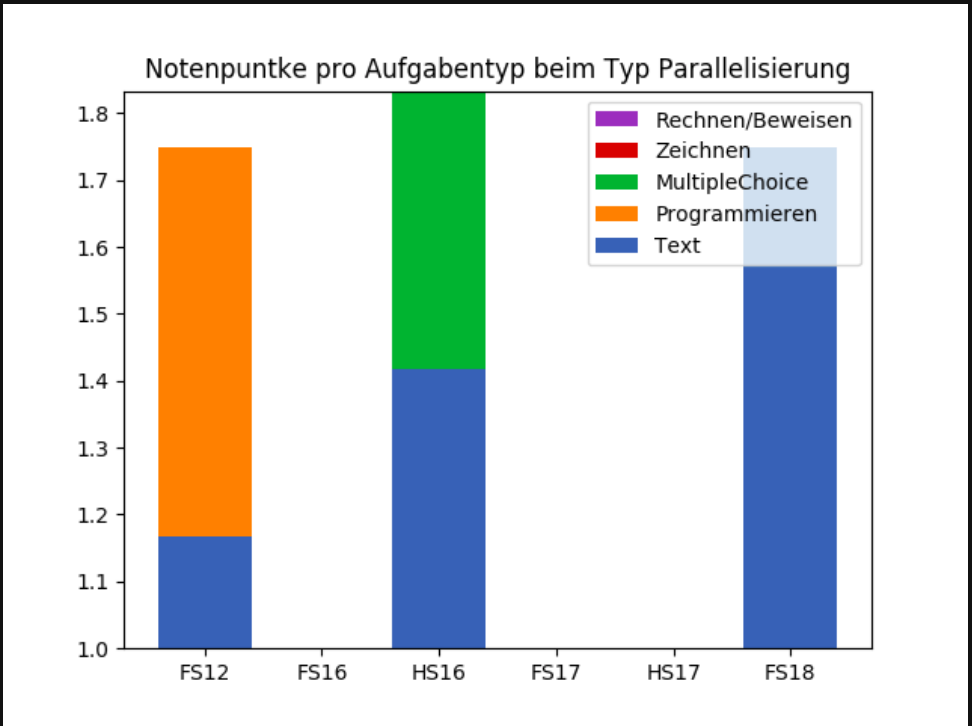
Frühling 2016, 0.35 Notenpunkte



- `{XYZ}` ✓
- `{X{Y{Z}}}` ✗
- `{X,Y,{Z}}` ✗
- `XYZ}` ✗
- `{XYZ` ✗
- `{` ✓
- `{}` ✓
- `{XY,{}}` ✗

Prüfungsaufgabe

Parallelität



Frühling 2018, 0.75 Notenpunkte

Prüfungsaufgabe

Parallelität

Frühling 2018, 0.75 Notenpunkte

```java
public class ParIncr extends Thread {
    int i; // individuelle Variable
    static int j; // gemeinsame Var.
    static Object Sperre = new Object();

    public void run() {
        for (i = 0; i < 400000000; i++) {//400'000'000
            synchronized (Sperre) {
                j++;
            }
        }
        System.out.println("i: " + i + " j: " + j);
    }

    public static void main(String[] args) {
        ParIncr[] workers = new ParIncr[5];
        for (int k = 0; k < 5; k++) {
            workers[k] = new ParIncr();
            workers[k].start();
        }
        for (int k = 0; k < 5; k++) {
            try {
                workers[k].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- Höchste Ausgabe für j:
  - 2'000'000'000
- Ohne Sperre:
  - i bleibt gleich, aber j würde die 2'000'000'000 nicht erreichen.
- "Sperre" mit "this ersetzen:
  - "this" ist für jeden Thread ein anderes Objekt, somit können die Threads trotz synchronized gleichzeitig auf j zugreifen.

Prüfungsaufgabe

Parallelität

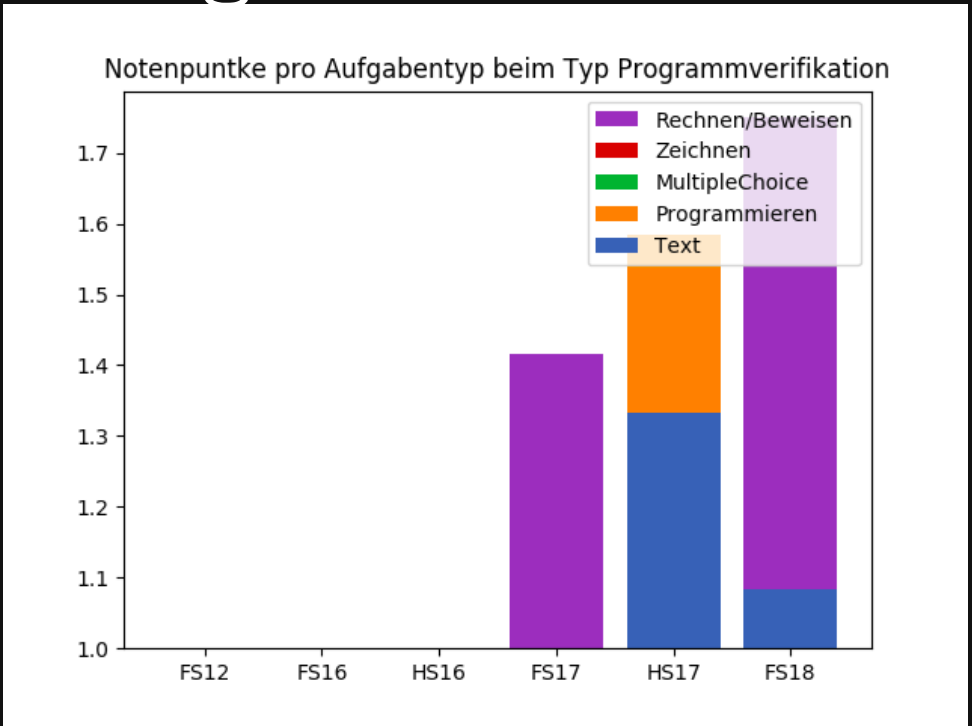
Frühling 2018, 0.75 Notenpunkte

```java
public class ParIncr extends Thread {
    int i; // individuelle Variable
    static int j; // gemeinsame Var.
    static Object Sperre = new Object();

    public void run() {
        for (i = 0; i < 400000000; i++) {//400'000'000
            synchronized (Sperre) {
                j++;
            }
        }
        System.out.println("i: " + i + " j: " + j);
    }

    public static void main(String[] args) {
        ParIncr[] workers = new ParIncr[5];
        for (int k = 0; k < 5; k++) {
            workers[k] = new ParIncr();
            workers[k].start();
        }
        for (int k = 0; k < 5; k++) {
            try {
                workers[k].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- "static" bei "Sperre" weglassen:
  - Da nun "Sperre" für jeden Thread ein anderes Objekt ist, können die Threads trotz synchronized gleichzeitig auf j zugreifen.
- Durch das Lock können die extra Prozessoren nicht parallel genutzt werden und es gibt einen Overhead beim Wechsel zwischen den einzelnen Threads.

Prüfungsaufgabe

Programmverifikation



Herbst 2017, 0.58 Notenpunkte

Prüfungsaufgabe

Programmverifikation

Herbst 2017, 0.58 Notenpunkte

```java
public static int mult(int i, int j) {
    int a = i;
    int b = j;
    int z = 0;
    while(b > 0) {
        if(b%2 != 0) {
            z = z + a;
            b = b - 1;
        }
        b = b / 2;
        a = 2 * a;
    }
    return z;
}
```

```java
public static int power(int i, int j) {
    int a = i;
    int b = j;
    int z = 1;
    while(b > 0) {
        if(b%2 != 0) {
            z = z * a;
            b = b - 1;
        }
        b = b / 2;
        a = a * a;
    }
    return z;
}
```

Prüfungsaufgabe

Programmverifikation

Herbst 2017, 0.58 Notenpunkte

- Theorietisch wäre hier jede Schleifeninvariante akzeptiert, also auch "z = z"
- Es muss einfach folgendes gelten:
  Falls die Invariante vor dem Schleifenkörper gilt, so muss sie auch nach dem Schleifenkörper gelten
- Eine Invariante, mit welcher der Beweis nachher funktioniert wäre
  $z \cdot a^b = i^j$

Prüfungsaufgabe

Programmverifikation

Herbst 2017, 0.58 Notenpunkte

Schleifeninvariante $z \cdot a^b = i^j$

- Fall 1: $b\%2 = 0$
  - $z_{n+1} = z_n$
  - $a_{n+1} = a_n^2$
  - $b_{n+1} = \frac{b_n}{2}$
  - $z_{n+1} \cdot a_{n+1}^{b_{n+1}} = z_n \cdot a_n^{(2 \cdot \frac{b_n}{2})}$
    $= z_n \cdot a_n^{b_n} = i^j$

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

Prüfungsaufgabe

Programmverifikation

Herbst 2017, 0.58 Notenpunkte

Schleifeninvariante $z \cdot a^b = i^j$

- Fall 2: $b\%2 \neq 0$
  - $z_{n+1} = z_n * a_n$
  - $a_{n+1} = a_n^2$
  - $b_{n+1} = \frac{b_n - 1}{2}$
  - $z_{n+1} \cdot a_{n+1}^{b_{n+1}} = z_n \cdot a_n \cdot a_n^{(2 \cdot \frac{(b_n-1)}{2})}$
    $= z_n \cdot a_n^{b_n} = i^j$

```java
public static int power(int i, int j) {
    int a = i;
    int b = j;
    int z = 1;
    while(b > 0) {
        if(b%2 != 0) {
            z = z * a;
            b = b - 1;
        }
        b = b / 2;
        a = a * a;
    }
    return z;
}
```

$\Rightarrow$ somit haben wir in beiden Fällen gezeigt, dass falls die Schleifeninvariante vor dem Schleifenkörper gilt, sie auch nach dem Schleifenkörper gelten muss.

Prüfungsaufgabe

Programmverifikation

Herbst 2017, 0.58 Notenpunkte

- Jetzt müssen wir zeigen, dass die Invariante vor der Schleife gilt:
  - $a = i$
  - $b = j$
  - $z = 1$
- $z \cdot a^b = i^j$
  - $1 \cdot i^j = i^j$
- Somit gilt sie auch nach der Schleife

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

Prüfungsaufgabe

Programmverifikation

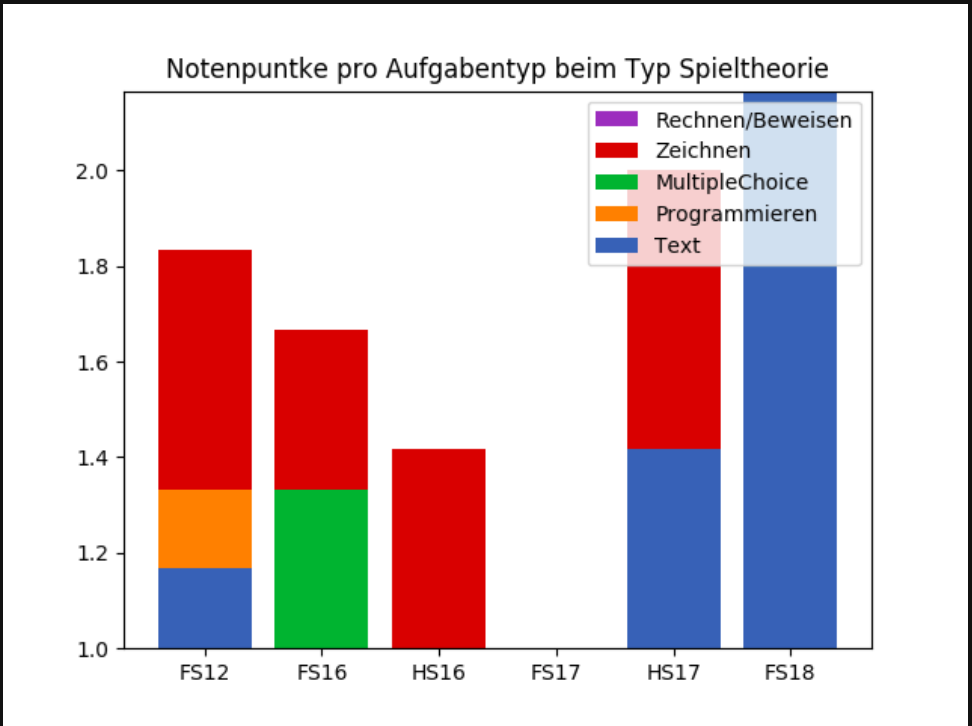
Herbst 2017, 0.58 Notenpunkte

- Nach der Schleife gilt also die Schleifeninvariante immernoch: $z \cdot a^b = i^j$
- Zusätzlich wissen, wir dass die Schleife abgebrochen hat, daher wissen wir, dass $b = 0$
- Zusammen ergibt sich $z \cdot a^0 = z = i^j$

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

Prüfungsaufgabe

Bäume

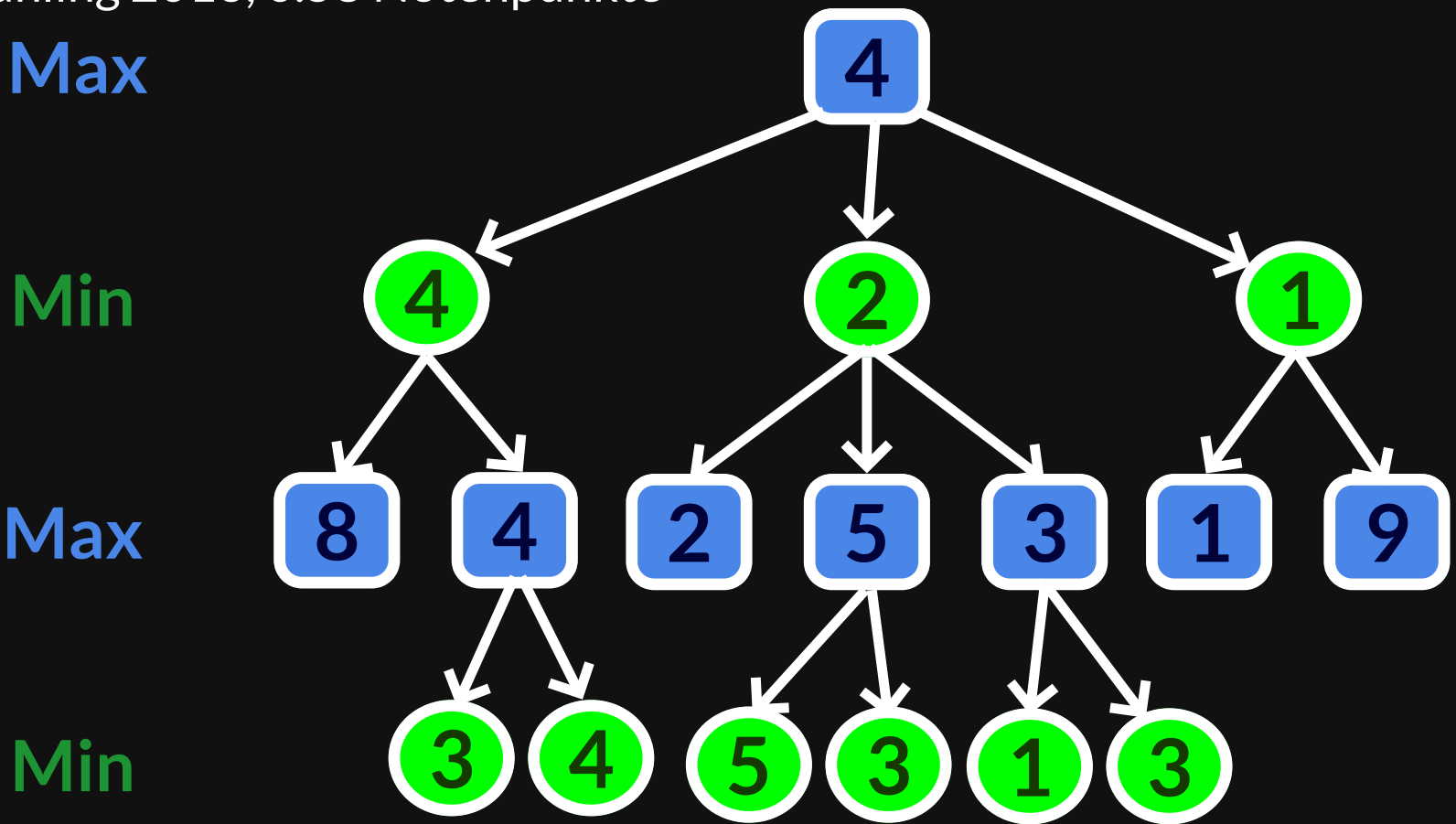


Frühling 2016, 0.58 Notenpunkte

Prüfungsaufgabe

Bäume

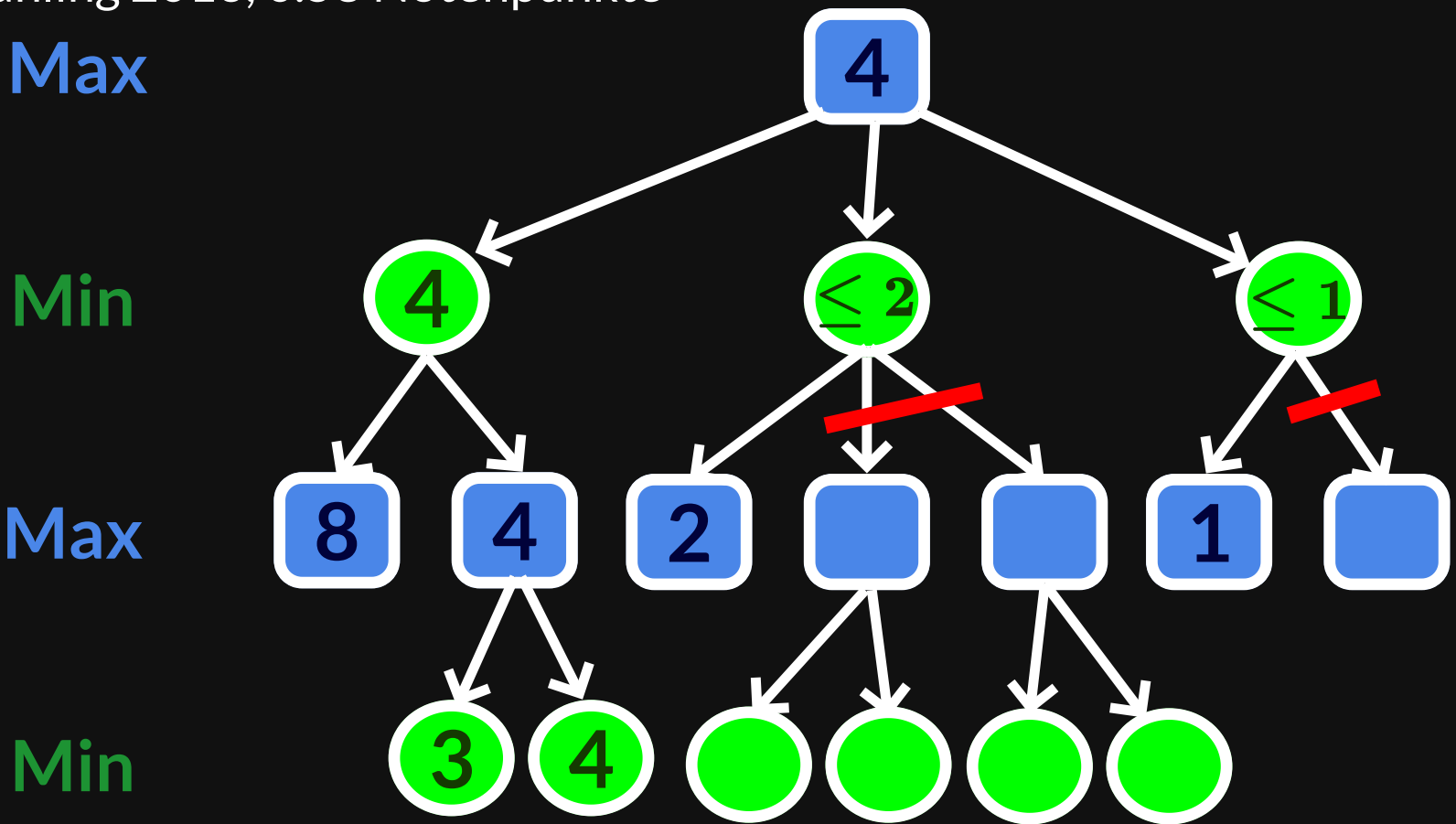
Frühling 2016, 0.58 Notenpunkte



Prüfungsaufgabe

Bäume

Frühling 2016, 0.58 Notenpunkte



Prüfungsaufgabe

Bäume

Frühling 2016, 0.58 Notenpunkte

- Der Alpha-Beta-Algorithmus und der Minimax-Algorithmus liefern immer den selben Gewinnwert der Wurzel.
  - **wahr** - Alpha-Beta schneidet nur Äste ab, welche für den Wert der Wurzel irrelevant sind.

- Der Parameter α des Alpha-Beta-Algorithmus ist eine obere Schranke für den Gewinnwert des MAX-Spielers.
  - **falsch** - Es ist eine untere Schranke

Prüfungsaufgabe

Bäume

Frühling 2016, 0.58 Notenpunkte

- Der Parameter β des Alpha-Beta-Algorithmus kann zum Wegfall von tiefer liegenden Zügen des MAX-Spielers führen.
  - **wahr** - Unter einem cut können immer sowohl MIN als auch MAX Knoten sein.

- Die Evaluationsreihenfolge kann einen wesentlichen Einfluss auf die Geschwindigkeit des Alpha-Beta-Algorithmus haben.
  - **wahr** - Wenn die besten Knoten am Anfang ausgewertet werden hat man früher ein engeres Alpha-Beta Intervall und kann somit mehr Schnitte machen.