Algorithmen und Datenstrukturen: Typische Komplexitätsklassen

Bezeichnung	Komplexitäts klasse	Anwendungsbeispiele	Code	Beschreibung
Konstant	O (1)	Zuweisung, Addition von Zahlen	`int size = 2 + 3;`	Hier haben wir zwei Operationen: Zum einen die Addition und zum anderen die Zuweisung des Wertes an die Variable `size`. Beides braucht zwar unterschiedlich lange (gemessen an der Ausführungszeit), jedoch ist sehr schnell durchführbar und nicht abhängig von der Eingabegrösse n. Deshalb konstanter Zeitaufwand

Logarithmisch	O (log n)	Binäre Suche	`int l = -1, h = data.length;`
			`while (h - l > 1) {`
			`  int m = (l + h) / 2;`
			`  if ((data[m]) <= gesucht) l = m;`
			`  else h = m;`
			`}`
			`return l;`
				Bei jedem Schritt wird die Suchmenge auf die Hälfte reduziert. (Gegensatz zu Verdoppelung, bei 2^n) Das führt zu einer Logarithmischen Laufzeit.

Linear	O (n)	Finden des minimalen Wertes in einem Array der Länge n	`int min = data[0];`
			`for (int i=1; i < data.length; i++){`
			`  if (data[i] < min)`
			`  min= data[i];`
			`}`
				Um das Minimum zu finden muss jeder Wert einmal angesehen werden. Das Array hat die Länge n. Deshalb braucht es n Zugriffe. Der Suchaufwand steigt linear zur Anzahl Elemente.

n-log-n	O (n log n)	Merge-Sort	Siehe Merge-Sort	Bei Merge-Sort wird das Problem jeweils in zwei gleich grosse Probleme aufgeteilt, separat gelöst und dann wieder zusammengeführt. Herleitung über Rekursive Funktion: T(n) = 2 T(n/2) + n

Quadratisch	O (n^2)	Vergleichen alle Paare, z.B 2-Sum	`int count =0;`
			`for (int i=0; i < data.length; i++){`
			`  for (int j =i+1; j < data.length; j++){`
			`  if (data[i] + data[j] == 0) ++count;`
			`}}`
				Hier werden alle Paare verglichen und gezählt, wie oft die Summe 0 ergibt. Dazu muss jeder Wert (also n Werte) mit jedem anderen Wert (nochmals n Werte) verglichen werden. Das führt zu quadratischem Aufwand.

Kubisch	O (n^3)	Vergleichen aller Tripppel, z.B 3-Sum	`int count = 0;`
			`for (int i = 0; i < data.length; i++){`
			`  for (int j = i + 1; j < data.length; j++){`
			`  for (int k = j + 1; k < data.length; k++){`
			`  if (data[i] + data[j] + data[k] == 0)`
			`  ++count;`
			`}}}`
				Hier werden Trippels verglichen und gezählt, wie oft die Summe 0 ergibt. Erklärung ist dieselbe wie bei 2-Sum, nur, dass hier noch eine weitere Schleife hinzukommt.

Exponentiell	O (2^n)	Brute-Force, z.B Sudoku oder Queens-Problem	Sudoku-Solver	Bei Backtracking (ohne Branch-and-Bound) werden alle Möglichkeiten ausprobiert. Beim Sudoku gibt es beispielsweise pro Feld die Möglichkeit aller Zahlen von 1- 9. Haben wir 2 Felder, gibt es insgesamt 81 Kombinationen (1-1, 1-2, usw. bis 9-9), also 9^2. Für n Felder gibt es somit 9^n Möglichkeiten.