

# Organisatorisches

- ▶ Teams dürfen **nicht** gewechselt werden!

# Sortieren

# INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):  
  IF LENGTH(LIST) < 2:  
    RETURN LIST  
  PIVOT = INT(LENGTH(LIST) / 2)  
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])  
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])  
  // UMMMMMM  
  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 JOBIINTERVIEWQUICKSORT(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]
```

# Allgemeines

- ▶ Stabil vs. Instabil
- ▶ Vergleichsbasiert bestenfalls in  $O(n \log(n))$
- ▶ Kostenarten: Vergleiche, Inversionen

# Allgemeines

- ▶ Stabil vs. Instabil
- ▶ Vergleichsbasiert bestenfalls in  $O(n \log(n))$
- ▶ Kostenarten: Vergleiche, Inversionen
- ▶ Wichtige Algorithmen:
  - ▶ Mergesort
  - ▶ Insertionsort
  - ▶ Selectionsort
  - ▶ Quicksort
  - ▶ Radixsort (später)
  - ▶ Heapsort (noch später)

# Insertionsort

- ▶  $O(n^2)$
- ▶ Potentiell schnell für sehr kleine Arrays

# Selectionsort

- ▶  $O(n^2)$  Vergleiche
- ▶  $O(n)$  Inversionen

# Mergesort

- ▶  $O(n \log n)$
- ▶ Stabil
- ▶ Teile und Herrsche (Divide and Conquer)



# Quicksort

- ▶ erwartet  $O(n \log n)$ , worst-case  $O(n^2)$
- ▶ Instabil
- ▶ Teile und Herrsche (Divide and Conquer)
- ▶ In der Praxis sehr effizient

# Sonstige

## Slowsort

- ▶ Vervielfache und Kapituliere (Multiply and Surrender)
- ▶ Mergesort mit extrem ineffizientem merge
- ▶  $\Omega\left(n^{\frac{\log_2(n)}{2+\epsilon}}\right)^1$

---

<sup>1</sup><https://de.wikipedia.org/wiki/Slowsort>

# Sonstige

## Slowsort

- ▶ Vervielfache und Kapituliere (Multiply and Surrender)
- ▶ Mergesort mit extrem ineffizientem merge
- ▶  $\Omega\left(n^{\frac{\log_2(n)}{2+\epsilon}}\right)^1$

## Bogosort

1. Wenn sortiert: terminiere
2. Mische zufällig und gehe zu 1.

---

<sup>1</sup><https://de.wikipedia.org/wiki/Slowsort>

# Kreativaufgaben

# Kreativaufgaben

1. Gegeben:  $\text{median}(\text{Array}[n]) \in O(n)$ , unsortiertes Array A
  - ▶ Finde das  $\frac{1}{3}$ -Perzentil in  $O(n)$
  - ▶ Finde alle  $\frac{1}{3^k}$ -Perzentile in  $O(n)$  (nicht in  $O(n \cdot k)$ )
  - ▶ Es existieren inplace-Lösungen

# Kreativaufgaben

1. Gegeben:  $\text{median}(\text{Array}[n]) \in O(n)$ , unsortiertes Array A
  - ▶ Finde das  $\frac{1}{3}$ -Perzentil in  $O(n)$
  - ▶ Finde alle  $\frac{1}{3^k}$ -Perzentile in  $O(n)$  (nicht in  $O(n \cdot k)$ )
  - ▶ Es existieren inplace-Lösungen
2. Finde einen vergleichsabsierten Sortieralgorithmus der
  - ▶ langsamer als Slowsort ist und
  - ▶ immer terminiert

# Kreativaufgaben

1. Gegeben:  $\text{median}(\text{Array}[n]) \in O(n)$ , unsortiertes Array A
  - ▶ Finde das  $\frac{1}{3}$ -Perzentil in  $O(n)$
  - ▶ Finde alle  $\frac{1}{3^k}$ -Perzentile in  $O(n)$  (nicht in  $O(n \cdot k)$ )
  - ▶ Es existieren inplace-Lösungen
2. Finde einen vergleichsabsierten Sortieralgorithmus der
  - ▶ langsamer als Slowsort ist und
  - ▶ immer terminiert
3. Finde einen vergleichsbasierten Sortieralgorithmus der
  - ▶ minimal viele Permutationen durchführt
  - ▶  $O(n^2)$  Vergleiche braucht