

1 - Asymptotische Aufwandsordnungen

1. Sortieren Sie für jede Gruppe die Funktionen aufsteigend nach ihrer asymptotischen Aufwandsordnung (Θ)

(a)

$$f_1(n) = n^{0.999999} \log n$$

$$f_2(n) = 10000000n$$

$$f_3(n) = 1.000001^n$$

$$f_4(n) = n^2$$

(b)

$$f_1(n) = 2^{2^{1000000}}$$

$$f_2(n) = 2^{1000000n}$$

$$f_3(n) = \binom{n}{2}$$

$$f_4(n) = n\sqrt{n}$$

(c)

$$f_1(n) = n^{\sqrt{n}}$$

$$f_2(n) = 2^n$$

$$f_3(n) = n^{10} \cdot 2^{\frac{n}{2}}$$

$$f_4(n) = \sum_{i=1}^n (i+1)$$

2. Bestimmen Sie die Laufzeiten folgender Algorithmen:

(a)

```
def foo(arr):  
    sum = 0  
    product = 1  
    for elm in arr:  
        sum += elm  
    for elm in arr:  
        product *= elm  
    return sum, product
```

(b)

```
def print_pairs(arr):  
    for i in range(len(arr)):  
        for j in range(len(arr)):  
            print(str(arr[i]) + ', ' + str(arr[j]))
```

```
(c)      def print_pairs(arr1, arr2):
          for i in range(len(arr1)):
              for j in range(len(arr2)):
                  print(str(arr1[i] + ', ' + str(arr2[j])))
```

```
(d)      def print_hello_world(n):
          for i in range(n):
              for j in range(100000):
                  print('Hello_World_' + str(n))
```

```
(e)      def reverse(arr):
          for i in range(int(len(arr) / 2)):
              arr[i], arr[len(arr) - 1 - i] =
                  arr[len(arr) - 1 - i], arr[i]
          return arr
```

```
(f)      def factorial(n):
          if n == 0:
              return 1
          return n * factorial(n - 1)
```

```
(g)      def print_powers_of_2(n):
          number = 1
          while number <= n:
              print(number)
              number *= 2
```

```
(h)      def print_powers_of_k(n, k):
          number = 1
          while number <= n:
              print(number)
              number *= k
```

3. (a) Ein Folge von n ganzen Zahlen $p(1), p(2), \dots, p(n)$ ist gegeben. Jedes Element der Folge ist verschieden und erfüllt $1 \leq p(x) \leq n$. Schreiben Sie einen Algorithmus, der für jedes x mit $1 \leq x \leq n$ eine ganze Zahl y findet, so dass $p(p(y)) = x$.
- (b) Bestimmen Sie die Zeitkomplexität Ihres Algorithmus aus Aufgabe a in Abhängigkeit von der Anzahl n der Elemente der Folge.
- (c) Falls der Algorithmus nicht in $\mathcal{O}(n)$ läuft, finden Sie einen Algorithmus, der dies in $\mathcal{O}(n)$ tut.

4. Car Parking - Google Interview Aufgabe

- (a) Entwickeln Sie einen Algorithmus, der folgendes Problem löst:



Rearrange an array using swap with 0.

You have two arrays `src`, `tgt`, containing two permutations of the numbers $0..n-1$. You would like to rearrange `src` so that it equals `tgt`. The only allowed operations is "swap a number with 0", e.g. $\{1,0,2,3\} \rightarrow \{1,3,2,0\}$ ("swap 3 with 0"). Write a program that prints to `stdout` the list of required operations.

Practical application:

Imagine you have a parking place with n slots and $n-1$ cars numbered from $1..n-1$. The free slot is represented by 0 in the problem. If you want to rearrange the cars, you can only move one car at a time into the empty slot, which is equivalent to "swap a number with 0".

Example:

`src={1,0,2,3}; tgt={0,2,3,1};`

- (b) Führen Sie eine Effizienzanalyse für Ihren Algorithmus durch.