Übungen zu Objektorientierte Programmierung 2

frieder.loch@ost.ch

Woche 5: O-Notation

Aufgabe 1: Laufzeit auslesen und Code verbessern

In dieser Aufgabe befassen Sie sich mit drei Codebeispielen, die nicht optimal implementiert sind. Schätzen Sie zunächst die Laufzeit jedes Codebeispiels. Verbessern Sie anschliessend den Code und schätzen Sie die Laufzeit der optimierten Version.

Maximum

Diese Methode such das maximum aus einem Integer-Array und gibt es zurück.

```java
public int findMaximum(int[] nums) {
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < nums.length; i++) {
        boolean isMax = true;
        for (int j = 0; j < nums.length; j++) {
            if (i != j && nums[j] > nums[i]) {
                isMax = false;
                break;
            }
        }
        if (isMax) {
            max = Math.max(max, nums[i]);
        }
    }
    return max;
}
```

DuplicateRemover

Diese Methode löscht alle duplikate aus einem Integer-Array und gibt einen Array ohne die duplikate zurück.

```java
public int[] removeDuplicates_slow(int[] nums) {
    int n = nums.length;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (nums[i] == nums[j]) {
                for (int k = j; k < n - 1; k++) {
                    nums[k] = nums[k + 1];
                }
                n--;
                j--;
            }
```

```
        }
    }
    int[] result = new int[n];
    for (int i = 0; i < n; i++) {
        result[i] = nums[i];
    }
    return result;
}
```

SumChecker

Diese Methode überprüft, ob es in einem Integer-Array zwei Zahlen gibt, welche als Summe den `target` ergeben. - Falls ja ist der Rückgabewert die indexe der beiden Zahlen im Array - Falls nein wird eine IllegalArgumentException geworfen

```
public int[] twoSum_slow(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[i] + nums[j] == target) {
                return new int[]{i, j};
            }
        }
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

Aufgabe 2: Matrix-Multiplikation

Implementieren Sie eine Matrix-Multiplikation und schätzen Sie die Laufzeit Ihrer Implementation.

Wenn A eine $m * n$ Matrix ist und B eine $n * p$. Damit die Multiplikation überhaupt möglich ist muss n denselben Wert haben. Das Resultat ist dann eine $m * p$ Matrix.

$$\begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n} \\ a_{10} & a_{11} & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m0} & a_{m1} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} & \dots & b_{0p} \\ b_{10} & b_{11} & \dots & b_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n0} & b_{n1} & \dots & b_{np} \end{pmatrix}$$

Berechung der Resultat Matrix:

$$\begin{pmatrix} a_{00}b_{00} + \dots + a_{0n}b_{n0} & a_{00}b_{01} + \dots + a_{0n}b_{n1} & \dots & a_{00}b_{0p} + \dots + a_{0n}b_{np} \\ a_{10}b_{00} + \dots + a_{1n}b_{n0} & a_{10}b_{01} + \dots + a_{1n}b_{n1} & \dots & a_{10}b_{0p} + \dots + a_{1n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m0}b_{00} + \dots + a_{mn}b_{n0} & a_{m0}b_{01} + \dots + a_{mn}b_{n1} & \dots & a_{m0}b_{0p} + \dots + a_{mn}b_{np} \end{pmatrix}$$