a)

```
public SquareMatrix multiply(SquareMatrix other) {
	if (values.length != other.values.length) {
		throw new IllegalArgumentException("Dimensions do not match!");
                                                                        // O(1)
	}
	SquareMatrix result = newEmptyInstance();                           // O(1)
	for (int i = 0; i < size(); i++) {                                  // O(n)
		for (int k = 0; k < size(); k++) {                          // O(n)
                        // c_{i,k} = \sum_{j=0}^{n} a_{i,j} * b_{j,k}
			double sum = 0D;                                    // O(1)
			for (int j = 0; j < size(); j++) {                  // O(n)
                                sum += get(i, j) * other.get(j, k);         // O(1)
			}
			result.set(i, k, sum);                              // O(1)
		}
	}
	return result;                                                      // O(1)
}
```

-> O(1) + O(1) + O(n) * O(n) * ((O(1) + O(n) * O(1)) + O(1)) + O(1) => O(n^3)

```
public SquareMatrix multiply(double alpha) {
	SquareMatrix result = newEmptyInstance();                           // O(1)
	for (int i = 0; i < size(); i++) {                                  // O(n)
		for (int j = 0; j < size(); j++) {                          // O(n)
			result.set(i, j, alpha * get(i, j));                // O(1)
		}
	}
	return result;                                                      // O(1)
}
```

-> O(1) + O(n)*O(n)*O(1) + O(1) => O(n^2)

b)

Die Angaben sind Zeitangaben in Nanosekunden, die die jeweilige Funktion für die Ausführung mit den angegebenen Matrizengrößen gebraucht hat.

Matrixgröße	Skalarmultiplikation	Matrixmultiplikation	Minimum	Maximum
1	24000	36000	170000	32000
1	2000	1000	2000	1000
1	1000	1000	0	1000
5	44000	4000	6000	18000
5	80000	16000	6000	4000
5	44000	4000	5000	4000
10	410000	14000	19000	13000
10	379000	13000	20000	14000
10	314000	12000	21000	14000
50	18027000	286000	443000	290000
50	4622000	326000	476000	367000
50	302000	371000	501000	331000
100	1329000	1222000	1671000	1467000
100	1376000	1003000	1474000	1591000
100	1363000	1020000	1485000	1628000
500	1247554000	7474000	2218000	2676000
500	1249379000	1848000	623000	769000
500	1356337000	2575000	519000	772000
1000	18564905000	24313000	1964000	2505000
1000	19842570000	3042000	2546000	2296000
1000	18971333000	4868000	1747000	1945000

Man kann erkennen, dass die erste Ausführung mit Größe 1 immer überproportional langsamer ist als die anderen. Das ist auch bei mehreren Ausführungen der Tests reproduzierbar. Deswegen wird diese bei der Berechnung der Durchschnitte rausgelassen. Teilweise sind auch in anderen Größen Ausführungen deutlich langsamer als andere. Das kann auch daran liegen, dass Zeitangaben nicht die beste Methode ist, Komplexitäten für Algorithmen festzustellen. Es spielen auch andere Faktoren mit rein, die das Ergebnis verfälschen. Z.B. könnte irgendein anderer Prozess auf meinem Computer gerade den Test unterbrochen haben und dadurch verzögern.

Durchschnitte:

Matrixgröße	Skalarmultiplikation	Matrixmultiplikation	Minimum	Maximum
1	1500	1000	1000	1000
5	56000	8000	5667	8667
10	367667	13000	20000	13500
50	7650333	327667	473333	329333
100	1356000	1081667	1543333	1562000
500	1284423333	3965667	1120000	1405667
1000	19126269333	10741000	6257000	2248667

c)

1) $O(100 * N^2) \subseteq O(N^2)$, *wahr da Faktor für* N^2 *als* 101(*z.B.*) *wählbar* $O(N^2) \subseteq O(N^3)$,
da jedes $x \in O(N^2)$ *auch Element der höheren oberen Schranke* $O(N^3)$

2) $10 * N + 20 \in O(N)$, *wahr da Faktor für* N *als* 11(*z.B.*) *wählbar* $\rightarrow$ *größeres Wachstum*
(*für* $N > 20$ *größer*)

3) $10 * N^{\frac{3}{2}} \in O(N\ logN) \Leftrightarrow 10 * \sqrt{N}^3 \in O(N\ logN) \Leftrightarrow 10 * N * \sqrt{N} \in O(N\ logN)$, *falsch*
da c^N *größeres Wachstum* N^c(*hinreichend belegt in vorlesung*)
die inversen Operationen besitzen entsprechend ein umgedrehtes Wachstumsverhältnis
$\rightarrow$ *logN kleineres Wachstum als* $\sqrt{N}$ $\rightarrow$ *Aussage falsch*