# O-Notation

Um Laufzeiten verschiedener Algorithmen und somit deren Effizienz vergleichen zu können, wurde das O-Kalkül entwickelt. Anstatt die Anzahl der verwendeten Variablen oder Zuweisungen zu vergleichen, wird hierbei so weit abstrahiert, bis nur noch die Anzahl Durchläufe je Objekt übrigbleibt. (Beispiel: For Each Schleife über alle n-Objekte eines Arrays: O(n)).

## Idee

### Abstraktionsschritte

1. Art der Elementaroperationen wird nicht unterschieden (z.B. Variablenzuweisungen, oder Additionen)
2. Menge aller Eingaben wird aufgeteilt in sog. Komplexitätsklassen (Umfang der Eingabe, z.B. n Elemente einer Liste)
3. Betrachtung nur der Größenordnung der Laufzeitfunktion T(n) durch weglassen additiver und multiplikativer Konstanten.

### Umfang und Aufwand

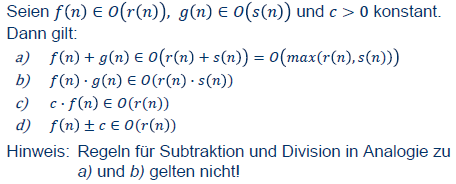
Hängt der Aufwand nicht nur vom Umfang (n-Elemente einer Liste) sondern auch vom Aufwand ab, wird der Algorithmus in drei Stufen bewertet:

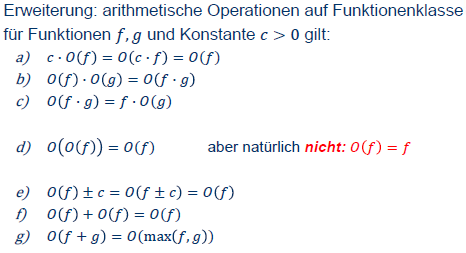
* Worst-case Bsp: Eine Liste wird komplett durchlaufen, da das gesuchte Element das letzte ist
* Best-case (Kürzester / kleinster Aufwand: Man schlägt das Telefonbuch auf und zeigt direkt auf die gesuchte Nummer)
* Average-case (Durchschnittlicher Aufwand. Treten Worst und Best Case je zu einer 50% Häufigkeit auf, wäre der Durchschnitt 0,5\* ( T­­­worst(n) + Tbest(n) )
* Abstraktion: O(n)-Notation

### Asymptotischer Aufwand

Die exakten Aufwände zweier Algorithmen können sich stellenweise sehr ähneln, aber auch zu anderen Zeiten stark auseinandergehen. Daher ist lediglich die Größenordnung der Aufwände für einen Vergleich relevant.

## Rechenregeln

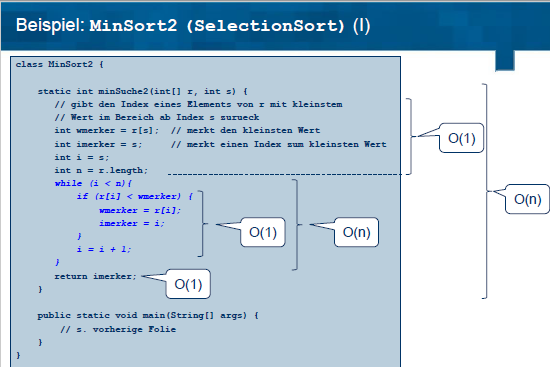
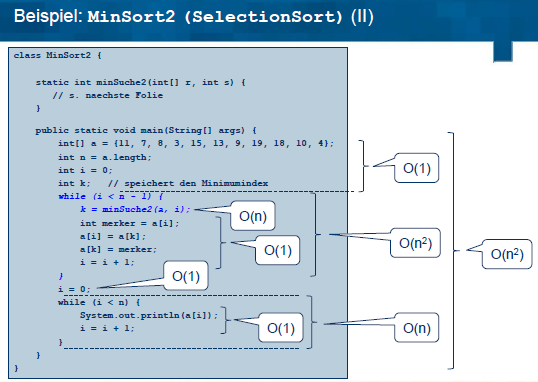
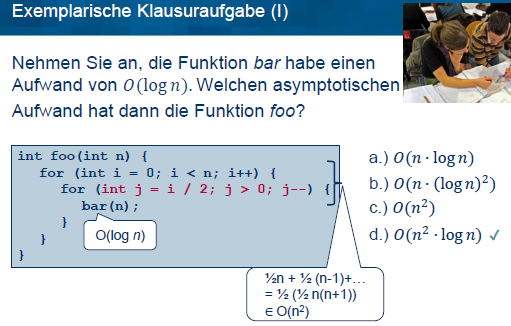
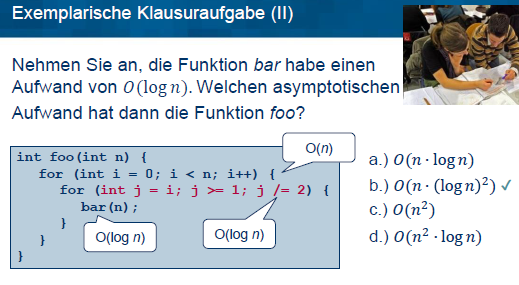




# Beispiele

1. Der Aufwand eines Schleifendurchlaufs sei O(n). Die Schleife wird O(g(n)) mal durchlaufen.

Aufwand für die gesamte Schleife: O ( f(n) \* g(n) ) entspricht O(n²)

1. Eine Schleife wird n-mal durchlaufen. Innerhalb der Schleife entsteht ein konstanter Aufwand (z.B. eine Addition oder Zuweisung). Der Aufwand beträgt somit O(n)
2.   
   Betrachtung der Aufwände vom kleinsten zum Größten. Zeilen, welche einfach durchlaufen werden haben einen Aufwand von O(1). Die Zuweisungen innerhalb der While Schleife O(1) werden n mal Durchlaufen O(n). Da O(1) in O(n) enthalten ist, wenn man sich die zugehörigen Funktionsgraphen vorstellt, wählt man die obere und damit die, für die Größenordnung relevantere, Schranke O(n).
3.   
   Der Aufwand des Funktionsaufrufs k = minSuche2(a,i) entspricht O(n) (siehe blaue Schrift)  
   Da dieser Aufwand durch die While Schleife n mal entsteht, berechnet man für den Gesamtaufwand der While Schleife O(n \* n) = O(n²)  
   Die zweite While Schleife darunter hat einen Aufwand von O(n) und erhöht somit die obere Schranke (man denke an den Funktionsgraph) nicht. O(n) < O(n²)  
   Somit ist der Gesamtaufwand des Sortieralgorithmus O(n²)
4.   
   Vorgehensweise:   
   Die Funktion bar(n) hat laut Angabe den Aufwand O(log n). Betrachtet man zusätzlich die erste For-schleife, welche n mal durchlaufen wird ist der Aufwand O(n \* log n).  
   Die zweite Schleife wird mit steigendem i immer öfter Durchlaufen und hat somit einen Aufwand von O(n). Mit den bisherigen Ergebnissen bedeutet dies für den Gesamtaufwand: O( n \* n \* log n) = O( n² \* log n)
5. 

# Typische Laufzeitklassen und zugehörige Algorithmen

