Lösung zur Aufgabe 1

Gegeben seien die folgenden Java-Methoden:

```
public static void meth(int n) {
  step1();
  for (int i=1; i <= n; i++) {
    step1();
    step2(n);
    for (int j=1; j <= n; j++) {
      step1();
    }
  }
}

public static void step2(int k) {
  for (int i=1; i <= k; i++) {
    step1();
    step1();
    for (int j=1; j <= 2*k; j++) {
      step1();
    }
  }
}
```

Bestimmen Sie die Ausführungszeit (den Aufwand) der Methode `meth` in Groß-O-Notation in Abhängigkeit von ihrem Parameter n unter der Annahme, dass `step1` die dominante Grundoperation der beiden Methoden ist.

Ermitteln Sie zunächst die Anzahl der durchgeführten Grundoperationen und begründen Sie dabei Ihre Rechenschritte jeweils kurz. Geben Sie dann am Ende die entsprechende Groß-O-Notation an.

Lösung:

`step1` wird in der Methode `step2` in der inneren Schleife $2n$ mal ausgeführt und für jeden Durchgang der äußeren Schleife somit $2n + 2$ mal. Da die äußere Schleife n mal durchlaufen wird ergibt sich für `step2` ein Aufwand von $2n^2 + 2n$.

In `meth` wird in der innersten Schleife `step1` n mal ausgeführt. In jedem Schleifendurchlauf der äußeren Schleife wird `step1` daher $n + 1$ mal und zusätzlich (wegen des `step2`-Aufrufs) $2n^2 + 2n$ mal, also insgesamt $2n^2 + 3n + 1$ mal ausgeführt. Da die äußere Schleife n mal durchlaufen wird ergibt sich für `meth` ein Aufwand von $n \cdot (2n^2 + 3n + 1) + 1 = 2n^3 + 3n^2 + n + 1$.

In Groß-O-Notation: $O(n^3)$.

Lösung zur Aufgabe 2

$$\text{sum} : P(\mathbb{R}) \to \mathbb{R}$$
$$\text{sum}(S) = \sum_{w \in S} w$$

$$\text{nor} : P(\mathbb{R}) \to P(\mathbb{R})$$
$$\text{nor}(S) = \{v \mid v = \frac{w}{\text{sum}(S)} \wedge w \in S\}$$

algorithm sum (S) für $S = \{s_1, s_2, \ldots, s_n\}, s_i \in \mathbb{R}$

 $x := 0$

 for $i := 1$ to n do

  $x := x + s_i$

 return x

algorithm nor (S) für $S = \{s_1, s_2, \ldots, s_n\}, s_i \in \mathbb{R}$

 $T := S$

 $h :=$ sum(T)

 for $i := 1$ to n do

  $t_i := t_i/h$

 return T

```
static double sum (double[] s) {
  double x = 0;
  for (int i=0; i<s.length; i++)
    x = x + s[i];
  return x;
}
```

```
static double[] nor (double[] s) {
  double[] t = new double[s.length];
  double h = sum(s);
  for (int i=0; i<s.length; i++)
    t[i] = s[i]/h;
  return t;
}
```

Aufgabe 3 (T) (Analyse von Algorithmen)

Gegeben seien die folgenden Java-Methoden:

```
public static void work(int n) {
  domStep();
  domStep();
  for (int i=1; i <= n; i++) {
    domStep();
    domStep();
    domStep();
    action(n*n);
    for (int j=1; j <= n; j++) {
      action(n);
    }
  }
}

public static void action(int k) {
  for (int i=1; i <= k; i++) {
    domStep();
    domStep();
    for (int j=1; j <= 7*k; j++) {
      domStep();
  }
}
```

Bestimmen Sie die Ausführungszeit (den Aufwand) der Methode `work` in Abhängigkeit von ihrem Parameter n unter der Annahme, dass `domStep` die dominante Grundoperation der beiden Methoden ist.

Ermitteln Sie zunächst die Anzahl dieser durchgeführten Grundoperationen und begründen Sie dabei Ihre Rechenschritte jeweils kurz. Geben Sie dann am Ende die entsprechende Groß-O-Notation an.

Lösung:

`domStep` wird in der Methode `action` in der inneren Schleife $7n$ mal ausgeführt und für jeden Durchgang der äußeren Schleife somit $7n + 2$ mal. Da die äußere Schleife n mal durchlaufen wird ergibt sich für `action` ein Aufwand von $7n^2 + 2n$.

In `work` wird in der innersten Schleife `action` n mal ausgeführt. In jedem Schleifendurchlauf der äußeren Schleife wird `domStep` 3 mal und zusätzlich wegen des `action`-Aufrufs $n*n$ mal ausgeführt. Das ergibt $3 + (7(n*n)^2 + 2(n*n)) + n \cdot (7n^2 + 2n))$ mal, also insgesamt $3 + 4n^2 + 7n^3 + 7n^4$ mal ausgeführt. Da die äußere Schleife n mal durchlaufen wird ergibt sich für `work` ein Aufwand von $n \cdot (3 + 4n^2 + 7n^3 + 7n^4) + 2 = 2 + 3n + 4n^3 + 7n^4 + 7n^5$.

In Groß-O-Notation: $O(n^5)$.