Informatik II - Übung 10

Pascal Schärli

aacchpss@pascscha.ch

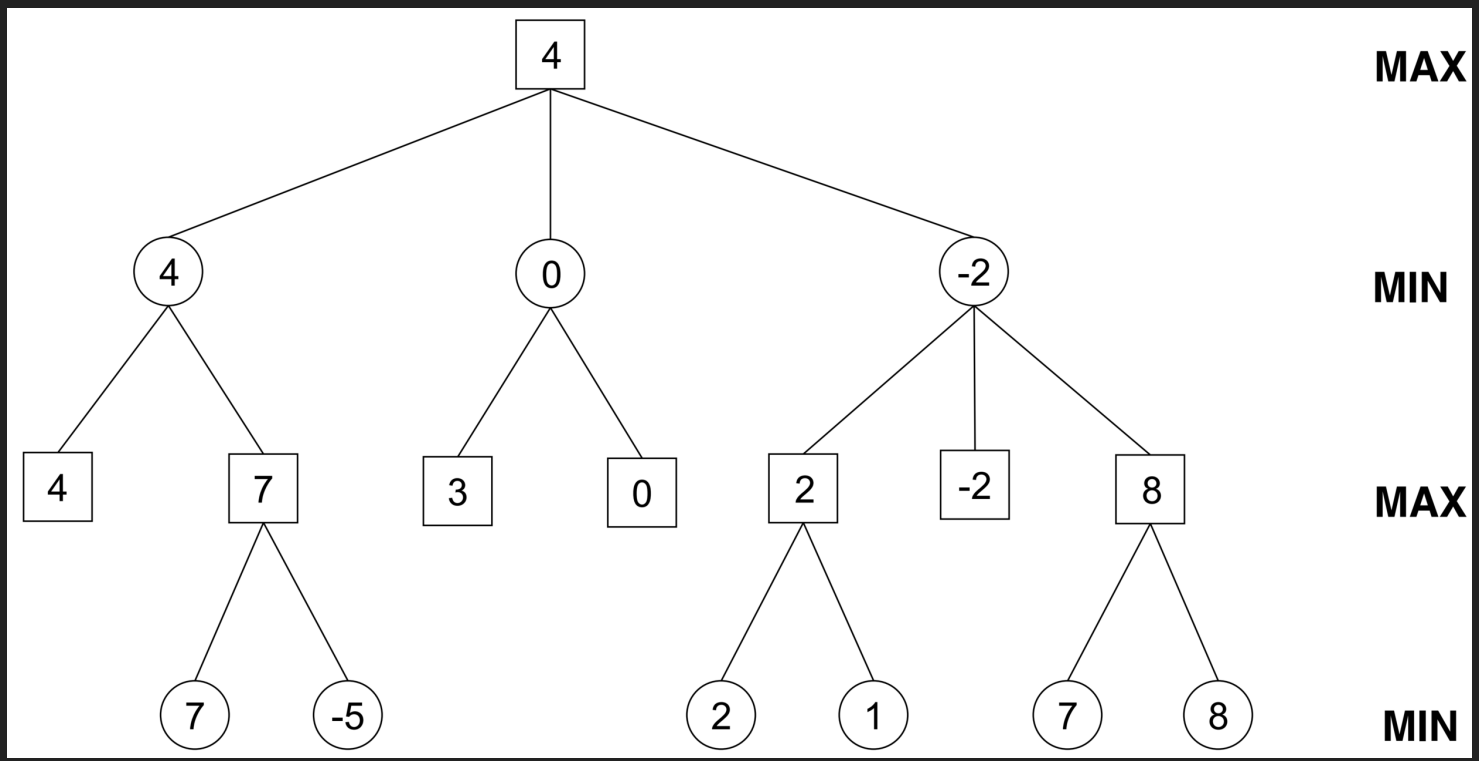
27.11.2020

ETH zürich

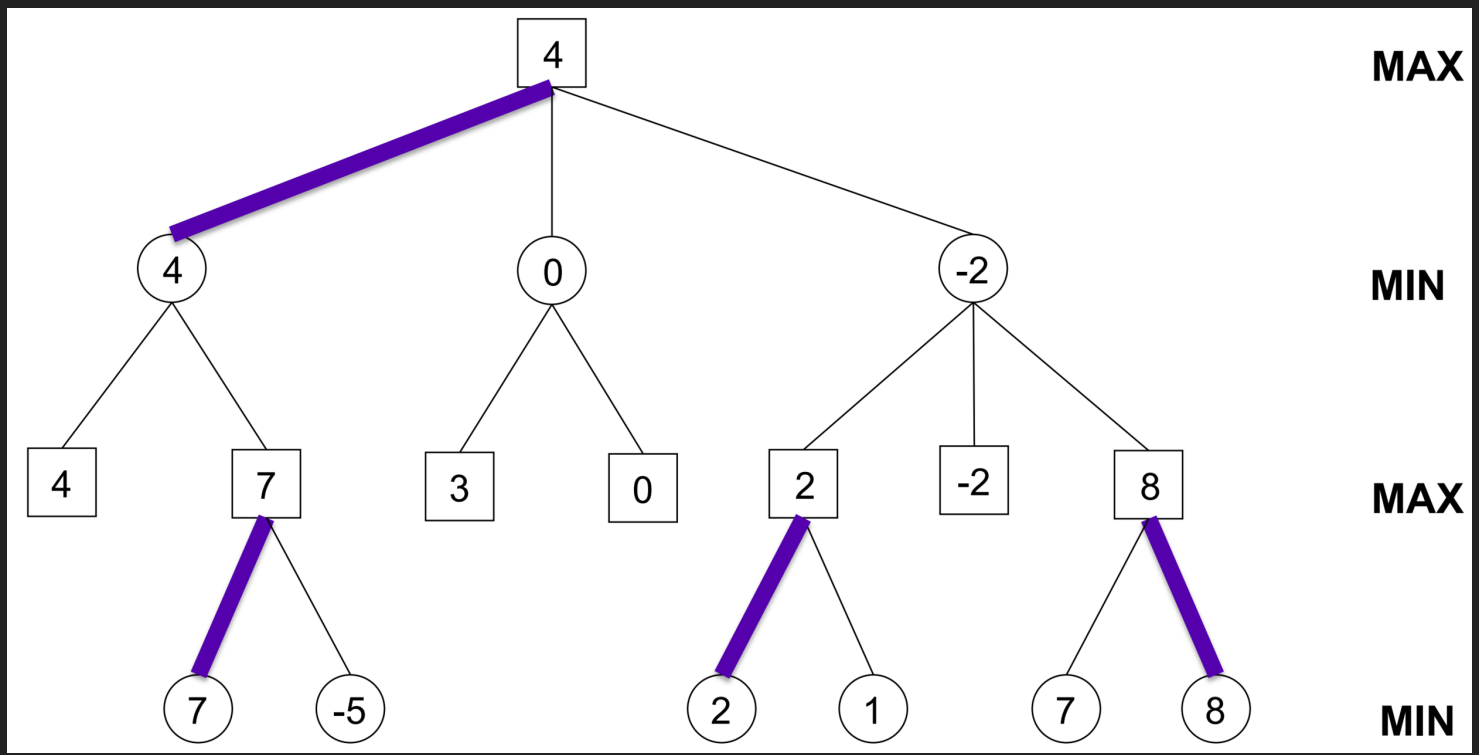
Nachbesprechung

Bild: look again by corpus delicti from the Noun Project

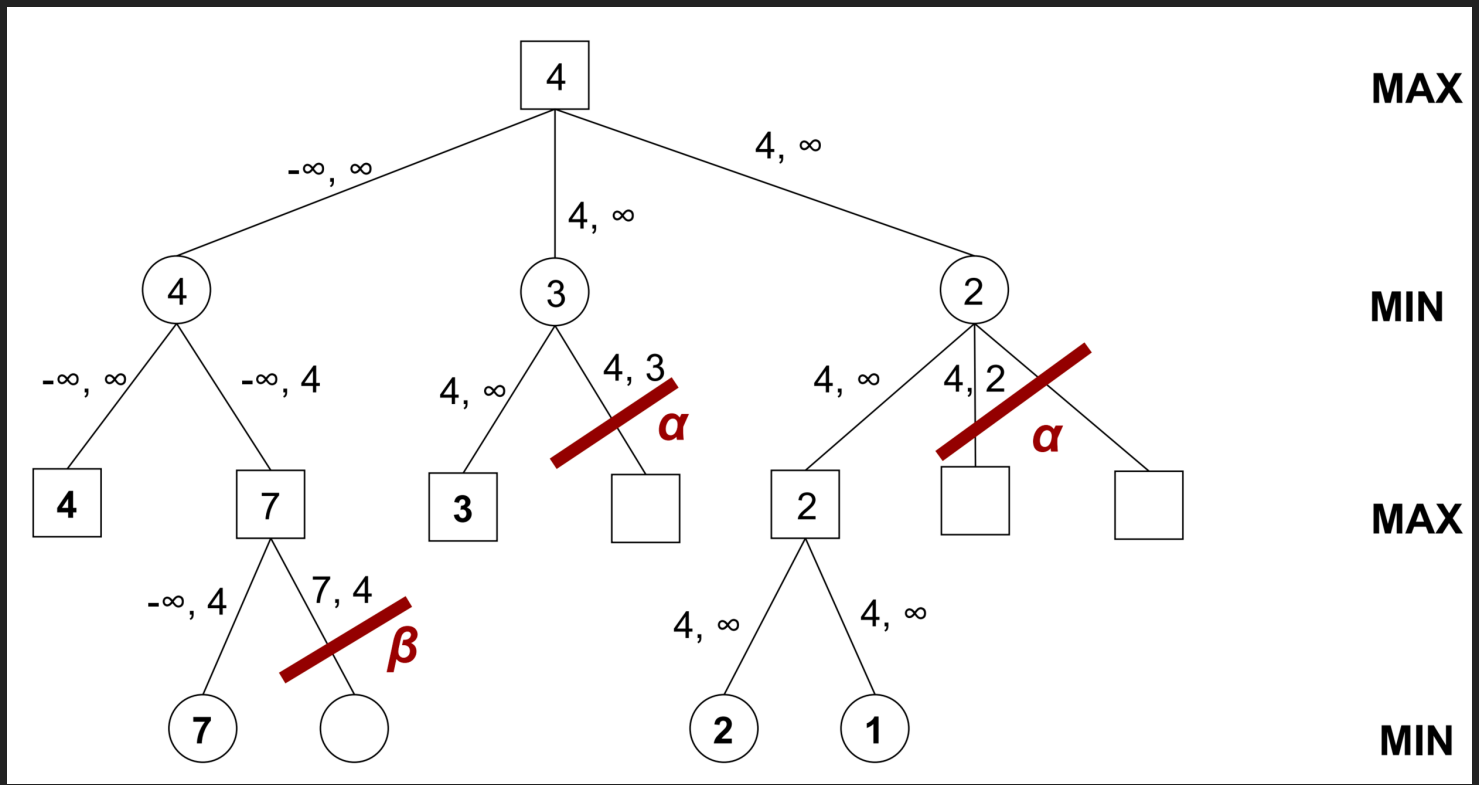
Spieltheorie



Spieltheorie



Spieltheorie



Vorlesung

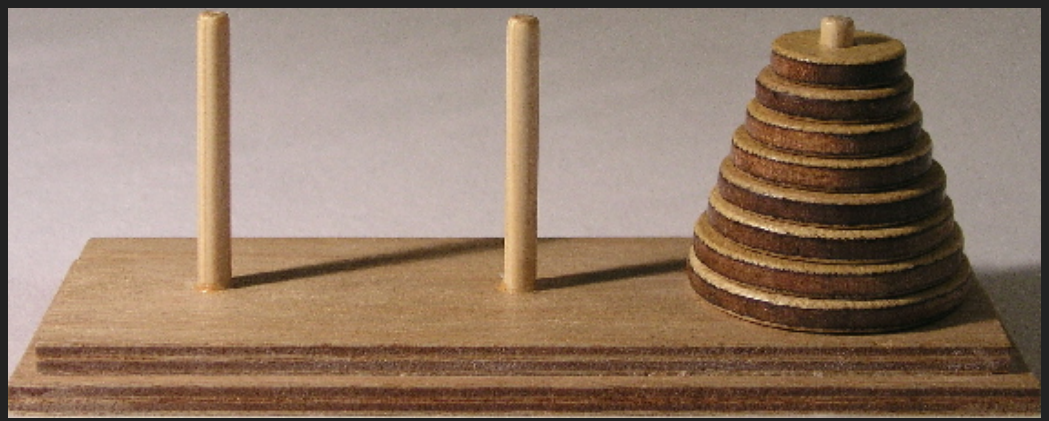
Divide and conquer

- Komplexe Probleme können auf einfache Teilprobleme reduziert werden
- Diese Teilprobleme kann man weiter unterteilen, bis das Problem simpel genug ist
- Teile und Herrsche
- Heute:
  - Türme von Hanoi
  - Mergesort

Türme von Hanoi

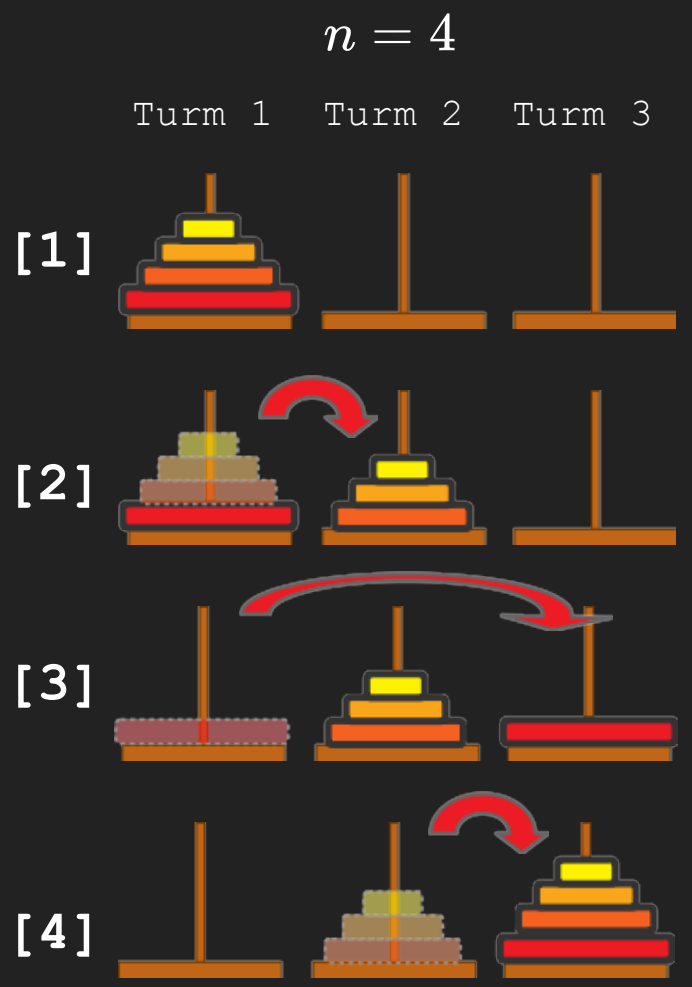
- Das Ziel ist es, alle Scheiben von Turm 1 nach Turm 3 zu verschieben.
- Man darf immer nur eine Scheibe auf einmal verschieben
- Es darf nie eine grössere Scheibe auf einer kleineren liegen.

Turm 1 Turm 2 Turm 3



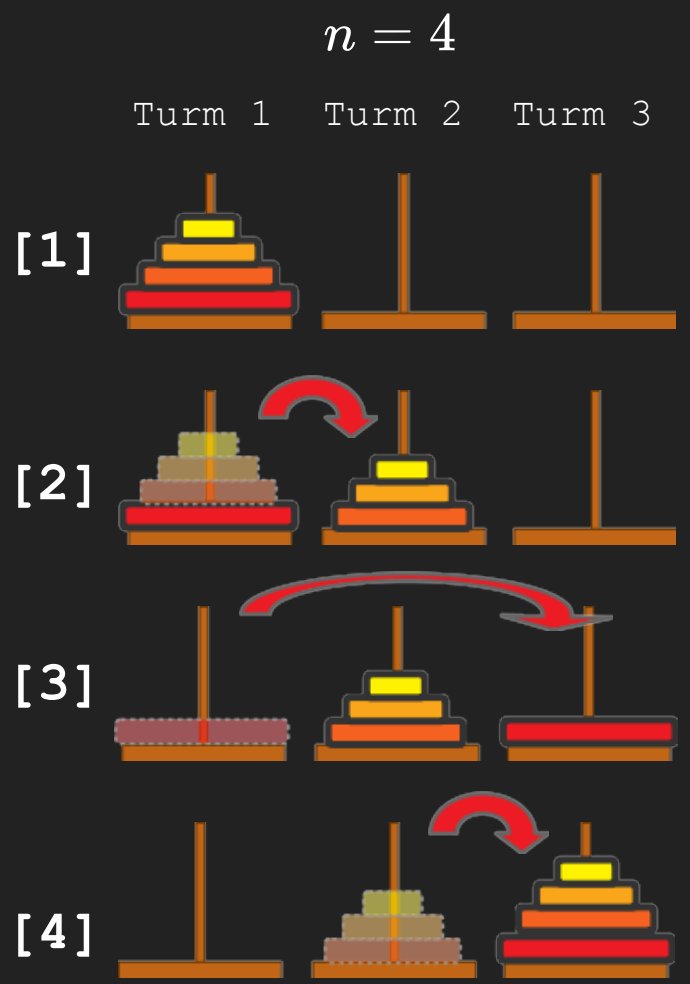
Türme von Hanoi

1. Wir wollen n Scheiben von Turm 1 nach Turm 3 verschieben:
2. $n-1$ Scheiben von Turm 1 nach Turm 2 verschieben
3. Grosse Scheibe von Turm 1 nach Turm 3 verschieben
4. $n-1$ Scheiben von Turm 2 nach Turm 3 verschieben



Türme von Hanoi

- Im Zug 2 und 4 verschieben wir mehr als eine Scheibe
- Wir müssen einen Weg finden die $n-1$ Scheiben zu verschieben ohne die Regeln zu verletzen
- Wir verwenden unseren divide and conquer Algorithmus für die $n-1$ Scheiben



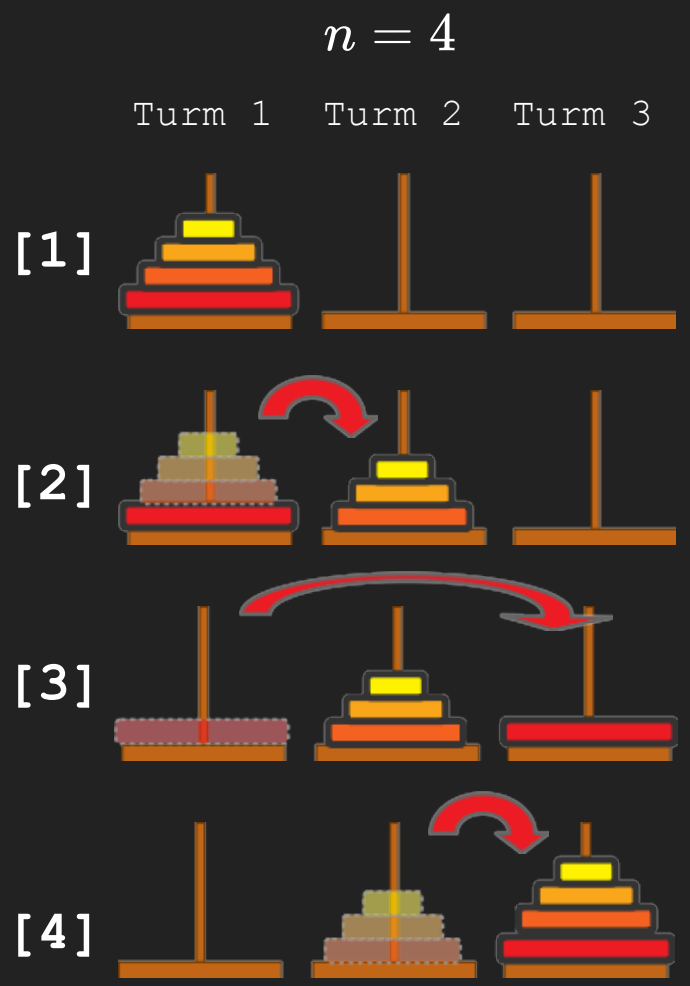
Türme von Hanoi

1. Wir wollen $n-1$ Scheiben von Turm 1 nach Turm 2 verschieben:
2. $n-2$ Scheiben von Turm 1 nach Turm 3 verschieben
3. Eine Scheibe von Turm 1 nach Turm 2 verschieben
4. $n-2$ Scheiben von Turm 3 nach Turm 2 verschieben

Türme von Hanoi

Dies können wir in einem Rekursiven Algorithmus festhalten:

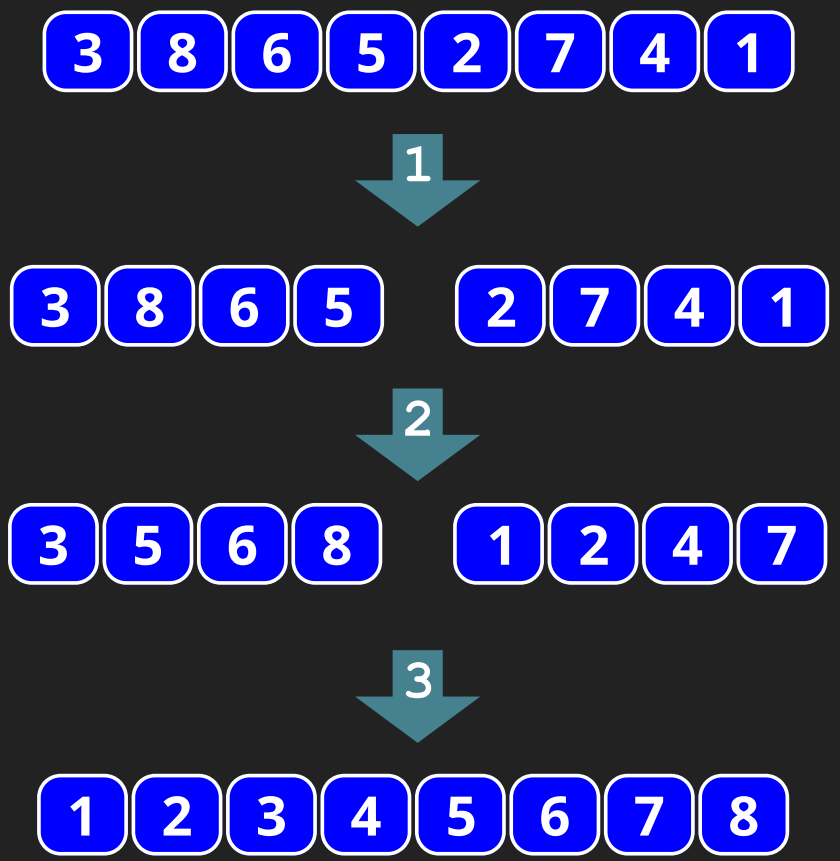
```
1 hanoi(int n, int von, int nach){
2     if(n == 1) movetop(von,nach);
3     int other = 6 - von - nach;
4     hanoi(n-1,von,other);          // [2]
5     movetop(von,nach);             // [3]
6     hanoi(n-1,other,nach);         // [4]
7 }
```



Mergesort

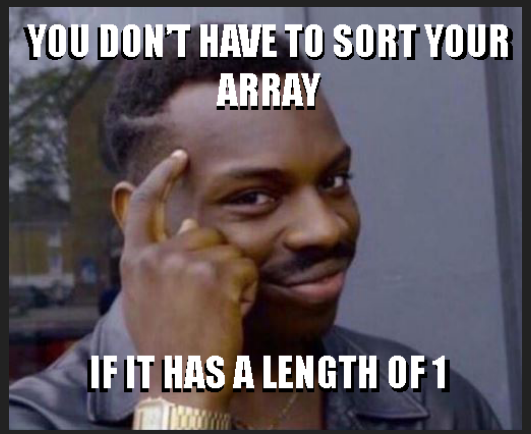
Mergesort ist ein Sortieralgorithmus, welcher ebenfalls durch divide and conquer funktioniert

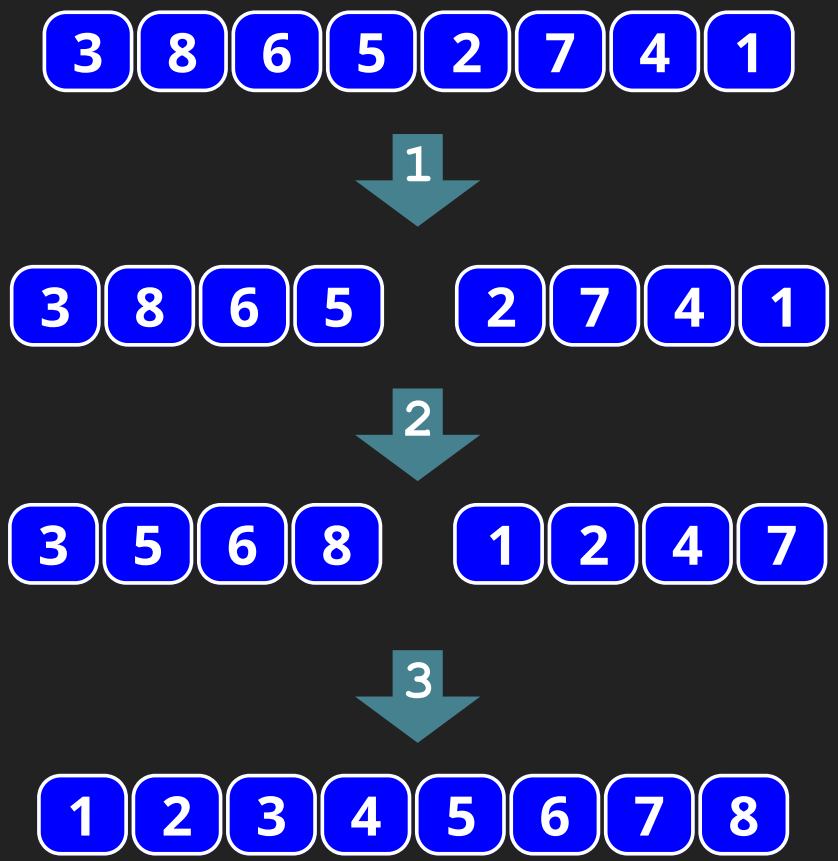
1. Teile die Liste in zwei halb so grosse Listen auf
2. Sortiere die halb so grossen Listen
3. Verbinde die beiden sortierten Listen zu einer einzigen sortierten Liste



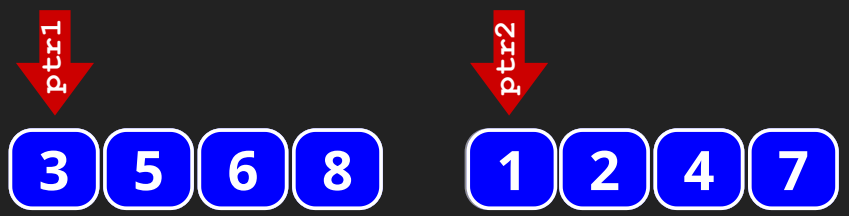
Mergesort

- Im Punkt 2 können wir die kleinen Teillisten wieder mit Mergesort sortieren
- Dies wiederholen wir, bis die Liste nur noch ein Element beinhaltet





Mergesort



Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

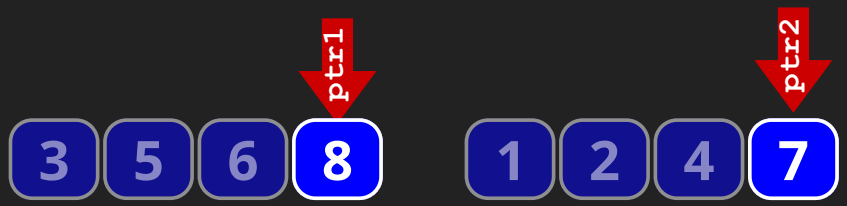
Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort



Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

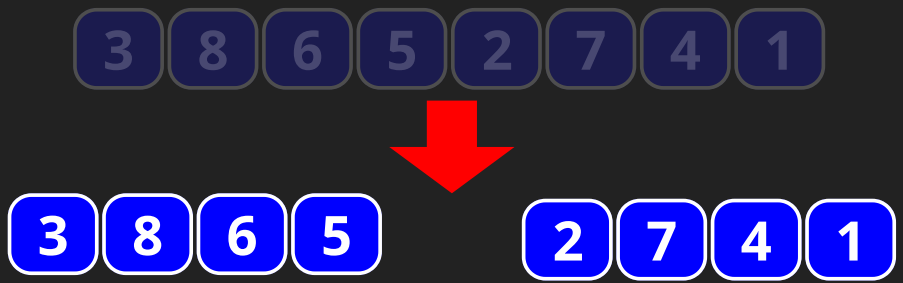
Um zwei Listen zu Mergen kann man:

- einen Zeiger auf den Anfang von beiden Teillisten setzen

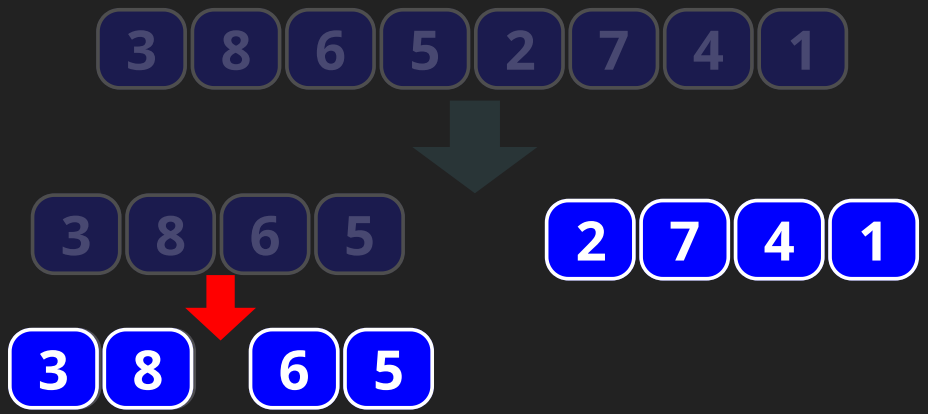
- Das kleinere Element dem Resultat hinzufügen und diesen Zeiger vergrössern

Mergesort

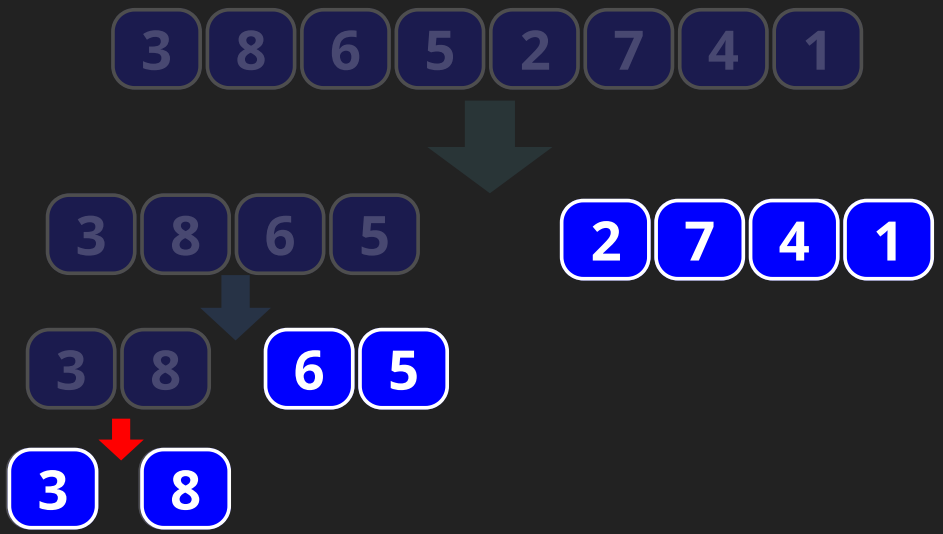
Mergesort



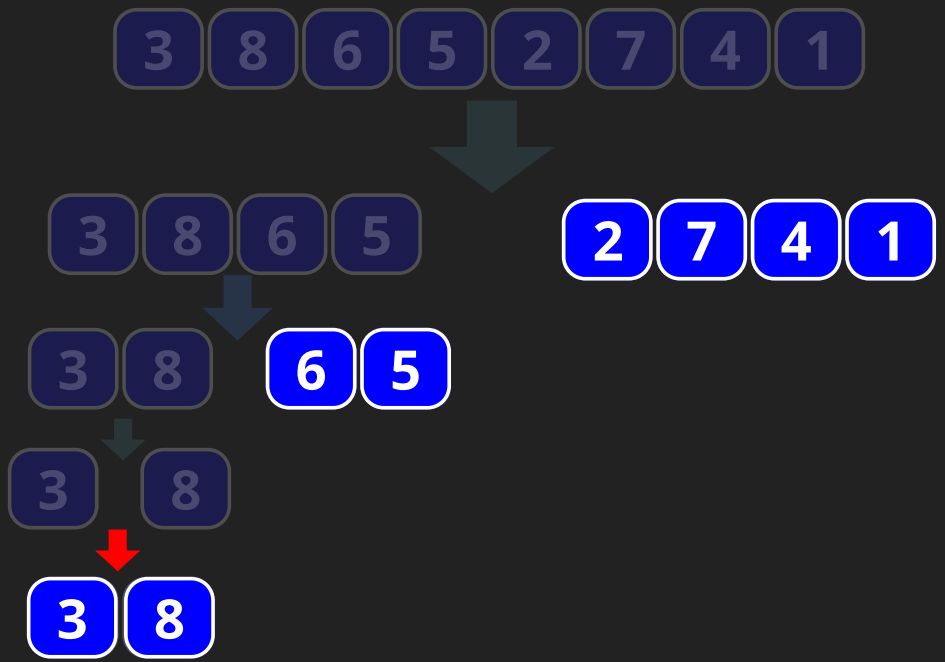
Mergesort



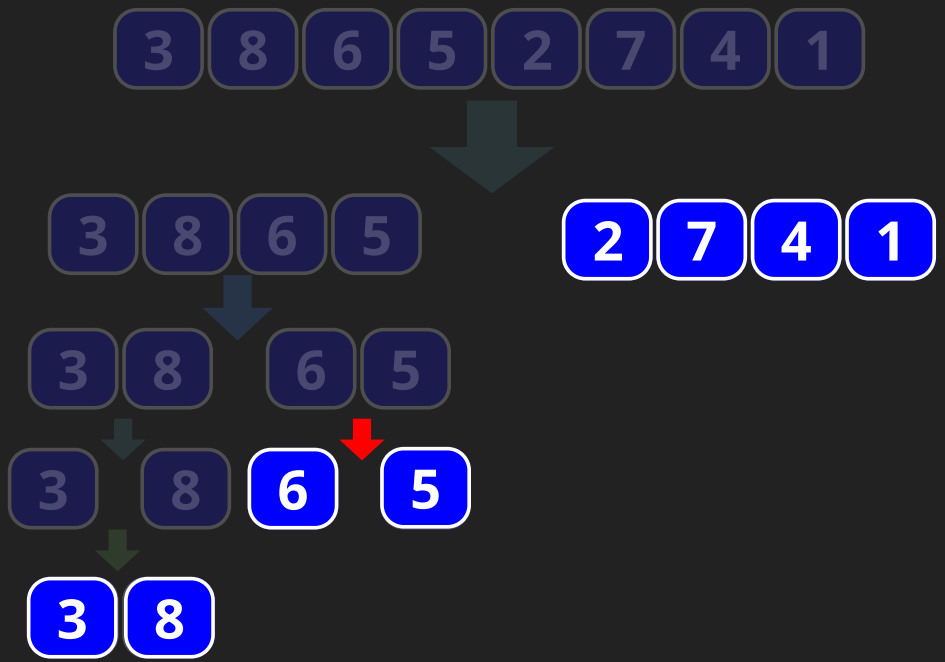
Mergesort



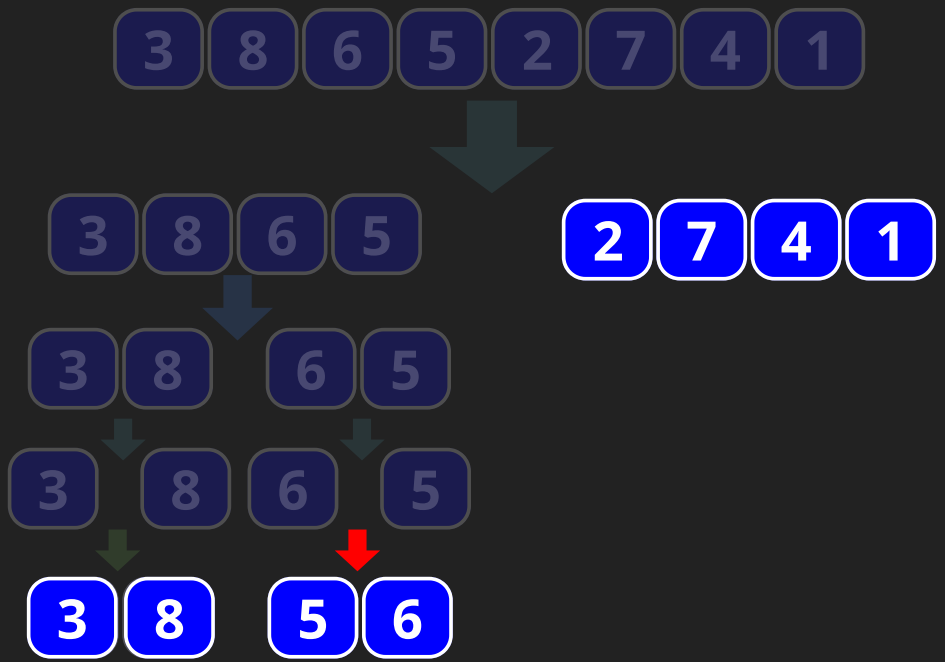
Mergesort



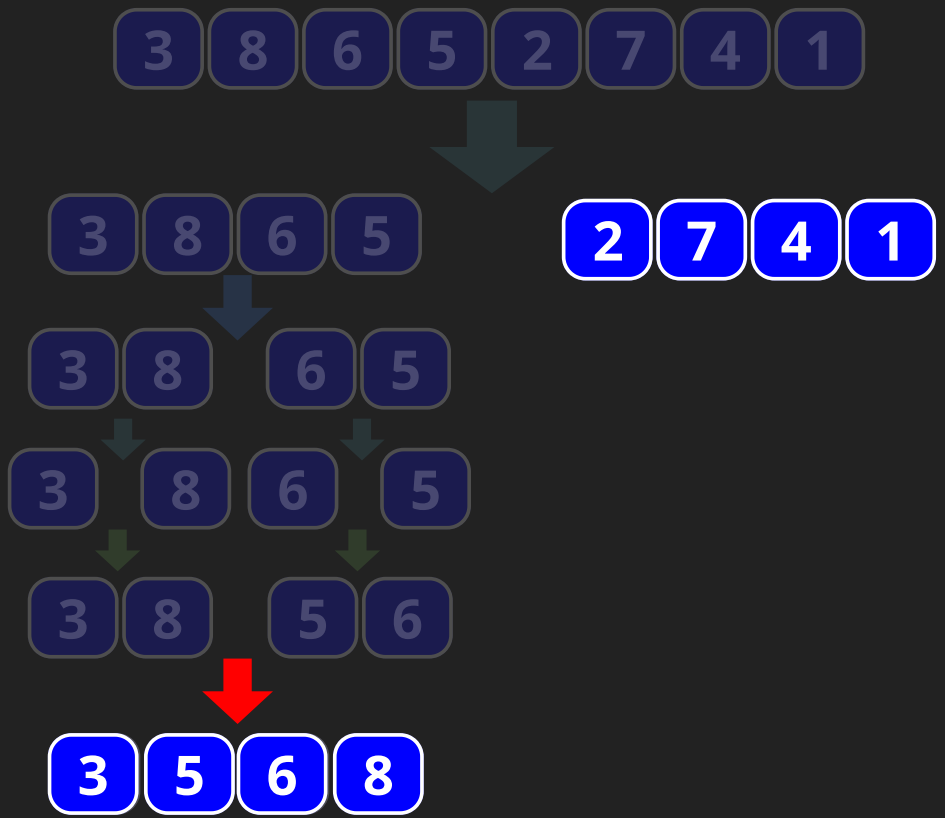
Mergesort



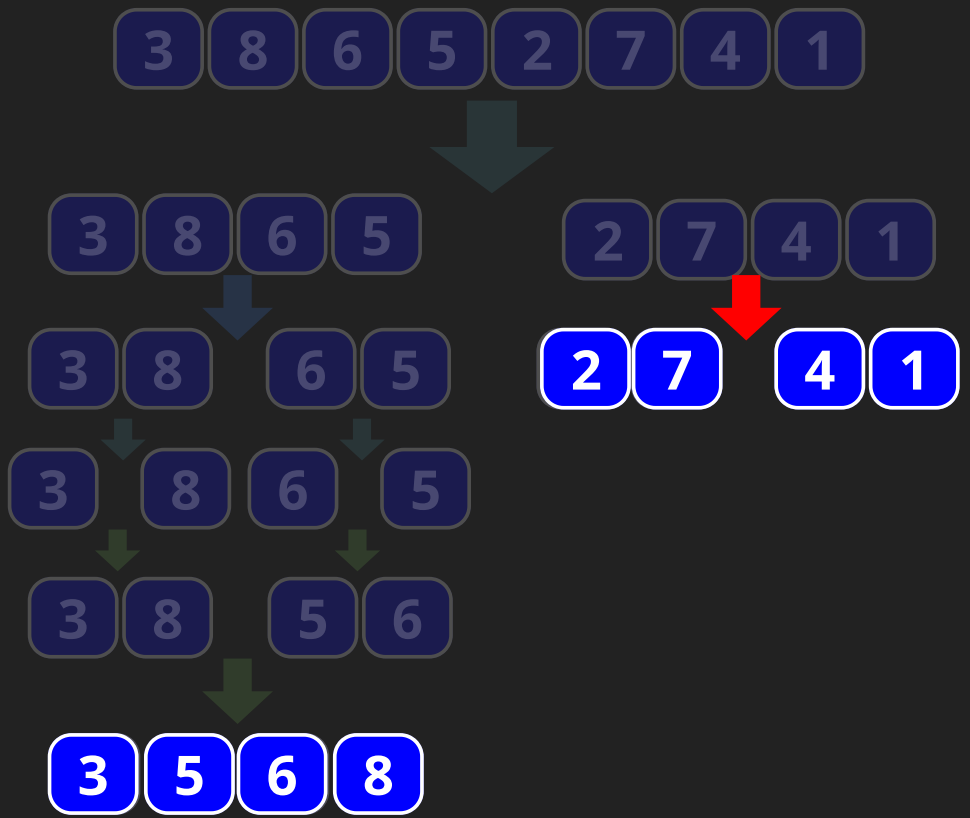
Mergesort



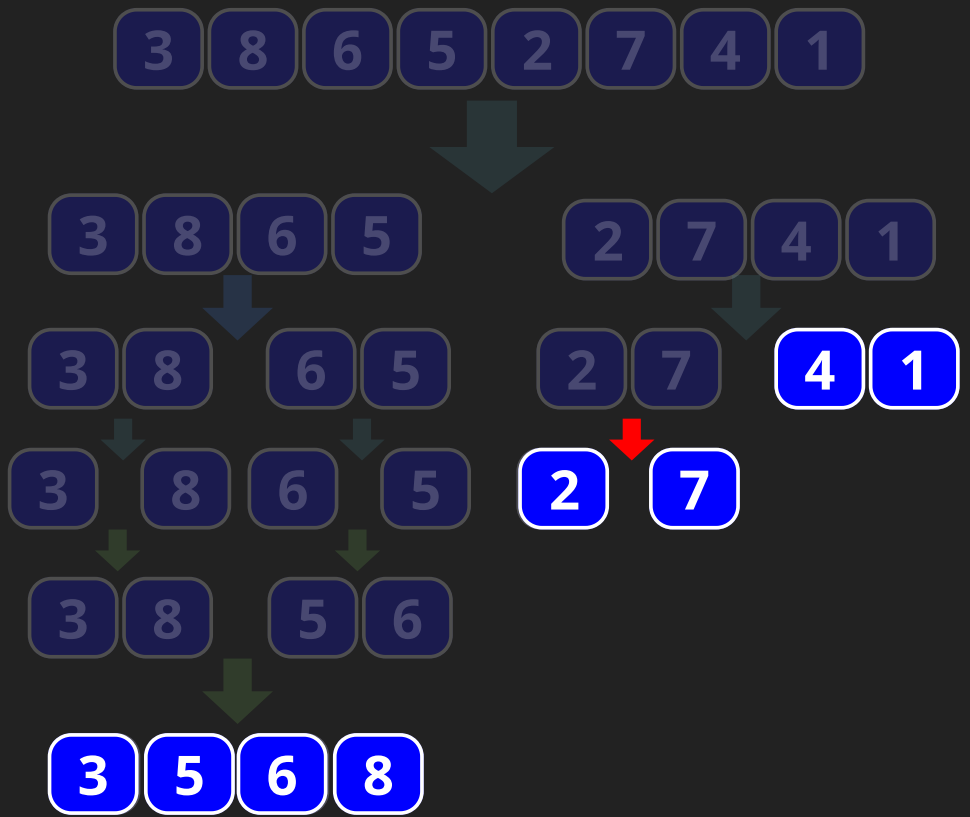
Mergesort



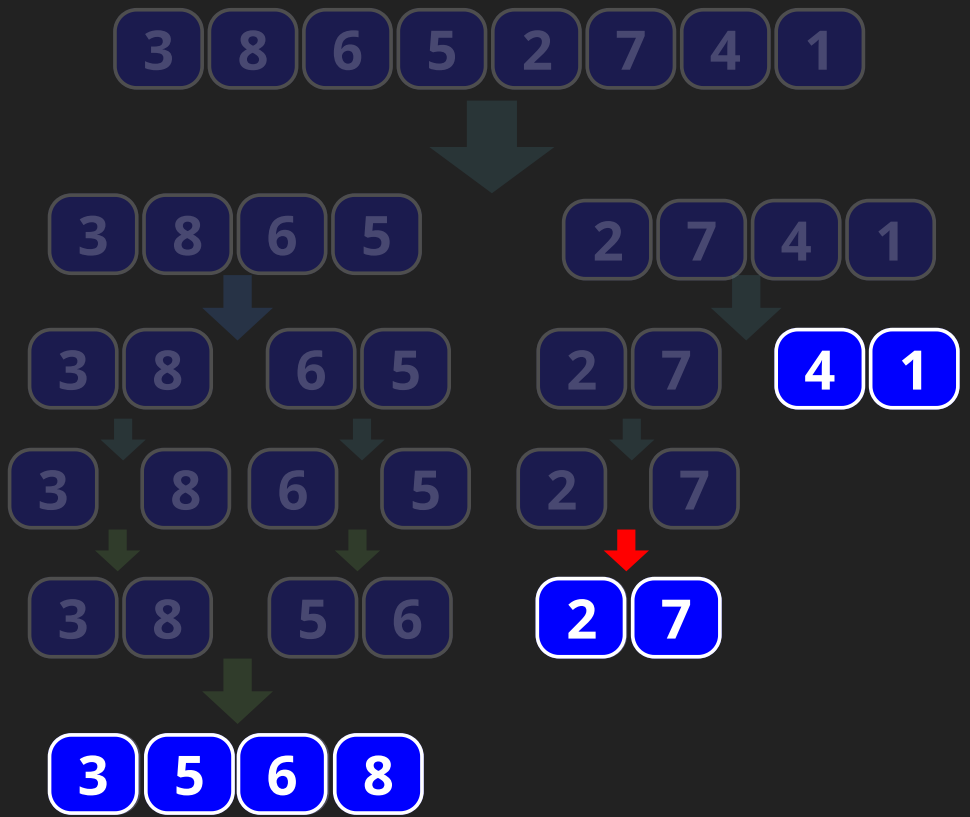
Mergesort



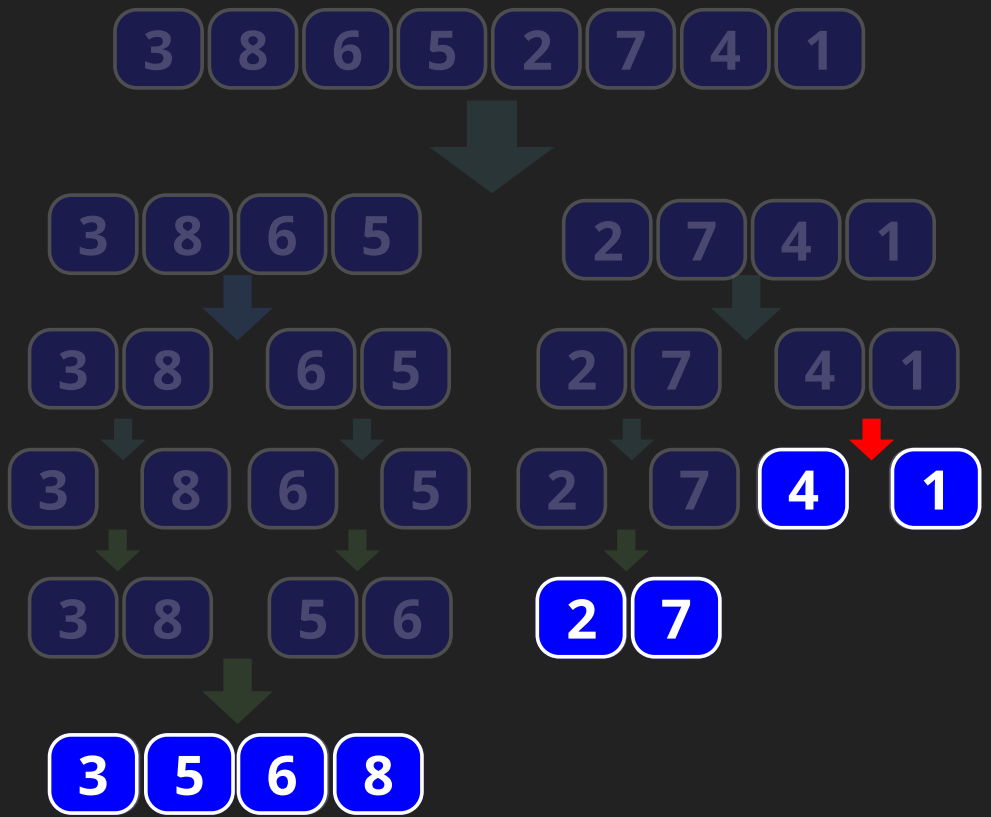
Mergesort



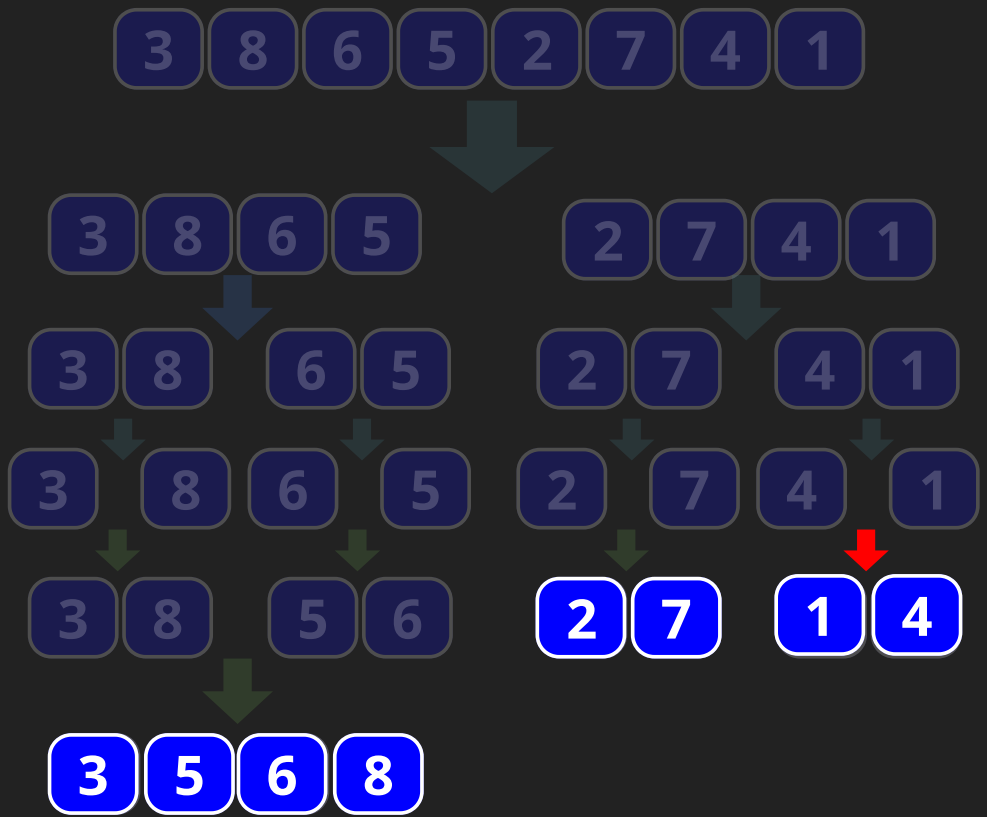
Mergesort



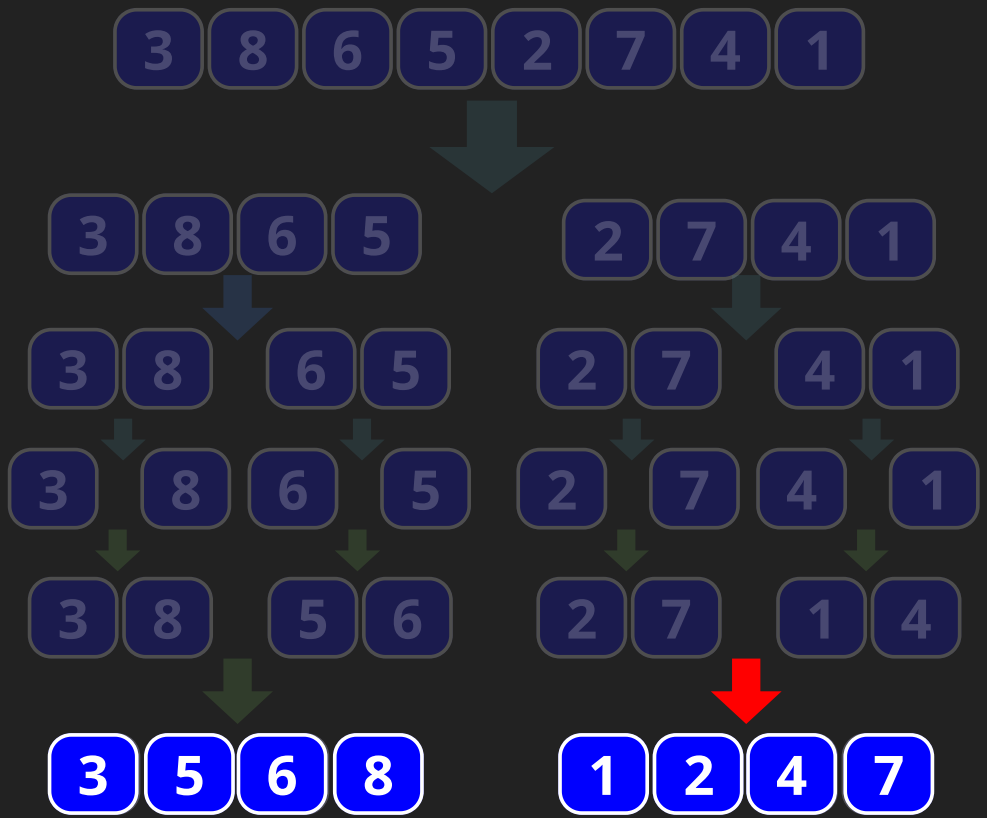
Mergesort



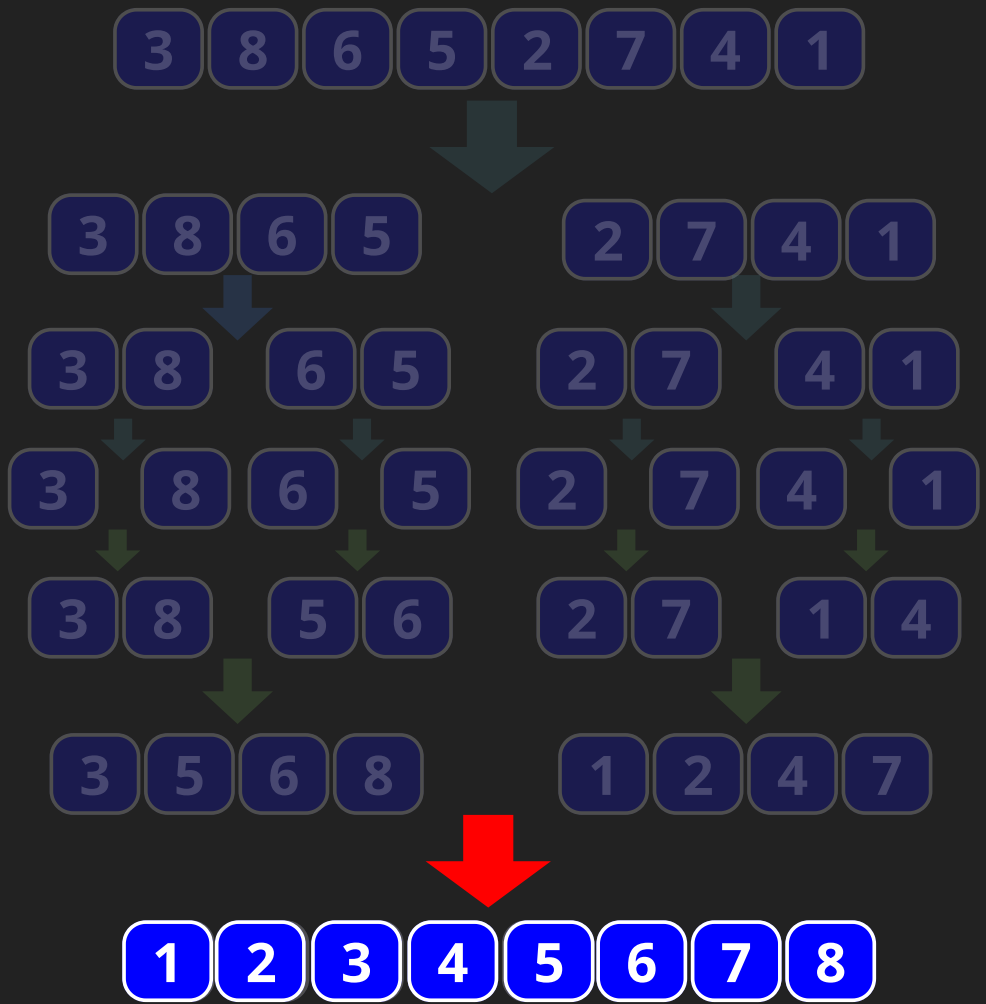
Mergesort



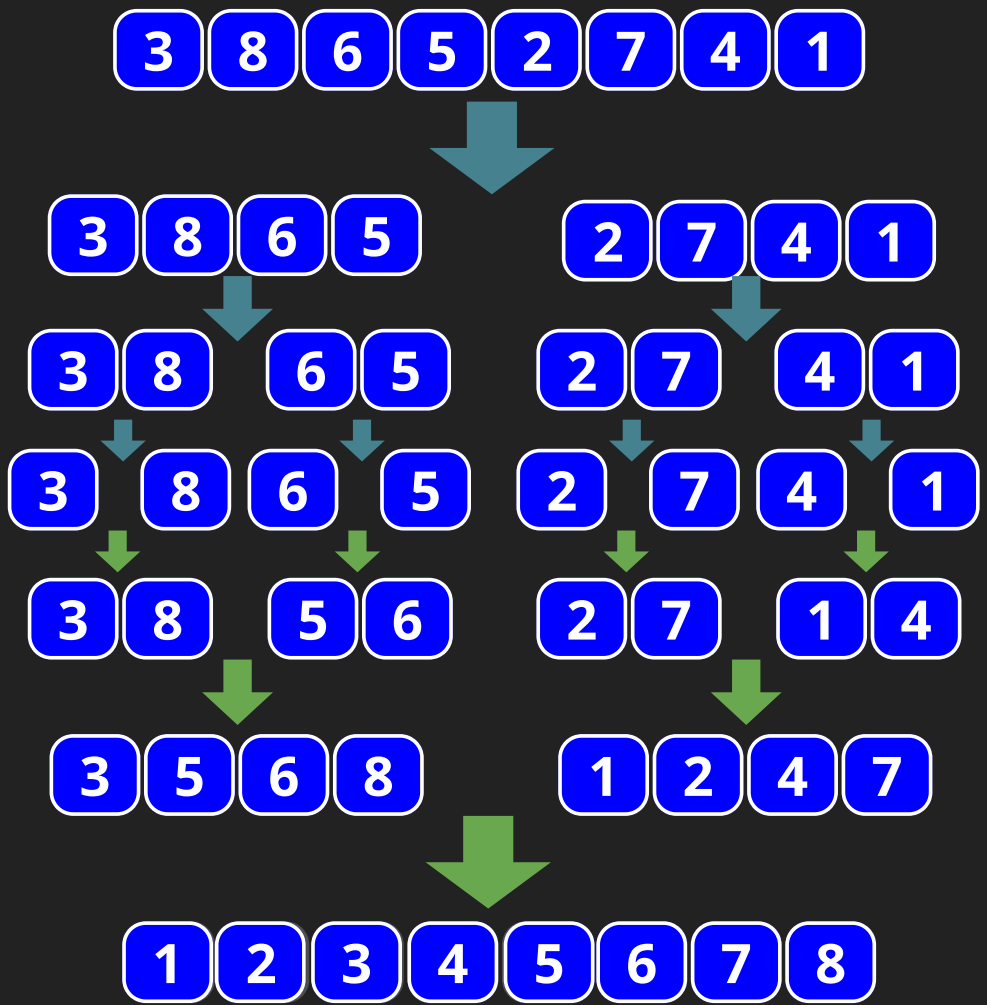
Mergesort



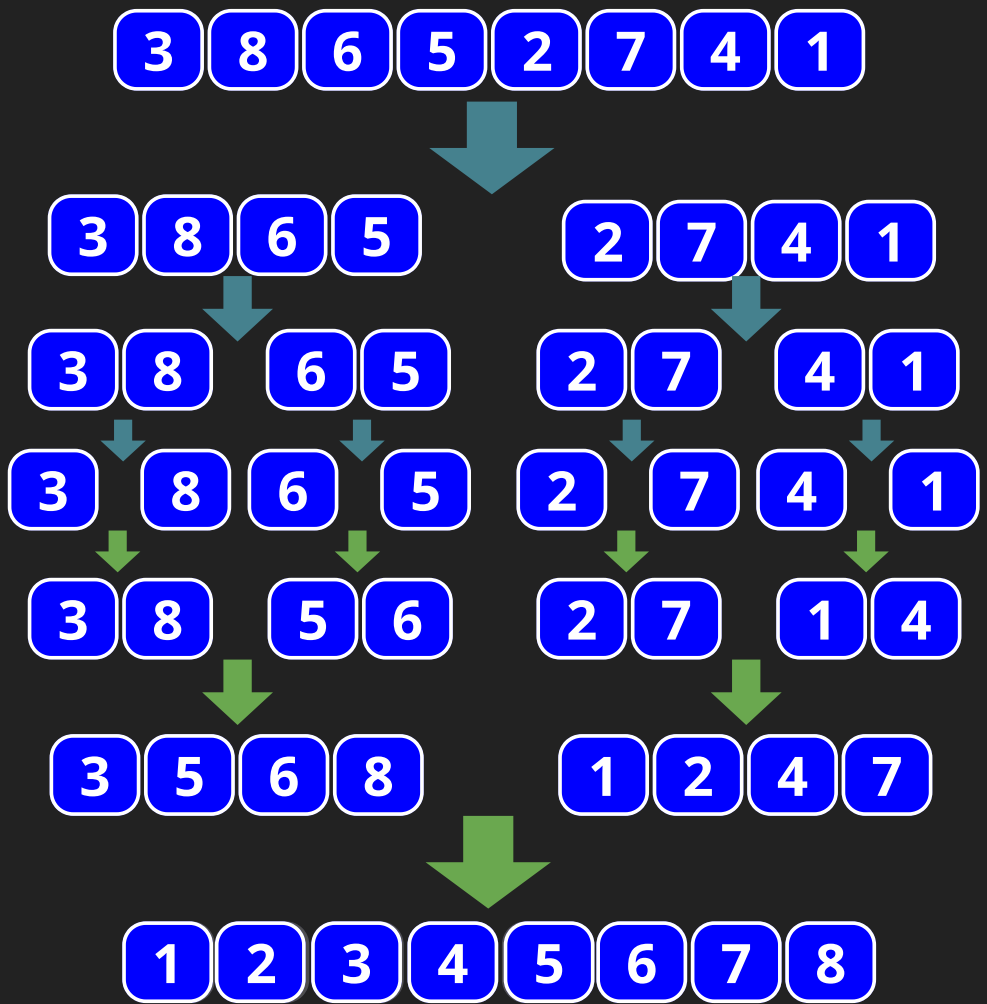
Mergesort


3 8 6 5 2 7 4 1
3 8 6 5
2 7 4 1
3 8 6 5
2 7 4 1
3 8 6 5
2 7 4 1
3 8 5 6
2 7 1 4
3 5 6 8
1 2 4 7
1 2 3 4 5 6 7 8

Mergesort


3 8 6 5 2 7 4 1
3 8 6 5
2 7 4 1
3 8 6 5
2 7 4 1
3 8 6 5
2 7 4 1
3 8 5 6
2 7 1 4
3 5 6 8
1 2 4 7
1 2 3 4 5 6 7 8

Mergesort


3 8 6 5 2 7 4 1
3 8 6 5
2 7 4 1
3 8 6 5
2 7 4 1
3 8 6 5
2 7 4 1
3 8 5 6
2 7 1 4
3 5 6 8
1 2 4 7
1 2 3 4 5 6 7 8

Laufzeitkomplexität

- Beschreibt Anzahl Rechenschritte die ein Algorithmus für das Lösen in Abhängigkeit von der Inputgrösse n braucht
- Wir betrachten die **asymptotische** Laufzeit, also nicht wie gross der Zeitaufwand ist, sondern wie schnell er im Verhältnis zur Inputgrösse wächst.
- Diese Abschätzungen machen wir mit der Landau-Notation (Big O notation)
- Wir betrachten nur den am schnellsten Wachsenden Summanden und ignorieren alle konstanten Faktoren

Laufzeitkomplexität

$$\frac{n}{n+5} \Rightarrow O(1)$$

$$\frac{1}{2^{n!}} + 10 + n^2 + 3 * n^5 \Rightarrow O(n^5)$$

$$\frac{1}{7}(n! + 2^n + n^2) \Rightarrow O(n!)$$

$$log(n) + n * log_3(n) \Rightarrow O(n * log(n))$$

$$2^{n+5} \Rightarrow O(2^n)$$

Laufzeitkomplexität Mergesort

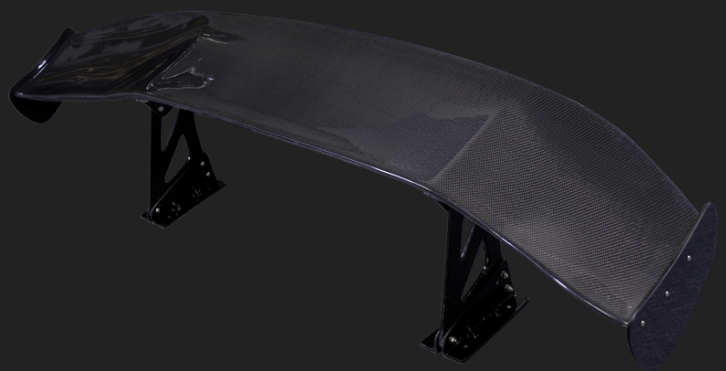
$$T(n) \approx 2^1 \cdot T(\frac{n}{2^1}) + n$$
$$T(n) \approx 2^1 \cdot (2 \cdot T(\frac{n}{2^2}) + \frac{n}{2}) + n$$
$$T(n) \approx 2^2 \cdot T(\frac{n}{2^2}) + 2 \cdot n$$
$$T(n) \approx 2^3 \cdot T(\frac{n}{2^3}) + 3 \cdot n$$
$$T(n) \approx 2^4 \cdot T(\frac{n}{2^4}) + 4 \cdot n$$
$$\dots$$

$$T(n) \approx 2^{log_2(n)} \cdot T(1) + log_2(n) \cdot n$$
$$T(n) \approx n \cdot T(1) + log_2(n) \cdot n$$

$\Rightarrow$ Laufzeitkomplexität von Mergesort: $O(n \cdot log(n))$

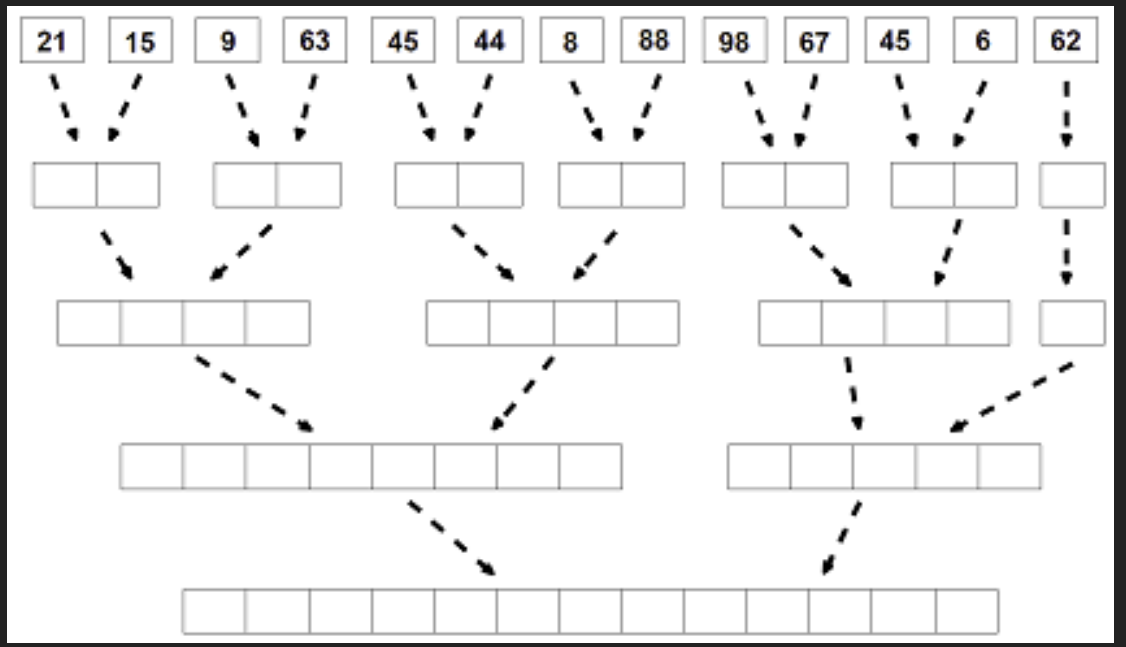

K!

Vorbesprechung



Mergesort Textaufgaben

- Führt Mergesort von Hand aus
- Plottet die Messungen von Aufgabe 3 mit einem log-log plot



Mergesort Programmieren

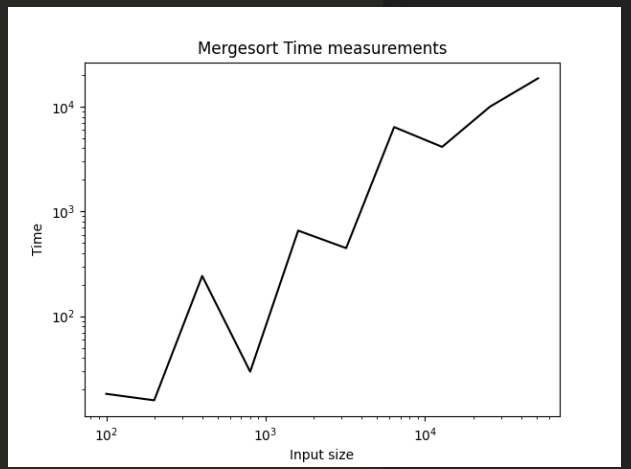
- Erstellt eine Hilfsfunktion, welche den Array in einem Bestimmten Intervall [begin, end) sortiert
- Macht Messungen um zu schauen ob die Ausführungszeit wirklich Logarithmisch steigt

```java
public static void main(String[] args) {
    int counts[] = {100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200, 102400};

    for (int count : counts) {
        double result = measure(count);
        System.out.println(String.format("%d: %.2f ms", count, result));
    }
}
```

Mergesort Textaufgaben

Tipp: Die coolen Kids heutzutage benutzen Python mit Matplotlib

```python
#!/usr/bin/python3
from matplotlib import pyplot as plt
import random

if __name__ == "__main__":

    counts = [100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, 51200]
    # TODO: Replace line below with your measurements:
    times = [random.random()*c for c in counts]

    # Set title and axis labels
    plt.title("Mergesort Time measurements")
    plt.xlabel("Input size")
    plt.ylabel("Time [s]")

    # Set axis scale to log-log plot
    plt.xscale("log")
    plt.yscale("log")

    # Plot our data on the plot
    plt.plot(counts, times, color="black")

    # Save an image of the plot to plot.png
    plt.savefig("plot.png")

    # Show the resulting plot
    plt.show()
```



(not sponsored)

Türme von Hanoi

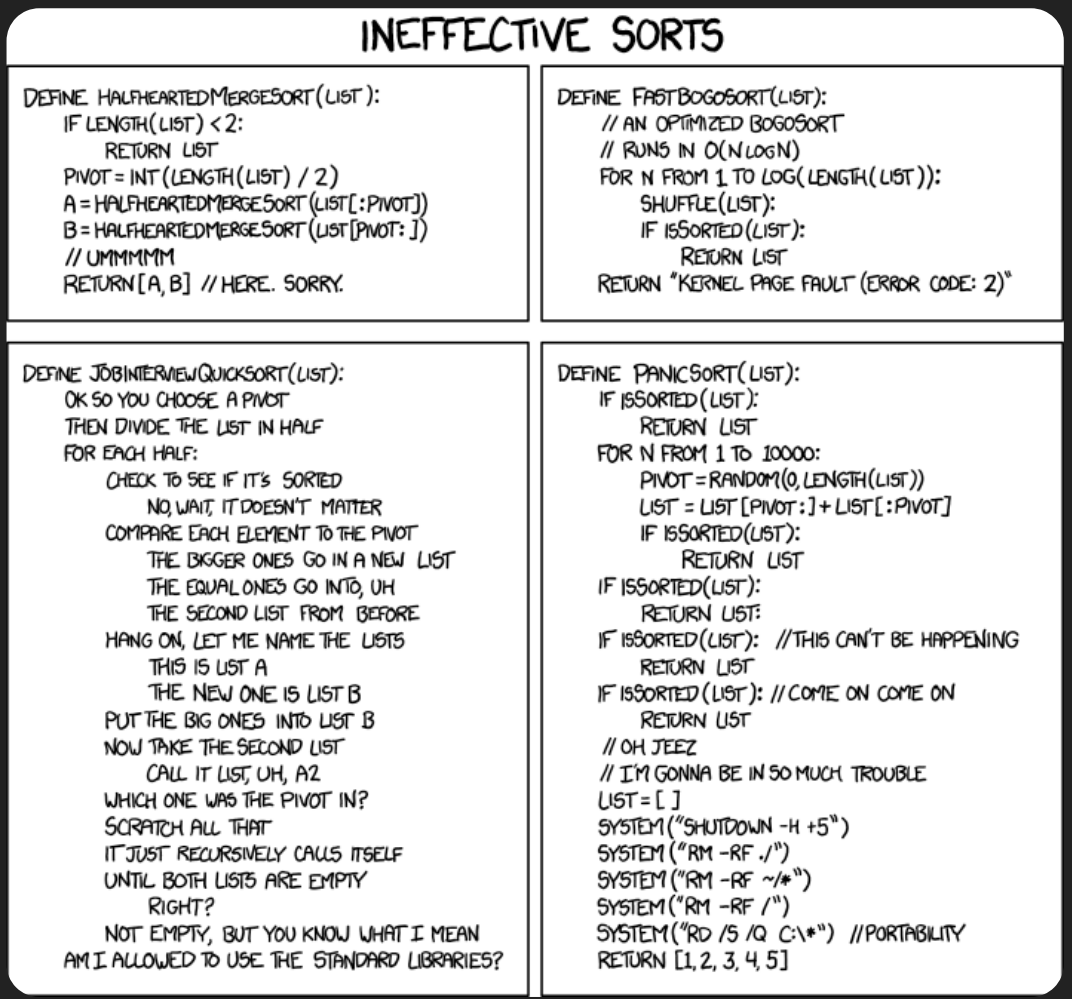
- Bestimmt für jeden Schritt welche Türme jeweils *nicht* verwendet werden um alle Scheiben von Turm 1 nach Turm 3 zu verschieben

- Erstellt einen Iterativen Algorithmus um das Problem zu lösen
- Funktioniert der Iterative Algorithmus auch bei Höhe 5?

Reversi [Teil 4]

- Implementiert Alpha/Beta Algorithmus
- Verwendet euren MiniMax Spieler von letzter Woche oder die Musterlösung als Vorlage
- Gebt zusätzlich noch alpha / beta Parameter mit, um eure Cuts zu machen.
- Versucht weiter eure Feldbewertung zu verbessern

Viel Spass!



INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST):  //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): // COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*")  //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```