

Informatik II - Übung 12

Pascal Schärli

CheapHeap@pascscha.ch

11.12.2020

ETH zürich

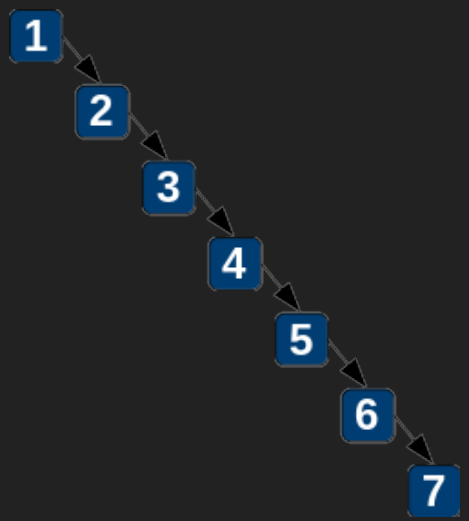
Nachbesprechung



Sortieren mit Suchbäumen

aufsteigend vorsortiert

```
[1, 2, 3, 4, 5, 6, 7]
```

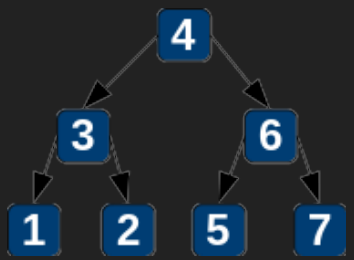


worst case:

$O(n^2)$

gut durchmischt

```
[4, 3, 6, 7, 1, 2, 5]
```

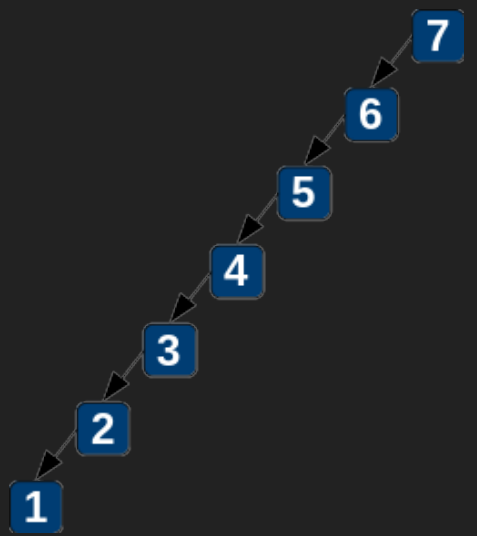


best & average case:

$O(n \cdot log_2(n))$

absteigend vorsortiert

```
[7, 6, 5, 4, 3, 2, 1]
```



worst case:

$O(n^2)$

Komplexitätsanalyse und O-Notation

```
1 // Fragment 6
2 for (int i=0; i<n; i++)
3     for (int j=0; j<n*n; j++)
4         for (int k=0; k<j; k++)
5             a++;
```

- Zeile 2 macht, dass die inneren Loops n-mal ausgeführt werden.
- Zeile 3 macht, dass der inneren Loop n^2-mal ausgeführt wird.
- Der letzte Loop führt **a++** j mal aus. Da j von 0 bis n^2 geht, hat der innerste Loop auch die Komplexität $O(n^2)$.
- Somit hat das gesamte Fragment die Komplexität $O(n * n^2 * n^2) = O(n^5)$

Komplexität

Laufzeit	max. Grösse alt	max. Grösse neu
n	M_1	$M_1' = 3M_1$
n^2	M_2	$M_2' = \sqrt{3}M_2$
2^n	M_3	$M_3' = M_3 + \log_2 3$
$\log n$	M_4	$M_4' = (M_4)^3$

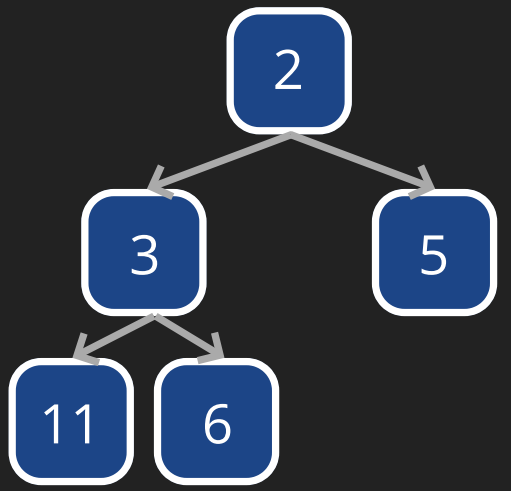
- $M_i' = f_i^{-1}(k \cdot f_i(M_i))$
- $M_3' = log_2(3 \cdot 2^{M_3}) = log_2(3) + M_3 \neq log_2(3) \cdot M_3$
- $M_4' = 2^{3 \cdot log_2(M_4)} = M_4^3 \neq 2^3 \cdot M_4$

Best of Vorlesung

Heaps

- Wie in der letzten Serie gesehen hat man bei Suchbäumen das Problem, dass diese entarten könnten.
  - $\Rightarrow$ Heaps

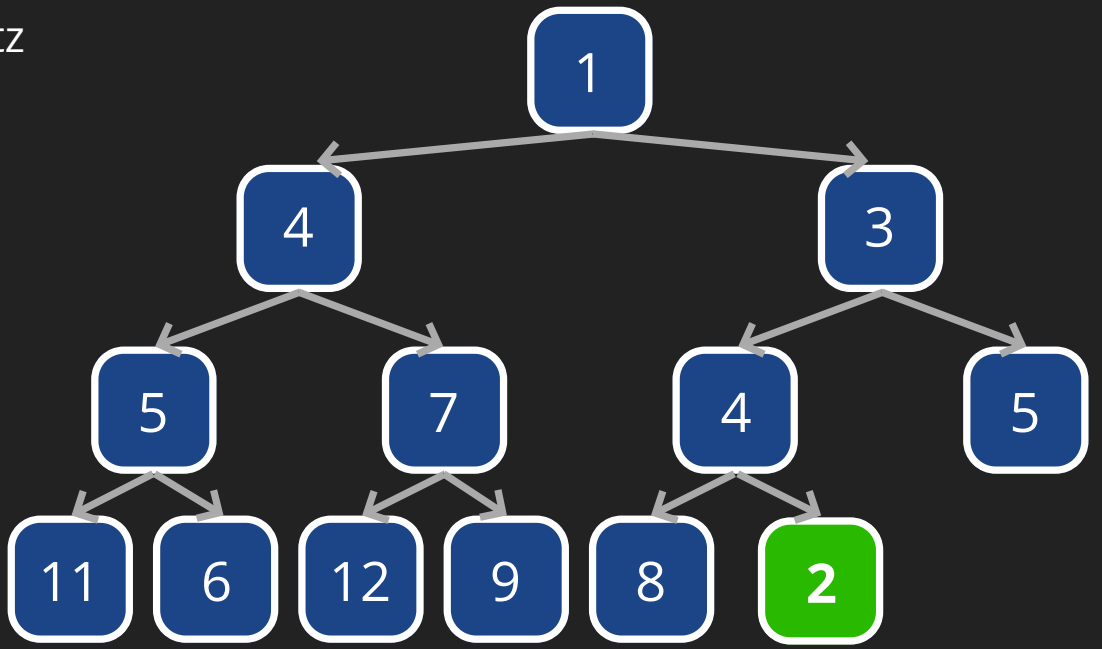
- Ein Heap ist ein Binärbaum mit den folgenden Eigenschaften:
  1. Alle bis auf die letzte Ebene sind vollständig besetzt
  2. Die letzte Ebene wird von rechts nach links aufgefüllt
  3. Die Elemente sind niveauweise sortier:
     - Max-Heap: Alle Kinder sind kleiner als die Wurzel
     - Min-Heap: Alle Kinder sind grösser als die Wurzel



Heaps

Element einfügen

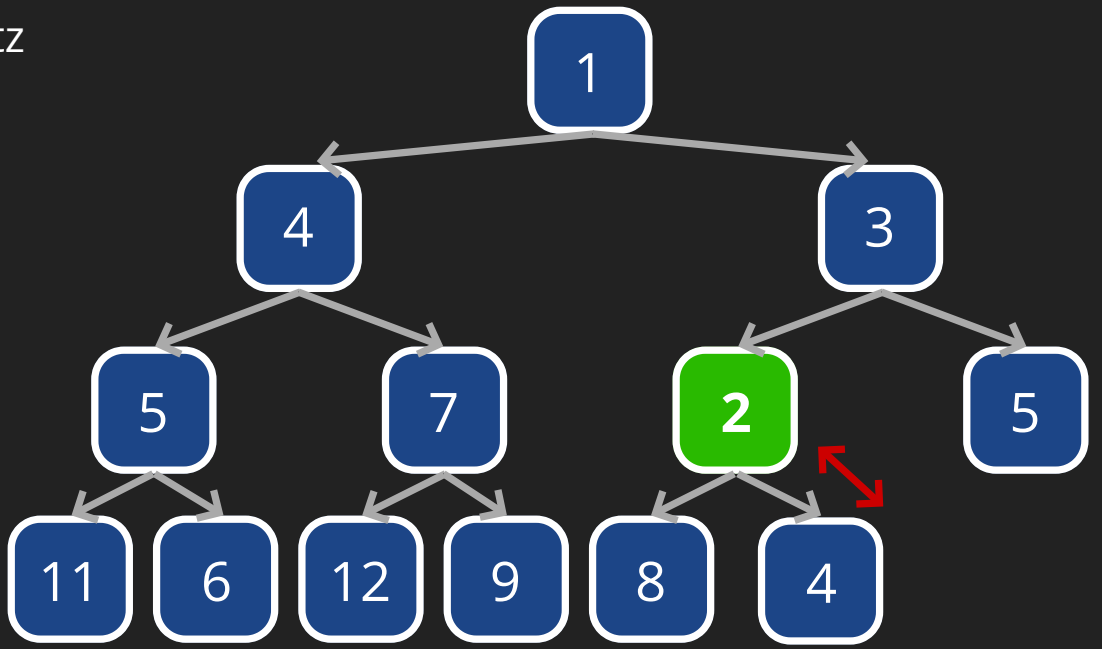
- Um ein Element einzufügen, fügt man es in der Untersten Ebene beim ersten freien Platz ein.
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird das Element so lange mit seinem Vorgänger getauscht, bis es grösser als sein Vorgänger ist.



Heaps

Element einfügen

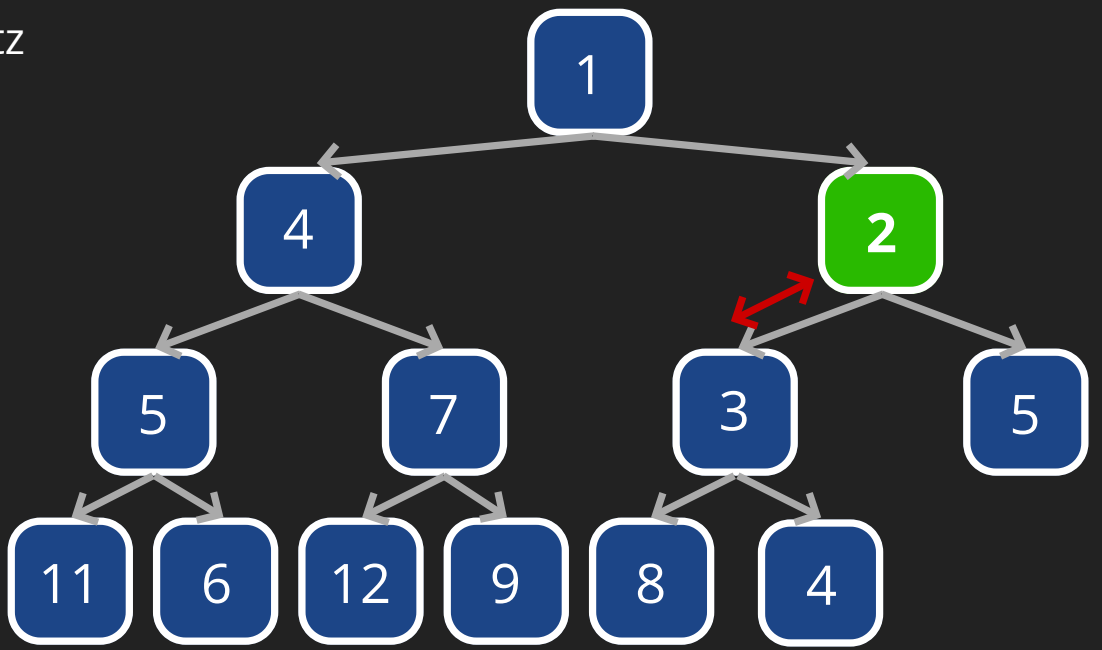
- Um ein Element einzufügen, fügt man es in der Untersten Ebene beim ersten freien Platz ein.
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird das Element so lange mit seinem Vorgänger getauscht, bis es grösser als sein Vorgänger ist.



Heaps

Element einfügen

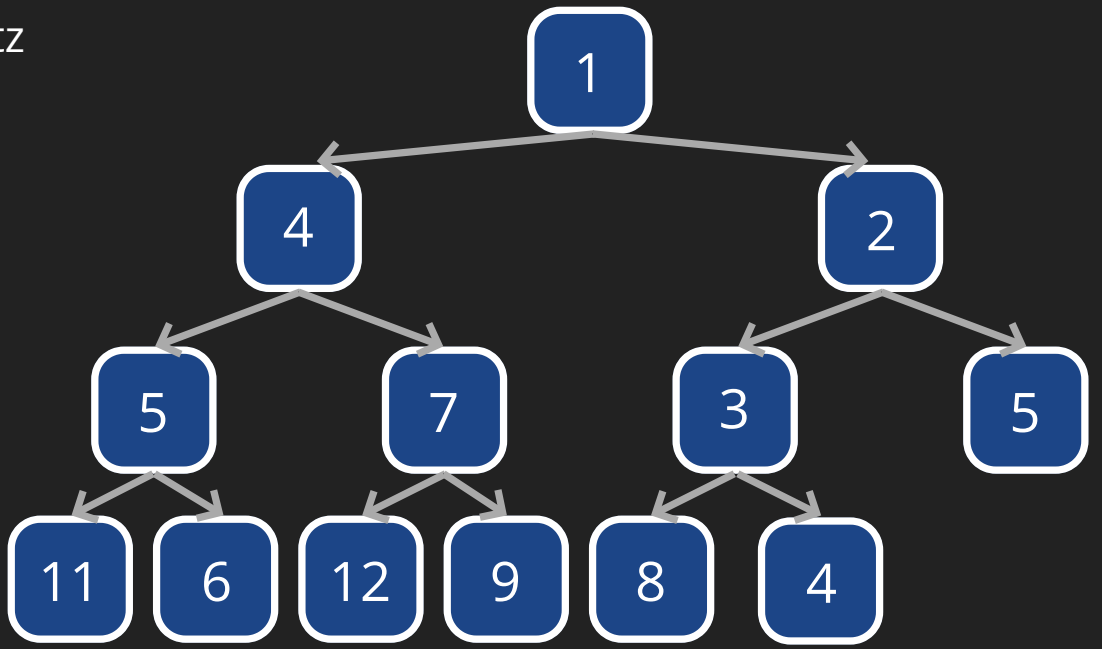
- Um ein Element einzufügen, fügt man es in der Untersten Ebene beim ersten freien Platz ein.
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird das Element so lange mit seinem Vorgänger getauscht, bis es grösser als sein Vorgänger ist.



Heaps

Element einfügen

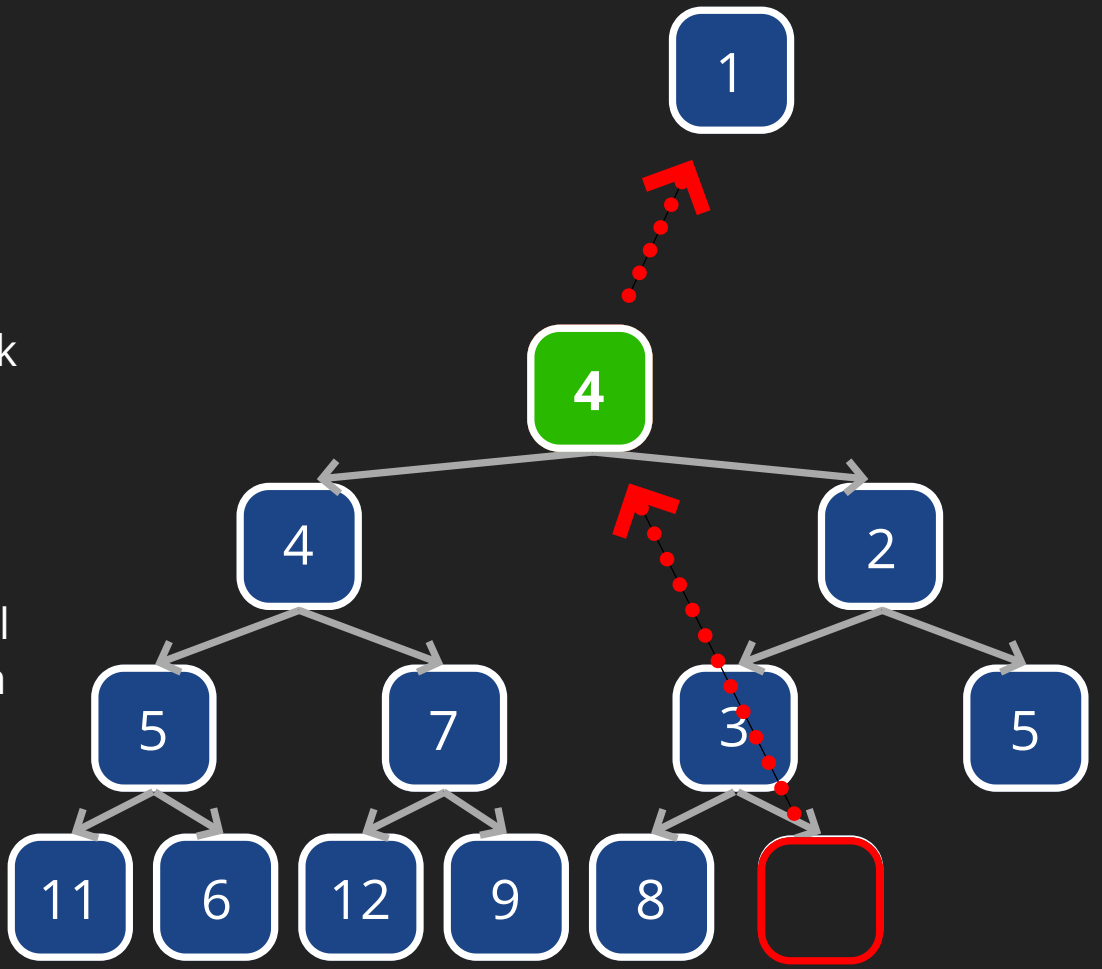
- Um ein Element einzufügen, fügt man es in der Untersten Ebene beim ersten freien Platz ein.
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird das Element so lange mit seinem Vorgänger getauscht, bis es grösser als sein Vorgänger ist.



Heaps

Wurzel entfernen

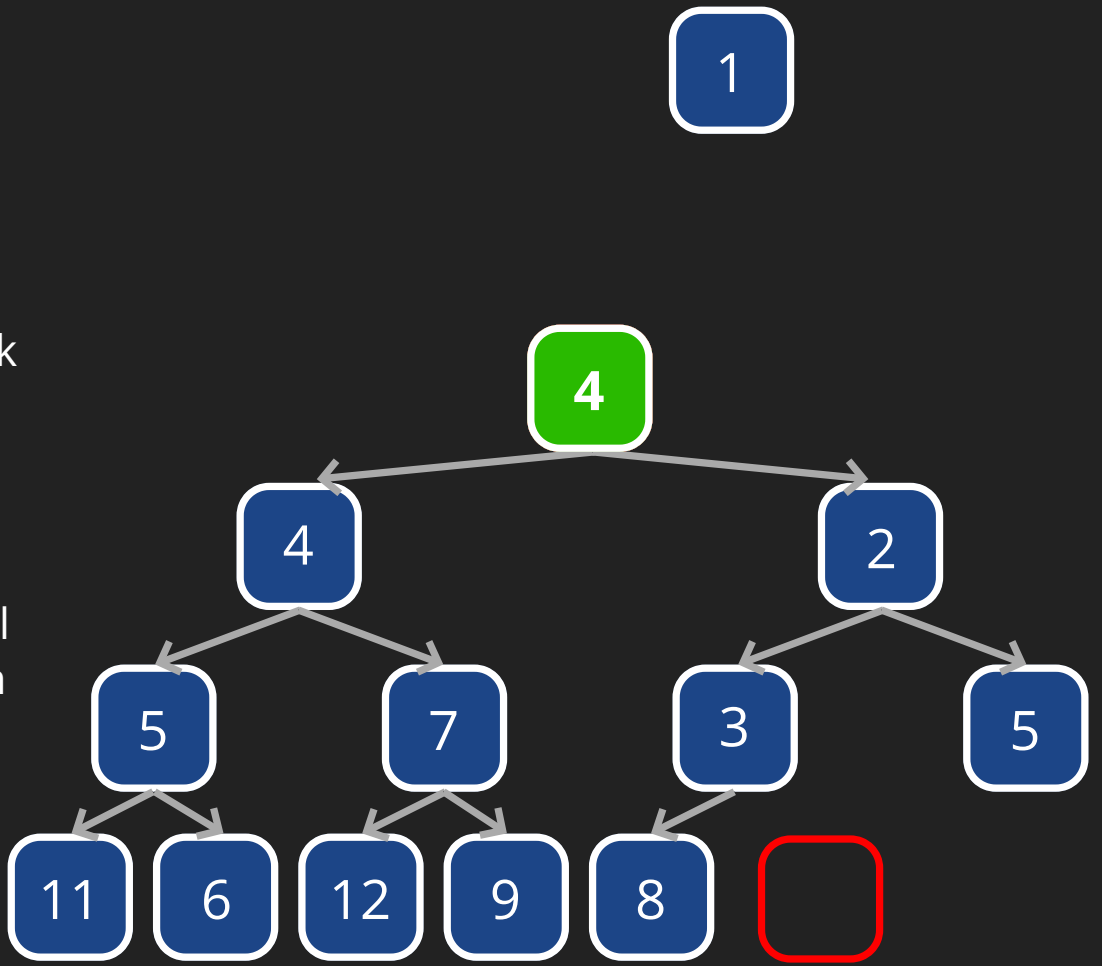
- Um die Wurzel zu entfernen ersetzt man die Wurzel mit dem Letzten Element im Stack
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird die neue Wurzel so lange mit seinem kleinsten Kind getauscht, bis es kleiner als alle Kinder ist.



Heaps

Wurzel entfernen

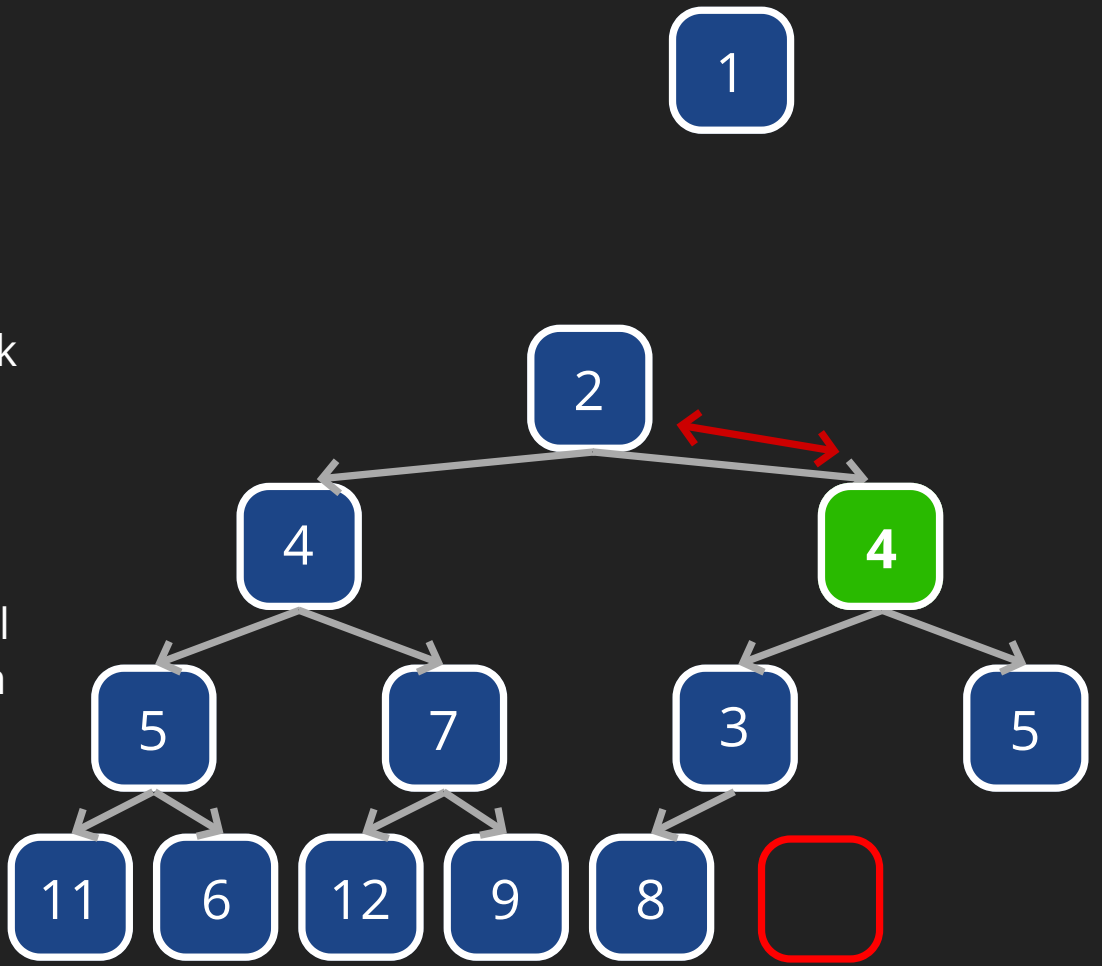
- Um die Wurzel zu entfernen ersetzt man die Wurzel mit dem Letzten Element im Stack
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird die neue Wurzel so lange mit seinem kleinsten Kind getauscht, bis es kleiner als alle Kinder ist.



Heaps

Wurzel entfernen

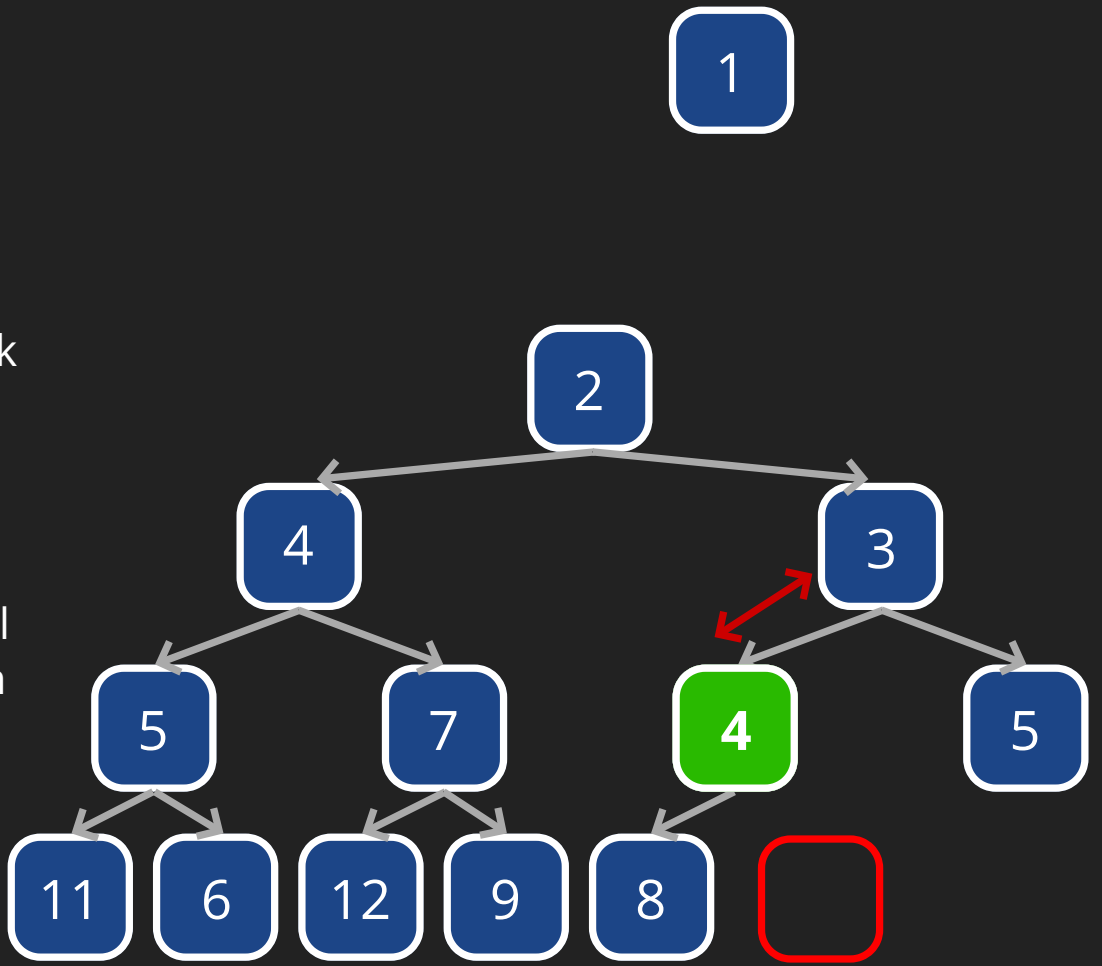
- Um die Wurzel zu entfernen ersetzt man die Wurzel mit dem Letzten Element im Stack
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird die neue Wurzel so lange mit seinem kleinsten Kind getauscht, bis es kleiner als alle Kinder ist.



Heaps

Wurzel entfernen

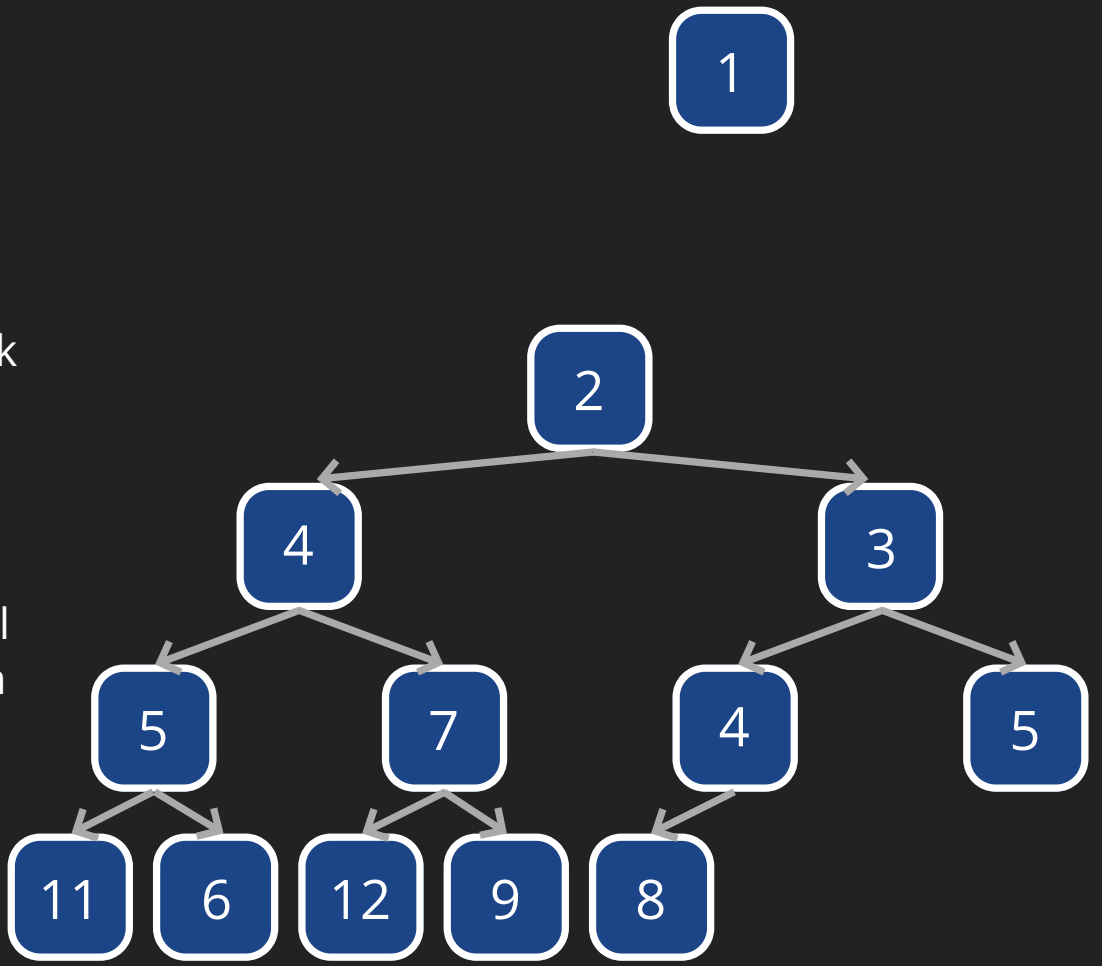
- Um die Wurzel zu entfernen ersetzt man die Wurzel mit dem Letzten Element im Stack
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird die neue Wurzel so lange mit seinem kleinsten Kind getauscht, bis es kleiner als alle Kinder ist.



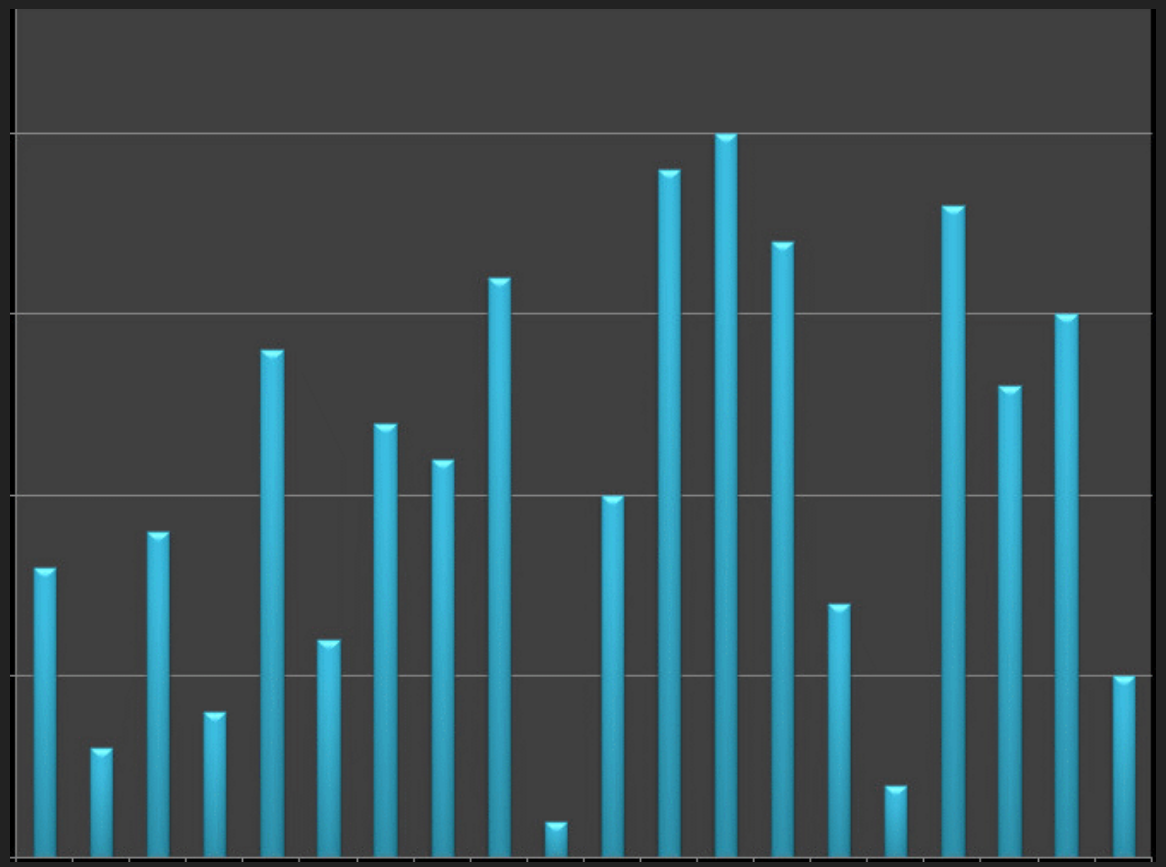
Heaps

Wurzel entfernen

- Um die Wurzel zu entfernen ersetzt man die Wurzel mit dem Letzten Element im Stack
- Dies könnte in einem ungültigen Heap resultieren
- Deshalb wird die neue Wurzel so lange mit seinem kleinsten Kind getauscht, bis es kleiner als alle Kinder ist.



Heap Sort ♥



Heap Sort ♥

- Wie schon in der Übung 2 gelernt haben, lässt sich ein Binärbaum in einem Array darstellen.

- Somit kann man einen Array ohne zusätzlichen Speicherplatz sortieren:
  - Zuerst erstellen wir eine Heapstruktur im Array
  - Danach können wir immer das grösste bzw kleinste Element aus dem Heap entfernen

Heap Sort ♥



unstrukturiert
im heap
sortiert

Heap Sort ♥

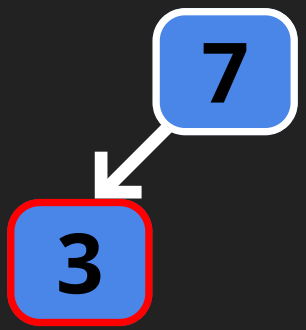


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

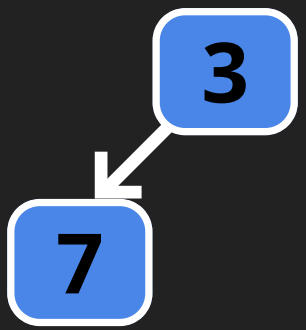


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

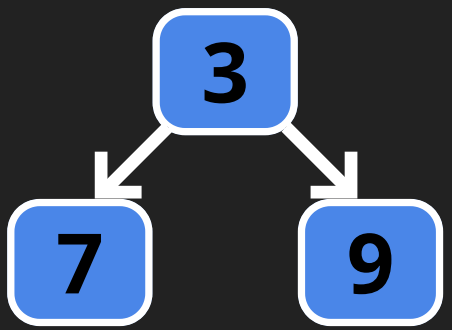


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

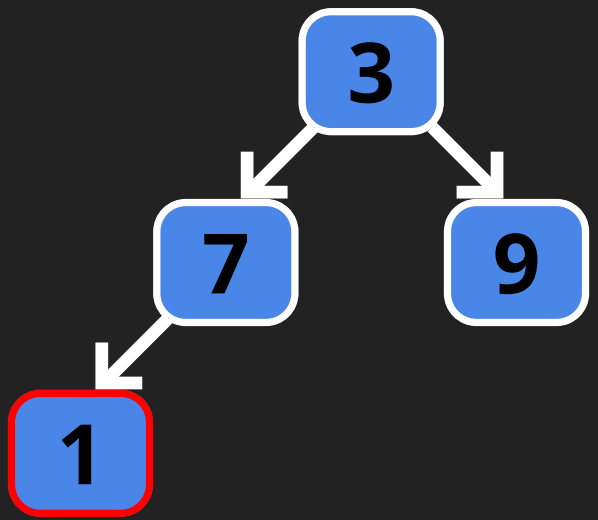


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

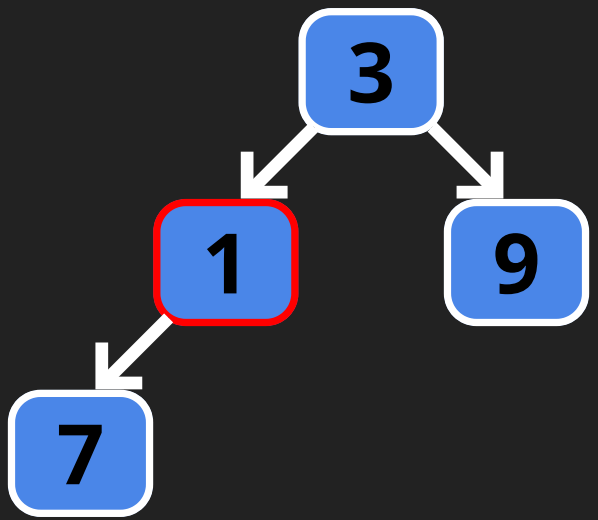


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

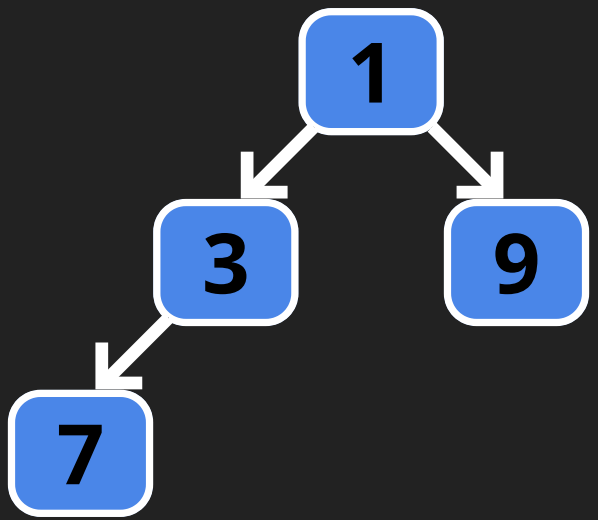


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

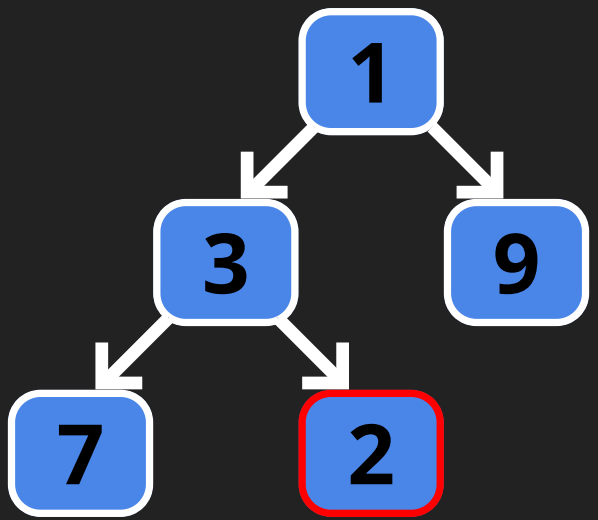


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

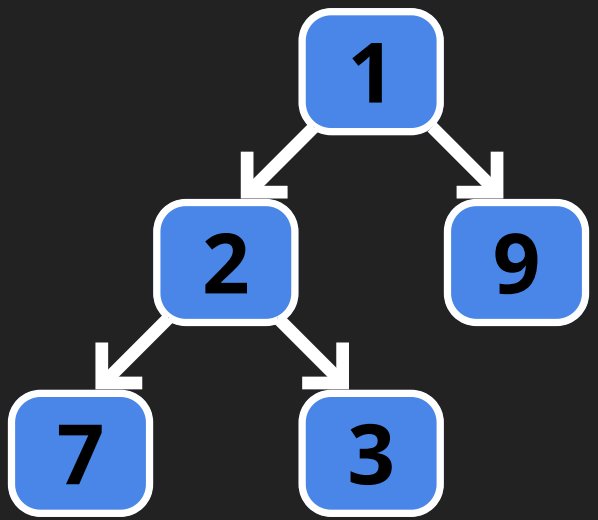


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

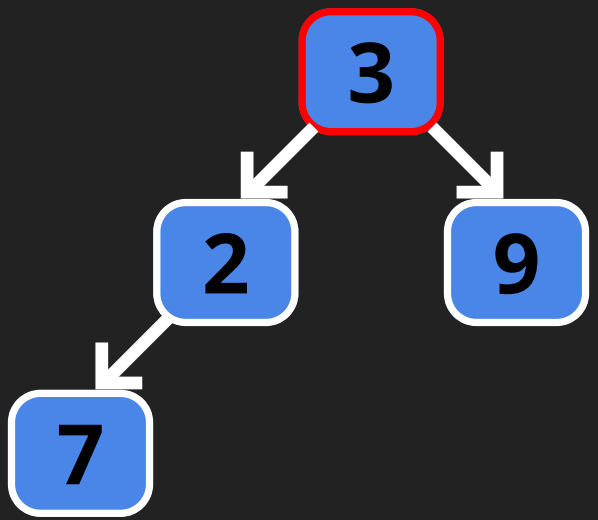


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

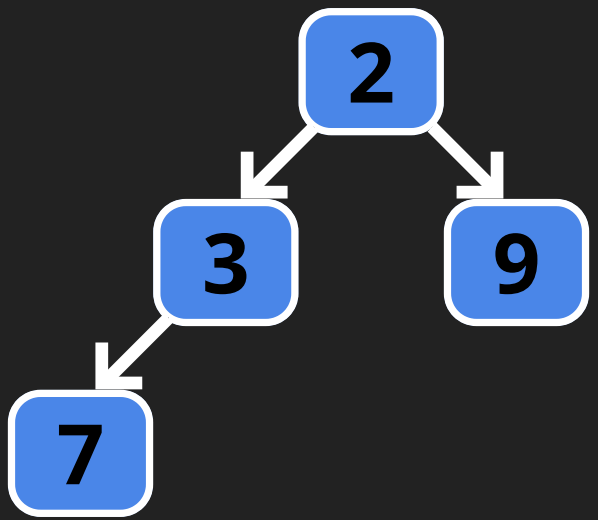


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

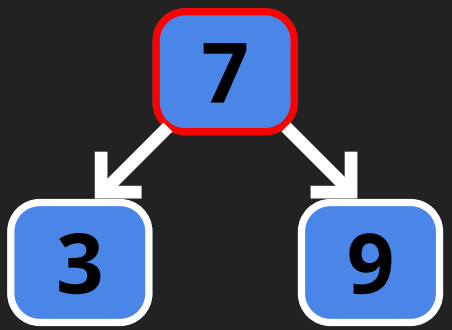


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

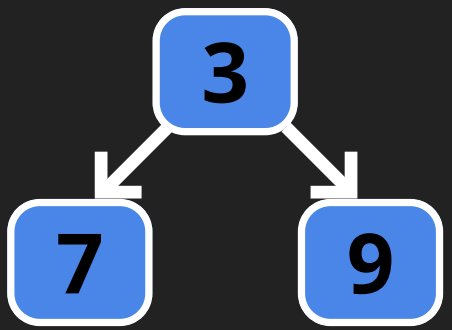


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

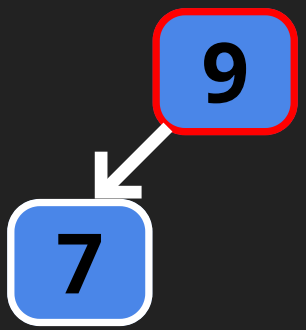


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

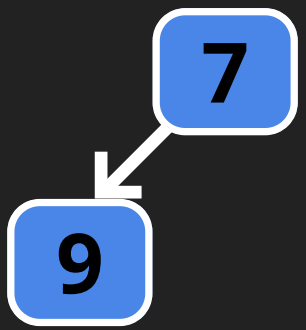


min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥



min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥



min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Heap Sort ♥

min heap (nur als visualisierung vom Array)



unstrukturiert
im heap
sortiert

Parallel processing

- Wenn man verschiedene Tasks parallel Ausführt, kann die Laufzeit von Programmen reduziert werden.

- Um dies zu ermöglichen gibt es zwei Varianten:
  - parallele Prozesse
  - parallele Threads

Parallel processing

- Parallele Prozesse:
  - Laufen in einem anderen Kontext
  - Keinen Zugriff auf Variablen von anderen Prozessen

- Parallele Threads:
  - Laufen im selben Kontext
  - Können auf Variablen von anderen Threads im selben Kontext zugreifen
  - Schnelleres Task-Switching
  - Daten können geteilt werden
  - Kann aber zu Problemen in der Synchronisation geben, wenn man Daten mit anderen Threads teilt

Multithreading - start

```java
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++){
            new Worker().start();
        }
        System.out.println("Done!");
    }
}
```

```java
public class Worker extends Thread {
    static int j = 0;
    int k;

    @Override
    public void run() {
        for (int i = 0; i < 4000000; i++) {
            j++;
            k++;
        }
        System.out.println("k: " + k + " j:" + j);
    }
}
```

```
Done!
k: 4‘000‘000, j: 3‘752‘884
k: 4‘000‘000, j: 10‘806‘936
k: 4‘000‘000, j: 7‘307‘897
k: 4‘000‘000, j: 5‘322‘116
k: 4‘000‘000, j: 4‘366‘084
```

- Mit ".start()" Kann man einen Thread starten. Dieser wird dann im Hintergrund ausgeführt, das heisst es wird nicht gewartet bis die Funktion fertig ist, sondern das Programm geht direkt weiter.
- Die "run" Funktion vom Thread wird ausgeführt wenn ".start()" aufgerufen wird.

Multithreading - synchronized

```java
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++){
            new Worker().start();
        }
        System.out.println("Done!");
    }
}
```

```java
public class Worker extends Thread {
    static int j = 0;
    int k;
    static Object lock = new Object();

    @Override
    public void run() {
        for (int i = 0; i < 4000000; i++) {
            synchronized(lock){
              j++;
              k++;
            }
        }
        System.out.println("k: " + k + " j:" + j);
    }
}
```

```
Done!
k: 4‘000‘000, j: 15‘786‘100
k: 4‘000‘000, j: 18‘432‘690
k: 4‘000‘000, j: 19‘344‘207
k: 4‘000‘000, j: 19‘408‘717
k: 4‘000‘000, j: 20‘000‘000
```

- Mit "synchronized" kann man gewisse Programmabschnitte synchronisieren
- Man braucht ein geteiltes Objekt (hier "lock") als Schloss
- Es kann immer nur ein Thread gleichzeitig im synchronisierten Bereich sein.

Multithreading - join

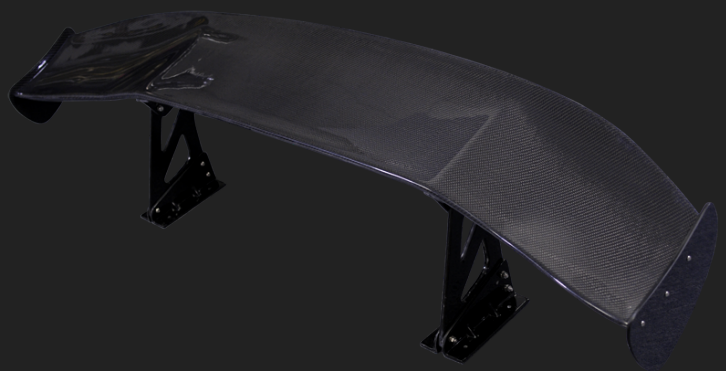
```
1  public static void main(String[] args) {
2      Worker workers[] = new Worker[5];
3      for (int i = 0; i < 5; i++) {
4          workers[i] = new Worker();
5          workers[i].start();
6      }
7      for (int i = 0; i < 5; i++) {
8          try {
9              workers[i].join();
10         } catch (InterruptedException e) {
11             e.printStackTrace();
12         }
13     }
14     System.out.println("Done!");
15 }
```

```
k: 4‘000‘000, j: 15‘786‘100
k: 4‘000‘000, j: 18‘432‘690
k: 4‘000‘000, j: 19‘344‘207
k: 4‘000‘000, j: 19‘408‘717
k: 4‘000‘000, j: 20‘000‘000
Done!
```

- Mit "join" kann man warten bis ein Thread fertig ist.
- So ist es mehrere Threads zu starten und dann zu warten bis alle fertig sind.


K!

Vorbesprechung



Heapsort Textaufgaben

1. Wie viel Elemente sind in einem Heap der Höhe h mindestens/maximal?

2. Repräsentiert ein sortiertes Array einen gültigen Heap?
   Ist die Array Repräsentation eines Heaps immer sortiert?

3. Heapsort von Hand

Unsortierte Folge:	2	0	3	4	7	5
Schritt 1:						
Schritt 2:						
Schritt 3:						
Schritt 4:						
Schritt 5:						
Heap:						
Schritt 1:						
Schritt 2:						
Schritt 3:						
Schritt 4:						
Schritt 5:						
Sortierte Folge:						

Heapsort Programmieren

- Implementiert Heapsort (min Max-Heap)
- Teilt den Algorithmus in Teilprobleme auf:
  1. Heap aufbau
  2. Heap abbau
- Verwendet dazu Hilfsfunktionen um beim einfügen/entfernen von Elementen zu helfen:
  1. raise: Element so weit wie nötig nach oben wandern lassen
     - Ist Element schon am richtigen Platz?
     - sonst swap mit parent und raise an der neuen Position rekursiv ausführen
  2. sink:Element so weit wie nötig nach unten wandern lassen
     - Ist Element schon am richtigen Platz?
     - Sonst swap mit dem grösseren Kind und sink an der neuen Position rekursiv ausführen
  3. swap, father, leftChild, rightChild

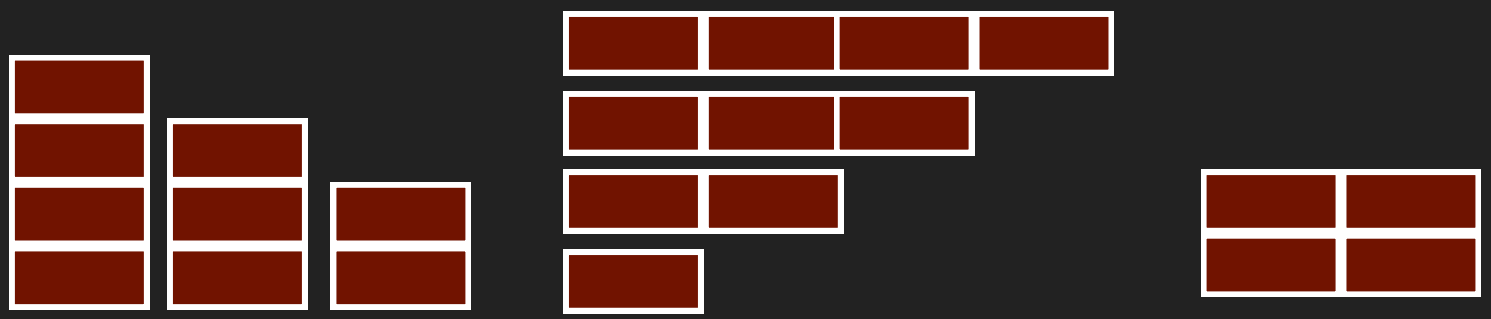
Parallelisiertes Mergesort

- Passt den Mergesort Algorithmus aus Serie 10 so an, dass er mit mehreren Threads laufen kann.
- Schreibt dazu eine Worker Klasse:

```
1   public Worker(MergeSort sort, int[] items, int threadCount, int begin, int end) {
2     this.sort = sort;
3     this.items = items;
4     this.threadCount = threadCount;
5     this.begin = begin;
6     this.end = end;
7   }
8
9   @Override
10  public void run() {
11    if(threadCount <= 1)
12      // Call Mergesort function from Exercise 10
13    else {
14      // 1. Divde array into two parts
15      // 2. Create a new worker for left half
16      // 3. Create a new worker for right half
17      // 4. Wait for both workers to finish
18      // 5. Merge results of both workers and store it in sortedItems
19    }
20  }
21
22  public int[] getResult() {
23    return sortedItems;
24  }
```

Rekursives Problemlösen

- Wir haben n Rechtecke gegeben. Wie viele Rechtecke kann man mit diesen n Rechtecken als Bauklötze bauen?
- Beispiel $n = 4$: **8**



- Findet eine rekursive Formel: $f(n) = g(f(n-1))$

- Gute Übung für Rekursives Problemlösen, jedoch nicht so relevant wie die ersten Aufgaben.

Reversi

- Nächste Woche ist Reversi Tournier (Fr 18.12 13:00 Zoom), es findet keine Übung statt.
- Da auf dem Server eine Alte Java-Version läuft, funktionieren einige eurer Spieler nicht auf dem Server.
- Um dieses Problem zu umgehen ist es möglich eure Spieler mit einer älteren Java Version zu Kompilieren. Julia hat dazu eine super Anleitung gemacht: https://n.ethz.ch/~jgygax/Slides/Reversi-Spieler_upload.pdf
- Mitmachen lohnt sich, der Event ist wirklich cool, falls Ihr bis jetzt noch nichts habt, könnt Ihr auch einfach den AlphaBeta Spieler von der Musterlösung nehmen und die eval()-Funktion verbessern

Viel Spass, schöne Weinachten, gute Lernphase, bleibt gesund!

Ich freue mich euch am Reversi Tournier und am PVK nochmals zu treffen.

