ETH zürich

Informatik II - Übung 10

Katja Wolff
katja.wolff@inf.ethz.ch

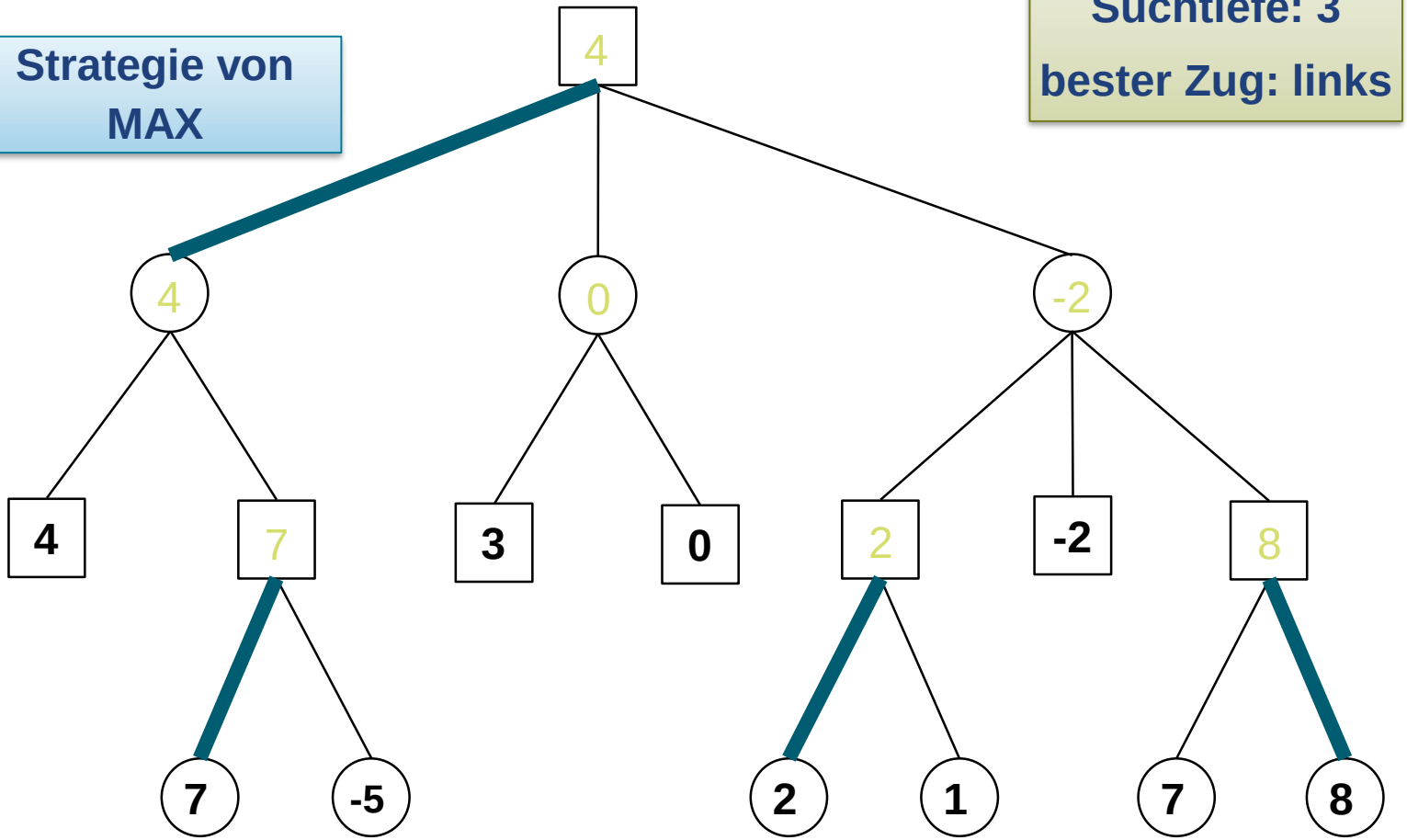
U9.A1: Mini-Max-Algorithmus

Höhe: 4

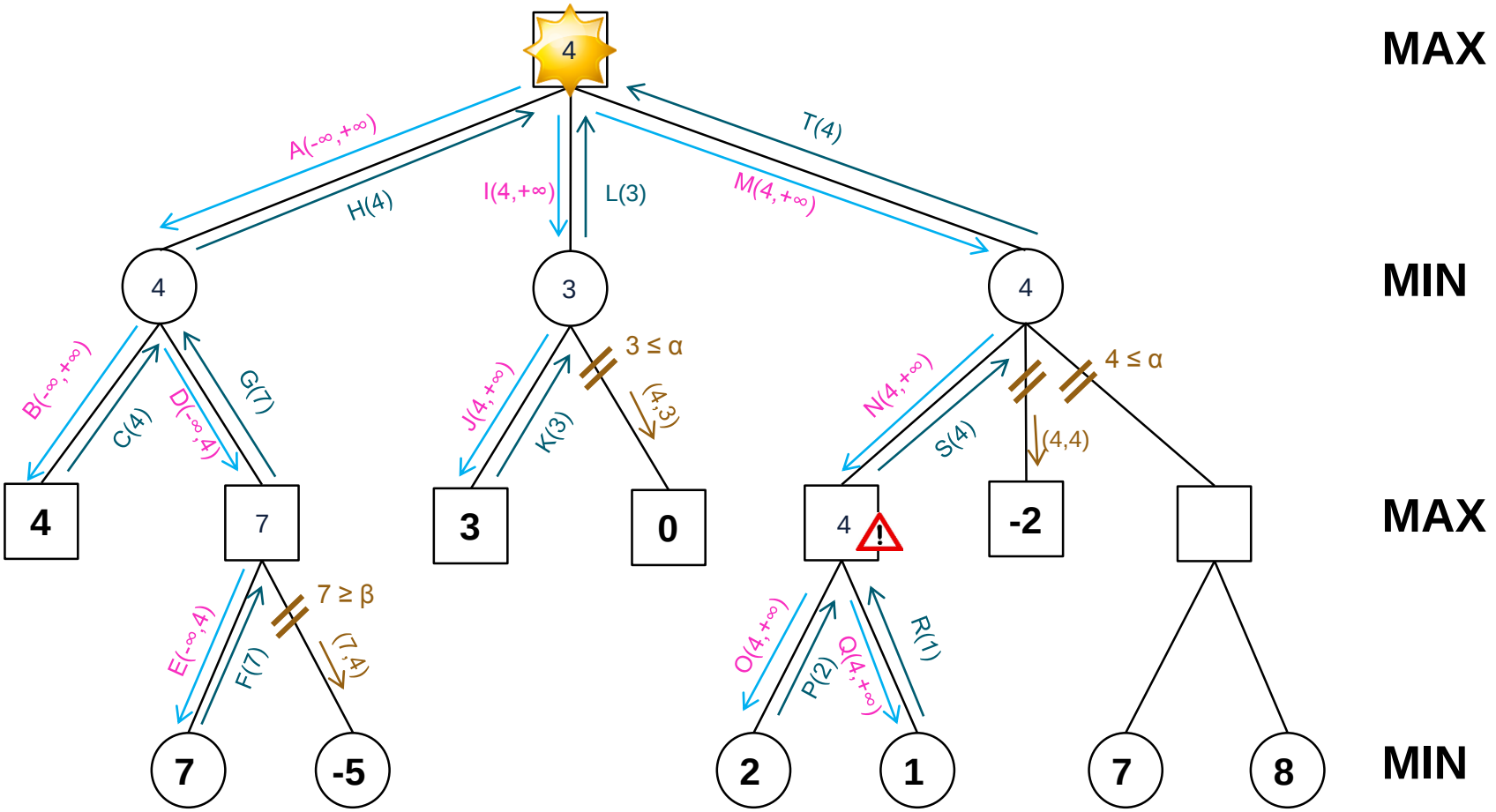
Suchtiefe: 3

bester Zug: links

Strategie von MAX



U9.A1: Alpha-Beta-Pruning



U9.A1: Alpha-Beta-Pruning

Online Beispiel durchgerechnet:

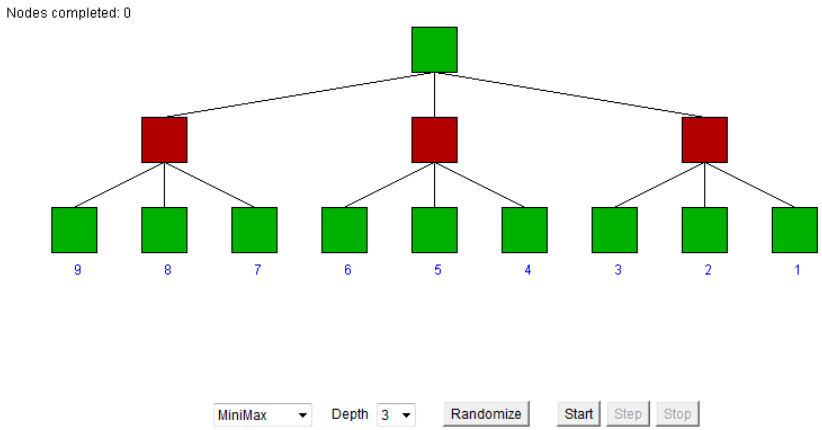
http://www.vs.inf.ethz.ch/edu/FS2012/I2/slides/Info2-ITET-AlphaBeta.pdf
(user: i2 password: i22012)

Online JAVA Applet:

http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

Another online demo:

http://www.emunix.emich.edu/~evett/AI/AlphaBeta_movie/sld001.htm



U9.A1: Alpha-Beta-Pruning

- Der α-β-Algorithmus
  - Reduziert den Spielbaum durch Schnitte
  - Liefert den gleichen Minmax-Wert **der Wurzel** wie der MinMax-Algorithmus: Der MinMax-Algorithmus bewertet den vollständigen Suchbaum. Dabei werden aber auch Knoten betrachtet, die in das Ergebnis (die Wahl des Zweiges an der Wurzel) nicht einfliessen. Die Alpha-Beta-Suche ignoriert eben genau diese Knoten

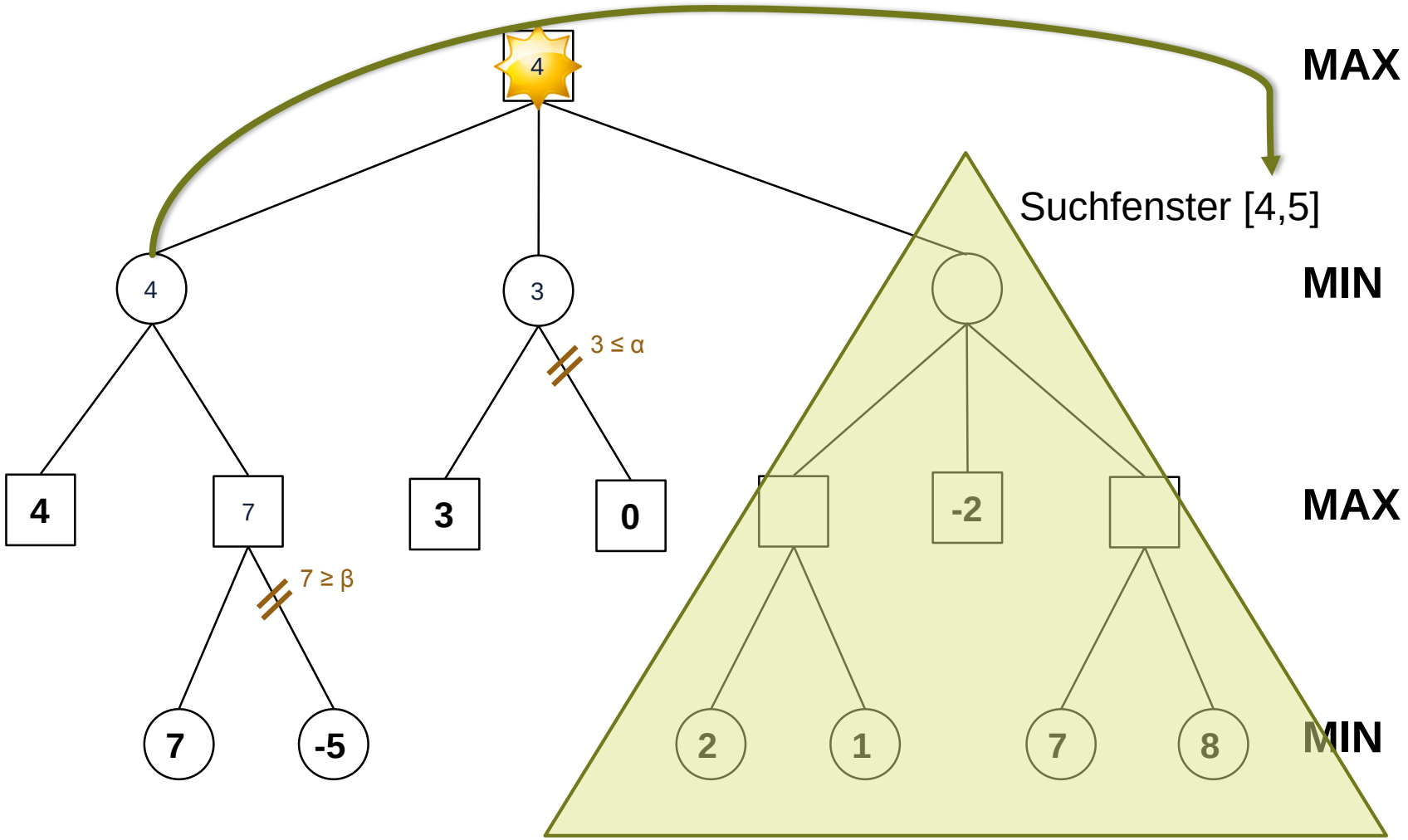
- **α:** Relevant für Auswertung von **Min-Knoten** (Evaluierung der weiteren Nachfolger kann abgebrochen werden, sobald der berechnete Rückgabewert kleiner-gleich α ist)

  grösster bisher bekannter Wert aller MAX-Vorfahren des MIN-Knotens

- **β:** Relevant für Auswertung von **Max-Knoten** (Evaluierung der weiteren Nachfolger kann abgebrochen werden, sobald der berechnete Rückgabewert grösser-gleich β ist)

  kleinster bisher bekannter Wert aller MIN-Vorfahren des MAX-Knotens

U9.A1: Last Move Improvement



Ergebnis ≤ 4: rechter Zug uninteressant
Ergebnis ≥ 5: bester Wert hier, also wähle diesen Zug

U9.A2: Reversi

HumanPlayer	RandomPlayer	GreedyPlayer	MinMaxPlayer	α-β-Player
nextMove() wartet auf Eingabe von der Kommando-zeile	nextMove() wählt einen zufälligen (aber gültigen!) nächsten Zug	nextMove() wählt nächsten Zug anhand einer einfachen, nicht-rekursiven Bewertungs-funktion	nextMove() wählt nächsten Zug anhand Min-Max-Analyse mit neuer Bewertungs-funktion	nextMove() wählt nächsten Zug anhand α-β-Analyse mit eigener Bewertungs-funktion
Download	Übung 7	Übung 8	Übung 9	Übung 10

U9.A2: MinMax-Player

- `min()` und `max()` rufen sich abwechselnd auf
- Suchtiefe `depth` wird in jedem Aufruf angepasst
- Jedes mal:
  - Get available moves
  - Für jeden Move:
    - Führe Move durch
    - Rufe min-Spieler (bzw. max-Spieler) auf
    - Wähle bestes Ergebnis

```
public Coordinates nextMove(GameBoard gb){
   BestMove bestMove = null;

   bestMove = max(d, gb, 0);

   return bestMove.coord;
}
```

```
class BestMove {
        public Coordinates coord;
        public int value;
        public boolean cut;
//whether it was cut at the
//maximum recursion depth
}
```

U9.A2: MinMax-Player

```
private BestMove max(int maxDepth, GameBoard gb, int depth){
   if( depth == maxDepth )   return new BestMove(eval(gb), null, true);
   ArrayList<Coordinates> availableMoves = getMovesFor(myColor, gb);
   if( availableMoves.isEmpty() ){
      if( gb.isMoveAvailable(otherColor) ){ //check whether PASS could be good
         BestMove result = min(maxDepth, gb, depth + 1);
         return new BestMove(result.value, null, false);
      } else
         return new BestMove(finalResult(gb), null, false);
   }
   BestMove bestMove = new BestMove(minEval(gb) - 1, null, false);
   for( Coordinates coord : availableMoves ) {
      GameBoard hypothetical = gb.clone();
      hypothetical.checkMove(myColor, coord);
      hypothetical.makeMove(myColor, coord);
      BestMove result = min(maxDepth, hypothetical, depth + 1);
      bestMove.cut = bestMove.cut || result.cut;
      if( result.value > bestMove.value ){
         bestMove.coord = coord;
         bestMove.value = result.value;
      }
   }
   return bestMove;
}
```

U9.A2: Und nun mit Timeout

- Vor Ablauf von `timeLimit` Millisekunden soll `nextMove()` einen gültigen Zug zurückgegeben

```
public Coordinates nextMove(GameBoard gb) {
   long timeout = System.currentTimeMillis() + timeLimit - 10;
   BestMove bestMove = null;
   try{
      for(int maxDepth=1; ; ++maxDepth)
         bestMove = max(maxDepth, timeout, gb, 0);
   }catch(Timeout e){
   }
   return bestMove.coord;
}
```

U9.A2: Und nun mit Timeout

- Einbindung in `min()` bzw. `max()`

```
class Timeout extends Throwable{
}
```

```
private BestMove max(int maxDepth, long timeout, GameBoard gb, int depth)
                                                    throws Timeout
{
   if( System.currentTimeMillis() > timeout )
      throw new Timeout();

   if( depth == maxDepth ){
      return new BestMove( eval(gb), ... );

   ...

   return bestMove;
}
```

Möglichkeiten für die Bewertungsfunktion

- Vorschläge für mögliche, statische Bewertungen
  - *Mobilität*
    - Wieviele Züge sind für mich/den Gegner möglich?
  - *Reihen*
    - Wie viele Reihen zusammenhängender Steine gibt es und
    - Wie lang sind diese? Auch die Lage der Reihen ist interessant!
    - Ein voll besetzter Rand ist sehr gut, während eine lange Sequenz in der zweiten Reihe dem Gegner u.U. gute Züge verschafft
  - *Wie viele Steine …*
    - werden durch einen Zug umgedreht und in wie viele Richtungen? Liegen die Steine im Inneren oder am Rand?
  - *Wie viele Steine*
    - einer Farbe liegen? (Das ist vielleicht die Bewertungsfunktion für das Endspiel, wenn eine vollständige Analyse des Suchbaums möglich ist; für das Mittelspiel wohl eher ungeeignet)
  - *Positionen*
    - Auf dem Feld bewerten (z.B. Eckpunkte)

Weitere Literatur

- Inspirationsquelle:
  - „The Development of a World Class Othello Program“, Kai-Fu Lee and Sanjoy Mahajan, 1990
- Zum Herunterladen von der Reversi-Webseite
  - username: i2bib
  - password: reversi
- Artificial Intelligence: A Modern Approach (Stuart Russell and Peter Norvig, 2nd Edition, 2003)

Hinweise zu Blatt 10

1) Merge sort

2) Türme von Hanoi

3) Reversi: Alpha-beta player

U10.A1: Merge sort

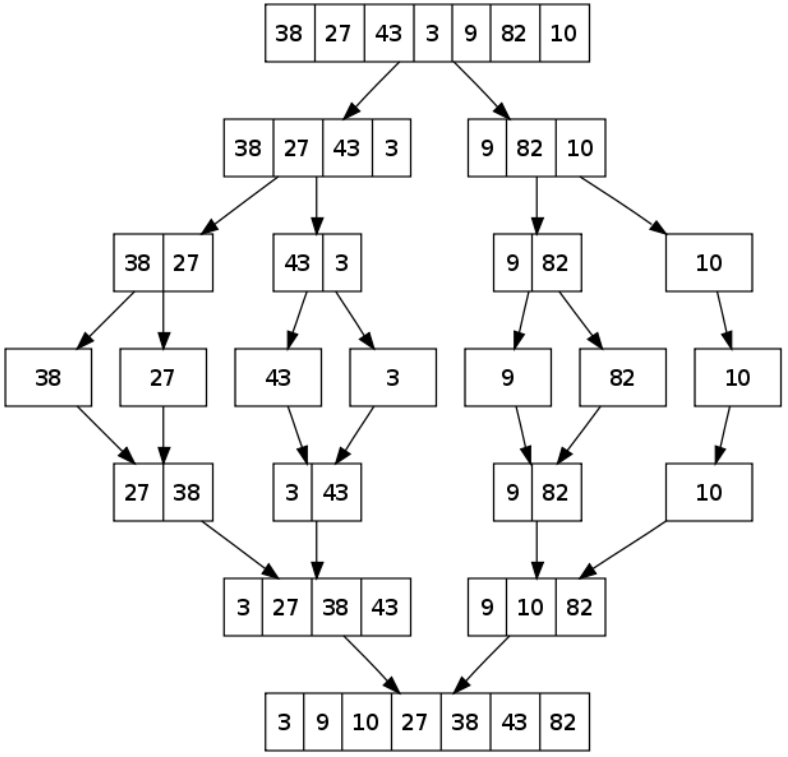
- Rekursiver Sortieralgorithmus, der nach dem Prinzip **teile und herrsche** arbeitet
- Erstmals 1945 von John von Neumann vorgestellt

- Prinzip: Divide et impera
  - Trenne (die Gegner)
    und beherrsche (sie dadurch)
  - Politische und militärische Strategie
  - Bereits im römischen Reich angewendet

John von Neumann
1903 Budapest – 1957 Washington

U10.A1a – Handarbeit

- Merge sort
  - Betrachtet die zu sortierenden Daten als Liste und zerlegt sie in kleinere Listen
  - Die sortierten kleinen Listen werden dann im Reissverschlussverfahren zu grösseren Listen zusammengefügt (*merged*)



U10.A1b – Implementierung

- `ISort`: definiert die Schnittstelle
  - `ISort.sort()`: nimmt eine `ArrayList` und gibt eine neue, sortierte `ArrayList` zurück
- `MergeSort.java` (erstellen)
  - Implementiert die Schnittstelle `ISort`
  - *Tipp*: rekursive Helfermethode
  - *Tipp*: Beim Teilen müssen keine neuen Listen erstellt werden, man kann mit begin-end arbeiten

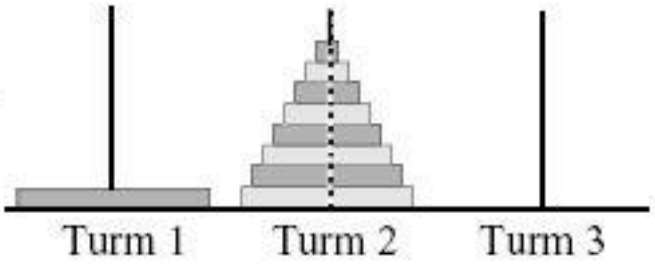
```
public ArrayList<T> sort(ArrayList<T> items) {
        return sortRec(items, 0, items.size());
}

private ArrayList<T> sortRec(ArrayList<T> items, int
begin, int end) {
        ...
}
```

U10.A1b: Measure.java

- 10 "Messpunkte„ (Arraygrößen)
  - Beachtet, dass ihr die Zufallsarrays ausserhalb der Zeitmessungen erstellt!
- Messungen wiederholen
  - je ein Extremwert ignorieren (min und max)
  - über n Messungen mitteln (insgesamt n+2 Messläufe)
- Diagramm erstellen
  - beliebiges Tool (Bsp: GNUplot, Excel, Matlab, …)
  - Abgabe als Bild
  - Interpretation wichtig!

U10.A2 – Türme von Hanoi



- Aus der Vorlesung bekannt
  - Rekursive Lösung des Problems

- Die einzige Möglichkeit, die unterste (grösste) Scheibe von Turm 1 nach Turm 3 zu bewegen

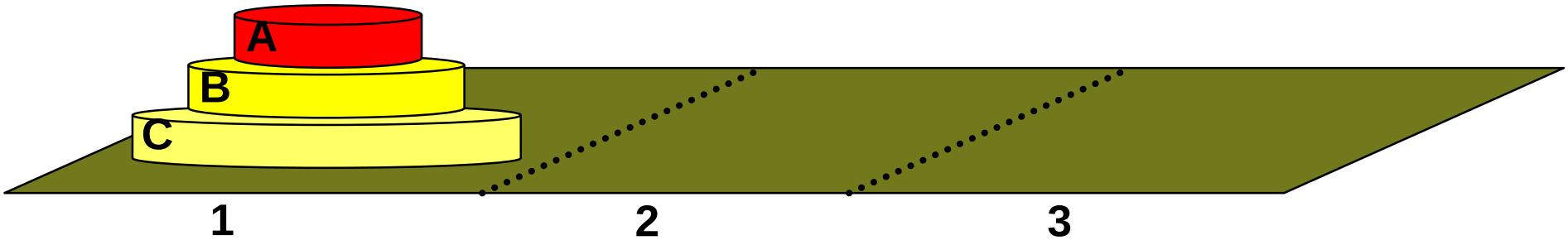
  (a) Auf Turm 1 befindet sich sonst nichts

  (b) Turm 3 ist leer

- Aus (a) und (b) folgt:
  - Alle anderen Scheiben befinden sich auf Turm 2!
  - Es müssen zunächst die n-1 anderen Scheiben von Turm 1 zu Turm 2 gebracht werden

U10.A2 – Türme von Hanoi

- Lösung des 3-Scheiben-Falles
  - Nennen wir die 3 Felder 1,2,3 und die Scheiben A,B,C
  - Ein Zahl-Buchstabenpaar gibt an welche Scheibe wohin bewegt werden soll (C2: die grösste der 3 Scheiben soll auf den mittleren Turm gelegt werden)
- Lösungsschritte:
  - A3, B2, A2, C3, A1, B3, A3 (7 Schritte)



U10.A2 – Türme von Hanoi

- Gesetzmässigkeiten erkennen:
  - Für jeden Schritt bei der Ausführung des rekursiven Algorithmus aus der Vorlesung wird genau ein Turm nicht benötigt.
  - Geben Sie für die 15 Schritte bei der Umschichtung eines Turms der Höhe 4 die Folge der Nummern derjenigen Türme an, die nicht „angefasst“ werden

- Beschreiben Sie Ihren „entwickelten“ Algorithmus in Pseudocode
  - Für den Anfangsturm der Höhe 4
  - Sind Anpassungen für Türme der Höhe 5 nötig?

U10.A3: Alpha-beta player

- Reversi-Spieler, der nach dem α-β-Verfahren vorgeht (sonst wie Minimax-Spieler)

- α-β-Algorithmus
  - Implementierung des Verfahrens aus der Vorlesung
  - Zugabbruch nach wie vor durch ein Timeout

Reversi-Turnier!

- Mittwoch, 31.05.2017, ab 12:30 Uhr, im CABinett (Stuz2)
  http://www.vs.inf.ethz.ch/edu/FS2017/I2/reversi/

- **Abgabe**
  - Bis Mittwoch, den 23.05.2017, 23:59 Uhr (Zürich Time)
  - Zusätzlich: An Leyna Sadamori (leyna.sadamori@inf.ethz.ch)
  - Alleine oder in Zweiergruppen (Aber: die Preise sind für Zweiergruppen ausgelegt!)

…viel Spass!