```
public void bubbleSort(int[] arr) {
    boolean swapped = true;
    int j = 0;
    int tmp;
    while (swapped) {
        swapped = false;
        j++;
        for (int i = 0; i < arr.length - j; i++) {
            if (arr[i] > arr[i + 1]) {
                tmp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = tmp;
                swapped = true;
            }
        }
    }
}
```

- Probleme der Klasse **P**: mit deterministischenAlgorithmen in polynomialerZeit lösbar (polynomial)
- Probleme der Klasse **NP**: in polynomialerZeit nur mit nichtdeterministischen Algorithmen lösbar, sonst exponentielle Komplexität (nichtdeterministisch polynomial)

Eigenschaften Algorithmen
Terminierung-„in endlicher Laufzeit“ Ein Algorithmus ist **terminierend**, wenn er (bei jeder erlaubten Eingabe von Parameterwerten) nach endlich vielen Schritten abbricht.
Determinismus–eine Art „Vorbestimmtheit“, legt die Wahlfreiheit bei der Ausführung des Verfahrens fest.

Es gibt zwei Aspekte von Determinismus: **deterministisch** und **determiniert**.
Ein Algorithmus mit *deterministischem Ablauf* legt die Schrittfolge der auszuführenden Schritte eindeutig fest. Es gibt keine Variationsmöglichkeit in der Reihenfolge der Schritte.

Ein Algorithmus liefert ein *determiniertes Ergebnis*, wenn die selben Eingabewerte zu eindeutigen Ausgabewerten führen, die sich auch bei mehrfacher Durchführung des Algorithmus nicht ändern.

Rechnerprogramme erfüllen meistens beide Aspekte.
Nichtdeterminierte Ergebnisse sind meistens unerwünscht (, für die Analyse der Algorithmen aber wichtig). Sie können mit Hilfe von Zufallsgeneratoren simuliert werden.
Nichtdeterministische Abläufe sind wichtig beim Entwurf von Algorithmen.

Entwurfsprinzipien
Problemlösetechniken, Vorgehensweisen, die beim Algorithmenentwurf hilfreich sind:
- Schrittweise Verfeinerung - Problemreduzierung durch Rekursion -Einsatz von Algorithmenmustern

Schrittweise Verfeinerung
- Intuitive Problemlösungstechnik aus dem Alltag
- „top-down“ Methode, vom Allgemeinen zum Spezifischen

Ablauf:
- Beginne mit einer kurzen Folge abstrakter Lösungsschritte (in natürlicher Sprache)
- Verfeinere die Anweisungen Schritt für Schritt mit mehr Details (in Pseudocode)

Problemreduzierung durch Rekursion
- Rekursion ist ein spezifischer Lösungsansatz der Informatik.
- Ist in anderen Ingenieurwissenschaften nicht bekannt:

Ein Programm kann sich selbst aufrufen, ein mechanisches Bauteil kann physikalisch nicht sich selbst als Bauteil enthalten.
- Das rekursive Anwenden einer Lösungsstrategie auf Teilprobleme ist ein Algorithmenmuster.
- Rekursion macht die Korrektheit des Lösungsweges nachvollziehbar.

Einsatz von Algorithmenmustern
- Für bestimmte Problemklassen gibt es „Musterlösungen“.
- Besonders bei NP-vollständigen Problemen sind effektive Lösungsstrategien sehr wertvoll.
- Muster sind generische Algorithmen, die an die konkrete Aufgabe angepasst werden können.
- In modernen Programmiersprachen sind Algorithmenmuster bereits vorhanden: als parametrisierbare Algorithmen, veränderbar via Vererbung mit Überschreiben.
- Es ist wichtig, einige Muster zu kennen und sie auf verwandte Probleme übertragen zu können.

Das Greedy-Prinzip:
In jedem Teilschritt wird versucht, so viel wie möglich zu erreichen. Das heißt, es wird immer der Schritt gewählt, der den meisten sofortigen Gewinn bringt.
greedy= gierig
Voraussetzungen:
- Es handelt sich um ein Optimierungsproblem.
- Die Lösungen bestehen aus Eingabewerten. Alle Lösungen lassen sich Schritt für Schritt durch Hinzunahme von Eingabewerten aufbauen.
- Es existiert eine Bewertungsfunktion für Lösungen (und Teillösungen

Divide-and-Conquer
Andere Namen: Teile und Herrsche, Divide et Impera
Divide: Teile ein komplexes Problem in kleinere, isolierte, ähnliche Teilprobleme (Teile so lange weiter auf, bis die Lösung den kleinen Problemen trivial ist)
Conquer: Löse die Teilprobleme (auf dieselbe art, rekursiv)
Merge: Füge die Lösungen der Teilprobleme zur Gesamtlösung zusammen.
Beispiel: Gute Sortieralgorithmen -MergeSort, QuickSort
Typische Komplexitätsklassen: $O(\mathrm{n}\cdot log(n)$,_$O(log(n))$
Backtracking
bedeutet Zurückverfolgen, ist ein Rücksetzverfahren
Idee: „trial-and-error“ Prinzip
Alle Lösungen werden systematisch getestet, dabei werden schlechte Lösungen verworfen und mit guten Lösungen wird weitergearbeitet.
Garantiert optimale Lösung: Wird häufig durch Rekursion umgesetzt. Ist sehr berechnungsaufwändig, da alle Lösungsmöglichkeiten untersucht werden.
Typische Komplexitätsklasse: $O(2\text{^}n)$

Dynamische Programmierung
Idee: Man berechnet häufig auftretende Teillösungen. Eine Gesamtlösung wird aus den Teillösungen zusammengesetzt.
Funktioniert „bottom-up“ (von unten nach oben): kleinste Teillösungen werden zuerst berechnet und aufgehoben, um daraus schrittweise komplexere Teillösungen zusammenzusetzen.
Findet die optimale Lösung: Vermeidet Mehrfachberechnungen: Teillösungen werden in einer Tabelle abgespeichert. (Divide-and-Conquerwürde die gleichen Teilprobleme immer wieder lösen!) - Braucht zusätzlichen Speicherplatz für die Teillösungen
Hohe Komplexität. Typische Komplexitätsklasse: $O(2\text{^}n)$